



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Identificazione di Vulnerabilità Android mediante Tool: un'Analisi Prestazionale

RELATORE

Prof. Fabio Palomba

Dott. Emanuele Iannone

Dott. Giammaria Giordano

Università degli Studi di Salerno

CANDIDATO

Eduardo Scarpa

Matricola: 0512109503

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

sesa^{lab}
SOFTWARE ENGINEERING
SALERNO

*Ai miei nonni.
A mio zio Donato,
ovunque tu sia, dillo alla luna.*

Sommario

Al giorno d'oggi, lo stile di vita delle persone dipende sempre più dalle applicazioni mobile, come lo shopping, la gestione finanziaria e la "banale" navigazione su internet. Proprio per questo, gli sviluppatori si concentrano principalmente sull'implementazione delle app e sul miglioramento dell'esperienza dell'utente, ignorando i problemi di sicurezza. Dall'ultimo decennio fino ad adesso, gli smartphone sono diventati parte integrante della vita di tutti. Avendo la capacità di gestire molte applicazioni utili, gli smartphone sfoggiano, grazie a bravi sviluppatori, funzionalità impeccabili che portano alla loro crescita esponenziale. Grazie all'enorme base di utenti e all'ampia gamma di funzionalità proposte, queste piattaforme mobile sono diventate una popolare fonte di informazioni per il pubblico. Un pubblico così vasto su questa piattaforma la rende anche un enorme bersaglio per gli attacchi informatici, Android, pur essendo così popolare, non fornisce un framework di sicurezza per difendersi da questi attacchi. Questo studio si pone l'obiettivo di analizzare dei tool che identificano vulnerabilità in un applicazione Android. Lo studio si focalizza principalmente su quanto possano essere prestazionali l'uso di questi tool nell'individuare vulnerabilità software. Durante il lavoro di questa tesi sono stati analizzati alcuni studi svolti in passato in questo ambito e le prestazioni ottenute da quest'ultimi, sulla base di questi studi, ottenendo dei risultati, si è riuscito a ricavarne una classificazione dei tool in base alle loro prestazioni nell'individuare la presenza di vulnerabilità software all'interno una applicazione Android, lo studio si conclude analizzando le prestazioni ottenute lanciando tutti i tool presi in esame su un dataset di applicazioni intenzionalmente vulnerabili.

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 OS	5
2.1.2 Vulnerabilità Software	8
2.1.3 Tool di Analisi di Vulnerabilità	10
2.2 Stato dell'arte	13
2.2.1 Problema in esame	13
3 Metodologia	15
3.1 Metodo di ricerca	15
3.2 Approccio ai tool	17

3.3	Applicazioni e tool selezionati per l'analisi	21
3.3.1	Applicazioni intenzionalmente vulnerabili	21
3.3.2	Tool selezionati	25
3.4	Classi di valutazione	26
3.5	Confronto dei tool	27
4	Analisi dei Risultati	28
4.1	Oversecured	28
4.1.1	Classe di valutazione	30
4.1.2	Margine di errore	30
4.2	Yaazhini	31
4.2.1	Classe di valutazione	32
4.2.2	Margine di errore	32
4.3	MobSF	33
4.3.1	Classe di valutazione	34
4.3.2	Margine di errore	34
4.4	Pithus	35
4.4.1	Classe di valutazione	36
4.4.2	Margine di errore	36
4.5	Risultati ottenuti	37
5	Conclusioni e sviluppi futuri	39
	Bibliografia	40

Elenco delle figure

2.1	Architettura del sistema Android	7
4.1	Interfaccia del tool Oversecured	29
4.2	Interfaccia del tool Yaazhini	31
4.3	Interfaccia del tool MobSF	33
4.4	Interfaccia del tool Pithus	35
4.5	Precisione tool	37

Elenco delle tabelle

3.2	Vulnerabilità dell'app. AndroGoat	20
3.3	Applicazioni intenzionalmente vulnerabili	24
3.4	Esempio tabella di valutazione dei tool selezionati	26
4.1	Tabella di valutazione temporanea. Analisi eseguita sui seguenti tool: Oversecured	30
4.2	Calcolo margine di errore, di tutte le applicazioni, del tool Oversecured	30
4.3	Tabella di valutazione temporanea. Analisi eseguita sui seguenti tool: Oversecured, Yaazhini	32
4.4	Calcolo margine di errore, di tutte le applicazioni, del tool Yaazhini .	32
4.5	Tabella di valutazione temporanea. Analisi eseguita sui seguenti tool: Oversecured, Yaazhini, MobSF	34
4.6	Calcolo margine di errore, di tutte le applicazioni, del tool MobSF . .	34
4.7	Tabella di valutazione finale. Analisi eseguita su tutti i tool selezionati	36
4.8	Calcolo margine di errore, di tutte le applicazioni, del tool Pithus . .	36
4.9	Statistiche descrittive	37
4.10	Classificazione finale dei tool	38

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

Oggigiorno, gli smartphone sono diventati una necessità per tutti. Andando indietro di qualche anno, gli anni '90 hanno visto la crescita nell'uso dei PDA (personal digital assistant) e ci è voluto ben poco a trasformarli in dispositivi mobili, conosciuti popolarmente proprio come smartphone. Gli smartphone Android dominano il mercato della telefonia, un dato ci afferma che c'è stata una crescita annua del 24,9% dal 2011 al 2017 e 1,7 miliardi di dispositivi sono stati prodotti entro il 2017 [43]. Questi dispositivi mobile durante questi anni hanno avuto uno sviluppo importante fino a potersi confrontare con i personal computer [25]. La natura open source e la sua vasta gamma di funzionalità sono la ragione principale del dominio di Android sul mercato degli smartphone. È un sistema operativo attualmente sviluppato da Google ed ha una quota di mercato di circa l'87,5% della quota di mercato globale a partire dal terzo trimestre del 2016 [25], e Google Play, lo store ufficiale per applicazioni Android, conta 2,8 milioni di app da Marzo 2009 fino a Settembre 2022 [4]. In tal senso, essere la piattaforma mobile più popolare la rende un bersaglio attraente per gli attacchi alla sicurezza. Come già detto, l'aumento del numero di applicazioni ha anche amplificato il numero di malware e i "pirati informatici" sono diventati

più innovativi con la progettazione di essi. I malware possono sfruttare il proprio dispositivo mobile e possono accedere ad informazioni personali e sensibili, come ad esempio contatti o messaggi [23]. Android non si limita solo agli smartphone, ma si estende anche alle TV, auto e sistemi di automazione, quindi dal momento in cui il mercato è così vasto e categorie così diverse, gli utenti tendono ad avere fiducia e scelgono di installare applicazioni basandosi sulla sua rappresentazione testuale o su screenshot, concedendo completamente al programma installato ogni tipo di autorizzazione. Store di terze parti come *Slide ME*¹, *F-Droid*², *etc.*, forniscono applicazioni a pagamento o gratuite. Coloro che utilizzano queste ultime diventano prede facili per malware: con l'installazione di queste app "gratuite" è facile lasciare informazioni sensibili agli informatici malintenzionati.

Proprio per questo motivo, uno dei concetti principali dell'ambito della sicurezza informatica è quello di *vulnerabilità software*. Nel particolare, le vulnerabilità facilitano ad un utente malintenzionato l'iniezione di malware all'interno dell'applicazione vulnerabile, dando così accesso a tutte le risorse e dati dell'hosting³.

1.2 Motivazione ed obiettivi

Come già accennato nella sezione precedente, negli ultimissimi anni c'è stata una grande evoluzione ed è per questo motivo che esistono dei tool al fine di identificare applicazioni vulnerabili. La presente tesi presenta un'analisi approfondita sui tool, questi ultimi garantiscono maggiore sicurezza all'interno di applicazioni Android e, inoltre, consentono di capire quale di questi utilizzare per uno specifico problema.

L'obiettivo principale è quello di riuscire ad espandere le conoscenze sui sopracitati tool di analisi di vulnerabilità, confrontandoli e studiandone l'efficienza per la rivelazione di vulnerabilità nei software Android e, di conseguenza, valutare se effettivamente questi ultimi individuano le vulnerabilità che indicheremo mediante un dataset di applicazioni intenzionalmente vulnerabili.

¹<http://slideme.org/>

²<https://f-droid.org/>

³Servizio di rete che consiste nell'allocare su un server delle pagine di un sito web o di un'applicazione web.

1.3 Risultati ottenuti

In questo studio si forniscono il seguente contributo:

- Analisi prestazionali, mediante metriche, dei tool presi in esame;
- Calcolo vulnerabilità individuate delle applicazioni intenzionalmente vulnerabilità dai tool presi in esame;
- Calcolo di statistiche descrittive sui tool presi in esame.

I risultati ottenuti mostrano il tool che presenta le migliori prestazioni tra i tool scelti è Oversecured che rientra in una classe di valutazione B, rileva il 63,75% delle vulnerabilità delle applicazioni utilizzate e presenta una media prestazionale di 0.387, notevolmente più bassa rispetto agli altri tool analizzati.

1.4 Struttura della tesi

All'interno di questa sezione ci sarà la struttura della tesi, la quale è composta da cinque capitoli con le rispettive sotto sezioni.

Il primo capitolo fornisce un'introduzione al lavoro svolto, descrivendo il mondo degli smartphone in generale e alcuni dati che fanno capire quanto siano presi di mira questi ultimi, le motivazioni e gli obiettivi da raggiungere. Infine una breve panoramica dei risultati ottenuti.

Il secondo capitolo oltre a descrivere concetti fondamentali per comprendere a pieno il lavoro svolto, presenta un'analisi di alcune tecniche di detection di vulnerabilità.

All'interno del terzo capitolo si discute della metodologia utilizzata sia per lo studio e l'analisi degli articoli/siti web sia sull'impostazione delle classi di valutazioni e le metriche necessarie per il confronto tra i tool.

Il quarto capitolo è diviso in cinque sezioni: la prima analizza i risultati ottenuti dal tool Oversecured, la seconda i risultati ottenuti dal tool Yaazhini, la terza analizza i risultati ottenuti dal tool MobSF, la quarta analizza i risultati ottenuti dal tool Pithus e la quinta vengono riuniti tutti i risultati ottenuti e commentati.

Nell'ultimo capitolo si riassume il lavoro svolto e si analizza i risultati ottenuti, proponendo diversi spunti di riflessione che potrebbero essere utilizzati in futuro per migliorare il lavoro.

Background e stato dell'arte

2.1 Background

2.1.1 Android OS

Android è un sistema operativo open source per dispositivi mobili costituito da uno stack software che include un sistema operativo, un middleware e applicazioni di base [21]. È la piattaforma più popolare per i dispositivi mobile poiché possiede il 74,5% del mercato¹. C'è stato un aumento significativo nello sviluppo di applicazioni di terze parti da parte di singoli sviluppatori e aziende per rispondere a questo cambiamento di mercato. Le caratteristiche più interessanti di Android sono la sua distribuzione sul mercato di applicazioni open source e senza restrizioni così da consentire agli sviluppatori indipendenti di sviluppare le proprie applicazioni e distribuirle sul mercato [25, 24].

Le applicazioni Android vengono installate come *Android Package* compressi (APK) che includono tutti i file, le librerie, e metadati per le app da eseguire. Inoltre, uno strumento avanzato è Android Debug Bridge (ADB) che può essere utilizzato per installare applicazioni tramite il kit di sviluppo software Android (SDK).

¹<https://gs.statcounter.com/os-market-share/mobile/worldwide>

Analisi architettura Android

Android ha un'architettura di tipo gerarchico, strutturata a layer a complessità crescente dal basso verso l'alto. I layer comprendono un sistema operativo, un insieme di librerie native per le funzionalità core della piattaforma, una implementazione della VM e un insieme di librerie Java.

Al livello più basso troviamo il Kernel Linux. Android utilizza Linux per i driver dei dispositivi, la gestione della memoria, la gestione dei processi e la rete [18, 11].

Il livello successivo contiene le librerie native di Android. Queste si presentano tutte scritte internamente in C/C++, ma richiamate attraverso interfacce Java. In questo livello si trovano il Surface Manager (per la composizione delle finestre), la grafica 2D e 3D, i codec multimediali (MPEG-4, H.264, MP3, ecc.), il database SQL (SQLite) e il motore del browser web nativo (WebKit).

A seguire è presente Android Runtime, che comprende la macchina virtuale Dalvik. Quest'ultima esegue file `dex` convertiti, in fase di compilazione, da file `.class` di Java e Jar standard [11].

Le librerie Java di base fanno parte del runtime di Android. Sono scritte in Java, come tutto ciò che si trova al di sopra di questo livello. Android fornisce un sottoinsieme sostanziale dei pacchetti di Java 5 Standard Edition, tra cui Collections, I/O ecc...

Il livello successivo è il livello Application Framework. Alcune parti di questo tool-kit sono fornite da Google, mentre altre sono estensioni o servizi personalizzati dai produttori. Il componente più importante del framework è l'Activity Manager, che gestisce il ciclo di vita delle applicazioni e un "back-stack" (la pila di attività che costituisce un task) [11].

Infine, il livello superiore è quello delle Applicazioni, dove di solito si trovano le utility del telefono e del browser web. Le applicazioni rappresentano un importante collo di bottiglia per la sicurezza in quanto uno sviluppatore potrebbe accidentalmente inserire delle vulnerabilità al suo interno, di conseguenza verrà posta particolare attenzione ad esse. Le applicazioni Android sono costruite utilizzando blocchi di componenti essenziali, ognuno dei quali esiste come entità propria e svolge un ruolo specifico; ogni elemento è un pezzo unico che contribuisce a definire il comporta-

mento complessivo dell'applicazione. È da notare che alcuni di questi elementi sono il punto di ingresso per gli utenti al fine di interagire con l'applicazione e, in molti casi, si può notare come essi dipendano da altri elementi. Nella Figura 2.1 è mostrato com'è suddivisa l'architettura Android [11].

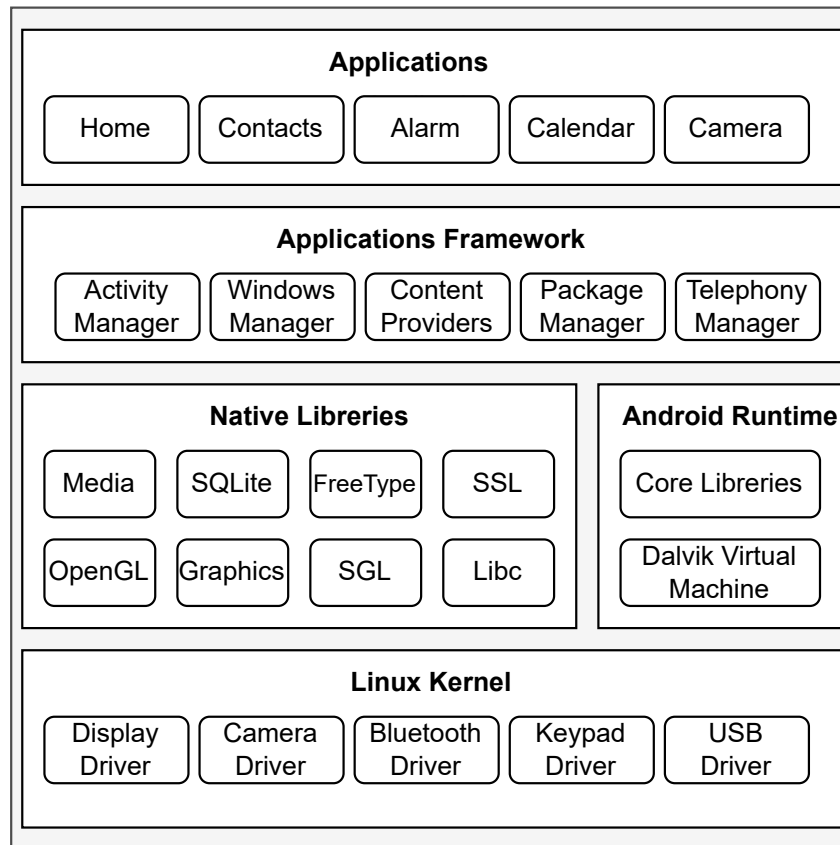


Figura 2.1: Architettura del sistema Android

Il kernel in uso è un kernel della serie Linux 2.6, modificato per esigenze particolari in gestione della batteria, gestione della memoria e l'ambiente di runtime. In alto a destra il kernel esegue alcuni *Linux Daemons* (un processo di servizio che viene eseguito in background e supervisiona il sistema o fornisce funzionalità ad altri processi) come *bluetooth* per il supporto Bluetooth e *wpa_supplicant* per la crittografia WiFi. Poiché Android dovrebbe funzionare su dispositivi con poca memoria principale e CPUs a bassa potenza, vengono compilate le librerie per le attività intensive di CPU e GPU con codice nativo ottimizzato per dispositivo. Sono state sviluppate librerie di base come *libc* o *libm* soprattutto per il basso consumo di memoria e per problemi di licenza su Android [11].

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 informatici ². Un'app vulnerabile con determinate autorizzazioni critiche può eseguire comportamenti sensibili alla sicurezza per conto di un'app dannosa. Questo tipo di vulnerabilità della sicurezza può presentarsi in numerose forme e l'exploit di una o più vulnerabilità che può provocare l'effetto di *privilege escalation* (inteso come oltrepassare delle autorizzazioni), perdite di dati, attacco alle componenti hardware di un dispositivo, ecc.. [33]. Applicazioni come Grindr, OKCupid, Bumble, Moovit, Edge, XRecorder, Viber e Booking, potrebbero risultare essere a rischio attacco a causa di una nota vulnerabilità, identificata come CVE-2020-8913 (Common Vulnerabilities and Exposures, o CVE, è un dizionario di vulnerabilità e falle di sicurezza note pubblicamente), legata all'esecuzione di codice arbitrario locale [40]. Un utente malintenzionato potrebbe creare un APK che prende di mira un'applicazione specifica e, se una vittima dovesse installare quest'applicazione consentirebbe all'attaccante di ottenere l'accesso completo alle 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 [25].

Per centrare meglio il concetto, ad esempio la vulnerabilità *Insecure Data Storage* con CWE-922 (Common Weakness Enumeration, o CWE, è un sistema di categorie per punti deboli e vulnerabilità hardware e software) dove un'applicazione mobile può memorizzare diversi tipi di dati (cookie, file di testo, impostazioni, ecc.) attraverso vari supporti di archiviazione. Il software memorizza informazioni sensibili senza limitare adeguatamente l'accesso in lettura o scrittura da parte di utenti non autorizzati.

La crittografia dei dati sensibili utilizzati nell'applicazione in modo efficiente è una condizione necessaria per garantirne la riservatezza. Se ben progettate, le appli-

²<http://informatica.abaluth.com/sicurezza/attacchi/exploit/>

cazioni in esecuzione su iOS o Android memorizzano i dati che non devono essere condivisi in una directory sicura. Tutti i dati dell'applicazione vengono archiviati in un'unica directory e, in genere, solo l'applicazione ha accesso diretto a quest'ultima e nessun'altra può accedervi. Tuttavia, entrambi i sistemi operativi dispongono di modi per consentire la condivisione dei dati di un'applicazione [15].

Ad esempio, le applicazioni possono richiedere autorizzazioni per accedere alla rubrica dell'utente, alla cartella download o alla posizione. Queste caratteristiche dovrebbero essere gestite con cura, poiché un'applicazione dannosa sul dispositivo, con le stesse autorizzazioni, potrebbe leggere i dati da altre applicazioni. Per ridurre il rischio, una buona pratica è quella di chiedere solo le autorizzazioni necessarie.

Quindi, per una conservazione sicura di dati, è essenziale che i dati siano protetti e crittografati in modo efficace. Per le applicazioni mobile ciò significa crittografare tutte le informazioni (cioè convertire i dati in testo illeggibile per l'essere umano) e applicare le autorizzazioni appropriate. Sia iOS che Android offrono servizi di archiviazione sicuri chiamati Keychain (per iOS) e Keystore (in Android) che consentono di crittografare i dati.

2.1.3 Tool di Analisi di Vulnerabilità

Sono disponibili molti tool per rilevare le vulnerabilità nelle applicazioni Android. I loro obiettivi principali sono:

- Test per la sicurezza complessiva delle app;
- Valutazione e analisi;
- Rilevamento fughe di dati.

Questi tool ci daranno come output le vulnerabilità che, se sfruttate, possono essere dannose per la sicurezza dell'utente e del dispositivo. Questo output è basato su delle analisi che rilevano il comportamento dannoso all'interno delle applicazioni e che sono mirate ad impedire l'installazione di queste app sul dispositivo.

Esistono principalmente due tipi di analisi eseguite dai tool, ovvero l'analisi statica e l'analisi dinamica. Nell'**analisi statica** si analizza il codice senza eseguirlo e ha il vantaggio di essere più veloce in quanto non ha bisogno di un ambiente runtime sandbox. Nell'**analisi dinamica** si analizza il comportamento del codice in esecuzione e la sua interazione con il sistema; necessita di più tempo rispetto all'analisi statica ma è più efficace [37].

Analisi Statica

L'analisi statica generalmente comporta l'acquisizione del codice sorgente - o in alcuni casi del codice oggetto - di un programma ed esaminarlo senza eseguirlo. I risultati vengono generati controllando il codice strutturale³, le sequenze di istruzioni e il modo in cui i valori delle variabili vengono elaborati nelle diverse chiamate di funzione. Un tipico processo di analisi statica inizia rappresentando il codice dell'applicazione analizzata come modelli astratti (ad es. Call Graph (CG), Control-Flow Graph (CFG) o Unified Modeling Language (UML) class/sequence diagram) in base allo scopo dell'analisi [6].

³È un primo metodo di codifica in cui si codificano i dati in base a domande o argomenti di ricerca.

In letteratura si discute di molte tecniche di analisi statica, ovvero:

- Control-Flow Analysis: è una tecnica che si concentra sul controllo di flusso⁴ in una struttura chiamante. Serve per determinare l'ordine delle operazioni in un programma. Questo potrebbe essere, ad esempio, determinare percorsi di esecuzione, ma anche vincoli di precedenza tra diverse operazioni [42, 31];
- Data-Flow Analysis: è una tecnica per raccogliere informazioni sul possibile insieme di valori calcolati in varie posizioni in un programma [26, 44];
- Points-to analysis: è una tecnica di analisi del codice statico che stabilisce quali puntatori, o riferimenti heap, possono puntare a quali variabili o posizioni di archiviazione [26];
- Call-Graph (CG) algorithm: è un control-flow graph [12, 22] che rappresenta le relazioni di chiamata tra subroutine in un programma. Ogni nodo rappresenta una funzione e ogni arco (f, g) indica che la funzione f chiama la funzione g . Un ciclo nel grafico indica chiamate di procedure ricorsive.
 - Class Hierarchy analysis (CHA) [13].
 - Rapid Type analysis (RTA) [9].
 - Variable Type Analysis (VTA) [39].
 - Andersen [7].
 - Steensguard [38].

I vantaggi dell'analisi statica sono i seguenti: (i) viene analizzato tutto il codice sorgente e il file manifest, (ii) è più veloce rispetto all'analisi dinamica e (iii) richiede meno risorse hardware rispetto all'analisi dinamica.

Molte vulnerabilità possono essere rilevate solo utilizzando l'analisi statica, come: perdite di dati sensibili, accessi non autorizzati a risorse protette o private, per rilevare l'abuso di autorizzazioni, per il rilevamento di clone e infine anche problemi legati alla crittografia e al consumo elevato di energia [26]. Questo perchè esamina tutti i

⁴Per controllo di flusso si intende il controllo sul flusso di esecuzione di un programma da parte del processore operato grazie alle strutture di controllo tipiche del linguaggio di programmazione in cui è scritto il codice sorgente del programma stesso.

possibili percorsi di esecuzione e i valori delle variabili, non solo quelli richiamati durante l'esecuzione. L'analisi statica può rivelare errori che potrebbero non essere individuate neppure dopo molti anni. Pertanto quest'ultimo aspetto dell'analisi statica è particolarmente prezioso per la garanzia della sicurezza.

Analisi Dinamica

La seconda tecnica è l'analisi dinamica, che viene utilizzata per analizzare l'applicazione eseguendola in un ambiente runtime sandbox (ambiente di test virtuale separato dal sistema operativo principale in cui vengono eseguite le applicazioni). L'analisi dinamica è generalmente considerata più complessa dell'analisi statica, in quanto richiede l'installazione delle applicazioni e la simulazione dell'input dell'utente (es. tocchi e click). I test di analisi dinamica più comuni sono: Fuzz Testing, Concolic Testing e Search-based Testing [19].

Fuzz Testing è una tecnica di test del software automatizzata, che prevede la fornitura di dati non validi, inaspettati o casuali come input per un programma. Il programma viene quindi monitorato tramite le eccezioni, per la ricerca di arresti anomali o per potenziali perdite di memoria [27, 17].

Concolic Testing è una tecnica di verifica del software ibrida, che unisce esecuzioni sia simboliche che concrete. L'esecuzione simbolica tratta le variabili del programma come variabili simboliche, mentre l'esecuzione concreta esegue test su particolari percorsi di input [28, 14].

Search-based Testing è un'ottimizzazione meta-euristica per automatizzare un'attività di test, come *simulated annealing* e *genetic algorithm* (sono entrambi metodi per risolvere problemi di ottimizzazione) [30].

2.2 Stato dell'arte

All'interno di questa sezione si discute di alcune delle tecniche più caratteristiche e già presenti sull'individuazione di vulnerabilità. Gli approcci analizzati provengono da:

- Approccio basato sull'utilizzo di Julian Test Suite 1.1: Detecting security vulnerabilities with static analysis – A case study. Realizzato da Midya Alqaradaghi Gregory Morse e Tamás Kozsik [5];
- Approccio di benchmarking per valutare e confrontare l'efficacia dei tool per il rilevamento delle vulnerabilità: Benchmarking Vulnerability Detection Tools for Web Services. Realizzato da Nuno Antunes e Marco Vieira [8].

2.2.1 Problema in esame

Secondo lo studio [16] 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 può essere quando l'applicazione ha l'attributo **"android:allowBackup"** impostato su **true** nel file `AndroidManifest.xml`, come mostrato all'interno dello snippet di codice sottostante.

```
1 <application android:theme="@android:style/Theme.Holo.Light.DarkActionBar
    " android:label="@string/app_name" android:icon="@mipmap/ic_launcher"
    android:debuggable="true" android:allowBackup="true">
```

Ciò consente a un utente malintenzionato di eseguire un backup dei dati dell'applicazione tramite ADB anche se il dispositivo non è rootato. È possibile utilizzare il comando `adb backup` per creare un backup del pacchetto. A seconda della versione del sistema operativo Android in uso potrebbe essere richiesta una password che sarà necessaria in seguito per decomprimere il file di backup.

Si è analizzato questo articolo [5] che esegue un confronto di prestazioni di quattro tool di analisi statica open-source (PMD, SpotBugs, Find Security Bugs e SonarQube) sul codice sorgente Java. L’analisi è stata condotta sulla collezione di programmi C/C++ e Java chiamato *Juliet Test Suite*, ampiamente utilizzata, in relazione a 6 vulnerabilità selezionate dall’elenco ufficiale *Top 25 list of Common Weakness Enumeration*⁵. In questo studio sono state calcolate le metriche di analisi per aiutare gli sviluppatori Java a decidere quale tool utilizzare per la verifica delle vulnerabilità di sicurezza dei loro programmi. È emerso che determinate vulnerabilità vengono rilevate maggiormente solo con determinati tool. A differenza di quest’analisi ove viene stabilito il tool più prestante andando a calcolare il margine di errore e una % di vulnerabilità che riesce ad individuare su un elenco di applicazioni intenzionalmente vulnerabili, lo studio che utilizza *Julian Suite Test 1.1* si concentra su delle vulnerabilità in particolare e ne fa il couting per ogni tool in esame stabilendone l’efficienza seguendo delle metriche stabilite.

Mentre, nell’articolo [8] si propone un approccio di benchmarking per valutare e confrontare l’efficacia dei tool di rilevamento delle vulnerabilità negli ambienti dei servizi web. Questo approccio è stato utilizzato per definire un benchmark concreto e mirato per i tool di rilevamento delle vulnerabilità di SQL Injection. Diversi strumenti sono stati confrontati, inclusi strumenti commerciali e open-source. Viene dimostrato che il benchmark proposto può essere facilmente utilizzato per valutare e confrontare tool di penetration testing e analizzatori di codici statici. In effetti, le metriche di benchmark hanno fornito un modo semplice per classificare gli strumenti nell’ambito del benchmarking. Le proprietà del benchmark sono state convalidate e discusse in dettaglio e ne suggeriscono l’utilizzo di esse.

⁵https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

3.1 Metodo di ricerca

La metodologia di ricerca adottata per questa tesi si concentra principalmente sulla lettura e lo studio di articoli che trattano le vulnerabilità sul software Android, estrapolando informazioni sui tool di identificazione di vulnerabilità Android disponibili in modo tale da raggruppare in classe e creare una sezione dedicata dove si fornisce una raccomandazione del tool più efficiente da utilizzare.

Le fonti sono state individuate preferendo specialmente *fonti accademiche* in quanto risultano avere più peso ed importanza e, sicuramente, più convincenti di altre.

Un primissimo passo è stato quello di analizzare articoli e siti web, prevalentemente in lingua inglese, per la ricerca di tool da configurare ed intuitibili nell'utilizzo. Come strumento principale è stato usato il motore di ricerca *Google Scholar*¹ e il database *sciencedirect*² che dà la possibilità di accedere ad un notevole elenco di articoli, tra riviste, tesi e report, attraverso l'inserimento di una query.

Di seguito ci sono le query utilizzate per la ricerca di articoli ove sono state inserite un insieme di parole chiavi centrate tutte sull'individuazione di vulnerabilità, sulla sicurezza di un applicazione android, sui framework o tool utilizzati per l'intercettazione/l'individuazione di esse o metodi preventivi e quindi soluzioni per evitare le

¹<https://scholar.google.com/>

²<https://www.sciencedirect.com/>

vulnerabilità all'interno di una applicazione. La ricerca è iniziata cercando sui vari database parole chiavi come "android vulnerability", "vulnerability analysis" e simili, per poi man mano analizzare gli articoli e andare a raffinare la query con l'aggiunta di ulteriori keyword in base alle parole chiavi identificate dagli autori.

("android vulnerability" OR "security android vulnerability" OR "mobile android security" OR "android mobile application vulnerability" OR "tool vulnerability") AND ("analysis" OR "solution" OR "prevention method" OR "analysis application" OR "framework" OR "detection")

Gli articoli considerati per lo studio presentano data di pubblicazione non inferiore al 1 Gennaio 2015, per evitare di analizzare dati obsoleti. Inizialmente il focus è stato per la ricerca e, in generale, sulle vulnerabilità sui sistemi Android. Dopo aver analizzato i tool presenti all'interno, il focus passa su di essi e la ricerca, successivamente, avviene sui singoli tool. Svolgendo questo metodo iterativamente.

Sono stati stabiliti dei criteri per indicare la qualità generale di un articolo, si è guardato generalmente la qualità della scrittura verificando il livello grammaticale, se utilizzasse un inglese di medio/alto livello e se fornisse delle citazioni a riferimenti esterni (altri articoli o link a siti web) dei tool che ha menzionato.

Inoltre nella maggior parte degli articoli studiati, l'argomento trattato si focalizzava principalmente sulla sicurezza informatica, come attacchi DoS (denial-of-service) e la protezione e mitigazione di essi [29, 40], o anche su studi scientifici dei malware [1]; queste informazioni sono state ignorate in quanto irrilevanti ai fini della tesi.

Una regola di base seguita nella ricerca effettuata è il confronto delle fonti. Uno studio o un dato non possono essere ritenuti affidabili se viene consultata una sola fonte, seppure autorevole. Non è stato raro, comunque, che due fonti abbiano riportato dati discordanti o addirittura contraddittori. In questi casi, l'unica soluzione è stato continuare il confronto con ulteriori fonti, valutandone sempre l'affidabilità e l'attualità di esse.

3.2 Approccio ai tool

I tool sono stati selezionati mediante dei criteri prestabiliti:

- Il tool deve essere stato menzionato esplicitamente in un articolo analizzato;
- 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 configurabile sul sistema nel quale sono stati svolti i test;
- Il tool deve svolgere l'analisi statica o dinamica;
- Il tool deve necessariamente avere una documentazione ufficiale.

Di seguito sono stati inseriti i tool trovati all'interno degli articoli e siti web analizzati:

- *AndroBugs*³ [32];
- *FlowDroid*⁴ [36];
- *MobSF*⁵ [10];
- *Oversecured*⁶ [41];
- *Pithus*⁷ [3];
- *QARK*⁸ [25];
- *SandDroid*⁹ [34];
- *Virustotal*¹⁰ [35];
- *Yaazhini*¹¹ [2].

I nomi dei tool sono stati ottenuti principalmente mediante la consultazione degli articoli. Per ogni singolo tool è stata svolta un'analisi al fine di accertarsi che questo rispettasse tutti i criteri e in caso negativo veniva scartato.

Una volta messi in chiaro e identificati i diversi tool l'obiettivo è stato quello di verificare il loro corretto funzionamento. Le documentazioni di alcuni tool sono state facilmente individuabili in quanto gli autori assieme al codice sorgente, forniscono il file spesso chiamato *readme* con istruzioni e linee guida per configurare e buildare l'app se necessario. In caso di un *web tool application*, la documentazione è, invece, presente sul sito web con sezioni in chiaro dedicate appositamente. Nella documentazione si scartano tutte le informazioni inerenti alla configurazione del tool per un

³<https://github.com/AndroBugs/AndroBugsFramework#readme>

⁴<https://github.com/secure-software-engineering/FlowDroid#readme>

⁵<https://github.com/MobSF/Mobile-Security-Framework-MobSF#readme>

⁶<https://oversecured.com/docs/api/>

⁷<https://beta.pithus.org/about/>

⁸<https://github.com/linkedin/qark#readme>

⁹<http://sanddroid.xjtu.edu.cn/static/resources/SandDroidUserManual.pdf>

¹⁰<https://support.virustotal.com/hc/en-us/categories/360000162878-Documentation>

¹¹<https://www.vegabird.com/docs/yaazhini/>

sistema operativo diverso dalla macchina sui cui si testa lo strumento. Il sistema utilizzato per il lancio dei tool utilizza un sistema operativo Ubuntu versione 22.10, 8gb ram ed un processore Intel® Core™ i5 di settima generazione x86-64.

Per poter proseguire con la parte successiva ciascun tool è stato lanciato con diverse configurazioni fino a quando non è avvenuta un'esecuzione senza eccezioni. Questa fase prevede un'allocazione di massimo 3 giorni per ciascun tool, e se dopo questi 3 giorni non si è ancora scoperto il modo di eseguire il tool senza errori, quest'ultimo verrà estromesso dallo studio.

Per la prova di esecuzione è stata utilizzata un'applicazione giocattolo, ossia un'app di cui sono note le vulnerabilità e che andrà in input ai tool, viene quindi utilizzata come una "prova" per testare l'affidabilità di quest'ultimo e per capire se può essere ammesso allo studio. Dopo aver effettuato un'approfondita ricerca ed aver ottenuto un elenco di applicazioni vulnerabili sul blog *hacknopedia*¹² la scelta è ricaduta sull'applicazione AndroGoat in quanto, tra i risultati ottenuti dalla ricerca, è quello che presenta una ricca lista di vulnerabilità elencate. Le vulnerabilità sono presenti nella Tabella 3.2.

¹²<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 dell'app. AndroGoat

L'applicazione è stata realizzata da Satish Patnayak¹³ ed è la prima app vulnerabile sviluppata utilizzando Kotlin¹⁴. Quest'ultimo consiste in un linguaggio di programmazione sviluppato dall'azienda di software JetBrains, si basa sulla JVM (Java Virtual Machine) ed è ispirato ad altri linguaggi di programmazione, tra i quali Scala e lo stesso Java [20].

Tutto questo servirà a comprendere se il tool presenterà delle eccezioni durante l'esecuzione oppure serviranno più di 3 giorni per eseguirlo.

3.3 Applicazioni e tool selezionati per l'analisi

3.3.1 Applicazioni intenzionalmente vulnerabili

Il primo dataset utilizzato per lo scopo della tesi consiste in un elenco di applicazioni con all'interno vulnerabilità inserite intenzionalmente.

Questo elenco è stato ottenuto mediante *hacknopedia*¹⁵, un blog sull'informatica che si concentra principalmente sulla sicurezza delle informazioni, notizie sulla sicurezza, tutorial e ricerche. La lista è stata messa a disposizione ai fini didattici e formativi.

L'elenco è composto da quattordici applicazioni, per ognuna di queste è presente il nome, la descrizione di tutte le vulnerabilità presenti e un link che reindirizza sulla repository di GitHub.

¹³<https://github.com/satishpatnayak>

¹⁴<https://github.com/satishpatnayak/AndroGoat>

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

Di seguito nella Tabella 3.3 sono riportate le applicazioni presenti all'interno del dataset con le relative vulnerabilità:

App.	Vulnerabilità
DVBA	<ul style="list-style-type: none"> Anti-debugging checks Hardcoded sensitive information Logcat leakage Insecure storage Exported activities Deep links Vulnerable SSL/TLS Application backup is allowed
InsecureBankv2	<ul style="list-style-type: none"> Intent Sniffing and Injection Weak Authorization mechanism Vulnerable Activity Components Root Detection Emulator Detection Insecure Content Provider access Insecure Webview implementation Weak Cryptography implementation Insecure Logging mechanism Android Pasteboard vulnerability Application Debuggable Android keyboard cache issues Android Backup vulnerability Runtime Manipulation Insecure SDCard storage Insecure HTTP connections Hardcoded secrets Username Enumeration issue Developer Backdoors Weak change password implementation

App.	Vulnerabilità
DIVA	Insecure Logging Hardcoding Issues Insecure Data Storage Input Validation Issues Access Control Issues
BeetleBug	Hardcoded Secrets Insecure Data Storage Sensitive Information Disclosure Vulnerable Android IPC Components (Broadcast, Content Prov.) Vulnerable Webviews Fingerprint Authentication By-pass Insecure Deep links Firebase Database Misconfiguration SQLite Injection Input Validation (XSS)
Sieve	SQL Injection Data leakage Directory Traversal Insecure Content Provider access
PIIVA	Weak encryption Man in the middle attack Object deserialization Hardcoded keys SQL Injection Predictable random number generator Exported broadcast receiver Unencrypted SQLite database Path Traversal

App.	Vulnerabilità
InsecureShop	Hardcoded Credential Insufficient URL Validation Weak Host Validation Check Arbitrary Code Execution Access to Protected Components Unprotected Data URIs Theft of Arbitrary Insecure Broadcast Receiver AWS Cognito Misconfiguration Insecure use of FilePaths in FileProvider Use of Implicit intent to send a broadcast with sensitive data Intercepting Implicit intent to load arbitrary URL Insecure Implementation of setResult in exported Activity Insecure Content Provider Lack of SSL Certificate Validation Insecure Webview Properties Enabled Insecure Data Storage Insecure Logging

Tabella 3.3: Applicazioni intenzionalmente vulnerabili

3.3.2 Tool selezionati

Una volta definita la lista dei tool presente nella sezione 3.2 è stata fatta una selezione tra quelli trovati durante la ricerca. Questi tool sono stati selezionati seguendo i criteri scritti nella sezione 3.2 e rispettando le regole presenti nella fase di configurazione.

Di seguito è riportato l'elenco con all'interno i tool presi in esame:

- MobSF - Web Application;
- Oversecured - Web Application;
- Pithus - Web Application;
- Yaazhini.

Le applicazioni non verranno distribuite per tool, ma tutte le applicazioni intenzionalmente vulnerabili all'interno del dataset andranno analizzate su tutti i tool scelti.

3.4 Classi di valutazione

I tool sono stati prima eseguiti e poi valutati in base ai risultati ottenuti da essi. È stato controllato se effettivamente individuassero le vulnerabilità con quelle dichiarate all'interno del dataset di *hacknopedia* preso in considerazione.

Sono state individuate delle classi di valutazione per stabilire quali tool sono efficienti nell'individuazione delle vulnerabilità. I tool sono stati suddivisi in 4 classi:

- **CLASSE [A]:** il tool rileva un numero maggior o uguale dell'80% delle vulnerabilità;
- **CLASSE [B]:** il tool rileva un numero maggiore o uguale del 60% e minore del 80% delle vulnerabilità;
- **CLASSE [C]:** il tool rileva un numero maggiore o uguale del 45% e minore del 60% delle vulnerabilità;
- **CLASSE [D]:** il tool rileva un numero maggiore o uguale dello 0% delle vulnerabilità e minore del 45% delle vulnerabilità.

Nome	Valutazione	
	% vulnerabilità	Classe
tool ₁	90%	A
tool ₂	1%	D
tool ₃	1%	D
tool ₄	1%	D
tool ₅	90%	A
tool ₆	1%	D

Tabella 3.4: Esempio tabella di valutazione dei tool selezionati

La Tabella 3.5 sovrastante è un esempio di come verranno presentati e suddivisi i diversi tool presi in considerazione.

3.5 Confronto dei tool

Stabilite le classi di appartenenza di ogni singolo tool, si dovrà svolgere un confronto tra i tool appartenenti alle stesse classi, in modo da creare una classifica tra le suddette. Bisognerà confrontare il margine di errore sull'identificazione delle vulnerabilità.

Sono state stabilite delle metriche per calcolare, in maniera affidabile, il margine di errore. Si andrà ad analizzare manualmente gli output dei tool, verificando che questi riportino effettivamente delle vulnerabilità potenziali. La precisione è definita come il rapporto tra le vere vulnerabilità positive e il numero totale di vulnerabilità rilevate, ovvero le vere vulnerabilità più le vulnerabilità false positive ($VT + VF$). Formalmente, per ogni tool la precisione è stata calcolata come segue:

$$Precisione = \frac{VT}{VT + VP}$$

Analisi dei Risultati

4.1 Oversecured

Oversecured è un tool commerciale (web application) che offre una licenza Premium ad un costo non molto economico, circa 229\$ ad applicazione. Il tool può essere usato gratuitamente a fini didattici, pertanto sono stati contattati gli autori che hanno fornito una chiave di accesso per poter permettere di effettuare analisi complete alle applicazioni senza alcun costo.

Oversecured esegue automaticamente l'analisi di tutte le vulnerabilità mobile note, inclusa l'esecuzione di codice arbitrario, il furto di file arbitrari e il cross-site scripting.

Ispeziona tutte le possibili fonti di dati pericolosi elaborati dall'app e emette informazioni su eventuali punti pericolosi che potrebbero portare a vulnerabilità, con un basso tasso di falsi positivi. Tutto ciò è stato ottenuto grazie a molti anni di ricerca.

Si presenta con un interfaccia grafica nella quale è possibile inserire in input l'APK che si vuol analizzare e selezionare o meno una checkbox per un avviso via email a lavoro terminato. Nella Figura 4.1 sottostante è possibile vedere come si presenta il tool con la propria interfaccia.

The screenshot displays the 'New Scan' interface of the Oversecured tool. On the left, under the heading 'New Scan', there is a section 'Uploaded Files (1)' with a dashed blue box for file upload. Inside the box, it says 'Drag & drop Android or iOS files' and '400 MB max per file'. Below this, it lists file types: 'Available for Android: .apk .abb' and 'Available for iOS: .zip (zipped Xcode project)'. An 'or' separator is between the two lists. At the bottom of the box is an orange button labeled 'Upload a file'. On the right side of the interface, there is a sidebar with several options: a checked checkbox for 'Use available scans', '7 scans available', a 'Coupon code' input field with an 'Activate' button, a link to 'Buy scan packs', an 'Order Summary' section showing 'Total Price' as '\$0', and a checkbox for 'Notify me by email when the scan is complete'. At the bottom of the sidebar is a large black button labeled 'Scan'.

Figura 4.1: Interfaccia del tool Oversecured

4.1.1 Classe di valutazione

Dopo una attenta analisi è possibile affermare che il tool Oversecured rientra nella **Classe B** in quanto riesce ad individuare il 63,75% delle vulnerabilità previste dal dataset delle applicazioni intenzionalmente vulnerabili.

La tabella di valutazione sarà, momentaneamente, la sottostante con solo il tool Oversecured all'interno di essa.

Nome	Valutazione		
	% vulnerabilità	-	Classe
Oversecured	63,75%	-	B

Tabella 4.1: Tabella di valutazione temporanea. Analisi eseguita sui seguenti tool: Oversecured

4.1.2 Margine di errore

È stato calcolato il margine di errore nell'identificazione delle vulnerabilità per ogni applicazione. I dati sono rappresentati nella Tabella 4.1 dove viene suddivisa per ogni applicazione.

DamnVB App	InsecureBankV2	DamnIV App	BeetleBug
0.545	0.555	0.25	0.454
Sieve App	PInsecureVA	InsecureShop	Media Prestazionale
0.16	0.272	0.473	0.387

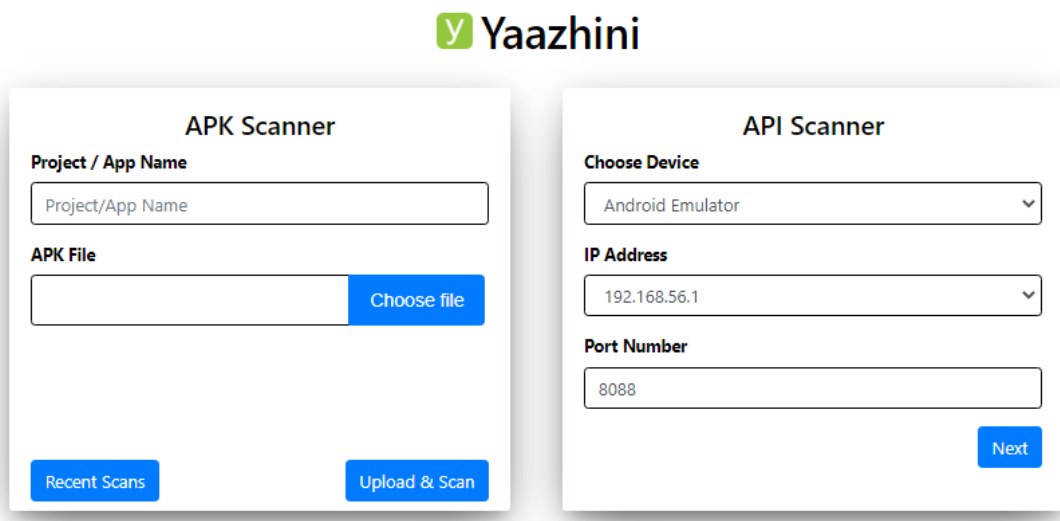
Tabella 4.2: Calcolo margine di errore, di tutte le applicazioni, del tool Oversecured

4.2 Yaazhini

Yaazhini è fruibile gratuitamente e aiuta gli utenti ad analizzare i file APK dell'applicazione Android trattata a scoprire e segnalare molte delle sue vulnerabilità utilizzando un approccio estensivo che comprende sia metodi automatizzati che manuali.

Si presenta con un interfaccia grafica intuibile nella quale è possibile sia analizzare un API che un APK.

In input ci andrà l'APK che si vuol analizzare ed inoltre c'è una funzione che ci permette di creare un progetto riguardante quella determinata analisi. Nella Figura 4.2 sottostante è possibile vedere come si presenta il tool con la propria interfaccia.



The image displays the Yaazhini web interface, featuring a logo with a green 'Y' and the text 'Yaazhini'. Below the logo are two side-by-side forms. The left form, titled 'APK Scanner', includes a 'Project / App Name' text input field, an 'APK File' section with a file selection input and a blue 'Choose file' button, a 'Recent Scans' button, and a blue 'Upload & Scan' button. The right form, titled 'API Scanner', includes a 'Choose Device' dropdown menu (set to 'Android Emulator'), an 'IP Address' dropdown menu (set to '192.168.56.1'), a 'Port Number' text input field (set to '8088'), and a blue 'Next' button.

Figura 4.2: Interfaccia del tool Yaazhini

4.2.1 Classe di valutazione

Dopo una attenta analisi è possibile affermare che il tool Yaazhini rientra nella **Classe D** in quanto riesce ad individuare il 32,81% delle vulnerabilità previste dal dataset delle applicazioni intenzionalmente vulnerabili.

La tabella di valutazione sarà, momentaneamente, la sottostante con il tool Oversecured e Yaazhini all'interno di essa.

Nome	Valutazione		
	% vulnerabilità	-	Classe
Oversecured	63,75%	-	B
Yaazhini	32,81%	-	D

Tabella 4.3: Tabella di valutazione temporanea. Analisi eseguita sui seguenti tool: Oversecured, Yaazhini

4.2.2 Margine di errore

È stato calcolato il margine di errore nell'identificazione delle vulnerabilità per ogni applicazione. I dati sono rappresentati nella Tabella 4.4 dove viene suddivisa per ogni applicazione.

DamnVB App	InsecureBankV2	DamnIV App	BeetleBug
0.888	0.909	1	N/A
Sieve App	PInsecureVA	InsecureShop	Media Prestazionale
0.5	0.562	0.809	0.778

Tabella 4.4: Calcolo margine di errore, di tutte le applicazioni, del tool Yaazhini

N.B: La sigla N/A corrisponde ad *analisi non applicabile* in quanto il tool non riesce ad analizzare l'applicazione in questione.

4.3 MobSF

MobSF è un tool fruibile gratuitamente (web application) ideato da Ajin Abraham¹. L'acronimo sta per Mobile Security Framework ed è una web application automatizzata (Android/iOS/Windows) in grado di effettuare penetration testing², analisi del malware e di valutare la sicurezza di un'applicazione eseguendo analisi statiche e dinamiche.

Si presenta con un interfaccia grafica nella quale è possibile inserire in upload l'APK che si vuol analizzare. Nella Figura 4.3 sottostante è possibile vedere come si presenta il tool agli utenti. Dopo aver dato in upload l'APK, questo sarà presente in un elenco di applicazioni facilmente individuabile mediante un bottone collocato in alto a sinistra.

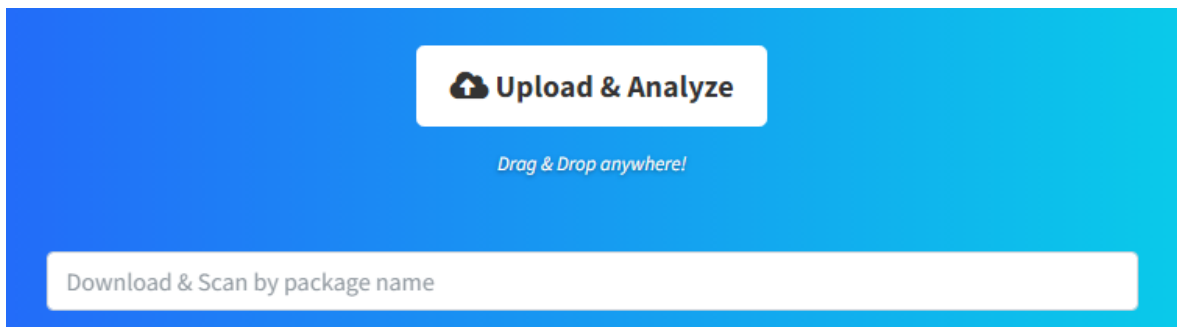


Figura 4.3: Interfaccia del tool MobSF

¹<https://twitter.com/ajinabraham>

²Valutazione della sicurezza di un sistema informatico o di una rete.

4.3.1 Classe di valutazione

Dopo una attenta analisi è possibile affermare che il tool MobSF rientra nella **Classe C** in quanto riesce ad individuare il 48,61% delle vulnerabilità previste dal dataset delle applicazioni intenzionalmente vulnerabili.

La tabella di valutazione sarà, momentaneamente, la sottostante con il tool Oversecured, Yaazhini e MobSF all'interno di essa.

Nome	Valutazione		
	% vulnerabilità	-	Classe
Oversecured	63,75%	-	B
Yaazhini	32,81%	-	D
MobSF	48,61%	-	C

Tabella 4.5: Tabella di valutazione temporanea. Analisi eseguita sui seguenti tool: Oversecured, Yaazhini, MobSF

4.3.2 Margine di errore

È stato calcolato il margine di errore nell'identificazione delle vulnerabilità per ogni applicazione. I dati sono rappresentati nella Tabella 4.6 dove viene suddivisa per ogni applicazione.

DamnVB App	InsecureBankV2	DamnIV App	BeetleBug
0.533	0.571	0.357	0.833
Sieve App	PInsecureVA	InsecureShop	Media Prestazionale
0.363	0.5	0.68	0.548

Tabella 4.6: Calcolo margine di errore, di tutte le applicazioni, del tool MobSF

4.4 Pithus

Pithus è una web application fruibile gratuitamente e open-source per analizzare le applicazioni Android.

Le analisi si basano su molteplici tool noti come: APKiD³, ssdeep⁴, Dexofuzzy⁵, QuarkEngine⁶, AndroGuard⁷, MobSF⁸, Exodus-core⁹. Non appena viene dato in upload l'APK viene analizzato dai tool appena citati e ne raggruppa i risultati.

Si presenta agli utenti con un'interfaccia semplice nella quale è possibile inserire in upload l'APK che si vuol analizzare. Come mostrato nella Figura 4.4, c'è un criterio limite imposto, ed è possibile caricare un APK con una dimensione massima di 65.3 MB. Successivamente il tool indirizzerà l'utente direttamente al foglio dell'analisi.

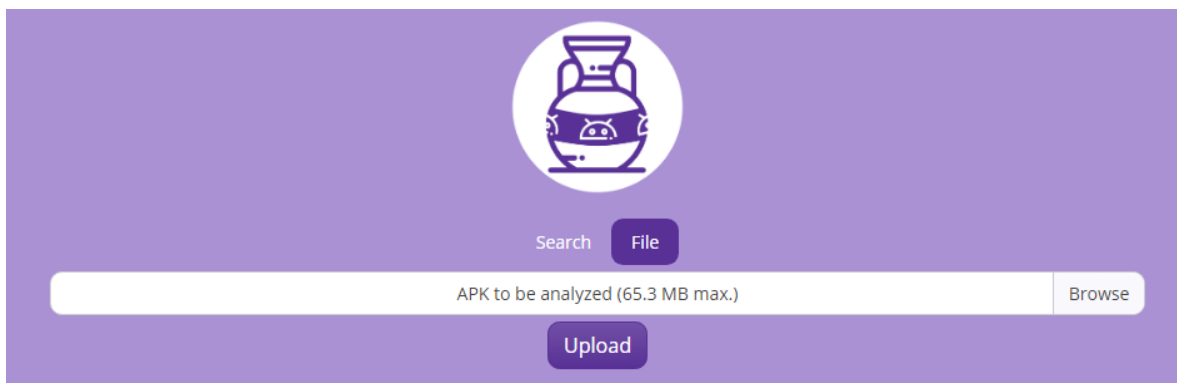


Figura 4.4: Interfaccia del tool Pithus

³<https://github.com/rednaga/APKiD>

⁴<https://github.com/DinoTools/python-ssdeep>

⁵<https://github.com/ESTsecurity/Dexofuzzy>

⁶<https://github.com/quark-engine/quark-engine>

⁷<https://github.com/androguard/androguard>

⁸<https://github.com/MobSF/Mobile-Security-Framework-MobSF>

⁹<https://github.com/Exodus-Privacy/exodus-core>

4.4.1 Classe di valutazione

Dopo una attenta analisi è possibile affermare che il tool Pithus rientra nella **Classe C** in quanto riesce ad individuare il 58,89% delle vulnerabilità previste dal dataset delle applicazioni intenzionalmente vulnerabili.

La tabella di valutazione **finale** sarà la sottostante con il tool Oversecured, Yaazhini, MobSF e Pithus all'interno di essa.

Nome	Valutazione		
	% vulnerabilità	-	Classe
Oversecured	63,75%	-	B
Yaazhini	32,81%	-	D
MobSF	48,61%	-	C
Pithus	58,89%	-	C

Tabella 4.7: Tabella di valutazione finale. Analisi eseguita su tutti i tool selezionati

4.4.2 Margine di errore

È stato calcolato il margine di errore nell'identificazione delle vulnerabilità per ogni applicazione. I dati sono rappresentati nella Tabella 4.8 dove viene suddivisa per ogni applicazione.

DamnVB App	InsecureBankV2	DamnIV App	BeetleBug
0.727	0.833	0.5	0.769
Sieve App	PInsecureVA	InsecureShop	Media Prestazionale
N/A	0.473	0.739	0.673

Tabella 4.8: Calcolo margine di errore, di tutte le applicazioni, del tool Pithus

N.B: La sigla N/A corrisponde ad *analisi non applicabile* in quanto il tool non riesce ad analizzare l'applicazione in questione.

4.5 Risultati ottenuti

I risultati ottenuti dall'analisi svolta per l'individuazione delle vulnerabilità mediante tool stabilendone l'efficienza di questi sono riportati nella seguente tabella:

Variabili	Oversecured	Yaazhini	MobSF	Pithus
% vulnerabilità	63,75%	32,81%	48,61%	58,89%
Margine di errore				
Media aritmetica	0.387	0.778	0.548	0.673
Mediana	0.454	0.849	0.533	0.733
Minimo	0.160	0.500	0.357	0.473
Massimo	0.555	1.000	0.833	0.833
Primo quartile	0.250	0.547	0.363	0.493
Terzo quartile	0.545	0.932	0.680	0.785
Scarto interquartile	0.295	0.385	0.317	0.292
Dati anomali	nessuno	nessuno	nessuno	nessuno

Tabella 4.9: Statistiche descrittive

Una delle metriche prese in considerazione è la precisione ottenuta dai vari tool presi in esame, quest'ultima da come si evince dalla Figura 4.5 ha registrato risultati molto alti per tutti i tool utilizzati, tranne per un tool ovvero Oversecured che ha registrato una maggiore precisione con un risultato di 0.387, mentre il risultato peggiore l'ha ottenuto il tool Yaazhini con un risultato di 0,778.

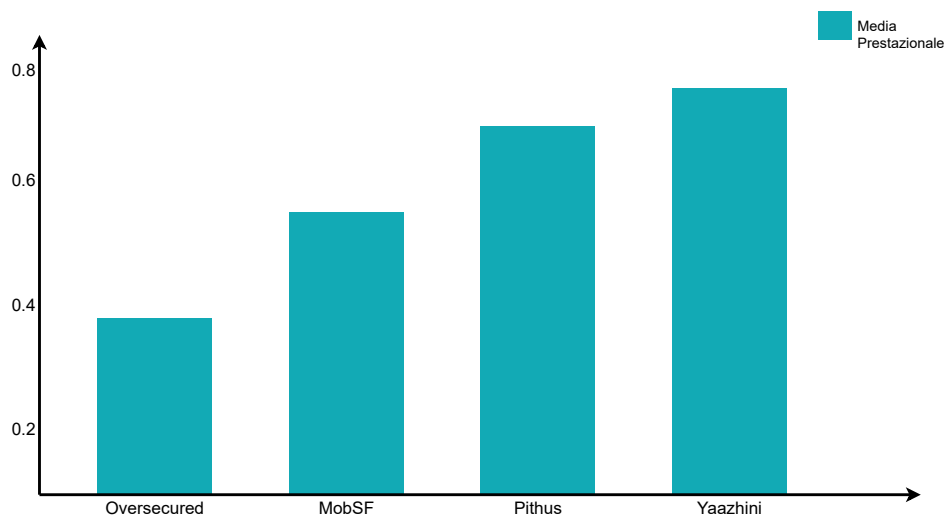


Figura 4.5: Precisione tool

Analizzando i risultati ottenuti dall'analisi svolta si ottiene che:

- Il tool Oversecured presenta una precision più alta, ma soprattutto una media aritmetica del margine di errore molto inferiore rispetto agli altri tool, questo implica ad essere il tool più prestante rispetto agli altri presi in esame.

Si è stabilita una classificazione finale ottenuta andando a confrontare la media prestazionale dei tool appartenenti alla medesima classe. La tabella sottostante ne riporta i risultati.

Nome	Classificazione dei tool		Classe
	% vulnerabilità	Media prestazionale	
Oversecured	63,75%	0.387	B
MobSF	48,61%	0.548	C
Pithus	58,89%	0.673	C
Yaazhini	32,81%	0.778	D

Tabella 4.10: Classificazione finale dei tool

Conclusioni e sviluppi futuri

I risultati principali indicano che il problema dovrebbe essere ulteriormente approfondito. Questa tesi ci ha mostrato quanto siano importanti le vulnerabilità sul ciclo di vita del software e come queste possono influire sui consumatori, 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 e analisi dinamiche, 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, uno dei lavori futuri più importanti potrebbe essere l'inserimento di altre metriche per calcolare l'accuracy e la recall, o altre che riescano a raccogliere informazioni interessanti e che possano avere un impatto maggiore sull'analisi finale. Un altro importante sviluppo potrebbe essere quello di una ri-costruzione di un nuovo dataset di applicazioni intenzionalmente vulnerabili, aggiungendone un numero maggiore delle presenti per rendere più precise le statistiche finali. In aggiunta a quest'ultimo, si potrebbe incrementare il numero dei tool andando ad analizzarne di più, stabilendo altri criteri di selezione più tolleranti.

Bibliografia

- [1] Current android malwares, available at <https://forensics.spreitzenbarth.de/android-malware/>. (Citato a pagina 16)
- [2] Yaazhini - free android apk api vulnerability scanner, available at https://idiopathic24.rssing.com/chan-13017459/all_p147.html. (Citato a pagina 18)
- [3] Investigating android malware with pithus, the website is available at <https://cryptax.medium.com/investigating-android-malware-with-pithus-17d2143cc528%7d>, July 2021. (Citato a pagina 18)
- [4] Statista. 2022. Number of available applications in the google play store from december 2009 to september 2022, 2022. (Citato a pagina 1)
- [5] Midya Alqaradaghi, Gregory Morse, and Tamás Kozsik. Detecting security vulnerabilities with static analysis – a case study. Pollack Periodica, 17(2):1 – 7, 2022. (Citato alle pagine 13 e 14)
- [6] Amr Amin, Amgad Eldessouki, Menna Tullah Magdy, Nouran Abdeen, Hanan Hindy, and Islam Hegazy. Androshield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach. Information, 10(10), 2019. (Citato a pagina 10)
- [7] Lars Ole Andersen and Peter Lee. Program analysis and specialization for the c programming language. 2005. (Citato a pagina 11)

- [8] Nuno Antunes and Marco Vieira. Benchmarking vulnerability detection tools for web services. In 2010 IEEE International Conference on Web Services, pages 203–210, 2010. (Citato alle pagine 13 e 14)
- [9] David F. Bacon and Peter F. Sweeney. Fast static analysis of c++ virtual function calls. In Proceedings of the 11th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOP-SLA '96, page 324–341, New York, NY, USA, 1996. Association for Computing Machinery. (Citato a pagina 11)
- [10] Patrick C. K. Hung Farkhund Iqbal Liaqat Ali Benjamin Yankson, Javed Vali. Security assessment for zenbo robot using drozer and mobsf frameworks. (Citato a pagina 18)
- [11] Stefan Brahler. Analysis of the android architecture. Karlsruhe institute for technology, 7(8), 2010. (Citato alle pagine 6 e 7)
- [12] D. Callahan, A. Carle, M.W. Hall, and K. Kennedy. Constructing the procedure call multigraph. IEEE Transactions on Software Engineering, 16(4):483–487, 1990. (Citato a pagina 11)
- [13] Jeffrey Dean, David Grove, and Craig Chambers. Optimization of object-oriented programs using static class hierarchy analysis. In Proceedings of the 9th European Conference on Object-Oriented Programming, ECOOP '95, page 77–101, Berlin, Heidelberg, 1995. Springer-Verlag. (Citato a pagina 11)
- [14] Ehsan Edalat, Babak Sadeghiyan, and Fatemeh Ghassemi. Considroid: A concolic-based tool for detecting sql injection vulnerability in android apps, 2018. (Citato a pagina 12)
- [15] Johannes Feichtner. A comparative study of misapplied crypto in android and ios applications. pages 96–108. INSTICC, SciTePress, 2019. (Citato a pagina 9)
- [16] 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, 2021. (Citato a pagina 13)

- [17] Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. Grammar-based white-box fuzzing. PLDI '08, page 206–215, New York, NY, USA, 2008. Association for Computing Machinery. (Citato a pagina 12)
- [18] GOOGLE INC. (Hrsg.). Android software development kit (sdk). google inc. (Citato a pagina 6)
- [19] Arunkumar Balakrishnan Hrushikesha Mohanty, J. R. Mohanty. Springer Singapore, 1st edition. (Citato a pagina 12)
- [20] Dmitry Jemerov and Svetlana Isakova. Kotlin in action. Simon and Schuster, 2017. (Citato a pagina 21)
- [21] Jamil Khan and Sara Shahzad. Android architecture and related security risks. 05:14, 2015. (Citato a pagina 5)
- [22] Uday Khedker, Amitabha Sanyal, and Bageshri Sathe. Data flow analysis: theory and practice. CRC Press, 2017. (Citato a pagina 11)
- [23] Jinsung Kim, Younghoon Ban, Eunbyeol Ko, Haehyun Cho, and Jeong Hyun Yi. Mapas: a practical deep learning-based android malware detection system. 21(4):725–738, Aug 2022. (Citato a pagina 2)
- [24] ALI FEIZOLLAH NOR BADRUL ANUAR ROSLI SALLEH LORENZO CAVALLARO KIMBERLY TAM, Royal Holloway. The evolution of android malware and android analysis techniques. (Citato a pagina 5)
- [25] Keyur Kulkarni and Ahmad Y Javaid. Open source android vulnerability detection tools: A survey, 2018. (Citato alle pagine 1, 5, 8 e 18)
- [26] Li Li, Tegawendé F. Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Outeau, Jacques Klein, and Le Traon. Static analysis of android apps: A systematic literature review. Information and Software Technology, 88:67–95, 2017. (Citato a pagina 11)
- [27] Riyadh Mahmood, Naeem Esfahani, Thabet Kacem, Nariman Mirzaei, Sam Malek, and Angelos Stavrou. A whitebox approach for automated security

- testing of android applications on the cloud. pages 22–28, 2012. (Citato a pagina 12)
- [28] Rupak Majumdar and Koushik Sen. Hybrid concolic testing. pages 416–426, 2007. (Citato a pagina 12)
- [29] Alessio Merlo Mauro Migliardi. Dispositivi mobili: nuovi problemi di sicurezza. (Citato a pagina 16)
- [30] Phil McMinn. Search-based software testing: Past, present and future. pages 153–163, 2011. (Citato a pagina 12)
- [31] Zhaoyi Meng, Yan Xiong, Wenchao Huang, Fuyou Miao, Taeho Jung, and Jianmeng Huang. Divide and conquer: Recovering contextual information of behaviors in android apps around limited-quantity audit logs. In Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE '19, page 230–231. IEEE Press, 2019. (Citato a pagina 11)
- [32] 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 18)
- [33] Heng Yin Mu Zhang. Appsealer: Automatic generation of vulnerability-specific patches for preventing component hijacking attacks in android applications. (Citato a pagina 8)
- [34] Rafay Hassan Niazi, Jawwad Ahmed Shamsi, Tahir Waseem, and Muhammad Mubashir Khan. Signature-based detection of privilege-escalation attacks on android. In 2015 Conference on Information Assurance and Cyber Security (CIACS), pages 44–49, 2015. (Citato a pagina 18)
- [35] Linhai Song Gang Wang Peng Peng, Limin Yang. Opening the blackbox of virustotal: Analyzing online phishing scan engines. (Citato a pagina 18)
- [36] Lina Qiu, Yingying Wang, and Julia Rubin. Analyzing the analyzers: Flowdroid/iccta, amandroid, and droidsafe. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, page

- 176–186, New York, NY, USA, 2018. Association for Computing Machinery. (Citato a pagina 18)
- [37] Faysal Hossain Shezan, Syeda Farzia Afroze, and Anindya Iqbal. Vulnerability detection in recent android apps: An empirical study. pages 55–63, 2017. (Citato a pagina 10)
- [38] Bjarne Steensgaard. Points-to analysis in almost linear time. In Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '96, page 32–41, New York, NY, USA, 1996. Association for Computing Machinery. (Citato a pagina 11)
- [39] Vijay Sundaresan, Laurie Hendren, Chrislain Razafimahefa, Raja Vallée-Rai, Patrick Lam, Etienne Gagnon, and Charles Godin. Practical virtual method call resolution for java. SIGPLAN Not., 35(10):264–280, oct 2000. (Citato a pagina 11)
- [40] Alessia Valentini. Famose applicazioni android mettono a rischio milioni di utenti: che c'è da sapere. (Citato alle pagine 8 e 16)
- [41] Janos Vrancsik and Dora Pinter. Detect app vulnerabilities with the oversecured step, 2021. (Citato a pagina 18)
- [42] Xiaolei Wang, Sencun Zhu, Dehua Zhou, and Yuexiang Yang. Droid-antirm: Taming control flow anti-analysis to support automated dynamic analysis of android malware. In Proceedings of the 33rd Annual Computer Security Applications Conference, ACSAC '17, page 350–361, New York, NY, USA, 2017. Association for Computing Machinery. (Citato a pagina 11)
- [43] Yang Wang, Jun Zheng, Chen Sun, and Srinivas Mukkamala. Quantitative security risk assessment of android permissions and applications. Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. (Citato a pagina 1)
- [44] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. ACM Trans. Priv. Secur., 21(3), apr 2018. (Citato a pagina 11)

Ringraziamenti

A conclusione di questa ricerca, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non sarebbe mai esistito.

Innanzitutto ringrazio il mio relatore Fabio Palomba e i miei correlatori Giammaria Giordano ed Emanuele Iannone per avermi dato la possibilità di intraprendere questo percorso ed arricchire le mie conoscenze nell'ambito. In questi mesi di lavoro siete stati super fondamentali, avete saputo guidarmi con consigli e suggerimenti utili, ma soprattutto un ringraziamento va alla vostra infinita disponibilità e puntualità. A qualunque ora siete sempre stati pronti a colmare tutti i miei dubbi.

Ai miei genitori, fonte di costante supporto, colonne portanti della mia vita, grazie che mi avete permesso di compiere gli studi in maniera serena senza mai dubitare delle mie capacità, da loro ho ricevuto l'educazione, l'esempio e l'affetto tanto grande quanto ogni figlio potrebbe desiderare. Con immenso sacrificio mi hanno permesso di arrivare alla fine di questo percorso, sono loro ai quali devo tutte le mie più grandi vittorie. Spero di rendervi sempre orgogliosi di me.

A mio fratello Gabriele, a te che sei la mia spalla e la mia ombra, grazie per avermi sempre appoggiato, spronato e per aver colmato ogni mia insicurezza. Grazie perchè anche se siamo lontani entrambi sappiamo che cammineremo sempre fianco a fianco, mai dietro, mai avanti, ma accanto. E lo so che di qualunque cosa avrò bisogno, mi girerò e troverò te. Non hai mai smesso di credere in me. Grazie per essere la mia certezza.

Ad una persona speciale che qualche giorno fa ha compiuto gli anni e sono sicuro che in questo momento sarebbe fiero di me. All'angelo più bello del paradiso, Zio Donato, mi sono laureato, per me e per te. Mi hai sempre accompagnato ad ogni tassello di questo percorso senza mai mollarmi un attimo, ti ho sentito vicino ad ogni esame e ad ogni istante della mia vita. Devo ringraziarti perchè nonostante tu ci abbia lasciato, mi hai lasciato con le Persone più forti che io conosca, mi hai lasciato con i nonni, dovresti vederli, sono un fiume in piena inesauribili che mi tengono al sicuro e non mi fanno mai mancare niente. Questo mio traguardo è il tuo regalo di compleanno che hai sempre desiderato, perdonami per i pochi giorni di ritardo, Zio. *Ovunque tu sia.. dillo alla luna.* Mi raccomando, salutami i nonni Gabriele e Maddalena, spero siate orgogliosi di me, questo traguardo è anche per voi.

Ai miei nonni Eduardo e Felicia, i miei secondi genitori oramai, grazie per avermi trasmesso i valori più importanti della vita, la persona che sono oggi è soprattutto grazie a voi. Grazie perchè siete forti e non vi abbattate mai nonostante le mille difficoltà, grazie per avermi viziato e amato, grazie per avermi insegnato a cadere e a rialzarmi. Grazie perchè ci siete sempre.

Ai miei zii e ai miei cugini grazie per rendere speciale la mia famiglia. Sono felice che mi abbiate trasmesso la passione e l'entusiasmo nel fare ogni cosa, anche le più semplici, e perciò voglio ringraziarvi anche per questo. Grazie Valerio per le visite pomeridiane durante lo studio, grazie Irene un tuo sorriso e una tua risata migliorano le giornate, vi ringrazio tutti, ma proprio a tutti, siete fondamentali.

Ad Alfredo con il quale ho condiviso l'intero mio percorso universitario, siamo stati sempre complici e adesso possiamo dire che ce l'abbiamo fatta finalmente, insieme, come avevamo previsto qualche anno fa. L'avresti mai detto? Grazie per avermi sempre incoraggiato a non mollare mai e a guardare avanti a testa alta per arrivare fino a questo momento. Grazie per aver creduto in me quando nessuno lo faceva e grazie per la quotidianità passata assieme.

Ai miei amici dell'Università che sono stati sempre al mio fianco. In particolare Nicola Pio Gagliardi, Silvio Venturino, Catello Staiano, Alessandro Aquino, Umberto Della Monica, Antonio Romano e Carmine Leo. Grazie per avermi accompagnato in questi tre anni universitari così pieni di stimoli e avventure, quante ne abbiamo passate.. non si contano.

A tutti i miei amici, quelli delle mille risate, delle vacanze, dei momenti di spensieratezza, delle parole di conforto, delle esperienze condivise insieme, che sono stati parte fondamentale della mia crescita. Grazie al mio gruppo di sempre: Alessandro, Dario, Felice, Felicio e Salvatore, senza di voi non sarebbe stato lo stesso. Grazie ad Armando diventato importante, Pako Bear con Marti per i sorrisi della bimba durante il giorno, Luca Giordano, Ciro anche se a Torino, Saso e tutti gli altri.. siete davvero troppi, ma siete compresi tutti, vi ringrazio per esserci sempre. Ringrazio Giovanni, un fratello, che mi è vicino dall'asilo, e anche se ora non ci vediamo e sentiamo più così spesso sono fiero di te e di quello che dimostri ai tuoi genitori e a tutti, tutti i giorni. Sono sicuro che ci saremo sempre per entrambi.

A Sara, la quale, tralasciando tutto, mi ha accompagnato un anno e mezzo del mio percorso universitario. Le nostre strade si sono separate, ma un pezzo di questo traguardo va anche a te perchè ci sei sempre stata anche quando io non te lo permettevo. Grazie perchè mi hai sempre capito al contrario di me con te, grazie perchè ridimensionavi le mie paure, le mie ansie e il mio stress facendolo tuo, ti invidio perchè sei sempre stata forte ad attutire i colpi degli ostacoli che ti si piazzavano davanti. Ti auguro il meglio perchè la bontà unica che ti hanno trasmesso i tuoi splendidi genitori e tuo fratello va premiata. Fai sempre tutto ciò che ti rende felice, te lo meriti.

Per terminare, questo obiettivo va anche a chi non ha mai creduto in me, abbattendomi e rendendomi le cose più complicate. Un grazie a me stesso per non aver mollato davanti a montagne di ostacoli che mi sono ritrovato davanti. La dedico ai sacrifici che pensavo di non essere in grado di sostenere ed alla tenacia che mi ha fatto arrivare in fondo.

Questo piccolo traguardo è solo l'inizio di tanti altri.