Università degli studi di Cagliari
Corso di Laurea Triennale in Informatica

Studio ed analisi di librerie crittografiche impiegate da malware Android

Luca Puzzoni
Relatore: Davide Maiorca

25 Febbraio 2021

Alla mia famiglia
Alla mia ragazza
Ai miei migliori amici

Indice

Capitolo 1

Introduzione

Ad oggi, Android è il sistema operativo per dispositivi mobili più diffuso al mondo, con oltre 2.5 miliardi di dispositivi venduti. Questo sistema operativo, nel corso degli anni, ha mostrato una evidente maturazione e crescita tecnologica. Tale maturazione è apprezzabile osservando la figura 1.1, che illustra la crescita del numero di applicazioni disponibili su Android da Dicembre 2009 a Gennaio 2020 [1].

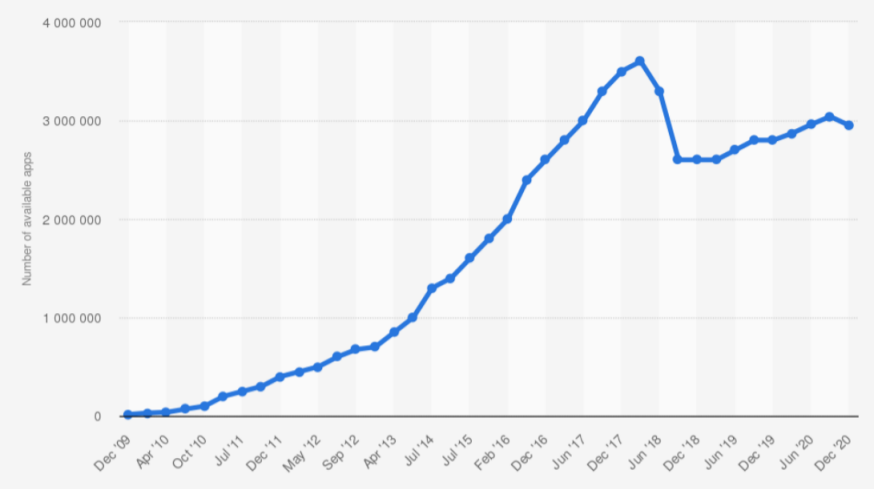

Figura 1.1: Diffusione delle applicazioni Android tra il 2009 e il 2020

La curva mostra come, tra il 2014 e il 2018, le applicazioni disponibili per questo sistema operativo siano quasi quadruplicate. Questo fenomeno nasce principalmente per il fatto che, per effetto dell'aumento del numero di dispositivi venduti e per il loro sempre maggiore impiego nella vita di tutti i giorni, si ha

sempre più bisogno di applicazioni che siano in grado di assolvere alle necessità quotidiane.

Allo stesso tempo, però, con la continua presentazione di nuove applicazioni per questo pubblico sempre più ampio, sempre più malintenzionati hanno pensato di includere nelle loro applicazioni, spesso scambiate per normali giochi e utility di vario tipo, codice malevolo che fosse in grado di danneggiare la vittima su più fronti. I vari report di sicurezza mostrano che gli interessi degli attaccanti possono essere l'ottenimento di informazioni sensibili, il danneggiamento del dispositivo e il "rapimento" delle informazioni della vittima mediante algoritmi crittografici applicati alla memoria del dispositivo, rendendole quindi inaccessibili. Quest'ultima categoria di attacchi prende il nome di ransomware perchè l'attaccante, per riportare i dati della vittima al loro stato originale, pretende un riscatto o `ransom`.

Per far fronte a questo genere di problemi, Google (proprietario di Android e del progetto AOSP) istituisce nel 2015 un team specializzato nella sicurezza su Android. Dal 2015 ad oggi, questo team è riuscito ad ottenere risultati eccellenti, implementando con il tempo tecnologie via via più raffinate per il riconoscimento del codice malevolo nell'atto di upload dell'applicazione sul noto market di Google, il `Google Play Store` e nelle versioni, anche meno recenti, del sistema operativo.

Inoltre, il team per la sicurezza ha avviato un programma noto come `Android security rewards program`, che consente a ricercatori da tutto il mondo di incrementare il livello di sicurezza di Android e di ricevere, a fronte di queste migliorie, un compenso. Dall'ultimo report pubblicato da Google a Marzo del 2019 possiamo apprezzare questi risultati che vengono sintetizzati come [2]:

1. Sorpasso dei 3 milioni di dollari in investimenti nel programma `Android security rewards program`.
2. +84% di dispositivi che ricevono update di sicurezza se confrontati allo stesso semestre dell'anno precedente.

1.1 Diffusione dei Malware su Android

Nonostante gli sforzi effettuati anche da Google per il miglioramento della sicurezza su Android, i malware per questa piattaforma hanno continuato a diffondersi rapidamente.

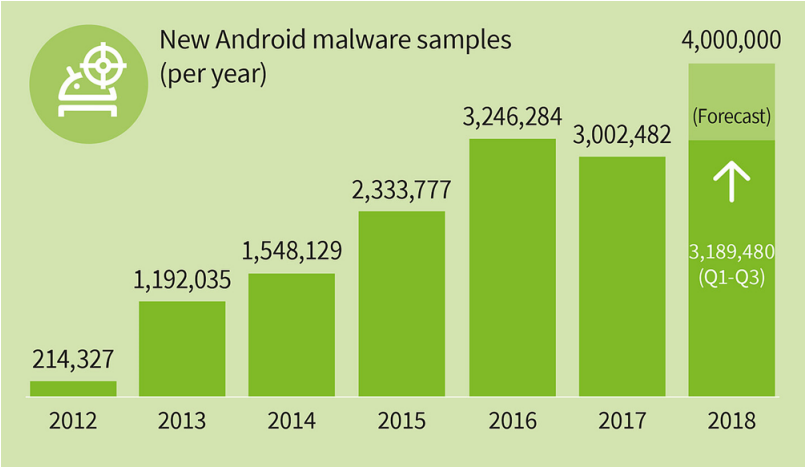

Figura 1.2: Distribuzione dei malware per Android tra il 2012 e il 2018

La figura 1.2 riassume uno studio effettuato nel 2018 dalla società tedesca G-Data che da 35 anni si occupa di cyber security in oltre 60 paesi. In questo studio, il colosso tedesco mostra l'entità della crescita degli attacchi informatici a dispositivi Android [3].

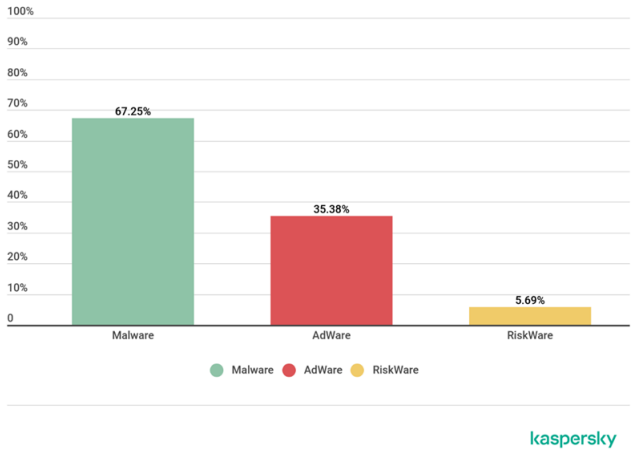

Figura 1.3: Distribuzione malware secondo Kaspersky

Il grafico in figura 1.3 invece, redatto dal noto produttore di sistemi antivirus Kaspersky, rivela la distribuzione dei vari tipi di malware su Android nel secondo quadrimestre del 2020. Scendendo nel dettaglio dell'indagine condotta dal team russo, notiamo che nel solo secondo semestre di quest'anno sono stati effettuati

1.245.894 attacchi, e di questi più di 42.000 riguardano solo attacchi di tipo trojan-horse con ransomware o attacchi ai vari servizi di mobile banking. La grande crescita dei ransomware illustra come gli attaccanti stiano ricorrendo sempre di più a tecniche di crittografia per effettuare i loro attacchi. [4].

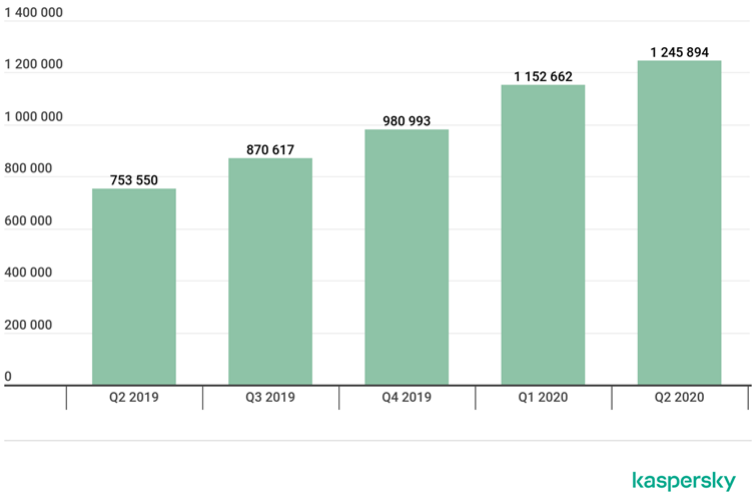

Figura 1.4: Diffusione malware secondo Kaspersky

Questi dati dimostrano che la diffusione massiva di questo sistema ha alimentato un settore del cyber crime che continuerà ad avanzare. Per questo motivo, bisogna adottare nuovi sistemi di difesa in grado di riconoscere il problema alla fonte prima che questo possa arrecare danno.

In virtù di questo fatto, risulta ancora più importante esaminare certe caratteristiche dei malware Android che, in generale, sono state poco studiate. In questa tesi, ci soffermeremo in particolare sull'utilizzo della crittografia in questo tipo di attacchi. Questo perchè la crittografia può essere adottata per scopi etici, come l'autenticazione, la firma digitale e le comunicazioni sicure, ma, come appena visto, anche per creare danno ad un dispositivo o ad un utente.

1.2 Crittografia e Malware

Con il termine crittografia si intende la pratica di rendere illeggibile una determinata informazione a tutti coloro che non sono in possesso della cosiddetta chiave. Nella quotidianità, la crittografia viene adottata per tantissimi scopi etici. Basti pensare all'adozione di questa nelle comunicazioni via web, nelle trattazioni con le banche multicanale che ci consentono di effettuare i nostri movimenti comodamente da casa, nel salvataggio delle nostre password presso database di siti web affidabili come i noti Amazon, Google, Facebook, e via

via discorrendo. Nel corso dei secoli, la crittografia si è notevolmente evoluta, passando dal noto `Cifrario di Cesare` a cifrari ben più noti e complessi come l'AES (adottato in larga scala dal governo degli Stati Uniti d'America) e RSA (Anche questo di origine americana e brevettato al MIT, fonda la sua sicurezza sulla complessità di fattorizzazione di numeri primi molto grandi).

Dal punto di vista informatico, la crittografia è quindi un tassello fondamentale, soprattutto data la diffusione in larga scala delle reti di calcolatori sulle quali possono essere eseguiti, per esempio, sistemi di gestione domotici o sistemi informativi aziendali. Nondimeno, la sicurezza dei dati assume un aspetto ancora più importante quando si pensa al trasferimento di dati collegati a questo genere di sistemi su Internet.

Un malware crittografico è un programma che, per raggiungere il suo scopo, adotta delle primitive crittografiche. Le strategie di attacco di un malware possono essere varie. In linea di massima i malware crittografici possono operare come segue:

1. **Download e decodifica**: Ormai tutti i dispositivi mobili, soprattutto quelli Android, sono costantemente collegati alla rete. Con questa strategia di attacco, il malware attende che accada un evento che gli consenta di scaricare del codice malevolo criptato da un server al quale è collegato, decodificarlo e successivamente eseguirlo a Runtime. Questa, in realtà, è un'evoluzione di un'altra tecnica per la quale si aveva già all'interno dell'applicazione del codice criptato che veniva decodificato ed eseguito al momento dell'installazione. Questa tecnica viene bloccata grazie ad un nuovo meccanismo di sicurezza di Google che impone la completa compilazione del codice in fase di creazione dell'APK, facendo in modo che Google Play Protect (Sistema di verifica delle applicazioni di Google) sia in grado eventualmente di rilevarlo.

2. **Codifica e ricatto**: Come accennato in precedenza, un ransomware è un software malevolo che codifica le informazioni della vittima e le riporta allo stato originale solo a fronte del pagamento di un compenso all'attaccante (da cui il termine ransom, riscatto). Ora, immaginando quanto la vita privata e professionale delle persone sia ormai altamente condizionata e legata all'utilizzo di uno smartphone, possiamo effettivamente capire quanto questo genere di software sia pericoloso e possa avere un impatto molto critico sulla privacy degli utenti e sulla loro vita economica, anche perché i riscatti sono spesso alti e pagati in Bitcoin. In assenza di appropriati sistemi di difesa, una volta infettato un dispositivo, non è detto che l'attaccante non possa chiedere molteplici riscatti alla stessa vittima.

1.3 Contributi

Lo studio proposto in questa tesi si occupa di approfondire come la crittografia viene impiegata sui malware Android. In particolar modo, l'obiettivo è comprendere quali librerie crittografiche sono effettivamente impiegate in applica-

zioni Android, cercando di proporre possibili interpretazioni per il loro utilizzo. I contributi di questo lavoro possono essere riassunti in tre punti principali:

1. La realizzazione di un lavoro di studio dello stato dell'arte, e soprattutto di ricerca di quali librerie crittografiche (sia di sistema, che di terze parti) possano essere utilizzate su applicazioni Android.

2. Lo sviluppo di un modulo di analisi di applicazioni Android capace di estrarre le classi crittografiche relative alle librerie studiate durante la prima parte. Questo strumento è stato integrato in un sistema più complesso denominato AndroidMalwareCrypto, creato in collaborazione con la Masaryk University di Brno, il cui scopo è quello di fornire un dettagliato rapporto su come la crittografia viene impiegata su applicazioni Android.

3. Il test del modulo realizzato su un dataset di applicazioni android malevole.

I risultati ottenuti mostrano come, in maniera quasi sorprendente, i malware preferiscono utilizzare librerie di sistema consolidate, ignorando sostanzialmente le caratteristiche di sicurezza di tali librerie e concentrandosi sulla loro affidabilità. Questo pone diverse sfide agli analisti di malware, in quanto i comportamenti di tali applicazioni risultano essere molto simili, nella sostanza, alle applicazioni benigne.

.

Capitolo 2

Background Android

2.1 Introduzione ad Android

Android è un sistema operativo prodotto da Google e basato sul Kernel Linux. È stato pensato, in prima istanza, per sistemi come smartphone e tablet ma, con il passare degli anni, si è sempre più diffuso in ambiti come smart TV, smartwatches e smart glasses. Android è gestito dal progetto AOSP (Android Open Source Project) e distribuito con licenza Apache 2.0 che consente di non includere software coperto da copy-left.

Dal punto di vista strutturale Android è un sistema operativo, come molti altri, basato su dei livelli, ognuno dei quali si occupa di una determinata parte del funzionamento del sistema [5]. A basso livello, Android possiede un Kernel Linux che si occupa di fornire le chiamate di sistema che sono in grado di controllare i dispositivi hardware. In questo livello si trovano tutti i driver relativi ai dispositivi hardware.

Ad un livello intermedio si trovano le librerie e l'application framework. Le librerie sono raccolte di funzioni che mettono in comunicazione il kernel e l'application framework. Per quanto concerne l'application framework, questo mette a disposizione dei servizi che vengono offerti al livello superiore, i più rilevanti sono:

- **Activity Manager**: fornisce informazioni sull'interazione tra activity, services e sul processo che contiene l'esecuzione dell'applicazione.
- **Location Manager**: fornisce accesso al sistema di localizzazione e alla gestione del GPS.
- **Package Manager**: fornisce informazioni relative ai pacchetti delle applicazioni installate sul dispositivo in quel momento.
- **Telephony Manager**: fornisce l'accesso alle informazioni relative a servizi telefonici del dispositivo. Le applicazioni sono in grado, grazie a questa classe, di determinare lo stato dei servizi telefonici comprese alcune informazioni sull'abbonamento attivo.

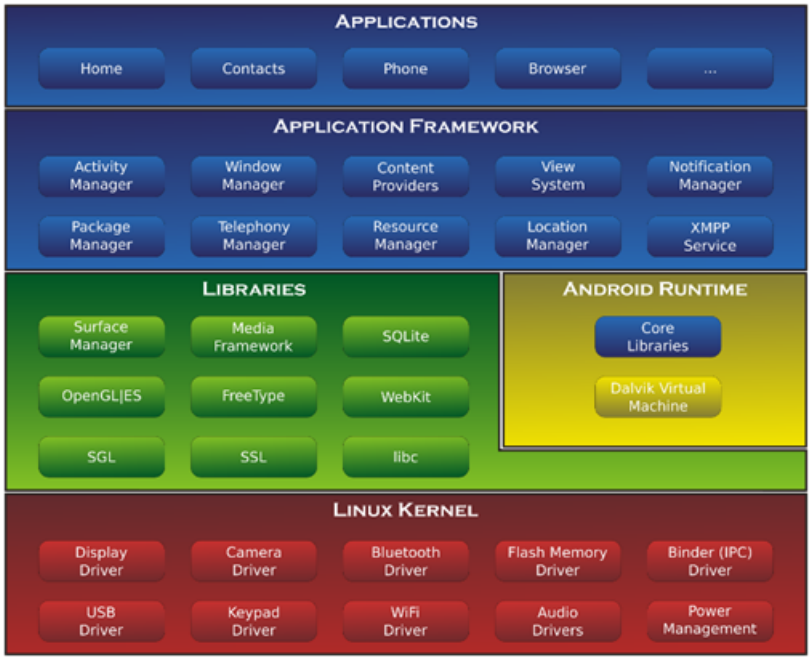

Figura 2.1: Struttura del sistema operativo Android

Accanto alle librerie, è presente l'Android Runtime il quale è formato dalle Core Libraries, ovvero le librerie che definiscono il nucleo primario del sistema operativo e da una specifica Runtime (Dalvik Virtual Machine o ART).

L'ultimo livello, quello di interesse per l'utente finale, è lo strato applicativo. In questo livello vengono inserite tutte le applicazioni installate sul sistema operativo. Le applicazioni che si possono trovare in questo livello sono in grado di utilizzare tutte le funzionalità contenute nell'application framework. Ciò consente, nel caso di attacchi, di mascherare le reali intenzioni di un attaccante dietro un'applicazione dalle sembianze normali. Una possibile vittima, se non particolarmente attenta, non farà caso alle autorizzazioni richieste dall'applicativo e questo potrebbe esporla a tutti i rischi descritti nel capitolo introduttivo ai malware.

Dal 2014, la Dalvik JVM viene completamente sostituita dalla runtime ART (Android RunTime) la quale cambia approccio del sistema operativo all'installazione e allo sviluppo delle applicazioni, e consente il passaggio da una struttura che consentiva la compilazione di alcune parti di codice solo in caso in cui queste venissero effettivamente richieste (che avveniva per effetto di una interpretazione Just In Time del codice), ad una situazione nella quale tutto il codice dell'applicazione viene compilato in fase di installazione.

2.2 Struttura Applicazioni Android

Con APK (Android Package) si intende una particolare categoria di file compresso che viene adottata per l'installazione delle applicazioni su sistemi operativi Android. Un file APK, in linea generale, contiene sempre i seguenti file:

- **META-INF**: cartella contenente le chiavi dei certificati pubblici (ma solo per quelle applicazioni che sono firmate);
- **res**: cartella con il file delle risorse tra le quali, ad esempio, l'elenco delle stringhe visualizzate nell'interfaccia o i colori usati per i componenti grafici;
- **AndroidManifest.xml**: costituisce il manifesto dell'applicazione e ne riporta i metadati;
- **classes.dex**: file contenente il codice compilato dei vari file contenenti codice sorgente;
- **resources.arsc**: file binario con la mappatura id-risorsa. L'id è un identificatore utilizzato per riconoscere, in maniera univoca, una determinata risorsa.

La creazione di un applicativo Android non troppo complicato è un'operazione che risulta abbastanza semplice. La stessa Google mette a disposizione dei suoi sviluppatori un IDE (Integrated Development Environment), noto come `Android Studio`, che consente di creare direttamente il file di installazione dell'applicazione, la quale può essere scritta in Kotlin o in Java, e la cui interfaccia è definita in uno o più appositi file XML. Per semplicità di esposizione tratterò, da questo momento, il codice di un'applicazione Android dando per scontato che questa venga scritta in Java.

2.3 Componenti Applicazione

Un'applicazione Android adotta il modello MVC (Model-View-Controller). Questa caratteristica consente di suddividere i vari oggetti in base alle loro responsabilità:

- **Model**: classi degli oggetti gestiti dall'applicazione;
- **View**: classi degli oggetti che consentono di visualizzare la GUI. Fanno parte di questa categoria tutti i file con estensione .xml che descrivono l'interfaccia grafica, le stringhe, i colori e le immagini usati nell'interfaccia;
- **Controller**: classi degli oggetti che gestiscono la logica dell'applicativo. Questa categoria è costituita dalle activities e dai services.

Per quanto riguarda la parte di implementazione della logica, Android consente di scrivere codice adottando principalmente due linguaggi:

1. **Kotlin:** Linguaggio di programmazione ideato e sviluppatto dalla nota azienda Jetbrains;

2. **Java:** Noto linguaggio di programmazione ideato dalla Sun Microsystems e attualmente di proprietà di Oracle.

Come accennato nella sezione introduttiva 1, questo studio sarà incentrato sull'ambiente Java. Un'applicazione per Android è strutturata in classi che possono rappresentare normali oggetti nel dominio del problema, o anche componenti dell'applicazione stessa come sliders e dialog fragments.

Sono ovviamente ammessi, come in ogni altro programma Java, interfacce, classi astratte, accessi al database e anche richieste http. Sono presenti poi le classi che descrivono il funzionamento dell'applicazione dal punto di vista logico. Queste classi descrivono gli algoritmi definiti dal programmatore e possono essere Activity (chiamate che generano le interfacce utente) o Servizi (componenti dell'applicazione che sono in grado di eseguire processi, anche di durata indefinita, in background) [5].

Diversamente dalle applicazioni classiche (dove il codice viene eseguito sequenzialmente se non per costrutti di iterazione e condizione) e dalle normali applicazioni Java (nelle quali il codice viene contenuto in delle classi e ne descrive il comportamento), in un'applicazione Android abbiamo anche il concetto di *evento*. Un evento è a sua volta un metodo la cui esecuzione dipende da una variabile aleatoria e da l'avvio all'esecuzione delle procedure e dei metodi che definiscono l'algoritmo vero e proprio.

2.3.1 Eventi

Trattandosi di un'applicazione Android, definiamo con il termine *evento* il verificarsi di una condizione che avvia l'esecuzione di un determinato pezzo di codice. In una normale applicazione Android, gli eventi che possiamo controllare sono diversi. Di seguito, si riporta un grafico estrapolato dal sito developers.android.com che rappresenta il ciclo di vita di un'Activity.

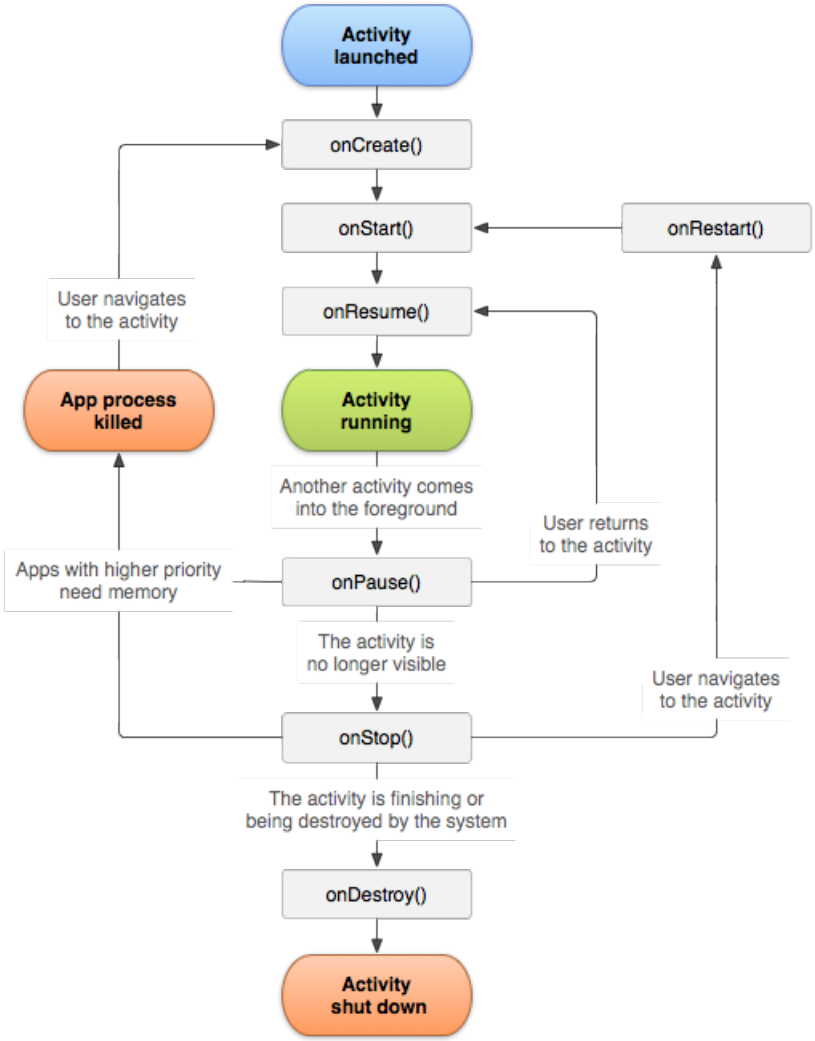

Figura 2.2: Ciclo di vita di un'activity

Presentiamo rapidamente i vari eventi riportati nella figura 2.2:

- **onCreate:** Chiamata quando l'activity viene creata per la prima volta;
- **onStart:** Viene chiamata quando si avvia l'esecuzione di una determinata activity;
- **onResume:** Chiamata quando l'activity è stata messa in pausa e ha ripreso la sua interazione con l'utente;

- **onPause:** Il contrario della onResume, viene eseguita quando l'utente cessa momentaneamente la sua interazione con l'activity;
- **onStop:** Chiamata quando l'activity cede il posto ad un'altra activity e questa rimane per un lungo periodo in interazione con l'utente;
- **onRestart:** L'activity che era in uno stato di stop, se nuovamente richiesta, triggera la onRestart.
- **onDestroy:** L'activity termina il suo ciclo di vita e viene distrutta.

2.4 Creazione di un'applicazione per Android

Per la creazione di un applicazione per Android abbiamo bisogno, oltre che delle conoscenze teoriche, di un'ambiente di sviluppo che ci consenta non solo di definire la nostra logica (scrivendo il codice Java) e di definire la nostra interfaccia (scrivendo il codice XML) ma che si occupi anche del coordinamento delle risorse e dell'organizzazione della fase di compilazione e impacchettamento di questa applicazione.

Per questo obiettivo, Google ci viene incontro e ci mette a disposizione un IDE dedicato proprio allo sviluppo di applicazioni Android, chiamato Android Studio. Tale IDE non è in grado di fare tutto da solo, ma ha bisogno di informazioni aggiuntive che possono essere configurate dal programmatore e che riguardano le Activity, i permessi richiesti dall'applicazione e altre informazioni utili per il sistema opertivo e per Google Play.

Tutte queste informazioni vengono inserite in un importantissimo file, denominato `AndroidManifest.xml`, e che di fatto costituisce il manifesto dell'applicazione [5]. Tale file risulta rilevante per capire quali permessi vengono richiesti dall'applicazione in relazione alle operazioni che questa dovrebbe svolgere. Per esempio, si ipotizzi che un'utente installi un'applicazione che promette di essere una calcolatrice. Se nell'AndrodidManifest.xml venisse inserito il permesso di accedere alla memoria del dispositivo, e dall'esame dei package risultasse che l'applicazione utilizzi primitive crittografiche, questo risulterebbe sospetto. Infatti, non richiedendo il problema di partenza queste caratteristiche per essere risolto, potrebbe indurre a pensare che l'applicazione svolga funzioni aggiuntive rispetto a quelle inizialmente dichiarate.

2.5 Crittografia su Android

Un'applicazione Android può adottare la crittografia per svariate motivazioni di sicurezza, come ad esempio lo scambio di messaggi in servizi come WhatsApp. Per adottare la crittografia all'interno di un'applicazione, abbiamo bisogno di una o più primitive crittografiche e di una parte di codice che si occupi di effettuare la codifica e la decodifica delle informazioni.

L'inserimento di questa parte di codice in un APK può essere fatto principalmente in tre modi:

1. **Uso di librerie scritte ad-hoc**: In linguaggi come Java è comune l'uso delle librerie, i cui vantaggi riguardano soprattutto la semplicità di ottenimento delle funzioni crittografiche necessarie. Inoltre, la maggior parte di queste librerie implementa varie primitive crittografiche, e questo dà la possibilità agli sviluppatori di scegliere quali utilizzare, o eventualmente combinarle in una certa sequenza.

2. **Uso di codice nativo**: con il miglioramento delle tecnologie informatiche, questo studio si è spostato anche sulla capacità dei calcolatori di svolgere operazioni in maniera molto veloce, e avere così la possibilità di creare dei cifrari molto più avanzati. Nel 1972 vediamo nascere il linguaggio C per il quale esistono, al giorno d'oggi, centinaia di librerie che si occupano di crittografia. Dal momento che nella programmazione, soprattutto quella orientata agli oggetti, è prevalente il principio DRY (Don't Repeat Yourself), in Java esiste una particolare sintassi che consente l'importazione di codice scritto in linguaggio C. Possiamo immaginare come sia semplice prendere una libreria C che si occupa di crittografia, importarla in un codice Java di un'applicazione Android, e utilizzarne le funzioni all'interno del normale codice Java. Fra i vantaggi di questo approccio, si osserva che le funzioni scritte in C sono computazionalmente più veloci di quelle scritte in Java. Inoltre, le librerie C sono generalmente meglio testate.

3. **Scrittura del codice ex-novo per l'obiettivo**: Questa modalità ci consente di avere funzioni crittografiche custom, ma allo stesso tempo si potrebbero introdurre falle di sicurezza nelle procedure di codifica che non renderebbero i dati effettivamente sicuri.

2.6 Malware su Android

Con le tecnologie dell'informatica moderna, è possibile sviluppare non solo software capace di risolvere problemi ad elevata complessità computazionale (o di affrontare addirittura delicate questioni etiche), ma anche di realizzare programmi che hanno l'obiettivo di recare danno alla macchina ospitante e fornire un guadagno ai il loro creatori. Tali software sono tipicamente noti come *malware*.

Nel 2019, il noto produttore di anti-virus Kaspersky ha rilasciato una buona sintesi sulle possibili le categorie di malware [6], che di seguito riportiamo:

- **Trojan**: Questo tipo di malware viene definito `cavallo di Troia` per la sua abilità di essere spesso mascherato da software legittimo. In realtà, viene usato dai cyber criminali per ottenere l'accesso sul sistema dell'utente in modo da svolgere varie operazioni tra le quali aprire backdoor, installare rootkit, lanciare exploit oppure di adottare le strategie tipiche dei trojan banker, ddos o ransom.

- **Banker**:Un trojan banker è un malware sviluppato al fine di rubare i dati di accesso ai sistemi di Online-Banking.

- **Ddos**: Un malware della categoria Ddos (Distributed Denial of Service) è un software malevolo che, una volta installato, consente all'attaccante di aggiungere il dispositivo colpito ad una rete definita come bot-net (Network of BOT), una rete di computer, spesso anche molto vasta, che può essere controllata da un'individuo al fine di sferrare attacchi DOS verso server, generando un'altissimo traffico dovuto all'altissimo numero di richieste effettuate dai bot.

- **Ransomware**: Un ransomware o trojan ransom è un particolare tipo di malware che, una volta installato, codifica le informazioni dell'utente e richiede un riscatto per averle indietro.

- **Worm**: Con il termine worm (dall'inglese verme) s'intende una categoria di malware che è in grado di creare molto velocemente tantissimi duplicati di se stesso e di auto diffondersi. Al contrario dei normali virus (i quali hanno bisogno di un programma host per operare), questi sono completamente indipendenti.

- **Spyware**: Gli spyware sono una categoria di software che potrebbe essere vista come una categoria lecita. Essi consentono di ottenere informazioni, spesso, sulle abitudini di un utente. Vengono definiti spie per la loro capacità di ottenere queste informazioni a insaputa dell'utente. Per questo motivo, mettono a disposizione delle funzionalità che possono essere utilizzate da un'attaccante.

- **Adware**: Adware è il nome dato a programmi progettati per visualizzare annunci pubblicitari sul computer, reindirizzare le richieste di ricerche verso siti Web pubblicitari e raccogliere dati di marketing sull'utente, ad esempio quali tipi di siti Web visita, allo scopo di mostrare annunci personalizzati.

- **Pornware**: Pornware è il nome che viene attribuito a quella classe di software, installato deliberatamente dall'utente o a sua insaputa, che riproducono materiale pornografico sul dispositivo sul quale sono installati.

- **Riskware**: I Riskware sono tutti quei software che, nelle mani di utenti malintenzionati, sono in grado di creare un danno all'utente che possa essere un furto di dati o addirittura una riduzione drastica delle prestazioni del dispositivo che viene colpito.

- **Cryptojackers**: I cryptojacker sono i software che effettuano il cryptojacking, ovvero l'utilizzo non autorizzato di un dispositivo al fine di usarne la potenza computazionale per il mining (pratica di estrazione di crypto valuta). In seguito ad un attacco di questo genere di malware si riscontra un'importante consumo energetico e un notevole rallentamento del dispositivo per l'utente, questo avviene perchè un gran numero di risorse della vittima vengono costantemente allocate per soddisfare l'algoritmo di mining. Ovviamente l'attaccante che ottiene crypto valuta senza dover sostenere gli importanti costi dovuti all'energia consumata dalla `farm` ottiene

un guadagno a discapito degli utenti colpiti. Fortunamatamente, sempre secondo le stime di Kaspersky, dal punto di vista aziendale solo il 18% delle grandi aziende viene colpito da questo genere di attacco, probabilmente per via delle loro reti interne spesso molto estese e performanti.

Queste categorie di malware, nate ovviamente come ogni altro software su computer, sono realizzabili anche per dispositivi Android, e tutti i rischi descritti precedentemente sono quindi riportati anche su dispositivi mobili.

Capitolo 3

Stato dell'arte Scientifico

I lavori scientifici legati allo studio della crittografia su Android hanno riguardato soprattutto lo studio del non corretto utilizzo di API crittografiche in applicazioni benigne. L'obiettivo finale è quello di evitare l'utilizzo scorretto delle API crittografiche e questo obiettivo si può perseguire attraverso diverse fasi, elencate di seguito.

1. **Riconoscimento**. In prima istanza, ci si è concentrati sul riconoscere l'utilizzo scorretto di determinate API. Questo avviene definendo delle regole che delineano il comportamento scorretto e che vengono verificate sulla chiamata effettiva dell'API.
2. **Verifica**. Il secondo step prevede la verifica dell'efficacia di queste regole nella rilevazione di comportamenti anomali.
3. **Raccolta API**. Vengono osservate tutte le librerie crittografiche in cui sono presenti API scorrette e memorizzate.
4. **Correzione**. L'ultima fase prevede, infine, la correzione automatica dell'errore rilevato.

Di seguito, le quattro fasi descritte sopra verranno discusse più nel dettaglio.

3.1 Fase di Riconoscimento

Al fine del riconoscimento del comportamento scorretto delle API crittografiche, si è pensato di definire delle regole che potessero essere utilizzate dagli sviluppatori in modo da evitare di incorrere in questi errori. La definizione di queste regole può essere fatta in modi diversi:

- **Creazione manuale delle regole**: Gli sviluppatori di applicazioni per Android utilizzano le API crittografiche per la messa in sicurezza di informazioni sensibili. Uno studio portato avanti da M. Egele mette in evidenza

i dati ottenuti dall'analisi, statica e dinamica, svolta su oltre 10.000 applicazioni tra le 11.000 che utilizzano API crittografiche su Google Play. Lo studio mostra che, tra queste, l'88% compie almeno un errore di comportamento nell'API crittografica [7]. Anche nell'articolo [8], si svolge uno studio simile ma con un dataset molto più ridotto e i risultati sono praticamente identici.

Questi studi consentono di prendere coscienza del fatto che gli abusi sulle API crittografiche sono un problema che affligge una parte abbondante delle applicazioni offerte sullo store di Google. Tuttavia,questi sono dati osservati direttamente dalle analisi senza formulare un modello di riconoscimento di tali comportamenti. Questo obiettivo viene perseguito da Shuai, Guowei e Tao [9]. In questo studio, riescono creare un modello per il riconoscimento dei comportamenti scorretti e a sviluppare un sistema in grado di utilizzare questo modello per rilevare eventuali errori, in automatico, all'interno di un dataset. Il CMA (Crypto Misuse Analyzer) si occupa di riconoscere, staticamente, i rami che utilizzano le primitive crittografiche e successivamente memorizzare le chiamate utilizzate. Solo al termine di questa analisi tutte queste chiamate vengono confrontate con il modello e vengono segnalati i comportamenti scorretti rilevati. Il problema di questi procedimenti sta nel fatto che non sono facilmente scalabili.

- **Riconoscimento delle correzioni da GIT**: Questo genere di studio si fonda sul ragionamento che i commit che vanno a sostuire vecchie parti di codice molto spesso sostituiscono delle parti dove sono presenti dei bug. Su questo assunto si può basare la conclusione che, se la parte di codice sostituita è la chiamata ad una API crittografica, molto probabilmente la versione precedente presentava delle falle nella sicurezza o degli errori di utilizzo. Inoltre, bisogna tenere presente il progresso tecnologico e l'evoluzione del codice. Infatti, l'adozione di nuove primitive crittografiche e l'aggiornamento delle vecchie rende velocemente obsolete le vecchie API. In questo contesto, Paletov [10] approccia il problema analizzando le modifiche contenute nei commit di GIT. Sostanzialmente si ha un'analisi di migliaia di modifiche e, in base a determinate astrazioni, si è in grado di distinguere le modifiche inutili e di prendere in considerazione le sole modifiche alle API crittografiche, le quali potrebbero introdurre migliorie dal punto di vista della sicurezza e consentono di ottenere delle regole di utilizzo delle API stesse. Il sistema sviluppato da Paletov in `javax.crypto` filtra il 99% del codice superfluo e, come previsto, del codice ottenuto l'80% prevede modifiche di sicurezza.

I risultati ottenuti da Paletov sono sorprendenti. Tuttavia, sono state rilevate in seguito diverse criticità. In [11], si mette in evidenza che non è possibile definire manualmente tutte le regole per tutte le API, ma che un'approccio automatico come quello definito da Paletov non può essere considerato sicuro al 100%, in quanto la stragande maggioranza delle API contiene degli errori. Dallo studio portato avanti su oltre 40.000 applicazioni reali, in moltissimi casi

un'approccio automatico produce falsi risultati perchè la correzione potrebbe, anche se accidentalmente, essere essa stessa l'errore.

3.2 Fase di Verifica

Una volta ottenute le regole di utilizzo, si passa alla fase che prevede di verificarle su dataset reali. Dagli studi appena evidenziati, emerge che l'analisi dinamica è funzionale nel caso di piccoli dataset, mentre l'analisi statica adottata da Egele [7] risulta molto efficace su dataset di grandi dimensioni.

Al fine di ridurre il divario tra sviluppatori ed esperti crittografi, lo studio proposto in [12], anziché adottare le medesime regole statiche adottate negli studi precedenti, definisce un linguaggio formale denominato CrySL, che può essere adottato dagli esperti di crittografia per definire il giusto modo di utilizzare le API crittografiche che loro stessi mettono a disposizione degli sviluppatori. In questo studio viene non solo definito il linguaggio, ma viene anche implementato un compilatore in grado di convertire le direttive CrySL in micro sistemi di analisi statica. Questo sistema consente agli sviluppatori Java e Android di verificare da subito il codice da loro scritto tramite le direttive CrySL. L'autore, in questo studio, è riuscito a rilevare almeno un errore nel 95% dei 10.000 casi esaminati. Questo è un risultato senza precedenti soprattutto contando che, secondo lo stesso autore, basta anche un solo errore per rendere l'applicazione completamente inefficace dal punto di vista della sicurezza.

Dal punto di vista del protocollo TSL, invece, lo studio in [13] basato su attacchi MiTM (Man in The Middle) dimostra che questo protocollo rientra tra i più sicuri rilevando delle falle sulle API nell'8% dei casi sugli oltre 13.000 applicativi esaminati.

3.3 Fase di Raccolta API

Una volta definite le modalità con le quali si può presentare un comportamento scorretto in un API crittografica, il passo successivo è quello di attribuire questi comportamenti scorretti a determinate librerie. In questo contesto, gli autori di [14], tramite il software da loro sviluppato noto come `BinSight`, hanno analizzato oltre 132.000 applicazioni raccolte nel 2012 e 2015, ed hanno stimato che il 90% degli errori nelle API riguarda circa 600 librerie di terze parti.

Un'altro problema rilevante è da ricercarsi nelle cosiddette `app clones`. Quese sono fondalmentalmente delle applicazioni lecite che sono state decompilate e ricompilate con l'aggiunta di codice malevolo. A riguardo, alcuni ricercatori cinesi in [15] hanno presentato l'approccio WuKong, un procedimento diffuso che mira al riconoscimento di questo genere di applicazioni. La prima fase prevede il riconoscimento di applicazioni sospette tramite il confronto di determinate caratteristiche semantiche, mentre la seconda fase, più specifica, cerca determinate librerie di terze parti.

3.4 Fase di Riconoscimento.

Una volta definito un dataset di regole utili al riconoscimento degli errori di comportamento relativi all'utilizzo di API crittografiche, si procede alla correzione di questi errori. Gli studi relativi allo sviluppo di Embrodery in [16] e Firebugs in [17], si riferiscono entrambi a dei sistemi in grado di rilevare un errore e di correggerlo automaticamente. La necessità di avere del software di questo tipo nasce dal fatto che, come indicato da Zhang in [16], gli altri produttori spesso interrompono gli aggiornamenti di sicurezza sui vecchi modelli (al contrario di Google, che tiene costantemente aggiornati tutti i modelli di Nexus), lasciando le eventuali falle presenti disponibili e facilmente sfruttabili dopo lo `zero-day`.

3.5 Studio della crittografia nei malware Android

Nel contesto della ricerca relativa all'utilizzo delle API crittografiche, tutti gli studi precedentemente citati trattano di applicazioni generiche e soprattutto prelevate dal Google Play Store (per il semplice fatto di essere popolari tra gli utenti), ma nessuno studio si è focalizzato prevalentemente su applicazioni malevole.

La fase sperimentale descritta nel prossimo capitolo è stata effettuata in collaborazione con uno studio presentato con la Masaryk University (Brno, Rep. Ceca), il quale ha l'obiettivo di colmare questa lacuna, stabilendo come determinate API crittografiche possano essere utilizzate su malware Android. Dato il tipo di ricerca che lo studio si propone, possono essere utilizzate diverse tecnologie per il rintracciamento di queste API, come regole ed euristiche, fino all'uso di tecniche di machine learning.

Capitolo 4

Analisi di Librerie Crittografiche

In questa tesi, ci si è occupati della ricerca e della raccolta di alcune librerie crittografiche, native e non, che possono essere adottate nel processo di sviluppo di un'applicazione Android. Con *libreria crittografica* si intende una raccolta di primitive crittografiche implementate mediante funzioni o mediante metodi di oggetti specifici. Dal punto di vista dell'implementazione queste primitive crittografiche costituiscono cifrari o funzioni di hash.

Un *cifrario* è un algoritmo il quale opera in modo da rendere semanticamente insensato un messaggio o un informazione. Gli esempi spaziano dai cifrari storici (come ad esempio il `cifrario di Cesare` o quello di `Vigenere`), fino a sistemi molto più complessi come RSA, DES e AES (Advanced Encryption System). In particolare, l'ultimo rappresenta l'algoritmo crittografico adottato dal Governo degli Stati Uniti d'America.

Una *funzione di hash* è una funzione matematica che consente di mappare una qualunque informazione in una stringa di lunghezza fissa. La differenza più importante tra i due sta nel fatto che un cifrario, con l'apposita chiave, può essere decodificato mentre una funzione di hash è per definizione non invertibile.

Di seguito, verranno elencate alcune librerie impiegate nella piattaforma Android, e verranno spiegate nel dettaglio le principali differenze tra queste. L'ecosistema Android mette a disposizione dei propri sviluppatori delle librerie che si occupano di implementare sia funzioni crittografiche che funzioni hash. Tuttavia, esistono anche svariate librerie alternative che spesso vengono proposte al posto di quelle di sistema. La scelta più adeguata fra librerie di sistema e di terze parti dipende da diversi fattori. Di seguito, si elencano i principali:

- **Varietà di cifrari e funzioni di hash:** Una delle variabili più importanti per la scelta di una libreria crittografica è il numero di cifrari che mette a disposizione degli sviluppatori.

- **Linguaggio adottato per la creazione della libreria:** Anche il linguaggio utilizzato dagli sviluppatori della libreria è importante per questo tipo di scelta. Alcuni linguaggi di programmazione (come il linguaggio C) consentono di scrivere programmi molto veloci (con ridotto consumo di memoria). In situazioni dove la velocità del linguaggio Java non è sufficiente per soddisfare i requisiti richiesti dagli sviluppatori, potrebbe essere una buona alternativa optare su una libreria sviluppata ad esempio in C.

- **Affidabilità libreria:** L'implementazione dei metodi contenuti in una libreria crittografica può contenere degli errori di programmazione (`bug`). Le librerie di terze parti spesso non vengono sottoposte allo stesso processo di `bug-fixing` portato avanti per librerie standard come `javax.crypto`. Questa caratteristica potrebbe spingere uno sviluppatore ad affidarsi maggiormente ad una libreria testata su larga scala. Per questo motivo, alcune librerie proprietarie vengono sviluppate tramite il processo TDD (Test Driven Development), processo che consente di definire dei test unitari prima ancora che il programmatore inizi a stilare il codice della libreria, riducendo così al minimo il numero di bug presenti, per via della scarsità dei test svolti, e rendendo più affidabili anche queste categorie di librerie.

Durante questo studio, verranno presentate librerie appartenenti principalmente a 3 categorie:

1. Librerie di sistema (Fornite da Java o ecosistema Android);
2. Librerie di terze parti (Scritte in Java);
3. Librerie di terze parti native (Scritte in altri linguaggi come C o C++);

4.1 Analisi di Librerie Crittografiche di Sistema

Il JDK (Java Development Kit) mette a disposizione delle implementazioni crittografiche che spaziano su diversi aspetti caratterizzati da una moltitudine di primitive e tecniche crittografiche che coprono diverse aree della sicurezza (come l'autenticazione e le comunicazioni sicure) e che vengono raccolte principalmente in due package: `javax.crypto` e `java.security`.

Entrambe fanno parte della così detta JCA (Java Cryptography Architecture), un framework che offre diverse tipologie di implementazioni tra cui:

- Funzioni di Hash;
- Crittografia simmetrica (ovvero basata su una chiave in grado di codificare e di decodificare) e asimmetrica (ovvero basata su una chiave pubblica utilizzata per codificare e una privata, nota solo al possessore, utilizzata per decodificare);
- Curve ellitiche;

- Generatori di numeri random sicuri.

Questi package implementano, tra gli altri, dei servizi che, per ragioni storiche, vengono definiti export-controlled. Stiamo parlando delle firme di *java.security* e di alcuni cifrari in *javax.crypto*. Dire che questi servizi sono export-controlled allude ad una ferrea regolamentazione statunitense in materia di esportazioni che si estende anche all'ambito del software e che, nel caso di Java, impone il possesso di una licenza o di un'eccezione di licenza prima di poter sbloccare ed utilizzare una parte del codice che, in questo caso, è costituita da servizi che implementano crittografia con chiavi di una certa dimensione. [18, 19]

Un'altra libreria di sistema, questa volta fornita dall'ecosistema Android è quella presente nel package `androidx.security.crypto`. Facente parte del progetto AndroidX questa libreria ha come scopo finale quello di rimpiazzare la vecchia libreria di supporto di Android, ovvero la JNA, rinnovandone le funzionalità e aggiungendone di nuove tra cui la crittografia. Il package precedentemente indicato contiene delle classi per la configurazione, creazione e decodifica di file criptati. Purtroppo, al momento implementa solo AES a 256bit con la Galois-Counter Mode (metodo di divisione in blocchi di alcuni cifrari a chiave simmetrica nota per la sua efficienza) [20].

4.2 Analisi di Librerie Crittografiche di Terze Parti

Fra le librerie di terze parti troviamo sia librerie completamente scritte in Java (e di proprietà di grosse multinazionali come Google e Facebook) e altre scritte in Java, ma con l'utilizzo di tecniche tra cui la chiamata `system.native` che consente l'import di un codice compilato, con l'obiettivo di rendere disponibili in Java vecchie librerie di altri linguaggi.

Alcune tra queste librerie sono:

- **Conceal**: Questa libreria, di proprietà di Facebook, presente nel package `com.facebook.crypto`, mette a disposizione dei metodi di codifica e decodifica molto più veloci di quelli presenti nella JCA e in altre librerie come Bouncy Castle [21] e si promette di mantenere gli stessi standard di velocità e sicurezza anche nella gestione di grosse moli di dati.
  Il grafico in figura 4.1 mostra la differenza tra le prestazioni di questa librerie e altre due librerie la JCA (rappresentata come Java) e Bouncy Castle, mostrando quanto effettivamente siano ridotti i tempi di elaborazione della libreria rispetto alla concorrenza.

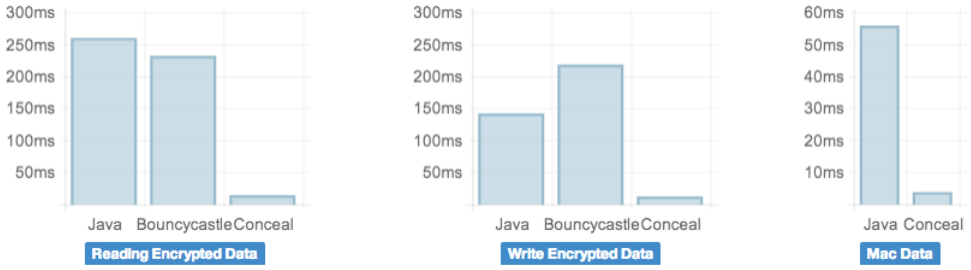

Figura 4.1: Prestazioni di Facebook Conceal paragonate ad alcune concorrenti

- **Tink**: Libreria di proprietà di Google, è un progetto open-source che promette di essere di facile utilizzo per gli sviluppatori. Secondo il team di crittografi e ingegneri di Google che la hanno realizzata, è in grado di ridurre le falle della crittografia grazie ad un processo di sviluppo ricco di revisioni del codice e test. Il codice, assieme alla sua descrizione, sono disponibili sulla pagina GitHub del progetto [22].

- **Bouncy Castle**: Bouncy Castle è una delle librerie crittografiche più conosciute. Offre delle Api crittografiche molto leggere, altri servizi come providers per JCA e JCE (Java Cryptography Extension), e dei metodi per la codifica e decodifica di certificati X.509 (standard per definire il formato dei certificati a chiave pubblica e delle autorità di certificazione) nelle versioni 1,2 e 3.

4.3 Analisi di Librerie Crittografiche Native

Come nell'introduzione, Java mette a disposizione dei propri sviluppatori delle tecnologie che consentono l'importazione di codice sorgente scritto in linguaggio C. Questa possibilità, combinata alla nota rapidità di esecuzione di questo linguaggio di programmazione, consente l'utilizzo di librerie crittografiche, alcune delle quali presenti da molto tempo, con un grande numero di primitive disponibili.

Lo stesso discorso viene naturalmente riproposto anche per il C++, che eredita gran parte dei pregi del linguaggio C aggiungendo la possibilità di programmare ad oggetti.

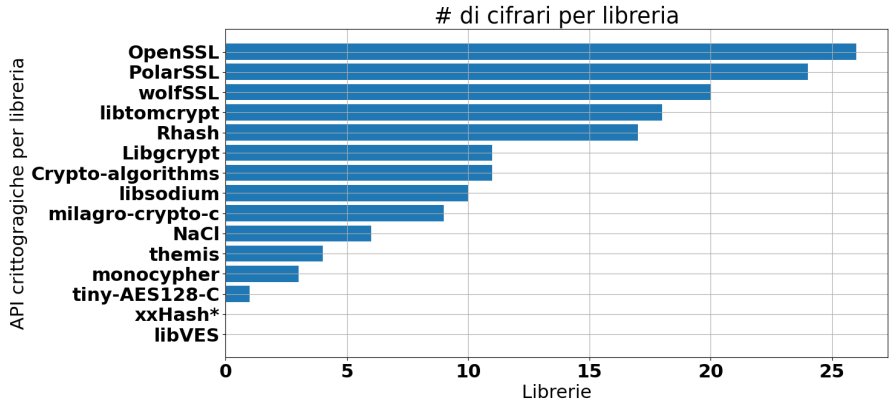

Figura 4.2: Numero di primitive crittografiche per libreria implementata in linguaggio C

Come si evince dalla figura 4.2, in termini di numero di primitive adottate, le due librerie di maggior interesse tra quelle sviluppate in linguaggio C sono OpenSSL e PolarSSL. OpenSSL è un'implementazione open source dei protocolli SSL e TLS [23] per la certificazione e la comunicazione cifrata, mentre PolarSSL implementa i medesimi protocolli ma è coperta da licenza Apache 2.0 (freeware). Queste due librerie offrono quindi un gran numero di primitive crittografiche e la rapidità del linguaggio C.Sono disponibili tantissimi wrapper che le rendono compatibili con Java, soprattutto per OpenSSL.

Un'altra libreria importante per quanto riguarda la categoria di librerie C è Libsodium la quale è il classico esempio di una vecchia libreria importata in Java tramite tecnologie come la JNI (Java Native Interface). Questa libreria infatti è portata in altri linguaggi come C++, Python e Java mediante i così detti wrapper. In questo caso è stata esaminata una delle tante librerie wrapper per Java che si trovano sul web, `libsodium-jni`. I metodi implementati da questa libreria mettono a disposizione implementazioni per diversi tipi di crittografia, tra cui le curve ellittiche, e le funzioni di hash, tra cui SHA512 e SHA256 [24]. Anche questa libreria è open-source e disponibile su GitHub [25].

Dal punto di vista del C++, si hanno ancora più possibilità. Le librerie C++ hanno la possibilità di utilizzare la programmazione ad oggetti, e questo rende molto più semplice l'implementazione di alcuni cifrari particolarmente complessi. Inoltre, tali librerie rendono più semplice la gestione, una volta importata, della configurazione dell'oggetto che si occuperà di fornire i metodi di codifica e decodifica. Nel grafico che segue, come nel precedente, viene mostrata la proporzione, in termini di numero di primitive crittografiche implementate in ogni libreria.

Possiamo confermare quello che è stato detto in precedenza e cioè che il C++ offre maggiori possibilità sul numero di primitive crittografiche disponibili.

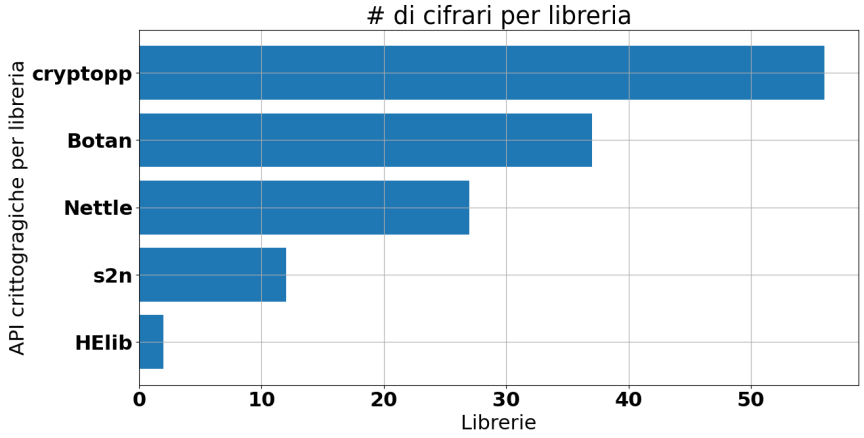

Figura 4.3: Numero di primitive crittografiche per libreria implementata in linguaggio C++

La libreria di maggior impatto è `cryptopp`. Questa libreria è ancora oggi in fase di espansione (l'ultima release risale al 2 Gennaio 2021) ed è disponibile su GitHub[26]. Le tecnologie crittografiche che offre riguardano le categorie dei cifrari a blocchi, varie funzioni hash, curve ellittiche e anche schemi di padding e di autenticazione [27].

4.4 Confronto tra le Librerie Crittografiche Studiate

In generale, i punti chiave per i quali uno sviluppatore preferisce una libreria rispetto alle altre sono principalmente tre:

1. **Semplicità**: Uno sviluppatore tende a preferire una determinata libreria perchè la trova semplice da utilizzare. Quando una libreria, soprattutto se dotata di oggetti, mette a disposizione delle configurazioni e dei metodi intuitivi, è sicuramente preferibile a una che non ha tali caratteristiche.

2. **Prestazioni**: Una delle caratteristiche fondamentali di una libreria, soprattutto se deve essere adottata nello sviluppo di un malware, è che data un operazione questa dev'essere svolta adottando il minor numero di risorse. Questo consente di avere prestazioni più elevate e di essere più discreti (ad esempio, riducendo il consumo di batteria del dispositivo target).

3. **Funzioni disponibili**: Se una libreria offre più API rispetto ad un'altra potrebbe essere più appetibile agli occhi di uno sviluppatore. Questo approccio aumenta il tempo di apprendimento per la singola libreria, ma

richiede l'apprendimento di una sola libreria per tante possibili implementazioni diverse. Grazie a questo assunto si può immaginare quanto, i tempi di sviluppo, vengano ridotti.

Tra le librerie crittografiche studiate, quelle in C++ sono le più promettenti, in quanto mettono a disposizione degli sviluppatori un gran numero di primitive crittografiche gestite tramite opportuni oggetti con dei parametri configurabili ben definiti. Questi parametri, in linea di massima, riguardano la primitiva che si intende adottare e, se necessaria, la tipologia di padding. Questo ragionamento potrebbe indurci a pensare che sia molto probabile che degli eventuali attaccanti preferiscano adottare librerie esterne tramite la JNI, la JNA o tramite degli appositi wrapper disponibili.

In realtà questo non è sempre vero. Nella valutazione sperimentale, si potrà osservare che l'attaccante tende a preferire l'uso di librerie di sistema tradizionali, prediligendo la semplicità e ricchezza della documentazione alle prestazioni.

Capitolo 5

Valutazione Sperimentale Librerie Crittografiche

Nell'ambito di questa tesi, è stato sviluppato un framework per l'estrazione di librerie e classi crittografiche da applicazioni Android. Il framework sviluppato si occupa di ricercare, tra gli import di ogni classe di ogni APK, l'eventuale traccia di librerie e classi crittografiche fra i package definiti nella sezione 4. Il sistema prende in ingresso un file di configurazione, il quale indica i parametri di base come la directory che contiene gli APK, il numero di di threads, da utilizzare, etc. In uscita, il sistema restituisce l'incidenza delle librerie crittografiche in tutto il dataset.

Il sistema riconosce l'utilizzo di librerie crittografiche in due casi:

1. Se l'APK importa librerie, di sistema o di terze parti scritte in Java, che implementano primitive crittografiche.
2. Se l'APK importa librerie che abilitano all'uso di tecnologie come JNI e NDK in quanto, tramite queste tecnologie, le eventuali API crittografiche potrebbero essere contenute nel codice scritto in un linguaggio diverso dal Java.

5.1 Strumento per la Ricerca delle Librerie

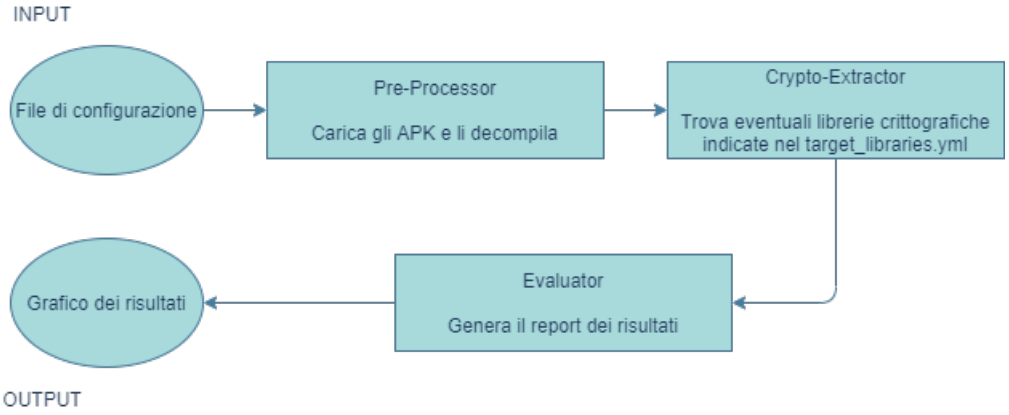

Figura 5.1: Struttura del tool di estrazione librerie crittografiche

Il framework è diviso in tre fasi distinte, ognuna dei quali si occupa di un determinato aspetto dell'analisi.

1. **Pre-processor**: Questa fase si occupa dell'estrazione di tutte le informazioni contenute nel APK. Nello specifico, tramite delle tecniche e degli strumenti di reverse engineering, si è in grado di ottenere gran parte del codice sorgente di cui è formato l'APK. Per ottenere questo risultato, il Pre-processor sfrutta due librerie Python per l'analisi e il reverse engineering di codice Java e Android. Queste sono Androguard [28] e PyJadx [29], basata sullo strumento di decompilazione Jadx [30]. Nello specifico, Androguard si occupa di fornire dei servizi di interpretazione del bytecode che costituisce il codice compilato di un APK, mentre PyJadx consente di ottenere il codice sorgente decompilato in Java attraverso l'analisi del bytecode. Il risultato dell'interazione di questi due è l'ottenimento di un listato codice ad alto livello che viene tradotto dal bytecode prelevato da un determinato APK.

   Questo processo, per aumentare l'efficienza, può essere parallelizzato. Il numero di threads da utilizzare per l'analisi e il dataset (costituito a sua volta da una lista di path di file.apk o da un path che indica una cartella dove questi file sono contenuti) costituiscono il file di configurazione di base.

2. **Crypto-extractor**: Questo livello costituisce il fulcro dello strumento di analisi. Si occupa di effettuare il `parsing` del codice ottenuto dal livello precedente, come mostrato in figura 5.1, e di ricercare all'interno di questo codice dei pacchetti e delle classi riconducibili a librerie elencate in un file di configurazione denominato `target_libraries.yml`.

Il risultato di questa elaborazione è un dizionario formato, come chiave, dal nome del della classe rilevata e, come valore, dal numero di volte in cui questa viene rilevata all'interno del dataset. La scelta di adottare un salvataggio con questa tipologia di oggetto è dovuta alla praticità con la quale si può interagire con il dizionario e alla semplicità con la quale questo può essere adottato per creare dei grafici i quali consentono una comprensione migliore per l'essere umano.

3. **Evaluator**: Una volta terminata l'esecuzione dei livelli 1 e 2, l'evaluator si occupa di trasformare il dizionario messo a disposizione dal livello 2 e di trasformarlo in un grafico grazie alla libreria `matplotlib` [31]. Risulta infatti più semplice carpire le informazioni di maggior rilievo semplicemente guardando il grafico generato dallo strumento e di organizzare, dato un certo dataset di librerie da ricercare tra gli APK, un trend in ordine decrescente delle prime 10 librerie rilevate per numero di rilevazioni.

Nelle prossime sezioni, verranno presentati i risultati di alcune analissi effettuate su due dataset di ransomware. I risultati sono stati sintetizzati in due grafici che mostrano quali sono le librerie e le classi che vengono maggiormente adottate per lo sviluppo di malware.

5.2 Analisi di un Dataset di Ransomware

Lo strumento di analisi è stato applicato per un primo esperimento su un dataset di 672 malware, composto da ransomware che sono stati rilasciati nel 2015 (Heldroid - [32] [33]).

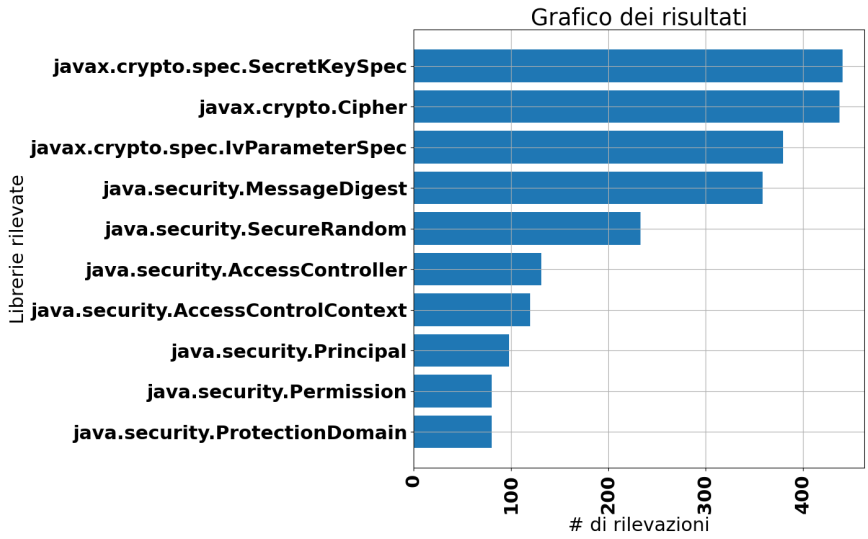

Figura 5.2: Risultati dell'analisi sul dataset Heldroid

L'obiettivo dell'analisi è comprendere come dei malware tipicamente orientati alla crittografia si comportino dal punto di vista delle API crittografiche. Il risultato di questa prima analisi mostra che tutti i malware richiamano pacchetti appartenenti alle librerie di sistema (quindi classi site sotto i pacchetti `java.security` e `javax.crypto`).

I risultati sopra ottenuti sono stati ulteriormente esaminati tramite di un dataset più grande, costituito da malware non necessariamente crittografici, ottenuto dall'Università di Cagliari con la Masaryk University (Brno, Rep. Ceca) e costituito da 886 malware.

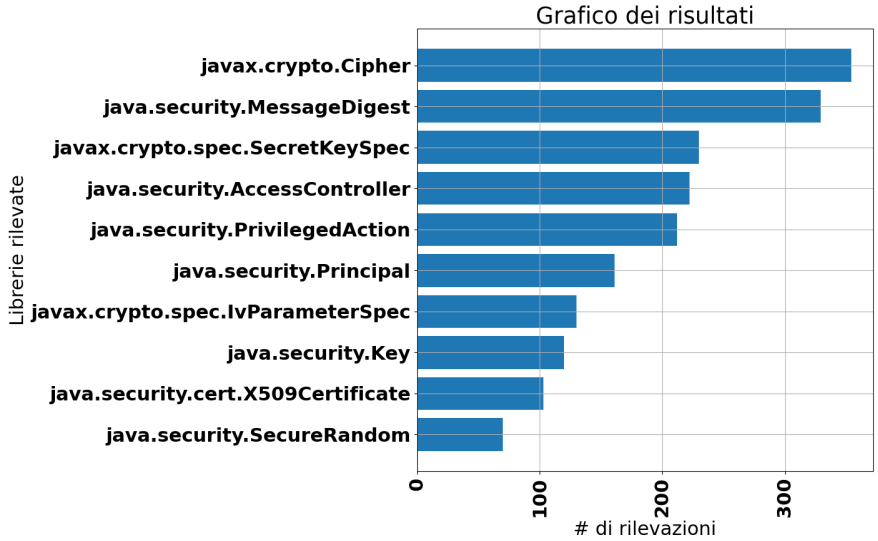

Figura 5.3: Risultati sul dataset di UniCa e Masaryk University

Come si evince dal grafico 5.3, il comportamento precedentemente rilevato è confermato. In particolare, si osserva che, fra le top-10 librerie crittografiche, nessuna appartiene a librerie di terze parti. Nella prossima sezione, si discuteranno alcune possibili motivazioni per tale comportamento.

5.3 Studio dei risultati

I risultati mostrati nei grafici 5.2 e 5.3 mettono in evidenza un comportamento abbastanza inaspettato. Il fatto che gli sviluppatori di malware prediligano le librerie di sistema per lo sviluppo dei loro applicativi maligni potrebbe essere dovuto ad una serie di fattori che seguono:

- L'attaccante vuole `mimetizzare` il malware come applicazione legittima. Per una libreria di sistema come `java.security` o come `javax.crypto`, risulta più plausibile che questa venga utilizzata in un'applicazione Android rispetto ad una qualunque libreria di terze parti.

- L'attaccante è `pigro` (in gergo, *lazy*): Il programmatore pigro è un programmatore che segue alla lettera il principio `Don't repeat yourself`. Un'attaccante potrebbe seguire tale principio estremizzandolo e imponendosi di adottare solo librerie già presenti nell'ecosistema che intende attaccare.

- Funzionalità fornite dalle librerie: Le librerie di sistema vantano dei metodi e delle funzionalità che consentono, ad esempio, di amministrare i permessi relativi all'applicazione e di gestire le risorse del dispositivo. Queste

funzionalità, per un malware che deve danneggiare soprattutto le risorse, risultano essere utili e preferite a implementazioni di terze parti.

- Affidabilità libreria: Un'attaccante, nella sua attività di sviluppo, ha bisogno di metodi affidabili. Se, per esempio, nella fase di codifica venisse sollevata un'eccezione le sue intenzioni potrebbero non andare a buon fine. Come evidenziato in 4 le librerie di sistema sono spesso sottoposte a lunghi processi di test è per questo hanno un'affidabilità praticamente garantita.

5.3.1 Classi e pacchetti rilevati

La classe che viene maggiormante rilevata, ricoprendo rispettivamente la prima e la seconda posizione in 5.3 e in 5.2 è la `javax.crypto.Cipher`.

Dalla consultazione della documentazione ufficiale Java su questa classe si possono ottenere informazioni davvero preziose. Queste informazioni riguardano le primitive crittografiche che questa classe implementa [34].

- AES
- DES
- RSA

Queste sono le primitive crittografiche implementate da questa classe. Il fatto che siano in numero così ristretto ci consente di avere un vantaggio. Infatti, sapendo che queste sono le primitive crittografiche maggiormente utilizzate si potrebbe applicare uno studio mirato sulla chiamata di metodi relativi a queste primitive.

Un'altra classe particolarmente rilevante è la `javax.crypto.spec.SecretKeySpec`.

Questa classe ha lo scopo di generare una chiave partendo da un certo vettore di byte adottando un certo algoritmo contenuto nella `Java Security Standard Algorithm Names List`[34].

Da questa informazione possiamo assumere che, se il vettore di inizializzazione utilizzato per la creazione della chiave è noto, e se l'algoritmo è a chiave simmetrica, si potrebbe utilizzare il vettore di inizializzazione (ottenuto tramite reverse engineering) per rigenerare la chiave e decodificare le eventuali informazioni codificate.

Dal punto di vista dei permessi che questi applicativi gestiscono non possono non essere citati:

- `java.security.Permission`: Classe che definisce i permessi su una determinata risorsa di sistema.
- `java.security.ProtectionDomain`: Classe che si occupa di gestire gruppi di permessi. Non solo è in grado di gestire i permessi statici dell'applicazione, ma per consentire l'utilizzo delle cosiddette `Dynamic Security Policies` è strutturato in modo da poter gestire anche il binding dinamico dei permessi. Questo significa che l'applicazione potrebbe modificare i

permessi richiesti quando è già stata installata sul dispositivo della vittima e attivando in questo modo l'esecuzione di determinati frammenti di codice che svolgono delle operazioni illecite sulla memoria o sulle informazioni private dell'utente.

5.4 Limiti dello Strumento di Analisi

Lo strumento di analisi sviluppato nel contesto di questo elaborato è uno script python, molto leggero e parallelizzabile, che consente di utilizzare al meglio le prestazioni della propria macchina creando vari threads e dividendo il carico del dataset su più esecuzioni simultanee. I punti chiave, quindi, sono semplicità di configurazione, parallelizzazione e rappresentazione dei risultati in una chiave grafica molto semplice ed intuitiva da interpretare.

In contrapposizione, questo strumento presenta, come ogni sistema informatico, delle imperfezioni. Nel caso specifico queste limitazioni riguardano principalmente 3 aspetti:

1. **Rilevazione di falsi positivi**: Le tecnologie come JNI e JNA complicano questo genere di analisi perchè consentono il caricamento di un file compilato che potrebbe avere un nome qualsiasi. Anche memorizzando i nomi rilevati nei cosiddetti `import nativi` basterebbe cambiare il nome del file utilizzato per il caricamento del codice malevolo per evitare il filtro e passare come applicazione legittima, tralasciando il fatto che il semplice atto utilizzo di tecnologie come JNI non implica che questo applicativo sia un malware. Inoltre, l'analisi si concentra sull'utilizzo delle librerie e lo strumento quindi non è in grado di creare un `grafo delle chiamate` per riconoscere eventuali accessi a primitive crittografiche.

2. **Vulnerabilità delle API custom**: Lo strumento non ha un sistema di esecuzione sicura del codice ottenuto dalla decompilazione dell'APK. Sapendo questo, se un'attaccante sviluppasse un'API crittografica custom scritta in Java direttamente come metodo di qualcuna delle sue classi, questa non verrebbe rilevata durante l'analisi. Per la rilevazione di queste API crittografiche, si potrebbe usare un sistema che simula l'esecuzione dell'applicazione ed addestrato a riconoscere determinati `pattern` nel codice (Sistemi basati su machine-learning).

3. **Dimensione del target_library.yml**: Questo file costituisce una sorta di database NoSQL per la memorizzazione delle librerie da ricercare nel codice decompilato degli APK. Vien da se che se questo file è troppo piccolo o è carente di alcuni pacchetti noti per essere riconducibili alla crittografia, alcuni malware che adottano queste librerie potrebbero passare il filtro. Bisogna avanzare nella ricerca delle varie librerie e nell'analisi dei vari APK in modo da rinvenire eventuali nuovi pacchetti crittografici che possano essere aggiunti al dataset delle librerie note per affinare l'analisi.

Capitolo 6

Conclusioni

Il problema della difesa dei dispositivi Android dall'attacco di malware crittografici riveste un'importanza che va, di giorno in giorno, crescendo. Dallo studio dello stato dell'arte, è emerso che i lavori proposti si sono occupati di studiare l'uso non corretto della crittografia su applicazioni legittime. I risultati di queste ricerche, applicati all'ambito dello sviluppo, consentono di ridurre gli errori di configurazione e di utilizzo delle librerie crittografiche e, nel caso in cui questi errori siano voluti da un'attaccante che li ha intenzionalmente creati come vulnerabilità sfruttabili da un futuro attaco, cercare di correggerli. Tuttavia, poco è stato detto a riguardo di come la crittografia viene utilizzata dagli attaccanti in applicazioni malevole.

Il lavoro proposto in questa tesi si è occupato, pertanto, dello studio di librerie crittografiche su applicazioni non legittime. Dopo aver illustrato quali librerie crittografiche possano essere, potenzialmente, utilizzate da applicazioni Android, è stato realizzato uno strumento capace, su larga scala, di estrarre tali librerie crittografiche dalle applicazioni. Questo strumento costituisce parte di un framework sviluppato in collaborazione con la Masaryk University (Brno Rep. Ceca), il quale si propone di fornire un dettagliato rapporto sull'utilizzo della crittografia su applicazioni Android malevole.

Lo strumento realizzato è stato poi testato su un dataset composto da oltre 1500 malware, mostrando che le principali librerie e classi crittografiche appartengono solo a librerie di sistema, comunemente utilizzate da applicazioni benigne. I risultati di questo studio consentono di stabilire che la differenza fra applicazioni legittime o meno è molto sottile nell'utilizzo della crittografia. In generale, gli attaccanti prediligono librerie consolidate dal punto di vista dell'affidabilità, piuttosto che funzionalità rare o avanzate. Pertanto, al fine di rilevare comportamenti malevoli dal punto di vista crittografico, è necessario utilizzare tecnologie avanzate (come il machine learning) per comprendere la relazione fra crittografia e contesto di esecuzione.

Bibliografia

[1] SkyTg24. Tutti i numeri di android, da startup a leader del mercato "mobile". https://tg24.sky.it/tecnologia/2018/06/07/android-dominio-mercato-mobile, 2018.

[2] Google. Android security and privacy 2018 year in review. https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf, 2019.

[3] GData. Cyber attacks on android devices on the rise. https://www.gdatasoftware.com/blog/2018/11/31255-cyber-attacks-on-android-devices-on-the-rise, 2018.

[4] Kaspersky. It threat evolution q2 2020. mobile statistics. https://securelist.com/it-threat-evolution-q2-2020-mobile-statistics/98337/, 2020.

[5] Developers. Android documentation. https://developer.android.com/docs, 2020.

[6] Kaspersky. Classificazioni del malware. https://www.kaspersky.it/resource-center/threats/types-of-malware, 2019.

[7] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer; Communications Security*, CCS '13, page 73–84, New York, NY, USA, 2013. Association for Computing Machinery.

[8] Alexia Chatzikonstantinou, Christoforos Ntantogian, Georgios Karopoulos, and Christos Xenakis. Evaluation of cryptography usage in android applications. In *Proceedings of the 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (Formerly BIONETICS)*, BICT'15, page 83–90, Brussels, BEL, 2016. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[9] Shao Shuai, Dong Guowei, Guo Tao, Yang Tianchang, and Shi Chenjie. Modelling Analysis and Auto-detection of Cryptographic Misuse in Android

Applications. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pages 75–80, Dalian, 2014. IEEE.

[10] Rumen Paletov, Petar Tsankov, Veselin Raychev, and Martin Vechev. Inferring crypto api rules from code changes. *SIGPLAN Not.*, 53(4):450–464, June 2018.

[11] J. Gao, P. Kong, L. Li, T. F. Bissyandé, and J. Klein. Negative results on mining crypto-api usage rules in android apps. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 388–398, 2019.

[12] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs. In Todd Millstein, editor, *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*, volume 109 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:27, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[13] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 50–61, New York, NY, USA, 2012. Association for Computing Machinery.

[14] Ildar Muslukhov, Yazan Boshmaf, and Konstantin Beznosov. Source attribution of cryptographic api misuse in android applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, page 133–146, New York, NY, USA, 2018. Association for Computing Machinery.

[15] Haoyu Wang, Yao Guo, Ziang Ma, and Xiangqun Chen. Wukong: A scalable and accurate two-phase approach to android app clone detection. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, page 71–82, New York, NY, USA, 2015. Association for Computing Machinery.

[16] X. Zhang, Yuanyuan Zhang, Juanru Li, Yikun Hu, Huayi Li, and D. Gu. Embroidery: Patching vulnerable binary code of fragmentized android devices. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 47–57, 2017.

[17] Larry Singleton, Rui Zhao, Myoungkyu Song, and Harvey Siy. Firebugs: Finding and repairing bugs with security patterns. In *Proceedings - 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2019*, Proceedings - 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2019, pages 30–34. Institute of Electrical and Electronics Engineers Inc., May 2019. 6th IEEE/ACM International Conference on Mobile

Software Engineering and Systems, MOBILESoft 2019 ; Conference date: 25-05-2019.

[18] Oracle. Java security overview. https://docs.oracle.com/javase/10/security/java-security-overview1.htm#JSSEC-GUID-0C458D46-BA4F-4091-817B-9902B6E18240, 2018.

[19] Ucop.edu. Frequently asked questions about export control compliance. https://www.ucop.edu/ethics-compliance-audit-services/compliance/international-compliance/export-faq.html#1.

[20] Developers. androidx.security.crypto references. https://developer.android.com/reference/androidx/security/crypto/package-summary, 2020.

[21] Facebook. Introducing conceal: Efficient storage encryption for android. https://engineering.fb.com/2014/02/03/android/introducing-conceal-efficient-storage-encryption-for-android/, 2014.

[22] Google. Tink. https://github.com/google/tink, 2020.

[23] UniPa. Introduzione a openssl. http://ocs.unipa.it/AL/al291.htm, 2000.

[24] NaCL. Nacl: Networking and cryptography library. https://nacl.cr.yp.to/index.html, 2016.

[25] Github user. libsodium-jni. https://github.com/joshjdevl/libsodium-jni, 2019.

[26] Weidai. Cryptopp. https://github.com/weidai11/cryptopp.

[27] Weidai. Crypto++ library. https://www.cryptopp.com/, 2014.

[28] Android. Android. https://github.com/androguard/androguard, 2019.

[29] abdihaikal. Pyjadx. https://github.com/abdihaikal/pyjadx, 2019.

[30] Skylot. jadx. https://github.com/skylot/jadx, 2020.

[31] matplot. matplotlib. https://matplotlib.org/, 2020.

[32] necst. heldroid. https://github.com/necst/heldroid, 2016.

[33] Niccolò Andronio, Stefano Zanero, and Federico Maggi. HelDroid: Dissecting and detecting mobile ransomware. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, volume 9404 of *Lecture Notes in Computer Science*, pages 382–404, 2015. https://github.com/phretor/publications/raw/master/files/papers/conference-papers/andronio_heldroid_2015.pdf.

[34] Oracle. Java doc. https://docs.oracle.com/en/java/javase/14/docs/api/, 2020.