



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Vulnerabilità Software in Android: Analisi della Comprensibilità dei Tool di Identificazione Automatica

RELATORE

Prof. Fabio Palomba

Dott. Emanuele Iannone

Dott. Giammaria Giordano

Università degli Studi di Salerno

CANDIDATO

Alfredo Cannavaro

Matricola: 0512108651

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

sesa^{lab}
SOFTWARE ENGINEERING
SALERNO

Ai miei genitori e mia sorella

Abstract

Nel 2023 l'uso dei dispositivi mobile aumenta ogni giorno. Un numero continuo e crescente di applicazioni mobile viene costantemente inserito negli Android Market come ad esempio Google Play Store per soddisfare le esigenze dei possessori di smartphone. Molte applicazioni Android non affrontano gli aspetti della sicurezza in modo appropriato. Ciò è spesso dovuto alla mancanza di meccanismi automatizzati per identificare, testare e correggere le vulnerabilità del codice sorgente nelle prime fasi di progettazione e sviluppo. Pertanto, c'è la necessità di risolvere tali problemi nelle fasi iniziali piuttosto che fornire aggiornamenti e patch alle applicazioni pubblicate. Questo studio si pone l'obiettivo di analizzare dei tool che identificano le vulnerabilità in un'applicazione Android. Lo studio si focalizza principalmente su quanto è comprensibile l'output di un tool. Durante il lavoro di tesi sono stati presi in considerazione alcuni studi che sono stati svolti in passato in questo ambito ed i dati raccolti da quest'ultimi. Sulla base di questi, sono stati ricavati tre tool che soddisfacessero dei requisiti prefissati e si è svolta un'analisi su un dataset di APK, raccogliendo le informazioni dell'output come il numero di vulnerabilità trovate ed le vulnerabilità più comuni. Infine lo studio si conclude con un'analisi dell'eccezioni restituite durante la fase di analisi, verificando se quest'ultime fossero esplicative ossia che facessero riferimento a parti di codice ben precise. I dati raccolti possono essere ampliati con l'inserimento di analisi più approfondite sulle vulnerabilità ottenute dai tool, fornendo una classifica rispetto alla loro pericolosità nel caso l'utente ignorasse quest'ultime e le eccezioni segnalate.

Indice

Elenco delle Figure	iii
Elenco delle Tabelle	iv
1 Introduzione	1
1.1 Contesto Applicativo	1
1.2 Motivazione ed obiettivi	2
1.3 Risultati ottenuti	3
1.4 Struttura della tesi	3
2 Background e Stato dell'arte	5
2.1 Background	5
2.1.1 Android	5
2.1.2 Vulnerabilità Software	6
2.1.3 Analisi statica, dinamica ed ibrida	8
2.1.4 Google Play Store	8
2.1.5 Librerie API e file APK	8
2.2 Stato dell'arte	9
2.2.1 Problema in analisi	9
2.2.2 Approccio con il dataset AndroZoo	10
2.2.3 Approccio con l'analisi statica	10

3	Metodologia	11
3.1	Metodologia di ricerca	11
3.2	Approccio con i Tool	12
3.3	Dataset	16
3.3.1	AndroZoo	16
3.3.2	Tool selezionati	18
3.4	Apktool	19
3.5	Analisi dei tool	21
4	Risultati ottenuti	22
4.1	Yaazhini	22
4.1.1	Raccolta dati di Yaazhini	24
4.2	MobSF	26
4.2.1	Raccolta dati di MobSF	27
4.3	Pithus	29
4.3.1	Raccolta dati di Pithus	30
4.4	Bilancio finale	33
5	Conclusioni e sviluppi futuri	35
	Bibliografia	36

Elenco delle figure

2.1	Livelli di architettura Android	6
3.1	Esempio di codice presente nel file <i>apktool.yml</i>	19
4.1	Interfaccia d'input di Yaazhini.	23
4.2	Interfaccia d'input di MobSF.	26
4.3	Interfaccia d'input di Pithus.	29

Elenco delle tabelle

3.2	Vulnerabilità della applicazione AndroGoat.	15
3.3	Lista dei 30 APK.	17
3.4	Tabella delle versioni Android con i rispettivi SDK usati.	20
3.5	Esempio di raccolta dei dati.	21
4.1	L'Analisi di Yaazhini sui 30 APK.	24
4.2	L'Analisi di MobSf sui 30 APK.	27
4.3	L'Analisi di Pithus sui 30 APK.	30
4.4	Raccolta finale dei dati.	33
4.5	Numero di istanze della vulnerabilità più comune	33

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

Il mondo della tecnologia ha visto negli ultimi anni uno sviluppo sempre più crescente e con la creazione dei dispositivi mobile come smartphone, tablet e notebook, tutti possono accedere al mondo virtuale in qualsiasi luogo si trova, basta avere una connessione ad internet. Questi dispositivi mobile nel corso degli anni hanno avuto uno sviluppo notevole tanto da potersi mettere a confronto o persino sostituire i personal computer. Inoltre sono personali, ciò significa che i dati generati comprendono parti più profonde della sfera personale e quindi proprio per questo motivo vanno protetti ma per farlo bisogna conoscere il sistema che li gestisce. Android è un sistema operativo ad accesso aperto sviluppato da Google ha una quota di mercato di circa l'87,5% [11].

Android è open source quindi il codice sorgente è disponibile a tutti. L'impatto sulla sicurezza della disponibilità del codice sorgente è stato oggetto di un certo dibattito (e lo è ancora). Tuttavia, la percezione che il codice open source comporti dei rischi sta diminuendo. *RedHat*, una società di software americana che fornisce prodotti software open source, ha realizzato delle interviste con 1.250 leader IT in tutto il mondo ed i risultati riportano che oltre il 30% degli intervistati vede una migliore

sicurezza come uno dei tre principali vantaggi mentre l'87% vede l'open source aziendale come 'altrettanto sicuro' rispetto al software proprietario [7]. Avendo il codice aperto, Android potrebbe risultare meno sicuro dato che esistono poche soluzioni automatizzate per aiutare a rilevare e impedire che contributi dannosi infettino i repository open source [11]. I produttori di dispositivi possono apportare modifiche alla versione esistente del sistema operativo per soddisfare le loro specifiche hardware. Sebbene sia un'ottima funzionalità, può essere una fonte di attività dannosa [11], inoltre gli sviluppatori di applicazioni non sempre dedicano le proprie energie e tempo sulla sicurezza in quanto si concentrano principalmente sulla domanda dei consumatori [11]. Nella sicurezza informatica uno dei concetti principali è quello della vulnerabilità software. In particolare, una vulnerabilità consente a un attaccante di accedere a tutte le risorse e tutti i dati dell'hosting, quindi del dispositivo che ospita queste app [20]. Per garantire che ciò non avvenga è necessario analizzare il codice e il comportamento dell'applicazione.

1.2 Motivazione ed obiettivi

Nel 2022 il numero di persone che posseggono un cellulare è in crescita [6], quest'ultime inseriscono i propri dati personali e scaricano app da Google Play o da terze parti. Proprio per questo motivo bisogna evitare che ci siano app vulnerabili e per fare ciò è necessario verificare con l'utilizzo di tool. Le motivazioni della stesura della presente tesi sono da ricercare nell'analisi dei tool che permettono di garantire la sicurezza della condivisione dei dati sulle app di Android e capire che tipi di tool utilizzare per uno specifico problema. Il problema principale è che a seconda del tipo di analisi che si svolge c'è sempre il rischio di non identificare tutti gli errori, ad esempio può non identificare il CWE-119(Improper Restriction of Operations within the Bounds of a Memory Buffer) che è causato da un errore di programmazione, ma l'effetto collaterale potrebbe essere disastroso, poiché ciò può cancellare dati, rubare informazioni riservate e persino l'intera applicazione potrebbe bloccarsi a causa di questo overflow del buffer [1]. Il fulcro centrale del presente lavoro di tesi è infatti la complessità dell'identificazione delle vulnerabilità software Android, che viene analizzata e approfondita nei paragrafi seguenti. In questa tesi l'obiettivo è di

espandere la conoscenza sui tool di analisi di vulnerabilità, confrontandoli e fornendo ulteriori approfondimenti sui tipi di analisi ed i tool utilizzati per la rivelazione di vulnerabilità nei software Android.

1.3 Risultati ottenuti

Questo studio fornisce il seguente contributo:

1. Quante vulnerabilità identifica ogni tool;
2. Raccolta delle vulnerabilità identificate da ogni tool ed un approfondimento sulle vulnerabilità più comuni;
3. Un approfondimento sulle eccezioni trovate e se puntano a problemi identificati nel codice.

I risultati rivelano che il tool che identifica più vulnerabilità è *Yaazhini* con 259 istanze e la vulnerabilità più comune è *Insertion of Sensitive Information into Log File* comparsa in 186 istanze durante l'analisi complessiva di tutte le app. *Yaazhini* è l'unico tool che non ha mai rilevato *Insertion of Sensitive Information into Log File* (CWE-532) mentre *MobSF* e *Pithus* ne hanno identificato rispettivamente 164 e 22 istanze. Per quanto riguarda le eccezioni, sono risultate abbastanza esplicite in tutti i tool, infatti quasi tutte hanno fatto riferimento a problemi identificati in sezione di codice ben specifiche come ad esempio l'eccezioni **Application Data can be Backed up** e **Broadcast Receiver is not Protected** che hanno segnalato all'utente su che variabile intervenire per eliminare la vulnerabilità.

1.4 Struttura della tesi

Questa sezione espone la struttura della tesi, la quale è composta da cinque capitoli con le rispettive sotto sezioni. Il primo capitolo fornisce una introduzione al lavoro svolto, descrivendo l'innovazione tecnologica degli ultimi anni e di come quest'ultima con lo sviluppo ha portato a delle innovazioni ma anche delle fragilità

nel campo della sicurezza chiarendo le motivazione ed gli obiettivi da raggiungere in questo studio.

Il secondo capitolo presenta dei concetti fondamentali da apprendere per comprendere i risultati delle analisi delle vulnerabilità che saranno svolte nei capitoli successivi. Un esempio sono le diverse tecniche di analisi che un tool può svolgere su un file .apk.

All'interno del terzo capitolo si discute della metodologia utilizzata sia per lo studio e analisi degli articoli/siti web sia sull'impostazione dei dati di analisi dei tool.

Il quarto capitolo si divide in sezioni dove ognuna descrive i risultati ottenuti da un tool ed il bilancio finale. Nell'ultimo capitolo si riassume il lavoro svolto e si propongono diversi spunti di riflessione che potrebbero essere utilizzati in futuro per migliorare ed ampliare il lavoro.

Background e Stato dell'arte

2.1 Background

2.1.1 Android

Android è la piattaforma mobile open source più popolare [11] da quando le vendite generali degli smartphone sono aumentate notevolmente nel 2010 [19]. Inizialmente diverse alternative competevano con Android fino a quando nel 2012 Android e iOS erano i sistemi operativi più emergenti. Dopo il 2012 Android ha iniziato a dominare il mercato, di conseguenza anche gli studi sulla architettura per scoprire dove potessero essere presenti delle vulnerabilità [19].

Analisi dell'architettura Android

Dal punto di vista dell'architettura, il sistema operativo Android è suddiviso in quattro livelli come è rappresentato nella Figura 2.1 : il livello del kernel, le librerie e il livello di runtime, il livello del framework delle applicazioni e il livello delle applicazioni.

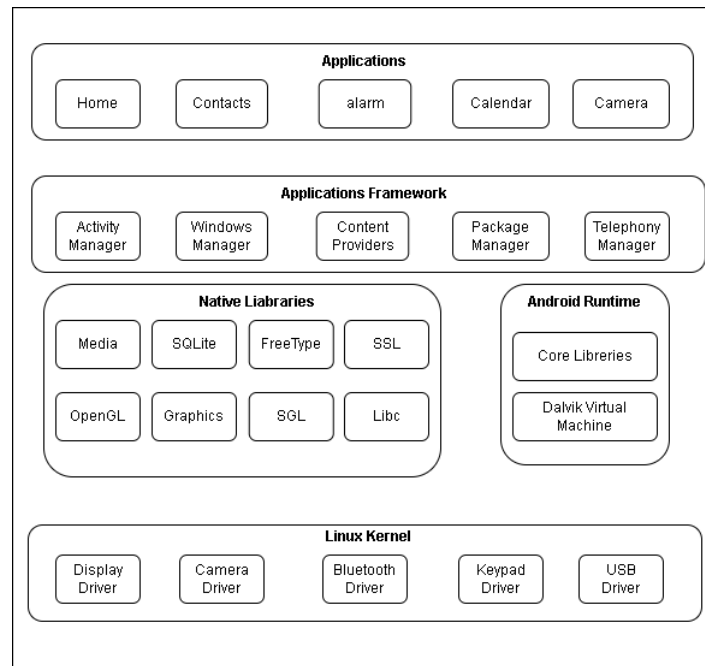


Figura 2.1: Livelli di architettura Android

Il kernel Android è una versione modificata del kernel Linux2.6 [10], che viene aggiornato di volta in volta con diverse versioni di Android. Le librerie forniscono supporto per grafica, funzionalità multimediali e archiviazione dei dati. Android runtime (ART) è il runtime gestito utilizzato dalle applicazioni e da alcuni servizi di sistema su Android. ART e il suo predecessore Dalvik sono stati originariamente creati appositamente per il progetto Android. ART come runtime esegue il formato eseguibile Dalvik e la specifica del bytecode Dex. ART e Dalvik sono runtime compatibili che eseguono Dex bytecode, quindi le app sviluppate per Dalvik dovrebbero funzionare durante l'esecuzione con ART [12] .

2.1.2 Vulnerabilità Software

Quando si parla di vulnerabilità si intende un difetto del codice, dovuto a qualche errore o superficialità nella realizzazione del programma, che si presta ad attacchi. Spesso la vulnerabilità è dovuta alla mancata validazione dell'input nel programma permettendo all'utente di inserire qualcosa che non è permesso. Se l'utente inserisce dati non corretti come minimo l'output non sarà corretto [22] . Un'app vulnerabile con determinate privilegi può eseguire comportamenti sensibili alla sicurezza dei dati. Questo tipo di vulnerabilità della sicurezza può presentarsi in numerose

forme e l'exploit di una o più vulnerabilità può provocare un effetto di "privilege escalation"(inteso come l'oltrepassare delle autorizzazioni), perdite di dati, attacco alle componenti hardware di un dispositivo, ecc [15]. Grindr, OKCupid, Bumble, Moovit, Edge, XRecorder, Viber e Booking, per esempio, potrebbero essere a rischio hacking a causa di una nota vulnerabilità, identificata come CVE-2020-8913(Il Common Vulnerabilities and Exposures, o CVE, è un dizionario di vulnerabilità e falle di sicurezza note pubblicamente) che è legata all'esecuzione di codice arbitrario locale [20]. Un utente malintenzionato potrebbe creare un apk che prende di mira un'applicazione specifica e se una vittima dovesse installare questo apk, consentirebbe agli attaccanti di ottenere l'accesso completo a tutte le risorse del dispositivo Android. Le app Android sono relativamente più facili da decodificare rispetto alle app native in un ambiente desktop, come gli eseguibili Windows e UNIX, perché i file bytecode Dalvik indipendenti dall'hardware conservano una grande quantità di informazioni delle sorgenti Java originali [11]. Un altro studio riguarda le librerie di terze parti e come potrebbero potenzialmente minacciare gli aspetti di sicurezza del software del codice sorgente. Un esempio di vulnerabilità per una applicazione Android è Insecure Content Provider Access che fa parte del CWE-284 (controllo di accesso improprio, l'acronimo CWE sta per "Common Weakness Enumeration" ed indica un sistema di categorie per punti deboli e vulnerabilità hardware e software) : i dati di un'applicazione sono privati, quindi non è possibile per un'applicazione accedere ai dati di un'altra per impostazione predefinita. Quando le applicazioni desiderano condividere i propri dati con altre si utilizza il Content Provider funge da interfaccia per la condivisione dei dati. I provider di contenuti utilizzano i metodi standard (come ad esempio Create, Read, Update, Delete) per accedere ai dati dell'applicazione. A ciascun provider viene assegnata una forma speciale di URI che inizia con " content:// ". Qualsiasi app che conosce questo URI può inserire, aggiornare, eliminare ed eseguire query sui dati dal provider. Potrebbero verificarsi alcuni casi in cui i provider non possono essere implementati per la condivisione dei dati con altre app o lo sviluppatore potrebbe voler concedere l'accesso solo alle app che dispongono delle autorizzazioni appropriate. In tali casi, se nell'app non vengono applicati controlli di sicurezza adeguati, ciò porta alla perdita di informazioni.

2.1.3 Analisi statica, dinamica ed ibrida

L'analisi di un codice può essere di tre tipi: statica, dinamica ed ibrida [18].

L'Analisi statica consiste nell'analizzare il codice sorgente dell'applicazione o il bytecode, cioè è l'insieme di tecniche che permettono l'analisi di un file sospetto senza metterlo in esecuzione.

Per analisi dinamica si intendono quell'insieme di tecniche che prevedono la messa in esecuzione in un ambiente controllato di un'app per cui si sospettano delle vulnerabilità.

La tecnica di analisi ibrida contiene le caratteristiche delle tecniche di analisi sia statiche che dinamiche. Pertanto, questo approccio può analizzare il codice sorgente e il comportamento in fase di esecuzione dell'applicazione.

2.1.4 Google Play Store

Il Play Store di Google è una piattaforma digitale per la distribuzione di applicazioni Android. Google Play non è una piattaforma ben controllata ed è una delle principali fonti di applicazioni vulnerabili[11]. È possibile trovare dei malware in app gratuite o versioni gratuite di app commerciali su app store di terze parti. L'utente è indotto a installare l'applicazione con contenuti dannosi. Ad esempio, gli annunci pubblicitari eseguiti tramite un'applicazione possono essere utilizzati per indurre gli utenti a installare applicazioni dannose [11]. Una volta che Google ha rilasciato una versione del sistema operativo, il produttore o gli operatori delle app potrebbero non aggiornare le loro applicazioni al nuovo sistema, quindi un dispositivo potrebbe avere dei problemi di sicurezza ed instabilità.

2.1.5 Librerie API e file APK

API è l'acronimo di "Application Programming Interface" è una interfaccia che funge da intermediario per la comunicazione di due applicazioni. In pratica, esse sono indispensabili per rendere disponibili i servizi in risposta alle esigenze di business, grazie alle Api si può rendere fruibile via mobile un'applicazione enterprise (aziendale). Il significato di API, nello sviluppo, è dunque quello di semplificare la possibilità di integrazione tra un'applicazione e un'altra evitando ridondanze e

inutili repliche di codice. L'acronimo APK sta per "Android Package". Un file con estensione .apk è un file creato come file eseguibile utilizzando un IDE come ad esempio Google Android Studio e può essere caricato su Google Play Store per essere scaricato e installato dagli utenti finali. I file APK possono essere generati e resi disponibili per l'installazione manuale anche prima della pubblicazione su Google Play Store. Questo aiuta a testare le funzionalità e il comportamento del pacchetto APK prima di pubblicarlo. Pertanto, è necessario essere sicuri che il file APK provenga da una fonte attendibile e non contenga malware.

Uno sviluppatore utilizza il formato APK per comprimere dei file in un unico archivio, che inoltre ne riduce le dimensioni e li rende più facili da trasmettere e gestire; inoltre all'interno troviamo la versione SDK(Software Development Kit) usata dallo sviluppatore. SDK è un insieme di strumenti che possono essere utilizzati per creare applicazioni software destinate a una piattaforma specifica. L'APK contiene tutte le informazioni di un'app Android, tra cui la versione SDK di destinazione.

2.2 Stato dell'arte

2.2.1 Problema in analisi

Secondo lo studio [8] la sicurezza della comunicazione e la privacy dei dati sono la massima importanza nello sviluppo delle applicazioni. Eppure, regolarmente, ci sono segnalazioni di attacchi riusciti rivolti agli utenti Android. Molti di questi attacchi riguardano direttamente il codice a livello di applicazione scritto da un ampio gruppo di sviluppatori con esperienze di studio diverse. Per ridurre questi attacchi esistono dei tool che svolgono le analisi sulle applicazioni identificando le vulnerabilità all'interno del codice. Un esempio di vulnerabilità che i tool identificano facilmente è "SQL Injection" (CWE-89); fa parte delle 25 vulnerabilità più pericolose del 2022. Questa vulnerabilità è causata da una distrazione del programmatore che non svolge un controllo sulla sintassi SQL negli input controllabili dall'utente. Lo sviluppatore non svolgendo i dovuti controlli di sicurezza adeguati permette all'attaccante di alterare la logica della query per inserire istruzioni aggiuntive non previste, includendo l'esecuzione di comandi di sistema.

2.2.2 Approccio con il dataset AndroZoo

L'approccio utilizzato dal paper [8] è quello di esplorare dei legami di vulnerabilità nell'ecosistema delle app Android. Questo lavoro ha lo scopo di evidenziare le tendenze nel panorama della vulnerabilità, ottenere informazioni su cui la comunità può basarsi e fornire analisi quantitative per supportare la ricerca e la pratica nell'affrontare le vulnerabilità a differenza di questa tesi che si concentra principalmente sulla comprensibilità dell'output generato dai tool. Il paper utilizza il dataset AndroZoo che possiede una vasta raccolta di file.apk inclusi in 14 Android Market (compreso Google Play Store). Una volta estrapolato un dataset da AndroZoo, il paper analizza gli APK attraverso l'utilizzo di tre tool: *FlowDroid*, *AndroBugs* ed *IC3*. Attraverso un'analisi statica raccoglie il numero di vulnerabilità e ne fa una statistica generale identificando quelle più comuni ma non specificando se l'output fa riferimento a sezioni di codice dell'applicazione analizzata.

2.2.3 Approccio con l'analisi statica

L'approccio utilizzato dal paper [4] è quello di svolgere una analisi statica su un dataset di programmi scritti in java. Il paper utilizza come dataset dei dati *Juliet*¹ che contiene applicazioni scritte in java e C/C++. Il paper analizza il dataset attraverso l'utilizzo di 5 tool di cui 3 analizzano applicazioni C/C++ mentre i restanti 2 tool analizzano app Java. L'analisi consiste nel raccogliere il numero di vulnerabilità identificate da ogni tool per poi andare a stabilire la percentuale di precisione (analizzando quanti falsi positivi identifica un tool), a differenza di questa tesi che identifica invece quale è la vulnerabilità più comune.

¹<https://samate.nist.gov/SARD/test-suites>

3.1 Metodologia di ricerca

La metodologia di ricerca adottata per questa tesi si è concentrata principalmente sulla lettura e studio di articoli che trattano le vulnerabilità su sistema Android, estrapolando informazioni sui tool di identificazione di vulnerabilità Android disponibili. Un primo passo è stato quello di analizzare gli articoli e siti web scritti in lingua inglese per cercare tool da configurare ed intuibili nell'utilizzo. Come strumento per la ricerca degli articoli è stato utilizzato principalmente Google Scholar ¹ che è un database di articoli attraverso l'inserimento della parola chiave d'interesse nella stringa apposita. Inizialmente la ricerca è avvenuta sui vari database ricercando parole chiavi come "android vulnerabilities" oppure "android tool"; successivamente sono state aggiunte delle ulteriori keyword in base alle parole chiavi identificate dagli autori. Di seguito ci sono le query usate per la ricerca:

- Android Vulnerability Detection;
- Vulnerability Android;
- Android Vulnerability Analysis;
- Tool Android;

¹<https://scholar.google.com/>

- Tool APK Android;
- APK Tool;
- Android APK Tool;
- Tools Vulnerabilities Android;

Gli articoli considerati per lo studio hanno una data di pubblicazione non inferiore al 1 gennaio 2015 per evitare di analizzare dati troppo vecchi e quindi obsoleti. Sono stati stabiliti dei criteri per indicare la qualità generale di un articolo: si è guardato generalmente la qualità della scrittura verificando come è stato scritto a livello grammaticale, se utilizza un inglese di medio-alto livello e se fornisce dei riferimenti esterni (altri articoli o link a siti web) dei tool che ha menzionato. Le informazioni inerenti alla sicurezza informatica come i tipi di attacchi ad una rete (ad esempio battery-drain e DoS) [13] o i metodi di protezione di una rete sono stati ignorati per incentrare l'analisi sulla parte delle vulnerabilità software delle applicazioni Android e degli strumenti di analisi.

3.2 Approccio con i Tool

Sono stati stabiliti i seguenti criteri per la selezione dei tool:

- Il tool deve essere stato menzionato in un articolo analizzato;
- Il tool deve essere accessibile pubblicamente;
- Il tool deve essere fruibile in modalità gratuita o tramite licenze di prova gratuite;
- Il tool deve prendere in input una o più applicazioni Android da analizzare;
- Il tool deve restituire i risultati in formato human o machine-readable, riportanti le vulnerabilità presenti nelle applicazioni analizzate;
- Il tool deve essere configurabili sul sistema nel quale sono stati svolti i test;
- Il tool deve svolgere almeno l'analisi statica o dinamica;

- Il tool deve presentare una documentazione ufficiale.

Di seguito sono stati inseriti i tool degli articoli e siti web analizzati:

- *Flowdroid*² [17] ;
- *Mobsf*³ [5] ;
- *QARK*⁴ [11] ;
- *Androbugs* [14];
- *virustotal*⁵ [16] ;
- *pithus*⁶ [3] ;
- *yaazhini*⁷ [2] .

Una volta identificati i tool l'obiettivo è stato quello di verificare il loro funzionamento leggendo la documentazione fornita dall'autore del tool e configurandolo per buildare l'app se necessario. Le documentazioni ufficiali dei tool sono state individuate nel file spesso chiamato *readme* che si trova nel percorso di installazione del tool oppure nei siti ufficiali nel caso di una web application. Nella documentazione si scartano tutte le informazioni inerenti alla configurazione del tool per un sistema operativo diverso dalla macchina su cui si testa lo strumento. Il sistema utilizzato per la sperimentazione dei tool utilizza il sistema operativo Ubuntu versione 22.10, 8gb ram ed un processore intel i5 di 10th generazione con architettura x86-64. Per essere ammesso alla parte successiva dello studio ogni tool è stato lanciato con diverse configurazioni finché non completavano un'esecuzione senza eccezioni che lo arrestino. Per questa fase sono allocati massimo 3 giorni per ciascun tool. In breve, se dopo 3 giorni non si è ancora scoperto il modo di eseguire il tool senza errori, verrà estromesso dallo studio.

²<https://github.com/secure-software-engineering/FlowDroid/blob/develop/README.MD>

³<https://mobsf.github.io/docs>

⁴<https://github.com/linkedin/qark/blob/master/README.rst>

⁵<https://support.virustotal.com/hc/en-us/categories/360000162878-Documentation>

⁶<https://beta.pithus.org/>

⁷<https://www.vegabird.com/docs/yaazhini/settings/>

Per la prova di esecuzione è stata utilizzata una applicazione giocattolo ossia una app di cui sono note le vulnerabilità che andrà in input ai tool come “prova” per verificare l’affidabilità dei tool e capire se quest’ultimi possono essere ammessi allo studio. Dopo aver effettuato una approfondita ricerca ed aver ottenuto una lista di applicazioni vulnerabili sul sito *Hacknopedia*⁸, un blog sull’informatica che si concentra principalmente sulla sicurezza delle informazioni, la scelta è ricaduta su *AndroGoat* in quanto tra i risultati ottenuti dalla ricerca è quello che presenta una ricca lista di vulnerabilità elencate, in totale ventiquattro:

⁸<https://hacknopedia.com/2022/12/15/vulnerable-android-application-list/>

Vulnerabilità	Count
CWE-829 Root Detection	1
CWE-78 Emulator Detection	1
CWE-922 Insecure Data Storage - Shared Prefs	2
CWE-922 Insecure Data Storage - SQLite	1
CWE-922 Insecure Data Storage – Temp Files	1
CWE-922 Insecure Data Storage – SD Card	1
CWE-524 Keyboard Cache	1
CWE-778 Insecure Logging	1
CWE-20 Input Validations – XSS	1
CWE-20 Input Validations – SQLi	1
CWE-20 Input Validations – WebView	1
CWE-926 Unprotected Android Components – Activity	1
CWE-926 Unprotected Android Components – Service	1
CWE-926 Unprotected Android Components – Broadcast Receivers	1
CWE-798 Hard coding issues	1
CWE-319 Network intercepting – HTTP	2
CWE-319 Network intercepting – Certificate Pinning	1
CWE-16 Misconfigured Network_Security_Config.xml	1
CWE-489 Android Debuggable	1
CWE-530 Android allowBackup	1
CWE-939 Custom URL Scheme	1
CWE-327 Broken Cryptography	1

Tabella 3.2: Vulnerabilità della applicazione AndroGoat.

L'applicazione è stata realizzata da Satish Patnayak⁹ ed è la prima app vulnerabile sviluppata utilizzando Kotlin¹⁰ ossia un linguaggio di programmazione sviluppato dall'azienda di software JetBrains. Kotlin si basa sulla JVM (Java Virtual Machine) ed è ispirato ad altri linguaggi di programmazione tra i quali Scala e lo stesso Java [9]. Tutto questo è servito a capire se il tool presentasse delle eccezioni oppure impiegasse più di 3 giorni ad essere eseguito.

3.3 Dataset

3.3.1 AndroZoo

AndroZoo è una raccolta crescente di applicazioni Android raccolte da diverse fonti come ad esempio Google Play Store.

AndroZoo fornisce il dataset per dare un contributo agli studi di ricerca in corso, nonché per consentire nuovi potenziali argomenti di ricerca sulle app Android.

Rilasciando il dataset alla comunità di ricerca, **AndroZoo** punta a incoraggiare i ricercatori a impegnarsi in esperimenti riproducibili. Per ricevere il dataset bisogna inviare un'e-mail di domanda a *androzoo@uni.lu* indicando il nome del proprio istituto di ricerca e il nome della persona che richiede l'accesso. Per scaricare gli APK, AndroZoo mette a disposizione due metodi: Il primo è attraverso l'utilizzo di un url specifico ¹¹. All'interno dell'url si trovano le variabili *APIKEY* e *sha256* da settare. Sostituendo *APIKEY* con la product key che AndroZoo mette a disposizione per i ricercatori ed *sha256* con il codice sha256 dell'apk che si vuole scaricare, è possibile scaricare 1 APK alla volta dal dataset.

Il secondo comando può essere eseguito installando AndroZoo da terminale. Per installare AndroZoo bisogna avere Python 3.6 o superiore sul proprio pc. L'ultima versione di python (3.11.1) ha dei problemi con l'installazione di AndroZoo.

Dopo aver configurato AndroZoo, si può scaricare una raccolta di APK con le stesse caratteristiche attraverso l'utilizzo di una query. Di seguito è riportato un esempio di query scritta sul terminale:

```
az -n 10 -d 2015-12-11: -s :3000000 -m play.google.com.
```

⁹<https://github.com/satishpatnayak>

¹⁰<https://github.com/satishpatnayak/AndroGoat>

¹¹<https://androzoo.uni.lu/api/download?apikey=APIKEY&sha256=SHA256>

Ciò significa: scarica 10 APK con data di caricamento a partire dal 2015-12-11 (incluso), dimensione fino a 3000000 byte (incluso) e presente su Google Play Store.

La query utilizzata per lo studio:

`az -n 30 -d 2015-12-01: -s :1000000 -m play.google.com.` Questa query scarica 30 apk a partire dal 01/12/2015 con dimensione fino a 100000 byte e presente su Google Play Store. In questo studio la query ha generato gli apk rappresentati nella Tabella 3.3:

Tabella 3.3: Lista dei 30 APK.

Nome	Size	Nome	Size
ae.ephem.apk	600,8kB	com.nudge.me.DispF.apk	462,4kB
app.gurumeditation.radarpioggia.apk	134,0 kB	com.obnsoft.arduboyemu.apk	285,5 kB
calendario.semene.zero.apk	142,3 kB	com.waxrain.telnetd	735,7 kB
com.aboworks.fffplanner.apk	296,1 kB	eq3.zero.apk	160,3 kB
com.agoraefi.mobile.apk	949,0 kB	gui.loadcomph.apk	212,9 kB
com.babelprov.bnnp.apk	342,2 kB	keyboard.theme.k820000077.apk	795,4 kB
com.beautyprolink.ccoyle	492,5 kB	keyboard.theme.k820004274.apk	806,5 kB
com.chinesemenuonline.chinastarhighlandpark.apk	259,5 kB	me.psycoder.apkextractor.apk	968,9 kB
com.freestart.a1dogtraining.appyli.apk	71,0 kB	org.atari.montezuma.sio2bt.apk	80,8 kB
com.freestart.vincechalkhomeimprovements.appyli.apk	73,5 kB	org.visibleimpact.android.fieldapp.apk	949,9 kB
com.LaguIwanFalsLawasmp3.AlzenaApps	945,1 kB	rock.paper.scissors.lizard.spock.apk	489,7 kB
com.mazesystem.fusen.apk	709,7 kB	ru.com.vsedlyalubitei.futbola.apk	176,9 kB
com.nAppsIslamic.obereganie.AOVSTCVWKALGHRBO.apk	492,5kB	se.jensp.hastighetsmatare.apk	406,5 kB
com.neptunetg.r900rfest.apk	988,0 kB	sweethut.holiday.sweethutholiday.apk	410,2 kB
com.nomagic.colorrings.apk	603,9 kB	th.berm.kodi.boot	16,3 kB

3.3.2 Tool selezionati

Una volta definita la lista dei tool nella sezione 3.2 è stata fatta una selezione da quelli trovati durante la ricerca. Questi tool sono stati scelti seguendo i criteri scritti nel paragrafo precedente e rispettando le regole presenti nella fase di configurazione. Di seguito sono riportati i tre tool usati nello studio:

- MobSF [5] , una Web Application ¹² ;
- Pithus [3] , una Web Application ¹³ ;
- Yaazhini [2], una Windows application ¹⁴ .

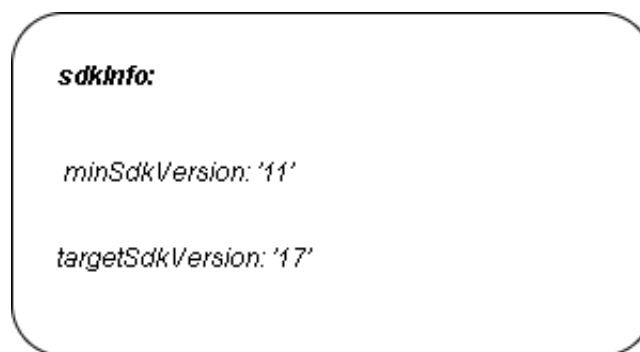
¹²<https://mobsf.live/>

¹³<https://beta.pithus.org/>

¹⁴<https://www.vegabird.com/yaazhini/>

3.4 Apktool

Per gestire la compatibilità dell'applicazione con una o più versioni della piattaforma, Android consente alle app di dichiarare le versioni dell'SDK della piattaforma supportate nei propri file *manifest* [21]. I file *manifest* si presentano come file xml non leggibili se non vengono decompilati. *Apktool* è uno strumento per effettuare la decompilazione dei file Android. Permette di disassemblare l'app e, cosa importante, permette anche di ricostruire il file APK dopo aver fatto eventuali modifiche. Con il comando `apktool d app.apk`, si decodificano i file binari che verranno riportati in una cartella apposita. All'interno della cartella nel file *apktool.yml* si trovano tutte le informazioni sulla versione SDK usata nella voce *sdkInfo*. In *sdkInfo* sono presenti le voci *minSdkVersion* e *targetSdkVersion* come è rappresentato nella figura 3.1 .

A screenshot of a code block with a rounded border. It shows the *sdkInfo* section of the *apktool.yml* file. The text is as follows:

```
sdkInfo:  
  
minSdkVersion: '11'  
  
targetSdkVersion: '17'
```

Figura 3.1: Esempio di codice presente nel file *apktool.yml*.

minSdkVersion è un numero intero che designa il livello API minimo richiesto per l'esecuzione dell'applicazione, in questo caso è 11.

targetSdkVersion invece specifica il livello API su cui l'applicazione è progettata per essere eseguita. Quindi in questo caso l'applicazione è progettata per essere eseguita sulla versione Android 4.

Tabella 3.4: Tabella delle versioni Android con i rispettivi SDK usati.

Versioni Android	SDK/API level
Android 13	Level 33
Android 12	Level 32 Level 31
Android 11	Level 30
Android 10	Level 29
Android 9	Level 28
Android 8	Level 27 Level 26
Android 7	Level 25 Level 24
Android 6	Level 23
Android 5	Level 22 Level 21
Android 4	Level 20 Level 19 Level 18 Level 17 Level 16 Level 15 Level 14
Android 3	Level 13 Level 12 Level 11
Android 2	Level 10 Level 9 Level 8 Level 7 Level 6 Level 5
Android 1	Level 4

Come è rappresentato nella Tabella 3.4 ad ogni versione Android corrisponde un SDK/API level quindi nell'esempio fatto precedentemente se io ho *minSdkVersion: '11'* vuol dire che l'APK supporta le librerie API fino alla versione Android 3. Decompilando attraverso *Apktool* il Dataset preso da *AndroZoo* si ottengono i parametri *minSdkVersion* e *targetSdkVersion* e quindi le versioni Android compatibili con ogni APK.

3.5 Analisi dei tool

L'analisi eseguita dai tre tool definiti precedentemente è stata sulla comprensibilità dell'output che quest'ultimi generano. Ogni tool ha analizzato i 30 APK del dataset AndroZoo e si sono raccolte delle informazioni sul comportamento dei tool rispetto a tre parametri:

- Il numero di vulnerabilità trovate dai tool;
- Le vulnerabilità più comuni trovate durante tutto il periodo di analisi;
- Quali eccezioni vengono generate verificando se quest'ultime fossero abbastanza esplicative (e quindi utili) per puntare a problemi identificati nel codice.

Tabella 3.5: Esempio di raccolta dei dati.

Tool	Numero di vulnerabilità	vulnerabilità più comune
Tool1	89	CWE-922
Tool2	70	CWE-78
Tool3	16	CWE-922
Tool4	123	CWE-926

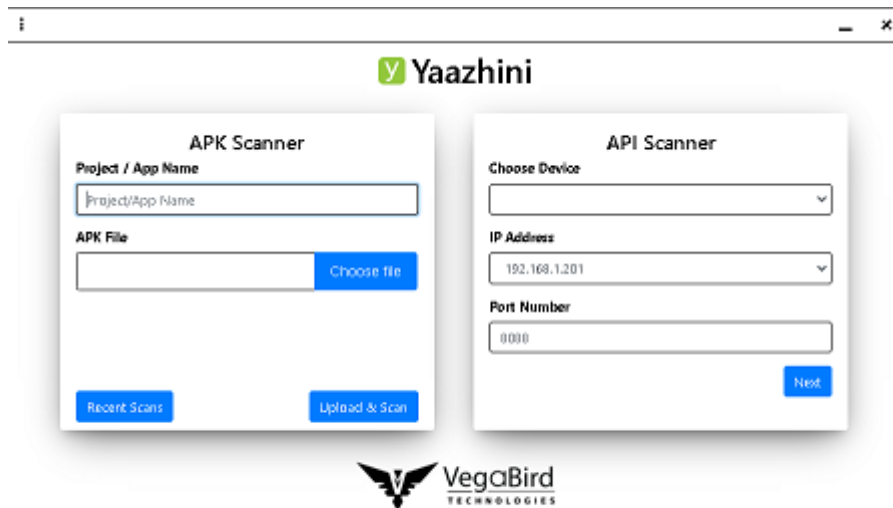
Risultati ottenuti

4.1 Yaazhini

Yaazhini è uno scanner di vulnerabilità intuitivo e gratuito per APK e API Android, progettato e sviluppato esclusivamente per identificare le vulnerabilità a livello di APK ¹. Include il modulo di scansione delle vulnerabilità dell'API, il modulo di scansione delle vulnerabilità dell'APK e il modulo della sezione dei report.

L'interfaccia grafica d'input è molto semplice e come è rappresentato dalla Figura 4.1 si divide in due parti, alla sinistra si trova la sezione per l'analisi dei file .apk, alla destra invece si trova la sezione API Scanner.

¹<https://www.vegabird.com/docs/yaazhini/apk/>



The image shows a web browser window with the Yaazhini logo at the top. Below the logo are two side-by-side input forms. The left form is titled 'APK Scanner' and contains a 'Project / App Name' text field, an 'APK File' text field with a 'Choose file' button, and two buttons at the bottom: 'Recent Scans' and 'Upload & Scan'. The right form is titled 'API Scanner' and contains a 'Choose Device' dropdown menu, an 'IP Address' dropdown menu with the value '192.168.1.201', a 'Port Number' text field with the value '8080', and a 'Next' button. Below the forms is the VegaBird Technologies logo.

Figura 4.1: Interfaccia d’input di Yaazhini.

L’applicazione necessita di una configurazione iniziale in quanto va installata tramite un file eseguibile che si scarica dal sito Vegabird ². Siccome in questa tesi si analizza un dataset di file .apk, si utilizza la sezione di sinistra. Il tool richiede in input un file .apk da analizzare ed il nome del progetto. In output, *Yaazhini* restituisce una selezione delle vulnerabilità che ha individuato in tre livelli: High, Medium e Low. Cliccando su uno dei tre livelli si apre un elenco di nomi di vulnerabilità. *Yaazhini* fornisce una sottosezione chiamata *DETAILS* che espone una descrizione generica della vulnerabilità selezionata ed una sezione chiamata *CODE* che mostra il codice dove è presente la vulnerabilità ed evidenzia in rosso quest’ultima.

²<https://www.vegabird.com/yaazhini/>

4.1.1 Raccolta dati di Yaazhini

L'Analisi di Yaazhini sui 30 APK estrapolati dal dataset AndroZoo ha riportato i seguenti risultati:

Tabella 4.1: L'Analisi di Yaazhini sui 30 APK.

APK	CWE-319	CWE-330	CWE-749	CWE-530	CWE-926	CWE-328	CWE-300	CWE-258	CWE-798	CWE-259
ephem	3	1	1	0	0	0	1	2	0	0
radarpioggia	0	0	1	1	0	0	0	0	0	0
zero	0	1	1	0	0	0	0	0	0	0
fffplanner	2	0	0	0	0	0	0	0	0	0
mobile	70	1	2	0	1	0	0	0	0	0
bnnp	3	0	1	1	0	0	0	0	0	0
ccoyle	3	0	1	1	1	0	0	0	0	0
chinastarhighlandpark	5	0	1	0	0	0	0	0	0	0
appyli	8	0	3	0	0	0	0	0	0	0
vincechal.appyli	8	0	3	0	0	0	0	0	0	0
AlzenaApps	11	0	2	0	0	3	0	0	0	0
fusen	11	16	2	1	5	1	0	0	0	0
AOVSTCVWKGALGHRBO	1	0	0	0	0	0	0	0	0	0
r900rftest	0	3	0	1	0	0	0	0	0	0
colorrings	0	1	0	0	0	0	0	0	0	0
DispF	0	3	1	0	0	0	1	0	0	0
arduboyemu	2	0	0	1	0	0	0	0	0	0
telnetd	4	0	0	0	4	1	0	0	0	0
eq.zero	0	1	1	0	0	0	0	0	0	0
loadcomph	2	3	1	0	0	0	3	3	0	0
k820000077	4	0	0	1	1	2	0	0	2	0
k820004274	4	0	0	1	1	2	0	0	2	0
apkextractor	0	1	0	1	0	1	0	0	0	0
sio2bt	0	0	0	1	0	0	0	0	0	0
fieldapp	0	1	0	1	0	1	0	0	0	1
spock	0	1	0	1	0	0	0	0	0	0
futbola	2	0	1	2	1	0	0	0	0	0
hastighetsmatore	0	0	0	1	0	0	0	0	0	0
sweettholiday	2	1	1	0	0	1	0	0	0	0
boot	0	0	0	1	0	0	0	0	0	0
TOTALE	145	34	23	16	14	12	5	5	4	1

Il numero di vulnerabilità totali identificate da Yaazhini è 259.

Come è rappresentato nella Tabella 4.1 la vulnerabilità più comune identificata da Yaazhini è *Insecure Communication* (CWE-319). Questa vulnerabilità comporta che se un software trasmette dei dati sensibili o critici in un canale di comunicazione, quest'ultimo può essere intercettato da attaccanti non autorizzati. Ciò comporta due conseguenze: la più grave consiste nell'esposizione dei dati di un singolo utente che potrebbe potenzialmente portare al furto dell'account; una seconda conseguenza riguarda l'ambito aziendale e consiste nella violazione della privacy che può sfociare in furto d'identità, frode, o danno reputazionale dell'azienda.

Inoltre l'analisi ha trovato 4 eccezioni:

- **Android external storage**, è una eccezione che segnala un problema di compatibilità dell'app con Android 11, infatti, da quest'ultima versione Android

ha cambiato il modo in cui funzionano le autorizzazioni di file/archiviazione. A partire dall'SDK 41, le app hanno come target Android 11. Per l'SDK 40 e inferiore si include `requestLegacyExternalStorage` in `AndroidManifest` per mantenere la compatibilità con versioni precedenti di Android.

- **WebView with insecure setting – `setAllowUniversalAccessFromFileURLs`**, l'app utilizza il metodo `setAllowUniversalAccessFromFileURLs`.

Questo metodo non è sicuro in quanto utilizza

`androidx.webkit.WebViewAssetLoader` per caricare il contenuto del file. L'abilitazione di questa impostazione consente agli script dannosi caricati in un `file://` di accedere a dati privati dell'app o persino credenziali utilizzate. Per le app con versione Android 4 e precedenti questo valore è impostato su `true` ma è necessario impostare esplicitamente questo valore su `false`.

- **Usage of critical permission - `install packages`**, l'app richiede una autorizzazione di tipo `android.permission.INSTALL_PACKAGES`, l'autorizzazione `INSTALL_PACKAGES` non può essere utilizzata dagli sviluppatori in base alla documentazione di Android Developer ³.

Per risolvere il problema basta utilizzare `REQUEST_INSTALL_PACKAGES`.

³https://developer.android.com/reference/android/Manifest.permission.html#INSTALL_PACKAGES

4.2 MobSF

MobSF è una web application utilizzata per la valutazione della sicurezza e l'analisi del malware delle applicazioni Android. MobSF può analizzare le applicazioni mobile in diversi formati come APK, IPA e APPX [5]. Per eseguire l'analisi statica basta recarsi al link del sito⁴ e caricare il file .apk nella sezione *upload* come è rappresentato nella 4.2.

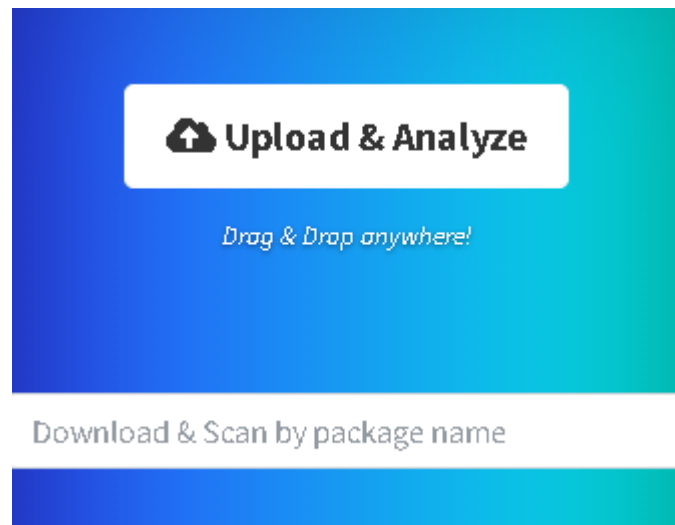


Figura 4.2: Interfaccia d'input di MobSF.

⁴<https://mobsf.live/>

4.2.1 Raccolta dati di MobSF

L'Analisi di MobSf sui 30 APK estrapolati dal dataset AndroZoo ha riportato i seguenti risultati:

Tabella 4.2: L'Analisi di MobSf sui 30 APK.

APK	CWE-532	CWE-276	CWE-330	CWE-89	CWE-327	CWE-200	CWE-295	CWE-649	CWE-749	CWE-919
ephem	10	0	2	1	1	0	1	2	0	0
radarpioggia	0	0	0	0	0	0	0	0	0	0
zero	1	1	0	1	0	0	0	0	0	0
fftplanner	1	2	0	2	0	0	0	0	0	0
mobile	3	0	0	0	0	0	0	0	0	0
bnp	1	2	0	0	0	0	0	0	1	1
cchoyle	4	0	0	0	0	0	0	0	1	1
chinastarhighlandpark	0	0	0	0	0	0	0	0	0	0
applyli	0	0	0	0	0	0	0	0	0	0
vincechal.applyli	0	0	0	0	0	0	0	0	0	0
AlzenaApps	1	0	0	0	0	0	0	0	0	0
fusen	10	12	13	0	1	0	0	0	0	0
AOVSTCVWKGALGHRBO	13	1	0	3	0	0	0	0	0	0
r900rftest	15	0	2	2	0	0	0	0	0	0
colorrings	0	0	0	0	0	0	0	0	0	0
DispF	11	1	1	0	0	1	1	0	0	0
arduboyemu	0	3	0	0	0	0	0	0	0	0
telnetd	7	0	0	0	1	0	0	0	0	0
eq.zero	4	1	1	0	0	0	0	0	0	0
loadcomph	5	1	1	0	0	0	1	0	0	0
k820000077	16	0	1	0	1	0	0	0	0	0
k820004274	16	0	1	0	1	0	0	0	0	0
apkextractor	5	1	1	0	2	6	0	0	0	0
sio2bt	3	2	0	0	0	0	0	0	0	0
fieldapp	31	2	2	2	1	0	0	0	0	0
spock	0	0	1	0	0	0	0	0	0	0
futbola	0	0	0	0	0	0	0	0	0	0
hastighetsmatore	1	0	0	0	0	0	0	0	0	0
sweethutholiday	5	1	1	0	1	1	0	0	0	0
boot	1	0	0	0	0	0	0	0	0	0
TOTALE	164	30	27	11	9	8	3	2	2	2

In totale MobSF ha identificato 258 vulnerabilità.

Come è rappresentato nella Tabella 4.2 la vulnerabilità più comune identificata da MobSF è *Insertion of Sensitive Information into Log File* (CWE-532). Questa vulnerabilità comporta che l'app registra le informazioni sensibili. Sebbene la registrazione di tutte le informazioni possa essere utile durante le fasi di sviluppo, è importante che i livelli di registrazione siano impostati in modo appropriato prima della spedizione di un file.apk, in modo che i dati degli utenti e le informazioni di sistema non vengano esposti accidentalmente a potenziali attaccanti.

I file di registro si dividono in:

- *File di registro del server* (ad es. server.log). Quest'ultimo può fornire nomi di percorsi completi, informazioni di sistema, e talvolta nomi utente e password.

- *File di registro* utilizzati per il debug, sono sempre file con estensione .log. Quest'ultimi sono utilizzati da tutti i tipi di software e sistemi operativi per tenere traccia di qualcosa che si è verificato, salvando tutte le informazioni dell'evento avvenuto nel sistema, come ad esempio la data e l'orario. Un programma di backup di file potrebbe utilizzare un file LOG per rivedere un precedente processo di backup, leggere gli eventuali errori riscontrati o vedere dove è stato eseguito il backup dei file.

Inoltre l'analisi ha trovato 3 eccezioni che si sono ripetute in più occorrenze:

- **App can be installed on a vulnerable Android version**, è una eccezione comparsa diverse volte durante la fase di testing. L'eccezione segnala che il programma ha un *minSdk* troppo basso ed il tool consiglia di utilizzare un *minSdk* con versione Android > 8 - API 26, in quanto da quest'ultima in poi si ricevono aggiornamenti di sicurezza più importanti.
- **Application Data can be Backed up [android:allowBackup] flag is missing**. Il flag [android:allowBackup] deve essere impostato su false. Siccome l'eccezione segnala che non è stato possibile trovare il flag, per impostazione predefinita è impostato su true e consente a chiunque di eseguire il backup dei dati dell'applicazione tramite adb(Android Debug Bridge è uno strumento compreso all'interno del software SDK e usato per mettere in comunicazione un dispositivo Android ed un computer). Consente agli utenti che hanno abilitato il debug USB di copiare i dati dell'applicazione dal dispositivo.
- **Broadcast Receiver is Protected by a permission, but the protection level of the permission should be checked**. Un Broadcast Receiver è una componente Android che consente a un'applicazione di rispondere ai messaggi trasmessi dal sistema operativo Android o da un'applicazione, normalmente è protetto da un permesso. L'eccezione segnala che il permesso non è definito nell'applicazione analizzata. Di conseguenza, il livello di protezione dell'autorizzazione dovrebbe essere controllato dove è definito altrimenti un'applicazione dannosa può richiedere e ottenere l'autorizzazione.

4.3 Pithus

Pithus è una web application in fase di beta che ha come compito principale quello di analizzare le applicazioni Android. Per svolgere le analisi Pithus utilizza diversi tools: APKiD3, ssdeep4, Dexofuzzy, QuarkEngine, AndroGuard7, MobSF, Exodus-core9. Non appena viene caricato un file.apk, i tool svolgono le analisi e Pithus raggruppa i risultati.

Pithus si presenta con una interfaccia semplice nella quale cliccando sul bottone upload è possibile inserire in input un file.apk con una dimensione massima di 65.3 MB come è rappresentato nella Figura 4.3. È possibile accedere al tool facilmente dal link del sito⁵.

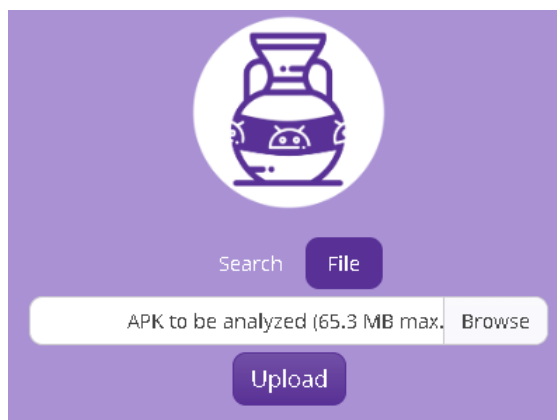


Figura 4.3: Interfaccia d’input di Pithus.

⁵<https://beta.pithus.org/>

4.3.1 Raccolta dati di Pithus

L'Analisi di Pithus sui 30 APK estrapolati dal dataset AndroZoo ha riportato i seguenti risultati:

Tabella 4.3: L'Analisi di Pithus sui 30 APK.

APK	CWE-532	CWE-276	CWE-330	CWE-327	CWE-312	CWE-89	CWE-295	CWE-200	CWE-749	CWE-919
ephem	1	1	1	1	0	1	1	0	0	0
radarpioggia	0	0	0	0	0	0	0	0	0	0
zero	1	1	1	0	0	0	0	0	0	0
fftplanner	1	1	0	0	0	1	0	0	0	0
mobile	1	0	0	0	0	0	0	0	0	0
bnp	1	1	0	0	0	0	0	0	1	1
cchoyle	1	0	0	0	0	0	0	0	1	1
chinastarhighlandpark	0	0	0	0	0	0	0	0	0	0
appyli	0	0	0	0	1	0	0	0	0	0
vincechal.appyli	0	0	0	0	1	0	0	0	0	0
AlzenaApps	1	0	0	0	0	0	0	0	0	0
fusen	1	1	1	1	1	0	0	0	0	0
AOVSTCVWKGALGHRBO	1	1	0	0	1	1	0	0	0	0
r900rftest	1	1	1	0	0	1	0	0	0	0
colorrings	0	0	1	0	0	0	0	0	0	0
DispF	1	1	1	0	0	0	1	1	0	0
arduboyemu	0	1	0	0	0	0	0	0	0	0
telnetd	1	0	0	1	0	0	0	0	0	0
eq.zero	1	1	1	0	0	0	0	0	0	0
loadcomph	1	1	1	0	0	0	1	0	0	0
k820000077	1	0	1	1	0	0	0	0	0	0
k820004274	1	0	1	1	0	0	0	0	0	0
apkextractor	1	1	1	1	0	0	0	1	0	0
sio2bt	1	1	0	0	0	0	0	0	0	0
fieldapp	1	2	1	1	1	1	0	0	0	0
spock	0	0	1	0	0	0	0	0	0	0
futbola	0	0	0	0	0	0	0	0	0	0
hastighetsmatore	1	0	0	0	0	0	0	0	0	0
sweetholiday	1	1	1	1	0	0	0	1	0	0
boot	1	0	0	0	0	0	0	0	0	0
TOTALE	22	16	14	8	5	5	3	3	2	2

In totale Pithus ha rilevato 80 vulnerabilità. Come è rappresentato nella Tabella 4.3 la vulnerabilità più comune identificata da Pithus è *Insertion of Sensitive Information into Log File* (CWE-532). Questa vulnerabilità è la più comune registrata anche per il tool MobSf[5] e comporta che le informazioni sensibili registrate dall'app siano esposte accidentalmente a potenziali attaccanti.

Le eccezioni trovate durante lo svolgimento dell'analisi sono le seguenti:

- **Application Data can be Backed up**, questa eccezione segnala che è consentito a chiunque eseguire il backup dei dati dell'applicazione attraverso l'utilizzo dell'adb (Android Debug Bridge, che è uno strumento compreso all'interno del software SDK e usato per mettere in comunicazione un dispositivo Android ed un computer) .

In particolare nella analisi svolta da Pithus questa eccezione compare in due forme diverse: nella prima forma segnala che il flag [android:allowBackup] è impostato a true ma deve essere impostato a false; nella seconda forma l'eccezione segnala che non riesce a trovare la variabile flag e quindi per impostazione predefinita è impostata a true.

- **Launch Mode of Activity (*nome del task*) is not standard.**, questa eccezione segnala che la variabile *launchMode* non è impostata a standard; infatti, come mostra la documentazione di Android Developers⁶, quest'ultima può avere la seguente sintassi:

```
android:launchMode=["standard" | "singleTop" | "singleTask" | "singleInstance" |
"singleInstancePerTask"]
```

Con "standard" o "singleTop" ci possono essere più istanze di una task invece con "singleTask", "singleInstance" e "singleInstancePerTask" esiste solo una istanza univoca di quella task.

- **Broadcast Receiver is not Protected - [android:exported=true]**, questa eccezione segnala che il Broadcast Receiver è accessibile a qualsiasi altra applicazione sul dispositivo. Il Receiver ha la seguente sintassi:

```
<receiver android:directBootAware=["true" | "false"]
    android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:icon="drawable resource"
    android:label="string resource"
    android:name="string"
    android:permission="string"
    android:process="string" >
... </receiver>
```

⁶<https://developer.android.com/guide/topics/manifest/receiver-element>

Dalla documentazione di Android Developers⁷ si vede che se la variabile *android:exported* è impostata a true, il Broadcast Receiver può ricevere messaggi da fonti non di sistema al di fuori della sua applicazione .

⁷<https://developer.android.com/guide/topics/manifest/receiver-element>

4.4 Bilancio finale

I risultati della Tabella 4.4 mostrano che il tool che ha identificato più vulnerabilità è *Yaazhini* con un numero di vulnerabilità pari a 259.

Tabella 4.4: Raccolta finale dei dati.

Tool	Numero di vulnerabilità	vulnerabilità più comune
Yaazhini	259	CWE-319
MobSF	258	CWE-532
Pithus	80	CWE-532

La vulnerabilità più comune è *Insertion of Sensitive Information into Log File*. Infatti, come si vede dalla Tabella 4.5 sono state trovate un totale di 186 istanze estrapolate da tutti i tool.

Tabella 4.5: Numero di istanze della vulnerabilità più comune

Tool	Insertion of Sensitive Information into Log File
Yaazhini	0
MobSF	164
Pithus	22
Totale	186

Per quanto riguarda le eccezioni, sono risultate abbastanza esplicite in tutti i tool, infatti quasi tutte hanno fatto riferimento a problemi identificati in sezione di codice ben specifiche.

L'eccezione **Application Data can be Backed up**, trovata più volte da diversi tool, per esempio, ha fatto riferimento alla variabile [android:allowBackup] avvertendo l'utente che non è stata trovata oppure che è settata a true (suggerendo all'utente di settarla a false).

L'app con più vulnerabilità analizzata da Yaazhini è *mobile* con un risultato pari a 74. La vulnerabilità apparsa maggiormente è CWE-319 con 70 occorrenze.

L'app con più vulnerabilità analizzata da MobSf è *fieldapp* con un risultato pari a 38. La vulnerabilità apparsa maggiormente è CWE-532 con 31 occorrenze.

L'app con più vulnerabilità analizzata da Pithus è *fieldapp* con un risultato pari a 7. La vulnerabilità apparsa maggiormente è CWE-276 con 2 occorrenze.

Si noti che delle applicazioni che posseggono più vulnerabilità, 2/3 di queste presentano un numero di occorrenze alto nella vulnerabilità più comune del tool utilizzato per analizzarle: ad esempio, l'app *mobile* analizzata con Yaazhini presenta 70 istanze di *Insecure Communication* che è appunto la vulnerabilità più comune.

Si noti inoltre come l'APK *fieldapp* risulti essere l'applicazione con più vulnerabilità quando analizzata con due tool su tre (MobSF e Pithus). In aggiunta, è interessante notare che, dove MobSF identifica trentuno CWE-532, Yaazhini ne individua zero e Pithus soltanto una. Per quanto riguarda la vulnerabilità CWE-276, invece, sia MobSF che Pithus ne riconoscono lo stesso numero di occorrenze, cioè due.

Conclusioni e sviluppi futuri

Questa tesi ha mostrato quanto siano importanti le vulnerabilità e come queste possano permettere ad attaccanti di rubare dati importanti o mettere fuori uso un dispositivo mobile. L'obiettivo principale di questo studio è stato di comprendere come le vulnerabilità possono essere individuate attraverso l'utilizzo di tool che svolgono analisi statiche, dinamiche ed ibride andando a segnalare tutte le vulnerabilità contenute all'interno di un'applicazione Android. Questo lavoro apre molte porte di sviluppo essendo come punto di partenza per altri studi, può essere ampliato analizzando un dataset più grande di applicazioni Android per rendere più precise le statistiche finali. Uno dei lavori futuri più importanti potrebbe essere l'inserimento di analisi più approfondite sulle vulnerabilità ottenute dai tool, fornendo una classifica rispetto alla loro pericolosità nel caso l'utente ignorasse quest'ultime. Un altro importante sviluppo potrebbe essere quello di raccogliere informazioni su ulteriori tool che svolgono l'analisi degli APK ed utilizzando lo stesso dataset di questo studio, si potrebbero confrontare i risultati ottenuti con quelli di questa tesi.

Bibliografia

- [1] Le 20 principali vulnerabilità di sicurezza di sans nelle applicazioni software, available at <https://ita.myservername.com/sans-top-20-security-vulnerabilities-software-applications>. (Citato a pagina 2)
- [2] Yaazhini - free android apk api vulnerability scanner, available at https://idiopathic24.rssing.com/chan-13017459/all/_p147.html. (Citato alle pagine 13 e 18)
- [3] Investigating android malware with pithus, the website is available at <https://cryptax.medium.com/investigating-android-malware-with-pithus-17d2143cc528>, July 2021. (Citato alle pagine 13 e 18)
- [4] Midya Alqaradaghi, Gregory Morse, and Tamás Kozsik. Detecting security vulnerabilities with static analysis—a case study. *Pollack Periodica*, 2021. (Citato a pagina 10)
- [5] Patrick C. K. Hung Farkhund Iqbal Liaqat Ali Benjamin Yankson, Javed Vali. Security assessment for zenbo robot using drozer and mobsf frameworks. (Citato alle pagine 13, 18, 26 e 30)
- [6] Andrea Canton. L'italia è il paese dei cellulari: 1,3 smartphone per ogni abitante, available at <https://www.avvenire.it/speciali/pagine/l-italia-e-il-paese-dei-cellulari-1-3-smartphone-per-ogni-abitante>. May 2022. (Citato a pagina 2)

-
- [7] Paul Cormier. The state of enterprise open source, available at <https://www.redhat.com/en/enterprise-open-source-report/2022>. (Citato a pagina 2)
- [8] Jun Gao, Li Li, Pingfan Kong, Tegawendé F Bissyandé, and Jacques Klein. Understanding the evolution of android app vulnerabilities. *IEEE Transactions on Reliability*, 70(1):212–230, 2019. (Citato alle pagine 9 e 10)
- [9] Dmitry Jemerov and Svetlana Isakova. *Kotlin in action*. Simon and Schuster, 2017. (Citato a pagina 16)
- [10] Jamil Khan and Sara Shahzad. Android architecture and related security risks. 05:14, 2015. (Citato a pagina 6)
- [11] Keyur Kulkarni and Ahmad Y Javaid. Open source android vulnerability detection tools: A survey, *arXiv*. 2018. (Citato alle pagine 1, 2, 5, 7, 8 e 13)
- [12] Jonathan Levin. Dalvik and art, *Andevcon*, 2015. (Citato a pagina 6)
- [13] Alessio Merlo Mauro Migliardi. Dispositivi mobili: nuovi problemi di sicurezza, available at <https://mondodigitale.aicanet.net/ultimo/index.xml>. (Citato a pagina 12)
- [14] Fadi Mohsen, Loran Oosterhaven, and Fatih Turkmen. Kotlindetector: Towards understanding the implications of using kotlin in android applications. *CoRR*, abs/2105.09591, 2021. (Citato a pagina 13)
- [15] Heng Yin Mu Zhang. Appsealer: Automatic generation of vulnerability-specific patches for preventing component hijacking attacks in android applications. (Citato a pagina 7)
- [16] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *Proceedings of the Internet Measurement Conference*, pages 478–485, 2019. (Citato a pagina 13)
- [17] Lina Qiu, Yingying Wang, and Julia Rubin. Analyzing the analyzers: Flow-droid/iccta, amandroid, and droidsafe. In *Proceedings of the 27th ACM SIGSOFT*

- International Symposium on Software Testing and Analysis*, pages 176–186, 2018. (Citato a pagina 13)
- [18] Janaka Senanayake, Harsha Kalutarage, Mhd Omar Al-Kadri, Andrei Petrovski, and Luca Piras. Android source code vulnerability detection: A systematic literature review. *ACM Comput. Surv.*, 55(9), January 2023. (Citato a pagina 8)
- [19] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR)*, 49(4):1–41, 2017. (Citato a pagina 5)
- [20] Alessia Valentini. Famose applicazioni android mettono a rischio milioni di utenti: che c'è da sapere, available at <https://www.cybersecurity360.it/>. (Citato alle pagine 2 e 7)
- [21] Daoyuan Wu, Ximing Liu, Jiayun Xu, David Lo, and Debin Gao. Measuring the declared sdk versions and their consistency with api calls in android apps. In *Wireless Algorithms, Systems, and Applications: 12th International Conference, WASA 2017, Guilin, China, June 19-21, 2017, Proceedings 12*, pages 678–690. Springer, 2017. (Citato a pagina 19)
- [22] Mu Zhang and Heng Yin. Appsealer: automatic generation of vulnerability-specific patches for preventing component hijacking attacks in android applications. In *NDSS*, 2014. (Citato a pagina 6)

Ringraziamenti

Dedico questo spazio del mio elaborato alle persone che hanno contribuito, con il loro supporto, alla realizzazione dello stesso. Ringrazio il mio relatore Fabio Palomba ed i suoi correlatori Emanuele Iannone e Giammaria Giordano per avermi guidato con grande professionalità e dedizione. Sono grato, inoltre, di aver avuto la possibilità, durante la stesura di questa mia tesi di laurea, di ampliare le mie conoscenze riguardo le materie apprese durante il mio percorso universitario e di esplorare nuovi rami del settore.

Un grazie di cuore al mio collega Eduardo Scarpa, con cui ho condiviso tutto il percorso universitario. Grazie per avermi supportato nei momenti difficili della mia carriera e per essere stato un prezioso compagno durante i pomeriggi trascorsi a studiare insieme.

Ringrazio i miei colleghi ed ormai amici Silvio Venturino ed Umberto Della Monica, con cui ho avuto il privilegio di condividere momenti anche al di fuori della vita universitaria e con cui spero di passarne di nuovi. Sono grato a Silvio, in particolare, per essermi stato sempre accanto; anche nei momenti più tristi e difficili, sei sempre riuscito a farmi sorridere e ti sarò sempre grato per il tuo sostegno.

Ringrazio, inoltre, Nicola Pio Gagliardi, Alessandro Aquino, Catello Staiano, Antonio Romano e Carmine Leo e tutti gli amici dell'Università, i quali sono stati sempre al mio fianco e che sono stati per me un importante sostegno durante il mio

intero percorso. Ricordo perfettamente ogni momento di allegria e divertimento passati con ognuno di voi e, quando mi sono iscritto all'università, mai avrei pensato di trovare delle persone così meravigliose.

Ringrazio Valentina, Alessandro e Giuseppe, anche se abbiamo avuto poche occasioni per incontrarci di persona, ho sempre sentito la vostra presenza e il vostro sostegno con me.

Vorrei inoltre ringraziare nonno Domenico e nonna Rosa, ricorderò sempre con affetto tutto l'amore che mi avete trasmesso nel corso degli anni.

Ringrazio di cuore chi NON ha creduto in me. Grazie a loro ho continuato a lottare ogni giorno, ogni ora della mia vita, fino a diventare la persona che sono oggi.

Ringrazio le persone più importanti della mia vita, mamma, papà e mia sorella Arianna per aver creduto in me ed avermi sostenuto in qualsiasi mia scelta. Siete la mia ancora e vi amo con tutto il cuore.

Infine ringrazio me stesso, per la forza di volontà, l'impegno ed i sacrifici fatti che mi hanno permesso di arrivare fin qui.