Ronan Smyth – L00176857

BA Computer Science  CA2

Cybersecurity Report

[

Contents

[Threat Modeling 1](#_Toc166450133)

[Native And Hybrid Mobile Applications 1](#_Toc166450134)

[Threat Modeling of a Native Application 1](#_Toc166450135)

[Assets & Relevant Controls 1](#_Toc166450136)

[Additional Controls 2](#_Toc166450137)

[Threat Actors 3](#_Toc166450138)

[Threat Modeling Hybrid App 4](#_Toc166450139)

[Assets & Relevant Controls 4](#_Toc166450140)

[Additional Controls 5](#_Toc166450141)

[Threat Actors 6](#_Toc166450142)

[Reverse Engineering 0](#_Toc166450143)

[Purposes of Reverse Engineering 0](#_Toc166450144)

[Reverse Engineering Steps 0](#_Toc166450145)

[Android 0](#_Toc166450146)

[iOS 1](#_Toc166450147)

[Tampering 1](#_Toc166450148)

[Types of Tampering 1](#_Toc166450149)

[Reverse Engineering Process 0](#_Toc166450150)

[Mobile Application Security Testing 0](#_Toc166450151)

[Types of Mobile Security Testing 0](#_Toc166450152)

[SAST 0](#_Toc166450153)

[DAST 0](#_Toc166450154)

[Pentesting 0](#_Toc166450155)

[Sandboxing 0](#_Toc166450156)

[Tools 1](#_Toc166450157)

[Rooting and Jailbreaking 0](#_Toc166450158)

[Rooting 0](#_Toc166450159)

[Reasons for Rooting 0](#_Toc166450160)

[Steps 0](#_Toc166450161)

[Jailbreaking 0](#_Toc166450162)

[Reasons for Jailbreaking 0](#_Toc166450163)

[Steps 1](#_Toc166450164)

[Zed Attack Proxy (ZAP) Lab 0](#_Toc166450165)

[Alerts 0](#_Toc166450166)

[Spider Scan 2](#_Toc166450167)

[Active Scan 2](#_Toc166450168)

[Webgoat 0](#_Toc166450169)

[SQL Injection 0](#_Toc166450170)

[XSS 4](#_Toc166450171)

[References 0](#_Toc166450172)

# Threat Modeling

## Native And Hybrid Mobile Applications

Native Apps are built in specific programming languages (Swift, Objective-C, or Java) depending on whether they are being developed for iOS or Android. They function without an internet connection. They are obtained on an app store and take full advantage of mobile device features which makes them faster and more efficient than other architectures. On the whole, they are more expensive than hybrid app architectures, having separate codebases across platforms which leads to more code being written and therefore more maintenance.

Hybrid apps combine elements of native and web apps. Users download them from app stores, but they are developed with web technologies like JavaScript, CSS, and HTML5 within the shell of a native app. This allows hybrid apps to use device features while being cost-effective and easier to develop across multiple platforms (Amazon, 2024).

## Threat Modeling of a Native Application

WhatsApp is an end-to-end encrypted (E2EE) messaging app that supports text, audio, and media communication between users, including group messaging. E2EE ensures messages are only readable by the sender and recipient. Even WhatsApp is unable to access the content. The encryption employs a Signal protocol which combines asymmetric and symmetric cryptography. Each message is encrypted with a unique key, and keys are stored locally on devices for added security. This system enhances privacy by safeguarding messages from potential eavesdropping, even if a device is compromised. (Chowdhury *et al.*, 2023).

WhatsApp is considered a native app rather than a hybrid app. Native apps are specifically designed and optimized for their respective operating systems, utilizing the core programming languages and integrated development environments of those platforms, such as Java or Kotlin for Android and Objective-C or Swift for iOS. WhatsApp takes advantage of the full capabilities of the device hardware, ensuring high performance and a seamless user experience. This approach allows WhatsApp to efficiently implement complex features such as end-to-end encryption, real-time messaging, and voice and video calling, which are deeply integrated with the device's operating system functionalities (Li et al, no date).

### Assets & Relevant Controls

1. **A1 - User Data:** Includes chat histories, media files, and contact lists.
   * **C1 - Protected by End-to-End Encryption (E2EE):** Ensures that all communications are encrypted from the sender to the recipient, safeguarding the confidentiality and integrity of user data during transmission. Protects data in transit between the WhatsApp app on mobile devices and the WhatsApp server, as well as through WhatsApp Web.
   * **C2 - Backup Encryption:** User data backups stored on Google Drive are encrypted to protect data privacy and integrity even off the device.
2. **A2 - Encryption Keys:** These keys are essential for securing communications and are stored within secure elements of the device.
   * **C3 - App Sandbox:** Protects encryption keys by isolating the WhatsApp application from other applications on the device, preventing unauthorized access.
3. **A3 - App Infrastructure:** Includes the WhatsApp app and server components.
   * **C4 - Automatic Updates:** Helps in maintaining the security of the app infrastructure by ensuring the latest security patches and updates are applied.
   * **C5 - Two-Step Verification:** Adds an additional layer of security when accessing or reinstating the app on new devices.
4. **A4 - Communication Protocols:** Includes protocols like end-to-end encryption utilized in data transmissions.
   * **C1 - End-to-End Encryption (E2EE):** As above.
5. **A5 - Backup Data:** Consists of user data backups stored on Google Drive.
   * **C2 - Backup Encryption: As above.**
6. **A6 - User Interface Components:** These components facilitate user interaction with the WhatsApp app.
   * **C3 - App Sandbox:** As above.
7. **A7 - Codebase and Development Infrastructure:** Represents the software code and development tools for building and maintaining WhatsApp.
   * **C6 - Automatic Updates:** Ensures the app remains secure against vulnerabilities by updating the codebase and development infrastructure regularly.
8. **A8 - Regulatory Compliance Data:** Ensures WhatsApp operates in compliance with legal and regulatory standards.
   * **C7 - Privacy Settings:** Allows users to control the visibility of their information and manage consent, aiding in compliance with privacy regulations.

### Additional Controls

The following controls cover not only the assets stated above but also the general security of the system by and large.

1. **C8–** **App Store Verification:** The app store client and server are maintained and monitored to a very high standard so that malicious use is difficult.
2. **C9 – Binary Hardening:** Securing systems by reducing the attack surface. analysing or manipulating binary files to protect against exploitation of such.
3. **C10** – **Rooting/Jailbreak Detection:** Checking system files, properties, modifications and permissions to ensure a threat actor has not gained administrative privileges.
4. **C11 –** **App Permissions Enforcement:** The mobile devices security configuration will not allow the application access to functions to which it does not have explicit permission to access.

A diagram of a mobile device

Description automatically generated

Figure - Threat Modelling Diagram for WhatsApp.

### Threat Actors

1. **T1 - Remote Hackers**: These actors could try to intercept or manipulate data in transit, perform denial-of-service attacks, or exploit vulnerabilities in the server software.
2. **T2 - Malware Applications(Root)**: Malware could attempt to breach the sandbox or exploit other apps to gain unauthorized access to WhatsApp data.
3. **T3 - Malware Applications(Sandbox)**: These could attempt to exploit shared resources or vulnerabilities within the sandbox or operating system to access or manipulate WhatsApp data.
4. **T4 - Insider Threats (Disgruntled Employees)**: These individuals might have legitimate access but could misuse their permissions to access or leak data.
5. **T5 - Phishing Attackers**: By deceiving users into providing sensitive information or clicking malicious links, these attackers could gain access to user accounts or install malware.
6. **T6 - Third-party Service Providers** They could potentially access or inadvertently leak user data stored in backups.
7. **T7 - Government Agencies or Other Surveillance Bodies**: Potential legal or covert surveillance might aim to intercept communications or access stored data.
8. **T8 - End Users**: Users might disable security features, use weak authentication methods, or install malicious apps that compromise the device.

## Threat Modeling Hybrid App

The Gmail app is an example of a hybrid approach to mobile app development, combining native and web elements to optimize user experience, functionality and ease of development and maintenance. Initially, Gmail utilized a predominantly shell app model with significant HTML content, which impacted usability negatively. Recognizing these limitations, Google shifted towards a more native-centric design, improving interface responsiveness and aesthetics such as animations. However, certain parts of the app, like the email list and message view, still employ HTML within web views. This hybrid strategy allows Google to enhance user interactions while maintaining some cross-platform efficiencies, particularly for displaying HTML content in emails (Shackles, 2012).

### Assets & Relevant Controls

1. **A1 - User Emails and Attachments** 
   * **C1 - Data Encryption:** To ensure data confidentiality.
   * **C2 - Input Validation / Output Encoding:** To prevent malicious data from being processed.
2. **A2 - Authentication Tokens/Cookies** 
   * **C3 - TLS / Certificate Verification:** To secure data in transit.
   * **C4 - App Permissions Enforcement:** To restrict access to sensitive data.
3. **A3 - Gmail User Settings and Preferences** 
   * **C5 - Application Backup Configuration:** To secure backup data.
   * **C6 - User Authentication / Authorization:** To ensure only the rightful user can access and modify settings.
4. **A4 - User Contact List and Email Metadata** 
   * **C7 - Same Origin Policy**:To prevent data leakage via cross-origin resource sharing.
   * **C1 - Data Encryption:** To protect data both at rest and in transit.
5. **A5 - Gmail Account Information** 
   * **C6 - User Authentication / Authorization:** To validate user identity and permissions.
   * **C3 - TLS / Certificate Verification:** To protect account information during transmission.
6. **A6 - Device Sync Data for Multi-Device Access** 
   * **C4 - App Permissions Enforcement:** To control what data can be synced across devices.
   * **C8 - Binary Hardening:** To protect the app components handling synchronization from being tampered with.

### Additional Controls

The following controls cover not only the assets stated above but also the general security of the system by and large.

* **C2 - Input Validation / Output Encoding**: Prevents XSS and other injection attacks. Location: Mobile App (WebView).
* **C3 - TLS / Certificate Verification:** Ensures secure, encrypted communications between the client and Google servers. Location: App Store Client; Application Server.
* **C5 - Application Backup Configuration:** To secure backup data.
* **C8 - Binary Hardening**: Protects the Gmail app from reverse engineering and tampering. Location: Mobile App (Native Component).
* **C9 - JavaScript Obfuscation:** Protects against script injection and tampering. Location: Mobile App (WebView).
* **C10** – **App Sandboxing:** Isolates an application from others.
* **C11** – **App Store Verification:** The app store client and server are maintained and monitored to a very high standard so that malicious use is difficult.

1. **C12** – **Rooting/Jailbreak Detection:** Checking system files, properties, modifications and permissions to ensure a threat actor has not gained administrative privileges.

* **C13** – **Whitelist of sites available in Webview:** Only a predefined list of trusted websites can be accessed or displayed within the WebView component of an app.

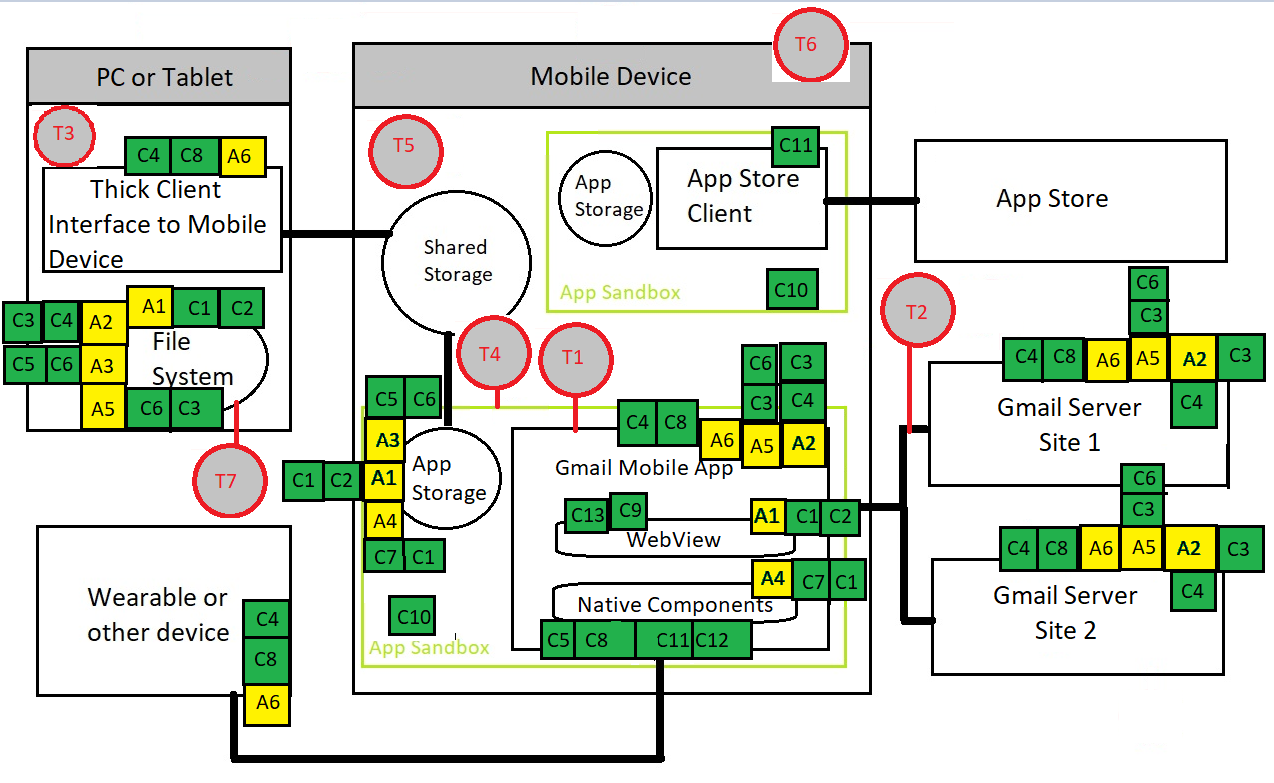


Figure - Threat Modeling Diagram for Gmail.

### Threat Actors

1. **T1 - Malicious App Users:** Attempt to exploit Gmail via compromised user credentials.
2. **T2 - Network Attackers:** Intercept or manipulate data in transit.
3. **T3 - Phishing Websites:** Deceive users into divulging credentials.
4. **T4 - Malware Applications(Sandbox):** Access Gmail data through other compromised apps within the same device.
5. **T5 - Malware Applications(Root):** Exploit root privileges to bypass security controls and access Gmail data.
6. **T6 - Users with Physical Access:** Access or manipulate Gmail data directly from the device.
7. **T7 - Compromised Workstation Apps:** Exploit vulnerabilities in the workstation or thick client interfacing with the mobile device to access Gmail data.

# Reverse Engineering

Reverse engineering is slightly different across platforms because the architecture of Android and iOS applications are different.

Android applications must be compiled into APK (android package kit) format. These applications are generally built in Java or Kotlin. Apk format is a compressed format containing all of the data and resources a mobile application needs and reverse engineering on android apps starts with the APK file. APK files are a variant of [Java Archive](https://www.theserverside.com/definition/JAR-file-Java-ARchive) files(Gillis, 2023).

Apk files are compressed and unreadable. Thus, they must be converted to another format and decompiled to extract the source code. Tools can be used to obtain the APK files, convert them to .dex files and finally view the information therein.

Tools such as dex2jar and enjarify are commonly used to convert .dex or .apk files to .jar files, respectively. Other tools like Bytecode Viewer and JD-GUI facilitate the reading allowing deeper inspection of the code.

The process for iOS has similarities and differences. Instead of APK files, iOS uses iOS Appstore Packages, IPAs for short, which, like APKs, are a compressed formats derivative of Apple development languages, Swift and Objective-C.

Reverse engineering iOS apps involves decrypting and extracting the IPA to obtain executable files, which can then be disassembled using tools like Hopper or IDA Pro to study the assembly code. iOS apps are compiled into binary code which is harder to reverse than Android’s .dex files.

In order to perform reverse engineering on both Android and iOS applications, a rooted or jailbroken device, respectively, is generally necessary so as to bypass integrated detection applications which will inhibit some tools which may be used. Seemingly, this is more common for iOS devices due to higher security features (OWASP, 2024).

## Purposes of Reverse Engineering

**Study:** Understand how applications work, especially in academic or training sectors.

**Code Analysis:** Analyse software to detect flaws for the purposes of improving code.

**Compatibility:** Create versions of products available on one platform which can work on other platforms, without access to the original source code.

**Enhancement:** Enhance existing applications and/or integrate new features.

**Security:** Identify and fix security flaws.

**Intellectual Property:** Protect intellectual property by understanding how pirated versions of software operate.

## Reverse Engineering Steps

### Android

1. **Extract the APK:** Use tools like Apk Extractor to get the APK file from an installed app. Android devices may need to be rooted depending on the tools being used.
2. **Decompile the APK:** Convert the APK to a JAR using dex2jar or enjarify, so code can be read.
3. **Analyse the Code:** Open the JAR in a tools like JD-GUI or Bytecode Viewer to read the source code.

### iOS

1. **Decrypt the IPA:** Use tools like Clutch or Rasticrac on a jailbroken iOS device to decrypt the IPA file.
2. **Extract the Executable:** Extract the decrypted IPA to access the binary executable.
3. **Disassemble the Binary:** Use a disassembler like Hopper or IDA Pro to analyse the binary code.

## Tampering

Tampering, as the name suggests, refers to the modification of an application’s code or behaviour, normally for some sort of malicious purpose.

Of course, for Android devices that will mean modifying an APK while for iOS an IPA will be modified.

### Types of Tampering

* Code Injection: Adding new code to change application behaviour.
* Behaviour Modification: Altering the logic of existing code to change its function.
* Resource Replacement: Swapping out legitimate resources (like databases or multimedia files) with malicious ones.

More specific examples of tampering may include, but are not limited to:

1. Inserting malware into apps.
2. Bypassing licensing or payment checks.
3. Modifying an app to access unauthorized features or data.
4. Insertion of a backdoor.
5. Removal of ads.
6. Bypass in-app purchases.
7. Bypass subscription checks.

## Reverse Engineering Process

A diagram of a computer program

Description automatically generatedIn the notes for reverse engineering, the process is described as a conversion from .apk to .dex and then to .jar.

It seems in the lab we skip the .dex step using enjarify to converty directly from .apk to .jar.

Required tools and downloads:

1. Python
2. Java JRE
3. APKTool
4. Sample .apk file.

Figure - Reverse engineering process from lectures.

1. Enjarify
2. JD-Gui
3. Bytecode Viewer

After downloading the tools and sample .apk file, and adding paths to the environment variables, the sample file was copied to the apktool folder and renamed test.apk.

In the command line, in the directory of the folder that both the apktool and test.apk were found, the apktoll was run on the test file, as shown in figure 1.

A computer screen shot of a computer screen

Description automatically generated

Figure - Running apktool.

The tool produced a folder called test within which we can find some information about the internal structure of test.apk. In addition, the AndroidManifest.xml file, which is the principal file for mobile application development, can be seen below in figure 2.

A screenshot of a computer

Description automatically generatedThe reverse engineering lab stated that this manifest file would be .apk, whereas it is an .xml.

I am unclear why this difference has been encountered but a file entitles MANIFEST.MF was found in the original folder.

Figure - AndroidManifest.xml

A .jar file needed to be created to be read by tools bytecode-viewer and the Java Decompiler Gui. Figure 3, below, shows the enjarify tool converting our test.apk file to test-enjarify.jar, also shown in figure 3 in the enjarify-master folder to the left.

A screenshot of a computer screen

Description automatically generated

Figure - Converting .apk to .jar with enjarify.

The test-enjarify.jar file is the read using JD-Gui to show the code therein.

A screenshot of a computer

Description automatically generatedThe sample chosen for this reverse engineering was the first on the list of samples from the website.

Figure - Converted .apk file viewed from JD-Gui.

I have been unable to discern what it does though I can see that it contains code written in Kotlin and involves the use of coroutines, asynchronous programming.

The program seems incredibly broad, involving huge amounts of functionality and vast quantities of classes.

I encountered the same issue when using byte-code viewer to analyse the code. In figure 6, key words like “main” were searched in the hope that some main class could be located. Considerable time has been spent on this but I have been unable to determine what the program does.

A screenshot of a computer program

Description automatically generated

Figure - Jar file viewed from Bytecode Viewer.

Finally, the following command line argument was used to try to run the code. An error message was received as to the file being invalid or corrupt.



Figure - Trying to run the .jar file.

# Mobile Application Security Testing

Describe the types of mobile security testing (pen testing, static, and dynamic) and name the tools (penetration as well as vulnerability testing) for mobile applications.

## Types of Mobile Security Testing

### SAST

Static Application Security Testing (SAST) involves examining an application’s components without executing them, by analysing the source code with or without tools. This can help to identify problematic coding patterns and potential vulnerabilities.

SAST tools automatically check source code for compliance with a predefined rules or best practices list. Examples include SonarLint or any other linting tool, Snyk for a deeper security focus and any of the tools listed on [OWASP's Source Code Analysis Tools page](https://owasp.org/www-community/Source_Code_Analysis_Tools).

### DAST

Dynamic Application Security Testing (DAST) examines applications during runtime. DAST is focused on finding security vulnerabilities or weak spots in a program while it is running.

It includes intercepting traffic, dumping memory, checking local storage, and breaking SSL.

Proxy tools like Burp Suite and Proxyman can used for intercepting and manipulating traffic while a program is running. Furthermore, vulnerability scanning tools, like Nikto, are used to identify vulnerabilities in running applications, and are listed on [OWASP's Vulnerability Scanning Tools page](https://owasp.org/www-community/Vulnerability_Scanning_Tools).

### Pentesting

Penetration Testing (Pentesting) is a general security test (incorporating SAST and DAST) of an application carried out when it is finished (or almost finished), including:

1. Reconnaissance
2. Static analysis
3. Dynamic analysis
4. Exploitation
5. Reporting

Reconnaissance tools are used for information gathering about the application and the company. This may include port scanning, Google hacking or website recon tools.

DAST and SAST tools are also used in the relative stages of penetration testing to discover vulnerabilities.

## Sandboxing

A sandbox is an isolated testing environment that enables users to run programs or execute files without affecting the application, system, or platform on which they run. However, as I understand it, the meaning is a little different in the context of mobile applications whereby sandboxing does not only refer to testing environments.

The Android platform takes advantage of the Linux user-based protection to identify and isolate app resources. This isolates applications from each other and protects the apps and the greater system from malicious apps. To do this, Android assigns a unique user ID (UID) to each Android application and runs it in its own process(Android, 2024).

Not surprisingly, iOS sandboxes applications in a very similar way to protect the apps and the system by and large, from malicious software. Each app operates within its own home directory while system files and resources are read-only.

### Tools

Some tools which can be used for mobile application development sandboxing include:

1. Android Studio’s Emulator, which is built into Android Studio, allows developers to safely test their applications in a controlled environment that mimics a range of devices.
2. iOS Simulator is a part of Xcode which allows developers to test iOS apps on a Mac, offering an environment that simulates iOS devices without the need for actual hardware.
3. Google Firebase Test Lab application testing, including sandboxing aspects, on the cloud. It provides a wide range of simulated devices.
4. Docker, although not specifically for mobile applications, can be used to containerize backend services that mobile apps interact with, ensuring that any service the mobile app uses during testing does not affect the underlying host system.

# Rooting and Jailbreaking

## Rooting

Rooting is a process of gaining root access or administrative privileges on Android devices’ operating system. This is something like having administrative privileges on a Windows computer or having root access in Unix/Linux systems.

### Reasons for Rooting

**Customization**: Allows for extensive customization of the device’s software and user interface.

**Removing Bloatware**: Enables the removal of pre-installed applications and system software that the device manufacturer or carrier might have added.

**Enhanced Performance**: Users can overclock or underclock the device’s processor to enhance performance or battery life.

**Installing Unauthorized Apps**: Rooting makes it possible to install apps and software that are not available through the official Google Play Store, including those that require deep system access.

**Updating to the Latest Android Version**: Allows the latest updates to the Android operating system without having to wait for the official roll-out from the manufacturer or carrier.

### Steps

1. **Backup Your Data**: Rooting can potentially lead to data loss.
2. **Enable Developer Options**.
3. **Enable OEM Unlocking and USB Debugging**.
4. **Unlock the Bootloader**: Generally, involves connecting your device to a PC and using commands via ADB (Android Debug Bridge) and may require getting an unlock key from the manufacturer.
5. **Install a Custom Recovery**:

Tools (like TWRP - Team Win Recovery Project) allow you to install custom firmware, which can be flashed onto the device using ADB commands.

1. **Root the Device**:

With a custom recovery tool installed, you can then flash a root package (like Magisk) to achieve root access. Once installed, you have root access.

## Jailbreaking

Jailbreaking, like rooting, is the process of removing software restrictions imposed by the device's manufacturer, specifically on iOS devices. This is done by exploiting vulnerabilities in Apple's software to gain root access to the operating system, allowing the user to bypass limitations set by Apple.

### Reasons for Jailbreaking

**Customization**: Allows deeper customization of the user interface and the ability to install themes and tweaks that change the look and feel of the device.

**Installation of Unauthorized Apps**: Enables the installation of apps and extensions that are not available on the Apple App Store, providing more software variety and features.

**Enhanced Features and Functions**: Users can add functionalities that are not available in the standard iOS configuration, such as setting default apps, adding widgets, or enabling additional control settings.

### Steps

1. **Backup Your Data**: As before, it is advisable to back up the device using iCloud or iTunes to prevent data loss.
2. **Disable Security Features**:
   * Turn off **Find My iPhone**.
   * Disable **Passcode and Touch ID/Face ID**.
3. **Enable Airplane Mode**.
4. **Download Jailbreaking Software on Computer**: Find reliable jailbreaking software compatible with your iOS version. Examples include:
   * Checkra1n
   * Unc0ver
   * Taurine
5. **Connect Device to Your Computer**.
6. **Start the Jailbreak Process**: Follow the specific instructions provided by the jailbreaking tool.
7. **Complete the Jailbreak**: The tool will execute the necessary steps to jailbreak the device.
8. **Install Cydia**: Cydia is a popular application for managing additional tweaks and apps on jailbroken devices. Most jailbreaking tools will automatically install Cydia.
9. **Re-enable Security Features**: Once the jailbreak is complete, turn off Airplane Mode and re-enable your security settings, including Find My iPhone and passcode or biometric locks(Avast, 2024).

# Zed Attack Proxy (ZAP) Lab

OWASP ZAP (Zed Attack Proxy) is an open-source security tool used for finding vulnerabilities in web applications.

ZAP scans a given website, in this case <www.lyit.ie>, looking for known weaknesses.

A screenshot of a computer

Description automatically generated

Figure - Zap Quick Start screen.

## Alerts

Figure 2 below, shows the results of a ZAP scan which identifies 18 security alerts of varying severity in terms of risk. The numbers in parentheses (e.g., 372, 535) represent the count of instances where these issues were found during the scan.

A screenshot of a computer

Description automatically generated

Figure - ZAP alerts box for <www.lyit.ie>.

In this case, an automated scan of the website <www.lyit.ie>, identified a number of potential security issues which could be exploited by threat actors. Here's a list of some of the most prolific alerts identified and what they indicate:

1. PII Disclosure: Potential exposure of Personally Identifiable Information, which could compromise user privacy and is possibly in breach of GDPR.
2. Content Security Policy (CSP) Header Not Set: The application does not have a CSP header configured, increasing vulnerability to Cross-Site Scripting (XSS) attacks.
3. Cross-Domain Misconfiguration: Possible improper cross-domain configuration which could lead to unauthorized access or data leakage.
4. Hidden File Found: Indicates that the scanner has discovered files that should be hidden and may contain sensitive information.
5. Vulnerable JS Library: The web application is using JavaScript libraries known to have vulnerabilities.
6. Cookie without SameSite Attribute: Cookies are set without the SameSite attribute, making the site more susceptible to Cross-Site Request Forgery (CSRF) attacks, a new entry on the most recent OWASP Top 10.
7. Cross-Domain JavaScript Source File Inclusion: Points to potential security risks from including JavaScript files from different domains.
8. Server Leaks Version Information via "Server" HTTP Response Header Field: The server is disclosing its software version in HTTP headers, which provides attackers with helpful information.
9. Strict-Transport-Security Header Not Set: HTTP Strict Transport Security (HSTS) is not configured, which prevents HTTPS enforcement.
10. Modern Web Application: General alert indicating the use of modern web technologies, which could have specific security implications.
11. Re-examine Cache-control Directives: Suggests that the application's cache-control headers are not optimally set, potentially leading to sensitive data being stored in cache accessible by unauthorized parties.
12. Session Management Response Identified: Indicates potential issues with how sessions are managed and maintained, which could affect the security of user sessions.
13. User Controllable HTML Element Attribute (Potential XSS): Points to potential XSS vulnerabilities where users can control HTML element attributes that are not properly sanitized.

Figure 2 shows that 1 high risk, 4 medium risk, 5 low risk and 8 informational alerts have been found. ZAP also scans and detects false positives of which there are none. What is striking is the sheer number of each alert, the highest being 4,257 detections of “Cookies without SameSite Attribute”. Each of these alerts represents a potential security risk and should be investigated and mitigated appropriately to secure the application against possible exploitation.

## Spider Scan

A screenshot of a computer

Description automatically generated

Figure - ZAP Spider Scan feature.

Spider scan is a part of the ZAP tool which automatically discovers URLs on a given website. It begins by visiting a list of URLs called “seeds” before identifying the hyperlinks in the page and adding them to the list. The process iterates until the website has been scanned completely.

## Active Scan

A screenshot of a computer

Description automatically generated

Figure - ZAP active scan output.

Active scan looks for security vulnerabilities using common attacks against elements of the webpage.

“Active scanning is an attack on those targets” (ZAP, 2024).

The active scan feature can only find certain vulnerabilities. Logical vulnerabilities, for instance broken access control, cannot be detected by any vulnerability scanning. For that reason, it is usually advisable to carry out manual penetration testing in addition to active.

# Webgoat

“WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open-source components” (*OWASP WebGoat | OWASP Foundation*, 2024).

Note: It is 23:05 on 12/05 less than an hour before this report has to be submitted. I have spent a low estimate of 40 hours on this assignment. I’ve said it before, I find these assignments excessively time-consuming and finally, I haven’t had time to do any write up for the WebGoat section. I have included only screenshots showing that I have completed the work.

## SQL Injection

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a phone

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer screen

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure - '; UPDATE employees SET salary = '9999999' WHERE userid = '3SL99A' –

## XSS

A screenshot of a shopping cart

Description automatically generated

Figure – This WebGoat exercise has changed since the lab nots were made.

A white rectangular object with black text

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure – Some instructions from the lab are outdated and no longer work.

# References

Amazon (2024) *Web Apps vs. Native Apps vs. Hybrid Apps - Difference Between Types of Web and Mobile Applications - AWS*, *Amazon Web Services, Inc.* Available at: <https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/> (Accessed: 9 May 2024).

Chowdhury, P.D. *et al.* (2023) ‘Threat Models over Space and Time: A Case Study of E2EE Messaging Applications’. arXiv. Available at: <http://arxiv.org/abs/2301.05653> (Accessed: 9 May 2024).

Conklin, L. (2024) *Threat Modeling Process | OWASP Foundation*. Available at: <https://owasp.org/www-community/Threat_Modeling_Process> (Accessed: 9 May 2024).

Li, C., Sanchez, D. and Hua, S. (no date) ‘WhatsApp Security Paper Analysis’.

Null, C. (no date) *5 mobile threat modeling mistakes and how to avoid*, *TechBeacon*. Available at: <https://techbeacon.com/security/5-mobile-app-threat-modeling-mistakes-how-avoid-them> (Accessed: 9 May 2024).

Patidar, P.K. *et al.* (2023) ‘A Threat modeling approach to analyze and mitigate WhatsApp attacks: A Review’, in *2023 IEEE International Students’ Conference on Electrical, Electronics and Computer Science (SCEECS)*. *2023 IEEE International Students’ Conference on Electrical, Electronics and Computer Science (SCEECS)*, Bhopal, India: IEEE, pp. 1–6. Available at: <https://doi.org/10.1109/SCEECS57921.2023.10062984>.

Sahoo, P.K. (2024) *How can you use threat modeling to identify threats to your mobile apps?* Available at: <https://www.linkedin.com/advice/0/how-can-you-use-threat-modeling-identify-threats> (Accessed: 9 May 2024).

Shackles, G. (2012) *On Gmail and the Future of Hybrid Apps*, *On Gmail and the Future of Hybrid Apps*. Available at: <https://gregshackles.com/on-gmail-and-the-future-of-hybrid-apps/> (Accessed: 9 May 2024).

Singh, A.K. and Goyal, N. (2020) ‘Understanding and Mitigating Threats from Android Hybrid Apps Using Machine Learning’, in *2020 IEEE International Conference on Big Data (Big Data)*. *2020 IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA: IEEE, pp. 1–9. Available at: <https://doi.org/10.1109/BigData50022.2020.9377952>.

GeeksForGeeks (2022) *Methods of Generating APK of Android Application*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/methods-of-generating-apk-of-android-application/> (Accessed: 12 May 2024).

Gillis, A. (2023) *What is an APK file (Android Package Kit file format)? | Definition from TechTarget*, *WhatIs*. Available at: <https://www.techtarget.com/whatis/definition/APK-file-Android-Package-Kit-file-format> (Accessed: 12 May 2024).

Kotlin (2022) *Coroutines | Kotlin*, *Kotlin Help*. Available at: <https://kotlinlang.org/docs/coroutines-overview.html> (Accessed: 12 May 2024).

Lenovo (2024) *Discovering the Hidden Gems Inside an APK | Lenovo IE*. Available at: <https://www.lenovo.com/ie/en/glossary/apk/> (Accessed: 12 May 2024).

Manqueros, R. (2023) ‘Reverse engineering an Android app’, *Medium*, 10 March. Available at: <https://rene-manqueros.medium.com/reverse-engineering-an-android-app-40996ab29122> (Accessed: 12 May 2024).

OWASP (2024) *iOS Anti-Reversing Defenses - OWASP Mobile Application Security*. Available at: <https://mas.owasp.org/MASTG/iOS/0x06j-Testing-Resiliency-Against-Reverse-Engineering/#reverse-engineering-tools-detection> (Accessed: 12 May 2024).

Android (2024) *Application Sandbox*, *Android Open Source Project*. Available at: <https://source.android.com/docs/security/app-sandbox> (Accessed: 12 May 2024).

Apple (2021) *Security of runtime process in iOS and iPadOS*, *Apple Support*. Available at: <https://support.apple.com/en-ie/guide/security/sec15bfe098e/web> (Accessed: 12 May 2024).

OWASP (2024a) *Source Code Analysis Tools | OWASP Foundation*. Available at: <https://owasp.org/www-community/Source_Code_Analysis_Tools> (Accessed: 9 May 2024).

OWASP (2024b) *Vulnerability Scanning Tools | OWASP Foundation*. Available at: <https://owasp.org/www-community/Vulnerability_Scanning_Tools> (Accessed: 9 May 2024).

RedLegg Blog (2023) *12 Mobile Application Penetration Testing Tools And Services*. Available at: <https://www.redlegg.com/blog/12-mobile-application-pen-testing-tools-and-services> (Accessed: 12 May 2024).

Avast (2024a) *What Is Jailbreaking and Is It Safe?*, *What Is Jailbreaking and Is It Safe?* Available at: <https://www.avast.com/c-jailbreaking> (Accessed: 12 May 2024).

Avast (2024b) *What Is Rooting? The Risks of Rooting Your Android Device*, *What Is Rooting? The Risks of Rooting Your Android Device*. Available at: <https://www.avast.com/c-rooting-android> (Accessed: 12 May 2024).

GeeksforGeeks (2020) *What is Android Rooting?*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/what-is-android-rooting/> (Accessed: 12 May 2024).

GeeksforGeeks (2022) *How To Root Your Smartphone?*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/how-to-root-your-smartphone/> (Accessed: 12 May 2024).

GeeksforGeeks (2023) *10 Best Root Apps For Android Phone [2023]*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/best-root-apps-for-android-phone/> (Accessed: 12 May 2024).

kaspersky (2023) *What is Jailbreaking – Definition and Explanation*, *www.kaspersky.com*. Available at: <https://www.kaspersky.com/resource-center/definitions/what-is-jailbreaking> (Accessed: 12 May 2024).

Microsoft (no date) *What does it mean to jailbreak your phone—and is it legal?*, *Microsoft 365*. Available at: <https://www.microsoft.com/en-us/microsoft-365-life-hacks/privacy-and-safety/what-is-jailbreaking-a-phone> (Accessed: 12 May 2024).

wikiHow (no date) *How to Jailbreak Your Phone (with Pictures)*, *wikiHow*. Available at: <https://www.wikihow.com/Jailbreak-Your-Phone> (Accessed: 12 May 2024).

*OWASP WebGoat | OWASP Foundation* (no date). Available at: <https://owasp.org/www-project-webgoat/> (Accessed: 12 May 2024).

ZAP (2024a) *ZAP – Active Scan*. Available at: <https://www.zaproxy.org/docs/desktop/start/features/ascan/> (Accessed: 12 May 2024).

ZAP (2024b) *ZAP – Getting Started*. Available at: <https://www.zaproxy.org/getting-started/> (Accessed: 12 May 2024).

ZAP (2024c) *ZAP – Spider*. Available at: <https://www.zaproxy.org/docs/desktop/addons/spider/> (Accessed: 12 May 2024).

ZAP (2024d) *ZAP – ZAP Alert Details*. Available at: <https://www.zaproxy.org/docs/alerts/> (Accessed: 12 May 2024).