UNIVERSITÀ DEGLI STUDI DI GENOVA
Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Costruzione di modelli comportamentali per l'analisi di sicurezza di applicazioni iOS

Relatore:
Prof. Alessio MERLO

Candidato:
Gianluca BARBERA
Matricola n. 3515987

Anno Accademico 2016 – 2017

Acronimi

API Application Program Interface.

CISC Complex Instruction Set Computer.

DRM Digital Rights Management.

ECID Electronic Chip ID.

IDE Integrated Development Environment.

MVC Model View Controller.

PCTL Probabilistic real time Computation Tree Logic.

RISC Reduced Instruction Set Computer.

UID device's Unique ID key.

Indice

Elenco delle figure

Elenco delle tabelle

Non potevo non inserire una dedica alla mia famiglia
a mia mamma, a mio papà e a mia sorella
alle nonne e a tutti gli altri parenti

Ai miei amici, ai compagni di studio
e al gruppo del CSecLab

Alla mia ragazza Ylenia per il suo sostegno

Grazie a tutti per avermi accompagnato
durante questi anni di studio

Capitolo 1

Introduzione

La Apple è al secondo posto nella classifica degli smartphone più venduti (dati aggiornati al primo trimestre del 2017), subito dopo Samsung e seguita da Huawei, con una quota di mercato del 13,7% e con più di 50 milioni di iPhone in circolazione. (dati di Gartner, Inc.[1])

Figura 1.1: Logo di iOS 11

Noti per la loro appartenenza ad un sistema chiuso, i dispositivi mobili della Apple sono sempre stati considerati molto sicuri e, sebbene in parte ciò risulti vero, il mito della sicurezza dell'Apple Store[2] è stato più volte sfatato nel tempo. Il caso di AceDeceiver[3] (il quale sfruttava una vulnerabilità di FairPlay[4] DRM) è una recente dimostrazione di quanto il processo automatico di controllo delle applicazioni dello Store non sia infallibile.

Installando una app sul nostro smartphone potremmo chiederci se esista un modo per analizzarne il comportamento prima di avviarla, oppure se ci sia uno strumento che ci permetta di verificare se un'applicazione violi

[1] `http://www.gartner.com/newsroom/id/3725117`
[2] `https://www.apple.com/itunes/`
[3] `https://researchcenter.paloaltonetworks.com/2016/03/`
[4] `https://developer.apple.com/streaming/fps/`

politiche di sicurezza generali o specifiche per le nostre necessità prima di installarla. Allo stato attuale un simile strumento non esiste, in quanto manca un approccio sound e completo per ricostruire il comportamento delle applicazioni iOS[5].

In questa tesi viene proposto un metodo che, data un'applicazione iOS, permetta di costruire un grafo che modelli il flusso delle chiamate del programma, nel quale siano evidenziate in particolare quelle verso funzioni critiche per la sicurezza del dispositivo e dell'utente. In tal modo, si vuole costruire un modello delle applicazioni che consenta di identificare, in modo automatico, eventuali comportamenti che violino specifiche politiche di sicurezza tramite analisi statica.

E' doveroso sottolineare che un tale approccio al problema dell'analisi dei comportamenti di applicazioni mobili non è nuovo, in quanto applicato già nel mondo Android[6], tuttavia, per i sistemi iOS al momento non esistono tecniche o strumenti applicabili alle tecnologie più recenti introdotte dalla Apple.

1.1 Apple e iOS

Apple Inc.[7] è una società americana che produce sistemi per l'ufficio, computer e dispositivi multimediali con sede a Cupertino ed è attualmente una delle aziende più conosciute, discusse e produttive al mondo. Con i suoi smartphone, spesso raggiunge record di vendita di 150 miliardi di dollari.

Apple ha creato diversi prodotti negli ultimi 10 anni tra cui 22 modelli di iPhone e 10 versioni di iOS (il suo sistema operativo mobile) di cui ha recentemente annunciato la versione 11[8] (che verrà rilasciata nell'autunno 2017).

[5] `https://www.apple.com/ios/`
[6] `https://www.android.com/`
[7] NASDAQ: AAPL
[8] Il logo di iOS 11 è mostrato in figura 1.1

I dispositivi

Attualmente iOS è disponibile su 5 categorie di dispositivi (vedi fig. 1.2)

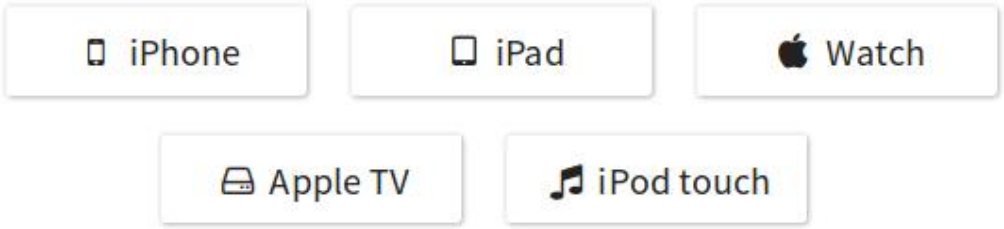

Figura 1.2: Tipologie di dispositivi iOS.

Ad oggi, questa è la lista aggiornata dei dispositivi usciti.

- iPhone 7 Plus (GSM)
- iPhone 7 (GSM)
- iPhone 7 Plus (Global)
- iPhone 7 (Global)
- iPhone SE
- iPhone 6s+
- iPhone 6s
- iPhone 6
- iPhone 6+
- iPhone 5s (Global)
- iPhone 5s (GSM)
- iPhone 5c (Global)
- iPhone 5c (GSM)
- iPhone 5 (Global)
- iPhone 5 (GSM)
- iPhone 4[S]
- iPhone 4 (CDMA)
- iPhone 4 (GSM / 2012)
- iPhone 4 (GSM)
- iPhone 3G[S]
- iPhone 3G
- iPhone 2G
- iPad7,4
- iPad7,3
- iPad7,2
- iPad7,1
- iPad Pro 12.9-inch (Cellular)
- iPad Pro 12.9-inch (WiFi)
- iPad Pro 9.7-inch (Cellular)
- iPad Pro 9.7-inch (WiFi)
- iPad 5 (Cellular)
- iPad 5 (WiFi)
- iPad Air 2 (Cellular)
- iPad Air 2 (WiFi)
- iPad Mini 4 (Cellular)
- iPad Mini 4 (WiFi)
- iPad Mini 3 (China)
- iPad Mini 3 (Cellular)
- iPad Mini 3 (WiFi)
- iPad Mini 2 (China)
- iPad Mini 2 (Cellular)
- iPad Mini 2 (WiFi)
- iPad Air (China)
- iPad Air (Cellular)
- iPad Air (WiFi)
- iPad 4 (Global)
- iPad 4 (GSM)
- iPad 4 (WiFi)
- iPad 3 (GSM)
- iPad 3 (CDMA)
- iPad 3 (WiFi)
- iPad Mini (Global)
- iPad Mini (GSM)
- iPad Mini (WiFi)
- iPad 2 (Mid 2012)
- iPad 2 (CDMA)
- iPad 2 (GSM)

- iPad 2 (WiFi)
- iPad 1
- Apple Watch Series 1 (42mm)
- Apple Watch Series 1 (38mm)
- Apple Watch Series 2 (42mm)
- Apple Watch Series 2 (38mm)
- Apple Watch (42mm)
- Apple Watch (38mm)
- Apple TV 4 (2015)
- Apple TV 3 (2013)
- Apple TV 3
- Apple TV 2G
- iPod touch 6
- iPod touch 5
- iPod touch 4
- iPod touch 3
- iPod touch 2G
- iPod touch 1G

Il sistema iOS

iOS (precedentemente iPhone OS) è un sistema operativo mobile creato e sviluppato dalla Apple Inc. esclusivamente per il suo hardware. È il sistema che è attualmente installato in molti dei dispositivi dell'azienda ed è secondo per diffusione su scala globale dopo Android.

Da iOS 1 molte cose sono cambiate, oltre alle versioni del sistema, le modifiche nelle interazioni con l'hardware e le fasi d'avvio, anche il linguaggio di programmazione delle app è stato sostituito.

In iOS le applicazioni sono chiamate IPA[9] e contengono al loro interno file eseguibili decifrati solo a run-time. Fino al 2014, sviluppare IPA per iOS richiedeva un'approfondita conoscenza di Objective-C[10] con l'utilizzo di puntatori e dichiarazioni esplicite di classi e tipi primitivi, adesso invece l'attenzione della Apple si è diretta verso un linguaggio più semplice e facile da imparare: Swift.

Adottato dalla Apple nel 2014 [11], Swift 2 è stato subito inserito su Xcode[12], nonostante per lo sviluppo venisse ancora consigliato dall'azienda stessa l'utilizzo del precedente linguaggio. Il passaggio fra la versione 2 e la 3, avvenuto nell'autunno 2016, ha provocato diversi problemi a causa di una mancata retrocompatibilità, causando un rallentamento nell'abbandono di

[9] Le IPA sono l'equivalente per iOS delle APK di Android

[10] `developer.apple.com/documentation/objectivec`

[11] Williams, Owen (June 2, 2014). *Tim Berners-Lee's sixtieth birthday Apple announces Swift, a new programming language for iOS.* The Next Web.

[12] `https://developer.apple.com/xcode/`

Objective-C, pronosticabile con l'avvento della prossima versione di Swift[13], annunciata per l'autunno 2017.

> Swift è destinato ad essere più resiliente al codice errato ("più sicuro") rispetto ad Objective-C, e più conciso. Tuttavia, esso supporta molti concetti fondamentali che sono associati ad Objective-C; [...] . Per la sicurezza, Swift aiuta a prevenire errori di programmazione comuni come i puntatori nulli [...]. Ma di maggior rilievo è il fatto che Swift aggiunga il concetto di estensibilità del protocollo, un sistema applicabile a tipi, structs e classi. Apple lo promuove come un vero cambiamento nei paradigmi della programmazione definendolo "protocol-oriented programming" (programmazione orientata al protocollo) (Wikipedia: Swift)[14]

A proposito del sistema, è importante fare una distinzione fra le varie versioni, le quali introducono cambiamenti minimi dal punto di vista dell'esperienza utente, ma rilevanti per quanto riguarda la sicurezza. In alcune versioni, infatti, sono presenti delle vulnerabilità nel kernel[15] che possono essere sfruttate per la modifica del sistema stesso, ossia eseguire un jailbreak, l'acquisizione dei permessi di root, di cui parleremo in seguito, concentrandoci per ora sul comprendere la procedura di aggiornamento ad una versione successiva o precedente del sistema iOS (questo perché può portare all'installazione di un jailbreak aprendo così la strada anche ad altri tipologie di analisi).

Upgrade e Downgrade

Esistono due metodi per eseguire l'upgrade o il downgrade (rispettivamente l'aggiornamento del sistema ad una versione successiva e il ripristino ad una versione precedente) del dispositivo: attraverso iTunes[16] o tramite altri

[13]Di Swift 4 è già disponibile un ebook sul sito `developer.apple.com`: *The Swift Programming Language (Swift 4)*

[14]`https://en.wikipedia.org/wiki/Swift_(programming_language)`

[15]Nucleo del sistema operativo

[16]`https://www.apple.com/itunes/`

strumenti, come Prometheus[17] (ora integrato in futurerestore, sviluppato da tihmstar), che funzionano principalmente con dispositivi jailbroken[18].

Apple non incoraggia il downgrade dei dispositivi e limita questa pratica tramite l'interruzione (perlopiù senza annunci) della validità della firma di una versione non più supportata. Questi periodi di tempo sono chiamati finestre di firma (signing windows) e consentono sia l'upgrade che il downgrade alle versioni firmate. Durante una signing window è possibile anche scaricare i blob SHSH2[19], file che consentiranno poi, tramite strumenti non ufficiali, di procedere all'upgrade/downgrade anche al di fuori del periodo di validità di tale versione.

I blobs sono collegati al modello e all'ECID del dispositivo e sono disponibili unicamente durante i periodi di validità della signing window, inoltre non è possibile utilizzare un blob di un altro dispositivo per aggiornare il proprio.

Recentemente Apple ha deciso di intraprendere una "guerra" contro il jailbreak riducendo i periodi delle finestre di firma per prevenire lo sfruttamento delle vulnerabilità e quindi, potenzialmente, poter eseguire il jailbreak su una versione ancora supportata, avviando inoltre un Bug Bounty Program (programma a ricompense per la ricerca di errori nel codice) con premi fino a 200000 $ per chi riesce ad individuare problemi sulla sicurezza in iOS.

Jailbreak

> Il jailbreaking è il processo di rimozione delle restrizioni software imposte da Apple su iOS e tvOS: questo è ottenuto utilizzando

[17] `https://github.com/tihmstar/futurerestore`

[18] Dispositivi che hanno acquisito i permessi di root prendono l'aggettivo "jailbroken" che indica appunto l'installazione del jailbreak nel sistema

[19] I file SHSH2 si chiamano così per via della loro estensione, i dati al loro interno sono rappresentati in formato plist (Property List, è un formato tipico dei sistemi MACOS per la rappresentazione di oggetti serializzati, utilizza una struttura codificata in XML)

> una serie di exploit[20] software per ottenere l'accesso come root su iOS, permettendo l'installazione di applicazioni aggiuntive non disponibili tramite l'Apple App Store ufficiale.
>
> L'iOS jailbreaking risale all'originale iPhone nel luglio 2007 e continua ancora oggi. Apple ha risposto con gli aggiornamenti di iOS: patching e aggiornamento hardware. Le comunità di Jailbreaking non sono state legalmente minacciate. Lo stato giuridico del jailbreaking è poco chiaro nella maggior parte dei paesi; mentre molti proibiscono la manomissione di sistemi digitali, altri tollerano i jailbreak purché non violino i diritti d'autore.
> (Wikipedia: iOS Jailbreaking)[21]

Il jailbreak, potrebbe sembrare l'equivalente dei permessi di root su Android, in realtà è un'operazione più pericolosa per quanto concerne la sicurezza del dispositivo infatti, con questa operazione le applicazioni evadono i meccanismi di protezione come la sandbox (che vedremo in seguito) ed eseguono codice con permessi di amministratore.

Consci dei rischi, con il dispositivo jailbroken è però possibile utilizzare molti strumenti potenti; un esempio è Theos[22], il quale consente di eseguire analisi per la reverse engineering[23] di un'applicazione .deb (esten-

[20] Un exploit è un termine usato in informatica per identificare una tipologia di script, virus, worm o binario che, sfruttando una specifica vulnerabilità presente in un sistema informatico, permette l'esecuzione di codice malevolo su di esso con lo scopo di far ottenere all'attaccante l'acquisizione dei privilegi amministrativi (`https://it.wikipedia.org/wiki/Exploit`)

[21] `https://en.wikipedia.org/wiki/IOS_jailbreaking`

[22] `http://iphonedevwiki.net/index.php/Theos/Setup`

[23] Il processo di reverse engineering (anche chiamato in italiano ingegneria inversa) consiste nell'analisi dettagliata del funzionamento, progettazione e sviluppo di un oggetto (dispositivo, componente elettrico, meccanismo, software, ecc.) al fine di produrre un nuovo dispositivo o programma che abbia un funzionamento analogo, magari migliorando o aumentando l'efficienza dello stesso, senza in realtà copiare niente dall'originale; inoltre, si può tentare di realizzare un secondo oggetto in grado di interfacciarsi con il primo (`https://it.wikipedia.org/wiki/Reverse_engineering`)

sione normalmente utilizzata dalle applicazioni per jailbreak) o per creare *Tweaks*[24].

Nella maggior parte dei casi, con il jailbreak, è anche possibile aggiornare il dispositivo ad una versione successiva o precedente, utilizzando Prometheus. Questo processo richiede solo il blob SHSH2 (e a volte una nonce[25]) con la sola limitazione di poter eseguire l'aggiornamento entro la stessa versione, aggiornando ad esempio da 10 a 10.2 senza poter passare da 10 a 9.3.

Tra i principali utenti e i gruppi attualmente attivi nel mondo jailbreak figurano PanGu[26], KeenLab[27], Luca Todesco (noto come qwertyoruiop[28]), Jay Freeman (saurik[29]) e Team Tigris (capitanato da Antique_Dev[30]).

Figura 1.3: Logo di Cydia.

Jailbreak ha anche un proprio app store: Cydia (logo in figura 1.3), sviluppato da Jay Freeman (anche chiamato "saurik"). Il nome "Cydia" deriva dalla carpocapsa del melo (una particolare falena il cui nome scientifico è Cydia pomonella), il proverbiale "verme nella mela".

> Cydia è un'alternativa all'App Store Apple per i dispositivi "jailbroken" (in questo momento sono inclusi iPhones, iPads e iPod Touch) specializzata nella distribuzione di tutto ciò che non è un' "app".
>
> Cydia non è disponibile nell'App Store di Apple, né è un si-

[24]Applicazioni, estensioni o temi per jailbreak, la maggior parte dei tweaks sono discussi sulla comunità di Reddit `https://www.reddit.com/r/jailbreak/`

[25]Numero, in teoria, utilizzabile una volta sola, serve ad evitare che un messaggio già inviato sia rispedito uguale (replay attack)

[26]`http://en.pangu.io/`

[27]`http://keenlab.tencent.com/en/`

[28]`https://twitter.com/qwertyoruiopz`

[29]`https://twitter.com/saurik`

[30]`https://twitter.com/antique_dev`

to web, ma può essere installato sul dispositivo utilizzando un "jailbreaking" (`http://cydia.saurik.com/`)

Inoltre strumenti come Clutch[31] permettono di decriptare l'eseguibile di un'IPA. Tutto ciò per sottolineare l'utilità di un telefono jailbroken per l'analisi delle applicazioni iOS, purtroppo al momento della stesura della tesi non era disponibile il Jailbreak[32] per i device a disposizione (vedi figura 1.4).

Figura 1.4: Stato del jailbreak per l'ultima versione di iOS. - canijailbreak.com

Questo non dev'essere visto come un aspetto negativo; certamente, allo stato attuale, l'assenza del jailbreak limita le possibilità di analisi delle applicazioni ma, in questa tesi si discute un approccio alternativo che consente di analizzare la sicurezza delle applicazioni non cifrate e ricavarne un modello comportamentale. Discuteremo la possibile espansione del progetto in caso di rilascio del jailbreak nella sezione Sviluppi Futuri.

[31] `https://github.com/KJCracks/Clutch`
[32] `https://canijailbreak.com/`

1.2 Organizzazione della Tesi

Data l'impossibilità di utilizzare un dispositivo jailbroken, in questa tesi ci siamo concentrati sull'analisi delle applicazioni iOS, col fine di estrarre informazioni sulla struttura del file eseguibile per analizzarne il comportamento in termini di chiamate API critiche costruendone un modello per eseguire un'analisi della sicurezza.

L'obiettivo finale di questo progetto è realizzare uno strumento in grado di analizzare i possibili flussi di esecuzione di un applicazione iOS tramite l'analisi statica dei binari Mach-O[33], compilati partendo da codice Swift. I modelli generati saranno utilizzati per verificare formalmente alcune proprietà dell'applicazione.

Tuto ciò avviene attraverso la creazione di un Inter-Procedural Control Flow Graph (di cui tratteremo la definizione in seguito) dell'applicazione, e la sua traduzione in un modello PRISM (uno strumento per la modellazione formale e analisi di sistemi che presentano un comportamento casuale o probabilistico, anche questo verrà visto nel dettaglio più avanti). Quest'ultimo è testato su proprietà scritte in PCTL [34].

Strumenti

Per la parte hardware abbiamo a disposizione:

- Computer MAC con OSX Sierra[35] 10.12.4 (16 GB di RAM)
  Richiesto per questo progetto per la compatibilità con gli strumenti hardware e software utilizzati.

- iPhone 6s con iOS 10.2.1
  Utilizzato per la creazione per il testing di applicazioni.

[33] Mach-O è il formato dei file eseguibili in iOS

[34] La Probabilistic Computation Tree Logic è discussa nel dettaglio nel paper "A logic for reasoning about time and reliability" [1]

[35] `https://support.apple.com/it-it/HT201475`

Per la parte software invece abbiamo utilizzato:

- Xcode[36]
  IDE utilizzato per la creazione di applicazioni ioS sia in Swift che in Objective-C[37]

- PyCharm [38]
  IDE utilizzato per la creazione di applicazioni in Python[39]. Qui impiegato per la creazione del tool di reverse engineering degli eseguibili delle applicazioni iOS.

- radare2 [40]
  Disassemblatore e framework per la reverse engineering.

- QEMU [41], Oracle VirtualBox [42], Cydia Impactor [43]
  Altri tools usati per il testing

[36] `https://developer.apple.com/xcode/`
[37] I due principali linguaggi per lo sviluppo di applicazioni iOS, discuteremo in seguito delle loro caratteristiche
[38] `https://www.jetbrains.com/pycharm/`
[39] `https://www.python.org/`
[40] https://github.com/radare/radare2
[41] `http://www.qemu.org/`
[42] `https://www.virtualbox.org/`
[43] `http://www.cydiaimpactor.com/`

Capitolo 2

iOS Security

2.1 Panoramica sulla sicurezza del sistema

Apple ha progettato la piattaforma iOS incentrandola sulla sicurezza del proprio sistema. Dalla fase di avvio al download delle firme, esistono diversi livelli nell'infrastruttura di sicurezza dei dispositivi mobili.

Fase di avvio

In fase di avvio viene chiamato il Secure Boot Chain, un meccanismo che si occupa della verifica delle firme del loader di basso livello[1], del sistema di baseband mobile[2] e (dalla serie A7[3]) del Secure Enclave.

La System Software Authorization, presente nel Secure Boot Chain, è un meccanismo che impedisce l'esecuzione di una versione personalizzata del sistema, blocca inoltre il downgrade non autorizzato per impedire la possibile esecuzione di un sistema con vulnerabilità.

Il Secure Enclave fornisce protezione sui dati e dispone di una propria procedura d'avvio sicuro, mantenendo l'integrità della protezione anche se il kernel è stato compromesso.

[1] Componente di sistema che si occupa del caricamento dei programmi e delle librerie
[2] Sistema del controllo della ricezione
[3] Introdotta a partire dall'iPhone 5s

Touch ID

Il Touch ID utilizza le impronte digitali per rendere la crittografia del portachiavi[4] più robusta. Viene anche utilizzato per crittografare i file e, se l'opzione è attiva, dopo 10 letture di impronte digitali sbagliate, attiva la cancellazione di tutti i dati nel dispositivo[5].

Protezione dei file

Ci sono quattro classi di protezione dei file (di cui possiamo osservare il meccanismo di cifratura in figura 2.1):

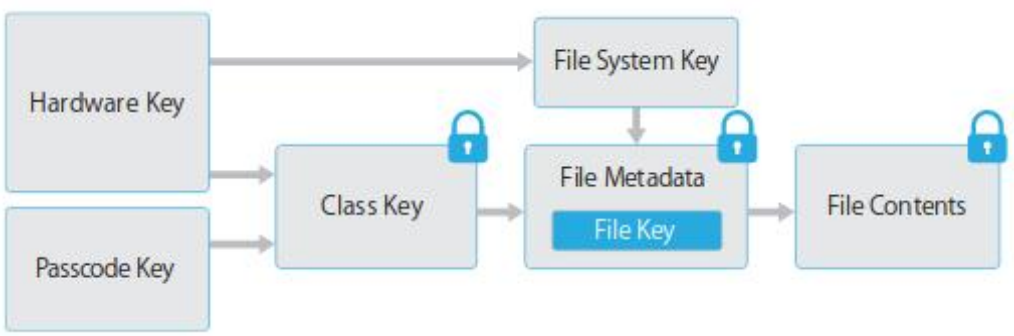

Figura 2.1: Meccanismo di protezione dei file su iOS.

1. **NSFileProtectionComplete**:
   La chiave della classe viene protetta da un'altra, derivata dal codice utente e dallo UID [6] del dispositivo. La chiave scade quando lo schermo del telefono viene bloccato.

[4]Il portachiavi contiene nomi utente e password salvati su Safari (il browser predefinito del telefono), dati delle carte di credito e credenziali delle reti Wi-Fi

[5]da 1 a 4 impronte sbagliate non sono introdotti ritardi per un altro tentativo; 5 errori, un minuto; 6, cinque minuti, 7-8, quindici minuti, 9, un'ora

[6]*Lo UID è unico per ogni dispositivo e non viene registrato dalla Apple o da i suoi rivenditori.* -Apple

2. **NSFileProtectionCompleteUnlessOpen**:
   Il file può essere aperto solo da chi ha la chiave temporanea e da chi lo ha già sbloccato. La chiave scade quando il telefono si riavvia.

3. **NSFileProtectionCompleteUntilFirstUserAuthentication**:
   Livello **predefinito** di protezione dei dati, una volta che l'utente accede (terminata quindi la fase di avvio del dispositivo), il file viene sbloccato.

4. **NSFileProtectionNone**:
   Nessuna protezione, il file è solo crittografato con lo UID.

2.2 Meccanismi di sicurezza delle App

Esistono diversi meccanismi di protezione per le applicazioni iOS che impediscono l'esecuzione di codice non autorizzato.

2.2.1 Firme e certificati

iOS richiede che tutti gli eseguibili debbano essere firmati con un certificato[7] valido rilasciato dalla Apple Certification Authority[8]. La firma identifica in modo univoco lo sviluppatore (singolo o azienda) che è stato registrato presso il programma Apple Developer[9].

> iOS consente agli sviluppatori di incorporare i framework all'interno delle proprie applicazioni, che possono essere utilizzati dall'applicazione stessa o da estensioni incorporate in essa. Per pro-

[7] I certificati impiegati si basano sullo standard X.509 (`http://www.ietf.org/rfc/rfc5280.txt`)

[8] In crittografia, una Certificate Authority o Certification Authority (CA), letteralmente Autorità Certificativa, è un ente di terza parte (trusted third party), pubblico o privato, abilitato a rilasciare un certificato digitale tramite procedura di certificazione che segue standard internazionali e conforme alla normativa europea e nazionale in materia. - Wikipedia

[9] La registrazione avviene online (`https://developer.apple.com/programs/enroll/`) e ha un costo (dati aggiornati a luglio 2017) da 99 $ all'anno per i singoli a 299 $ all'anno per la soluzione enterprise

> teggere il sistema e altre applicazioni dal caricamento di codice di terze parti all'interno dello spazio di indirizzi, il sistema eseguirà una convalida della firma sul codice di tutte le librerie dinamiche, alle quali un processo si connette, al momento del lancio. Questa verifica viene eseguita tramite l'identificatore di squadra (ID del team), estratto da un certificato rilasciato da Apple. Un identificatore di squadra (Team ID) è una stringa alfanumerica di 10 caratteri; ad esempio, 1A2B3C4D5F. Un programma può collegarsi a qualsiasi libreria fornita con il sistema purché possegga lo stesso identificatore di squadra. Da notare che poiché gli eseguibili che si avviano assieme al sistema, non hanno un identificatore di squadra, quindi possono collegarsi solo alle librerie che si avviano con il sistema stesso. (iOS Security White Paper [2])

Possiamo ugualmente generare un .ipa non firmato copiando la cartella con l'estensione .app che si trova in **Products** su Xcode in un'altra denominata **Payload** e utilizzando il seguente comando

```
zip -0 -y -r myAppName.ipa Payload/
```

Se il proprio cellulare non è jailbroken, non si può installare un'applicazione non firmata a meno che non si possegga un certificato di sviluppatore valido (o un ID Apple registrato ma, varrebbe solo per il proprio dispositivo ed avrebbe una durata limitata[10]) in tal caso, si potrebbe utilizzare Cydia Impactor [11].

Uno sviluppatore può ottenere la sua chiave pubblica digitando nel terminale

```
security find-identity -v -p codesigning
```

[10] In origine i certificati "self-signed" (quelli utilizzabili da un account gratuito) avevano una validità di 90 giorni, recentemente Apple l'ha ridotta a 7

[11] Cydia Impactor è uno strumento che consente di installare applicazioni non firmate sul proprio dispositivo tramite l'applicazione della firma al momento dell'installazione

Per firmare un'applicazione, Xcode utilizza uno strumento denominato *codesign* che può essere invocato dalla linea di comando, ad esempio per recuperare informazioni sui certificati.

```
codesign -vv -d appName.app
```

Vediamo un esempio di ciò che il lancio del comando produce su 4 applicazioni.

Smash Hit 1.4.0.ipa - smashhit.app

Executable=/.../smashhit.app/smashhit
Identifier=**com.mediocre**.smashhit
Format=app bundle with Mach-O universal (armv7 arm64)
CodeDirectory v=20200 size=6585 flags=0x0(none) hashes=320+5
location=embedded
Signature size=3487
Authority=Apple iPhone OS Application Signing
Authority=Apple iPhone Certification Authority
Authority=Apple Root CA
Info.plist entries=34
TeamIdentifier=**JFJ4SDLLV4**
Sealed Resources version=2 rules=14 files=2327
Internal requirements count=1 size=104

Commute 1.4.0.ipa - commute.app

Executable=/.../commute.app/commute
Identifier=**com.mediocre**.commute
Format=app bundle with Mach-O universal (armv7 arm64)
CodeDirectory v=20200 size=5284 flags=0x0(none) hashes=255+5
location=embedded
Signature size=3487
Authority=Apple iPhone OS Application Signing
Authority=Apple iPhone Certification Authority

Authority=Apple Root CA
Info.plist entries=35
TeamIdentifier=**JFJ4SDLLV4**
Sealed Resources version=2 rules=14 files=771
Internal requirements count=1 size=100

PinOut 1.0.2.ipa - pinout.app
Executable=/.../pinout.app/pinout
Identifier=**com.mediocre**.pinout
Format=app bundle with Mach-O universal (armv7 arm64)
CodeDirectory v=20200 size=9331 flags=0x0(none) hashes=284+5
location=embedded
Signature size=3925
Authority=Apple iPhone OS Application Signing
Authority=Apple iPhone Certification Authority
Authority=Apple Root CA
Info.plist entries=34
TeamIdentifier=**JFJ4SDLLV4**
Sealed Resources version=2 rules=14 files=1446
Internal requirements count=1 size=100

Si possono ricavare diverse informazioni eseguendo questo comando, ad esempio si può legare l'*Identifier* al *Team Identifier*, si può avere una stima della dimensione dell'applicazione e si può ricavare l'architettura del processore per la quale l'app è stata compilata. Da notare nell'esempio soprastante che la differenza nella Signature size potrebbe essere causata da un rinnovo della firma. [12]
Eseguendo lo stesso comando su una mia app invece si ottiene:

[12]pinout.app è infatti più recente rispetto alle altre due

GB-FoodTracker.ipa - FoodTracker.app
Executable=/.../FoodTracker.app/FoodTracker
Identifier=com.gianluca.barbera.FoodTracker
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20100 size=2193 flags=0x2(adhoc) hashes=63+3
location=embedded
Signature=**adhoc**
Info.plist entries=30
TeamIdentifier=not set
Sealed Resources version=2 rules=13 files=101
Internal requirements count=0 size=12

Si può subito notare che l'app è compilata per un'architettura diversa (quella thin, usata per il debug con l'emulatore di Xcode), ha meno elementi nell'Info.plist, ma soprattutto la firma è segnata come *adhoc*.

Per poter firmare l'app occorre un certificato valido.
Verifico "manualmente" qual è il mio certificato con il comando

```
security find-identity -v -p codesigning
```

e firmo la mia app usando, nel mio caso, il comando

```
codesign -f -s
"iPhone Developer: gianlucam08@gmail.com (NGZ3ULM865)"
FoodTracker.app
```

Essendo valida, ricontrollando la firma del file, ottengo

Executable=/.../FoodTracker.app/FoodTracker
Identifier=com.gianluca.barbera.FoodTracker
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20200 size=2208 flags=0x0(none) hashes=63+3
location=embedded

Signature size=4715
Authority=**iPhone Developer: gianlucam08@gmail.com (NGZ3ULM865)**
Authority=**Apple Worldwide Developer Relations Certification Authority**
Authority=**Apple Root CA**
Signed Time=10 nov 2016, 14:15:30
Info.plist entries=30
TeamIdentifier=9HZX8SJTWS
Sealed Resources version=2 rules=13 files=101
Internal requirements count=1 size=200

Un certificato valido potrebbe però esserlo, come in questo caso, solo per l'installazione sul proprio dispositivo poiché non inserito nei certificati riconosciuti dell'Apple CA per la distribuzione sullo store (iTunes). Infatti, l'applicazione sviluppata è eseguibile solo su emulatore di Xcode o sull'iPhone utilizzato per la tesi.

La firma, permette inoltre di verificare che un'app non sia stata alterata. Infatti è sufficiente lanciare da terminale

```
codesign --verify NomeApp.app
```

e se non viene stampato alcun messaggio d'errore vuol dire che l'applicazione non è stata modificata. Inoltre un'applicazione, una volta firmata, viene protetta sullo Store anche dal Fair Play DRM sviluppato dalla Apple [13].

[13] In origine sviluppato per la protezione dei brani musicali su iTunes

2.2.2 Permessi, Entitlements e Sandbox

I meccanismi che regolano le azioni delle applicazioni sono:

I permessi, come per le applicazioni Android[14], devono essere specificati in un file chiamato *Info.plist*. Se un'applicazione fa uso di alcune API ritenute (da Apple) critiche dal punto di vista della sicurezza, per poterle utilizzare, lo sviluppatore deve dichiararle nel file assieme ad una motivazione[15]. Qualora venissero chiamate funzioni di una classe di API non dichiarata o non autorizzata, l'applicazione si bloccherebbe andando in crash.

Gli *Entitlements* (diritti) specificano quali risorse di sistema possono essere utilizzate in un'applicazione e sotto quali condizioni. Essi rappresentano, in pratica, le configurazioni della sandbox in termini di azioni consentite o meno.
Sono scritti in un file chiamato entitlements.xcent.

Tra le chiavi utilizzate nel file entitlements.xcent troviamo

- `get -task -allow` - Utilizzata quando un'applicazione viene compilata per la modalità di debug ed è valida solo per gli sviluppatori.
- `push -notification` - L'applicazione utilizza le notifiche push [16].
- `beta -reports -active` - L'applicazione è in beta[17] ed è disponibile su TestFlight Platform[18].
- `team -identifier` - Identificativo del team.
- `application -identifier` - Bundle ID.

[14] In Android è utilizzato il file AndroidManifest.xml per specificare i permessi
[15] Nelle recenti versioni l'assenza del motivo per l'impiego del permesso genera il crash dell'applicazione
[16] La notifica push è una tipologia di messaggistica istantanea con la quale il messaggio perviene al destinatario senza che questo debba effettuare un'operazione di scaricamento (modalità pull) - Wikipedia
[17] versione non definitiva
[18] `https://developer.apple.com/testflight/`

- `keychain -access -groups` - L'applicazione utilizza i Keychain Services (API per l'accesso al portachiavi).

Una *sandbox*, è un meccanismo che permette l'esecuzione del software in un'area ben definita e circoscritta del sistema operativo per facilitarne la separazione dagli altri processi e per agevolare il controllo delle risorse a cui un processo può accedere. Esistono diversi modi per implementare le sandbox, la Apple utilizza un sistema di diritti di esecuzione con una divisione in due classi utente: *Administrator* (app di sistema) e *User* (app scaricate dallo store).

Ogni applicazione iOS viene eseguita nella sua sandbox e non può accedere allo spazio riservato di un'altra ne leggere i dati da essa generati. Solo applicazioni appartenenti allo stesso team possono condividere lo spazio dei dati (il riconoscimento avviene attraverso il Team ID).

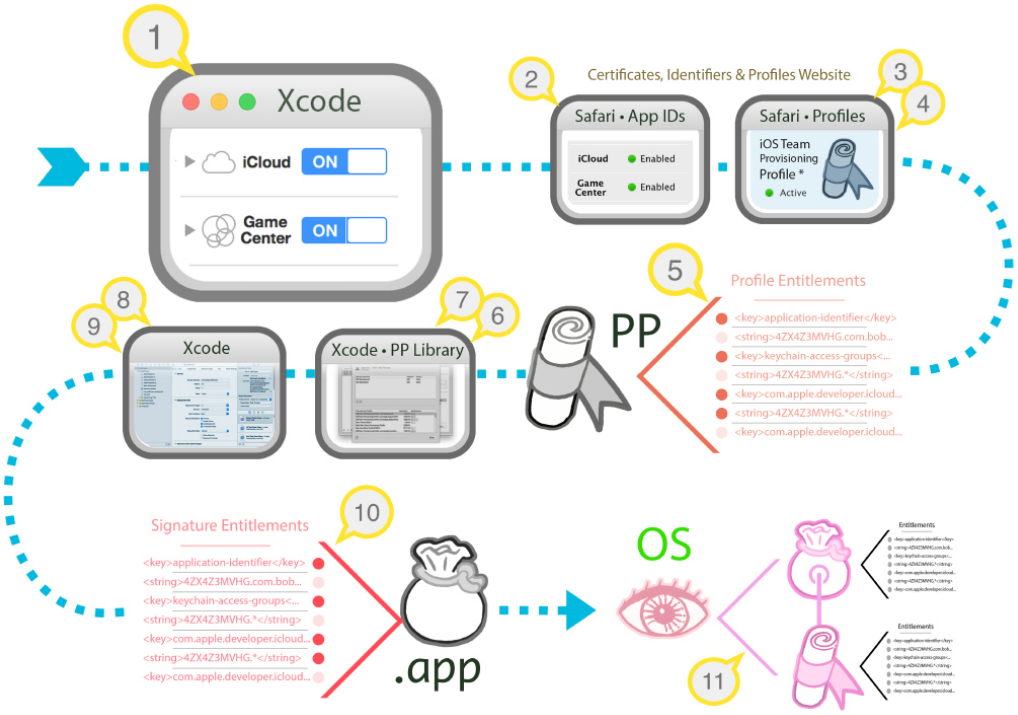

Figura 2.2: iOS Permission Process. (Technical Note 2415)

Per verificare se un'applicazione può essere eseguita su un dispositivo specifico, entrano in gioco i profili di provisioning[19], generati da Apple, che consentono di collegare un'applicazione a un determinato sviluppatore per verificare se quest'ultimo ha i permessi di utilizzo di determinate API come, ad esempio, l'accesso alla posizione e l'utilizzo di Siri [20]. L'intero processo è riportato in figura 2.2

[19]Il termine provisioning assume in informatica il significato di "assegnamento di risorse"
[20]Assistente digitale di Apple

Capitolo 3

Le Applicazioni iOS

Le applicazioni disponibili per iOS sono visualizzabili all'interno del App Store, ma possono essere scaricate solo se si dispone di un ID Apple valido[1], alcune di esse richiedono un pagamento, che può essere effettuato tramite carte, crediti o anche addebitando il costo sul conto telefonico[2].

3.1 La struttura delle IPA

IPA è un'estensione utilizzata per le applicazioni sull'App Store iTunes[3]; il file in sé è semplicemente uno ZIP e ha la struttura sotto riportata:

- `/Payload/`
  La cartella ***Payload*** è quella che contiene tutti i dati dell'app.
- `/Payload/Application.app`

[1]Per farlo è sufficiente registrarsi sul sito della Apple, la procedura è gratuita e richiede pochi passaggi

[2]Solo per gli operatori aderenti all'iniziativa

[3]Al momento Apple non riporta una definizione ufficiale dell'acronimo, probabilmente derivato inizialmente da **iP**hone **A**rchive, poi definito come **i**OS App Store **Pa**ckage ma spesso utilizzato dalle community su internet con il significato di **iP**hone **A**pplication

- `/iTunesArtwork`
  iTunesArtwork è l'icona dell'applicazione visualizzata in iTunes.

- `/iTunesArtwork@2x` La *@2x* è la stessa icona con dimensioni raddoppiate (utilizzata da dispositivi a risoluzione maggiore).

- `/iTunesMetadata.plist`
  iTunesMetadata.plist contiene vari tipi di informazioni, tra cui il nome e l'ID dello sviluppatore, l'identificatore del bundle, le informazioni sul copyright, il genere, il nome dell'app, la data di uscita, la data di acquisto, ecc.

- `/WatchKitSupport/WK`
  WatchKitSupport è utilizzato per applicazioni che si interfacciano con gli smartwatch Apple. Il framework WatchKit contiene infatti le classi per manipolare l'interfaccia di un'applicazione watchOS, la quale contiene uno o più controller di interfaccia, ognuno dei quali può avere tabelle, pulsanti, cursori e altri tipi di elementi visivi. L'estensione WatchKit utilizza le classi di questo framework per configurare gli elementi visivi e per rispondere alle interazioni utente.

Application.app è invece la cartella che contiene i file eseguibili e i dati, riportati come immagini e file .plist[4]. Assieme all'eseguibile, uno dei file più importanti è ***Info.plist*** che contiene tutte le informazioni di base dell'app, i nomi dei bundle[5] e le autorizzazioni richieste dall'applicazione.

I file *exec* sono perlopiù dei FAT Binary (maggiori informazioni su Wikipedia)[6], in pratica, contengono il codice eseguibile per ciascuna architettura

[4] I file Property List (.plist) rappresentano dati in forma di dizionario, utilizzano una codifica UTF-8 e la sintassi XML (definita da `http://www.apple.com/DTDs/PropertyList-1.0.dtd`)

[5] Il bundle è l'organizzazione di una cartella, ci sono vari tipi di bundle tra cui l'Application Bundle che rappresenta la struttura di un .app. Spesso il concetto di bundle viene ricondotto a quello di package (come in java)

[6] `https://en.wikipedia.org/wiki/Fat_binary`

per cui l'applicazione è compatibile, il che si traduce in file con dimensioni maggiori. Per lo studio dell'applicazione, ci interessa solo un'architettura (se disponibile, quella ARM64, altrimenti si analizza comunque la più recente come ad esempio ARMv7), quindi rimuoviamo le altre tramite il comando `'lipo -thin nomeEseguibile'`. Come costruire un'applicazione con un binario di tipo FAT viene riportato in appendice (a pag 57).

3.2 I Linguaggi

Come già anticipato nell'introduzione, negli ultimi 3 anni Apple ha deciso di cambiare direzione nella programmazione delle applicazioni e ha introdotto Swift. Mentre prima tutto veniva scritto in Objective-C (e tuttora le librerie principali di iOS sono scritte in quel linguaggio), ora il compilatore utilizzato su Xcode (LLVM [7]) compila per ARM sia codice in Objective-C che in Swift.

3.2.1 Objective-C

Introdotto nel sistema NeXTSTEP da Steve Jobs durante il periodo di esilio dalla Apple e poi fuso con Mac OS 9 creando Mac OS X, Objective-C è un linguaggio di programmazione orientato agli oggetti, che aggiunge la messaggistica in stile Smalltalk[8] al linguaggio di programmazione C[9]. È stato il principale linguaggio di programmazione utilizzato da Apple per i sistemi operativi OS X e iOS e le relative interfacce di programmazione delle applicazioni (API) Cocoa e Cocoa Touch prima dell'introduzione di Swift.

Objective-C è molto potente, offre moltissime feature e, come abbiamo già detto, la maggior parte delle librerie di iOS sono scritte in questo linguaggio, ma è anche molto difficile da imparare e da manutenere. La memoria,

[7] In precedenza acronimo di Low Level Virtual Machine, LLVM è un compilatore open source (`llvm.org/`)

[8] `http://smalltalk.org/`

[9] `https://en.wikipedia.org/wiki/C_(programming_language)`

infatti, è un problema costante per un programmatore Objective-C: non esiste un conteggio di riferimento automatico degli oggetti né un meccanismo automatico di pulizia (come ad esempio la garbage collection di Java), il che costringe il programmatore ad un'attenta pianificazione e gestione degli oggetti allocati.

3.2.2 Swift

Swift[10] è un linguaggio di programmazione general-purpose attualmente sviluppato da Apple e compatibile con iOS, MACOS X e Linux[11]. E' progettato per funzionare con i framework Cocoa e Cocoa Touch [12] di Apple e con il codice esistente in Objective-C (objC), è compilato con il framework open source LLVM ed è stato incluso in Xcode dalla versione 6.

Figura 3.1: Swift Logo

Ci sono stati dei problemi con la migrazione da Swift 2 a Swift 3 poiché quest'ultimo non è retro compatibile, per questo Apple, con l'annuncio di Swift 4 (presente da Xcode 9) ha mantenuto compatibilità con Swift 3.2. Da evidenziare come la transizione di generazione delle applicazioni sia ancora in corso, soprattutto per la necessità di allinearsi con un linguaggio che si è evoluto rapidamente nell'ultimo periodo. Questo si traduce, attualmente, in una presenza maggiore di applicazioni scritte in Objective-C.

Guardando invece ai pregi, Swift è sicuramente più facile da imparare, più veloce e più sicuro di Objective-C, inoltre una grande innovazione di Swift è stata l'introduzione dell'attributo *optional*. Questo consente ad esempio,

[10]Logo riportato in figura 3.1

[11]Per Linux è attualmente disponibile solo a 32-bit

[12]Cocoa and Cocoa Touch sono framework di sviluppo rispettivamente per gli ambienti OS X e iOS. Sia Cocoa che Cocoa Touch includono codice Objective-C e 2 frameworks: Cocoa, include Foundation e AppKit ed è utilizzato per sviluppare applicazioni per OS X; Cocoa Touch include Foundation e UIKit ed è utilizzato per sviluppare applicazioni per iOS

l'impiego di costruttori di oggetti che possono fallire (Failable Initializer), obbligando lo sviluppatore a pianificare alternative in fase di scrittura. Inoltre, mentre in Objective-C un metodo chiamato su un oggetto nullo viene convertito automaticamente in una nop [13], in Swift viene lanciata un'eccezione [14] che dev'essere obbligatoriamente gestita nella fase di scrittura del programma.

Swift e il meccanismo delle Witness Tables

Quando si hanno classi/struct[15] basate su protocolli di Swift, e su di loro vengono chiamate funzioni definite dai protocolli, il meccanismo della V-Table[16] non è in grado di codificare il loro comportamento in fase di esecuzione.

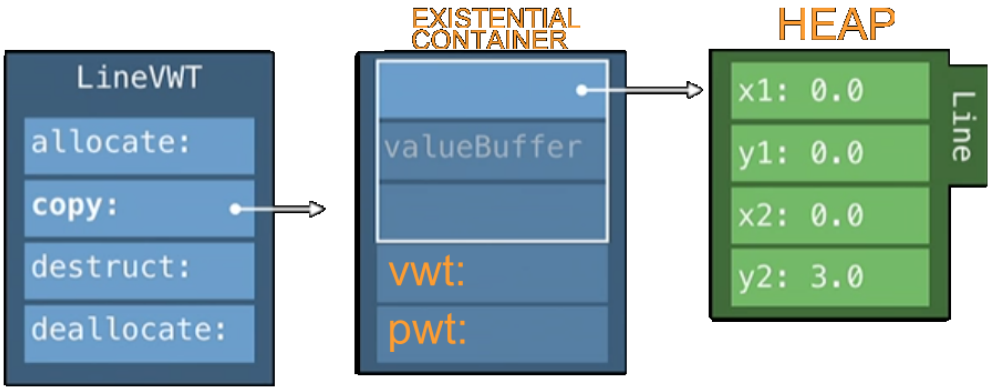

Figura 3.2: iOS Protocol Witness Table (WWDC 2015)

Si utilizza quindi il meccanismo della "Protocol Witness Table" creando, in pratica, una tabella per ogni tipo di struttura che implementa il protocollo. Il puntatore di questa tabella viene inserito in un contenitore di dimensioni fisse (5 words[17]) denominato "Existential Container". La dimensione fissa

[13] **No Op**eration, istruzione senza alcuna conseguenza

[14] In informatica, il termine eccezione è usato per descrivere l'occorrenza di diversi tipi di condizioni o eventi che alterano il normale flusso di controllo ed esecuzione di un microprocessore programmabile o di un programma. - Wikipedia

[15] Ad una conferenza WWDC2015 è stato mostrato che, in alcuni casi, l'esecuzione del codice in Swift è spesso ottimizzabile sostituendo al posto delle classi, le struct poiché esse hanno un impatto minore sulla memoria essendo caricate sullo stack anziché sulla heap [3]

[16] La Virtual method Table si occupa del re-indirizzamento dinamico dei metodi ma non è impiegata da Swift per risolvere il meccanismo del polimorfismo legato ai protocolli

[17] 1 word = 16 bit

consente di salvare elementi di diversa grandezza nello stesso spazio di memoria infatti, ci sono tre parole per la struttura, se questa occupa più spazio allora il puntatore viene salvato nella Heap (vedi figura 3.2 [3]), altrimenti direttamente nel contenitore. Negli altri due campi, troviamo un puntatore alla Value Witness Table e un puntatore alla Protocol Witness Table.

La *Value Witness Table* si occupa di allocare memoria, copiare dati, distruggere gli oggetti e de-allocare la memoria.

Questo meccanismo consente di risolvere il problema dell'inizializzazione a run time degli oggetti e la risoluzione delle funzioni, ciò si traduce nella presenza nell'eseguibile della chiamata alla Witness Table di uno specifico protocollo.

3.3 Ciclo di vita delle applicazioni per iOS

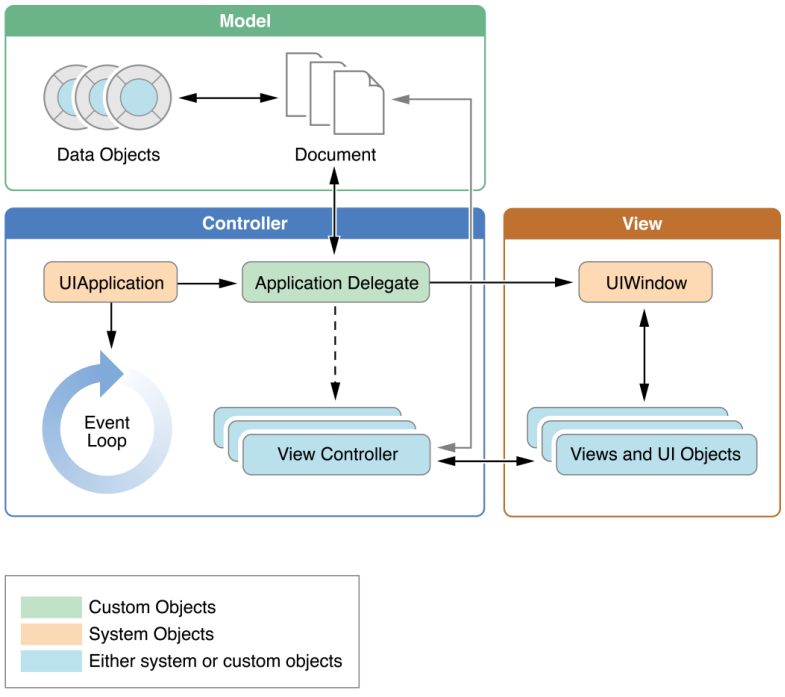

Figura 3.3: iOS MVC Lifecycle - developer.apple.com

Dietro alle applicazioni iOS c'è il pattern event-driven[18] Model View Controller (vedi figura 3.3).

L'*ApplicationDelegate* è il cuore del codice event-driven, si occupa infatti di rispondere ai vari eventi che si verificano durante l'esecuzione. Questo oggetto si relaziona con l'UIApplication e si fa carico di gestire l'inizializzazione delle applicazioni, le transizioni di stato, molti eventi di alto livello dell'app ed è anche l'unico la cui presenza è garantita in ogni applicazione.

L'applicazione dispone di stati di esecuzione (la loro gerarchia è riportata in figura 3.4) rappresentati dai seguenti metodi:

- *application:willFinishLaunchingWithOptions*: la prima opportunità dell'applicazione per eseguire il codice dal momento del lancio.
- *application:didFinishLaunchingWithOptions*: consente di eseguire qualsiasi inizializzazione finale prima che l'applicazione venga visualizzata all'utente.
- *applicationDidBecomeActive*: Consente all'app di sapere che sta per essere eseguita in primo piano.
- *applicationWillResignActive*: l'applicazione sta per uscire dalla visualizzazione in primo piano. Questo metodo mette l'applicazione in uno stato di riposo (idle).
- *applicationDidEnterBackground*: L'applicazione è in esecuzione in background e potrebbe essere terminata in qualsiasi momento.
- *applicationWillEnterForeground*: L'applicazione sta uscendo dal background e sta tornando in primo piano, ma non è ancora attiva.

[18] In informatica la programmazione a eventi è un paradigma di programmazione dell'informatica. Mentre in un programma tradizionale l'esecuzione delle istruzioni segue percorsi fissi, che si ramificano soltanto in punti ben determinati predefiniti dal programmatore, nei programmi scritti utilizzando la tecnica a eventi il flusso del programma è largamente determinato dal verificarsi di eventi esterni. - Wikipedia

- *applicationWillTerminate*: L'applicazione viene terminata. Questo metodo non viene chiamato se l'applicazione è sospesa.

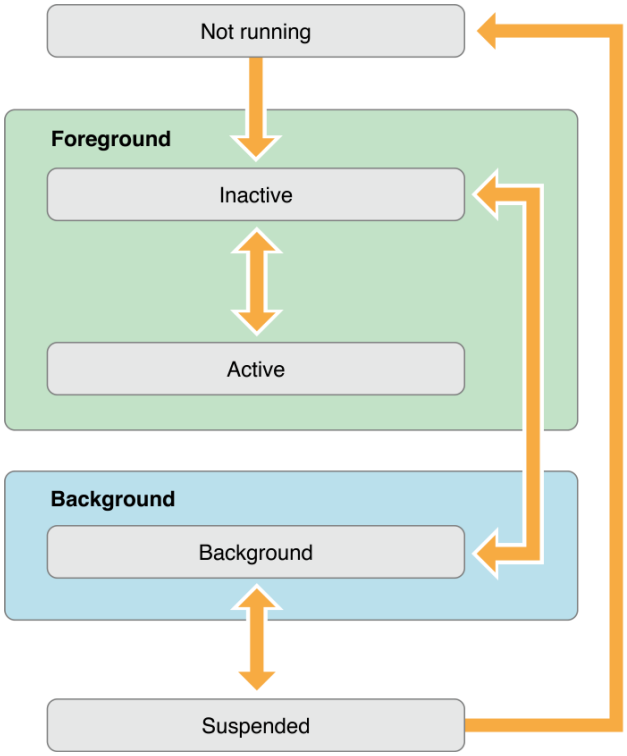

Figura 3.4: iOS App States - developer.apple.com

3.4 ARM64

> *ARM*, originariamente *Acorn RISC Machine*, successivamente *Advanced RISC Machine*, è una famiglia di processori per computer (RISC), configurata per diversi ambienti. (ARM architecture on Wikipedia)[19]

Gli *ARM64*, spesso identificati come *ARMv8-A* a 64 bit Apple A(X[20]), sono una nuova serie di processori introdotta a partire dal lancio dell'iPhone

[19] `https://en.wikipedia.org/wiki/ARM_architecture`
[20] i processori a 64 bit appartengono alle serie da A7 in su

5s. L'innovazione principale è l'architettura a 64 bit, con un nuovo set di istruzioni e una maggiore efficienza rispetto alle versioni precedenti. La tesi si è concentrata su questa nuova categoria di processori che, nel tempo, diverranno sicuramente i più diffusi. In figura 3.5 è possibile notare l'evoluzione nel tempo dei processori installati sui principali dispositivi con iOS. (Fonte: dorianroy.com/blog/)[21]

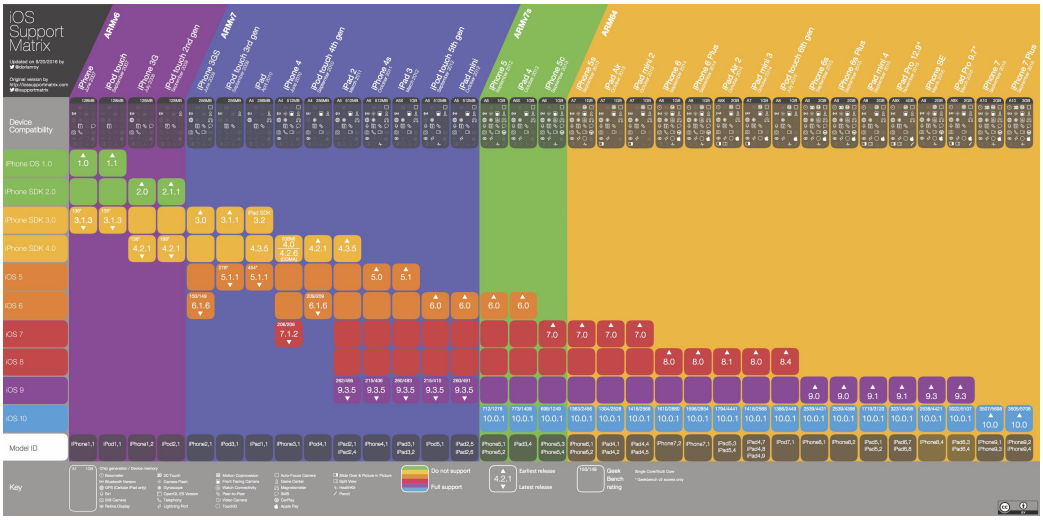

Figura 3.5: iOS Support Matrix, ARM version.

Per la fase di training sul linguaggio assembly ARM, si è pensato prima di utilizzare QEMU, un software che implementa un particolare sistema di virtualizzazione che consente di emulare un'architettura tra quelle compatibili, configurandola con un'immagine di un ARM v8. Purtroppo non è stato possibile compilare codice Swift sull'immagine creata.

La seconda scelta è stata l'uso di un Raspberry Pi3 che monta una CPU ARMv8 a 64 bit ma, purtroppo alla data corrente, non esiste una versione compilata di Swift per Linux a 64 bit, quindi il codice binario generato, non era perfettamente paragonabile a quello di Xcode.

[21] `http://dorianroy.com/blog/`

Alla fine, si è optato per lo studio del codice generato direttamente dall'iPhone 6s, anche se certamente più complesso perché ottenuto compilando l'intera applicazione mobile.

Il codice ARM possiede diverse convenzioni che vengono rispettate dal compilatore di Xcode: quelle sui registri sono riportate in tabella 3.1. Ogni istruzione in ARM è condizionata, ovvero, viene eseguita solo se la condizione sul valore del flag è rispettata; in appendice è riportata la tabella completa dei "Condition Codes.

Tabella 3.1: ARM registers convention

Registro	Special	Ruolo nella procedura
SP		The Stack Pointer
r30	LR	The Link Register
r29	FP	The Frame Pointer
r28 - r19		Callee-saved registers
r18		Platform Register (se necessario)
r17	IP1	Il secondo intra-procedure-call temporary register
r16	IP0	Il primo intra-procedure-call temporary register
r15 - r9		Temporary registers
r8		Indirect result location register
r7 - r0		Parameter/result registers

3.4.1 Il codice assembly per ARM64

Solitamente, durante la programmazione, non si scende al livello di linguaggio assembly, in questa tesi, si parte dall'ipotesi di non avere a disposizione il codice sorgente quindi, l'unico modo per studiare il comportamento di un'applicazione è quello di leggere il file eseguibile (non criptato).

Il codice ha istruzioni di lunghezza fissa, rendendo più facile la fase di decodifica, l'architettura è di tipo RISC, ha quindi meno istruzioni rispetto x86 (che è invece di tipo CISC).

Perché è importante analizzare il codice ARM? Ci sono punti nel codice in cui, ad esempio, c'è un salto ad un indirizzo e bisogna poterne tener traccia

per la creazione dell'IPCFG (che vedremo in seguito), è inoltre importante conoscere le convenzioni sui registri e le istruzioni opzionali (vedi tabella 5.1 in appendice).

Sono presenti diverse istruzioni di salto da evidenziare: *Branch with Link* (`BL`) che è una chiamata ad una procedura con ritorno, *Branch Register with Link* (`BLR`) che, invece di specificare l'indirizzo nell'istruzione, fa riferimento all'indirizzo salvato nel registro (questa istruzione non è facile da analizzare) e la *Branch to Register* (`BR`) che salta direttamente senza ritorno all'indirizzo salvato nel registro (molto importante da decodificare se non si vuole sconnettere il grafo).

3.5 Analisi con radare2

Il progetto radare è partito come editor esadecimale scriptabile in grado di recuperare file su disco e, nel tempo, sono stati inglobali altri componenti per l'analisi dei file.

Radare2 supporta diverse architetture: 6502, 8051, CRIS,
H8/300, LH5801, T8200, arc, ARM, avr, bf, blackfin, xap, dalvik, dcpu16, gameboy, i386, i4004, i8080, m68k, malbolge, mips, msil, msp430, nios II, powerpc, rar, sh, snes, sparc, tms320 (c54x c55x c55+), V810, x86-64, zimg, risc-v.

Con radare2 è possibile eseguire diversi livelli di analisi, nello strumento sviluppato noi, utilizziamo il livello più semplice ('aa'), in quanto richiede meno tempo e produce già un livello sufficiente di analisi delle applicazioni.

Questa è l'analisi lanciata:

```
Analyze all flags starting with sym and entry0 (aa)
```

Il livello di analisi più dettagliato consente di controllare meglio il codice e, a volte, ricostruire i nomi di più funzioni ma, come già detto, è più gravoso in termini di memoria e tempo.

Questa è l'elenco di analisi al livello massimo ('aaaa'):

`Analyze len bytes of instructions for references (aar)`

`Analyze function calls (aac)`

`Emulate code to find computed references (aae)`

`Analyze consecutive function (aat)`

`Type matching analysis for all functions (afta)`

Non è lo scopo della tesi di elencare tutta la lista di comandi di radare2, ma bisogna sicuramente nominare il comando `agC`. Questo infatti, stampa il Call Graph completo del programma in codice graphviz. Il grafico è di solito veramente grande (vedi una sua sezione nella figura 3.6) quindi, nel nostro tool, esploriamo la struttura leggendo il formato json del grafico (ottenibile tramite il comando `agCj`).

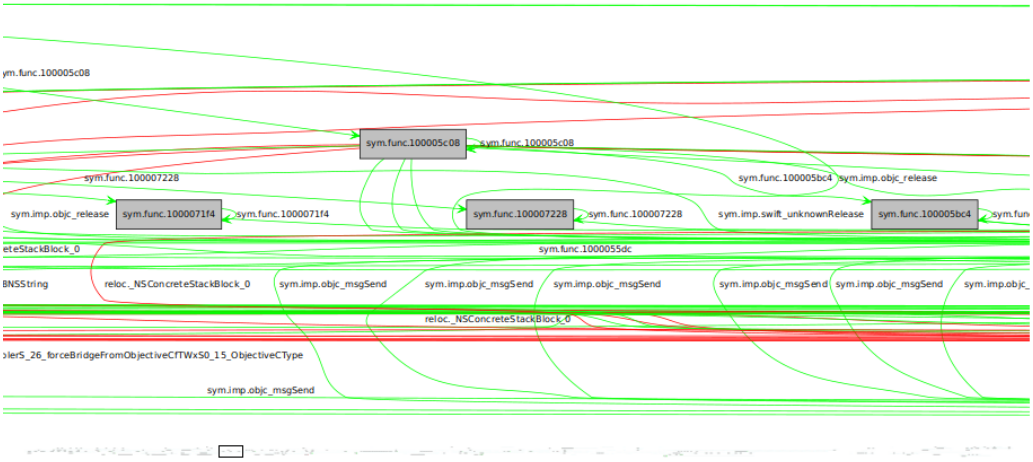

Figura 3.6: Sezione di Call Graph di una piccola app.

Capitolo 4

Modelli

4.1 Inter Procedural Control Flow Graph

Un'applicazione è composta da una serie di procedure delle quali, come visto in precedenza, possiamo modellare l'insieme delle chiamate con un Call Graph. Il Call Graph, tuttavia, non ci da informazioni sulle condizioni sotto le quali viene fatta una chiamata.

Per capire come funziona la singola funzione, dobbiamo analizzarla costruendone il *Control Flow Graph* (CFG) dividendo ogni procedura in *basic blocks* collegati da archi. Gli archi uscenti possono essere singoli, se rappresentano salti diretti (collegamento senza condizione tra i due blocchi), o doppi se rappresentano salti condizionali (dipendenti da). Possiamo vedere un esempio di CFG in figura 4.1

Dividiamo i basic blocks in due categorie:

- Senza chiamate a procedure esterne (che chiameremo nodi τ[1])
- Chiamata a una o più procedure
  - Chiamate API

[1]Seguendo la nomenclatura di mCRL2, dove τ indica una transizione senza azioni

– Chiamate a funzioni del programma

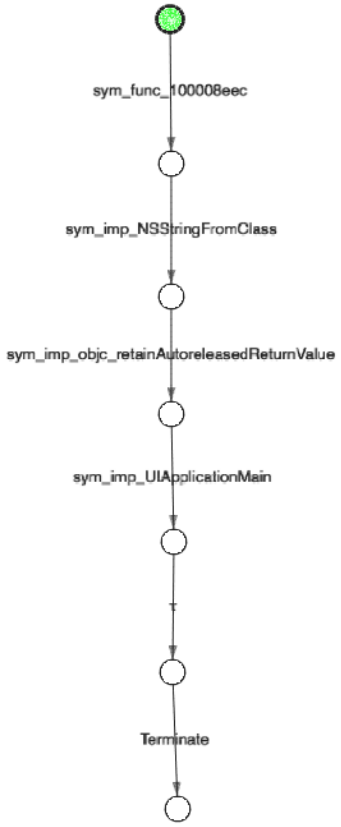

Figura 4.1: Control Flow Graph dell'Entry Point.

Applichiamo dunque le seguenti regole:
Una sequenza di nodi τ dev'essere semplificata con un singolo nodo τ.
Le chiamate API devono essere salvate con un altro nome rispetto quelle a funzioni poiché utilizzate in seguito per la verifica delle proprietà.
Le chiamate a funzione del programma connettono i singoli CFGs portando alla creazione di quello che prende il nome di *Inter-Procedural Control Flow Graph* (IPCFG).

Per dare una notazione rigorosa e differenziare il CFG dall'IPCFG possiamo sintetizzare che: il CFG rappresenta il flusso di controllo di una singola procedura, l'IPCFG rappresenta il flusso di controllo dell'intero programma.

4.1.1 Basic blocks e chiamate

Un basic block consiste essenzialmente in una sequenza di istruzioni che non vengono interrotte da salti, o da chiamate ad un'altra routine.

Etichettiamo ogni basic block con il suo indirizzo virtuale, solo il primo di ogni procedura è chiamato con il nome della funzione. Identificare un basic block in un codice assembly è facile, il vero problema è, per il disassemblatore, risolvere i nomi di tutte le funzioni chiamate. Quest'ultimo passaggio richiede solitamente molto tempo.

4.1.2 La creazione dell'IPCFG

Una volta raccolte le informazioni viste in precedenza, si può calcolare per ogni funzione presente nel Call Graph, il CFG e connetterlo quindi a quello di ogni funzione chiamata.

Si parte dal punto d'ingresso del programma (entry point) e si procede ricorsivamente per ogni funzione chiamata, agganciando i vari CFG fino a creare l'Inter-Procedural Control Flow Graph.

Questo metodo ha però un problema: la gestione dei loop. Infatti se una procedura chiama se stessa, o se viene chiamata più volte, potrebbe generare tratti con più di una freccia uscente dalla porzione di IPCFG rappresentante quella funzione. Ci ritroveremmo quindi con un'ambiguità che genera false transizioni di stato. Per gestire questa situazione, occorre implementare algoritmi efficienti per la gestione dei grafi o ricorrere ad un'altra rappresentazione.

Si è scelto quindi di utilizzare mCRL2.

4.2 Dall'IPCFG al modello mCRL2

> MCRL2 si basa sull'Algebra dei Processi di Comunicazione (ACP), estesa per includere dati e tempo. Come in ogni algebra dei processi, un concetto fondamentale in mCRL2 è il processo; questi, possono eseguire azioni ed essere composti per formare nuovi processi utilizzando operatori algebrici. Un sistema è di solito costituito da diversi processi (o componenti) in parallelo. Un processo può trasportare i dati come parametri. Lo stato di un processo è una combinazione specifica dei valori dei parametri. Questo stato può influenzare le possibili azioni che il processo può eseguire. A sua volta, l'esecuzione di un'azione può comportare un cambiamento di stato. Ogni processo ha uno spazio di stati corrispondente o un Labeled Transition System (LTS) che contiene tutti gli stati che il processo può raggiungere, insieme alle possibili transizioni tra questi stati. [4] (mcrl2.org)[2]

Di tutta la potenza espressiva di mCRL2 (la cui sintassi è discussa in [4]), noi utilizziamo solo un piccolo sottoinsieme formato dai simboli "`.`", che rappresenta i salti singoli, e "`+`" usato per i salti condizionali (else-if branches).

Un modello mCRL2 è composto da 3 parti principali:

- `act`
- `proc`
- `init`

In `act` vengono inserite le etichette delle transizioni (o archi) che, nel nostro caso, saranno o τ o API.

In `proc` invece, sono inseriti i CFG delle funzioni con la rappresentazione mediante `.` e `+` descritta sopra.

[2] `http://www.mcrl2.org`

Infine in `init` c'è l'entry point del programma.

4.3 Generazione del Modello PRISM

mCRL2 è potente ed efficiente dal punto di vista della rappresentazione dell'IPCFG, ma meno per la verifica delle proprietà. A questo scopo il modello mCRL2, dopo essere stato semplificato, evidenziando solo le API che ci interessano per la verifica delle proprietà, viene tradotto in un modello PRISM.

> PRISM è un model checker probabilistico, uno strumento per la modellazione formale e l'analisi di sistemi che presentano un comportamento casuale o probabilistico. È stato utilizzato per analizzare i sistemi di diversi domini applicativi, inclusi i protocolli di comunicazione e multimediali, gli algoritmi distribuiti randomizzati, i protocolli di sicurezza, i sistemi biologici e molti altri. (prismmodelchecker)[3]

I file .prism hanno la seguente struttura

```
ctmc

//id di ogni processo
const int id_INIT = 0;
const int id_p1 = 1;
(...)
const int id_pN = N;

// inizializzazione top stack
global tops : [0..N] init 0;
// inizializzazione top program counter
```

[3] `http://www.prismmodelchecker.org/`

```
global topc : [0..M] init 0;

global act1 : [0..1] init 0;
global act2 : [0..1] init 0;
(...)
global actM : [0..1] init 0;

// modellazione entry point
module INIT
        [] tops = id_INIT & topc = 0 -> (topc' = 1)
        (...)
endmodule

module p1
        [] tops = id_p1 & topc = 0 -> (topc' = 1)
    (...)
endmodule

(...)

module pN
        [] tops = id_pN & topc = 0 -> (topc' = 1)
    (...)
endmodule
```

4.4 Test delle Proprietà

Le proprietà che vengono testate sono rappresentate da formule PCTL, la cui sintassi viene discussa nel dettaglio in [1]. Nel nostro caso generalmente hanno la seguente forma:

P>0 [PROP_CTL]

Le politiche di sicurezza sono inspirate dalle OWASP Mobile Top 10 2016 [5], che descrive i 10 maggiori rischi della sicurezza delle applicazioni mobili [4]. Attualmente sono:

- Uso improprio della piattaforma
- Memorizzazione non sicura dei dati
- Comunicazione non sicura
- Autenticazione non sicura
- Crittografia insufficiente
- Autorizzazioni non sicure
- Bassa qualità del codice del client
- Manomissione del codice
- Reverse Engineering
- Funzionalità estranee

Per la nostra applicazione vengono tradotte in test di questo tipo

```
P>0 [G((api_func_1 = 0) & (api_func_2 = 0))]
```

Queste proprietà possono sono scritte in un file `.pctl` e sono quindi utilizzate per testare il modello PRISM tramite il comando

```
prism -nopre -cuddmaxmem 4g -javamaxmem 4g
-ex prism_model_file.prim properties_file.pctl
```

[4] aggiornate al 13 Febbraio 2017

Vedere le note a piè di pagina per la spiegazione del comando (fonte `http://www.prismmodelchecker.org/manual/`) [5] [6] [7] [8]

I modelli ottenuti possono essere molto grandi (possono tranquillamente superare le 500 000 righe) e l'analisi può richiedere molto tempo; poichè richiedono fino a 4GB, nel computer utilizzato potevano essere analizzate le proprietà di quattro modelli in parallelo.

[5]Per impostazione predefinita, gli algoritmi di controllo probabilistico dei modelli PRISM utilizzano un passo di pre-computazione iniziale che utilizza tecniche basate su grafici per individuare efficacemente casi banali dove le probabilità sono 0 o 1. Ciò può spesso portare a prestazioni migliori e anche ridurre gli errori di round-off. Occasionalmente, tuttavia, è possibile disattivare questo passaggio per l'efficienza (ad esempio se si è certi che non esistano pochi stati o che il processo di pre-computazione sia lento). Ciò può essere fatto con inserendo -nopre. È anche possibile disattivare i singoli algoritmi per la probabilità 0/1 utilizzando gli switch -noprob0 e -noprob1.

[6]CUDD, la libreria BDD e MTBDD sottostante utilizzata in PRISM ha un limite di memoria superiore. Per impostazione predefinita, questo limite è di 1 GB. Per impostare il limite, dalla riga di comando basta utilizzare -cuddmaxmem.

[7]La macchina virtuale Java (JVM) utilizzata per eseguire PRISM ha anch'essa limiti di memoria superiori. A volte questo limite verrà superato e vedrete un errore della forma java.lang.OutOfMemory. Per risolvere questo problema, è possibile aumentare il limite di memoria. Su piattaforme Unix, Linux o Mac OS X, questo può essere fatto utilizzando -javamaxmem, passandolo a PRISM da riga di comando o tramite GUI xprism.

[8]PRISM contiene quattro motori principali che implementano la maggior parte delle funzionalità di controllo del modello: "MTBDD" (-mtbdd), "sparse" -sparse, "ibrido" -hybrid e "explicit" -explicit (o -m, -s, -h e -ex, rispettivamente))

Capitolo 5

iOS Exec Reverse Engineering Tool

Lo scopo di questa tesi è la creazione di uno strumento automatico che, data un'applicazione iOS, analizzi l'eseguibile ARM in essa contenuto e crei il corrispettivo modello (descritto in precedenza) per scoprire se vi sono violazioni delle politiche di sicurezza.
Lo strumento, implementato in Python 3, è stato realizzato con un'architettura modulare (vedi figura 5.1).

5.1 L'architettura

5.1.1 Input Validator

Il tool accetta in ingresso applicazioni `.ipa` o anche `exec` binari. Controlla se l'eseguibile è di tipo FAT e, nel caso, ne estrae solo l'architettura ARM64 (o, se non disponibile, la più recente). Quindi controlla se l'eseguibile è criptato e, in caso di risposta negativa, estrae le informazioni di base (dimensioni, architettura, sviluppatore, ...), quindi salva il risultato in un file.

Figura 5.1: Architettura iOS Exec Reverse Engineering Tool.

5.1.2 API Functions Web Extractor

Modulo indipendente ma necessario al funzionamento del tool. Non essendo disponibile un database delle API, lo creiamo noi scaricando direttamente le informazioni dalla documentazione sul sito della Apple[1]. Il database viene salvato in un file SQLite che, per ogni tipologia di API, identifica le classi che la compongono; successivamente, per ogni classe, ne estrae i metodi. E'composto quindi: da una tabella che lega le categorie con le classi, una seconda tabella che collega le classi con le funzioni, infine una tabella che descrive le funzioni specificando tipo di ritorno, nome, parametri ed un campo conte-

[1] `https://developer.apple.com/documentation/`

nente il "*secCriticalFlag*". Il secCriticalFlag è un flag di default inizializzato a *false*, una volta terminata la raccolta dei dati per il popolamento del database, viene lanciata una seconda analisi sulla documentazione per ricavare dati aggiornati sulle API dichiarate "security critical". Fatto ciò, si aggiorna il database che verrà utilizzato dagli altri moduli. Questo modulo non è in cascata con gli altri, sia poiché il popolamento del database richiede tempo, sia perché non è necessario ripetere questa operazione per ogni analisi.

5.1.3 Exec Function Analyser

Questo modulo si occupa di ottenere la lista delle funzioni presente nell'eseguibile, e il loro indirizzo virtuale. Crea dunque due dizionari: uno che mappa i nomi delle funzioni con gli indirizzi, e l'altro che, invece, mappa gli indirizzi con i nomi delle funzioni. In questo modo, sono rapidamente traducibili i salti a indirizzo e le etichette dei basic blocks. Questa è la parte che richiede più tempo perché il disassemblatore (nel nostro caso radare2) impiega molto tempo nell'analisi dell'eseguibile. Un'indicazione sulle tempistiche prestazionali è descritta nel capitolo successivo.

5.1.4 IPCFG Extractor

Come già visto nel capitolo precedente, generiamo l'Inter-Procedural Control Flow Graph analizzando i CFG di ogni funzione. Per la divisione in basic blocks, analizziamo il codice ARM, classificando ogni blocco come τ solo se non ha chiamate API a funzioni. Per ogni blocco quindi conosciamo indirizzo e archi uscenti. Questo servirà per la prossima fase.

5.1.5 mCRL2 Converter

Passiamo ora alla traduzione in mCRL2 seguendo la sintassi del linguaggio precedentemente descritta. Vengono generati 2 file; il primo rappresentante l'IPCFG non sfoltito, il secondo generato a partire dal primo, ma eliminando

le chiamate API, non *security critical*, diminuendo così la dimensione del modello. Questa fase di "pruning" del grafo può essere modificata specificando quali funzioni mantenere assieme a quelle critiche.

5.1.6 PRISM Converter

Il modello mCRL2 semplificato è tradotto in un modello PRISM. Questo contiene una maggior ridondanza rispetto al precedente, ma è anche più efficiente dal punto di vista della verifica delle proprietà.

5.1.7 Property Checker

Le proprietà sono scritte in PCTL. Questa parte richiede 4 GB di memoria RAM libera, poiché si appoggia ad una Java Virtual Machine e, in presenza di modelli molto grandi, può richiedere molto tempo.

5.1.8 Manifest Reader

Questo modulo analizza il file `Info.plist` di un .ipa ed estrae i permessi richiesti. Ciò potrà essere, in futuro, collegato al modulo Exec Function Analyzer. Infatti, si potrebbero elencare tutti i permessi; richiesti e utilizzati, richiesti e non utilizzati, e anche i permessi utilizzati ma non richiesti, nonostante quest'ultimo punto probabilmente non esista in applicazione per i dispositivi non jailbroken (ma potrebbe anche identificare un punto di crash nell'applicazione).

5.1.9 IPA Classifier

Questa è più un utility, poiché permette di smistare in cartelle differenti (una per ogni tipo di architettura e linguaggio) la cartella contenente IPA data come input. Ciò è stato utile per sottoporre ad analisi le applicazioni scritte in Swift.

5.2 Risultati

Sono state analizzate diverse applicazioni scaricate da uno store non ufficiale (`iphonecake.com`). Non è stato possibile prelevarle dall'Apple Store originale poiché per l'analisi occorrono, al momento, IPA decriptate.

Delle applicazioni scaricate, servendoci del classificatore automatico del nostro tool, abbiamo selezionato quelle scritte in Swift.

E' stato preso in considerazione un campione di 224 applicazioni recenti. L'architettura e la parte predominante del codice di tali applicazioni è distribuita come segue:

- 2 app per ARMv7 (16-bit) scritte in C++
- 2 app per ARMv7 (16-bit) scritte in Objective-C
- 48 app per ARMv7 (32-bit) scritte in C
- 62 app per ARMv7 (32-bit) scritte in C++
- 41 app per ARMv7 (32-bit) scritte in Objective-C
- 43 app per ARMv7 (32-bit) scritte in Swift
- 4 app per ARMv8 (64-bit) scritte in C
- 2 app per ARMv8 (64-bit) scritte in C++
- 2 app per ARMv8 (64-bit) scritte in Swift
- 18 FAT (sia 32 che 64-bit) o

Dal grafico in figura 5.2 risulta evidente la predominanza delle applicazioni "non Swift".

Il grafo IPCFG (di cui riportiamo un esempio in fig. 5.3) è costruibile indipendentemente dal linguaggio utilizzato ma per la fase di "pruning" è necessario, al momento, che le chiamate API siano invocate con Swift. Questo

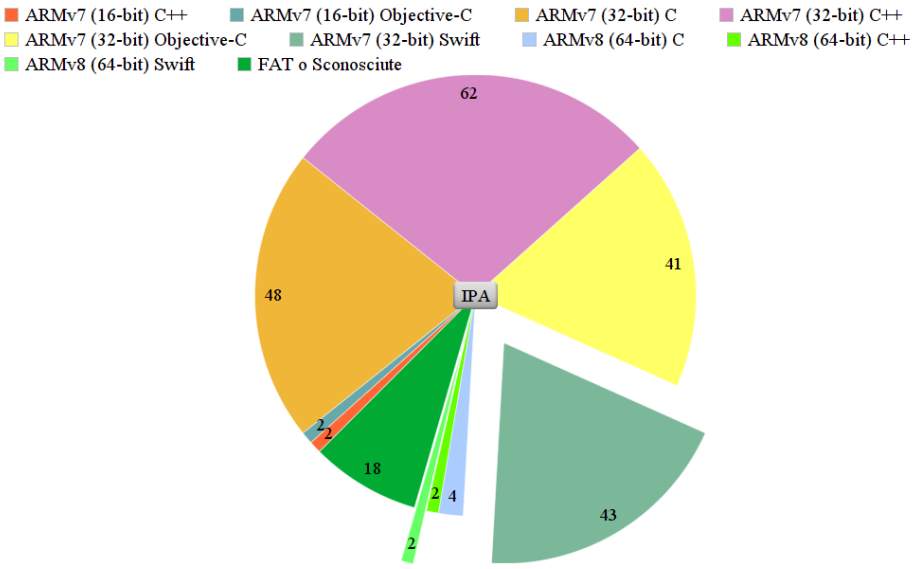

Figura 5.2: I linguaggi delle IPA.

poiché il modulo che salva le API nel database preleva solo le informazioni relative alle funzioni in Swift.

I grafi generati hanno le seguenti proprietà:

- Sono orientati con un vertice di partenza ("entry point")[2] e un vertice di terminazione ("terminate") [3]
- Ad esclusione dei vertici vicini[4] a quelli di partenza e di terminazione, il restanti tendono a formare cicli (Tipico delle applicazioni iOS gestite con gli Event Loop, vedi figura 3.3)

[2]Rappresenta il punto di partenza del programma, nel modello mCRL2 è il primo processo in init

[3]Rappresenta il punto di terminazione dell'applicazione, può essere raggiunto per chiusura normale o per crash dell'applicazione

[4]Per vicini si intendono vertici collegati da percorsi non più lunghi di 7

- Sono connessi, poiché la parte di codice non raggiungibile viene scartata dal modello.

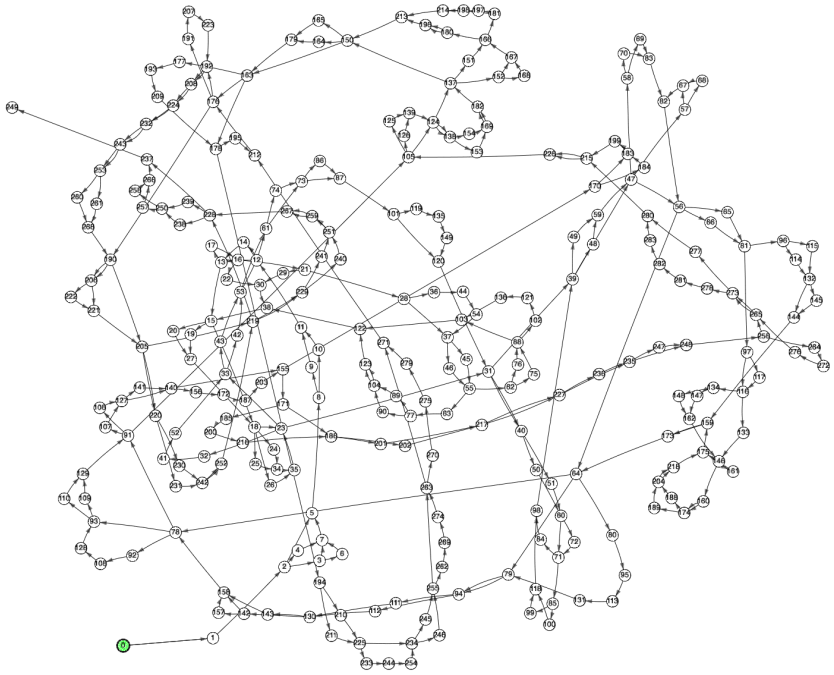

Figura 5.3: IPCFG di una piccola applicazione.

Le proprietà testate sono nella forma

```
P>0 [G(api_func_1 = 0)]
```

su API scelte "manualmente" tra l'elenco di quelle segnalate security critical dal nostro tool ottenendo, a volte, risultato negativo. Questo fatto è attribuibile a 2 motivi: la chiamata è dichiarata in una funzione mai invocata, oppure non si è riusciti a raggiungere la funzione con l'analisi.

Bisogna sottolineare che, seppur l'approccio sia quello corretto, non è facile trattare la complessità delle applicazioni con il disassemblatore attuale. C'è quindi la necessità di migliorare le tecniche per riuscire a coprire tutte le possibili chiamate.

5.3 Problema della scalabilità

Nel grafico in figura 5.4 sono mostrati, per un campione di 12 applicazioni (di dimensione media di 20 MB), i tempi di esecuzione (in secondi) dei vari moduli del tool. Tempi più lunghi coincidono con dimensioni maggiori dell'eseguibile e, pertanto, implicano una maggiore complessità del modello generato.

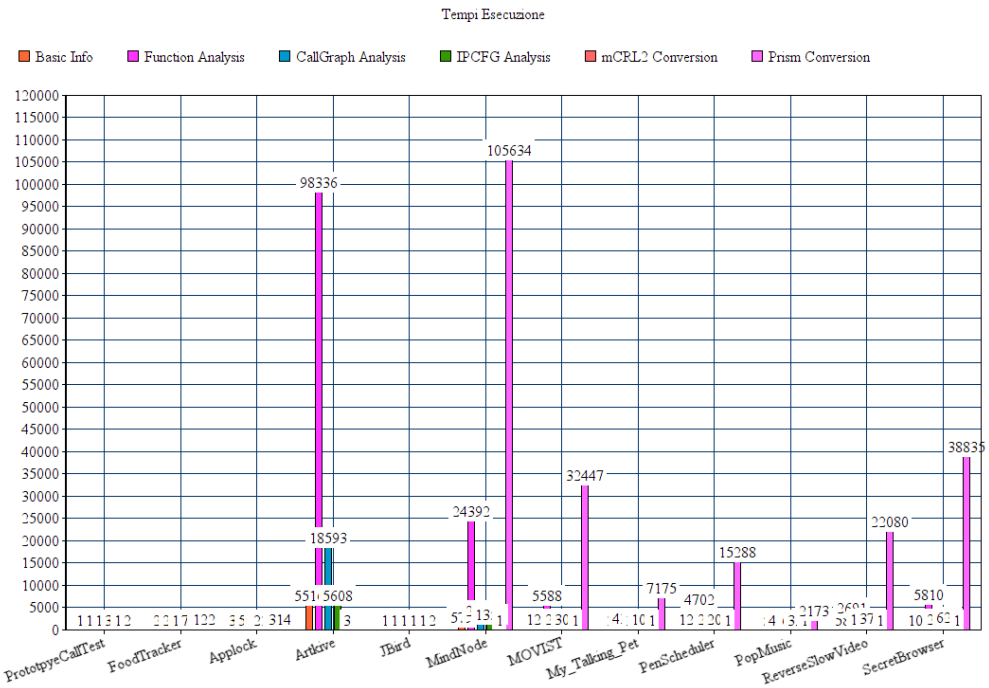

Figura 5.4: Grafico dei tempi di analisi.

Vi sono analisi che superano la giornata e mezza per eseguire la generazione del modello, impattando sulla scalabilità della proposta. Possibili soluzioni sono:

- migliorare le tecniche di analisi sul file binario (per ora dipendenti dall'approccio utilizzato da radare)

- riuscire ad eseguire in parallelo l'analisi frammentando l'area di eseguibile da analizzare

5.4 Il problema dell'esecuzione Multi-thread

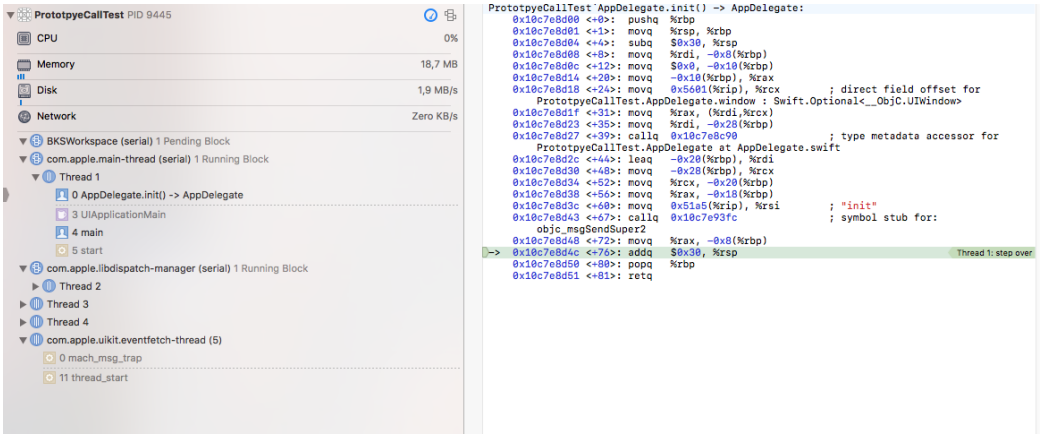

Figura 5.5: Esecuzione dei thread in fase d'avvio.

L'invocazione dell'UIApplicationMain causa il lancio di 6 thread diversi (vedi figura 5.5) che eseguono in parallelo la fase di inizializzazione dell'applicazione. L'invocazione di questa procedura non è facile da seguire nel codice binario, ed ha causato inizialmente una frammentazione del grafico. Per superare questo ostacolo, si è deciso di collegare direttamente il metodo applicationDidFinishLaunching all'UIApplicationMain, dove l'applicazione inizia effettivamente ad eseguire il suo codice (come mostrato nel flusso di lancio di figura 5.6).

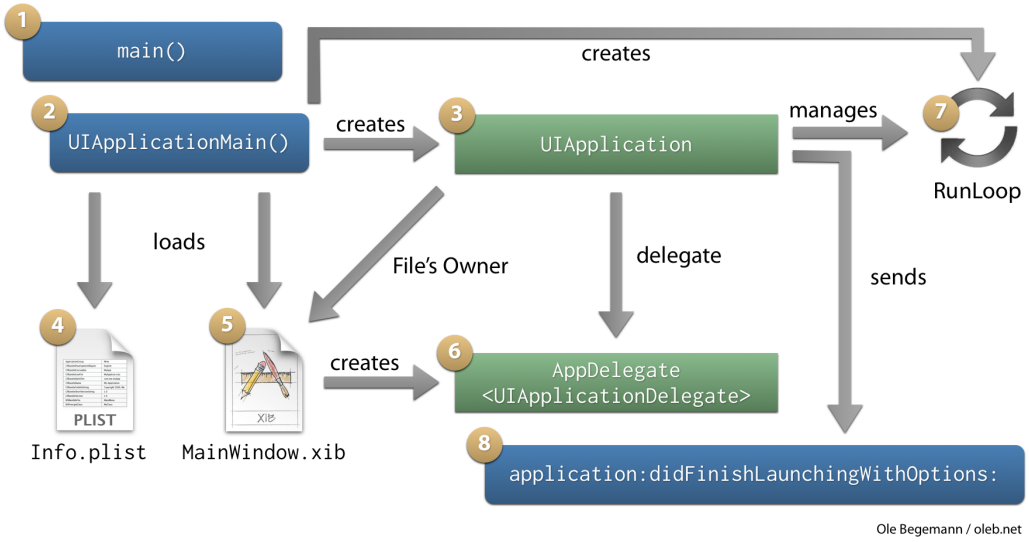

Figura 5.6: iOS App Launch Flow.

Consideriamo anche i possibili percorsi dovuti all'improvvisa chiusura dell'applicazione, alla transizione in background e alla ripresa dell'esecuzione, che possono avvenire in qualunque momento, chiamando ogni volta i rispettivi metodi. Questo porta ad un'esplosione dello spazio degli stati nel modello che purtroppo deve essere tracciata per non perdere informazioni essenziali.

Un'altra complicazione deriva dalla natura Event-Driven dell'interfaccia e quindi, dalla necessità di riconoscere sequenze di interazione con pulsanti, gesti e altre azioni che innescano una chiamata a una funzione. Anche la loro sequenza non può essere determinata a priori, e non è attualmente possibile ricostruire una gerarchia di interfacce; questo provoca anche un aumento esplosione di percorsi nel modello.

Related Works

Ci sono diversi strumenti che hanno guidato ed ispirato la creazione del nostro tool, alcuni di questi non sono più manutenuti, altri sono compatibili solo con le vecchie versioni, pur rimanendo utili per analizzare la precedente generazione di applicazioni iOS.

Class-dump (sviluppato da Steve Nygard [6]) è una utility da riga di comando per esaminare le informazioni (di runtime Objective-C) memorizzate nei file Mach-O (file binari in MACOS X e iOS). Recupera le dichiarazioni delle classi, delle categorie e dei protocolli. Queste sono le stesse informazioni fornite usando 'otool -ov', ma presentate nella forma di dichiarazioni in codice Objective-C, quindi molto più compatte e leggibili. Purtroppo, non è più aggiornato e non è compatibile con Swift.
Rimane comunque un ottimo strumento per analizzare le applicazioni scritte in Objective-C, anche se destinate a diminuire nel tempo.

MobSF (Mobile Security Framework - disponibile su GitHub [7]) consente l'analisi statica delle applicazioni iOS attraverso la lettura degli header, recupera informazioni sulle librerie utilizzate e sui flag di compilazione. Ha uno stile grafico molto user-friendly, ma non consente ne la verifica formale ne un controllo delle politiche di sicurezza, poiché non effettua l'ispezione del codice.
Questo strumento rimane, al momento, uno dei pochi progetti open source per l'analisi delle applicazioni di iOS.

Se si dispone di un iPhone jailbroken allora la prima applicazione da

sottoporre ad analisi è sicuramente DVIA [8]. Sviluppata con l'intento di evidenziare eventuali vulnerabilità del codice, piuttosto che un'applicazione, Damn Vulnerable iOS Application è uno strumento per iniziare lo studio per eseguire la reverse engineering delle applicazioni Apple per iOS. Purtroppo, questo rimane negli obiettivi futuri poiché richiede il jailbreak.

> Damn Vulnerable iOS App (DVIA) è un'applicazione iOS che è altamente vulnerabile. Il suo obiettivo principale è quello di fornire una piattaforma per gli appassionati di sicurezza mobile / professionisti o studenti per testare le loro capacità di test di penetrazione di iOS in un ambiente legale. Questa applicazione copre tutte le vulnerabilità comuni riscontrate nelle applicazioni iOS (segue le OWASP top 10 mobile risks[5]) e contiene diverse sfide in cui l'utente può provare a cimentarsi.

Retargetable Decompiler [9] è un interessante progetto avviato da AVG (poi proseguito da Avast dopo l'acquisizione del primo da parte di quest'ultimo) che consente l'analisi di piccoli eseguibili e che permette di ricostruire la struttura del codice sorgente in C o Python. Purtroppo ha gli stessi problemi di scalabilità incontrati nella tesi (con tempi anche maggiori dei nostri) e quindi esaurisce le risorse nel tentativo di analizzare l'applicazione.

Ultimo, ma non per importanza, è PiOS[6][10]. Progetto pubblicato da 4 ricercatori nel 2013. Questa tesi condivide lo stesso approccio all'analisi di applicazioni iOS che si trova nel paper, utilizza strumenti e metodologie differenti per analizzare le applicazioni scritte in Objective-C, ma segue lo stesso filone logico. Purtroppo il codice di PiOS non è disponibile e tutto il lavoro di questa tesi è stato, pertanto, svolto ex-novo.

[5] `https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10`
[6] `http://www.syssec-project.eu/m/page-media/3/egele-ndss11.pdf`

Conclusione e Sviluppi Futuri

La tesi ha proposto un nuovo approccio per l'analisi statica delle applicazioni iOS, basato sulle tecnologie allo stato dell'arte, focalizzandosi su Swift, che diventerà lo standard di sviluppo delle future applicazioni per iOS. Allo stato attuale, la tecnologia permette di condurre un'analisi statica su un file IPA che descriva il comportamento dell'applicazione in termini di API invocate (filtrando via quelle non desiderate). Non potendo ricavare la gerarchia dell'interfaccia grafica, resta comunque aperto il problema della semplificazione della sequenza delle chiamate, dovuto alla natura event-driven delle applicazioni, che possono essere invocate da elementi grafici.

Tra i possibili ampliamenti e sviluppi futuri vi è, inoltre, l'attuale necessità di migliorare il database delle API, espandendolo alle funzioni utilizzate in Objective-C. Questo consentirebbe la compatibilità di analisi con applicazioni meno recenti e permetterebbe di semplificare lo studio delle proprietà del modello.

Inoltre, occorre effettuare un'analisi più approfondita delle componenti multithread delle applicazioni.

Al fine di ridurre i tempi, è necessario ottimizzare l'analisi del codice assembly. Si può procedere modificando il codice del disassemblatore attualmente impiegato, o sviluppando uno strumento da zero includendo la funzione, se possibile, di eseguire l'analisi sulla singola applicazione in parallelo, scomponendo il codice in più sezioni.

Un altro possibile sviluppo è l'utilizzo del jailbreak per estendere l'ana-

lisi delle applicazioni anche in fase di esecuzione, recuperare le informazioni sull'interfaccia, eseguire l'iniezione di codice per recuperare informazioni critiche. Ma soprattutto, si potrebbe utilizzare Clutch [7] al fine di analizzare le applicazioni scaricate direttamente da iTunes.

Una volta implementati gli sviluppi precedenti, si potrà generare un numero maggiore di analisi dei modelli, producendo così campioni statistici rilevanti sulla sicurezza delle applicazioni iOS.

[7]Strumento per decriptare le applicazioni iOS

Appendice

Creare un'applicazione con un binario FAT

```
#!/bin/sh

UNIVERSAL_OUTPUTFOLDER=${BUILD_DIR}/${CONFIGURATION}-universal

# make sure the output directory exists
mkdir -p "${UNIVERSAL_OUTPUTFOLDER}/iOS"

# Step 1. Build Device and Simulator versions on iOS
xcodebuild -workspace "${PROJECT_NAME}.xcworkspace" -scheme "${PROJECT_NAME}"
-sdk iphonesimulator -destination 'platform=iOS Simulator,name=iPhone 6' clean build
xcodebuild -workspace "${PROJECT_NAME}.xcworkspace" -scheme "${PROJECT_NAME}"
-sdk iphoneos clean build

# Step 2. Copy the framework structure (from iphoneos build) to the universal folder
cp -R "${BUILD_DIR}/${CONFIGURATION}-iphoneos/${PROJECT_NAME}.framework"
"${UNIVERSAL_OUTPUTFOLDER}/iOS"

# Step 3. Create universal binary file using lipo and place the combined executable
# in the copied framework directory
lipo -create -output
"${UNIVERSAL_OUTPUTFOLDER}/iOS/${PROJECT_NAME}.framework/${PROJECT_NAME}"
"${BUILD_DIR}/${CONFIGURATION}
-iphonesimulator/${PROJECT_NAME}.framework/${PROJECT_NAME}"
"${BUILD_DIR}/${CONFIGURATION}-iphoneos/${PROJECT_NAME}.framework/${PROJECT_NAME}"

# Step 4. Convenience step to copy the framework to the project's directory
mkdir -p "${TMPDIR}/${PROJECT_NAME}/Frameworks/iOS"

cp -R "${UNIVERSAL_OUTPUTFOLDER}/iOS/${PROJECT_NAME}.framework"
"${TMPDIR}/${PROJECT_NAME}/Frameworks/iOS"

# Step 6. Create .tar.gz file for posting on the binary repository
cd "${TMPDIR}"

# We nest the framework inside a Frameworks folder so that it unarchives correctly
tar -zcf "${PROJECT_NAME}.framework.tar.gz" "${PROJECT_NAME}/Frameworks/"
mv "${PROJECT_NAME}.framework.tar.gz" "${PROJECT_DIR}/"

# Step 7. Convenience step to open the project's directory in Finder
#open "${PROJECT_DIR}"
```

(source `https://stackoverflow.com/questions/39890114/`)

ARM Condition Codes

Tabella 5.1: ARM Condition Codes

Encoding	Code	Meaning	Condition flags
0000	EQ	**EQ**ual	Z = 1
0001	NE	**N**ot **E**qual	Z = 0
0010	CS	**C**arry **S**et	C = 1
0010	HS	equal or **H**igher than (CS alias)	C = 1
0011	CC	**C**arry **C**lear	C = 0
0011	LO	**LO**wer than (CC alias)	C = 0
0100	MI	**MI**nus, negative	N = 1
0101	PL	*P*ositive or *L*ow (zero)	N = 0
0110	VS	o**V**erflow (**S**igned)	V = 1
0111	VC	o**V**erflow (unsigned)	V = 0
0111	HI	**HI**gher than	C = 1 AND Z = 0
1000	LS	**L**es**S**er than or equal to	C = 0 OR Z = 1
1001	GE	**G**reater than or **E**qual to	N == V
1010	LT	**L**ess **T**han	N != V
1011	GT	**G**reater **T**han	Z == 0 AND N == V
1101	LE	**L**ess than or **E**qual to	Z ==1 OR N != V
1110	AL	**AL**ways executed	any
1111	NV	always executed (*deprecated*)	any

(source `http://infocenter.arm.com`)

Bibliografia

[1] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[2] ios security, ios 10. *iOS Security—White Paper*, 2017.

[3] Apple Worldwide Developers Conference 2015. Optimizing swift performance. `https://developer.apple.com/videos/play/wwdc2015/409/`. Accessed 21 agosto 2017.

[4] Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[5] OWASP. Owasp top 10 mobile risks. `https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10`. Accessed 21 agosto 2017.

[6] Class dump. `http://stevenygard.com/projects/class-dump/`. Accessed 21 agosto 2017.

[7] Mobile security framework. `https://github.com/MobSF/Mobile-Security-Framework-MobSF`. Accessed 21 agosto 2017.

[8] HighAltitudeHacks.com. Damn vulnerable ios application. `http://damnvulnerableiosapp.com/`. Accessed 21 agosto 2017.

[9] AVG and AVAST. Retargetable decompiler. `https://retdec.com/`. Accessed 21 agosto 2017.

[10] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Pios: Detecting privacy leaks in ios applications. In *NDSS*, pages 177–183, 2011.