App Security Report

For Internal Purpose

دائـــــرة الـصـــحـــة
DEPARTMENT OF HEALTH

24 Health Monitor

net.t247tech.healthmonitor

1.0

Application Details

Application Name	24 Health Monitor
Platform	Android
Application Namespace	net.t247tech.healthmonitor
Version	1.0
Version Code	10000
Application SHA1 Hash	4e614020e8511358b5ba75a53e81fcfd7d270836
Application MD5 Hash	c762ea15358693865edccc43d35cb919

Audit Details

File ID	151
Audit Date	Oct. 9, 2025, 4:01 a.m.
Scan Status	✅ Static ✅ Dynamic ✅ API
Scans included in the Report	Static, Dynamic, API

Table of Contents

Report Summary
- Audit Summary
- Security Rating
Root Detection
- Compliant Solution
- Business Implication
- Related Vulnerabilities
General Server Vulnerabilities
Javascript CORS enabled in Webview
- Noncompliant Code Example
- Compliant Solution
- Related Vulnerabilities
Hooking Detection
- Compliant Solution
- Business Implication
WebView Exploits
- Compliant Solution
- Business Implication
Weak PRNG (Pseudorandom number generator)
- Noncompliant Code Example
- Compliant Solution
- Business Implication
StrandHogg Vulnerability
- Noncompliant Code Example
- Compliant Solution
- Business Implication
- Related Vulnerabilities

MediaProjection: Android Service Allows Recording of Audio, Screen Activity
- Compliant Solution
- Related Vulnerabilities
Disabled SSL CA Validation and Certificate Pinning
- Compliant Solution
- Business Implication
- Related Vulnerabilities
Android Developer options status detection
- Compliant Solution
- Business Implication
Android Developer Bridge(ADB) status detection
- Compliant Solution
- Business Implication
Bytecode Obfuscation
- Noncompliant Code Example
- Compliant Solution
- Related Vulnerabilities
Enabled Android Application Backup
- Noncompliant Code Example
- Compliant Solution
- Related Vulnerabilities
Keylogger Protection
- Compliant Solution
- Business Implication

Report Summary

This report is generated based on the findings during the automated auditing process. It also contains the process of discovering those vulnerabilities in the first place, and ways to remediate those issues.

Audit Summary

Root Detection Dynamic Since a rooted device is much more at risk of being compromised, it is important to know about it. Detecting whether the device is rooted or not is essential for further security measures.	6.8 High
General Server Vulnerabilities API The API may be susceptible to general server vulnerabilities, which can lead to further attacks	7.4 High
Javascript CORS enabled in Webview Static Javascript in the Webview having CORS enabled to be loaded from file and any arbitrary URL	8.1 High
Hooking Detection Dynamic Hooking is a technique where an external piece of code intercepts or manipulates the normal execution flow of an application or the operating system. While legitimate uses of hooking exist, such as for debugging and development purposes, malicious applications may use hooking to compromise the security and integrity of the system. Hooking detection is a technique to detect whether an application is being hooked at run time.	5.7 Medium
WebView Exploits Dynamic WebView can be susceptible to various exploits including client side Javascript injection and network sniffing if improperly implemented.	5.4 Medium
Weak PRNG (Pseudorandom number generator) Static Weak PRNG vulnerabilities stem from insufficiently random initializations, improper algorithms, or inadequate entropy sources. Such vulnerabilities can result in unauthorized access, data breaches, and compromised cryptographic operations.	6.1 Medium

StrandHogg Vulnerability Static One or more than one public facing activities of the application is vulnerable to StandHogg vulnerability. In StrandHogg and regular task hijacking, malicious applications typically deploy one of the following techniques or a selection in tandem: ■ **Task Affinity Manipulation**: The malicious application leverages two activities, M1 and M2, wherein M2.taskAffinity = com.victim.app and M2.allowTaskReparenting = true. In the eventuality that the malicious app is opened on M2, M2 would be relocated to the front and the user will interact with the malicious application once the victim application has initiated. ■ **Single Task Mode**: In the eventuality that the victim application sets launchMode to singleTask, malicious applications can leverage M2.taskAffinity = com.victim.app to hijack the victim's application task stack. ■ **Task Reparenting**: In the eventuality that the victim application sets taskReparenting to true, malicious applications can move the victim's application task to the malicious application's stack. However, in the case of StrandHogg 2.0, all exported activities without a launchMode of singleTask or singleInstance are affected on vulnerable Android versions.	6.5 Medium
MediaProjection: Android Service Allows Recording of Audio, Screen Activity Static Protect all sensitive windows within the App by enabling the FLAG_SECURE flag. This flag will prevent Apps from being able to record the protected windows. Also, the flag will prevent users from taking screenshots of these windows (by pressing the VOLUME_DOWN and POWER buttons). As such screenshots are stored on the SDCard by default, they are accessible to all Apps and sensitive data may be exposed.	6.8 Medium
Disabled SSL CA Validation and Certificate Pinning Static A host or service's certificate or public key can be added to an application at development time, or it can be added upon first encountering the certificate or public key. The former - adding at development time - is preferred since preloading the certificate or public key out of band usually means the attacker cannot taint the pin.	5.9 Medium
Android Developer options status detection Dynamic Implementing Developer options protection in Android applications is crucial to ensure security and prevent unauthorized access to sensitive features and data. Implementing Developer option protection in your Android application enhances its security and integrity. By detecting and responding to the enabling of developer options, you can prevent unauthorized access and protect sensitive data.	3.4 Low

Android Developer Bridge(ADB) status detection Dynamic Implementing ADB detection in Android applications is crucial to ensure security and prevent unauthorized access to sensitive features and data. ADB detection enhances app security by guarding against unintended exposure, unauthorized access, and tampering. By detecting and responding to the enabling of ADB on the device, you can prevent unauthorized access and protect sensitive data.	**3.4** Low
Bytecode Obfuscation Static Java source code is typically compiled into Java bytecode – the instruction set of the Java virtual machine. The compiled Java bytecode can be easily reversed engineered back into source code by freely available decompilers. Bytecode Obfuscation is the process of modifying Java bytecode (executable or library) so that it is much harder to read and understand for a hacker but remains fully functional.	**2.3** Low
Enabled Android Application Backup Static Application backup might contain sensitive information private data of the app into their PC	**3.3** Low
Keylogger Protection Static Keyloggers are malicious software designed to capture and record keystrokes or input data from users, posing a significant threat to personal and sensitive information. They operate by monitoring keyboard activity, potentially stealing passwords, credit card details, and other confidential data. Keylogger protection involves implementing secure input methods, such as custom keyboards and virtual keyboards, to prevent unauthorized access to typed information. Effective protection ensures that even if a keylogger is present, it cannot intercept or compromise the data being entered.	**3.9** Low
Hardcoded Secrets Static No hardcoded keys or credentials were found **Note:** This vulnerability was manually overridden to Passed by aimran on 09 Oct 2025. This override is applied only to the current file. **Reason:** FP	Passed
Application Logs Static Dynamic No application logs were found. **Note:** This vulnerability was manually overridden to Passed by aimran on 09 Oct 2025. This override is applied only to the current file. **Reason:** FP	Passed

Insecure Content Security Policy API The Content Security Policy is configured with strong protections, effectively mitigating the risk of script injection and other attacks.	Passed
CORS Misconfigurations: Reflected Origin in response headers API The Origin header is validated and only allowed origins are reflected in the Access-Control-Allow-Origin header.	Passed
SSL/TLS Renegotiation Vulnerability API The the host is not vulnerable to SSL/TLS Re-Negotiation	Passed
TLS Protocol Downgrade Attack API The the host is not vulnerable to TLS Protocol Downgrade Attack	Passed
Sensitive information in Sqlite database Dynamic Application is not vulnerable to SQLITE db	Passed
Android Component Hijacking via Intent Static Application has no vulnerable components for Intent hijacking	Passed
Do not allow WebView to access sensitive local resource through file scheme Static Webview file-scheme is not being used or vulnerable	Passed
Android Fragment Injection Static Application is not vulnerable to Fragment Injection	Passed
CORS Misconfigurations: Wildcard Origin in response headers API No CORS wild character vulnerabilities were found in HTTP headers.	Passed
Cross-site-scripting Vulnerabilities in HTTP Body API No Cross-site-scripting vulnerabilities were found in HTTP body.	Passed
String Validation Vulnerabilities in HTTP Requests API No string validation vulnerabilities were found in HTTP requests.	Passed

SQL Injection Vulnerabilities in HTTP Requests API No SQL injection vulnerabilities were found in HTTP request.	Passed
Regex DoS Vulnerabilities in HTTP Requests API No Regex DoS vulnerabilities were found in HTTP request.	Passed
LDAP Injection Vulnerabilities in HTTP Requests API No LDAP injection vulnerabilities were found in HTTP request.	Passed
Integer Overflow Vulnerabilities in HTTP Requests API No integer overflow vulnerabilities were found in HTTP request.	Passed
Command Injection Vulnerabilities in HTTP Requests API No command injection vulnerabilities in HTTP request were found.	Passed
Insecure Hashing Algorithms Dynamic No insecure hash functions are used.	Passed
Non-signature Protected Exported Services Static The application does not export any service without a ProtectionLevel.	Passed
Unprotected Exported Receivers Static The application does not export Broadcast Receivers.	Passed
PhoneGap Debug Logging Static PhoneGap debug logs are not enabled, or PhoneGap is not being used.	Passed
PhoneGap Error URL Redirection Vulnerability Static The application is not vulnerable to error URL redirection vulnerability, or is not using PhoneGap.	Passed
PhoneGap JavaScript Injection Static The application does not seem to be affected by PhoneGap javascript injection.	Passed

Insufficient Transport Layer Protection Static Dynamic Application seems to have SSL enabled and sufficient protection is being used to prevent Information leakage.	Passed
App Extending WebViewClient Static Application seems to be handling WebViewClient correctly, or has no implementation of WebViewClient.	Passed
Cloud Storage Bucket - Config & Metadata Exposure API The cloud storage bucket has proper configuration and metadata protection in place.	Passed
Broken SSL Trust Manager Static Trust Managers for SSL, if any, seem to be properly implemented.	Passed
Unprotected Services Static Exported services in the app are sufficiently protected.	Passed
Cloud Storage Bucket - Sensitive Data Leaks API The cloud storage bucket has proper protection in place for sensitive data.	Passed
Cloud Storage Bucket - Data Access & Tampering API The cloud storage bucket has proper data access and tampering protection in place.	Passed
Cloud Storage Bucket - Enumeration & Access Control API The cloud bucket demonstrates proper security measures by implementing robust access control mechanisms and restricting bucket enumeration, ensuring resilience against unauthorized access.	Passed
CORS Misconfigurations: Dynamic Origin in response headers API The server employs safe and explicit origin validation instead of regex patterns.	Passed
CORS Misconfigurations: Null Origin in response headers API The server does not allow the null origin in CORS requests, mitigating potential abuse.	Passed

Forbidden Error Bypass API The server does not allow unauthorized access or bypassing of 403 Forbidden responses, indicating that the API access controls are functioning as expected.	Passed
JWT None algorithm API The host is not vulnerable to JWT None Algorithm vulnerability.	Passed
Intent Redirection Vulnerability Static Application is not vulnerable to intent redirection vulnerability.	Passed
Keyboard Cache Exposure Static The application takes proper security measure to disable caching for sensitive input fields and hence is not vulnerable to keyboard cache exposure.	Passed
Android Tapjacking Static The application has implemented proper security checks and measures in place and hence is not vulnerable to tapjacking.	Passed
Janus Vulnerability (CVE-2017-13156) Static Application is signed using v2 signature scheme or above, and is not vulnerable to Janus.	Passed
Network Security Misconfiguration Static The application has properly configured network security configuration	Passed
TLS/SSL CRIME Attack API The the host is not vulnerable to TLS/SSL CRIME Attack	Passed
TLS ROBOT Attack API The the host is not vulnerable to ROBOT Attack	Passed
Heartbleed Vulnerability API The the host is not vulnerable to HEARTBLEED bug	Passed
OpenSSL CCS Injection Vulnerability API The the host is not vulnerable to OpenSSL CCS Injection	Passed

HTTP Host Header Injection API The the host is not vulnerable to Host header injection attacks	Passed
HTTP TRACE method is enabled API The application backend server doesn't have TRACE method enabled	Passed
Deprecated setPluginState in WebView Static The application does not uses setPluginState in WebView.	Passed
Java Object Deserialization Vulnerability Static The application is not vulnerable to object deserialization.	Passed
External data in raw SQL queries Static The application does not seem to use raw SQL query.	Passed
Insecure Broadcast Receivers registered dynamically Static There is no insecure Dynamically registered Broadcast Receiver	Passed
Surreptitious Sharing on Android Static Surreptitious Sharing might not be in use in this application	Passed
Sending Address Book Data over Unencrypted Insecure Transport Layer Static The Application doesn't sends Address Book over Insecured Network	Passed
Response Body Contains Non-HTTPS Links API Response body does not contain non-https links.	Passed
Cross Site Tracing Vulnerabilities API No Cross Site Tracing vulnerabilities were found.	Passed
XML-external-entity Injection Vulnerabilities in HTTP Body API No XML-external-entity injection vulnerabilities were found in HTTP body.	Passed

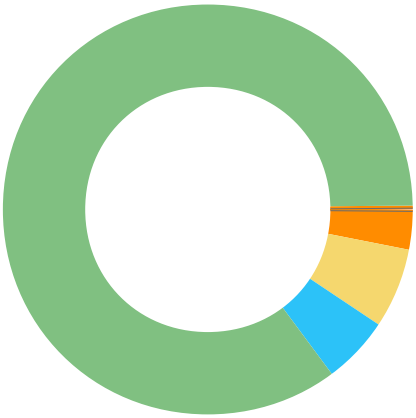
JSON Depth Overflow in HTTP Requests API No JSON depth overflows were found in HTTP request.	Passed
Buffer Overflow Vulnerabilities in HTTP Requests API Buffer overflow vulnerabilities were not detected in HTTP request.	Passed
Content Provider File Traversal Vulnerability Static Application does not seem to be vulnerable to content provider directory traversal attacks.	Passed
Non-signature Protected Exported Providers Static The application does not export any provider without a ProtectionLevel.	Passed
Non-signature Protected Exported Receivers Static The application does not export any Broadcast Receivers without a ProtectionLevel.	Passed
Non-signature Protected Exported Activities Static The application does not export any activity without a ProtectionLevel.	Passed
Unprotected Exported Provider Static The application does not export any providers insecurely.	Passed
Unprotected Exported Service Static The application does not export any services insecurely.	Passed
Unprotected Exported Activities Static The application does not insecurely export any activities.	Passed
PhoneGap Whitelisted URLs Static Application has proper whitelisted URLs, or does not use PhoneGap.	Passed
Cordova Remote Start Page Manipulation Vulnerability Static The application is not affected by CVE-3500, or Apache Cordova is not being used.	Passed

PhoneGap HTTPS Whitelist Bypass Static The PhoneGap application seems to be using proper regex check, or is not using PhoneGap.	Passed
Connection to External Redis Server Static The application does not communicate with an external Redis server.	Passed
PhoneGap HTTPS Bypass Vulnerability Static The application does not use a vulnerable version of PhoneGap, or is not using PhoneGap.	Passed
Remote URL Redirection Vulnerability Static The application is not vulnerable to URL redirection vulnerability, or is not using PhoneGap.	Passed
Derived Crypto Keys Static Dynamic Application seems to be using the correct cryptographic encryption method, if any.	Passed
Storing Information in Shared Preferences Dynamic No leakage of data were found via Shared Preference.	Passed
JavascriptInterface Remote Code Execution Static Application is safe from remote code execution through JavascriptInterface.	Passed
Unused Permissions Static Application seems to have just the essential set of permissions required.	Passed
HostnameVerifier Allowing All Hostnames Static AllowAllHostname is properly configured, or is disabled.	Passed
Insecure SSLSocketFactories Static No implementation errors were found.	Passed
Broken HostnameVerifier for SSL Static HostnameVerifiers for SSL, if any, seem to be verifying hostnames properly.	Passed

Improper Custom Permissions Static Custom permissions in the app have sufficient protection levels, or custom permissions are absent.	Passed
Application Debugging Static Debug was found to be disabled.	Passed
Improper Content Provider Permissions Static Application seems to properly implement SSL, or HTTPS is not implemented.	Passed

Priority Level	Number of failed test cases
Critical Risk	0
High Risk	3
Medium Risk	6
Low Risk	5

Security Rating



Critical	**0.0%**
High	**3.16%**
Medium	**6.32%**
Low	**5.26%**
Passed	**85.26%**
Untested	**0.0%**

Security Rating - **14.74 Unsecured**

Out of all the Passed vulnerabilities, 2 have been manually overridden to 'Passed'.

Root Detection

Root detection test case detects whether the application is running on a rooted device or not. If it can then a malicious application can access or modify the data of any application.

Risk Rating

High

Scan Type

Dynamic

This risk has been overridden from **Medium** to **High** by **aimran**. on **02 Oct 2025**
The reason is: "Root detection is mandatory"

CVSS

Version 3.0 Base Score **6.8**	Attack vector: **PHYSICAL**	Attack complexity: **HIGH**
	Privileges required: **HIGH**	User Interaction: **NONE**
	Scope: **CHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **HIGH**	Availability Impact: **LOW**

Regulatory

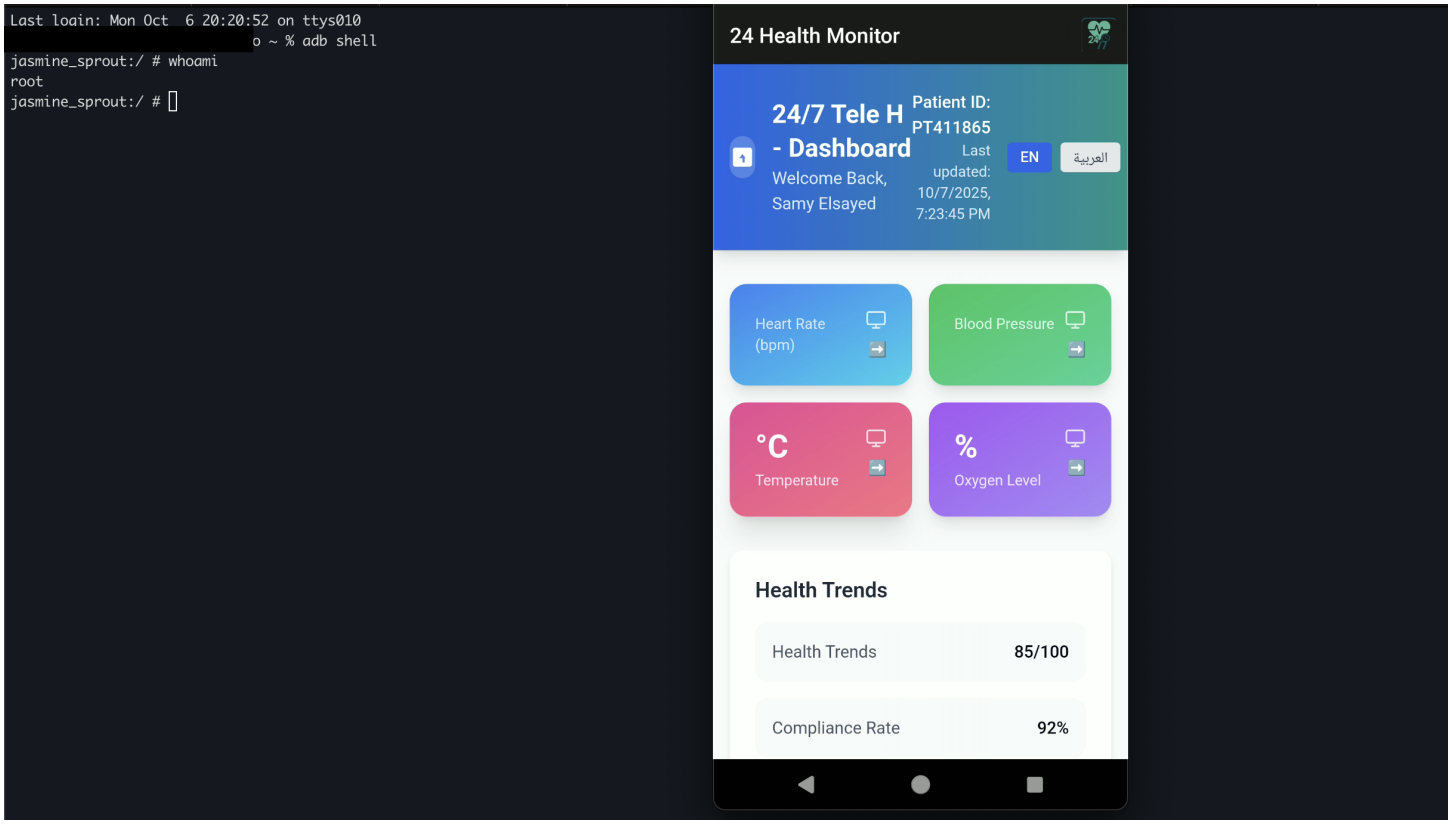
OWASP Mobile Top 10 (2024)	M7	Insufficient Binary Protections
CWE	CWE-693	https://cwe.mitre.org/data/definitions/693.html
MSTG	MSTG-RESILIENCE-1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.
OWASP MASVS (v2)	MASVS-RESILIANCE-1	The app validates the integrity of the platform.
PCI-DSS (v4.0)	7.1	**Restrict access to cardholder data by business need to know** Processes and mechanisms for restricting access to system components and cardholder data by business need to know are defined and understood
	7.2	**Restrict access to cardholder data by business need to know** Access to system components and data is appropriately defined and assigned

HIPAA	164.308(a)(4)	**Administrative Safeguards: Information Access Management** ■ **Isolating Health Care Clearinghouse Functions (Required)** If a health care clearinghouse is part of a larger organization, the clearinghouse must implement policies and procedures that protect the electronic protected health information of the clearinghouse from unauthorized access by the larger organization. ■ **Access Authorization (Addressable)** Implement policies and procedures for granting access to electronic protected health information, for example, through access to a workstation, transaction, program, process, or other mechanism. ■ **Access Establishment and Modification (Addressable)** Implement policies and procedures that, based upon the entity's access authorization policies, establish, document, review, and modify a user's right of access to a workstation, transaction, program, or process.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Since a rooted device is much more at risk of being compromised, it is important to know about it. Detecting whether the device is rooted or not is essential for further security measures.

```
The application has not properly implemented root detection.
```



Screenshot 2025-10-07 at 21.50.25.png

Compliant Solution

```
import android.content.Context;
import android.content.pm.ApplicationInfo;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.util.Log;
import android.content.pm.PackageManager;
import java.io.BufferedReader;
import java.io.File;
import java.io.InputStreamReader;
import java.util.List;

public class RootCheck {
    private static Context mContext;
    private static final String TAG = "RootCheck";
    private String[] binaryPaths= {
            "/data/local/",
            "/data/local/bin/",
            "/data/local/xbin/",
            "/sbin/",
            "/su/bin/",
            "/system/bin/",
            "/system/bin/.ext/",
            "/system/bin/failsafe/",
            "/system/sd/xbin/",
            "/system/usr/we-need-root/",
```

```
                "/system/xbin/",
                "/system/app/Superuser.apk",
                "/cache",
                "/data",
                "/dev"
        };

        private String[] dangerousPackages = {
                "com.devadvance.rootcloak",
                "com.devadvance.rootcloakplus",
                "com.koushikdutta.superuser",
                "com.thirdparty.superuser",
                "com.topjohnwu.magisk",
                "org.lsposed.manager",
                "com.devadvance.rootcloak2"
        };

        RootCheck(Context ct) {
            mContext = ct;
        }

        public String rootBeerCheck() {
            if(DetectTestKeys() || checkSuExists() || checkForBusyBoxBinary() || checkForSuBinary()
    || checkPackages(mContext))
                return "DEVICE IS ROOTED!";
            else
                return "DEVICE IS NOT ROOTED";
        }

        private boolean DetectTestKeys()  {
            String buildTags = android.os.Build.TAGS;
            return buildTags != null && buildTags.contains("test-keys");
        }

        private boolean checkForBinary(String filename) {
            for (String path : binaryPaths) {
                File f = new File(path, filename);
                boolean fileExists = f.exists();
                if (fileExists) {
                    return true;
                }
            }
            return false;
        }

        private boolean checkForSuBinary() {
            return checkForBinary("su");    //checking for su binary
        }

        private boolean checkForBusyBoxBinary() {
            return checkForBinary("busybox"); //checking for busybox
        }

        private boolean checkSuExists() {
```

```
        Process process = null;
        try {
            process = Runtime.getRuntime().exec(new String[]
                    {"/system /xbin/which", "su"}); //Checking if su binary exists
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(process.getInputStream()));
            String line = in.readLine();
            process.destroy();
            return line != null;
        } catch (Exception e) {
            if (process != null) {
                process.destroy();
            }
            return false;
        }
    }

    private boolean checkPackages(Context ctx)  {
        PackageManager pm = ctx.getPackageManager();
        for(String name:dangerousPackages){
            if(isPackageInstalled(name,pm)){    //Checking if dangerous applications are
installed
                return true;
            }
        }
        return false;
    }

    private boolean isPackageInstalled(String packageName, PackageManager packageManager) {
        try {
            return packageManager.getApplicationInfo(packageName, 0).enabled;
        } catch (PackageManager.NameNotFoundException e) {
            return false;
        }
    }

}
```

Business Implication

In the event, if an attacker can install a malicious application on the device then an attacker can perform malicious activities which can result in compromising the victim's data.

Related Vulnerabilities

- RootBear Library - Simple to use root checking Android library and sample app (https://github.com/scottyab/rootbeer)

General Server Vulnerabilities

This issue is not specific to a certain kind of vulnerabilities. It can be raised as a result of many different types of attacks and might indicate some server-side fault that may lead to further vulnerabilities

When an attacker explores a web site looking for vulnerabilities, the amount of information that the site provides is crucial to the eventual success or failure of any attempted attacks. If the application shows the attacker a stack trace, it relinquishes information that makes the attacker's job significantly easier. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

The application configuration should specify a default error page in order to guarantee that the application will never leak error messages to an attacker. Handling standard HTTP error codes is useful and user-friendly in addition to being a good security practice, and a good configuration will also define a last-chance error handler that catches any exception that could possibly be thrown by the application.

Risk Rating

High

Scan Type

API

CVSS

Version 3.0 Base Score **7.4**	Attack vector: **NETWORK**	Attack complexity: **HIGH**
	Privileges required: **NONE**	User Interaction: **NONE**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **HIGH**	Availability Impact: **NONE**

Regulatory

CWE	CWE-16	https://cwe.mitre.org/data/definitions/16.html
ASVS	V13.1.3	Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.
PCI-DSS (v4.0)	6.2	**Develop and maintain secure systems and applications** Bespoke and custom software are developed securely

HIPAA	164.312(c)(1)	**Technical Safeguards: Integrity** ■ **Mechanism to Authenticate Electronic Protected (Addressable)** Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner.
	164.312(a)(1)	**Technical Safeguards: Access Control** ■ **Unique User Identification (Required)** Assign a unique name and/or number for identifying and tracking user identity ■ **Emergency Access Procedure (Required)** Establish (and implement as needed) procedures for obtaining necessary electronic protected health information during an emergency. ■ **Automatic Logoff (Addressable)** Implement electronic procedures that terminate an electronic session after a predetermined time of inactivity. ■ **Encryption and Decryption (Addressable)** Implement a method to encrypt and decrypt electronic protected health information.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

The API may be susceptible to general server vulnerabilities, which can lead to further attacks

Weak Secret Key Use to Sign JWT

```
We identified that JWTs are signed using a weak or predictable secret key, which allows
attackers to forge or tamper tokens (by brute-forcing the HMAC secret or exploiting algorithm
confusion), enabling privilege escalation, account takeover, or unauthorized API access.
```

PoC

```
Please Refer to Screenshot Attached
```

JWT-Token /w Secret Key

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIxLCJlbWFpbCI6InNhbXlhYmRvMjAyMjFAZ21haWwuY29
tIiwicm9sZSI6InBhdGllbnQiLCJpYXQiOjE3NTk4NDU4MTYsImV4cCI6MTc1OTkzMjIxNn0.KfRqqsyM_c8qhiKD5yCKqE
GFwrTYWzzVI0gaQOH9kGg


Secret Key: "your-secret-key"
```

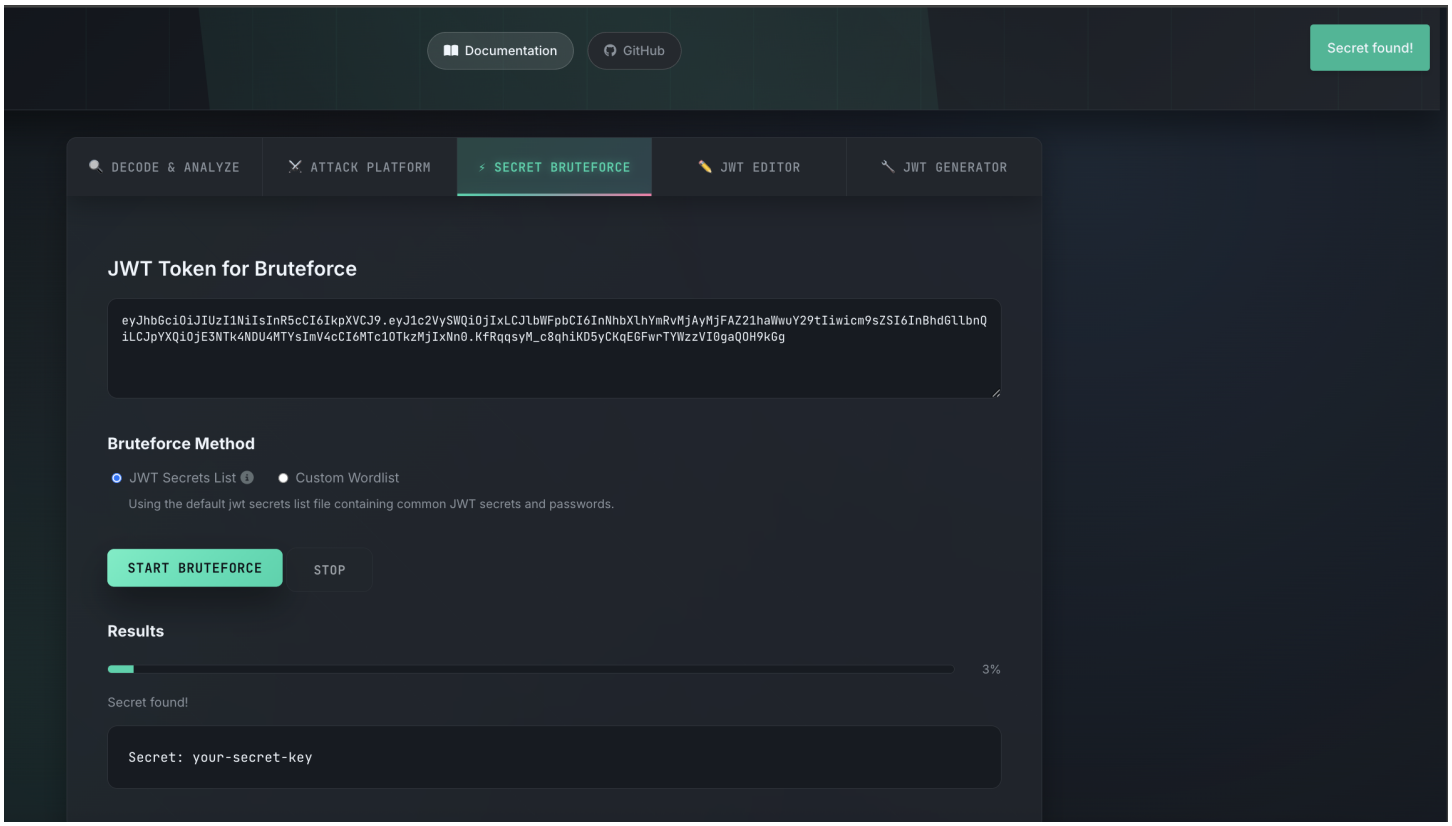
Steps To Reproduce

```
1. Log in to the application.
2. Retrieve the generated authentication token from the logged-in session.
```

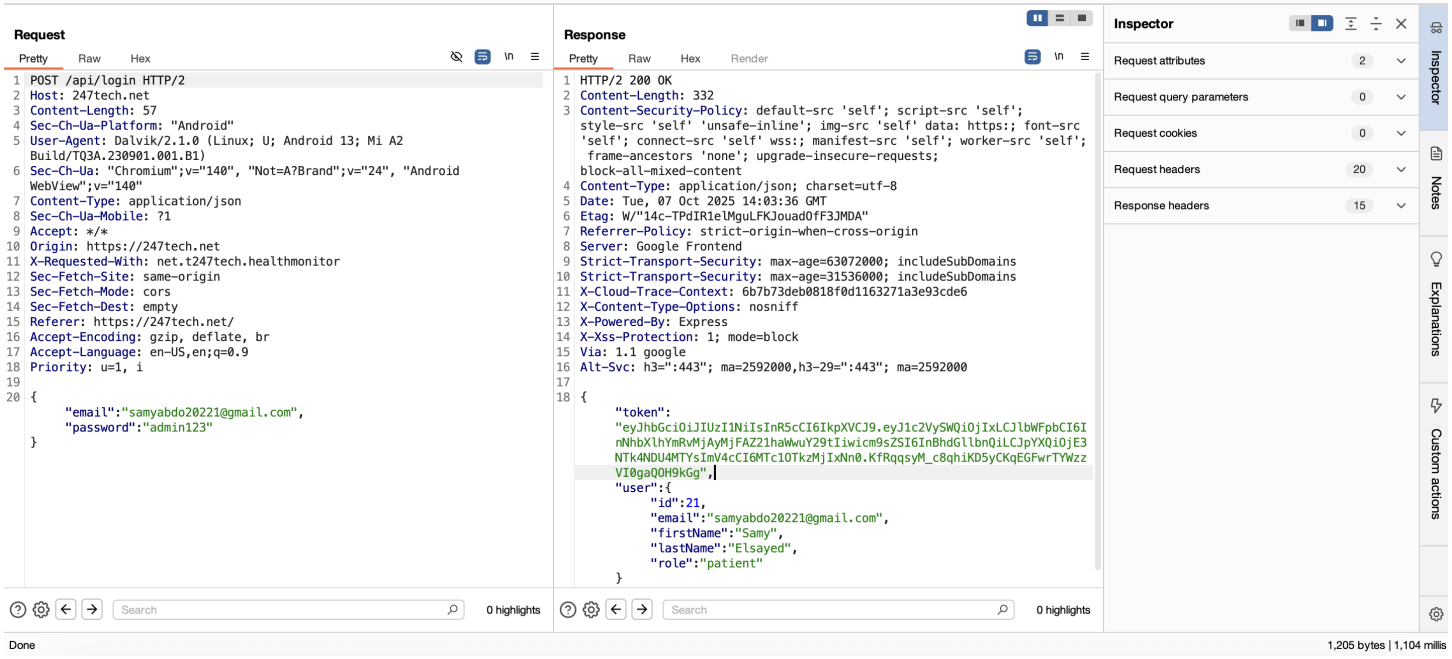
3. Submit the token to https://jwtauditor.com/
 and attempt to brute-force the signing key.
4. Observe the result indicating the token is signed with a weak secret key.

Remediation

Use strong, randomly generated secrets of sufficient entropy (≥256 bits) for HMAC (HS256) or—preferably—switch to asymmetric signing (RS256/ES256) and protect the private key.



jwt1.png



jwt2.png

Javascript CORS enabled in Webview

Cross-Origin Resource Sharing (CORS) is enabled in WebView. JavaScript used in mobile application can send and receive data from arbitrary remote hosts. This can be a risk if the remote host is impersonated or compromised.

Risk Rating

High

Scan Type

Static

CVSS

Version 3.0 Base Score **8.1**	Attack vector: **NETWORK**	Attack complexity: **LOW**
	Privileges required: **NONE**	User Interaction: **REQUIRED**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **HIGH**	Availability Impact: **NONE**

Regulatory

OWASP Mobile Top 10 (2024)	M4	Insufficient Input/Output Validation
	M8	Security Misconfiguration
CWE	CWE-942	https://cwe.mitre.org/data/definitions/942.html
MSTG	MSTG-PLATFORM-6	WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.
OWASP MASVS (v2)	MASVS-PLATFORM-2	The app uses WebViews securely.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Javascript in the Webview having CORS enabled to be loaded from file and any arbitrary URL

```
setAllowUniversalAccessFromFileURLs is set to True which is Insecured and found in Lnet/t247tech/healthmonitor/MainActivity;->onCreate (Landroid/os/Bundle;)V
```

```
setAllowFileAccessFromFileURLs is set to True which is Insecured and found in Lnet/t247tech/healthmonitor/MainActivity;->onCreate (Landroid/os/Bundle;)V
```

res/menu/main_menu.xml | res/xml/provider_paths.xml | AndroidManifest.xml | MainActivity

```
                layoutInflaterFactory2C0013C.f1534m.b = null;
            }
            layoutInflaterFactory2C0013C.b();
        }
        if (Build.VERSION.SDK_INT >= 35) {
            getWindow().getDecorView().setOnApplyWindowInsetsListener(new i0.d());
            getWindow().setDecorFitsSystemWindows(false);
            WindowInsetsController insetsController = getWindow().getInsetsController();
            if (insetsController != null) {
                insetsController.setSystemBarsAppearance(0, 8);
            }
        }
        this.f1539A = getSharedPreferences("net.t247tech.healthmonitor", 0);
        this.f1548x = (WebView) findViewById(R.id.activity_main_webview);
        View viewFindViewById = findViewById(R.id.container);
        i0.e eVar = new i0.e();
        WeakHashMap weakHashMap = K.f1647a;
        AbstractC0151A.u(viewFindViewById, eVar);
        this.f1548x.setWebChromeClient(new f(this));
        this.f1542D = (SwipeRefreshLayout) findViewById(R.id.swipeRefreshLayout);
        this.f1543E = (SwipeRefreshLayout) findViewById(R.id.swipeRefreshLayout);
        this.f1548x.getSettings().setDomStorageEnabled(true);
        this.f1548x.setDownloadListener(new i0.g(this));
        this.f1548x.setWebViewClient(new i0.h(this));
        WebView webView2 = this.f1548x;
        WebSettings settings = webView2.getSettings();
        settings.setJavaScriptEnabled(true);
        settings.setDomStorageEnabled(true);
        settings.setSupportMultipleWindows(false);
        settings.setJavaScriptCanOpenWindowsAutomatically(true);
        settings.setAllowFileAccess(true);
        settings.setBuiltInZoomControls(false);
        settings.setDisplayZoomControls(false);
        settings.setLoadWithOverviewMode(true);
        settings.setUseWideViewPort(true);
        settings.setAllowFileAccessFromFileURLs(true);
        settings.setAllowUniversalAccessFromFileURLs(true);
        settings.setSupportZoom(true);
        settings.setDatabaseEnabled(true);
        settings.setUserAgentString(System.getProperty("http.agent"));
        b bVar = new b();
        bVar.f1461a = this;
        webView2.addJavascriptInterface(bVar, "Android");
        if (!q(this)) {
            this.f1548x.loadUrl("file:///android_asset/htmlapp/helpers/error.html");
```

Screenshot 2025-10-07 at 21.35.33.png

Noncompliant Code Example

Example of an insecure code :

```
WebSettings settings = getSettings();
settings.setAllowUniversalAccessFromFileURLs(true)
settings.setAllowFileAccessFromFileURLs(true)
```

Compliant Solution

Below is an example of how to prevent the application from Javascript CORS issue:

```
WebSettings settings = getSettings();
settings.setAllowUniversalAccessFromFileURLs(false)
settings.setAllowFileAccessFromFileURLs(false)
```

Related Vulnerabilities

- Android Webkit Settings Docs

Hooking Detection

Hooking detection test case checks whether an application can detect if it's being hooked or not. If an application is being hooked, the application behaviour can be modified at run time.

Risk Rating

Medium

Scan Type

Dynamic

CVSS

Version 3.0 Base Score **5.7**	Attack vector: **PHYSICAL**	Attack complexity: **LOW**
	Privileges required: **LOW**	User Interaction: **REQUIRED**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **HIGH**	Availability Impact: **NONE**

Regulatory

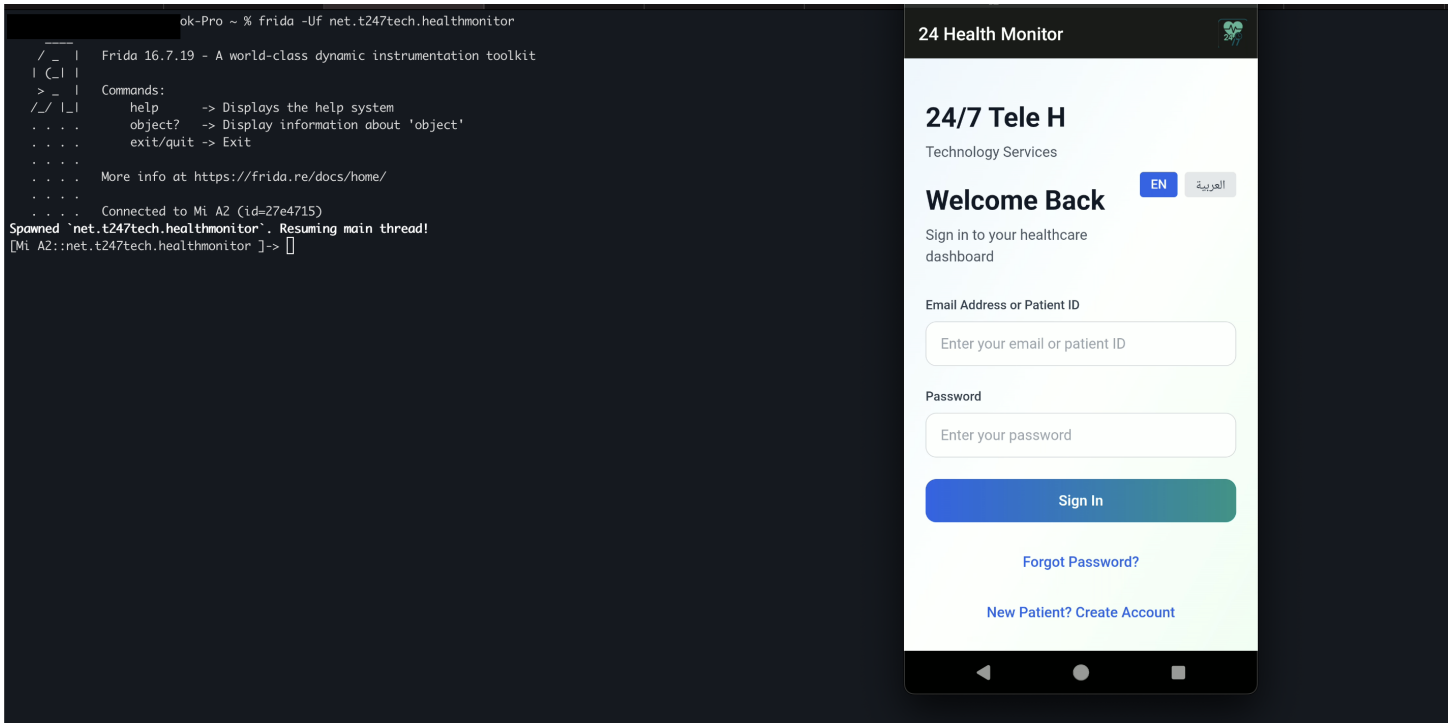
OWASP Mobile Top 10 (2024)	M7	Insufficient Binary Protections
CWE	CWE-693	https://cwe.mitre.org/data/definitions/693.html
MSTG	MSTG-RESILIENCE-1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.
OWASP MASVS (v2)	MASVS-RESILIANCE-1	The app validates the integrity of the platform.
PCI-DSS (v4.0)	7.1	**Restrict access to cardholder data by business need to know** Processes and mechanisms for restricting access to system components and cardholder data by business need to know are defined and understood
	7.2	**Restrict access to cardholder data by business need to know** Access to system components and data is appropriately defined and assigned

HIPAA	164.308(a)(4)	**Administrative Safeguards: Information Access Management** ■ **Isolating Health Care Clearinghouse Functions (Required)** If a health care clearinghouse is part of a larger organization, the clearinghouse must implement policies and procedures that protect the electronic protected health information of the clearinghouse from unauthorized access by the larger organization. ■ **Access Authorization (Addressable)** Implement policies and procedures for granting access to electronic protected health information, for example, through access to a workstation, transaction, program, process, or other mechanism. ■ **Access Establishment and Modification (Addressable)** Implement policies and procedures that, based upon the entity's access authorization policies, establish, document, review, and modify a user's right of access to a workstation, transaction, program, or process.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Hooking is a technique where an external piece of code intercepts or manipulates the normal execution flow of an application or the operating system. While legitimate uses of hooking exist, such as for debugging and development purposes, malicious applications may use hooking to compromise the security and integrity of the system. Hooking detection is a technique to detect whether an application is being hooked at run time.

```
The application has not implemented hooking detection or any kind of runtime protection system.
```



Screenshot 2025-10-07 at 21.54.10.png

Compliant Solution

To implement a hooking detection solution, the following code (in Java or Kotlin) can be used to enumerate packages and figure out malicious tools. This should be used in conjunction with the native (C/C++) code to make detection more stronger and efficient.

Java:

```
function checkSuspiciousMethodsinStackTrace() {
try {
    Set libraries = new HashSet();
    String mapsFilename = "/proc/" + android.os.Process.myPid() + "/maps";
    BufferedReader reader = new BufferedReader(new FileReader(mapsFilename));
    String line;
    while((line = reader.readLine()) != null) {
        if (line.endsWith(".so") || line.endsWith(".jar")) {
            int n = line.lastIndexOf(" ");
            libraries.add(line.substring(n + 1));
        }
    }
    for (String library : libraries) {
        if(library.contains("com.saurik.substrate")) {
            Log.wtf("HookDetection", "Substrate shared object found: " + library);
        }
        if(library.contains("XposedBridge.jar")) {
            Log.wtf("HookDetection", "Xposed JAR found: " + library);
        }
    }
    reader.close();
}
catch (Exception e) {
```

```
        Log.wtf("HookDetection", e.toString());
    }
    }
    function checkSuspiciousPackagesInstalled() {
        PackageManager packageManager = context.getPackageManager();
    List applicationInfoList  =
    packageManager.getInstalledApplications(PackageManager.GET_META_DATA);
    for(ApplicationInfo applicationInfo : applicationInfoList) {
        if(applicationInfo.packageName.equals("de.robv.android.xposed.installer")) {
            Log.wtf("HookDetection", "Xposed found on the system.");
        }
        if(applicationInfo.packageName.equals("com.saurik.substrate")) {
            Log.wtf("HookDetection", "Substrate found on the system.");
        }
    }
    }
```

Kotlin:

```
    import android.content.pm.ApplicationInfo
    import android.content.pm.PackageManager
    import android.util.Log
    import java.io.BufferedReader
    import java.io.FileReader
    import java.util.HashSet
    fun checkSuspiciousMethodsInStackTrace() {
        try {
            val libraries = mutableSetOf<String>()
            val mapsFilename = "/proc/${android.os.Process.myPid()}/maps"
            val reader = BufferedReader(FileReader(mapsFilename))
            var line: String?
            while (reader.readLine().also { line = it } != null) {
                if (line!!.endsWith(".so") || line.endsWith(".jar")) {
                    val n = line!!.lastIndexOf(" ")
                    libraries.add(line!!.substring(n + 1))
                }
            }
            for (library in libraries) {
                if (library.contains("com.saurik.substrate")) {
                    Log.wtf("HookDetection", "Substrate shared object found: $library")
                }
                if (library.contains("XposedBridge.jar")) {
                    Log.wtf("HookDetection", "Xposed JAR found: $library")
                }
            }
            reader.close()
        } catch (e: Exception) {
            Log.wtf("HookDetection", e.toString())
        }
    }
    fun checkSuspiciousPackagesInstalled(context: Context) {
        val packageManager: PackageManager = context.packageManager
        val applicationInfoList: List<ApplicationInfo> =
    packageManager.getInstalledApplications(PackageManager.GET_META_DATA)
        for (applicationInfo in applicationInfoList) {
```

```
            if (applicationInfo.packageName == "de.robv.android.xposed.installer") {
                Log.wtf("HookDetection", "Xposed found on the system.")
            }
            if (applicationInfo.packageName == "com.saurik.substrate") {
                Log.wtf("HookDetection", "Substrate found on the system.")
            }
        }
    }
```

Native (via Java Native Interface) in C/C++:

```
    static char keyword[] = "LIBFRIDA";
    num_found = 0;
    int scan_executable_segments(char * map) {
        char buf[512];
        unsigned long start, end;
        sscanf(map, "%lx-%lx %s", &start, &end, buf);
        if (buf[2] == 'x') {
            return (find_mem_string(start, end, (char*)keyword, 8) == 1);
        } else {
            return 0;
        }
    }
    void scan() {
        if ((fd = my_openat(AT_FDCWD, "/proc/self/maps", O_RDONLY, 0)) >= 0) {
        while ((read_one_line(fd, map, MAX_LINE)) > 0) {
            if (scan_executable_segments(map) == 1) {
                num_found++;
            }
        }
        if (num_found > 1) {
            /* Frida Detected */
        }
    }
    boolean is_frida_server_listening() {
        struct sockaddr_in sa;
        memset(&sa, 0, sizeof(sa));
        sa.sin_family = AF_INET;
        sa.sin_port = htons(27042);
        inet_aton("127.0.0.1", &(sa.sin_addr));
        int sock = socket(AF_INET , SOCK_STREAM , 0);
        if (connect(sock , (struct sockaddr*)&sa , sizeof sa) != -1) {
        /* Frida server detected. Do something… */
        }
    }
    public boolean checkRunningProcesses() {
    boolean returnValue = false;
    // Get currently running application processes
    List<RunningServiceInfo> list = manager.getRunningServices(300);
    if(list != null){
        String tempName;
        for(int i=0;i<list.size();++i){
        tempName = list.get(i).process;
        if(tempName.contains("fridaserver")) {
            returnValue = true;
```

```
        }
        }
    }
    return returnValue;
    }
```

Business Implication

Malicious applications may use hooking to compromise the security and integrity and bypass buiness logic checks performed by the application.

WebView Exploits

Android API offers WebView to deliver a web application (or just a web page) as a part of a client app. A common scenario in which using WebView is helpful is when you want to provide information in your application that you might need to update, such as an end-user agreement or a user guide. Another scenario is if your application provides data to the user that always requires an Internet connection to retrieve data, such as email. In this case, you might find that it's easier to build a WebView in your Android application that shows a web page with all the user data, rather than performing a network request, then parsing the data and rendering it in an Android layout.

Risk Rating

Medium

Scan Type

Dynamic

CVSS

Version 3.0 Base Score **5.4**	Attack vector: **NETWORK**	Attack complexity: **HIGH**
	Privileges required: **LOW**	User Interaction: **REQUIRED**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **LOW**	Availability Impact: **NONE**

Regulatory

CWE	CWE-749	https://cwe.mitre.org/data/definitions/749.html
MSTG	MSTG-PLATFORM-5	JavaScript is disabled in WebViews unless explicitly required.
OWASP MASVS (v2)	MASVS-PLATFORM-2	The app uses WebViews securely.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

WebView can be susceptible to various exploits including client side Javascript injection and network sniffing if improperly implemented.

```
Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
```

```
Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
```

URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

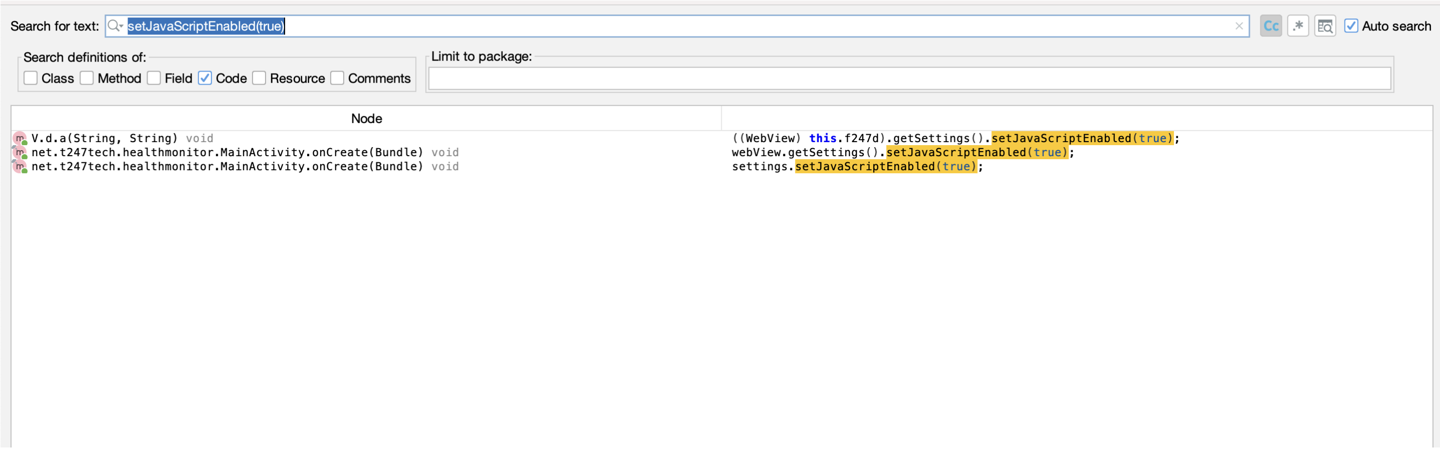
Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

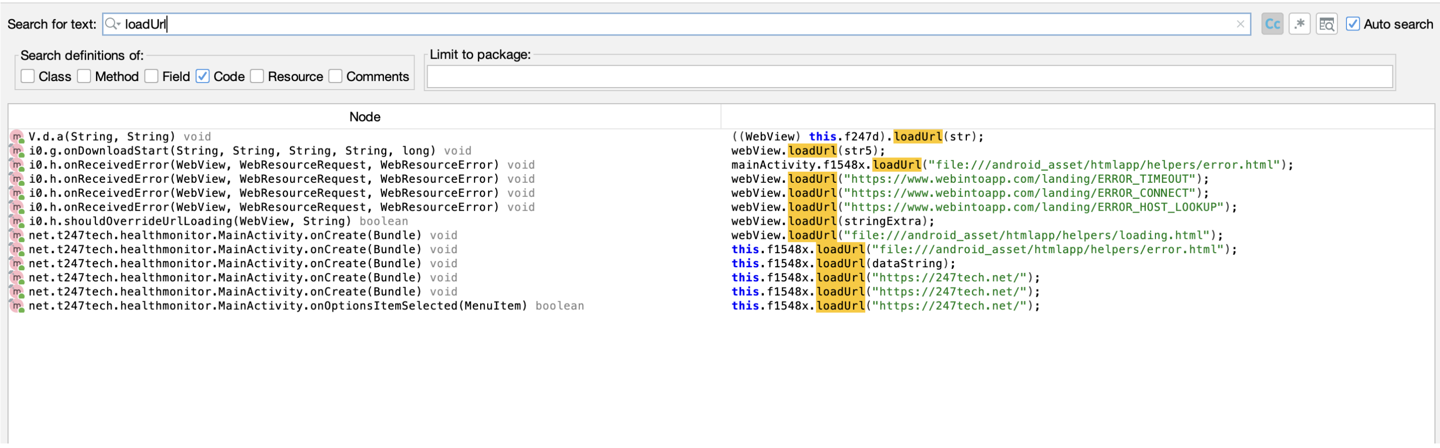
Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

```
Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
URL not loaded over HTTPS: file:///android_asset/htmlapp/helpers/loading.html
Attempt to load local resource: file:///android_asset/htmlapp/helpers/loading.html

Reference: android.webkit.WebView.loadUrl
Javascript enabled within WebView
```



Screenshot 2025-10-07 at 19.49.48.png



Screenshot 2025-10-07 at 20.02.20.png

Compliant Solution

When using WebView, ensure the following:

- Use WebView to load only trusted content
- Always load resources over HTTPS
- Avoid using Javascript within WebView. If Javascript is absolutely required, be sure that each context is escaped properly by using an XSS filter component such as the OWASP Java Encoder Project
- Accept only plain-text user input and sanitize it before displaying in WebView

Business Implication

An improperly implemented WebView instance may be vulnerable to XSS can be used to gain access to shared preference files using `file:///`. When Javascript is enabled, it may allow adversaries to perform XSS attacks. Furthermore, not loading WebView over HTTPS may allow attackers to sniff data from network transmissions and perform Man-in-the-Middle attack by injecting arbitrary JavaScript into the WebView.

Weak PRNG (Pseudorandom number generator)

Weak PRNG (Pseudorandom Number Generator) vulnerability in Android refers to a security weakness where the algorithms responsible for generating random numbers within an application lack the required level of unpredictability and randomness.

Risk Rating

Medium

Scan Type

Static

CVSS

Version 3.0 Base Score **6.1**	Attack vector: **PHYSICAL**	Attack complexity: **LOW**
	Privileges required: **NONE**	User Interaction: **NONE**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **HIGH**	Availability Impact: **NONE**

Regulatory

OWASP Mobile Top 10 (2024)	M10	Insufficient Cryptography
CWE	CWE-338	https://cwe.mitre.org/data/definitions/338.html
MSTG	MSTG-CRYPTO-6	All random values are generated using a sufficiently secure random number generator.
OWASP MASVS (v2)	MASVS-CRYPTO-1	The app employs current strong cryptography and uses it according to industry best practices.
PCI-DSS (v4.0)	6.1	**Develop and maintain secure systems and applications** Processes and mechanisms for developing and maintaining secure systems and software are defined and understood
	6.3	**Develop and maintain secure systems and applications** Security vulnerabilities are identified and addressed

HIPAA	164.312(c)(1)	**Technical Safeguards: Integrity** ■ **Mechanism to Authenticate Electronic Protected (Addressable)** Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner.
GDPR	Art-32-GDPR	Security of processing
	Art-25-GDPR	Data protection by design and by default

Risk Assessment

Weak PRNG vulnerabilities stem from insufficiently random initializations, improper algorithms, or inadequate entropy sources. Such vulnerabilities can result in unauthorized access, data breaches, and compromised cryptographic operations.

Activity := Le0/b;

```
An activity was found using an insecure randomization function via the Random class in Le0/b;->initialValue()Ljava/lang/Object;. As a result, the generated random data will be predictable and an attacker can guess or infer sensitive information
```

Activity := Lnet/t247tech/healthmonitor/MainActivity;

```
An activity was found using an insecure randomization function via the Random class in Lnet/t247tech/healthmonitor/MainActivity;->onCreate(Landroid/os/Bundle;)V. As a result, the generated random data will be predictable and an attacker can guess or infer sensitive information
```

Activity := LL/h;

```
An activity was found using an insecure randomization function via the Random class in LL/h;->run()V. As a result, the generated random data will be predictable and an attacker can guess or infer sensitive information
```

```
@Override // e.AbstractActivity$b221, androidx.activity.f, b.h, android.app.Activity
public final void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    setContentView(R.layout.activity_main);
    setRequestedOrientation(-1);
    g gVar = new g();
    gVar.f255d = new Handler();
    gVar.f256e = true;
    gVar.f258h = "https://webintoapp.com/ads/";
    gVar.f259i = "true";
    gVar.f260j = "true";
    gVar.f261k = "banner ";
    gVar.f262l = "";
    gVar.f263m = "true";
    gVar.f264n = "";
    gVar.f265o = "";
    gVar.f266p = "";
    gVar.f267q = "live";
    gVar.f268r = "";
    gVar.f269s = Boolean.FALSE;
    gVar.f = 3000;
    gVar.f257g = 60000;
    gVar.b = this;
    SharedPreferences defaultSharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);
    if (defaultSharedPreferences.getString("installkey", null) == null) {
        Random random = new Random();
        StringBuilder sb = new StringBuilder(32);
        for (int i2 = 0; i2 < 32; i2++) {
            sb.append("0123456789qwertyuiopasdfghjklzxcvbnm".charAt(random.nextInt(36)));
        }
        defaultSharedPreferences.edit().putString("installkey", sb.toString()).apply();
    }
    String string = defaultSharedPreferences.getString("installkey", null);
    gVar.f265o = string;
    gVar.f264n = "932449";
    gVar.f258h = "https://ads.webintoapp.com/ads/";
    D.g.I(gVar.b).a(new h(0, "https://ads.webintoapp.com/ads/ads-start.json?ai=932449&ik=" + string, new V.e(gVar), new d(9)));
    this.f1544F = gVar;
    WebView webView = (WebView) findViewById(R.id.activity_splash_webview);
    webView.setWebChromeClient(new WebChromeClient());
```

Screenshot 2025-10-07 at 21.53.19.png

Noncompliant Code Example

```
import java.util.Random;

public class generateRandom {
    public static void main(String args[]) {
        Random rand = new Random();
        int rand_int = rand.nextInt(1000);
    }
}
```

OR

```
import java.lang.Math;

public class generateRandom {
    public static void main(String[] args) {
        double randomValue = Math.random();
    }
}
```

Compliant Solution

```
import java.security.SecureRandom;

public class generateRandom {
    public static void main(String args[]) {
```

```
            SecureRandom rand = new SecureRandom();
            int rand_int = rand.nextInt(1000);
        }
    }
```

Business Implication

Weak PRNG (Pseudorandom Number Generator) vulnerabilities within Android applications can have far-reaching business consequences. Such vulnerabilities can lead to data breaches and unauthorized access, undermining user trust and potentially violating regulatory requirements. Financial losses may result from legal actions, compensations, and operational disruptions caused by security incidents. Moreover, a compromised reputation can lead to decreased user adoption, loss of competitive advantage, and even app removal from stores

StrandHogg Vulnerability

StrandHogg is a vulnerability in the task management system on Android that allows an attacker to hijack tasks. This can be exploited such that when the end user starts a legitimate app, instead of actually getting that app, the user is presented with a malicious look-a-like instead.

Risk Rating

Medium

Scan Type

Static

CVSS

Version 3.0 Base Score **6.5**	Attack vector: **LOCAL**	Attack complexity: **HIGH**
	Privileges required: **NONE**	User Interaction: **REQUIRED**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **LOW**	Availability Impact: **HIGH**

Regulatory

CWE	CWE-200	https://cwe.mitre.org/data/definitions/200.html
MSTG	MSTG-PLATFORM-9	The app protects itself against screen overlay attacks. (Android only)
OWASP MASVS (v2)	MASVS-PLATFORM-3	The app uses the user interface securely.
PCI-DSS (v4.0)	6.1	**Develop and maintain secure systems and applications** Processes and mechanisms for developing and maintaining secure systems and software are defined and understood
	6.3	**Develop and maintain secure systems and applications** Security vulnerabilities are identified and addressed
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

One or more than one public facing activities of the application is vulnerable to StandHogg vulnerability.

In StrandHogg and regular task hijacking, malicious applications typically deploy one of the following techniques or a selection in tandem:

- **Task Affinity Manipulation**: The malicious application leverages two activities, M1 and M2, wherein M2.taskAffinity = com.victim.app and M2.allowTaskReparenting = true. In the eventuality that the malicious app is opened on M2, M2 would be relocated to the front and the user will interact with the malicious application once the victim application has initiated.
- **Single Task Mode**: In the eventuality that the victim application sets launchMode to singleTask, malicious applications can leverage M2.taskAffinity = com.victim.app to hijack the victim's application task stack.
- **Task Reparenting**: In the eventuality that the victim application sets taskReparenting to true, malicious applications can move the victim's application task to the malicious application's stack.

However, in the case of StrandHogg 2.0, all exported activities without a launchMode of singleTask or singleInstance are affected on vulnerable Android versions.

Vulnerability

The launchMode for public activity net.t247tech.healthmonitor.MainActivity remains unset and hence defaults to standard, which mitigates task hijacking via StrandHogg 1.0 and other deprecated techniques documented since 2015 while persisting app vulnerability to StrandHogg 2.0. This vulnerability affects Android versions 3-9, but was only patched by Google on Android 8 and 9. This increases attack susceptibility for all users running Android 5-7.x, as well as users running unpatched Android 8-9.x devices.

Remediation

To mitigate this issue, it is recommended to implement as many of the following countermeasures as deemed feasible by the development team:

1. The task affinity of exported application activities should be set to an empty string in the Android manifest. This will force the activities to use a randomly-generated task affinity rather than the package name and hence prevent task hijacking, as malicious apps will not have a predictable task affinity to target.

2. The launchMode should then be altered to singleInstance (instead of singleTask, for instance). This will ensure continuous mitigation in StrandHogg 2.0 whilst improving security strength against outdated task hijacking techniques.

3. A custom onBackPressed() function could be implemented to override the default behavior.

4. The FLAG_ACTIVITY_NEW_TASK should not be set in activity launch intents. If deemed required, one should use the aforementioned in combination with the FLAG_ACTIVITY_CLEAR_TASK flag.

```
<activity
    android:name="net.t247tech.healthmonitor.MainActivity"
    android:exported="true"
    android:configChanges="screenSize|orientation"
    android:hardwareAccelerated="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<meta-data
    android:name="android.webkit.WebView.EnableSafeBrowsing"
    android:value="true"/>
<provider
    android:name="androidx.startup.InitializationProvider"
```

Screenshot 2025-10-07 at 21.51.29.png

Noncompliant Code Example

```
<activity android:theme="@style/AppTheme.NoActionBar"
android:name="com.demo.MainActivity" android:exported="true"
android:screenOrientation="portrait"
android:windowSoftInputMode="adjustNothing">
```

Compliant Solution

```
<activity android:theme="@style/AppTheme.NoActionBar"
android:name="com.demo.MainActivity" android:exported="true"
android:screenOrientation="portrait" android:windowSoftInputMode="adjustNothing"
android:launchMode="singleInstance" android:taskAffinity="">
```

Business Implication

A malicious app could leverage this weakness to manipulate the way in which users interact with the app. Specifically, this could be instigated by relocating a malicious attacker-controlled activity in the screen flow of the user, which may prove useful towards performing phishing, DoS or user-credential capture. This issue has been verified as a known exploit leveraged by banking malware Trojans in the past.

Related Vulnerabilities

- Towards Discovering and Understanding Task Hijacking in Android

MediaProjection: Android Service Allows Recording of Audio, Screen Activity

Starting with Android 5.0, Google introduced the android.media.projection API which allows any third-party App to perform screen capture and screen sharing (fixed in Android 8).

Such an App can capture everything on the device's screen, including sensitive activity from all other Apps such as password keystrokes, credit card data, etc. The capturing ability remains on even if the user terminates/closes the App, but not after a reboot.

Risk Rating

Medium

Scan Type

Static

CVSS

Version 3.0 Base Score **6.8**	Attack vector: **LOCAL**	Attack complexity: **LOW**
	Privileges required: **NONE**	User Interaction: **NONE**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **LOW**	Availability Impact: **NONE**

Regulatory

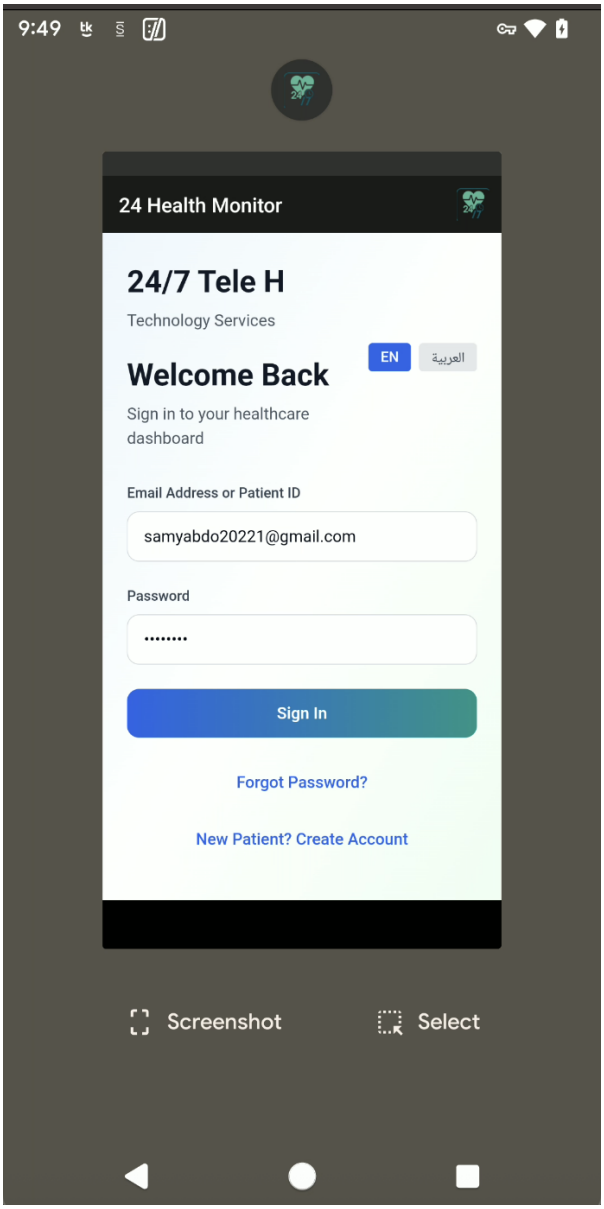
OWASP Mobile Top 10 (2024)	M8	Security Misconfiguration
CWE	CWE-200	https://cwe.mitre.org/data/definitions/200.html
MSTG	MSTG-STORAGE-9	The app removes sensitive data from views when moved to the background.
OWASP MASVS (v2)	MASVS-PLATFORM-3	The app uses the user interface securely.
PCI-DSS (v4.0)	3.1	**Protect Account Data** Processes and mechanisms for protecting stored account data are defined and understood
	3.2	**Protect Account Data** Storage of account data is kept to a minimum
	3.3	**Protect Account Data** Sensitive authentication data (SAD) is not stored after authorization

GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Protect all sensitive windows within the App by enabling the FLAG_SECURE flag. This flag will prevent Apps from being able to record the protected windows. Also, the flag will prevent users from taking screenshots of these windows (by pressing the VOLUME_DOWN and POWER buttons). As such screenshots are stored on the SDCard by default, they are accessible to all Apps and sensitive data may be exposed.

> The App does not protect sensitive screens from being displayed in screencasts initiated by third-party Apps



Screenshot 2025-10-07 at 21.49.30.png

Compliant Solution

Below is an example of how to use FLAG_SECURE inside your activity

```
public class SecureActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Set the Secure flag for this Window
        getWindow().setFlags(LayoutParams.FLAG_SECURE, LayoutParams.FLAG_SECURE);
    }
}
```

Related Vulnerabilities

- Android 5.0 Docs
- FLAG_SECURE docs

Disabled SSL CA Validation and Certificate Pinning

Certificate Pinning is the process of associating a host with their expected X509 certificate or public key. Once a certificate or public key is known or seen for a host, the certificate or public key is associated or 'pinned' to the host. If more than one certificate or public key is acceptable. In this case, the advertised identity must match one of the elements in the pinset.

Risk Rating

Medium

Scan Type

Static

CVSS

Version 3.0 Base Score **5.9**	Attack vector: **NETWORK**	Attack complexity: **HIGH**
	Privileges required: **NONE**	User Interaction: **REQUIRED**
	Scope: **UNCHANGED**	Confidentiality Impact: **HIGH**
	Integrity Impact: **LOW**	Availability Impact: **NONE**

Regulatory

OWASP Mobile Top 10 (2024)	M5	Insecure Communication
CWE	CWE-295	https://cwe.mitre.org/data/definitions/295.html
MSTG	MSTG-NETWORK-3	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.
OWASP MASVS (v2)	MASVS-NETWORK-2	The app performs identity pinning for all remote endpoints under the developer's control.
PCI-DSS (v4.0)	4.1	**Protect cardholder data with strong cryptography during transmission over open, public networks** Processes and mechanisms for protecting cardholder data with strong cryptography during transmission over open, public networks are defined and documented.
	4.2	**Protect cardholder data with strong cryptography during transmission over open, public networks** PAN is protected with strong cryptography during transmission
GDPR	Art-25-GDPR	Data protection by design and by default

Art-32-GDPR	Security of processing

Risk Assessment

A host or service's certificate or public key can be added to an application at development time, or it can be added upon first encountering the certificate or public key. The former - adding at development time - is preferred since preloading the certificate or public key out of band usually means the attacker cannot taint the pin.

```
SSL Pinning is not implemented in the Application.
```

Compliant Solution

Certificate Pinning can be done with these two options:

You can 1. pin the certificate or 2. pin the public key

If you choose public keys, you have two additional choices: - pin the `subjectPublicKeyInfo` or - pin one of the concrete types such as `RSAPublicKey` or `DSAPublicKey`.

The three choices are explained below in more detail. I would encourage you to pin the subjectPublicKeyInfo because it has the public parameters (such as {e,n} for an RSA public key) and contextual information such as an algorithm and OID. The context will help you keep your bearings at times, and the figure to the right shows the additional information available.

Certificate

The certificate is easiest to pin. You can fetch the certificate out of band for the website, have the IT folks email your company certificate to you, use openssl s_client to retrieve the certificate etc. At runtime, you retrieve the website or server's certificate in the callback. Within the callback, you compare the retrieved certificate with the certificate embedded within the program. If the comparison fails, then fail the method or function. There is a downside to pinning a certificate. If the site rotates its certificate on a regular basis, then your application would need to be updated regularly. For example, Google rotates its certificates, so you will need to update your application about once a month (if it depended on Google services). Even though Google rotates its certificates, the underlying public keys (within the certificate) remain static.

Public Key

Public key pinning is more flexible but a little trickier due to the extra steps necessary to extract the public key from a certificate. As with a certificate, the program checks the extracted public key with its embedded copy of the public key. There are two downsides to public key pinning. First, it's harder to work with keys (versus certificates) since you must extract the key from the certificate. Extraction is a minor inconvenience in Java and .Net, buts it's uncomfortable in Cocoa/CocoaTouch and OpenSSL. Second, the key is static and may violate key rotation policies.

Hashing

While the three choices above used DER encoding, its also acceptable to use a hash of the information. In fact, the original sample programs were written using digested certificates and public keys. The samples were changed to allow a programmer to inspect the objects with tools like dumpasn1 and other ASN.1 decoders.

Hashing also provides three additional benefits. First, hashing allows you to anonymize a certificate or public key. This might be important if you application is concerned about leaking information during decompilation and re-engineering. Second, a digested certificate fingerprint is often available as a native API for many libraries, so its convenient to use. Finally, an organization might want to supply a reserve (or back-up) identity in case the primary identity is compromised. Hashing ensures your adversaries do not see the reserved certificate or public key in advance of its use. In fact, Google's IETF draft websec-key-pinning uses the technique.

Business Implication

In the event that a user (anonymous or verified) is able to execute over-privileged functionality, the business may experience:

- Reputational Damage
- Fraud
- Information Theft

Related Vulnerabilities

- OWASP Transport Layer Protection Cheat Sheet
- IETF RFC 1421 (PEM Encoding)
- IETF RFC 4648 (Base16, Base32, and Base64 Encodings)
- IETF RFC 5280 (Internet X.509, PKIX)
- IETF RFC 3279 (PKI, X509 Algorithms and CRL Profiles)
- IETF RFC 4055 (PKI, X509 Additional Algorithms and CRL Profiles)
- IETF RFC 2246 (TLS 1.0)
- IETF RFC 4346 (TLS 1.1)
- IETF RFC 5246 (TLS 1.2)
- RSA Laboratories PKCS#1, RSA Encryption Standard
- RSA Laboratories PKCS#6, Extended-Certificate Syntax Standard

Android Developer options status detection

Developer Options in Android are a set of advanced settings intended primarily for app developers. These settings provide tools and functionalities that help developers test, debug, and optimize their applications. Common features include enabling USB debugging, showing CPU usage, capturing bug reports, and more.

Security Implications: - USB Debugging: Exposes device data to connected computers. - OEM Unlocking: Can lead to rooting, bypassing security controls. - Mock Locations: Can be used to spoof location data. - General System Exposure: Enabling Developer Options increases the attack surface.

Risk Rating

Low

Scan Type

Dynamic

CVSS

Version 3.0 Base Score **3.4**	Attack vector: **PHYSICAL**	Attack complexity: **LOW**
	Privileges required: **HIGH**	User Interaction: **REQUIRED**
	Scope: **CHANGED**	Confidentiality Impact: **NONE**
	Integrity Impact: **LOW**	Availability Impact: **LOW**

Regulatory

OWASP Mobile Top 10 (2024)	M6	Inadequate Privacy Controls
CWE	CWE-284	https://cwe.mitre.org/data/definitions/284.html
	CWE-541	https://cwe.mitre.org/data/definitions/541.html
	CWE-532	https://cwe.mitre.org/data/definitions/532.html
MSTG	MSTG-RESILIENCE-2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.
OWASP MASVS (v2)	MASVS-RESILIANCE-1	The app validates the integrity of the platform.
ASVS	V1.4.4	Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths.

	V1.8.2	Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.
PCI-DSS (v4.0)	7.1	**Restrict access to cardholder data by business need to know** Processes and mechanisms for restricting access to system components and cardholder data by business need to know are defined and understood
	7.2	**Restrict access to cardholder data by business need to know** Access to system components and data is appropriately defined and assigned
HIPAA	164.308(a)(4)	**Administrative Safeguards: Information Access Management** ■ **Isolating Health Care Clearinghouse Functions (Required)** If a health care clearinghouse is part of a larger organization, the clearinghouse must implement policies and procedures that protect the electronic protected health information of the clearinghouse from unauthorized access by the larger organization. ■ **Access Authorization (Addressable)** Implement policies and procedures for granting access to electronic protected health information, for example, through access to a workstation, transaction, program, process, or other mechanism. ■ **Access Establishment and Modification (Addressable)** Implement policies and procedures that, based upon the entity's access authorization policies, establish, document, review, and modify a user's right of access to a workstation, transaction, program, or process.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Implementing Developer options protection in Android applications is crucial to ensure security and prevent unauthorized access to sensitive features and data. Implementing Developer option protection in your Android application enhances its security and integrity. By detecting and responding to the enabling of developer options, you can prevent unauthorized access and protect sensitive data.

```
The application does not protect itself from launching if developer option is turned on.
```

Compliant Solution

To detect whether Developer Options are enabled on an Android device, you can use the Settings.Global API in Android. Here’s how you can do it in Java or Kotlin.

Java:

public class MainActivity extends AppCompatActivity { @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);

```
        if (isDeveloperOptionsEnabled(this)) {
            // show a warning and exit the app
        } else {
            // continue the app...
        }
    }

    public static boolean isDeveloperOptionsEnabled(Context context) {
        int devOptions = Settings.Global.getInt(context.getContentResolver(),
    Settings.Global.DEVELOPMENT_SETTINGS_ENABLED, 0);
        return devOptions == 1;
    }
```

}

Kotlin:

class MainActivity : AppCompatActivity() { override fun onCreate(savedInstanceState: Bundle?) { super.onCreate(savedInstanceState) setContentView(R.layout.activity_main)

```
        if (isDeveloperOptionsEnabled(this)) {
            // show a warning and exit the app
        } else {
            // continue the app...
        }
    }

    private fun isDeveloperOptionsEnabled(context: Context): Boolean {
        val devOptions = Settings.Global.getInt(context.contentResolver,
    Settings.Global.DEVELOPMENT_SETTINGS_ENABLED, 0)
        return devOptions == 1
    }
```

}

Business Implication

Running an application in Developer options enabled environment can expose security risks, impact performance, and affect user experience.

Android Developer Bridge(ADB) status detection

ADB (Android Debug Bridge) is a versatile command-line tool that provides a bridge between your computer and an Android device. It's essential for developers to test, debug, and optimize their apps. However, while ADB offers numerous functionalities, it also introduces potential security risks if not used carefully.

Security Implications: - Data exposure: Unauthorized access can lead to data theft. - Malware installation: Malicious actors can install malware. - Device compromise: Misuse can grant root access, compromising the device. - USB debugging vulnerabilities: Connected to an untrusted computer can expose the device. - Shell command risks: Incorrect or malicious use can lead to security vulnerabilities.

Risk Rating

Low

Scan Type

Dynamic

CVSS

Version 3.0 Base Score **3.4**	Attack vector: **PHYSICAL**	Attack complexity: **LOW**
	Privileges required: **HIGH**	User Interaction: **REQUIRED**
	Scope: **CHANGED**	Confidentiality Impact: **NONE**
	Integrity Impact: **LOW**	Availability Impact: **LOW**

Regulatory

OWASP Mobile Top 10 (2024)	M6	Inadequate Privacy Controls
CWE	CWE-284	https://cwe.mitre.org/data/definitions/284.html
	CWE-541	https://cwe.mitre.org/data/definitions/541.html
	CWE-532	https://cwe.mitre.org/data/definitions/532.html
MSTG	MSTG-RESILIENCE-2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.
OWASP MASVS (v2)	MASVS-RESILIANCE-1	The app validates the integrity of the platform.
ASVS	V1.4.4	Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths.

	V1.8.2	Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.
PCI-DSS (v4.0)	7.1	**Restrict access to cardholder data by business need to know** Processes and mechanisms for restricting access to system components and cardholder data by business need to know are defined and understood
	7.2	**Restrict access to cardholder data by business need to know** Access to system components and data is appropriately defined and assigned
HIPAA	164.308(a)(4)	**Administrative Safeguards: Information Access Management** ■ **Isolating Health Care Clearinghouse Functions (Required)** If a health care clearinghouse is part of a larger organization, the clearinghouse must implement policies and procedures that protect the electronic protected health information of the clearinghouse from unauthorized access by the larger organization. ■ **Access Authorization (Addressable)** Implement policies and procedures for granting access to electronic protected health information, for example, through access to a workstation, transaction, program, process, or other mechanism. ■ **Access Establishment and Modification (Addressable)** Implement policies and procedures that, based upon the entity's access authorization policies, establish, document, review, and modify a user's right of access to a workstation, transaction, program, or process.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Implementing ADB detection in Android applications is crucial to ensure security and prevent unauthorized access to sensitive features and data. ADB detection enhances app security by guarding against unintended exposure, unauthorized access, and tampering. By detecting and responding to the enabling of ADB on the device, you can prevent unauthorized access and protect sensitive data.

```
The application does not protect itself from launching if ADB is turned on.
```

Compliant Solution

To detect whether ADB (Android Debug Bridge) debugging is enabled on an Android device, you can use the Settings.Global API in Android. Here’s how you can do it in Java or Kotlin.

Java:

public class MainActivity extends AppCompatActivity { @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);

```
        if (isAdbEnabled(this)) {
            // show a warning and exit the app
        } else {
            // continue the app...
        }
    }

    public static boolean isAdbEnabled(Context context) {
        int adb = Settings.Global.getInt(context.getContentResolver(), Settings.Global.ADB_ENABLED,
    0);
        return adb == 1;
    }
```

}

Kotlin:

class MainActivity : AppCompatActivity() { override fun onCreate(savedInstanceState: Bundle?) { super.onCreate(savedInstanceState) setContentView(R.layout.activity_main)

```
        if (isAdbEnabled(this)) {
            // show a warning and exit the app
        } else {
            // continue the app...
        }
    }

    private fun isAdbEnabled(context: Context): Boolean {
        val adb = Settings.Global.getInt(context.contentResolver, Settings.Global.ADB_ENABLED, 0)
        return adb == 1
    }
```

}

Business Implication

Running an application in ADB enabled environment can expose security risks, impact performance, and affect user experience.

Bytecode Obfuscation

Generally, all mobile code is susceptible to reverse engineering. Some apps are more susceptible than others. Code written in languages / frameworks that allow for dynamic introspection at runtime (Java, .NET, Objective C, Swift) are particularly at risk for reverse engineering. Detecting susceptibility to reverse engineering is fairly straight forward. First, decrypt the app store version of the app (if binary encryption is applied). Code will be susceptible if it is fairly easy to understand the app's controlflow path, string table, and any pseudocode/source-code generated by these tools.

Bytecode obfuscation consists of multiple complementary techniques that can help create a layered defense against reverse engineering and tampering. Some typical examples of obfuscation techniques include:

- **Renaming** to alter the name of methods and variables to make the decompiled source much harder for a human to understand.
- **Control Flow Obfuscation**creates conditional, branching, and iterative constructs that produce valid executable logic, but yield non-deterministic semantic results when decompiled.
- **String Encryption** hides strings in the executable and only restores their original value when needed
- **Instruction Pattern Transformation** converts common instructions to other, less obvious constructs potential confusing decompliers.
- **Dummy Code Insertion** inserts code that does not affect the program's logic, but breaks decompilers or makes reverse-engineered code harder to analyze.
- **Unused Code and Metadata Removal** prunes out debug, non-essential metadata and used code from applications to reduce the information available to an attacker.

Risk Rating

Low

Scan Type

Static

CVSS

Version 3.0 Base Score **2.3**	Attack vector: **LOCAL**	Attack complexity: **LOW**
	Privileges required: **HIGH**	User Interaction: **NONE**
	Scope: **UNCHANGED**	Confidentiality Impact: **LOW**
	Integrity Impact: **NONE**	Availability Impact: **NONE**

Regulatory

OWASP Mobile Top 10 (2024)	M7	Insufficient Binary Protections
CWE	CWE-693	https://cwe.mitre.org/data/definitions/693.html
MSTG	MSTG-RESILIENCE-9	Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.

OWASP MASVS (v2)	MASVS-RESILIANCE-3	The app implements anti-static analysis mechanisms.
PCI-DSS (v4.0)	3.2	**Protect Account Data** Storage of account data is kept to a minimum
	3.3	**Protect Account Data** Sensitive authentication data (SAD) is not stored after authorization
	3.4	**Protect Account Data** Access to displays of full PAN and ability to copy cardholder data are restricted
	3.5	**Protect Account Data** Primary account number (PAN) is secured wherever it is stored
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Java source code is typically compiled into Java bytecode – the instruction set of the Java virtual machine. The compiled Java bytecode can be easily reversed engineered back into source code by freely available decompilers. Bytecode Obfuscation is the process of modifying Java bytecode (executable or library) so that it is much harder to read and understand for a hacker but remains fully functional.

```
Application is vulnerable to reverse engineering without any obfuscation
```

AndroidManifest.xml | b | MainActivity

```
        if (defaultSharedPreferences.getString("installkey", null) == null) {
            Random random = new Random();
            StringBuilder sb = new StringBuilder(32);
            for (int i2 = 0; i2 < 32; i2++) {
                sb.append("0123456789qwertyuiopasdfghjklzxcvbnm".charAt(random.nextInt(36)));
            }
            defaultSharedPreferences.edit().putString("installkey", sb.toString()).apply();
        }
        String string = defaultSharedPreferences.getString("installkey", null);
        gVar.f265o = string;
        gVar.f264n = "932449";
        gVar.f258h = "https://ads.webintoapp.com/ads/";
        D.g.I(gVar.b).a(new h(0, "https://ads.webintoapp.com/ads/ads-start.json?ai=932449&ik=" + string, new V.e(gVar), new d(9)));
        this.f1544F = gVar;
        WebView webView = (WebView) findViewById(R.id.activity_splash_webview);
        webView.setWebChromeClient(new WebChromeClient());
        webView.setWebViewClient(new WebViewClient());
        webView.getSettings().setJavaScriptEnabled(true);
        webView.loadUrl("file:///android_asset/htmlapp/helpers/loading.html");
        Toolbar toolbar = (Toolbar) findViewById(R.id.my_toolbar);
        LayoutInflaterFactory2C0013C layoutInflaterFactory2C0013C = (LayoutInflaterFactory2C0013C) i();
        if (layoutInflaterFactory2C0013C.f851j instanceof Activity) {
            layoutInflaterFactory2C0013C.A();
            D.g gVar2 = layoutInflaterFactory2C0013C.f856o;
            if (gVar2 instanceof O) {
                throw new IllegalStateException("This Activity already has an action bar supplied by the window decor. Do not request W
            }
            layoutInflaterFactory2C0013C.f857p = null;
            if (gVar2 != null) {
                gVar2.L();
            }
            layoutInflaterFactory2C0013C.f856o = null;
            if (toolbar != null) {
                Object obj = layoutInflaterFactory2C0013C.f851j;
                J j2 = new J(toolbar, obj instanceof Activity ? ((Activity) obj).getTitle() : layoutInflaterFactory2C0013C.f858q, layou
                layoutInflaterFactory2C0013C.f856o = j2;
                layoutInflaterFactory2C0013C.f854m.b = j2.f880q;
                toolbar.setBackInvokedCallbackEnabled(true);
            } else {
                layoutInflaterFactory2C0013C.f854m.b = null;
            }
            layoutInflaterFactory2C0013C.b();
        }
        if (Build.VERSION.SDK_INT >= 35) {
            getWindow().getDecorView().setOnApplyWindowInsetsListener(new i0.d());
```

Screenshot 2025-10-07 at 21.56.16.png

Noncompliant Code Example

Sample `proguard-rules.pro` lines which fails obfuscation when used:

```
-dontobfuscate
```

Or if no proguard or obfuscation is used then you can use number of freely available Java decompilers that can recreate source code from Java bytecode (executables or libraries).

Popular decompilers include:

- Bytecode Viewer - A Java 8 Jar & Android APK Reverse Engineering Suite (Decompiler, Editor, Debugger & More)
- CFR - Another Java decompiler
- JDGui - Yet another fast Java decompiler
- Fernflower - An analytical decompiler for Java
- JadX - tool to decompile APK and DEX files

Compliant Solution

To enable shrinking, obfuscation, and optimization, using proguard include the following in your project-level `build.gradle` file.

```
android {
    buildTypes {
```

```
        release {
            // Enables code shrinking, obfuscation, and optimization for only
            // your project's release build type.
            minifyEnabled true

            // Enables resource shrinking, which is performed by the
            // Android Gradle plugin.
            shrinkResources true

            // Includes the default ProGuard rules files that are packaged with
            // the Android Gradle plugin. To learn more, go to the section about
            // R8 configuration files.
            proguardFiles getDefaultProguardFile(
                    'proguard-android-optimize.txt'),
                    'proguard-rules.pro'
        }
    }
    ...
}
```

Sample `proguard-rules.pro` which you can use to obfuscate code:

```
// Basic proguard rules
-optimizations !code/simplification/arithmetic
-keepattributes <em>Annotation</em>
-keepattributes InnerClasses
-keepattributes EnclosingMethod
-keep class *<em>.R$</em>

-dontskipnonpubliclibraryclasses
-forceprocessing
-optimizationpasses 5
-overloadaggressively

// Removing logging code
-assumenosideeffects class android.util.Log {
public static *** d();
public static *** v();
public static *** i();
public static *** w();
public static *** e();
}

// Crashlytics code as given below which one can exclude

-keep class com.crashlytics.** { *; }
-keep class com.crashlytics.android.**
-keepattributes SourceFile,LineNumberTable
```

Related Vulnerabilities

- M9: Reverse Engineering
- Bytecode Obfuscation
- Android Developer Resource to shrink and obfuscate your code in proguard

Enabled Android Application Backup

The mobile application uses external backup functionality (default Android backup mechanism) that may store inside sensitive data from the application.

In certain conditions, this may lead to information disclosure (e.g. when a backup server or the Gmail account is compromised).

Risk Rating

Low

Scan Type

Static

CVSS

Version 3.0 Base Score **3.3**	Attack vector: **LOCAL**	Attack complexity: **LOW**
	Privileges required: **LOW**	User Interaction: **NONE**
	Scope: **UNCHANGED**	Confidentiality Impact: **LOW**
	Integrity Impact: **NONE**	Availability Impact: **NONE**

Regulatory

OWASP Mobile Top 10 (2024)	M8	Security Misconfiguration
CWE	CWE-16	https://cwe.mitre.org/data/definitions/16.html
MSTG	MSTG-STORAGE-8	No sensitive data is included in backups generated by the mobile operating system.
OWASP MASVS (v2)	MASVS-STORAGE-2	The app prevents leakage of sensitive data.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Application backup might contain sensitive information private data of the app into their PC

`android:allowBackup=true` is found in AndroidManifest.xml

```
<uses-permission android:name="net.t247tech.healthmonitor.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION" />
<application
    android:theme="@style/AppTheme"
    android:label="@string/app_name"
    android:icon="@mipmap/ic_launcher"
    android:screenOrientation="unspecified"
    android:allowBackup="true"
    android:hardwareAccelerated="true"
    android:supportsRtl="true"
    android:extractNativeLibs="false"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:appComponentFactory="androidx.core.app.CoreComponentFactory">
    <provider
        android:name="androidx.core.content.FileProvider"
        android:exported="false"
        android:authorities="net.t247tech.healthmonitor.provider"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/provider_paths"/>
    </provider>
    <activity
        android:name="net.t247tech.healthmonitor.MainActivity"
        android:exported="true"
        android:configChanges="screenSize|orientation"
        android:hardwareAccelerated="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
```

Screenshot 2025-10-07 at 21.55.23.png

Noncompliant Code Example

Example of an insecure code in AndroidManifest file:

```
android:allowBackup="true"
```

Compliant Solution

Below is an example of how to prevent the application from getting backed-up via adb in your AndroidManifest file:

```
android:allowBackup="false"
```

Related Vulnerabilities

- Android Manifest Docs
- Hacking Android Apps using Backup

Keylogger Protection

Keylogger protection is crucial for safeguarding sensitive information from malicious applications that may be installed on a device. A keylogger is a type of malware designed to capture and record keystrokes or other input data from the user. If such an application is present on the device, it can potentially intercept and steal personal or confidential information by tracking what you type.

Risk Rating

Low

Scan Type

Static

CVSS

Version 3.0 Base Score **3.9**	Attack vector: **LOCAL**	Attack complexity: **HIGH**
	Privileges required: **HIGH**	User Interaction: **NONE**
	Scope: **CHANGED**	Confidentiality Impact: **LOW**
	Integrity Impact: **LOW**	Availability Impact: **NONE**

Regulatory

OWASP Mobile Top 10 (2024)	M6	Inadequate Privacy Controls
CWE	CWE-693	https://cwe.mitre.org/data/definitions/693.html
OWASP MASVS (v2)	MASVS-RESILIANCE-2	The app implements anti-tampering mechanisms.
ASVS	V1.8.1	Verify that all sensitive data is identified and classified into protection levels.
	V1.8.2	Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.
PCI-DSS (v4.0)	7.1	**Restrict access to cardholder data by business need to know** Processes and mechanisms for restricting access to system components and cardholder data by business need to know are defined and understood

	7.2	**Restrict access to cardholder data by business need to know** Access to system components and data is appropriately defined and assigned
	6.2	**Develop and maintain secure systems and applications** Bespoke and custom software are developed securely
	6.3	**Develop and maintain secure systems and applications** Security vulnerabilities are identified and addressed
HIPAA	164.308(a)(4)	**Administrative Safeguards: Information Access Management** ■ **Isolating Health Care Clearinghouse Functions (Required)** If a health care clearinghouse is part of a larger organization, the clearinghouse must implement policies and procedures that protect the electronic protected health information of the clearinghouse from unauthorized access by the larger organization. ■ **Access Authorization (Addressable)** Implement policies and procedures for granting access to electronic protected health information, for example, through access to a workstation, transaction, program, process, or other mechanism. ■ **Access Establishment and Modification (Addressable)** Implement policies and procedures that, based upon the entity's access authorization policies, establish, document, review, and modify a user's right of access to a workstation, transaction, program, or process.
	164.312(c)(1)	**Technical Safeguards: Integrity** ■ **Mechanism to Authenticate Electronic Protected (Addressable)** Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner.
GDPR	Art-25-GDPR	Data protection by design and by default
	Art-32-GDPR	Security of processing

Risk Assessment

Keyloggers are malicious software designed to capture and record keystrokes or input data from users, posing a significant threat to personal and sensitive information. They operate by monitoring keyboard activity, potentially stealing passwords, credit card details, and other confidential data. Keylogger protection involves implementing secure input methods, such as custom keyboards and virtual keyboards, to prevent unauthorized access to typed information. Effective protection ensures that even if a keylogger is present, it cannot intercept or compromise the data being entered.

No Keylogger Protection Found

```
The application does not implement a virtual keyboard service
and thus is vulnerable to keylogger attacks
```

Compliant Solution

Application should utilize a virtual or custom keyboard. By doing so, the app can prevent keyloggers from capturing sensitive input. Below is an example of how to implement a virtual keyboard.

Create a new service

```
class MyInputMethodService : InputMethodService() {
    override fun onCreateInputView(): View {
        val keyboardView = layoutInflater.inflate(R.layout.keyboard_view, null) as KeyboardView
        return keyboardView
    }
}
```

Create an Input Method XML Configuration

```
<?xml version="1.0" encoding="utf-8"?>
<input-method xmlns:android="http://schemas.android.com/apk/res/android"
    android:settingsActivity=".SettingsActivity"
    android:service="com.example.MyInputMethodService">
    <subtype
        android:label="@string/subtype_label"
        android:locale="en_US"
        android:mode="keyboard"
        android:imeSubtypeLocale="en_US"/>
</input-method>
```

Create the keyboard layout XML

```
<?xml version="1.0" encoding="utf-8"?>
<Keyboard xmlns:android="http://schemas.android.com/apk/res/android">
    <Row>
        <Key android:codes="113" android:keyLabel="Q" />
        <Key android:codes="119" android:keyLabel="W" />
        <!-- Add more keys as needed -->
    </Row>
    <!-- Add more rows as needed -->
</Keyboard>
```

Create the keyboard view layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <com.example.KeyboardView
        android:id="@+id/keyboard_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Implement the keyboard logic and events

```
class MyInputMethodService : InputMethodService(), KeyboardView.OnKeyboardActionListener {
    private lateinit var keyboardView: KeyboardView
    private lateinit var keyboard: Keyboard
    override fun onCreateInputView(): View {
        keyboardView = layoutInflater.inflate(R.layout.keyboard_view, null) as KeyboardView
        keyboard = Keyboard(this, R.xml.keyboard_layout)
        keyboardView.keyboard = keyboard
        keyboardView.setOnKeyboardActionListener(this)
        return keyboardView
    }
    override fun onKey(primaryCode: Int, keyCodes: IntArray) {
        val inputConnection = currentInputConnection
        inputConnection?.commitText(primaryCode.toChar().toString(), 1)
    }
}
```

Add the service declaration in the manifest with correct permissions

```
<service
    android:name=".MyInputMethodService"
    android:permission="android.permission.BIND_INPUT_METHOD">
    <intent-filter>
        <action android:name="android.service.inputmethod.InputMethodService" />
    </intent-filter>
    <meta-data
        android:name="android.view.im"
        android:resource="@xml/input_method_config"/>
</service>
```

Business Implication

Without keylogger protection, there is a significant risk of sensitive information being stolen, which could lead to serious consequences such as identity theft and financial loss. This vulnerability exposes both the application and its users to potential security breaches.

References

1. Mobile Top 10 OWASP Categories | 2016
2. Web Top 10 OWASP Categories | 2013