



**UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO**

**Università degli studi di Bari
“Aldo Moro”**

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Magistrale in Sicurezza Informatica

**Report sulla sicurezza dell’applicazione Android
AigoSmart**

Docente:

Paolo

Buono

Studente:

Ludovico Nigro

ANNO ACCADEMICO 2023/2024

1. TEORIA.....	5
1.1 ANDROID.....	5
1.1.1 SISTEMA ANDROID	5
1.1.2 ARCHITETTURA ANDROID.....	5
1.1.3 APP ANDROID	6
1.2 PENETRATION TEST	8
1.2.1 FASI DEL MOBILE PEN TEST	9
1.2.2 ANALISI DELLE VULNERABILITÀ	9
1.2.3 ANALISI STATICHE E ANALISI DINAMICA.....	10
1.3 SICUREZZA ED ANDROID	10
1.4 VULNERABILITÀ OWASP (OPEN WEB APPLICATION SECURITY)	12
1.4.1.1 OWASP	12
1.4.1.1.1 OWASP Top 10.....	13
1.4.1.1.2 OWASP Mobile Top 10.....	14
1.4.1.1.3 OWASP MASVS	15
2. CASO DI STUDIO.....	16
2.1 FUNZIONALITÀ.....	17
2.2 STRUMENTI UTILIZZATI	18
2.2.1 VIRTUALBOX	18
2.2.2 KALI LINUX	18
2.2.3 ANDROID TAMER	18
2.2.4 GENYMOTION	18
2.2.5 ANDROID DEBUG BRIDGE (ADB)	18
2.2.6 DOCKER DESKTOP	19
2.2.7 MOBSF (MOBILE SECURITY FRAMEWORK).....	19
2.2.8 JAVA DECOMPILE (JD-GUI)	19
2.2.9 MARA (MOBILE APPLICATION REVERSE ENGINEERING AND ANALYSIS)	
19	
2.2.10 IMMUNIWEB.....	20
2.2.11 VISUAL STUDIO CODE	20
2.3 GUIDA ALL'USO DEGLI STRUMENTI UTILIZZATI.....	21
2.3.1 MOBSF	21
2.3.2 KALI LINUX & ANDROID TAMER.....	23
2.3.3 GENYMOTION	24
2.3.4 INSTALLAZIONE APP SU EMULATORE	25
2.3.5 ADB.....	26
3 PENETRATION TESTING.....	28
3.1 ANALISI AUTOMATICA.....	29

3.1.1	IMMUNIWEB.....	29
3.1.2	MOBSF.....	30
3.2	MASVS	31
3.2.1	MASVS-V1: Requisiti di Archiviazione Dati e Privacy [STORAGE].....	32
3.2.1.1	MASTG-TEST-0012	32
3.2.1.2	MASTG-TEST-0001	32
3.2.1.3	MASTG-TEST-0009	33
3.2.1.4	MASTG-TEST-0004	34
3.2.1.5	MASTG-TEST-0011	34
3.2.1.6	MASTG-TEST-0006	35
3.2.1.7	MASTG-TEST-0005	35
3.2.1.8	MASTG-TEST-0003	35
3.2.2	MASVS-V2: Requisiti Crittografici [CRYPTO].....	36
3.2.2.1	MASTG-TEST-0016	36
3.2.2.2	MASTG-TEST-0014	36
3.2.2.3	MASTG-TEST-0013	37
3.2.2.4	MASTG-TEST-0015	37
3.2.3	MASVS-V3: Requisiti di Autenticazione e Gestione Sessione [AUTH].....	38
3.2.3.1	MASTG-TEST-0017	38
3.2.3.2	MASTG-TEST-0018	38
3.2.4	MASVS-V4: Requisiti di Comunicazione di Rete [NETWORK]	39
3.2.4.1	MASTG-TEST-0021	39
3.2.4.2	MASTG-TEST-0023	40
3.2.4.3	MASTG-TEST-0020	40
3.2.4.4	MASTG-TEST-0019	40
3.2.4.5	MASTG-TEST-0022	40
3.2.5	MASVS-V5: Requisiti di Interazione con la Piattaforma [PLATFORM].....	41
3.2.5.1	MASTG-TEST-0030	41
3.2.5.2	MASTG-TEST-0007	42
3.2.5.3	MASTG-TEST-0024	42
3.2.5.4	MASTG-TEST-0028	42
3.2.5.5	MASTG-TEST-0032	43
3.2.5.6	MASTG-TEST-0037	43
3.2.5.7	MASTG-TEST-0031	43
3.2.5.8	MASTG-TEST-0033	44
3.2.5.9	MASTG-TEST-0010	44
3.2.5.10	MASTG-TEST-0008	44
3.2.5.11	MASTG-TEST-0035	44
3.2.6	MASVS-V6: Requisiti di Qualità del Codice e Impostazioni di Build [CODE]	45
3.2.6.1	MASTG-TEST-0036	45
3.2.6.2	MASTG-TEST-0042	46
3.2.6.3	MASTG-TEST-0026	46
3.2.6.4	MASTG-TEST-0002	46
3.2.6.5	MASTG-TEST-0025	46
3.2.6.6	MASTG-TEST-0027	46
3.2.6.7	MASTG-TEST-0034	47

3.2.6.8	MASTG-TEST-0044	47
3.2.7	MASVS-V7: Requisiti di Resilienza [RESILIENCE].....	48
3.2.7.1	MASTG-TEST-0045	48
3.2.7.2	MASTG-TEST-0049	48
3.2.7.3	MASTG-TEST-0047	48
3.2.7.4	MASTG-TEST-0038	49
3.2.7.5	MASTG-TEST-0050	49
3.2.7.6	MASTG-TEST-0040	50
3.2.7.7	MASTG-TEST-0041	50
3.2.7.8	MASTG-TEST-0051	50
3.2.7.9	MASTG-TEST-0046	50
3.2.7.10	MASTG-TEST-0039	50
3.2.7.11	MASTG-TEST-0048	51
4	VULNERABILITÀ	52
4.1	SCOPE	52
4.2	EXECUTIVE SUMMARY	52
4.3	VULNERABILITÀ	52
4.3.1	Vulnerabilità #1	52
4.3.2	Vulnerabilità #2	54
4.3.3	Vulnerabilità #3	55
4.3.4	Vulnerabilità #4	56
4.3.5	Vulnerabilità #5	56
4.3.6	Vulnerabilità #6	58
4.3.7	Vulnerabilità #7	59
4.3.8	Vulnerabilità #8	60
4.3.9	Vulnerabilità #9	61
4.3.10	Vulnerabilità #10.....	62
4.3.11	Vulnerabilità #11	63
5	CONCLUSIONE	65

1. TEORIA

1.1 ANDROID

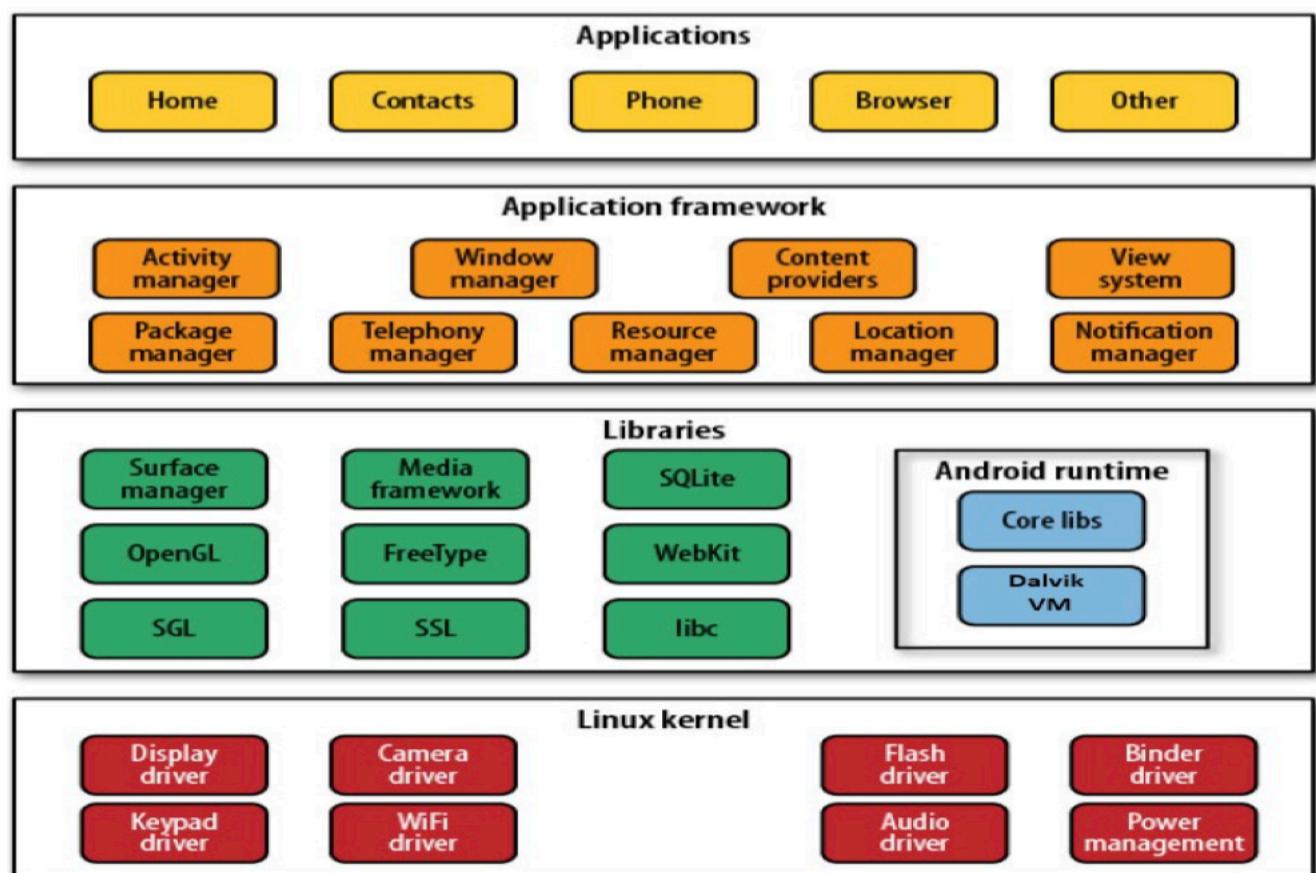
1.1.1 SISTEMA ANDROID

Android è un sistema operativo sviluppato da Google Inc., basato sul kernel Linux. La sua natura open source consente agli sviluppatori di software di accedere al codice sorgente del sistema operativo e modificarlo per creare nuove applicazioni o migliorare quelle esistenti. Originariamente concepito per smartphone e tablet, Android si è esteso a una vasta gamma di dispositivi grazie ai continui progressi tecnologici.

Oltre ai dispositivi mobili, come smartphone e tablet, Android ha trovato applicazione in settori come gli smartwatch e i televisori. È stato impiegato anche nella creazione di dispositivi per la realtà aumentata, la realtà virtuale e persino negli elettrodomestici come forni e frigoriferi, dando vita a una nuova era della tecnologia, quella della smart home e dell'Internet delle cose (IoT).

L'adozione diffusa di Android come sistema operativo ha comportato numerosi vantaggi. Innanzitutto, la vasta disponibilità di applicazioni sviluppate da terze parti aumenta considerevolmente le opzioni per gli utenti. Inoltre, la natura altamente personalizzabile di Android consente ai produttori di dispositivi di adattare l'interfaccia utente alle proprie esigenze specifiche.

1.1.2 ARCHITETTURA ANDROID



L'architettura di Android è strutturata su un modello a strati, che si compone di diversi componenti fondamentali:

1. **Kernel Linux**: Questo costituisce la base del sistema operativo e offre il supporto per i driver hardware, la gestione della memoria, il networking e altre funzionalità di base del sistema.
2. **Librerie native**: Si tratta di una serie di librerie scritte in linguaggio C/C++ che forniscono funzionalità avanzate per il sistema operativo, come la gestione dei file, la crittografia, la grafica, l'audio e altre operazioni di basso livello.
3. **Android Runtime**: Questo è l'ambiente di esecuzione per le applicazioni Android. Include il compilatore JIT (Just-In-Time) che traduce il codice delle applicazioni in codice macchina durante l'esecuzione. Dal 2014, Android Runtime (ART) ha sostituito il precedente Dalvik.
4. **Framework delle applicazioni**: È un insieme di librerie scritte in linguaggio Java che forniscono le API (Application Programming Interface) per lo sviluppo di applicazioni Android. Questo framework comprende librerie per la gestione dell'interfaccia utente, delle attività, dei servizi, delle notifiche, della persistenza dei dati e altre funzionalità di alto livello.
5. **Applicazioni**: Questi sono i programmi che gli utenti possono scaricare ed eseguire sui loro dispositivi Android. Le applicazioni Android sono principalmente sviluppate in linguaggio Java e possono interagire con il framework delle applicazioni per accedere alle funzionalità del sistema.

Questa **architettura** a strati consente una separazione chiara delle responsabilità e fornisce un ambiente flessibile e scalabile per lo sviluppo e l'esecuzione delle applicazioni Android.

L'architettura di Android si fonda sull'integrazione sinergica di componenti hardware e software, che lavorano in armonia per offrire funzionalità avanzate ai dispositivi Android. Grazie al modello a strati, vi è una chiara separazione tra i vari elementi del sistema, agevolando così la manutenzione e gli aggiornamenti delle diverse parti dell'architettura.

Questo approccio stratificato consente una gestione **efficiente** dei diversi componenti del sistema, facilitando sia lo **sviluppo** che la **manutenzione** del software. Inoltre, offre un'elevata flessibilità e scalabilità, permettendo l'integrazione agevole di nuove funzionalità e l'adattamento alle mutevoli esigenze del mercato e degli utenti.

Grazie a questa architettura ben strutturata, gli sviluppatori possono concentrarsi sulla creazione di esperienze utente innovative, mentre i produttori possono garantire prestazioni ottimali e aggiornamenti tempestivi per i dispositivi Android.

1.1.3 APP ANDROID

Gli **elementi fondamentali** di un'applicazione Android, utilizzati durante sia lo sviluppo che l'esecuzione dell'applicazione, sono i seguenti:

1. **Activity**: Rappresenta una singola schermata dell'applicazione, permettendo all'utente di visualizzare e interagire con l'interfaccia utente. Un'app può comprendere più activity, come ad esempio una schermata di login, una schermata delle impostazioni e una schermata principale.
2. **Service**: È un componente dell'applicazione che svolge attività in background senza necessità di interazione diretta con l'utente. Ad esempio, un'app di messaggistica può utilizzare un service per controllare periodicamente la presenza di nuovi messaggi e notificare l'utente in caso di necessità.
3. **Broadcast Receiver**: Riceve messaggi broadcast dal sistema operativo o da altre applicazioni. Ad esempio, un'applicazione di musica può utilizzare un broadcast receiver per ricevere notifiche sullo stato della riproduzione musicale.

4. **Content Provider:** Consente l'accesso ai dati dell'applicazione ad altre applicazioni. Ad esempio, un'app di contatti può utilizzare un content provider per permettere ad altre applicazioni di accedere ai contatti memorizzati.
5. **Intent:** È un oggetto che facilita l'avvio di altre activity, service o broadcast receiver, o la comunicazione di dati tra i vari componenti dell'applicazione o tra applicazioni diverse.
6. **Layout:** Definisce l'aspetto e la disposizione degli elementi dell'interfaccia utente dell'applicazione attraverso un file XML.
7. **Manifest:** Contiene le informazioni essenziali sull'applicazione, come il nome del pacchetto, la versione, l'elenco dei componenti (activity, service, broadcast receiver) e i permessi richiesti.
8. **Risorse:** File come immagini, icone e stringhe utilizzati dall'applicazione per personalizzare l'aspetto e il comportamento.

Il formato delle applicazioni Android è l'APK (Android Package), un file archivio compresso che include tutti gli elementi costitutivi dell'applicazione (codice sorgente, risorse, librerie e file di configurazione). Questo formato semplifica la distribuzione delle applicazioni, consentendo agli utenti di scaricare e installare facilmente le app dal Play Store di Google. Tuttavia, la facilità di modifica dell'APK può rappresentare una minaccia per la sicurezza se non vengono adottate adeguate misure di sicurezza.

1.2 PENETRATION TEST

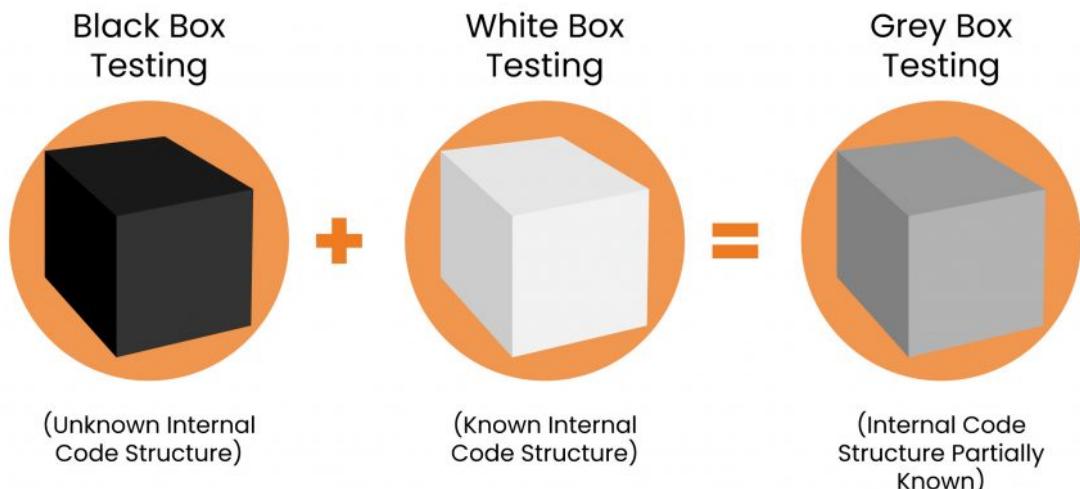
Il **Penetration Testing** su Android è un'attività cruciale per garantire la sicurezza delle applicazioni e dei dispositivi Android. Deve essere condotto regolarmente per garantire una protezione continua contro le minacce alla sicurezza.

Questo processo implica l'**analisi delle vulnerabilità** dei sistemi, delle applicazioni o dei dispositivi per individuare eventuali falle nella sicurezza.

Tale tipo di test è particolarmente importante per identificare le vulnerabilità nelle **applicazioni mobili**, le quali spesso hanno accesso a informazioni sensibili come i dati dell'utente, le credenziali di accesso e altri dati riservati.

Le vulnerabilità nelle applicazioni possono essere sfruttate dagli attaccanti per accedere a queste informazioni o per eseguire azioni dannose sul dispositivo o sulla rete.

Types Of Testing Methods

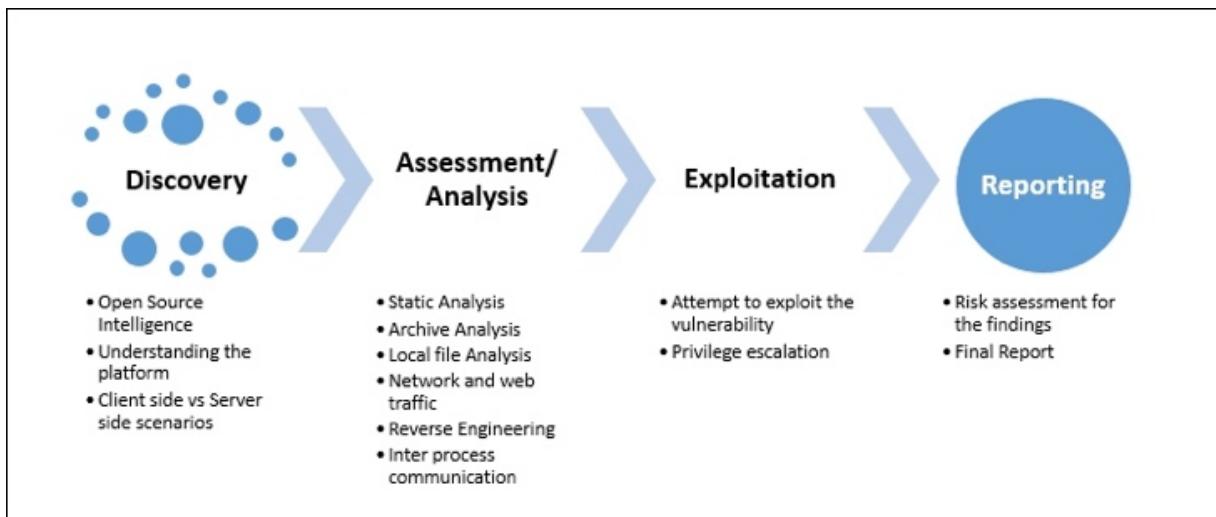


Il **Mobile Penetration Test**, o "Pen-Test" in breve, consiste nell'analizzare la sicurezza dei dispositivi mobili e delle applicazioni installate su di essi. Può essere eseguito in diversi modi, a seconda del livello di conoscenza dell'attaccante riguardo all'applicazione e al dispositivo soggetto al test. Le principali tipologie di Penetration Testing includono:

- **White Box:** Il tester ha accesso completo all'applicazione e al dispositivo, conosce la sua struttura, il codice sorgente e l'infrastruttura di rete. Questo tipo di test è spesso condotto internamente dalle aziende o dagli sviluppatori stessi per valutare la sicurezza del proprio prodotto.
- **Black Box:** Il tester non ha alcuna conoscenza pregressa sull'applicazione o sul dispositivo. Deve agire come un attaccante esterno che cerca di penetrare il sistema. Questo tipo di test è utile per valutare la capacità di difesa del sistema di fronte a un attaccante sconosciuto.
- **Gray Box:** Il tester ha una conoscenza parziale dell'applicazione o del dispositivo. Ad esempio, potrebbe conoscere la struttura dell'applicazione ma non il codice sorgente. Questo tipo di test può essere utilizzato per verificare la sicurezza dell'applicazione senza avere un accesso completo al sistema.

Ognuna di queste metodologie offre un approccio unico per valutare la sicurezza di un'applicazione o di un dispositivo Android, consentendo di identificare e risolvere eventuali vulnerabilità prima che vengano sfruttate da potenziali attaccanti.

1.2.1 FASI DEL MOBILE PEN TEST



Il **Mobile Penetration Test** è strutturato in quattro fasi fondamentali:

1. **Raccolta delle informazioni:** Questa fase consiste nella raccolta di informazioni sull'applicazione o sul dispositivo mobile da testare, nonché sulle tecnologie e sui protocolli utilizzati. L'obiettivo è acquisire il maggior numero possibile di informazioni per pianificare il test in modo efficace.
2. **Analisi delle vulnerabilità:** Durante questa fase, vengono eseguiti test per individuare le vulnerabilità dell'applicazione o del dispositivo. Questo può includere l'uso di strumenti per l'analisi statica e dinamica del codice, il testing dell'interfaccia utente e la verifica dei controlli di input/output.
3. **Sfruttamento delle vulnerabilità:** In questa fase, vengono eseguiti attacchi per sfruttare le vulnerabilità individuate nella fase precedente. Gli attacchi possono includere tentativi di bypassare i controlli di autenticazione, l'iniezione di codice malevolo o l'utilizzo di tecniche di ingegneria sociale.
4. **Fase di report:** Alla conclusione del test, viene generato un report dettagliato contenente tutte le informazioni raccolte nelle fasi precedenti, inclusi i risultati delle analisi delle vulnerabilità e degli attacchi eseguiti. Il report dovrebbe anche fornire raccomandazioni per migliorare la sicurezza dell'applicazione o del dispositivo.

Queste quattro fasi costituiscono un **processo completo per valutare la sicurezza** di un'applicazione o di un dispositivo mobile e per identificare e risolvere eventuali vulnerabilità che potrebbero compromettere la sicurezza complessiva del sistema.

1.2.2 ANALISI DELLE VULNERABILITÀ

L'analisi delle vulnerabilità rappresenta una fase cruciale del Penetration Test, incentrata sull'**individuazione delle debolezze** presenti in un sistema o in un'applicazione.

Nel contesto del Mobile Penetration Test, l'obiettivo principale è rilevare le vulnerabilità specifiche di un'applicazione Android e determinare se queste possono essere sfruttate da un attaccante per ottenere accesso non autorizzato ai dati o al sistema.

Questa fase si articola in **due sottofasi**:

1. **Reconnaissance:** Durante questa fase, si acquisiscono informazioni sul sistema o sull'applicazione da testare. Ciò può coinvolgere l'uso di strumenti di scansione automatica

- per identificare vulnerabilità note o la revisione manuale del codice dell'applicazione per individuare eventuali problemi di sicurezza.
2. **Documentazione:** Le vulnerabilità individuate vengono documentate in modo chiaro e completo, includendo dettagli tecnici sulla vulnerabilità stessa e raccomandazioni per la sua correzione. Questa documentazione deve essere comprensibile sia per il team di sviluppo che per il proprietario dell'applicazione, facilitando così la correzione delle vulnerabilità individuate.

In generale, l'analisi delle vulnerabilità è un processo iterativo che potrebbe richiedere più cicli di analisi e test per individuare tutte le vulnerabilità presenti in un'applicazione Android.

Tuttavia, una sua esecuzione accurata e completa può contribuire a ridurre i rischi di sicurezza e a proteggere i dati dell'utente e il sistema da potenziali attacchi.

1.2.3 ANALISI STATICHE E ANALISI DINAMICHE

L'**Analisi Statica (SAST)** e l'**Analisi Dinamica (DAST)** rappresentano due approcci fondamentali per condurre un Mobile Penetration Test, entrambi essenziali per garantire una valutazione completa della sicurezza di un'applicazione.

L'**analisi statica (SAST)** consiste nell'esaminare il codice sorgente dell'applicazione senza eseguire effettivamente l'applicazione stessa.

Il suo obiettivo principale è individuare le vulnerabilità nella progettazione e nel codice dell'applicazione.

Gli strumenti di analisi statica possono rilevare vulnerabilità comuni come buffer overflow, SQL injection, cross-site scripting e altre vulnerabilità di codice.

Dall'altro lato, l'**analisi dinamica (DAST)** si concentra sull'esecuzione effettiva dell'applicazione in un ambiente di test reale per identificare le vulnerabilità che potrebbero emergere durante le interazioni tra l'applicazione e il sistema operativo o altre applicazioni di terze parti presenti sul dispositivo. Questo tipo di analisi mira a individuare le vulnerabilità di sicurezza che possono emergere nel flusso di lavoro dell'applicazione, come l'invio di dati malevoli, l'intercettazione del traffico di rete o la manipolazione dei dati.

Questi due approcci sono **complementari** e vengono spesso **utilizzati insieme** per ottenere una valutazione completa della sicurezza dell'applicazione.

L'analisi statica è particolarmente utile per identificare le vulnerabilità a livello di codice sorgente, consentendo agli sviluppatori di correggere eventuali problemi durante la fase di sviluppo.

D'altra parte, l'analisi dinamica è efficace nel rilevare le vulnerabilità che emergono durante l'esecuzione effettiva dell'applicazione, offrendo una visione più realistica delle minacce potenziali a livello di **runtime**.

1.3 SICUREZZA ED ANDROID

La **sicurezza delle applicazioni Android** riveste un ruolo di primaria importanza poiché queste applicazioni spesso gestiscono dati sensibili degli utenti, come informazioni personali, credenziali di accesso e dati di pagamento.

Un elemento cruciale per garantire la sicurezza in ambito Android è l'**adozione di un'architettura sicura**. Questo implica la progettazione dell'applicazione in modo tale da proteggere i **dati sensibili** dagli attacchi esterni. Elementi chiave includono la **separazione del codice** dell'applicazione da eventuali codici esterni, l'implementazione di meccanismi di cifratura per proteggere i dati sensibili, una rigorosa gestione delle autorizzazioni di accesso ai dati e l'assicurazione della sicurezza delle comunicazioni.

Inoltre, è fondamentale che le applicazioni Android ricevano **aggiornamenti regolari**. Gli sviluppatori dovrebbero rilasciare patch di sicurezza e miglioramenti delle funzionalità per mantenere l'applicazione al passo con le minacce emergenti e le esigenze degli utenti. Parallelamente, gli utenti stessi dovrebbero essere diligenti nell'installare gli aggiornamenti appena disponibili per garantire una maggiore sicurezza.

Per proteggere i dati sensibili degli utenti durante la trasmissione e l'archiviazione, le applicazioni dovrebbero adottare robusti protocolli di **crittografia**. Ciò assicura che le informazioni sensibili siano inaccessibili a terze parti non autorizzate.

Infine, l'implementazione di un **sistema di autenticazione** robusto è essenziale per garantire che solo gli utenti autorizzati possano accedere ai dati sensibili. Questo può includere l'uso di password robuste, autenticazione biometrica o token di autenticazione per verificare l'identità degli utenti e prevenire l'accesso non autorizzato.

In sintesi, una combinazione di architettura sicura, aggiornamenti regolari, crittografia dei dati e autenticazione robusta sono **elementi chiave** per garantire la sicurezza delle applicazioni Android e proteggere i dati sensibili degli utenti da minacce esterne.

1.4 VULNERABILITÀ OWASP (OPEN WEB APPLICATION SECURITY)

1.1.1. OWASP

OWASP (Open Web Application Security Project) è una comunità globale che opera dal 2001 con l'obiettivo di migliorare la sicurezza delle applicazioni web attraverso sensibilizzazione, educazione e sviluppo di strumenti. Questa organizzazione, composta da volontari, si impegna nella creazione di risorse, linee guida, strumenti e progetti gratuiti per promuovere la sicurezza delle applicazioni web.

Tra le attività principali di OWASP vi sono la pubblicazione di **linee guida**, lo sviluppo di **strumenti e progetti open-source**, nonché l'organizzazione di eventi e corsi di formazione.

Un'altra importante iniziativa consiste nella pubblicazione di documenti che mettono in luce le principali vulnerabilità presenti nei sistemi software.

OWASP gestisce diversi **progetti significativi**, tra cui:

- **OWASP Top 10**: Una lista delle dieci vulnerabilità più critiche nelle applicazioni web, aggiornata periodicamente per riflettere le nuove minacce e le nuove tecnologie.
- **OWASP Mobile Top 10**: Una lista simile per le applicazioni mobile, che fornisce una descrizione, esempi di attacchi, possibili conseguenze e contromisure per ciascuna vulnerabilità elencata. È stata aggiornata per l'ultima volta nel 2021.
- **Mobile Application Security Verification Standard (MASVS)**: Un documento che presenta un framework per la verifica della sicurezza delle applicazioni mobili. Il MASVS stabilisce una serie di requisiti che le applicazioni mobili devono soddisfare per garantire un livello minimo di sicurezza. Il documento è suddiviso in tre livelli di sicurezza e fornisce una serie di test per verificare la conformità dell'applicazione ai requisiti.

Questi progetti forniscono un quadro essenziale per comprendere e affrontare le sfide relative alla sicurezza delle applicazioni web e mobili, offrendo alle organizzazioni e agli sviluppatori gli strumenti necessari per proteggere le loro applicazioni dagli attacchi informatici.

1.4.1.1. OWASP Top 10



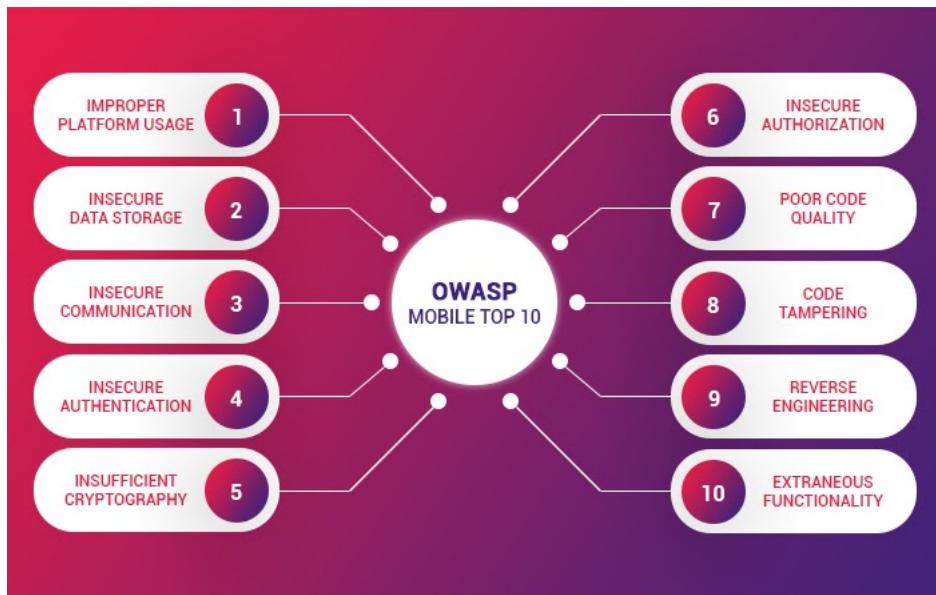
La **OWASP Top 10** è una lista consolidata delle **dieci vulnerabilità** più critiche che possono essere presenti nelle applicazioni web.

La versione più recente, la OWASP Top 10 2021, elenca le seguenti vulnerabilità:

1. **Broken Access Control (A01)**: Questa vulnerabilità si verifica quando un'applicazione web non implementa correttamente i controlli di autorizzazione, consentendo agli utenti di accedere a risorse o funzionalità a cui non dovrebbero avere accesso.
2. **Cryptographic Failures (A02)**: Si verifica quando vengono utilizzati in modo errato o debole meccanismi crittografici per proteggere dati sensibili, compromettendo la riservatezza e l'integrità dei dati.
3. **Injections (A03)**: Queste vulnerabilità includono le SQL injection e le XSS (Cross-Site Scripting), che consentono agli attaccanti di eseguire codice malevolo o ottenere accesso non autorizzato ai dati.
4. **Insecure Design (A04)**: Deriva da scelte di progettazione errate o inadeguate che rendono l'applicazione vulnerabile ad attacchi come spoofing e man-in-the-middle.
5. **Security Misconfigurations (A05)**: Riguarda impostazioni di configurazione errate o non sicure che possono essere sfruttate dagli attaccanti per ottenere accesso non autorizzato o compromettere l'integrità dei sistemi.
6. **Vulnerable and Outdated Components (A06)**: Si riferisce all'utilizzo di componenti software obsoleti o conosciuti per presentare falle di sicurezza note, che possono essere sfruttati dagli attaccanti.
7. **Identification and Authentication Flaws (A07)**: Include errori nel processo di identificazione e autenticazione degli utenti, come la mancata protezione delle credenziali di accesso.
8. **Software and Data Integrity Failures (A08)**: Si verifica quando un'applicazione web non garantisce l'integrità dei dati o del software che utilizza.
9. **Security Logging and Monitoring Flaws (A09)**: Riguarda errori o problemi nella registrazione e nel monitoraggio delle attività di sicurezza all'interno dell'applicazione.
10. **Server-Side Request Forgery (SSRF) (A10)**: Si verifica quando un'applicazione web consente a un attaccante di forzare il server a effettuare richieste a risorse esterne non autorizzate.

La OWASP Top 10 è diventata uno **standard** nel settore della sicurezza delle applicazioni web ed è utilizzata come punto di partenza per l'analisi della sicurezza delle applicazioni.

1.1.1.2. OWASP Mobile Top 10

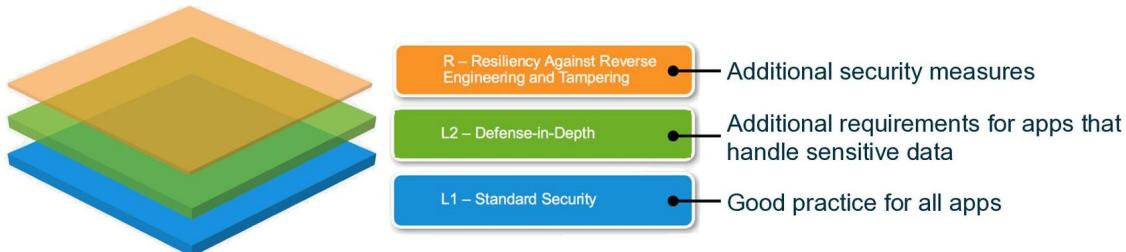


La **OWASP Mobile Top 10** del 2016 elenca dieci vulnerabilità principali che possono essere presenti nelle applicazioni mobili. Ecco una breve descrizione di ciascuna:

1. **M01 Improper Platform Usage:** Si verifica quando un'applicazione mobile non utilizza correttamente le funzionalità di sicurezza fornite dalla piattaforma mobile, aprendo la porta a vari tipi di attacchi.
2. **M02 Insecure Data Storage:** Avviene quando i dati sensibili vengono memorizzati in modo non sicuro sul dispositivo mobile, compromettendo la privacy degli utenti.
3. **M03 Insecure Communication:** Si riferisce alla comunicazione non sicura tra l'applicazione mobile e i server backend o altri servizi esterni, che può essere intercettata e manipolata dagli aggressori.
4. **M04 Insecure Authentication:** Riguarda le debolezze nell'implementazione del processo di autenticazione dell'applicazione mobile, che possono consentire agli aggressori di compromettere le credenziali degli utenti.
5. **M05 Insufficient Cryptography:** Si verifica quando la crittografia utilizzata nell'applicazione mobile è debole o non implementata correttamente, permettendo agli aggressori di violare la confidenzialità e l'integrità dei dati crittografati.
6. **M06 Insecure Authorization:** Si riferisce a difetti nell'implementazione del controllo delle autorizzazioni all'interno dell'applicazione mobile, che possono consentire a utenti non autorizzati di eseguire azioni privilegiate.
7. **M07 Poor Code Quality:** Deriva da una scarsa qualità del codice sorgente dell'applicazione mobile, che può includere vulnerabilità note o errori di programmazione.
8. **M08 Code Tampering:** Avviene quando il codice dell'applicazione mobile viene compromesso o alterato da un aggressore, mettendo a rischio l'integrità dell'applicazione e dei dati associati.
9. **M09 Reverse Engineering:** Si riferisce alla possibilità per un aggressore di eseguire il reverse engineering sull'applicazione mobile per identificare vulnerabilità o creare versioni modificate dell'applicazione.
10. **M10 Extraneous Functionality:** Riguarda l'inclusione di funzionalità non necessarie all'interno dell'applicazione mobile, che possono introdurre potenziali punti deboli.

1.1.1.3. OWASP MASVS

OWASP Mobile Application Security Verification Standard Security Levels



Il **Mobile Application Security Verification Standard (MASVS)** fornisce una guida completa per garantire la sicurezza delle applicazioni mobili.

Definisce tre livelli di verifica della sicurezza:

1. **MASVS-L1 (Standard Security)**: Questo livello rappresenta il fondamento della sicurezza delle app mobili. Include controlli essenziali come l'autenticazione, la gestione delle sessioni e la crittografia dei dati sensibili. Il suo obiettivo è proteggere le app dai rischi di base e prevenire gli attacchi comuni.
2. **MASVS-L2 (Defense in Depth)**: Questo livello è più avanzato e richiede controlli aggiuntivi per una protezione più completa contro minacce sofisticate. Si concentra sulla verifica dell'integrità del codice, la protezione degli algoritmi crittografici e altre misure di sicurezza avanzate.
3. **MASVS-R (Resilience)**: Questo livello è specificamente progettato per proteggere le app contro il reverse engineering. Suggerisce misure come l'opacizzazione del codice e l'implementazione di controlli anti-debugging per rendere più difficile l'analisi e la manipolazione dell'app da parte di malintenzionati.

L'implementazione dei livelli di verifica della sicurezza e del livello di resilienza definiti dal MASVS è cruciale per garantire che le app mobili soddisfino i requisiti di sicurezza e siano protette da minacce comuni e non comuni. Adottare queste misure contribuisce a rafforzare la sicurezza delle applicazioni mobili e a proteggere i dati sensibili degli utenti.

2. CASO DI STUDIO

L'applicazione presa in esame per il nostro caso di studio è stata l'app: **AigoSmart** presente sul Play Store e scaricabile gratuitamente.

La versione presente sullo store è la 2.3.6 uscita il 12 aprile 2024.

AigoSmart

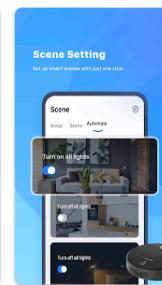
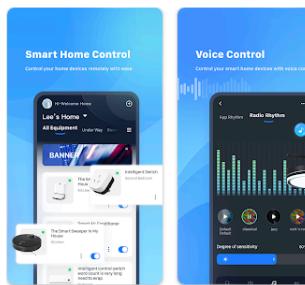
Aigostar

3,1★
2500 recensioni | 100.000+
Download | Per tutti ⓘ

Installa su altri dispositivi

Condividi

Questo app è disponibile per il tuo dispositivo



Assistenza per le app ▾

Altre app da provare →



Airbnb
4,5★



Flightradar24 Flight Tracker
Flightradar24 AB
4,6★

La versione presa in esame è la versione **1.9.0** del 18 aprile 2022, scaricata dal sito **APKPure**.

APKPure, lanciato nel 2014, si è rapidamente affermato come uno dei **principali store alternativi** a Google Play. Nonostante la sua popolarità, è importante essere consapevoli dei potenziali rischi legati alla privacy e alla sicurezza che presenta.

Questo servizio offre **un vasto catalogo di app Android** in formato .apk e .xapk, sia freeware che shareware, accessibili tramite download diretto.

Tuttavia, a differenza delle app disponibili sul Google Play Store, quelle su APKPure **non sono soggette agli stessi rigidi protocolli di sicurezza**. Ciò significa che potrebbero contenere vulnerabilità o addirittura malware. Le app Android, quando autorizzate, possono accedere a una vasta gamma di funzionalità del dispositivo, compresi i dati personali. In alcuni casi, possono farlo anche senza un'esplicita autorizzazione, comportando rischi per la privacy.

Per proteggere i tuoi dati, è consigliabile **crittografarli** e fare un uso oculato delle **autorizzazioni** concesse alle app Android. Inoltre, è fondamentale prestare attenzione alle app scaricate, indipendentemente dalla loro fonte. La sicurezza dei dati personali e la protezione dell'identità digitale dovrebbero essere priorità in ogni situazione online.

Riguardo all'app presa in esame, registra più di **100.000+ download** su playstore e disponibile in **13 lingue** (Italiano, Cinese Semplificato, Danese, Francese, Inglese, Norvegese, Olandese, Polacco, Portoghese, Spagnolo, Svedese, Tedesco, Ungherese)

2.1 FUNZIONALITÀ

AigoSmart è un'app basata su Android sviluppata da Aigostar che ti permette di gestire e controllare i tuoi elettrodomestici smart da remoto.

Permette anche ai tuoi dispositivi di percepire e interagire tra loro, offrendo un'esperienza senza interruzioni e conveniente.

L'app ti consente di impostare preimpostazioni per i tuoi elettrodomestici in modo che con un solo tocco, tutti i dispositivi collegati passino allo stato desiderato, garantendo il massimo comfort e sicurezza.

L'interfaccia di **AigoSmart** è intuitiva e facile da navigare. Puoi facilmente aggiungere i tuoi dispositivi smart all'app e controllarli con un semplice tocco. L'app fornisce anche aggiornamenti in tempo reale sullo stato dei tuoi elettrodomestici, offrendoti un controllo completo sulla tua casa.

Con **AigoSmart**, puoi creare una vita sicura e spensierata in una casa smart che si adatta alle tue esigenze e preferenze.

In generale, **AigoSmart** è un'app affidabile ed efficiente che semplifica la gestione dei tuoi elettrodomestici smart. La sua interfaccia intuitiva, combinata alle sue eccellenti funzionalità, la rende un must-have per chiunque voglia controllare i propri elettrodomestici da remoto.

2.2 STRUMENTI UTILIZZATI

In questa sezione viene svolta una panoramica degli **strumenti** che sono stati impiegati per svolgere le **verifiche** e le **indagini** condotte all'interno di tutto l'elaborato.

2.2.1 VIRTUALBOX



VirtualBox rappresenta un software per eseguire la virtualizzazione. È stato realizzato dalla società Oracle ed è un tool open source contraddistinto da una elevata qualità.

Attraverso quest'ultimo diviene possibile disporre di un calcolatore riprodotto all'interno di una cosiddetta **macchina virtuale**.

Essa rappresenta un'astrazione e bisogna necessariamente affiancarlo ad un sistema operativo.

2.2.2 KALI LINUX



Kali Linux è una distribuzione **Linux** sviluppata e manutenuta attivamente dalla Offensive Security, una multinazionale americana che si occupa di sicurezza informatica, penetration testing e informatica forense.

Essa si basa su **Debian**, un'altra ben più solida e matura distribuzione Linux.

È particolarmente rilevante in quanto dispone di una serie di **strumenti ad hoc** per eseguire Penetration Testing nonché Security Auditing.

2.2.3 ANDROID TAMER



Android Tamer è un'altra distribuzione **Linux** sviluppata per i professionisti che operano nel campo della **sicurezza Android**. Dispone di un'ampia varietà di software preinstallati per effettuare test di verifica della sicurezza di applicativi Android,

reverse engineering, penetration testing e malware analysis. Si basa su **Ubuntu 10.04 Long Term Support (LTS)**.

2.2.4 GENYMOTION

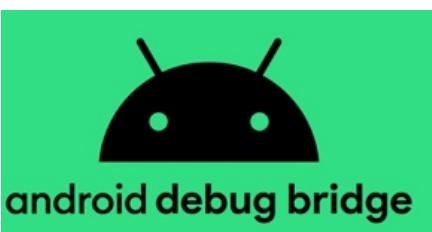


è necessario utilizzare **VirtualBox** per poter sfruttare le funzionalità offerte da Genymotion.

Genymotion è un software che permette l'**emulazione di dispositivi Android** con permessi di root preinstallati.

Diviene possibile utilizzare sia smartphone sia tablet, selezionando la versione del sistema operativo Android da

2.2.5 ANDROID DEBUG BRIDGE (ADB)



Android Debug Bridge, anche conosciuto come ADB, è un software a riga di comando che permette di comunicare con un dispositivo Android.

Il suo utilizzo permette di **semplificare l'interazione** con un dispositivo, poiché si va ad interagire direttamente con il sistema operativo.

Di conseguenza è possibile installare ed eseguire il **debug** delle applicazioni. Inoltre, permette l'accesso ad una shell di tipo Unix.

2.2.6 DOCKER DESKTOP



Docker Desktop è un'applicazione di installazione con un clic per l'ambiente Mac, Linux o Windows che consente di creare, condividere ed eseguire applicazioni e microservizi containerizzati. Fornisce una semplice GUI (Graphical User Interface) che consente di gestire i tuoi contenitori, applicazioni e immagini direttamente dalla tua macchina.

Docker Desktop riduce il tempo speso per configurazioni complesse in modo da poterti concentrare sulla scrittura di codice. Si occupa delle **mappature delle porte**, dei problemi del file system e di altre impostazioni predefinite e viene regolarmente aggiornato con correzioni di bug e aggiornamenti di sicurezza. Nel nostro caso è stato utilizzato per l'utilizzo del **tool MOBSF da locale**.

2.2.7 MOBSF (MOBILE SECURITY FRAMEWORK)



MobSF è un tool di **analisi automatica** per lo studio e la valutazione della sicurezza delle applicazioni mobili.

Esso può essere utilizzato in due modalità: **online**, ossia quella web accessibile tramite browser che anch'essa permette l'upload dell'apk, e la versione da **locale**, utilizzata per questo caso di studio. Viene consigliato dal **OWASP MSTG** per l'analisi statica di sicurezza delle applicazioni mobile e, oltre a restituire un **report** circa le vulnerabilità riscontrate ed ordinate sulla base della loro criticità, restituisce anche un indice numerico da 0 a 100 sulla sicurezza dell'app analizzata ed un grado di rischio da A, fino ad F in ordine crescente.

2.2.8 JAVA DECOMPILER (JD-GUI)



JD-GUI è un potente software open source di **decompilazione** Java. Viene fornito insieme ad una interfaccia grafica di supporto che permette l'accesso e la visualizzazione del codice Java delle classi presenti.

È necessario fornire in input un file **jar**, ottenuto per esempio attraverso MaraFramework. Pertanto, sarà possibile eseguire un reverse engineering di un **applicativo Android**.

2.2.9 MARA (MOBILE APPLICATION REVERSE ENGINEERING AND ANALYSIS)



MARA è un framework per l'ingegneria inversa e l'analisi di applicazioni mobili. È progettato per consentire agli sviluppatori e agli esperti di sicurezza di **esaminare** il funzionamento interno delle app mobili, comprese le loro componenti, le API utilizzate e i dati memorizzati.

MARA offre funzionalità come la **decompilazione** del bytecode, l'**estrazione** delle risorse, l'**analisi** delle vulnerabilità e la **tracciatura** delle chiamate di API.

È uno strumento utile per comprendere il **comportamento** delle **app mobili** e identificare possibili problemi di sicurezza o di privacy.

2.2.10 IMMUNIWEB



barra di ricerca ed in quel caso verrà analizzata l'ultima versione presente sul rispettivo Store. Successivamente partirà l'analisi che restituirà un **report** contenente le vulnerabilità riscontrate e divise secondo **4 livelli di pericolosità**: high risk, medium risk, low risk e warning ed inoltre, anche catalogate nelle **10 categorie della top ten OWASP**.

Vengono in fine mostrati tutti i permessi richiesti dall'app e anch'essi catalogati in **3 categorie**: dangerous, normal ed unknown.

2.2.11 VISUAL STUDIO CODE



Visual Studio Code, spesso abbreviato come VS Code, è un potente **editor** di codice sorgente sviluppato da Microsoft per piattaforme come Windows, Linux, macOS e anche per l'uso tramite browser web.

Tra le sue caratteristiche principali ci sono il supporto per il **debug**, l'evidenziazione della sintassi, il completamento automatico del codice, gli snippet, il refactoring del codice e un sistema di controllo delle versioni integrato

basato su Git. Gli utenti possono personalizzare l'editor secondo le proprie preferenze, modificando il tema, le scorciatoie da tastiera e le impostazioni, e possono arricchirne le funzionalità installando le estensioni disponibili.

2.3 GUIDA ALL'USO DEGLI STRUMENTI UTILIZZATI

2.3.1 MOBSF

Il file APK è stato sottoposto a **un'analisi automatica** utilizzando lo strumento **MOBsf**.

Questo strumento, una volta caricato il file, esegue **un'analisi dettagliata** dell'applicazione, valutando il livello di sicurezza e individuando eventuali vulnerabilità presenti.

Questo processo è cruciale per **valutare la sicurezza complessiva dell'app**, poiché consente di esaminare il file APK senza eseguirlo effettivamente, permettendo così di rilevare e identificare potenziali problemi di sicurezza senza dover interagire direttamente con l'app stessa.

MOBsf è uno strumento **ampiamente diffuso** nella comunità della sicurezza informatica per l'analisi statica di applicazioni mobili.

Questo strumento genera un punteggio di sicurezza per l'applicazione basato sulle vulnerabilità rilevate, offrendo quindi un'indicazione della solidità della sicurezza dell'app stessa.

Una volta completata l'analisi statica, il **punteggio di sicurezza** ottenuto dall'applicazione può essere utilizzato come punto di partenza per le attività di correzione e miglioramento. Questa prassi consente di individuare e affrontare le vulnerabilità prima che l'applicazione venga distribuita, contribuendo a fornire un'esperienza più sicura agli utenti finali.

Per eseguire MobSF, è necessario avere questi **prerequisiti**:

- Un computer con Docker installato
- Un'applicazione mobile da analizzare

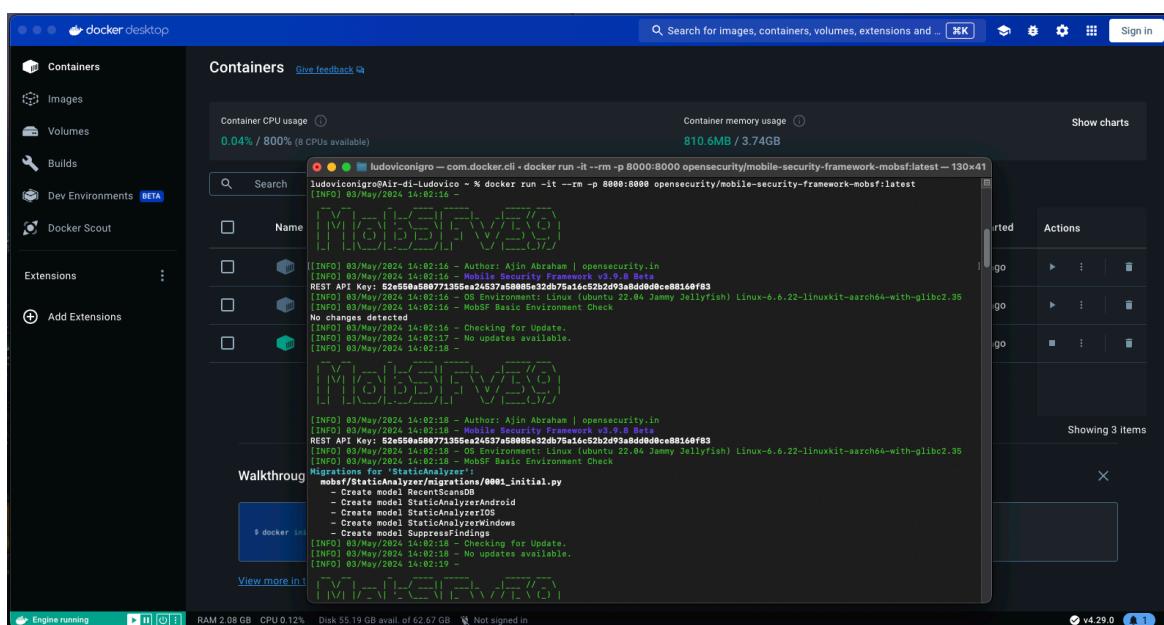
Il modo migliore per scaricare MobSF è utilizzare Docker. Per farlo, aprire un terminale ed eseguire il seguente comando:

```
docker pull opensecurity/mobile-security-framework-mobsf:latest
```

Questo comando scaricherà l'immagine Docker più recente di MobSF.

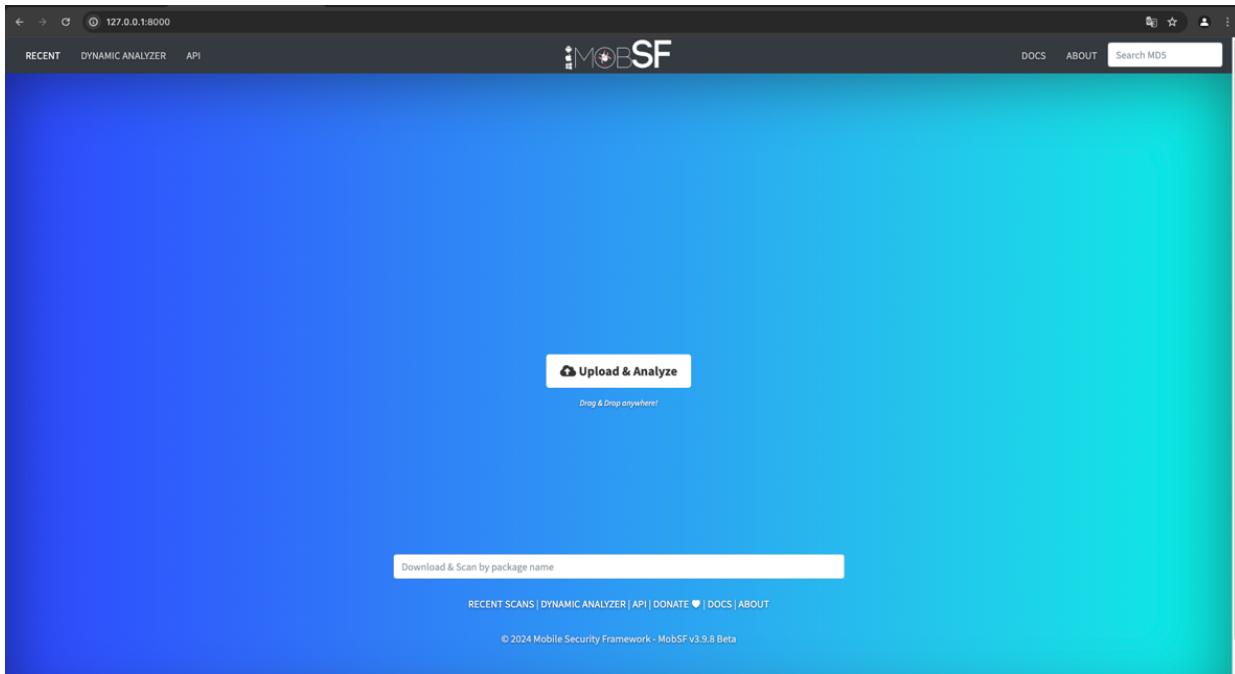
Una volta scaricata l'immagine Docker, è possibile eseguire MobSF con il seguente comando:

```
docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```



Questo comando eseguirà MobSF in modalità **interattiva** e **mapperà** la porta **8000** del computer locale alla porta 8000 del container Docker.

È possibile accedere all'**interfaccia web** di MobSF aprendo un browser web e andando su **http://127.0.0.1:8000**.



La figura illustra il procedimento per caricare e analizzare un'applicazione (APK) utilizzando il pulsante "Upload & Analyze".

Questa funzionalità permette agli utenti di caricare il file APK all'interno dello strumento **MOBsf** per avviare **un'analisi statica dell'applicazione**.

A screenshot of the MobSF static analyzer results for the AigoSmart_1.9.0_Apkpure.apk file. The interface is divided into several sections: 1. **APP SCORES**: Shows a security score of 26/100 and 4/432 for Traceroute Detection. 2. **FILE INFORMATION**: Provides details like File Name (AigoSmart_1.9.0_Apkpure.apk), Size (43.93MB), MD5, SHA1, SHA256, and Target SDK (30). 3. **PLAYSTORE INFORMATION**: Lists the app's title (AigoSmart), developer (Aigostar), release date (Jan 3, 2019), and a description mentioning it helps manage smart home appliances. 4. **APP INFORMATION**: Shows the app's name (AigoSmart), package name (com.aigostar.smart), main activity (com.aigostar.module.common.module.main.SplashActivity), target SDK (30), min SDK (21), max SDK (21), and Android version code (77). 5. **ACTIVITIES, SERVICES, RECEIVERS, PROVIDERS**: Summary statistics for the app's components: 179 Activities, 34 Services, 24 Receivers, and 6 Providers. Below these are detailed lists of exported activities, services, receivers, and providers with their respective counts (e.g., 8 Exported Activities, 16 Exported Services, 12 Exported Receivers, 0 Exported Providers).

2.3.2 KALI LINUX & ANDROID TAMER

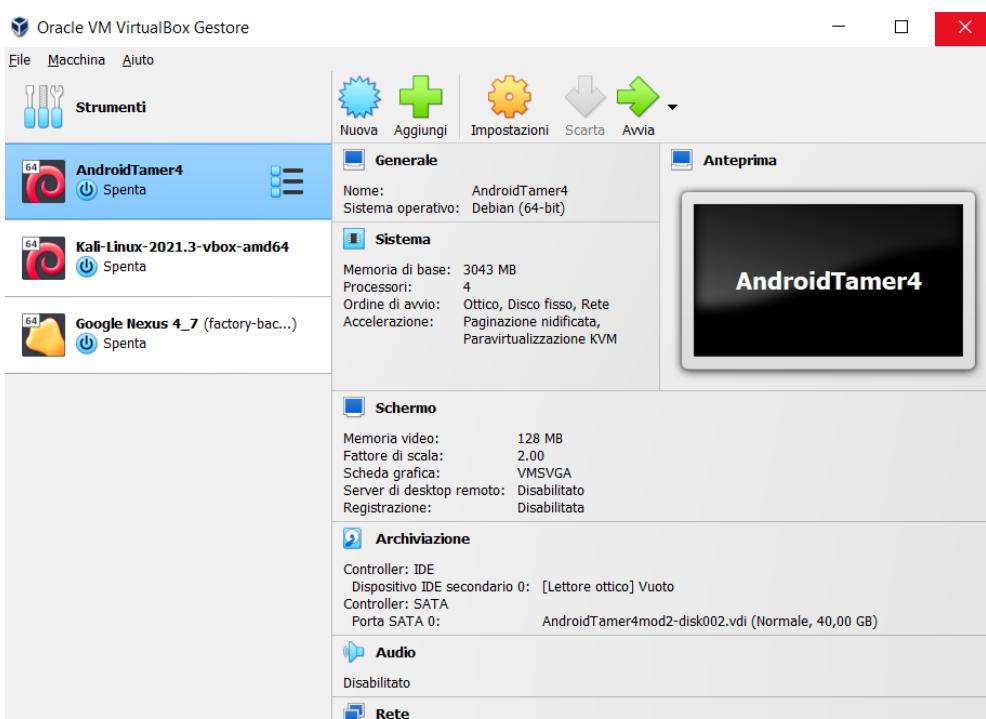
Per la **preparazione dell'ambiente** dove saranno effettuate le varie operazioni di Penetration Testing, i due tool fondamentali sono **VirtualBox** e **GenyMotion**, entrambi descritti nella sezione precedente, la cui installazione va effettuata sulla macchina host.

Per la creazione **macchina virtuale**, il primo passo consiste nell'importare la macchina virtuale con sistema operativo Android Tamer, che utilizzeremo per condurre i test.

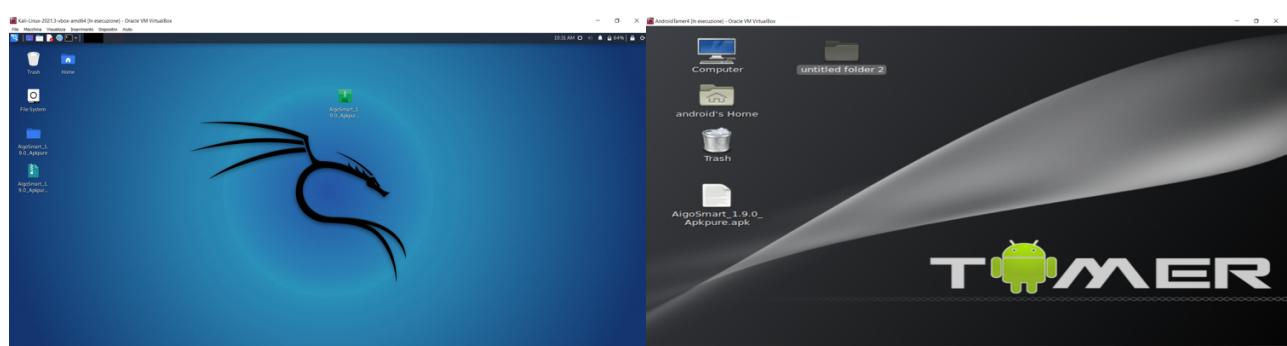
Per fare ciò, avviamo **VirtualBox** e procediamo con l'importazione del **file .ova**, il quale contiene una macchina virtuale già configurata e pronta all'uso, evitando così la necessità di creare una nuova macchina e installare il sistema operativo da zero.

È necessario scaricare il **file .ova** della macchina virtuale desiderata, nel nostro caso sia **Android Tamer** che **Kali Linux**.

Una volta ottenuto il file .ova, procediamo con l'importazione della macchina virtuale in VirtualBox. Dopo aver selezionato il file .ova, VirtualBox ci mostrerà le caratteristiche della macchina virtuale, che dovremo confermare prima di avviare l'importazione.



A questo punto basta **avviare** le macchine virtuali, e dopo aver inserito le **credenziali**, è possibile accedervi agli **strumenti**.



2.3.3 GENYMOTION

A questo punto, è necessario **virtualizzare il dispositivo mobile** su cui installeremo l'app da testare. Per fare ciò, utilizziamo **GenyMotion**.

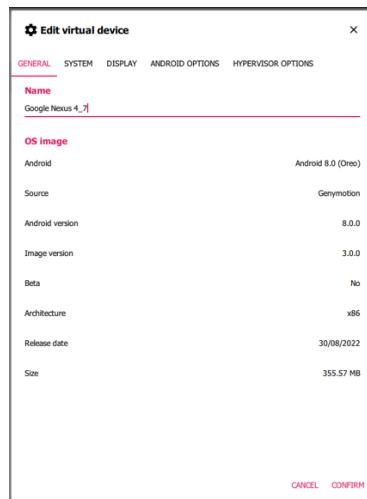
Avviando il **software**, possiamo **creare un nuovo dispositivo** e accedere a una vasta gamma di opzioni che coprono praticamente tutti i dispositivi presenti sul mercato Android, insieme alle varie versioni del sistema operativo e alle relative API.



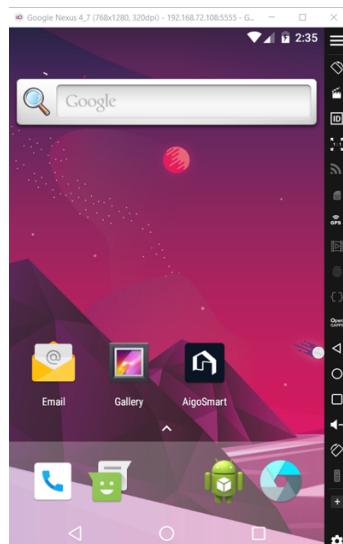
Possiamo scegliere un dispositivo scorrendo l'elenco disponibile o cercandolo tramite la barra di ricerca se desideriamo un modello specifico.

Una volta selezionato il dispositivo, possiamo personalizzarlo scegliendo la versione del sistema operativo, le API, il numero di CPU e le dimensioni dello schermo.

In questo caso di studio il dispositivo scelto presenta queste **caratteristiche**:



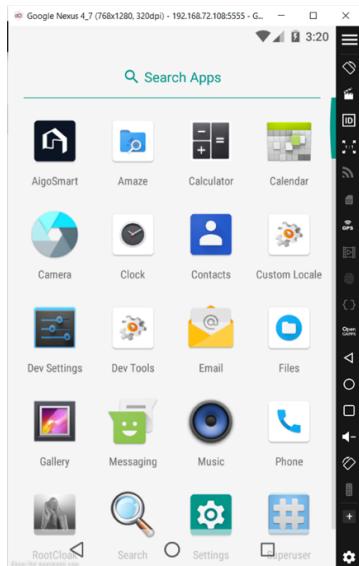
Dopo aver confermato le impostazioni, il dispositivo verrà installato, scaricando la versione specifica di Android selezionata, e sarà pronto per l'uso.



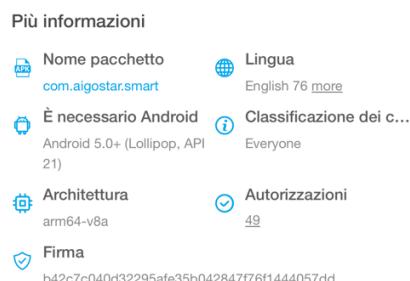
2.3.4 INSTALLAZIONE APP SU EMULATORE

Dopo aver **collegato** con successo **la macchina virtuale al dispositivo emulato**, il passo successivo è installare l'applicazione desiderata sul nostro emulatore.

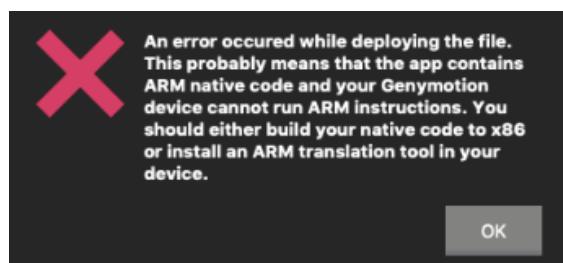
Per fare ciò, si è provveduto a scaricare **l'APK dell'app da fonti di terze parti** direttamente sulla macchina virtuale o sulla macchina host.



Nel caso di studio, l'applicazione girava su un'architettura **arm64-v8a** non accettata dall'emulatore poiché Genymotion supporta solo architettura **x86** e non architettura ARM.



Questo ha reso necessaria la ricerca di una soluzione alternativa per eseguire correttamente l'applicazione nel nostro ambiente virtuale.



Per risolvere questo inconveniente, è stato necessario applicare al nostro ambiente virtuale una soluzione nota come **Genymotion_ARM_Translation**.

Questo tool, disponibile su [GitHub](#), consente di eseguire applicazioni Android con architettura ARM su emulatori Genymotion che supportano solo architettura **x86**.

2.3.5 ADB

Una volta avviata sia la macchina virtuale che il dispositivo emulato, è necessario stabilire una **connessione** tra di essi per consentire la comunicazione.

Per fare ciò, è stato utilizzato **ADB (Android Debug Bridge)**.

Per iniziare la connessione è necessario individuare **l'indirizzo IP** del dispositivo emulato, che di solito è indicato nel **titolo** della finestra in cui viene emulato.

Nel caso di studio l'indirizzo IP equivale a **192.168.72.108:5555**.

Per connettere il dispositivo sarà necessario digitare nel **terminale della macchina virtuale** il comando `adb connect 192.168.72.108:5555`.

Questo comando informa il tool adb dell'indirizzo IP del dispositivo a cui desideriamo connetterci e ci restituirà una risposta positiva come output, confermando che la connessione è stata stabilita con successo.

Per recuperare il pacchetto dell'applicazione utilizzando **adb** su **Tamer**

1. *adb shell*
2. *pm list packages* (*per vedere la lista delle app installate*)
3. *individua nome pacchetto che ti serve*
4. */data/data/"nome pacchetto"* (*per entrare nella directory della applicazione*)
5. *ls* (*elenca cartelle nella directory*)
6. *apri altro terminale*
7. *adb pull /data/data/"tua app"* (*per scaricare il pacchetto dell'app*).

Per estrarre il codice sorgente sono stati utilizzati due tool differenti:

- **Maraframework:** → Le sue funzionalità includono:
 1. Decompilazione di file APK in classi Java
 2. Deobfuscation di APK protetti
 3. Analisi dettagliata di APK
 4. Scansione degli APK
 5. Analisi del file AndroidManifest.xml

The screenshot shows a terminal window with the following content:

```
root@kali:~/MARA_Framework# ./mara.sh
Home

[ M A R A F R A M E W O R K ]

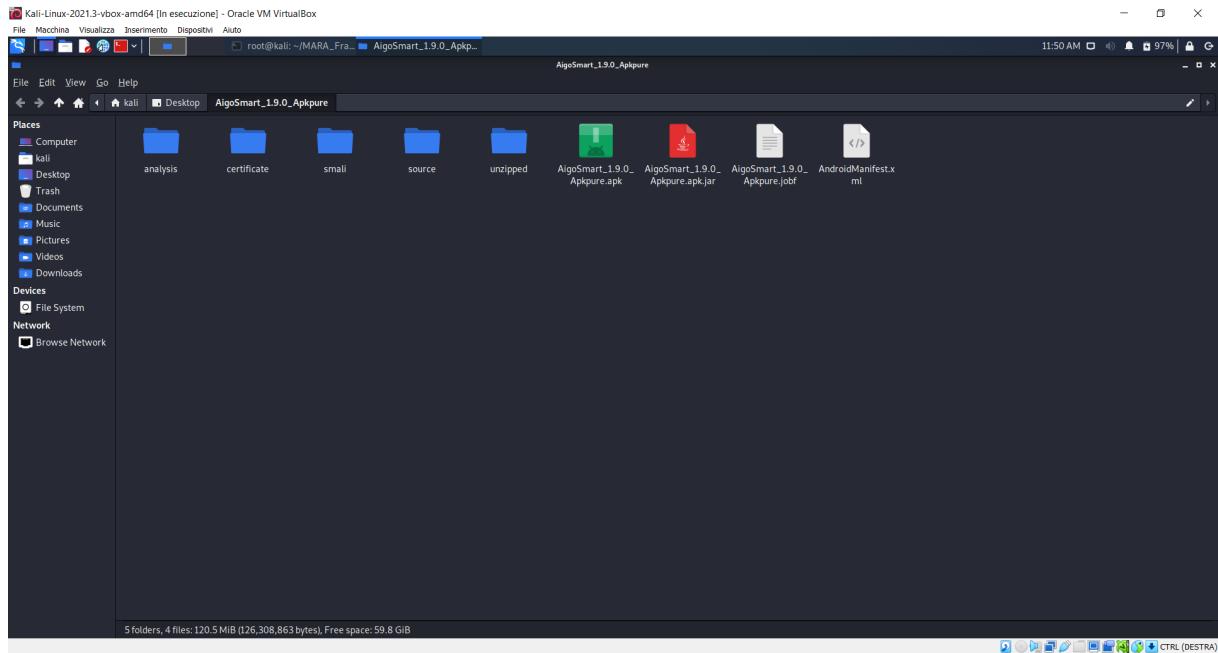
[M]obile [A]pplication [R]everse Engineering & [A]nalysis Framework
version: 0.2.2 beta
Developed by: Christian Kisutsa and Chrispus Kamau
URL: https://github.com/xtiankisutsa/MARA_Framework

Usage:
./mara.sh [options] <path> (.apk, .dex, .jar or .class)

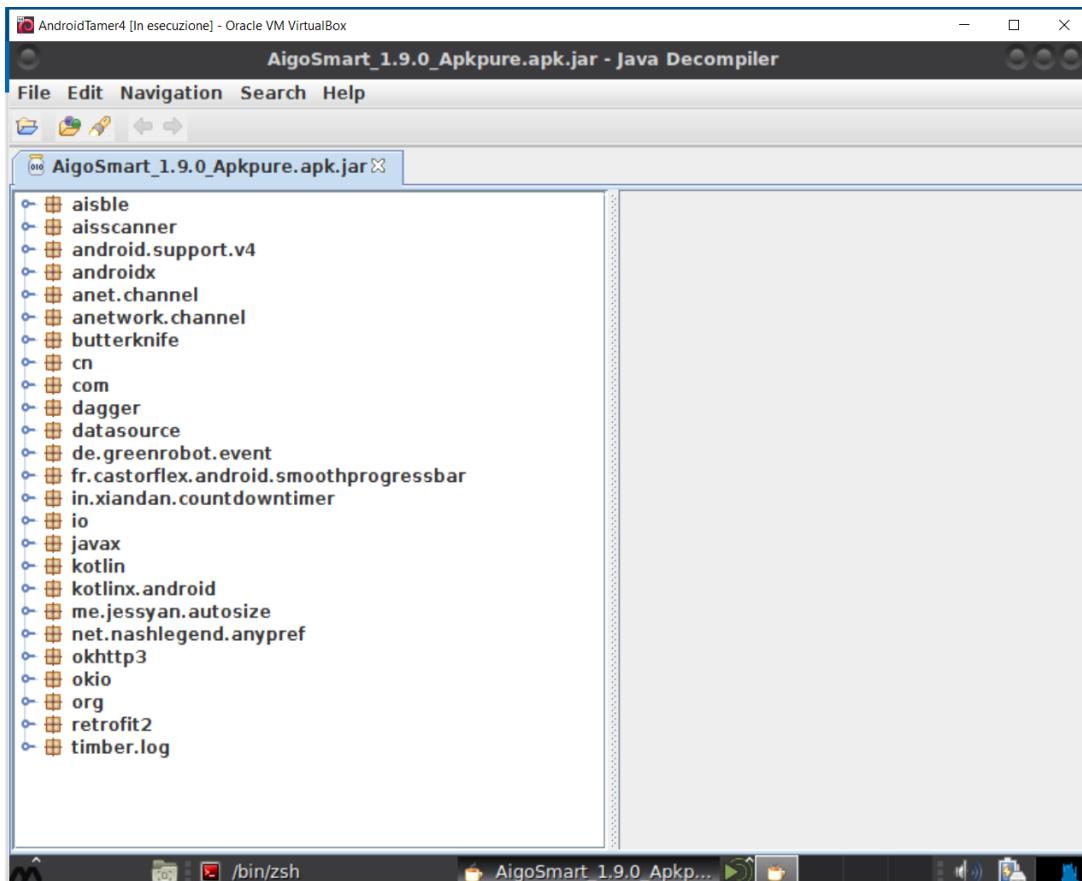
Options:
-s, --apk      - analyze apk file
-d, --dex      - analyze dex file
-j, --jar      - analyze jar file
-c, --class    - analyze class file
-m, --multiple-apk - analyze multiple apk files
-x, --multiple-dex - analyze multiple dex files
-r, --multiple-jar - analyze multiple jar files
-h, --help      - print this help

Example:
apk file analysis e.g ./mara.sh -s </path/to/apk/file>
dex file analysis e.g ./mara.sh -d </path/to/dex/file>
jar file analysis e.g ./mara.sh -j </path/to/jar/file>
class file analysis e.g ./mara.sh -c </path/to/class/file/>
multiple apk analysis e.g ./mara.sh -m </path/to/apk/folder/>
multiple dex analysis e.g ./mara.sh -x </path/to/dex/folder/>
multiple jar analysis e.g ./mara.sh -r </path/to/jar/folder/>
```

Nel caso di studio, questo tool è stato utilizzato per la **decompilazione** di file apk in classi java.



- **JD-GUI:** ottenuto il file .jar dal tool **Maraframework**, si è successivamente passato alla fase di **decompilazione** che permette l'accesso e la visualizzazione del codice Java delle classi presenti.



Per agevolare l'analisi, il codice è stato esaminato tramite l'utilizzo di **Visual Studio Code**.

3 PENETRATION TESTING

Per procedere con l'analisi della nostra app e quindi eseguire il Penetration Testing nell'ambiente di test, rincorreremo a due tipologie di analisi:

1. **Analisi automatica:** verranno utilizzate strumenti di **analisi automatica delle app**, i quali generano **un report delle vulnerabilità rilevate**. In questo modo otteniamo una visione generale delle vulnerabilità presenti nell'applicazione.
Tuttavia, è importante sottolineare che questi strumenti forniscono solo un **supporto iniziale**, poiché le vulnerabilità verranno analizzate in modo più dettagliato **manualmente**.
Questo approccio è essenziale poiché gli strumenti automatizzati possono generare un elevato numero di **falsi positivi**; quindi, non sono in grado di sostituire completamente l'analisi manuale. La **valutazione umana** è cruciale per comprendere appieno il contesto dell'applicazione e individuare eventuali vulnerabilità che potrebbero sfuggire agli strumenti automatizzati.
2. **Analisi manuale + MASVS:** verrà condotta un'analisi basata sulle nostre competenze, agendo come Pen Tester e utilizzando la nostra creatività e ingegno per ideare possibili scenari di attacco e test di abuso.
Questa fase comprenderà **l'esame** del codice sorgente, dei file presenti nella directory di installazione dell'app sul dispositivo e dei messaggi registrati nel logcat, insieme all'uso di strumenti specifici precedentemente menzionati.
Inoltre, verrà condotta un'analisi dell'applicazione considerando i requisiti del Mobile App Security Verification Standard (MASVS) e seguendo le linee guida del Mobile Application Security Testing Guide (MASTG), che indicano i test da eseguire per soddisfare ciascun requisito.

Questo approccio combinato ci consentirà di identificare e affrontare efficacemente le vulnerabilità presenti nell'applicazione.

3.1 ANALISI AUTOMATICA

Per condurre un'analisi automatica dell'applicazione in questione, sono stati impiegati i due strumenti di **analisi automatica** menzionati nella sezione precedente, **ImmuniWeb** e MobSF. Questo è stato fatto per individuare eventuali vulnerabilità aggiuntive non rilevate tramite l'analisi manuale.

3.1.1 IMMUNIWEB

ImmuniWeb è un servizio Web che offre un tool per l'analisi automatizzata delle applicazioni mobili. Esso genera un report dettagliato delle vulnerabilità individuate, classificate in quattro livelli di gravità e organizzate secondo le 10 categorie di vulnerabilità di OWASP. Inoltre, ImmuniWeb analizza e categorizza i permessi utilizzati dall'applicazione.

Caricato l'apk è stato ottenuto il **report delle vulnerabilità**.

Summary of Mobile Application Security Test
This application was tested 2 times during the last 12 months.

AigoSmart

	App version 1.9.0 App ID com.aigostar.smart Device type android Test started Dec 5th, 2023 09:34:19 GMT+1 Test finished Dec 5th, 2023 10:29:57 GMT+1 Test runtime 56 minutes APK source User upload	 Remove test	 PDF Download report
--	--	---	---

Mobile App Permissions and Privacy 36 PERMISSIONS **OWASP Mobile Top 10 Security Test** 7 MAJOR RISKS FOUND **Mobile App External Communications** 102 MAJOR RISKS FOUND **Software Composition Analysis** 73 COMPONENTS FOUND

Nel report è incluso anche il **rilevamento di fallo e debolezze** che potrebbero avere un impatto sull'app e che il test automatico ha riscontrato.

Alcune di tali **vulnerabilità** sono le seguenti:

Test di sicurezza Top 10 OWASP Mobile

L'audit automatico ha rivelato le seguenti fallo di sicurezza e punti deboli che possono avere un impatto sull'applicazione:

1 CRITICAL RISK	2 ALTO RISCHIO	5 MEDIUM RISK	6 BASSO RISCHIO	9 AVVERTENZA
---------------------------	--------------------------	-------------------------	---------------------------	------------------------

POSSIBLE MAN-IN-THE-MIDDLE ATTACK [M3] [CWE-297] CRITICAL

EXTERNAL DATA IN SQL QUERIES [M7] [CWE-89] HIGH

USAGE OF UNENCRYPTED HTTP PROTOCOL [M3] [CWE-319] HIGH

3.1.2 MOBSF

MobSF è uno strumento ampiamente utilizzato per l'analisi automatica delle applicazioni mobili. Esistono sia una versione online che una versione locale, quest'ultima usata per analizzare l'app del caso di studio, entrambe accessibili tramite browser e consentono di caricare il file APK da analizzare.

Oltre a fornire un dettagliato rapporto sulle vulnerabilità individuate e la loro gravità, MobSF assegna un **punteggio sulla sicurezza** dell'applicazione, valutato su una scala da 0 a 100.

Nel caso dell'app presa in esame il tool ha dato come risultato:

File Name: AigoSmart_1.9.0_Apkpure.apk

Package Name: com.aigostar.smart

Scan Date: Dec. 9, 2023, 11:07 a.m.

App Security Score: **26/100 (CRITICAL RISK)**

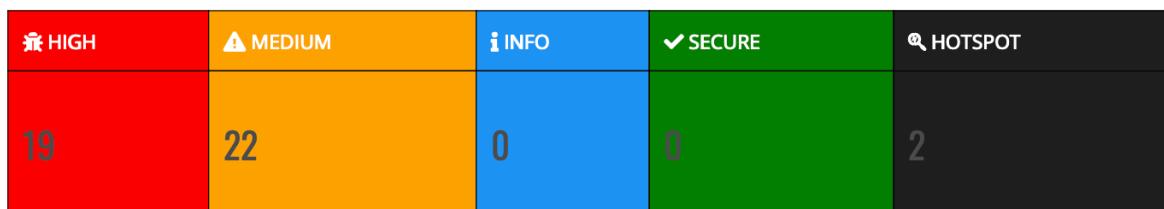
Grade:



Trackers Detection:

4/433

FINDINGS SEVERITY



FILE INFORMATION

File Name: AigoSmart_1.9.0_Apkpure.apk
Size: 43.93MB
MD5: 217d2fc2fabece3712af1cded6f6b68a
SHA1: 00fee6ee0022e075a5fcdf3e7b5a1230fdb3b792
SHA256: 6cbd0b278a01e7046a1026629e230e2141c20e04b7c47e8300958f3db4db8448

APP INFORMATION

App Name: AigoSmart
Package Name: com.aigostar.smart
Main Activity: com.aigostar.module.common.module.main.SplashActivity
Target SDK: 30
Min SDK: 21
Max SDK:
Android Version Name: 1.9.0

3.2 MASVS

Nel presente paragrafo sono registrati i controlli eseguiti secondo gli standard **dell'OWASP Mobile Application Verification Standard (MASVS)**, il punto di riferimento per la sicurezza delle app mobili nel settore.

Ogni test mira a verificare specifiche proprietà definite nei codici **MASVS** e **MSTG**.

Nel contesto dell'analisi dei controlli OWASP, viene delineata una serie di requisiti fondamentali che sottolineano l'importanza della sicurezza nelle fasi di progettazione e sviluppo delle applicazioni mobili.

- **MASVS-V1: Requisiti di Archiviazione Dati e Privacy [STORAGE]**

Proteggere i dati sensibili è cruciale in ambito mobile. I dati possono essere esposti ad altre app sullo stesso dispositivo o possono fuoriuscire su archivi cloud, backup o cache della tastiera. Ulteriori protezioni sono necessarie per prevenire l'accesso non autorizzato in caso di perdita o furto del dispositivo.

- **MASVS-V2: Requisiti Crittografici [CRYPTO]**

L'uso corretto della crittografia è essenziale per proteggere i dati archiviati sui dispositivi mobili. I controlli in questa sezione mirano a garantire l'implementazione corretta delle migliori pratiche del settore.

- **MASVS-V3: Requisiti di Autenticazione e Gestione Sessione [AUTH]**

Sono definiti requisiti di base per la gestione degli account utente e delle sessioni, anche se la maggior parte della logica avviene all'endpoint.

- **MASVS-V4: Requisiti di Comunicazione di Rete [NETWORK]**

L'obiettivo è garantire la riservatezza e l'integrità delle informazioni scambiate tra l'app mobile e i servizi remoti, inclusa l'impostazione di canali di comunicazione sicuri e l'utilizzo del protocollo TLS.

- **MASVS-V5: Requisiti di Interazione con la Piattaforma [PLATFORM]**

Questi controlli garantiscono l'uso sicuro delle API di piattaforma e dei componenti standard, coprendo anche la comunicazione tra le app.

- **MASVS-V6: Requisiti di Qualità del Codice e Impostazioni di Build [CODE]**

Si assicura che vengano seguite pratiche di scrittura del codice sicure e che le funzionalità di sicurezza offerte dal compilatore siano attivate.

- **MASVS-V7: Requisiti di Resilienza [RESILIENCE]**

Si raccomandano misure di difesa in profondità per aumentare la resilienza dell'app contro il reverse engineering e specifici attacchi lato client.

In sintesi, questi controlli delineano una serie di requisiti cruciali che devono essere integrati nel processo di sviluppo per garantire un livello adeguato di sicurezza nelle applicazioni mobili.

I risultati di ciascun test possono essere categorizzati come segue:

- **ESITO REQUISITO: rispettato**: il requisito è stato soddisfatto.

- **ESITO REQUISITO: non rispettato**: il requisito non è stato soddisfatto.

- **ESITO REQUISITO: N/A**: il requisito non è applicabile al contesto dell'analisi.

I risultati dei test saranno riportati seguendo l'ordine stabilito nel sito dello standard dell'OWASP Mobile Application Verification Standard (MASVS), garantendo coerenza con la struttura e la sequenza presenti nel documento ufficiale.

3.2.1 MASVS-V1: Requisiti di Archiviazione Dati e Privacy [STORAGE]

3.2.1.1 MASTG-TEST-0012 → Verifica dei criteri di sicurezza per l'accesso ai dispositivi

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-1	MSTG-STORAGE-11

ESITO REQUISITO: non rispettato

Dinamicamente, è stato verificato che l'accesso all'app non è moderato da alcun criterio di sicurezza minimo. Infatti, una volta autenticato, l'utente resta connesso all'applicazione, e chiunque abbia accesso al dispositivo può accedere all'applicazione senza dover autenticarsi (ad esempio tramite uso di password o di biometrie).

Inoltre, un parametro da considerare è relativo alla versione di android: l'app, come indicato dal file Manifest.xml e confermato dai tool automatici utilizzati, supporta una versione minima di Android (Min SDK) pari a 21. Questo significa che l'applicazione può essere installata e eseguita su dispositivi con Android 5.0 (Lollipop).

3.2.1.2 MASTG-TEST-0001 → Test dell'archiviazione locale per i dati sensibili

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-1	MSTG-STORAGE-1
	MSTG-STORAGE-2

ESITO REQUISITO: non rispettato

Nell'applicazione in esame, questo requisito non è verificato dato che nel Manifest è presente il permesso "WRITE_EXTERNAL_STORAGE" che permette di archiviare in una memoria esterna dati relativi all'applicazione.

Inoltre, eseguendo un'analisi dinamica dell'applicazione, risultano facilmente accessibili e privi di algoritmi critografici i dati sensibili quale i file relativi ai database e le preferenze condivise memorizzate come file XML (in /data/data/<nome-pacchetto>/shared_prefs).

```
UserInfo.xml (~/.shared_prefs) - Pluma
File Edit View Search Tools Documents Help
Open Save Undo Redo Cut Copy Find Replace
UserInfo.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="birthday">2019-03-12</string>
    <boolean name="email_verified" value="false" />
    <int name="gender" value="1" />
    <string name="iot_identity_id">50150g66d372341d12afe8d5a3ddaa4f4c522833</string>
    <string name="profile"></string>
    <boolean name="phone_number_verified" value="true" />
    <string name="photo">null</string>
    <string name="country_code">IT</string>
    <long name="user_id" value="1502629045119094784" />
    <string name="nickname">389****6277</string>
    <string name="phone_number">39-3894446277</string>
    <string name="email">null</string>
    <int name="external_providers" value="0" />
</map>
```

```
vbox86p:/data/data/com.aigostar.smart # ls
app_SGLib app_breeze app_webview code_cache files no_backup
app_accs app_textures cache databases lib shared_prefs
vbox86p:/data/data/com.aigostar.smart # cd databases
vbox86p:/data/data/com.aigostar.smart/databases # ls
accs.db
accs.db-journal
aigoStar-db
aigoStar-db-journal
com.google.android.datatransport.events
com.google.android.datatransport.events-journal
google_app_measurement.db
google_app_measurement.db-journal
google_app_measurement_local.db
google_app_measurement_local.db-journal
message_accs_db
message_accs_db-journal
```

3.2.1.3 MASTG-TEST-0009 → Test dei backup dei dati sensibili

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-2	MSTG-STORAGE-8

ESITO REQUISITO: rispettato

Verificando il file AndroidManifest.xml il seguente flag android:allowBackup risulta settato come false.

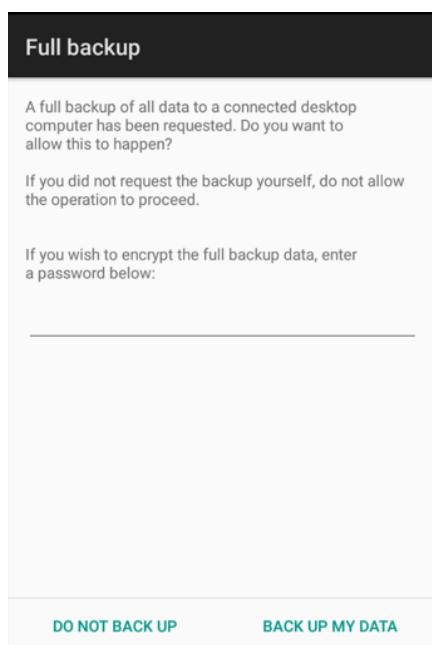
```
<application android:allowBackup="false"  
    android:appComponentFactory="androidx.core.app.CoreComponentFactory"  
    android:icon="@mipmap/ic_launcher" android:label="@string/app_name"  
    android:maxAspectRatio="2.4" android:name="com.aigo.smarthome.AppApplication"  
    android:networkSecurityConfig="@xml/network_security_config"  
    android:requestLegacyExternalStorage="true" android:resizeableActivity="true"  
    android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true"  
    android:theme="@style/AppTheme">
```

Se il valore del flag fosse stato “<true>” l’applicazione avrebbe potuto salvare qualche tipo di dati sensibili.

Riguardo l’analisi dinamica, tramite adb l’applicazione impedisce di eseguire il backup dell’app senza approvazione.

android@tamer ~> adb backup -f aigostar.ab com.aigostar.smart
Now unlock your device and confirm the backup operation.

139



3.2.1.4 MASTG-TEST-0004 → Determinare se i dati sensibili sono condivisi con terze parti tramite servizi integrati

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-2	MSTG-STORAGE-4

ESITO REQUISITO: rispettato

Questo requisito è stato confermato principalmente attraverso i risultati dei test automatici, come quelli prodotti da MobSF.

Dai risultati del test automatici generato da MobSF sono stati riscontrati i seguenti Trackers.

TRACKERS

TRACKER	CATEGORIES	URL
Bugly		https://reports.exodus-privacy.eu.org/trackers/190
Google Firebase Analytics	Analytics	https://reports.exodus-privacy.eu.org/trackers/49
Huawei Mobile Services (HMS) Core	Analytics, Advertisement, Location	https://reports.exodus-privacy.eu.org/trackers/333
OpenTelemetry (OpenCensus, OpenTracing)	Analytics	https://reports.exodus-privacy.eu.org/trackers/412

- **Google Firebase Analytics** = Firebase offre funzionalità come analisi, database, messaggistica e crash reporting. Presente in: 114303 applicazioni.
- **Servizi mobili Huawei (HMS) Core** = HMS Core è una raccolta di strumenti realizzati per i partner e gli sviluppatori di app di Huawei. Presente in: 7714 applicazioni.
- **OpenTelemetria (OpenCensus, OpenTracing)** = raccolta di API, SDK e strumenti. Si usa per strumentare, generare, raccogliere ed esportare dati di telemetria (metriche, log e tracce) per analizzare le prestazioni e il comportamento del software. Presente in: 2872 applicazioni.
- **Bugly** = un'azienda produttrice di software specializzata nell'individuazione e nel tracciamento dei problemi legati ai bug. fornisce rapporti anomali professionali e statistiche operative per gli sviluppatori mobili, aiuta gli sviluppatori a trovare e risolvere rapidamente le anomalie e allo stesso tempo a cogliere le dinamiche operative del prodotto e a seguire il feedback degli utenti in modo tempestivo. Presente in: 2054 applicazioni.

Inoltre, non è stata riscontrata la presenza della classe NotificationManager durante un'analisi manuale.

3.2.1.5 MASTG-TEST-0011 → Test della memoria per i dati sensibili

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-2	MSTG-STORAGE-10

ESITO REQUISITO: non rispettato

Attraverso un'analisi dinamica dell'applicazione i file contenenti dati sensibili quali le credenziali degli utenti vengono salvate in file.

3.2.1.6 MASTG-TEST-0006 → Determinare se la cache della tastiera è disabilitata per i campi di inserimento testo

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-2	MSTG-STORAGE-5

ESITO REQUISITO: rispettato

Dalla guida MASTG apprendiamo che, attraverso un'analisi dinamica dell'applicazione, è possibile notare che quando si tenta di inserire dati sensibili, come la password di accesso all'applicazione, non viene fornito alcun suggerimento di input da parte dell'applicazione.

Inoltre è importante visualizzare la versione minima richiesta dell'SDK, nel caso dell'app selezionata 21, nel manifesto di Android (android:minSdkVersion), poiché deve supportare le costanti utilizzate (ad esempio, per textWebPassword è richiesta la versione 11 dell'SDK di Android). Altrimenti, l'applicazione compilata non rispetterebbe le costanti del tipo di input utilizzate, consentendo la memorizzazione nella cache della tastiera.

3.2.1.7 MASTG-TEST-0005 → Determinare se i dati sensibili sono condivisi con terze parti tramite notifiche

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-2	MSTG-STORAGE-4

ESITO REQUISITO: rispettato

Questo requisito è stato confermato principalmente attraverso i risultati dei test automatici, come quelli prodotti da MobSF, che individuano servizi esterni all'applicazione nella sezione "tracker". Tuttavia, un'analisi del file Manifest ha rivelato che tali servizi esterni non erano inclusi nel pacchetto dell'applicazione. Inoltre, non è stata riscontrata la presenza della classe NotificationManager durante un'analisi manuale.

3.2.1.8 MASTG-TEST-0003 → Verifica dei registri per i dati sensibili

MASVS v2 ID	MASVS v1 IDs
MASVS-STORAGE-2	MSTG-STORAGE-3

ESITO REQUISITO: rispettato

Generalmente, tale requisito consiglia la non presenza di dati sensibili nei log delle applicazioni. Analizzando il codice è emerso che nessuna informazione sensibile venga stampata nei log.

3.2.2 MASVS-V2: Requisiti Crittografici [CRYPTO]

3.2.2.1 MASTG-TEST-0016 → Test della generazione di numeri casuali

MASVS v2 ID	MASVS v1 IDs
MASVS-CRYPTO-1	MASVS-CRYPTO-1

ESITO REQUISITO: rispettato

Il file RandomUtils.java definisce una classe Java chiamata “RandomUtils” che utilizza la classe “SecureRandom” per generare numeri casuali e stringhe casuali.

Esso è considerato un generatore sicuro:

- Utilizzo di SecureRandom: La classe utilizza “SecureRandom”, che è progettata per generare numeri casuali sicuri e adatta per scopi crittografici.
- Costruttore predefinito di SecureRandom: Il costruttore predefinito di “SecureRandom” viene utilizzato senza specificare un seed personalizzato. Questo significa che il generatore utilizza il valore di seed fornito dal sistema, garantendo casualità e sicurezza.
- Generazione di stringhe casuali sicure: Il metodo “getRandomString(int paramInt)” genera stringhe casuali selezionando caratteri casuali dalla stringa ‘RANDOM_BYTE’ utilizzando l’istanza “SecureRandom”, assicurando così la casualità e la sicurezza delle stringhe generate.

In sintesi, il file RandomUtils.java fornisce un generatore sicuro di numeri casuali e stringhe casuali utilizzando la classe ‘SecureRandom’ in modo corretto e senza compromettere la sicurezza.

3.2.2.2 MASTG-TEST-0014 → Verifica della configurazione degli algoritmi standard di crittografia

MASVS v2 ID	MASVS v1 IDs
MASVS-CRYPTO-1	MASVS-CRYPTO-2 MASVS-CRYPTO-3 MASVS-CRYPTO-4

ESITO REQUISITO: rispettato

Questo criterio si basa sull'uso da parte dell'applicazione di implementazioni standard di cifratura o librerie open source ben documentate.

Nel caso dell'applicazione in questione, questo criterio è soddisfatto poiché si utilizzano le librerie standard di Java.

3.2.2.3 MASTG-TEST-0013 → Test della crittografia simmetrica

MASVS v2 ID	MASVS v1 IDs
MASVS-CRYPTO-1	MASVS-CRYPTO-1

ESITO REQUISITO: rispettato

L'obiettivo di questo test è identificare tutti i punti dell'applicazione in cui viene utilizzata la crittografia simmetrica. Ovvero, identificare tutti i punti dove vengono utilizzati algoritmi simmetrici come AES e metodi per la generazione delle chiavi crittografiche. Conseguentemente, verificare se le chiavi simmetriche appena citate presentano vulnerabilità come, ad esempio, il loro utilizzo in modo hard-coded, l'essere legate alle risorse dell'applicazione o derivabili da valori noti. Per quanto riguarda l'applicazione in esame, tale requisito è soddisfatto in quanto gli unici casi in cui è l'utilizzo della crittografia simmetrica sono casi in cui le chiavi sono gestite correttamente.

Il seguente requisito prevede che l'applicativo utilizzi più di un metodo di crittografia e non soltanto la crittografia simmetrica con chiavi di tipo hardcoded come unico sistema crittografico. Il requisito non risulta soddisfatto in quanto, nell'applicativo si fa utilizzo sia della crittografia simmetrica sia di chiavi hardcoded.

L'applicazione non utilizza algoritmi di crittografia simmetrica come unico metodo crittografico per la protezione dei dati; tuttavia, attraverso l'analisi statica del codice, è stata effettuata una ricerca usando le parole chiave 'AES'. Inoltre, è stata individuata l'utilizzo di algoritmi di hashing come MD5 e SHA256.

3.2.2.4 MASTG-TEST-0015 → Verifica degli scopi delle chiavi

MASVS v2 ID	MASVS v1 IDs
MASVS-CRYPTO-2	MASVS-CRYPTO-5

ESITO REQUISITO: non rispettato

Questo requisito si basa sulla verifica che una chiave crittografica non venga riutilizzata per più scopi ed utilizzi diversi. Per quanto riguarda l'applicazione in esame, tale requisito non è rispettato in quanto è possibile identificare con chiarezza le chiavi crittografiche nel codice sorgente, in quanto esse sono hardcoded e sono ricavabili.

3.2.3 MASVS-V3: Requisiti di Autenticazione e Gestione Sessione [AUTH]

3.2.3.1 *MASTG-TEST-0017* → Test di conferma delle credenziali

MASVS v2 ID	MASVS v1 IDs
MASVS-AUTH-2	MSTG-AUTH-1 MSTG-STORAGE-11

ESITO REQUISITO: rispettato

Nel caso dell'applicazione in esame, forma di autenticazione testata nell'ambiente virtuale in cui il dispositivo è stato emulato, è l'uso di una password associata a un numero di telefono e/o indirizzo e-mail.

3.2.3.2 *MASTG-TEST-0018* → Test dell'autenticazione biometrica

MASVS v2 ID	MASVS v1 IDs
MASVS-AUTH-2	MSTG-AUTH-8

ESITO REQUISITO: non rispettato

L'applicazione consente l'accesso esclusivamente tramite credenziali di accesso e non supporta alcuna forma di autenticazione biometrica. Pertanto, il requisito di sicurezza relativo all'autenticazione biometrica non è applicabile.

3.2.4 MASVS-V4: Requisiti di Comunicazione di Rete [NETWORK]

3.2.4.1 MASTG-TEST-0021 → Test della verifica dell'identificazione dell'endpoint

MASVS v2 ID	MASVS v1 IDs
MASVS-NETWORK-1	MSTG-NETWORK-3

ESITO REQUISITO: rispettato

Il requisito in questione riguarda la verifica del certificato X.509 dell'endpoint remoto quando viene stabilito il canale sicuro. Si accettano esclusivamente certificati firmati da una Certificate Authority (CA) nota ed affidabile. Tale requisito risulta soddisfatto poiché l'analisi NIAP ha confermato che l'applicazione utilizza correttamente il certificato X.509.

Inoltre, tramite il tool MobSF siamo in grado di identificare il Subject Distinguished Name (DN) di un certificato X.509 del certificato X.509 utilizzato identificando l'entità a cui appartiene il certificato "X.509 Subject: C=CN, ST=ZheJiang, L=Hangzhou, O=tuya, OU=www.tuya.com, CN=www.tuya.com"

Ecco cosa significa ciascun campo:

- C: Indica il Paese (Country). In questo caso, "C=CN" sta per "Cina".
- ST: Sta per la provincia o lo Stato (State). "ST=ZheJiang" indica che si trova nella provincia di Zhejiang.
- L: Indica la città o la località (Locality). "L=Hangzhou" specifica che si trova a Hangzhou.
- O: Rappresenta l'organizzazione (Organization). "O=tuya" indica che l'organizzazione è "tuya".
- OU: Sta per Organizational Unit. "OU=www.tuya.com" specifica che si tratta dell'unità organizzativa "www.tuya.com".
- CN: Comune Name, ovvero il nome comune. "CN=www.tuya.com" indica che il nome comune è "www.tuya.com".

In sintesi, questo Subject DN identifica un certificato associato al sito web www.tuya.com, di proprietà dell'organizzazione Tuya, situato a Hangzhou, nella provincia di Zhejiang, in Cina.

CERTIFICATE INFORMATION

```
Binary is signed
v1 signature: True
v2 signature: True
v3 signature: True
v4 signature: False
X.509 Subject: C=CN, ST=ZheJiang, L=Hangzhou, O=tuya, OU=www.tuya.com, CN=www.tuya.com
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2018-11-27 07:04:51+00:00
Valid To: 2073-08-30 07:04:51+00:00
Issuer: C=CN, ST=ZheJiang, L=Hangzhou, O=tuya, OU=www.tuya.com, CN=www.tuya.com
Serial Number: 0x22f7fed4
Hash Algorithm: sha256
md5: 5737fa36e9e74ac5035a98af536b6ccf
sha1: b42c7c040d32295afe35b042847f76f1444057dd
sha256: 8b4f4903324878aa284c06824097e875667d0598a1b1bc7dce11f81c643fd117
sha512: a7447b2aa059764dae2828f2f53ed649e7a05c987733c48ffa3325d4c0c61ce6df7df9252d38350dab739a941acd85ceb4c620c6fd0fbf15881cce3f867d75c6
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: f9e7bf933639dbae8f41233843916b0d9b82ea4fb838cef75adadde8d5908b0d
Found 1 unique certificates
```

Inoltre, le applicazioni che puntano ad Android 7.0 (livello API 24) o superiore utilizzeranno una configurazione di sicurezza di rete predefinita che non si affida a nessuna CA fornita dall'utente, riducendo la possibilità di attacchi MITM attirando gli utenti a installare CA dannose. L'applicazione presa in esame richiede un targetSdkVersion di livello 30.

3.2.4.2 MASTG-TEST-0023 → Test del provider di sicurezza

MASVS v2 ID	MASVS v1 IDs
MASVS-NETWORK-1	MSTG-NETWORK-6

ESITO REQUISITO: rispettato

L'inserimento di questa versione dell'applicazione sul PlayStore e in altri Store ufficiali è sufficiente alla verifica del requisito.

3.2.4.3 MASTG-TEST-0020 → Test delle impostazioni TLS

MASVS v2 ID	MASVS v1 IDs
MASVS-NETWORK-1	MSTG-NETWORK-2

ESITO REQUISITO: non rispettato

Come Riportato anche dal tool Immuniweb, il codice analizzato è insicuro perché utilizza il verificatore di nomi host ALLOW_ALL_HOSTNAME_VERIFIER. Questo verificatore accetta qualsiasi nome host, indipendentemente dal fatto che corrisponda effettivamente al certificato SSL/TLS del server con cui si sta comunicando. Ciò significa che un aggressore che esegue un attacco MITM può intercettare il traffico tra l'app mobile e il server e impersonare il server.

```
public static final HostnameVerifier ALLOW_ALL_HOSTNAME_VERIFIER;

if (!paramBoolean)
{
    b.a(b.ALLOW_ALL_HOSTNAME_VERIFIER);
    localSSLSocketFactory = b.TRUST_ALL_SSL_SOCKET_FACTORY;}
```

Per proteggersi dagli attacchi MITM, è importante utilizzare un verificatore di nomi host verificato e assicurarsi che il sistema operativo e il software siano aggiornati all'ultima versione.

3.2.4.4 MASTG-TEST-0019 → Test della crittografia dei dati sulla rete

MASVS v2 ID	MASVS v1 IDs
MASVS-NETWORK-1	MSTG-NETWORK-1

ESITO REQUISITO: non rispettato

L'applicazione mobile utilizza il protocollo HTTP per inviare o ricevere dati. Il design del protocollo HTTP non prevede alcuna crittografia dei dati trasmessi e può essere facilmente intercettato se un aggressore si trova nella stessa rete o ha accesso al canale dati della vittima.

NETWORK SECURITY

HIGH:1 | WARNING: 0 | INFO: 0 | SECURE: 0

NO	SCOPE	SEVERITY	DESCRIPTION
1	*	high	Base config is insecurely configured to permit clear text traffic to all domains.

3.2.4.5 MASTG-TEST-0022 → Test dei depositi di certificati personalizzati e del pinning dei certificati

MASVS v2 ID	MASVS v1 IDs
MASVS-NETWORK-2	MSTG-NETWORK-4

ESITO REQUISITO: rispettato

Questo requisito si basa sul controllo dei certificati o chiavi pubbliche del server con cui comunicare da parte dell'app per verificare se siano gli stessi di cui ha effettuato il pinning.

Per quanto riguarda la nostra app, questo requisito è rispettato.

3.2.5 MASVS-V5: Requisiti di Interazione con la Piattaforma [PLATFORM]

3.2.5.1 MASTG-TEST-0030 → Verifica dell'implementazione vulnerabile di PendingIntent

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-1	MSTG-PLATFORM-4

ESITO REQUISITO: non rispettato

Analizzando il manifest.xml è stato trovato vari tag android:exported="true" associati a molti Receivers. Come riportato su MobSF:

14	Broadcast Receiver (com.google.firebaseio.iid.FirebaseInstanceIdReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: com.google.android.c2dm.permission.SEND [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
16	Broadcast Receiver (com.xiaomi.push.service.receivers.NetworkStatusReceiver) is not Protected. [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
17	Broadcast Receiver (com.alibaba.sdk.android.push.MiPushBroadcastReceiver) is not Protected. [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
18	Broadcast Receiver (com.alibaba.sdk.android.push.huawei.HuaweiPushReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
21	Broadcast Receiver (com.alibaba.sdk.android.push.MeizuPushReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
23	Broadcast Receiver (com.alibaba.sdk.android.push.impl.PushMessageReceiverImpl) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
26	Broadcast Receiver (androidx.work.impl.diagnostics.DiagnosticsReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.DUMP [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
27	Broadcast Receiver (com.instacart.library.truetime.BootCompletedBroadcastReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
35	Broadcast Receiver (com.taobao.agoo.AgooCommandReceiver) is not Protected. [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.

37	Broadcast Receiver (com.alibaba.sdk.android.push.SystemEventReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
31	Broadcast Receiver (com.taobao.accs.EventReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
32	Broadcast Receiver (com.taobao.accs.ServiceReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

3.2.5.2 MASTG-TEST-0007 → Determinazione dell'esposizione di dati memorizzati sensibili tramite meccanismi IPC

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-1	MSTG-STORAGE-6

ESITO REQUISITO: rispettato

L'analisi dei Content Providers esposti dall'applicazione è fondamentale per garantire la sicurezza dei dati archiviati. Il primo passo è esaminare il file AndroidManifest.xml per individuare i fornitori di contenuti tramite l'elemento <provider>. È essenziale verificare se il valore del tag export (android:exported) è impostato su "true" o "false". Se è "true", i dati potrebbero essere accessibili anche da altre applicazioni, quindi è importante definire adeguati permessi di lettura/scrittura.

Nel Manifest tutti i Provider identificati hanno il tag "android:exported" è settato a false perciò nessun dato sensibile viene esposto tramite meccanismi IPC." creami un testo unico di qualche riga

3.2.5.3 MASTG-TEST-0024 → Test dei permessi delle app

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-1	MSTG-PLATFORM-1

ESITO REQUISITO: non rispettato

Attraverso l'analisi del file AndroidManifest.xml è possibile identificare i permessi richiesti dall'applicazione. Utilizzando il tool MobSF, è stato possibile rilevare e organizzare tali permessi in una tabella, che riporta sia il nome del permesso che il relativo livello di protezione (normal, dangerous, signature). La tabella comprende 12 righe, ma è importante notare che alcuni permessi sono associati a un livello di protezione "unknown". Questi permessi potrebbero essere invalidi, indicando la presenza di permessi inutilizzati nel Manifest o che non sono stati riconosciuti correttamente dal tool di analisi.

PERMISSION	STATUS
android.permission.READ_LOG_S	unknown
android.permission.READ_SETTINGS	unknown
android.permission.RECEIVE_USER_PRESENT	unknown
android.permission.WRITE_MEDIA_STORAGE	unknown
com.aigostar.smart.permission.C2D_MESSAGE	unknown
com.aigostar.smart.permission.JPUSH_MESSAGE	unknown
com.aigostar.smart.permission.MIPUSH_RECEIVE	unknown
com.aigostar.smart.push.permission.MESSAGE	unknown
com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE	unknown
com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE	unknown
com.meizu.c2dm.permission.RECEIVE	unknown
com.meizu.flyme.push.permission.RECEIVE	unknown

3.2.5.4 MASTG-TEST-0028 → Test dei collegamenti profondi

MASVS v2 ID	MASVS v1 IDs

MASVS-PLATFORM-1	MSTG-PLATFORM-3
------------------	-----------------

ESITO REQUISITO: non rispettato

Secondo la guida ufficiale Android, gli URL specializzati, noti come deep link, consentono di indirizzare gli utenti verso contenuti specifici all'interno di un'applicazione. Tuttavia, sorge il rischio di collisioni poiché altre applicazioni potrebbero gestire lo stesso Intent a cui il link fa riferimento. Un deep link è definito attraverso il tag <intent-filter> nel file AndroidManifest.xml, che include informazioni come lo schema, l'host di riferimento e il package. Durante l'analisi del manifest, è stato individuato un corrispondente Intent Filter contenente l'attributo android:scheme. Questo indica che l'applicazione espone funzionalità tramite schemi URL personalizzati.

È importante gestire attentamente l'utilizzo dei deep link per evitare conflitti con altre app e garantire un'esperienza utente fluida e senza interruzioni.

3.2.5.5 MASTG-TEST-0032 → Test dei gestori del protocollo WebView

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-2	MSTG-PLATFORM-6

ESITO REQUISITO: non rispettato

La guida consiglia di cercare le funzioni:

- setAllowContentAccess,
- setAllowFileAccess,
- setAllow FileAccessFromFileURLs,
- setAllowUniversalAccessFromFileURLs.

La ricerca ha riscontrato un match con la funzione setAllowFileAccess impostato a 1 quindi a true. Il requisito dunque non è rispettato.

3.2.5.6 MASTG-TEST-0037 → Verifica della pulizia delle WebView

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-2	MSTG-PLATFORM-10

ESITO REQUISITO: rispettato

Durante l'analisi, sono stati individuati tutti i metodi relativi alla gestione dei WebViews, come setDomStorageEnabled, setAppCacheEnabled e setDatabaseEnabled. È stato valutato attentamente se l'applicazione gestisce correttamente l'eliminazione dei dati del WebView e se segue le best practices raccomandate dalla guida Android.

3.2.5.7 MASTG-TEST-0031 → Verifica dell'esecuzione di JavaScript nelle WebView

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-2	MSTG-PLATFORM-5

ESITO REQUISITO: non rispettato

L'applicazione utilizza WebView, abilitando il JavaScript attraverso la funzione setJavaScriptEnabled. Tuttavia, questa pratica non rispetta i requisiti di sicurezza raccomandati. Il codice mostra che il JavaScript viene abilitato in modo esplicito, senza garantire un canale di comunicazione sicuro tramite HTTPS su TLS per l'interazione con gli endpoint. Secondo le linee guida di sicurezza OWASP, è fondamentale che la comunicazione avvenga esclusivamente su canali protetti per garantire la sicurezza dei dati scambiati.

3.2.5.8 MASTG-TEST-0033 → Test degli oggetti Java esposti attraverso le WebView

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-2	MSTG-PLATFORM-7

ESITO REQUISITO: rispettato

Tale requisito è soddisfatto, in quanto viene utilizzato il metodo addJavascriptInterface quando deve essere rindirizzato il codice JavaScript ed il codice JavaScript non viene caricato dall'endpoint remoto.

3.2.5.9 MASTG-TEST-0010 → Individuazione di informazioni sensibili nelle schermate generate automaticamente

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-3	MSTG-STORAGE-9

ESITO REQUISITO: non rispettato

Questo requisito si basa sulla protezione dell'app dagli attacchi di tipo overlay.

Per quanto riguarda la nostra app, questo requisito non è soddisfatto in quanto eseguendo un'analisi dinamica la finestra non viene oscurata nella fase di immissione di dati sensibili come nel caso delle credenziali di accesso.

3.2.5.10 MASTG-TEST-0008 → Verifica della divulgazione di dati sensibili attraverso l'interfaccia utente

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-2	MSTG-STORAGE-7

ESITO REQUISITO: rispettato

Dall'analisi dinamica dell'applicazione, è evidente che durante le fasi di registrazione e accesso, il campo della password è adeguatamente mascherato nell'interfaccia utente. Questo soddisfa il requisito di sicurezza relativo alla protezione della password durante l'immissione.

3.2.5.11 MASTG-TEST-0035 → Verifica di attacchi overlay

MASVS v2 ID	MASVS v1 IDs
MASVS-PLATFORM-2	MSTG-PLATFORM-7

ESITO REQUISITO: parzialmente rispettato

Attacchi di sovrapposizione dello schermo sono difficili da prevenire in versioni dell'SDK inferiori alla versione 25. Tuttavia, questa vulnerabilità è stata completamente risolta per versioni dell'API successive alla 25.

Poiché l'app studiata ha un target SDK di 30 e un minimo SDK di 21, la vulnerabilità degli attacchi di sovrapposizione dello schermo dovrebbe essere completamente risolta. Tuttavia, è importante notare che le versioni dell'API successive alla 25 includono miglioramenti significativi nella prevenzione di tali attacchi, fornendo un livello più elevato di sicurezza.

3.2.6 MASVS-V6: Requisiti di Qualità del Codice e Impostazioni di Build [CODE]

3.2.6.1 MASTG-TEST-0036 → Verifica dell'aggiornamento forzato

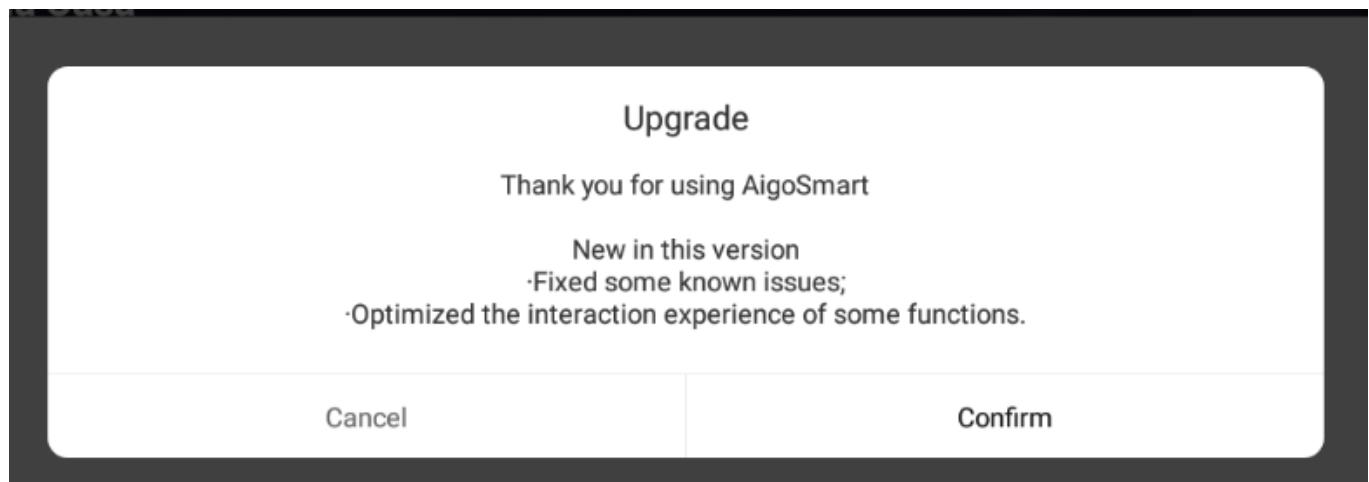
MASVS v2 ID	MASVS v1 IDs
MASVS-CODE-2	MSTG-ARCH-9

ESITO REQUISITO: non rispettato

Secondo la guida, nell'analisi dinamica, è importante testare l'aggiornamento corretto dell'applicazione.

La guida consiglia di scaricare una versione precedente dell'applicazione con una vulnerabilità di sicurezza, sia tramite una release dagli sviluppatori o utilizzando un'app store di terze parti.

Successivamente, verificare se si può continuare a utilizzare l'applicazione senza aggiornarla. Se viene mostrato un avviso di aggiornamento, verifica se puoi comunque utilizzare l'applicazione annullando l'avviso o aggirandolo attraverso un normale utilizzo dell'applicazione. Ciò include la convalida se il backend interrompe le chiamate ai backends vulnerabili e/o se la versione dell'app vulnerabile è bloccata dal backend. Infine, prova a manipolare il numero di versione di un'app compromessa e osserva come il backend risponde a questo (e se viene registrato in qualche modo). L'app presa in analisi avvisa l'utente tramite riquadri che la versione dell'applicazione risulta obsoleta ma permette all'utente di chiudere l'avvertimento e continuare ad usare l'app.



3.2.6.2 MASTG-TEST-0042 → Verifica delle debolezze nelle librerie di terze parti

MASVS v2 ID	MASVS v1 IDs
MASVS-CODE-3	MSTG -CODE-5

ESITO REQUISITO: non rispettato

L'analisi statica generata dallo strumento MobSF evidenzia le librerie utilizzate nell'applicazione insieme alle relative vulnerabilità.

Qui riportate alcune che presentano vulnerabilità.

2	lib/armeabi-v7a/libtnet-3.1.14.so	True info The binary has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.	False high This binary does not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return address. Use the option -fstack-protector-all to enable stack canaries. Not applicable for Dart/Flutter libraries unless Dart FFI is used.	None info The binary does not have run-time search path or RPATH set.	None info The binary does not have RUNPATH set.	False warning The binary does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets etc. Use the compiler option -D_FORTIFY_SOURCE=2 to fortify functions. This check is not applicable for Dart/Flutter libraries.	True info Symbols are stripped.
11	lib/arm64-v8a/libtnet-3.1.14.so	True info The binary has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.	False high This binary does not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return address. Use the option -fstack-protector-all to enable stack canaries. Not applicable for Dart/Flutter libraries unless Dart FFI is used.	None info The binary does not have run-time search path or RPATH set.	None info The binary does not have RUNPATH set.	False warning The binary does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets etc. Use the compiler option -D_FORTIFY_SOURCE=2 to fortify functions. This check is not applicable for Dart/Flutter libraries.	True info Symbols are stripped.
12	lib/arm64-v8a/liblvffmpeg.so	True info The binary has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.	False high This binary does not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return address. Use the option -fstack-protector-all to enable stack canaries. Not applicable for Dart/Flutter libraries unless Dart FFI is used.	None info The binary does not have run-time search path or RPATH set.	None info The binary does not have RUNPATH set.	False warning The binary does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets etc. Use the compiler option -D_FORTIFY_SOURCE=2 to fortify functions. This check is not applicable for Dart/Flutter libraries.	True info Symbols are stripped.

3.2.6.3 MASTG-TEST-0026 → Verifica degli intenti impliciti

3.2.6.4 MASTG-TEST-0002 → Test della memorizzazione locale per la convalida degli input

3.2.6.5 MASTG-TEST-0025 → Test per le falle di iniezione

3.2.6.6 MASTG-TEST-0027 → Test per il caricamento di URL in WebViews

MASVS v2 ID	MASVS v1 IDs
MASVS-CODE-4	MSTG-PLATFORM-2

ESITO REQUISITO: non rispettato

L'inclusione di input in query SQL grezze può potenzialmente portare a una vulnerabilità locale di tipo SQL injection nell'applicazione mobile, con conseguente compromissione di qualsiasi informazione sensibile memorizzata all'interno dei database.

L'approccio corretto consiste nell'utilizzare istruzioni SQL preparate che sfuggono al controllo dell'utente.

3.2.6.7 MASTG-TEST-0034 → Testare la persistenza degli oggetti

MASVS v2 ID	MASVS v1 IDs
MASVS-CODE-4	MSTG-PLATFORM-8

ESITO REQUISITO: rispettato

Dall'analisi statica manuale vediamo che vi è l'utilizzo della classe Serializable

3.2.6.8 MASTG-TEST-0044 → Assicurarsi che le funzioni di sicurezza gratuite siano attivate

MASVS v2 ID	MASVS v1 IDs
MASVS-CODE-4	MSTG-CODE-9

ESITO REQUISITO: N/A

Il requisito in questione riguarda la verifica dei meccanismi di protezione del codice binario all'interno delle librerie native, il che dipende principalmente dal linguaggio di programmazione utilizzato. Tuttavia, questo requisito risulta non verificabile poiché durante la fase di compilazione della release, le librerie native vengono eliminate dall'APK finale.

3.2.7 MASVS-V7: Requisiti di Resilienza [RESILIENCE]

3.2.7.1 MASTG-TEST-0045 → Test del rilevamento delle radici

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-1	MSTG-RESILIENCE-1

ESITO REQUISITO: non rispettato

L'applicazione non effettua alcun controllo riguardante l'esecuzione su un dispositivo con i permessi Root.

L'analisi è stata eseguita su un Android Emulatore Genymotion il quale possiede i permessi di root e l'applicazione è stata eseguita su un dispositivo con permessi di root senza che fosse stata interrotta o avesse emesso alcun avviso.

3.2.7.2 MASTG-TEST-0049 → Verifica del rilevamento dell'emulatore

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-1	MSTG-RESILIENCE-5

ESITO REQUISITO: non rispettato

Nonostante MobSF rilevi la presenza di controlli anti-VM, l'applicazione non implementa alcun controllo per rilevare l'esecuzione su un emulatore. Durante l'analisi, l'applicazione è stata eseguita in un emulatore senza essere interrotta o aver emesso alcun avviso, nonostante l'app venga eseguita capovolta. Da ciò si deduce che il requisito non è stato soddisfatto.

Anti-VM Code	Build.FINGERPRINT check Build.MANUFACTURER check Build.BOARD check possible Build.SERIAL check
--------------	---

3.2.7.3 MASTG-TEST-0047 → Verifica dei controlli di integrità dei file

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-2	MSTG-RESILIENCE-3

ESITO REQUISITO: non rispettato

Dinamicamente, è stato verificato che, modificando alcuni dati all'interno dei file, quali ad esempio le credenziali di accesso, presenti nella cartella shared preferences, l'applicazione non rileva e non risponde a tale manomissione.

Questa verifica è stata effettuata:

- eseguendo il comando per scaricare il pacchetto della macchina virtuale per scaricare il pacchetto della cartella contenente il file dell'app
"adb pull /data/data/"tua app"
- modificando il file UserInfo.xml
- eseguendo il push dei file manomessi
"adb push indirizzo/del/file/modificato /data/data/"tua app"/"shared_prefs"
- aprire l'emulatore e riavviare l'app

3.2.7.4 MASTG-TEST-0038 → Assicurarsi che l'applicazione sia firmata correttamente

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-2	MSTG-CODE-1

ESITO REQUISITO: rispettato

Per verificare questo requisito, è stato utilizzato sia il report generato dall'analisi di MobSF, sia con il jarsigner. Si può notare che l'applicazione risulta firmata ed esiste un certificato valido.

Questa analisi indica che il file binario dell'applicazione è firmato digitalmente, e sono presenti firme per le versioni 1, 2 e 3, ma non per la versione 4 del formato di firma APK (Android Package).

In termini pratici:

- "Binary is signed" significa che il file binario (APK) è stato firmato digitalmente.
- "v1 signature: True" indica che è presente una firma per la versione 1 del formato di firma APK.
- "v2 signature: True" indica che è presente una firma per la versione 2 del formato di firma APK.
- "v3 signature: True" indica che è presente una firma per la versione 3 del formato di firma APK.
- "v4 signature: False" indica che non è presente una firma per la versione 4 del formato di firma APK.

Le firme digitali sono importanti per verificare l'autenticità e l'integrità di un'applicazione Android.

```
android@tamer ~> jarsigner -verify -verbose -certs /home/android/Downloads/AigoSmart_1.9.0_Apkpure.apk
sm      93444 Thu Jan 01 01:01:02 EST 1981 AndroidManifest.xml
X.509, CN=www.tuya.com, OU=www.tuya.com, O=tuya, L=Hangzhou, ST=ZheJiang, C=CN
[certificate is valid from 11/27/18 2:04 AM to 8/30/73 3:04 AM]
[CertPath not validated: Path does not chain with any of the trust anchors]
```

CERTIFICATE INFORMATION

```
Binary is signed
v1 signature: True
v2 signature: True
v3 signature: True
v4 signature: False
X.509 Subject: C=CN, ST=ZheJiang, L=Hangzhou, O=tuya, OU=www.tuya.com, CN=www.tuya.com
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2018-11-27 07:04:51+00:00
Valid To: 2073-08-30 07:04:51+00:00
Issuer: C=CN, ST=ZheJiang, L=Hangzhou, O=tuya, OU=www.tuya.com, CN=www.tuya.com
Serial Number: 0x22f7fed4
Hash Algorithm: sha256
md5: 5737fa36e9e74ac5035a98af536b6ccf
sha1: b42c7c040d32295afe35b042847f76f1444057dd
sha256: 8b4f4903324878aa284c06824097e875667d0598a1b1bc7dce11f81c643fd117
sha512: a7447b2aa059764dae2828f2f53ed649e7a05c987733c48ffa3325d4c0c61ce6df7df9252d38350dab739a941acd85ceb4c620c6fd0fbf15881cce3f867d75c6
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: f9e7bf933639dbae8f41233843916b0d9b82ea4fb838cef75adadde8d5908b0d
Found 1 unique certificates
```

3.2.7.5 MASTG-TEST-0050 → Testare i controlli di integrità del run-time

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-2	MSTG-RESILIENCE-6

ESITO REQUISITO: non rispettato

L'app non è in grado di rilevare manomissioni, come dimostrato MASTG-TEST-0047, è quindi suscettibile a modifiche di dati, anche corrotti.

3.2.7.6 MASTG-TEST-0040 → Test per i simboli di debug

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-3	MSTG-CODE-3

ESITO REQUISITO: rispettato

L'applicazione è stata pubblicata in modalità "release", il che implica la rimozione dei simboli di debugging durante la creazione del pacchetto di distribuzione.

3.2.7.7 MASTG-TEST-0041 → Test per il debug del codice e la registrazione degli errori verbosi

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-3	MSTG-CODE-4

ESITO REQUISITO: rispettato

Nei log di logcat non appaiono log di debug.

3.2.7.8 MASTG-TEST-0051 → Verifica dell'offuscamento

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-3	MSTG-RESILIENCE-9

ESITO REQUISITO: non rispettato

Tale requisito non è soddisfatto in quanto per eseguire l'analisi del codice l'apk è stato prima analizzato attraverso il tool MaraFramework e poi è stata eseguita un ulteriore fase di decompilazione attraverso il tool JD-GUI.

3.2.7.9 MASTG-TEST-0046 → Verifica del rilevamento anti-debugger

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-4	MSTG-RESILIENCE-2

ESITO REQUISITO: non rispettato

L'applicazione non include alcuna protezione contro l'accesso da parte di debugger esterni. Infatti, è stato possibile eseguire vari test utilizzando direttamente Android Debug Bridge (adb), senza riscontrare anomalie durante l'utilizzo dell'app. Questo non soddisfa il requisito stabilito.

3.2.7.10 MASTG-TEST-0039 → Verifica se l'applicazione è debuggabile

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-4	MSTG-CODE--2

ESITO REQUISITO: rispettato

Nella build di rilascio, è essenziale disattivare le funzionalità di debugging. Per verificare ciò, abbiamo esaminato il tag nel file AndroidManifest.xml alla ricerca dell'attributo android:debuggable. Non essendo stato trovato, il valore predefinito è stato impostato su "false".

A conferma del riscontro ricevuto dal Manifest, come affermato dalla guida se si esegue il comando

\$ adb shell dumpsys nome_pacchetto | grep -c "DEBUGGABLE"

e ottieni un risultato pari a zero, significa che l'applicazione non è abilitata per il debug.

Successivamente, se esegui per avere un ulteriore conferma:

\$ run-as nome_pacchetto id

l'output, conferma che l'applicazione non è debuggabile.

Pertanto, il requisito è stato soddisfatto.

```
android@tamer ~> adb shell dumpsys package com.aigostar.smart | grep -c "DEBUGGABLE"
0
android@tamer ~> adb shell
vbox86p:/ # run-as com.aigostar.smart id
run-as: package not debuggable: com.aigostar.smart
```

3.2.7.11 MASTG-TEST-0048 → Verifica del rilevamento degli strumenti di reverse engineering

MASVS v2 ID	MASVS v1 IDs
MASVS-RESILIENCE-4	MSTG-RESILIENCE-2

ESITO REQUISITO: non rispettato

Questo requisito richiede che l'applicazione verifichi la presenza di strumenti per il reverse engineering o framework sul dispositivo e fornisca una notifica appropriata in caso di rilevamento. Tuttavia, per quanto riguarda l'applicazione in esame, questo requisito non può essere testato poiché non sono stati utilizzati strumenti di reverse engineering sul dispositivo emulato.

4 VULNERABILITÀ

Si riportano ora le vulnerabilità identificate all'interno dell'applicazione **AigoSmart** (com.aigostar.smart) a seguito della verifica dei controlli previsti dalle linee guida OWASP.

4.1 SCOPE

FILE INFORMATION

File Name: AigoSmart_1.9.0_Apkpure.apk

Size: 43.93MB

MD5: 217d2fc2fabece3712af1cded6f6b68a

SHA1: 00fee6ee0022e075a5fcdf3e7b5a1230fdb3b792

SHA256: 6cbd0b278a01e7046a1026629e230e2141c20e04b7c47e8300958f3db4db8448

APP INFORMATION

App Name: AigoSmart

Package Name: com.aigostar.smart

Main Activity: com.aigostar.module.common.module.main.SplashActivity

Target SDK: 30

Min SDK: 21

Max SDK:

Android Version Name: 1.9.0

4.2 EXECUTIVE SUMMARY

È stato identificato un insieme di vulnerabilità durante i recenti test condotti sulle misure di sicurezza dell'App AigoSmart, utilizzando il framework MASV dell'OWASP. Queste vulnerabilità rappresentano potenziali rischi per la sicurezza dei dati e dei dispositivi coinvolti nell'utilizzo dell'applicazione. Di seguito sono elencate le principali vulnerabilità riscontrate:

1. Verifica dei criteri di sicurezza per l'accesso ai dispositivi
2. Test dell'archiviazione locale per i dati sensibili
3. Test della crittografia dei dati sulla rete
4. Test della crittografia dei dati sulla rete (duplicata)
5. Verifica dell'implementazione vulnerabile di PendingIntent
6. Test dei permessi delle app
7. Verifica dell'esecuzione di JavaScript nelle WebView
8. Verifica dell'aggiornamento forzato
9. Test della memorizzazione locale per la convalida degli input
10. Verifica del rilevamento dell'emulatore
11. Verifica dell'offuscamento

Questi elementi saranno esaminati nel dettaglio per comprendere appieno il loro impatto sulla sicurezza e per sviluppare strategie mirate per mitigare i rischi associati.

4.3 VULNERABILITÀ

4.3.1 Vulnerabilità #1 → Verifica dei criteri di sicurezza per l'accesso ai dispositivi

Dinamicamente, è stato verificato che l'accesso all'app non è moderato da alcun criterio di sicurezza minimo. Infatti, una volta autenticato, l'utente resta connesso all'applicazione, e chiunque abbia accesso al dispositivo può accedere all'applicazione senza dover autenticarsi (ad esempio tramite uso di password o di biometrie).

Inoltre, un parametro da considerare è relativo alla versione di android: l'app, come indicato dal file Manifest.xml e confermato dai tool automatici utilizzati, supporta una versione minima di Android (Min SDK) pari a 21. Questo significa che l'applicazione può essere installata e eseguita su dispositivi con Android 5.0 (Lollipop).

MITIGAZIONI

Per affrontare i problemi individuati riguardanti la mancanza di criteri di sicurezza minimi e la gestione dell'autenticazione, così come la compatibilità con versioni obsolete di Android, ecco alcune soluzioni:

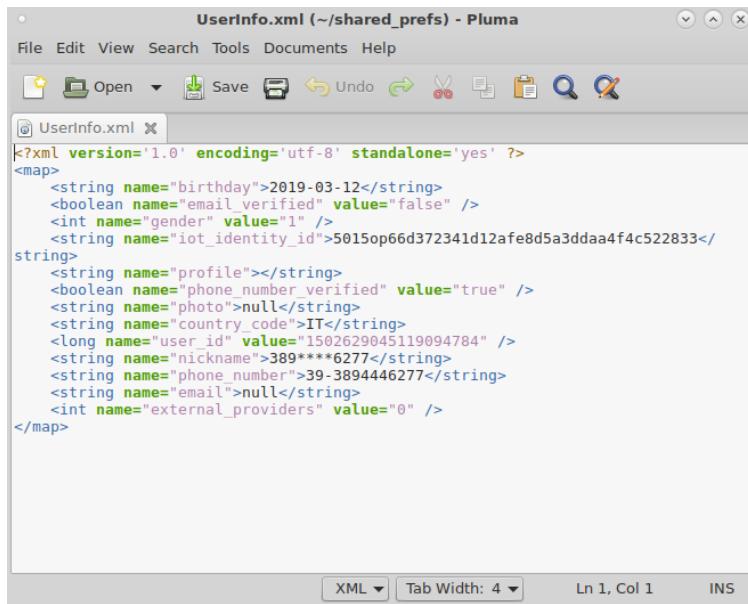
1. Implementazione di criteri di sicurezza per l'accesso.
2. Aggiornamenti di sicurezza obbligatori.
3. Aggiornamento del Min SDK.

Implementando queste soluzioni, si può migliorare significativamente la sicurezza e l'affidabilità dell'applicazione, riducendo al contempo il rischio per gli utenti e proteggendo i loro dati.

4.3.2 Vulnerabilità #2 → Test dell'archiviazione locale per i dati sensibili

Nell'applicazione in esame, questo requisito non è verificato dato che nel Manifest è presente il permesso “WRITE_EXTERNAL_STORAGE” che permette di archiviare in una memoria esterna dati relativi all'applicazione.

Inoltre, eseguendo un'analisi dinamica dell'applicazione, risultano facilmente accessibili e privi di algoritmi crittografici i dati sensibili quale i file relativi ai database e le preferenze condivise memorizzate come file XML (in /data/data/<nome-pacchetto>/shared_prefs).



The screenshot shows the Pluma text editor with the file "UserInfo.xml" open. The XML code contains sensitive user information such as birthday, email verification status, gender, IoT identity ID, profile, photo, country code, user ID, nickname, phone number, email, and external providers count. Below the editor, a terminal window displays the directory structure of the shared preferences folder, showing databases and various log files for com.aigostar.smart.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="birthday">2019-03-12</string>
    <boolean name="email_verified" value="false" />
    <int name="gender" value="1" />
    <string name="iot_identity_id">50150p66d372341d12afe8d5a3ddaa4f4c522833</string>
    <string name="profile"></string>
    <boolean name="phone_number_verified" value="true" />
    <string name="photo">null</string>
    <string name="country_code">IT</string>
    <long name="user_id" value="1502629045119094784" />
    <string name="nickname">389****6277</string>
    <string name="phone_number">39-389446277</string>
    <string name="email">null</string>
    <int name="external_providers" value="0" />
</map>
```

```
vbox86p:/data/data/com.aigostar.smart # ls
app_SGLib app_breeze app_webview code_cache files no_backup
app_accs app_textures cache databases lib shared_prefs
vbox86p:/data/data/com.aigostar.smart # cd databases
vbox86p:/data/data/com.aigostar.smart/databases # ls
accs.db
accs.db-journal
aigoStar-db
aigoStar-db-journal
com.google.android.datatransport.events
com.google.android.datatransport.events-journal
google_app_measurement.db
google_app_measurement.db-journal
google_app_measurement_local.db
google_app_measurement_local.db-journal
message_accs_db
message accs db-journal
```

MITIGAZIONI

Le informazioni sensibili, come le credenziali di accesso e le chiavi crittografiche, devono essere gestite e memorizzate utilizzando le API e le funzionalità di sicurezza fornite dal sistema operativo, come il KeyChain su iOS o il KeyStore su Android. È consigliabile sfruttare anche eventuali dispositivi hardware di sicurezza, come i moduli di sicurezza hardware (HSM), se supportati dal sistema o dal dispositivo in uso, per aggiungere un ulteriore livello di protezione.

Per garantire una maggiore sicurezza, è importante memorizzare gli hash delle password, dei PIN e delle chiavi crittografiche anziché i loro valori in chiaro. Questo rende più difficile per gli attaccanti ottenere l'accesso alle informazioni sensibili anche nel caso in cui riescano ad accedere al dispositivo. Inoltre, è essenziale gestire in modo sicuro le chiavi memorizzate all'interno del KeyChain/KeyStore, ad esempio utilizzando pratiche come l'uso di chiavi crittografiche a lunghezza adeguata, la limitazione degli accessi alle chiavi solo alle applicazioni autorizzate e l'implementazione di meccanismi di backup e ripristino sicuri per le chiavi.

Adottare queste pratiche consente di garantire un adeguato livello di sicurezza per le informazioni sensibili memorizzate all'interno dell'applicazione mobile, proteggendo così la riservatezza e l'integrità dei dati degli utenti.

4.3.3 Vulnerabilità #3 → Test della crittografia dei dati sulla rete

L'app risulta vulnerabile a un attacco Man-in-the-Middle (MITM). Questo tipo di attacco può verificarsi quando un malintenzionato si interpone tra il dispositivo di un utente e un server web. L'attaccante può quindi intercettare e modificare il traffico tra il dispositivo e il server, il che potrebbe consentirgli di rubare informazioni sensibili, come password o dati finanziari.

```
[line 290:     if (!z) {}  
[line 291:         b.a(b.ALLOW_ALL_HOSTNAME_VERIFIER);]  
[line 292:         SSLSocketFactory = b.TRUST_ALL_SSL_SOCKET_FACTORY;  
  
[line 12: public class b {}  
[line 13:     public static final HostnameVerifier ALLOW_ALL_HOSTNAME_VERIFIER =  
new a();]  
[line 14:     public static final SSLSocketFactory TRUST_ALL_SSL_SOCKET_FACTORY  
= C0012b.a();]
```

La vulnerabilità è dovuta al codice sopracitato che utilizza un verificatore di nomi host `ALLOW_ALL_HOSTNAME_VERIFIER`. Questo verificatore accetta qualsiasi nome host, indipendentemente dal fatto che corrisponda effettivamente al certificato SSL/TLS del server. Ciò significa che un malintenzionato può creare un falso server web con un nome host che corrisponde all'indirizzo web a cui sta tentando di accedere l'utente. L'utente quindi si conterrà al falso server web, che potrà intercettare e modificare il suo traffico.

MITIGAZIONI

Per risolvere questo problema, è necessario utilizzare un verificatore di nomi host più sicuro. Un verificatore di nomi host sicuro controllerà che il nome host del server corrisponda effettivamente al certificato SSL/TLS del server. Ciò aiuterà a garantire che l'utente si stia connettendo al server corretto e che il suo traffico non sia intercettato da un malintenzionato.

Un alternativa per correggere il codice:

```
URL url = new URL("https://example.org/");  
HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();  
  
HostnameVerifier hostnameVerifier = new MyHostnameVerifier();  
urlConnection.setHostnameVerifier(hostnameVerifier);
```

In questo codice, `MyHostnameVerifier` è una classe personalizzata che implementa l'interfaccia `HostnameVerifier`. Questa classe controllerà che il nome host del server corrisponda effettivamente al certificato SSL/TLS del server.

Ecco un esempio di implementazione della classe `MyHostnameVerifier`:

```
public class MyHostnameVerifier implements HostnameVerifier {  
  
    @Override  
    public boolean verify(String hostname, SSLSession session) {  
        X509Certificate cert = session.getPeerCertificateChain()[0];  
        try {  
            return cert.getSubjectDN().getName().equals(hostname);  
        } catch (SSLPeerUnverifiedException e) {  
            return false;  
        }  
    }  
}
```

Questo codice controllerà che il nome host del server corrisponda al nome comune del certificato SSL/TLS del server. Se i nomi non corrispondono, il codice restituirà false e la connessione verrà interrotta.

Oltre a utilizzare un verificatore di nomi host sicuro, è importante anche utilizzare SSL/TLS con cifratura forte. Ciò aiuterà a proteggere il traffico tra il dispositivo dell'utente e il server dall'intercettazione e dalla decodifica da parte di malintenzionati.

4.3.4 Vulnerabilità #4 → Test della crittografia dei dati sulla rete

L'applicazione mobile utilizza il protocollo HTTP per inviare o ricevere dati. Il design del protocollo HTTP non prevede alcuna crittografia dei dati trasmessi e può essere facilmente intercettato se un aggressore si trova nella stessa rete o ha accesso al canale dati della vittima.

NETWORK SECURITY

HIGH: 1 | WARNING: 0 | INFO: 0 | SECURE: 0

NO	SCOPE	SEVERITY	DESCRIPTION
1	*	high	Base config is insecurely configured to permit clear text traffic to all domains.

Se un malintenzionato intercetta i dati trasmessi tramite HTTP non crittografato, potrebbe potenzialmente rubare informazioni sensibili come:

- Password
- Informazioni finanziarie
- Dati personali
- Comunicazioni private

MITIGAZIONI

Il modo migliore per risolvere questa vulnerabilità è utilizzare il protocollo HTTPS al posto di HTTP. HTTPS è una versione crittografata di HTTP che protegge i dati da intercettazioni non autorizzate.

Alcuni esempi di come implementare HTTPS:

Utilizzare un certificato SSL/TLS sul server web.

Configurare il server web per utilizzare HTTPS come protocollo predefinito.

Aggiornare i collegamenti web da HTTP a HTTPS.

4.3.5 Vulnerabilità #5 → Verifica dell'implementazione vulnerabile di PendingIntent

Analizzando il manifest.xml è stato trovato vari tag `android:exported="true"` associati a molti Receivers. Come riportato su MobSF:

14	Broadcast Receiver (<code>com.google.firebaseio.iid.FirebaseInstanceIdReceiver</code>) is Protected by a permission, but the protection level of the permission should be checked. Permission: <code>com.google.android.c2dm.permission.SEND</code> [<code>android:exported=true</code>]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
16	Broadcast Receiver (<code>com.xiaomi.push.service.receivers.NetworkStatusReceiver</code>) is not Protected. [<code>android:exported=true</code>]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
17	Broadcast Receiver (<code>com.alibaba.sdk.android.push.MiPushBroadcastReceiver</code>) is not Protected. [<code>android:exported=true</code>]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
18	Broadcast Receiver (<code>com.alibaba.sdk.android.push.huawei.HuaweiPushReceiver</code>) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

21	Broadcast Receiver (com.alibaba.sdk.android.push.MeizuPushReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
23	Broadcast Receiver (com.alibaba.sdk.android.push.impl.PushMessageReceiverImpl) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
26	Broadcast Receiver (androidx.work.impl.diagnostics.DiagnosticsReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.DUMP [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
27	Broadcast Receiver (com.instacart.library.truetime.BootCompletedBroadcastReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
35	Broadcast Receiver (com.taobao.agoo.AgooCommandReceiver) is not Protected. [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
37	Broadcast Receiver (com.alibaba.sdk.android.push.SystemEventReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
31	Broadcast Receiver (com.taobao.accs.EventReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
32	Broadcast Receiver (com.taobao.accs.ServiceReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.

MITIGAZIONI

Per proteggere le activity dall'accesso non autorizzato, è consigliabile aggiungere o modificare l'attributo "android:exported=false" nel manifest. Questo impedisce alle activity di essere richiamate direttamente attraverso intent o da altre app installate sullo stesso dispositivo. È inoltre opportuno implementare meccanismi di protezione, rilevamento e blocco per prevenire tali operazioni non autorizzate.

4.3.6 Vulnerabilità #6 → Test dei permessi delle app

Le autorizzazioni in una app Android sono essenziali per preservare la privacy degli utenti. Quando una app necessita di accedere a dati sensibili o funzionalità del dispositivo, come la fotocamera o la connessione Internet, richiede le autorizzazioni appropriate. Il sistema operativo Android può concedere automaticamente tali autorizzazioni o richiedere l'approvazione dell'utente, a seconda della natura delle richieste. Durante lo sviluppo di un'app, gli sviluppatori devono dichiarare le autorizzazioni necessarie per garantire l'accesso alle funzionalità protette del dispositivo, assicurando che gli utenti siano consapevoli di tali richieste.

MobSF, nel nostro caso, ha identificato alcune autorizzazioni considerate "pericolose" o "sconosciute" dal punto di vista della sicurezza dell'applicazione e del dispositivo. Questo suggerisce che tali autorizzazioni potrebbero comportare rischi per la sicurezza o la privacy degli utenti.

com.meizu.c2dm.permission.RECEIVE	unknown	Unknown permission
com.meizu.flyme.push.permission.RECEIVE	unknown	Unknown permission
android.permission.READ_LOG_S	unknown	Unknown permission
android.permission.READ_SETTINGS	unknown	Unknown permission
android.permission.RECEIVE_USER_PRESENT	unknown	Unknown permission
android.permission.WRITE_MEDIA_STORAGE	unknown	Unknown permission
com.aigostar.smart.permission.C2D_MESSAGE	unknown	Unknown permission
com.aigostar.smart.permission.JPUSH_MESSAGE	unknown	Unknown permission
com.aigostar.smart.permission.MIPUSH_RECEIVE	unknown	Unknown permission
com.aigostar.smart.push.permission.MESSAGE	unknown	Unknown permission
com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE	unknown	Unknown permission
com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE	unknown	Unknown permission
android.permission.SYSTEM_ALERT_WINDOW	dangerous	display system-level alerts
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents
android.permission.WRITE_SETTINGS	dangerous	modify global system settings

PERMISSION	STATUS	INFO
android.permission.ACCESS_COARSE_LOCATION	dangerous	coarse (network-based) location
android.permission.ACCESS_FINE_LOCATION	dangerous	fine (GPS) location
android.permission.CAMERA	dangerous	take pictures and videos
android.permission.GET_ACCOUNTS	dangerous	list accounts
android.permission.GET_TASKS	dangerous	retrieve running applications
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	dangerous	mount and unmount file systems
android.permission.READ_EXTERNAL_STORAGE	dangerous	read external storage contents
android.permission.READ_LOGS	dangerous	read sensitive log data
android.permission.READ_PHONE_STATE	dangerous	read phone state and identity
android.permission.RECORD_AUDIO	dangerous	record audio

MITIGAZIONI

Per mitigare questi rischi, è cruciale che l'app gestisca attentamente l'utilizzo di tali autorizzazioni "pericolose" e assicuri che siano strettamente necessarie per le funzionalità principali dell'app stessa. Inoltre, è essenziale informare chiaramente gli utenti sull'utilizzo di tali autorizzazioni e consentire loro di concedere o revocare il consenso in modo esplicito. Questo garantirà che gli utenti siano pienamente consapevoli e in grado di controllare l'accesso dell'app ai loro dati sensibili e alle funzionalità del dispositivo.

4.3.7 Vulnerabilità #7 → Verifica dell'esecuzione di JavaScript nelle WebView

L'applicazione utilizza WebView per visualizzare pagine web all'interno dell'app stessa. Tuttavia, l'abilitazione del JavaScript attraverso la funzione setJavaScriptEnabled non rispetta i requisiti di sicurezza raccomandati. Il codice mostra che il JavaScript viene abilitato in modo esplicito, senza garantire un canale di comunicazione sicuro tramite HTTPS su TLS per l'interazione con gli endpoint. Secondo le linee guida di sicurezza OWASP, è fondamentale che la comunicazione avvenga esclusivamente su canali protetti per garantire la sicurezza dei dati scambiati.

Inoltre, il rendering di documenti HTML potrebbe contenere CSS e Javascript per aggiungere stili e dinamicità alla pagina. Tuttavia, l'utilizzo di Javascript potrebbe essere sfruttato da malintenzionati per eseguire codice malevolo, rappresentando una vulnerabilità. Per mitigare questo rischio, di default le WebView disabilitano l'esecuzione di Javascript. È possibile abilitarlo manualmente utilizzando l'oggetto WebSettings e il metodo setJavascriptEnabled(true), ma questa operazione deve essere effettuata con cautela per evitare possibili minacce alla sicurezza.

MITIGAZIONI

Se è assolutamente necessario abilitare l'esecuzione di JavaScript all'interno delle WebView, è essenziale farlo attraverso misure di sicurezza rigorose, come l'instaurazione di una connessione sicura tramite i protocolli SSL/TLS. Questo assicura che l'interazione con le WebView avvenga in un ambiente protetto, riducendo il rischio di esposizione a vulnerabilità e attacchi esterni.

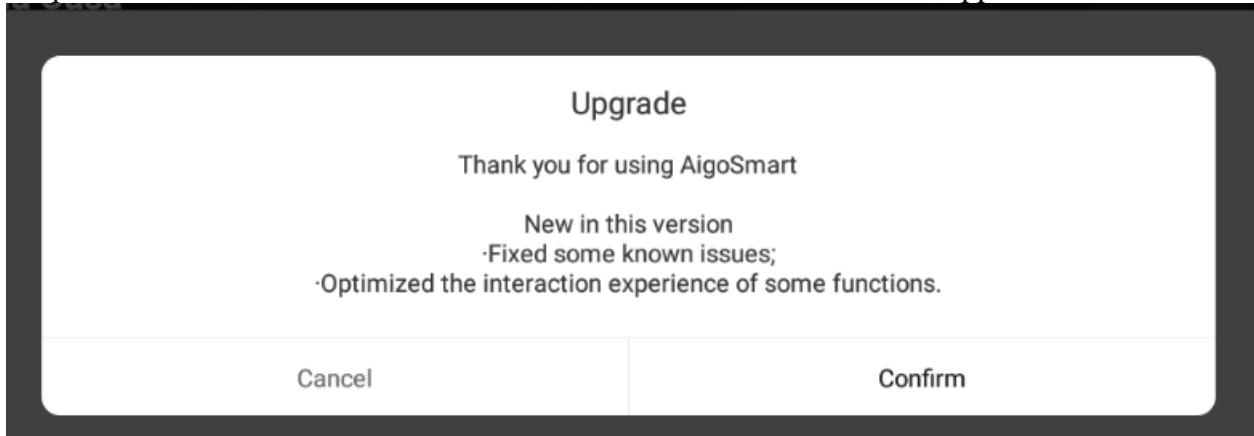
4.3.8 Vulnerabilità #8 → Verifica dell'aggiornamento forzato

Secondo la guida, nell'analisi dinamica, è importante testare l'aggiornamento corretto dell'applicazione.

La guida consiglia di scaricare una versione precedente dell'applicazione con una vulnerabilità di sicurezza, sia tramite una release dagli sviluppatori o utilizzando un'app store di terze parti.

Successivamente, verificare se si può continuare a utilizzare l'applicazione senza aggiornarla. Se viene mostrato un avviso di aggiornamento, verifica se puoi comunque utilizzare l'applicazione annullando l'avviso o aggirandolo attraverso un normale utilizzo dell'applicazione. Ciò include la convalida se il backend interrompe le chiamate ai backends vulnerabili e/o se la versione dell'app vulnerabile è bloccata dal backend. Infine, prova a manipolare il numero di versione di un'app compromessa e osserva come il backend risponde a questo (e se viene registrato in qualche modo).

L'app presa in analisi avvisa l'utente tramite riquadri che la versione dell'applicazione risulta obsoleta ma permette all'utente di chiudere l'avvertimento e continuare ad usare l'app.



MITIGAZIONI

Per imporre gli aggiornamenti dell'applicazione, si potrebbero considerare alcune strategie quali:

1. Puoi impedire agli utenti di accedere all'applicazione fintanto che non effettuano l'aggiornamento alla versione più recente
2. Forzare l'aggiornamento all'avvio.
3. Messaggi persistenti e non saltabili.
4. Disabilitazione graduale del supporto per versioni obsolete

Implementando una combinazione di queste strategie, puoi incoraggiare attivamente gli utenti a mantenere l'applicazione aggiornata per garantire la sicurezza e la corretta funzionalità.

4.3.9 Vulnerabilità #9 → Test della memorizzazione locale per la convalida degli input
L'utilizzo di input direttamente nelle query SQL espone l'applicazione mobile al rischio di SQL injection, una vulnerabilità che potrebbe permettere a un attaccante di manipolare le query SQL per accedere o modificare dati sensibili nei database sottostanti.

MITIGAZIONI

Per mitigare questo rischio, è fondamentale adottare un approccio sicuro utilizzando istruzioni SQL preparate. Ad esempio:

Example of secure code:

```
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET SALARY = ?  
WHERE ID = ?");  
pstmt.setBigDecimal(1, 153833.00)  
pstmt.setInt(2, 110592)
```

Example of insecure code:

```
db.rawQuery("SELECT username FROM users_table WHERE id = '"+ input_id +"');  
db.execSQL("SELECT username FROM users_table WHERE id = '"+  
input_id +"');
```

Queste istruzioni permettono di separare i dati dall'instradamento delle query, impedendo agli utenti malintenzionati di inserire comandi SQL dannosi.

Inoltre, l'uso di istruzioni SQL preparate rende possibile la parametrizzazione dei dati, garantendo che i valori inseriti siano trattati come dati e non come parte della query stessa. Questo approccio migliora la sicurezza dell'applicazione mobile proteggendo da potenziali attacchi di SQL injection e preservando l'integrità e la riservatezza dei dati sensibili.

4.3.10 Vulnerabilità #10 → Verifica del rilevamento dell'emulatore

L'applicazione, nonostante abbia implementato dei controlli sull'ambiente di esecuzione, non riesce a impedire il suo avvio su un dispositivo emulato.



Questo è preoccupante in quanto gli emulatori offrono un ambiente in cui è più semplice per un potenziale attaccante accedere ai dati del sandbox dell'applicazione, consentendo il furto di informazioni sensibili. Inoltre, l'utilizzo di emulatori facilita l'accesso agli strumenti di debugging, permettendo di ottenere informazioni sull'applicazione sia durante l'esecuzione sia quando non è attiva.

Anti-VM Code	Build.FINGERPRINT check Build.MANUFACTURER check possible Build.SERIAL check Build.TAGS check network operator name check
Anti-VM Code	Build.MANUFACTURER check Build.BOARD check possible Build.SERIAL check Build.TAGS check SIM operator check network operator name check possible ro.secure check
Anti-VM Code	Build.FINGERPRINT check Build.MODEL check Build.MANUFACTURER check Build.PRODUCT check Build.BOARD check possible Build.SERIAL check SIM operator check subscriber ID check

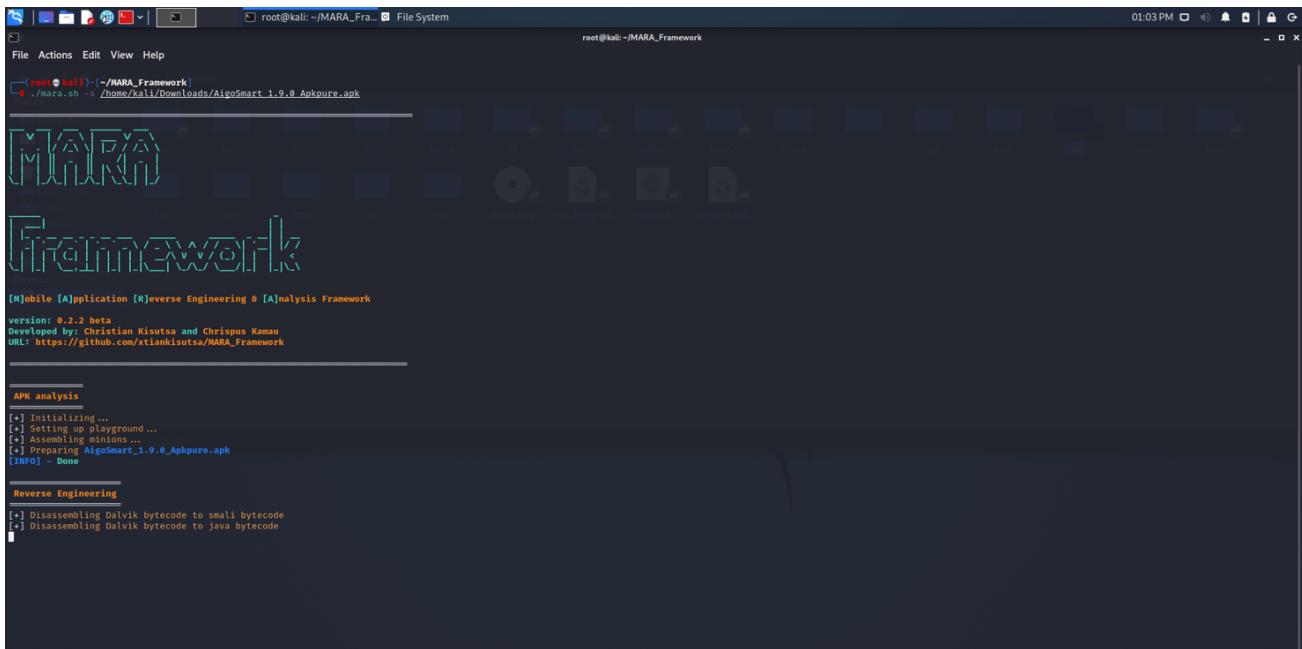
MITIGAZIONI

Per mitigare questi rischi, è necessario implementare controlli aggiuntivi che impediscono l'utilizzo dell'app su un emulatore. Ad esempio, si possono utilizzare controlli sulle proprietà del sistema, come la tipologia del processore, per distinguere un dispositivo reale da un emulatore. Tuttavia, è importante notare che esistono diverse tecniche per aggirare questi controlli di sicurezza, come l'iniezione di codice JavaScript tramite Frida.

Pertanto, oltre ad implementare i controlli sopra descritti, è fondamentale proteggere i dati sensibili utilizzando tecniche crittografiche appropriate e configurate correttamente. In questo modo, anche se un utente malintenzionato riuscisse ad eseguire l'applicazione su un emulatore, avrebbe difficoltà nell'accedere ai dati sensibili.

4.3.11 Vulnerabilità #11 → Verifica dell'offuscamento

L'applicazione non implementa alcuna difesa dagli attacchi di Reverse Engineering. In particolare, per l'analisi dell'applicazione è stato utilizzato il tool MaraFramework su Kali Linux fornendo il file apk dell'app.

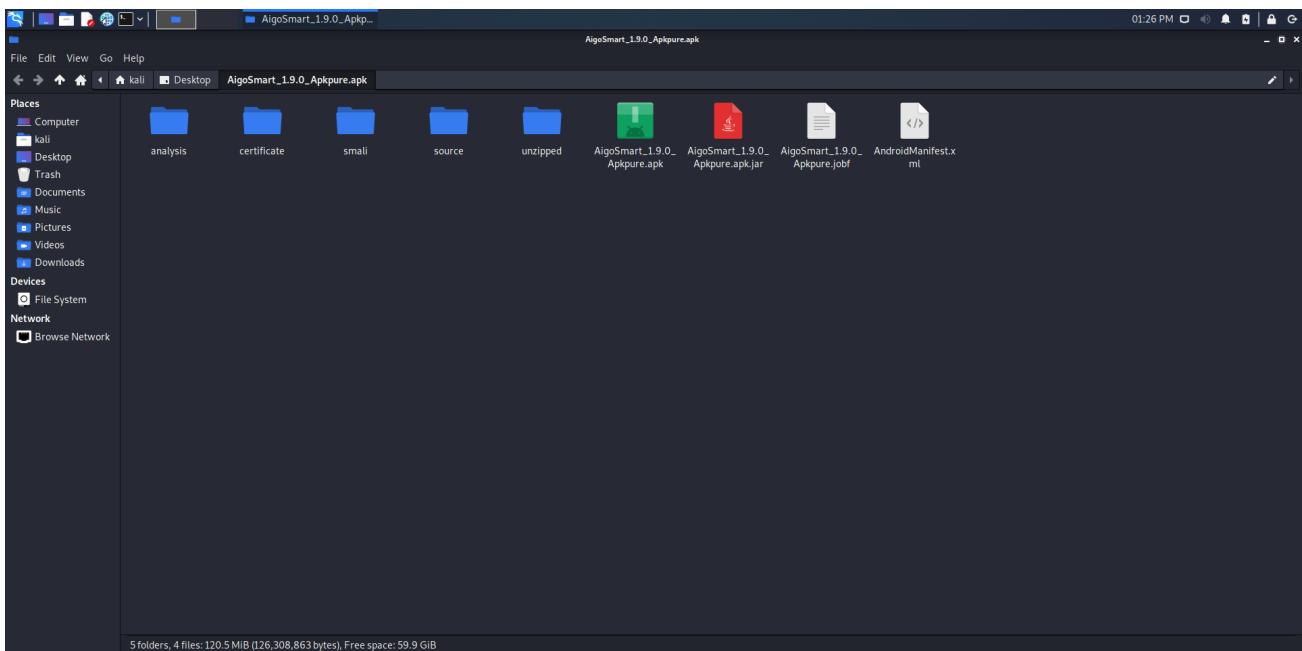


```
root@kali:~/MARA_Framework$ ./mara.sh -v
[+] Starting ...
[+] Setting up playground ...
[+] Assembling minions ...
[+] Preparing AlgoSmart_1.9.0_Apkpure.apk
[INFO] - Done

[APK analysis]
[+] Disassembling
[+] Setting up playground ...
[+] Assembling minions ...
[+] Preparing AlgoSmart_1.9.0_Apkpure.apk
[INFO] - Done

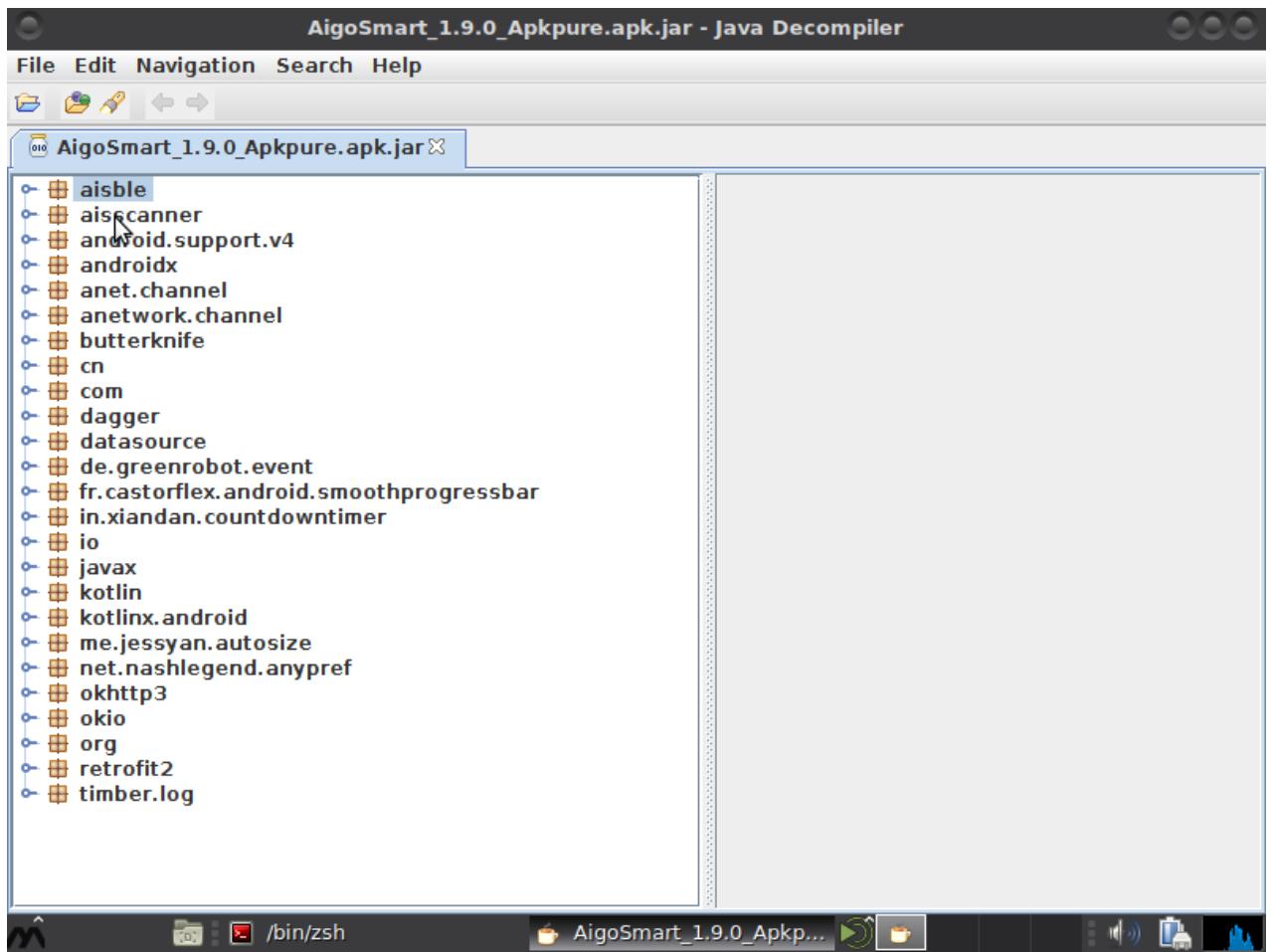
[Reverse Engineering]
[+] Disassembling Dalvik bytecode to small bytecode
[+] Disassembling Dalvik bytecode to java bytecode
```

Il tool ha dato come output un report di analisi, diviso in statica e dinamica, e ha fornito il file .jar dell'applicazione e il file Manifest.xml.



Successivamente il file .jar è stato elaborato dal tool fornito dalla macchina virtuale Android Tamer, JD-GUI.

Questo tool ha scompattato il file .jar nei file contenenti il codice sorgente.



MITIGAZIONI

Offuscare opportunamente il codice sorgente in fase di build dell'apk, eventualmente anche cifrando porzioni di codice particolarmente sensibili come, ad esempio, quelle che gestiscono dati sensibili.

5 CONCLUSIONE

In questo studio, si è condotta un'analisi approfondita dell'applicazione mobile "AIGOSMART" utilizzando una serie di metodologie e strumenti, tra cui VirtualBox, Kali Linux, Android Tamer, Genymotion, Android Debug Bridge (ADB), Docker Desktop, MobSF (Mobile Security Framework), Java Decomplier (JD-GUI), Mara (Mobile Application Reverse Engineering and Analysis) e ImmuniWeb.

L'obiettivo principale è stato **valutare l'app in base ai requisiti OSWAP** e identificare le vulnerabilità presenti.

Le vulnerabilità individuate rappresentano una minaccia significativa per la sicurezza dell'app e potrebbero consentire agli aggressori di compromettere l'integrità, la riservatezza e la disponibilità dei dati sensibili degli utenti. Tuttavia, nonostante le vulnerabilità identificate, è emersa **un'opportunità di miglioramento della sicurezza dell'app**.

Sono state fornite **raccomandazioni specifiche per affrontare le diverse vulnerabilità**, come l'implementazione di tecniche di protezione anti-reverse engineering, la gestione corretta dei dati sensibili, , l'utilizzo di connessioni crittografate tramite HTTPS e la gestione adeguata dei permessi dell'applicazione.

In conclusione, si sottolinea l'importanza di una valutazione rigorosa della sicurezza delle app mobili e si evidenzia che l'app analizzata presenta numerose vulnerabilità. Tuttavia, attraverso l'implementazione delle raccomandazioni fornite, è possibile migliorare significativamente la sicurezza dell'app e proteggere meglio i dati sensibili degli utenti.

Si ribadisce l'importanza di integrare la sicurezza fin dalle fasi iniziali dello sviluppo delle app al fine di mitigare le vulnerabilità e garantire una migliore protezione contro potenziali minacce.