

UNIVERSITÀ DEGLI STUDI DI GENOVA  
Facoltà di Ingegneria



*Corso di Laurea Magistrale in Ingegneria Informatica*

---

APP-IOTTE: UN FRAMEWORK PER  
INTEGRARE L'ANALISI DI SICUREZZA  
DELLA APP MOBILE E IoT BASATE SU  
ANDROID

*Relatore:*

**Prof. Alessio Merlo**

*Candidato:*

**Fabio Parodi**  
Matricola n. 3936429

*Correlatore:*

**Dott. Luca Verderame**

# Indice

---

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Struttura delle tesi . . . . .	3
<b>2</b>	<b>I problemi dell'IoT</b>	<b>4</b>
<b>3</b>	<b>Android</b>	<b>9</b>
3.1	Sicurezza . . . . .	11
3.1.1	Sandbox . . . . .	11
3.1.2	Permessi . . . . .	12
3.1.3	Autenticazione dell'utente . . . . .	13
3.1.4	Controllo della firma dell'applicazione . . . . .	14
3.1.5	Chiavi crittografiche . . . . .	14
3.1.6	Verified Boot . . . . .	15
<b>4</b>	<b>Android Things</b>	<b>17</b>
4.1	Introduzione . . . . .	17
4.2	Devices . . . . .	17
4.3	Gestione dei dispositivi . . . . .	19
4.4	Architettura . . . . .	20
4.5	Sicurezza . . . . .	25
4.5.1	Sandbox delle applicazioni . . . . .	25
4.5.2	Privilegi . . . . .	25
4.5.3	Cifratura e Keystore . . . . .	26
4.5.4	Verified Boot Hash . . . . .	26
4.5.5	Anti Rollback Attack . . . . .	26
4.5.6	Certificazione . . . . .	27
4.5.7	Attestazione remota . . . . .	28
4.5.8	Nessuna comunicazione con il sistema operativo . . . . .	29
4.6	Aggiornamenti . . . . .	30
4.7	Funzionamento . . . . .	34
4.8	Sicurezza dei due sistemi . . . . .	36

<b>5</b>	<b>Analisi di sicurezza</b>	<b>39</b>
5.1	Introduzione . . . . .	39
5.2	Analisi statica . . . . .	39
5.3	Analisi dinamica . . . . .	41
5.4	Mobile Security Analysis . . . . .	43
5.4.1	OWASP Mobile Security Project . . . . .	44
5.4.2	Approver . . . . .	47
5.5	IoT App Security Analysis . . . . .	48
<b>6</b>	<b>Metodologia</b>	<b>51</b>
6.1	Introduzione . . . . .	51
6.2	Analisi dell'Ecosistema Applicativo Mobile-IoT . . . . .	52
6.2.1	Funzionamento generale . . . . .	53
6.2.2	Algoritmo di correlazione "Naive" . . . . .	56
6.2.3	Algoritmo di correlazione "Repetitions" . . . . .	64
6.2.4	Differenze tra gli algoritmi . . . . .	71
6.2.5	Algoritmo combinations . . . . .	72
<b>7</b>	<b>AppIoTTE: un framework per l'analisi integrata</b>	<b>76</b>
7.1	AppIoTTE . . . . .	77
7.2	Analisi statica . . . . .	78
7.3	Analisi dinamica . . . . .	80
7.3.1	Dispositivi per il test . . . . .	80
7.3.2	Frida . . . . .	80
7.3.3	Analisi network . . . . .	82
7.3.4	Moduli implementati . . . . .	84
7.4	Architettura . . . . .	86
7.5	Reasoner . . . . .	91
7.5.1	Schema . . . . .	91
7.5.2	Elaborazione informazioni . . . . .	92
7.5.3	Filtraggio dati e correlazione . . . . .	93
7.5.4	Algoritmo di correlazione utilizzato . . . . .	93
<b>8</b>	<b>Casi d'uso</b>	<b>97</b>
8.1	Preparazione all'avvio . . . . .	97

8.1.1	Analisi Mobile . . . . .	97
8.1.2	Analisi dell'ecosistema mobile-IoT . . . . .	99
8.2	Test . . . . .	100
<b>9</b>	<b>Conclusioni e Sviluppi futuri</b>	<b>105</b>

# Acronimi

---

**A.T.** Android Things

**ADB** Android Debug Bridge

**API** Application Programming Interface

**ART** Android Runtime

**BSP** Board Support Package

**CA** certificate authority

**HAL** Hardware Abstraction Layer

**IPC** Inter Process Communication

**MAC** Controlli di Accesso Obbligatori

**OTA** Over-the-air

**SID** Security Identifier

**SO** Sistema Operativo

**SoM** System-on-Module

**TEE** Trusted Execution Environment

**UID** User Identifier

# Elenco delle figure

---

1	Top 10 OWASP IoT Vulnerabilities . . . . .	5
2	Architettura Android . . . . .	9
3	Sandbox delle applicazioni . . . . .	12
4	AndroidManifest.xml (in giallo le stringhe riportanti i permessi) . . . . .	13
5	Schema concettuale delle operazioni di cifratura . . . . .	15
6	Single-board computer “ <i>RaspberryPi 3 Model B</i> ” . . . . .	17
7	Schema di comunicazione <i>Google Weave</i> . A sinistra sono rappresentati i devices IoT, a destra i servizi Google. . . . .	19
8	Architettura Android . . . . .	20
9	Architettura Android Things a confronto con Android . . . . .	21
10	Architettura per dispositivi Android Things . . . . .	23
11	Catena di certificati di un device Android Things . . . . .	27
12	Significati dietro alle operazioni di verifica dei certificati . . . . .	28
13	Informazioni sulla versione A.T. . . . .	30
14	Schema degli aggiornamenti Android Things . . . . .	31
15	Diagramma di flusso della gestione degli aggiornamenti del dispositivo .	32
16	Partizione di memoria AB . . . . .	33
17	Rappresentazione dell’ecosistema telefono-device-web . . . . .	34
18	Top 10 OWASP IoT vulnerabilities, parti prese a carico da Android Things. . . . .	38
19	Obiettivi delle analisi combinate . . . . .	43
20	Rappresentazione delle analisi eseguite nell’OWASP framework . . . . .	46
21	Schema Approver . . . . .	47
22	Analisi OWASP eseguite su Android ed Android Things . . . . .	49
23	Ecosistema di dispositivi IoT . . . . .	50
24	Unione delle analisi dei singoli dispositivi con l’ecosistema . . . . .	52
25	Scenario 1: algoritmo naive senza disturbi . . . . .	59
26	Scenario 2: algoritmo naive con disturbi . . . . .	61
27	Scenario 3: algoritmo naive con disturbi e ritardi . . . . .	63
28	Analisi dell’ecosistema, con ripetizione del segnale B (destra) . . . . .	65
29	Analisi di uno scenario eterogeneo. In grigio, i segnali B analizzati . . . . .	69
30	Analisi dei vari segnali B, con le relative anomalie sottostanti . . . . .	70

31	Comportamento dei vari algoritmi di correlazione nei possibili scenari (parte 1) . . . . .	74
32	Comportamento dei vari algoritmi di correlazione nei possibili scenari (parte 2) . . . . .	75
33	Panoramica dei moduli AppIoTTE per l'analisi mobile (verde) ed IoT (rosso) . . . . .	77
34	Esempio di una cattura di chiamata API con Frida . . . . .	81
35	Funzionamento generale di un proxy . . . . .	83
36	Composizione del modulo “Analisi Interna” . . . . .	86
37	Composizione del macromodulo “Analisi Dinamica” . . . . .	88
38	Architettura dell'analisi congiunta tra mobile e dispositivo IoT . . . . .	89
39	Schema interno del reasoner AppIoTTE . . . . .	91
40	Risultati dell'analisi di un caso pessimo . . . . .	95
41	Comando ./setup_environment.sh a console . . . . .	97
42	Comando TALOS-PROFILE=dev bash start_environment.sh a console	98
43	Avvio delle analisi IoT tramite AppIoTTE . . . . .	99
44	Applicazione di controllo della centralina IoT . . . . .	100
45	Analisi dell'ecosistema smartHome . . . . .	102
46	Risultati correlazione DB . . . . .	103
47	Risultati correlazione comunicazioni non protette . . . . .	104

# 1 Introduzione

---

*Internet of Things* (IoT) -in italiano “*Internet degli oggetti*” – è un’espressione utilizzata da qualche anno per definire la rete di apparecchiature e dispositivi automatici connessi ad internet. Termine molto generale, che vuole porre l’attenzione sul fatto che ogni giorno si vedono connettere ad Internet un numero sempre maggiore di nuovi devices, e di nuovi utenti “umani”.

Capire l’importanza dell’IoT risulta molto semplice: oggetti normalmente incapaci di comunicare possono, tramite la rete, trasferire informazioni agli utenti o ad altri dispositivi, diventando intelligenti almeno in senso lato. Sono quindi solamente due i requisiti per far rientrare un oggetto tra i “Things”: avere un indirizzo IP che ne consenta l’identificazione univoca in rete e la capacità di scambiare dati attraverso la rete stessa, senza bisogno dell’intervento umano.

Il concetto di “rete di devices intelligenti” era già stato discusso agli inizi del 1982, ma il termine “IoT” fu introdotto nel 1999 da Kevin Ashton, in visione di una crescente automatizzazione di ogni genere di processo. Nel 2003 il rapporto devices/utenti connessi in rete si attestava intorno al 0.08, nel 2010 questo rapporto era già 1.84.

Mentre prima questi devices erano per lo più inerenti ad un contesto industriale, dove un operatore poteva supervisionare a schermo il funzionamento del device, oggi l’evoluzione nel campo dei processori e della microelettronica ha permesso lo sviluppo e la diffusione di smartdevices a prezzi contenuti, anche nella vita comune. I cellulari, nati inizialmente come strumenti di chiamata, si sono trasformati in computer portatili con i quali accedere facilmente alla rete, ovunque ci si trovi. Tutto questo ha portato ai giorni nostri, al boom del settore IoT.

Purtroppo il passaggio dalla produzione manifatturiera alla produzione di massa, risulta la maggior parte delle volte problematico, portando a galla errori di progettazione, ma soprattutto, fallo di sicurezza con protocolli insicuri, hardware facilmente manomissibili, sistemi operativi vulnerabili senza monitoraggio o rilascio di updates e “badpractice” [1]. “Numerosi dirigenti d’azienda riconoscono i potenziali vantaggi che l’IoT può offrire, ma sono profondamente preoccupati per i rischi. La maggioranza ammette di non comprendere appieno le minacce alla sicurezza informatica che l’IoT porta con sè”. [2]

Nel migliore dei casi il device risulta essere vulnerabile a furto di dati, mentre nel peggiore, risulta possibile modificarne completamente il funzionamento.

Nel 2016 “Mirai”<sup>1</sup>-un malwarebotnet specializzato nell’individuare, infettare ed alterare in maniera automatica il funzionamento di devices IoT allo scopo di effettuare attacchi DDoS su specifici bersagli- è stato rilasciato in rete con conseguenze disastrose. Le indagini successive hanno individuato circa 50.000 indirizzi IP in tutto il mondo, indirizzi di devices precedentemente infettati e resi parte della botnet.<sup>2</sup>

Nel maggio del 2018, dopo tre anni dal suo annuncio, Google ha rilasciato “Android-Things”, un sistema operativo basato sull’omonima versione, dedicato completamente al mondo dell’Internet degli Oggetti. Lo scopo è chiaro: proporsi come punto di riferimento in quest’area di mercato, prendendosi carico di parte degli oneri di sicurezza e proponendo agli sviluppatori IoT un’alternativa, mediante un linguaggio di programmazione già consolidato.

L’obiettivo della tesi è stato quello di studiare dal punto di vista della sicurezza un ecosistema complesso come quello mobile-IoT, focalizzandosi sulla sicurezza a livello applicativo. Questo ha portato allo studio di una metodologia basata su tecniche di analisi statica e dinamica, che permettesse l’assestamento del livello di sicurezza non solo sulle singole applicazioni, ma dell’intera infrastruttura di dispositivi mobile-IoT. Si è quindi progettato un framework, chiamato AppIoTTE, che implementi tale metodologia per le tecnologie Android ed Android Things, in grado di verificare in maniera automatica la presenza nell’ecosistema di superfici d’attacco, individuandone le cause scatenanti. La necessità di sviluppare questo strumento nasce proprio dall’introduzione di devices IoT nel mondo comune, in modo da fornire un utile strumento di analisi automatica agli sviluppatori software, la cui area di competenza potrebbe non includere la sicurezza; a causa della complessità dell’argomento, il bisogno di un simile strumento risulta elevato, in quanto attualmente non sono disponibili strumenti simili, in grado di analizzare l’intero ecosistema IoT. Viene infine applicato il framework d’analisi AppIoTTE su uno use case realistico, in modo da mostrare l’applicabilità di tale metodologia.

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Mirai\\_\(malware\)](https://en.wikipedia.org/wiki/Mirai_(malware))

<sup>2</sup> <https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html>

## 1.1 Struttura delle tesi

La tesi è organizzata nel seguente modo: viene inizialmente fatta una piccola introduzione sulle problemi di sicurezza che affliggono l'IoT, per poi analizzare Android Things, il nuovo sistema Operativo Google, con le sue differenze rispetto ad Android. Dopodichè vengono illustrate le tecniche di analisi di sicurezza e la metodologia sviluppata per lo studio di ecosistemi IoT, con la sua implementazione. Infine viene esposto il funzionamento del prototipo realizzato attraverso alcuni casi d'uso, e le considerazioni finali su eventuali sviluppi futuri.

La sezione 1 offre una panoramica degli argomenti trattati in questo lavoro di tesi, esponendone gli obiettivi finali.

In sezione 2 vengono elencate le vulnerabilità più gravi che affliggono il mondo dell'IoT.

In sezione 3 viene presentato sinteticamente il sistema operativo Android, fornendo una panoramica sulla sua architettura ed i principali meccanismi di sicurezza.

In sezione 4 viene presentato il sistema operativo per l'IoT Android Things, con la sua architettura e le differenze che lo contraddistinguono da Android.

In sezione 5 vengono mostrate le principali tecniche di analisi di sicurezza adottate dalla comunità scientifica. Viene spiegata la distinzione tra analisi statica e dinamica, mostrandone le applicazioni per il mondo mobile e per il mondo IoT.

In sezione 6 viene esposta nel dettaglio la metodologia adottata in questo lavoro di tesi; viene introdotto il framework AppIoTTE per l'analisi di ecosistemi IoT, mostrandone il funzionamento ed elencando gli algoritmi di correlazione utilizzati.

La sezione 7 descrive l'implementazione di AppIoTTE, discutendo le scelte implementative e gli strumenti utilizzati; vengono quindi esposti i moduli di analisi realizzati, ed il funzionamento del reasoner interno

La sezione 8 riporta il funzionamento del framework realizzato, con alcuni casi d'uso legati all'analisi di un ecosistema IoT ed un riepilogo dei risultati ottenuti.

Infine in sezione 9 vengono fatte alcune considerazioni sui risultati ottenuti durante lo svolgimento della tesi, esponendo miglioramenti applicabili e spunti su lavori futuri per continuare questo lavoro di tesi.

## 2 I problemi dell'IoT

---

Più dispositivi IoT si fanno strada nel mondo, distribuiti in ambienti incontrollati, complessi e spesso ostili, più aumentano le sfide da affrontare. Secondo il sondaggio sugli sviluppatori IoT 2017 di *Eclipse IoT Working Group*[3], la sicurezza rimane la principale preoccupazione in questo campo.

Si richiede, oltre ad un'alta preparazione su ciò che riguarda la sicurezza nel ciclo di vita del dispositivo, risorse per tenerlo aggiornato e fronteggiare così i nuovi problemi. Con eccezione delle grandi aziende, che possono permettersi uno sviluppo a 360 gradi continuato del prodotto, in generale si riscontrano gravi problemi lato software o hardware, dovuto al riadattamento di diverse tecnologie in un unico prodotto.

L'Open Web Application Security Project (OWASP), una comunità online di esperti nel settore dell'industria tecnologica coinvolta nel campo della sicurezza delle applicazioni web, ha stilato la lista delle 10 maggiori vulnerabilità che coinvolgono ad oggi il mondo dell'IoT [4], visibile in Fig. 1 in pagina seguente.

<b>1</b>	<b>Weak, Guessable, or Hardcoded Passwords</b> Use of easily bruteforced, publicly available, or unchangeable credentials, including backdoors in firmware or client software that grants unauthorized access to deployed systems.	
<b>2</b>	<b>Insecure Network Services</b> Unneeded or insecure network services running on the device itself, especially those exposed to the internet, that compromise the confidentiality, integrity/authenticity, or availability of information or allow unauthorized remote control...	
<b>3</b>	<b>Insecure Ecosystem Interfaces</b> Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering.	
<b>4</b>	<b>Lack of Secure Update Mechanism</b> Lack of ability to securely update the device. This includes lack of firmware validation on device, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates.	
<b>5</b>	<b>Use of Insecure or Outdated Components</b> Use of deprecated or insecure software components/libraries that could allow the device to be compromised. This includes insecure customization of operating system platforms, and the use of third-party software or hardware components from a compromised supply chain.	
<b>6</b>	<b>Insufficient Privacy Protection</b> User's personal information stored on the device or in the ecosystem that is used insecurely, improperly, or without permission.	
<b>7</b>	<b>Insecure Data Transfer and Storage</b> Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing.	
<b>8</b>	<b>Lack of Device Management</b> Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities.	
<b>9</b>	<b>Insecure Default Settings</b> Devices or systems shipped with insecure default settings or lack the ability to make the system more secure by restricting operators from modifying configurations.	
<b>10</b>	<b>Lack of Physical Hardening</b> Lack of physical hardening measures, allowing potential attackers to gain sensitive information that can help in a future remote attack or take local control of the device.	

Figura 1: Top 10 OWASP IoT Vulnerabilities

### **1) Password deboli, indovinabili o presenti all'interno del codice**

Le password costituiscono la barriera contro i tentativi d'accesso non autorizzati, proteggendo generalmente un ottenimento di privilegi da parte di un utente. L'utilizzo di password deboli o facilmente individuabili è rischioso, in quanto la potenza di calcolo di un processore è tale da poter effettuare milioni di tentativi ogni secondo; una password di soli 5 caratteri risulta essere individuabile in meno di un giorno <sup>3</sup>.

Nella maggior parte dei sistemi analizzati, dopo il deploy del sistema, vengono mante- nute password ed impostazioni di fabbrica; tutte informazioni facilmente ottenibili con un minimo di ricerca online [5]

La scalata dei privilegi costituisce un enorme vulnerabilità in qualsiasi sistema, ottenendo un livello di accesso non autorizzato sempre più profondo, fino alla sua forma più grave con l'ottenimento dei privilegi di amministratore, avendo così completo controllo del dispositivo.

### **2) Insecure Network Services**

Le trasmissioni Over-the-air (OTA) mettono in comunicazione devices separati anche da un'enorme distanza, ma aprono a problemi di confidenzialità, integrità e disponibilità del messaggio. L'uso di protocolli di comunicazione non sicuri, infatti, rischia di mettere a repentaglio il trasferimento di informazioni sensibili per la privacy dell'utente o il corretto funzionamento del dispositivo.

Tramite attacchi di tipo Man-in-the-Middle (che costituiscono il 64% degli attacchi sferrati contro dispositivi IoT[6] ) è possibile intercettare, ispezionare e alterare le comunicazioni di rete che avvengono tra dispositivi se non messe correttamente in sicurezza.

### **3) Insecure Ecosystem Interface**

In informatica, l'utente riesce a comunicare con il mondo online grazie alle interfacce fornite dalle applicazioni. Nella vita di tutti i giorni si fa spesso uso di web app, ma con l'aumento di potenza dei dispositivi mobile, oggi gli smartphone rappresentano l'interfaccia primaria per la gestione del mondo IoT; ma l'interfaccia utente rappresenta anche il principale vettore di attacco per ottenere l'accesso al dispositivo IoT.

Nonostante la sicurezza delle interfacce utente faccia parte di un ramo già consolidato, bisogna sempre tenere a mente che il device IoT non è un'estensione del sistema,

---

<sup>3</sup> <https://www.betterbuys.com/estimating-password-cracking-times/>

ma un “oggetto” indipendente che ha bisogno di altrettante attenzioni. Vi è quindi bisogno di un doppio livello di controllo, sia front-end che back-end, per assicurarsi della sicurezza dell’intera infrastruttura.

#### ***4) Lack of Secure Update Mechanism***

I continui progressi dell’informatica, rendono necessario tenere aggiornato il dispositivo, per fronteggiare situazioni non previste in fase di sviluppo.

Modalità d’aggiornamento sicure comprendono: modalità di consegna dell’update indipendenti dall’intervento dell’utente, autenticazione e verifica della firma dell’aggiornamento, meccanismi che evitino di riportare il device ad uno stato antecedente, considerato non sicuro e vulnerabile.

#### ***5) Use of Insecure or Outdated Components***

Non tutte le componenti software possono essere aggiornate; molto spesso i produttori cessano il supporto ai loro prodotti per molteplici ragioni, commerciali e/o economiche. Con il tempo possono essere scoperte nuove vulnerabilità o, semplicemente, rilasciati nuovi componenti più sicuri ed efficienti.

Questo concetto risulta applicabile sia a componenti software, che hardware, rendendo necessaria una attenzione particolare a quanto presente nel dispositivo, assicurandosi della validità di ogni componente.

#### ***6) Insufficient Privacy Protection***

Il problema della privacy è un problema che affligge tutti i device, non solo IoT. Molte applicazioni non prestano le necessarie cure al trattamento dei dati personali, conservandoli o addirittura ottenendoli in maniera forzata, contro le volontà dell’utente. L’analisi dei dati personali grezzi permette, al contempo, di mantenere un’alta precisione, ma il trasferimento di queste informazioni a terze parti, che siano aziende od attaccanti, risulta essere in contrasto con i diritti di privacy dell’utente stesso.

Risulta quindi necessario rinunciare ad operare con grande precisione su questi dati, effettuando invece su questi operazioni di anonimizzazione.

#### ***7) Insufficient Data Transfer and Storage***

Così come risulta necessario cifrare i dati durante il loro trasferimento tra dispositivi, così risulta necessario cifrarli al momento della loro conservazione. A nulla serve impe-

gnarsi a nasconderli, se un attaccante li può facilmente leggere dal dispositivo stesso. Occorre quindi applicare tecniche di cifratura ai database, o sui dati stessi, conservando le chiavi in un'area di memoria non accessibile.

#### ***8) Lack of device management***

Oltre all'implementazione delle tecniche corrette, è necessario continuare con le operazioni di analisi di sicurezza durante tutto il ciclo di vita del dispositivo. Alcuni esempi sono: conoscerne tutte le sue possibili modalità d'uso, analizzare adeguatamente le sue interazioni ed avere un adeguato piano di decommissionamento, al fine di rimuoverlo correttamente dal mercato.

#### ***9) Insecure default settings***

Le impostazioni di default, accettabili per facilitare lo sviluppo del dispositivo, risultano insufficienti durante l'uso attivo da parte di un utente, perché aperte a potenziali falle nella sicurezza. È necessario che l'utente abbia la possibilità di modificarne le impostazioni, secondo le proprie esigenze.

#### ***10) Lack of physical hardening***

I device sono oggetti a cui è possibile connettersi attraverso porte o sonde, risultando potenzialmente soggetti a studi di reverse engineering dell'hardware con l'obiettivo di ottenere indizi sul suo funzionamento interno. Tale possibilità costituisce una grave carenza nella sicurezza del device, da evitare attraverso un maggiore irrobustimento delle difese "fisiche" del dispositivo [7].

Date le elevate somiglianze, prima di studiare Android Things è necessario comprendere il normale funzionamento di un sistema operativo Android.

## 3 Android

Android<sup>4</sup> è un sistema operativo open source, inizialmente sviluppato per telefoni cellulari e tablet da *Google Inc.* e dall'*Open Handset Alliance* (OHA)<sup>5</sup>. Successivamente sono state proposte varie versioni per smartwatch, occhiali e televisori, riadattando principalmente l'interfaccia utente secondo le necessità/contesto.

Nel campo della telefonia mobile, è uno dei sistemi operativi più usati, con una percentuale di diffusione attualmente sul 74%<sup>6</sup>. La differenziazione hardware, il continuo sviluppo ed il supporto della community, sono le ragioni che hanno reso Android popolare nel mondo.

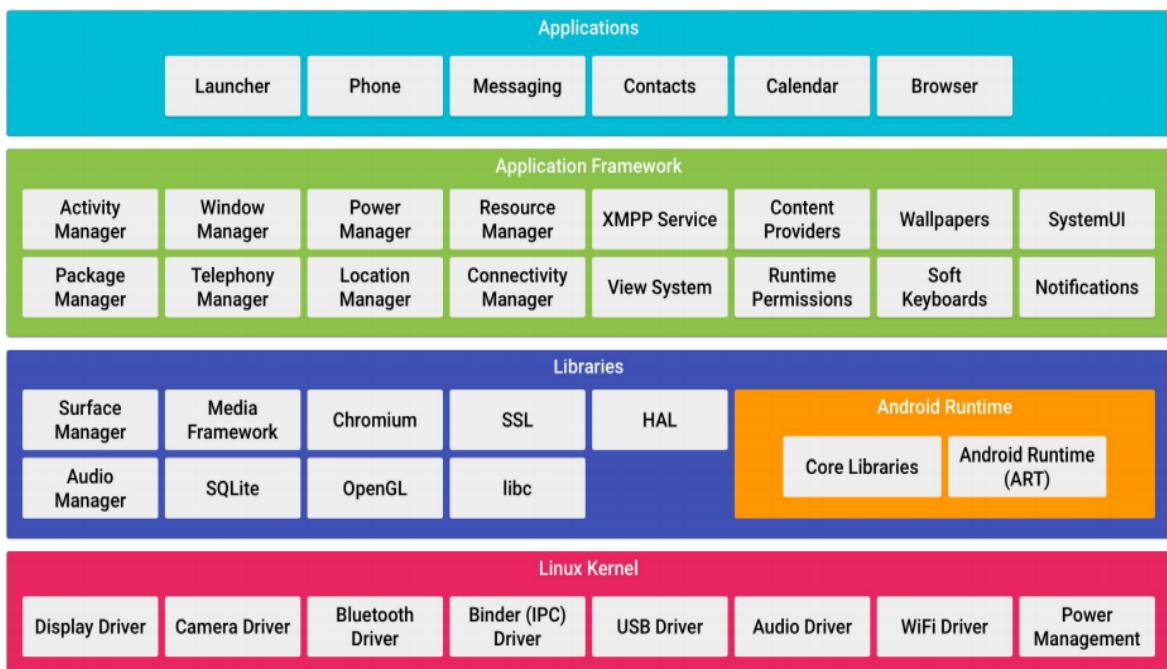


Figura 2: Architettura Android

Le fondamenta di questo sistema operativo sono costituite dal kernel Linux, sfruttandone i principali meccanismi di sicurezza: un modello multiutente basato sui permessi,

<sup>4</sup> <https://developer.android.com/guide/platform/>

<sup>5</sup> <https://www.openhandsetalliance.com/>

<sup>6</sup> <https://gs.statcounter.com/os-market-share/mobile/worldwide>

l’isolamento e la gestione dei processi e dei meccanismi di sicurezza di Inter Process Communication (IPC) estendibili. Caratteristica fondamentale è quella di proteggere i dati dell’utente e le risorse del dispositivo, evitando che le applicazioni accedano alle risorse di loro competenza.

Nel livello successivo, l’Hardware Abstraction Layer (HAL) ha il compito di fornire interfacce standard per esporre le funzionalità hardware del dispositivo ad un livello superiore, attraverso una serie di Application Programming Interface (API) scritte in linguaggio Java [Java API framework]

Ogni app viene eseguita in una macchina virtuale a lei dedicata, in modo da separare completamente applicazioni differenti. Dalla versione 5.0 di Android, la Dalvik virtual machine è stata sostituita con l’Android Runtime (ART), una versione modificata della Java Virtual Machine ottimizzata per richiedere la minor quantità di memoria possibile; a tale scopo il codice dell’applicazione viene convertito in formato “dex bytecode”, un formato speciale per minimizzare le risorse richieste, ideato per Android.

Il livello delle librerie native è costituito da una serie di librerie in C/C++, necessarie a molte dei servizi e delle applicazioni di sistema, come ART o HAL.

Il livello successivo riguarda l’Application Framework, dove tutte le funzionalità del SO sono disponibili attraverso API scritte in linguaggio Java.

L’ultimo livello è quello applicativo: Android viene rilasciato con un set di applicazioni di sistema già disponibili per l’utente, come e-mailing, SMS, Chiamate, Browser ed Internet, ma l’utente è libero di installarne di aggiuntive attraverso una rapida ricerca nell’Android Play Store.

### 3.1 Sicurezza

Android mette a disposizione una serie di meccanismi di sicurezza:

- La Sandbox <sup>7</sup>
- I permessi <sup>8</sup>
- Autenticazione dell'utente <sup>9</sup>
- Controllo della firma delle applicazioni <sup>10</sup>
- Chiavi crittografiche <sup>11</sup>
- Verified Boot <sup>12</sup>

#### 3.1.1 Sandbox

Essendo fondato su un kernel Linux, Android trae vantaggio dai suoi principali meccanismi di sicurezza per affrontare il problema dell'uso improprio delle risorse. In un sistema Linux quando viene avviato un nuovo processo, a questo viene assegnato un User Identifier (UID) rappresentante l'identità dell'utente; tale processo potrà quindi accedere ad una determinata risorsa, solamente se rientrante nel set di regole definite dal relativo proprietario;

Lo stesso meccanismo è presente in Android; al momento dell'installazione di un'applicazione, il sistema operativo le assegna un UID atto ad identificarne l'utente responsabile. Utente ed applicazione, anche senza esserne a conoscenza, vengono così legati tra loro. Spetterà quindi all'applicazione decidere chi, e con quali modalità, potrà accedere ad una propria risorsa, salvando i dati privati all'interno della directory della applicazione stessa, rendendoli inaccessibili da applicazioni terze.

Le applicazioni sono quindi eseguite in una sandbox, sia attraverso la loro esecuzione all'interno di un'apposita macchina virtuale, che attraverso la separazione dei relativi files.

---

<sup>7</sup> <https://source.android.com/security/app-sandbox>

<sup>8</sup> <https://developer.android.com/guide/topics/permissions/overview>

<sup>9</sup> <https://source.android.com/security/authentication>

<sup>10</sup> <https://source.android.com/security/apksigning>

<sup>11</sup> <https://source.android.com/security/keystore>

<sup>12</sup> <https://source.android.com/security/verifiedboot>

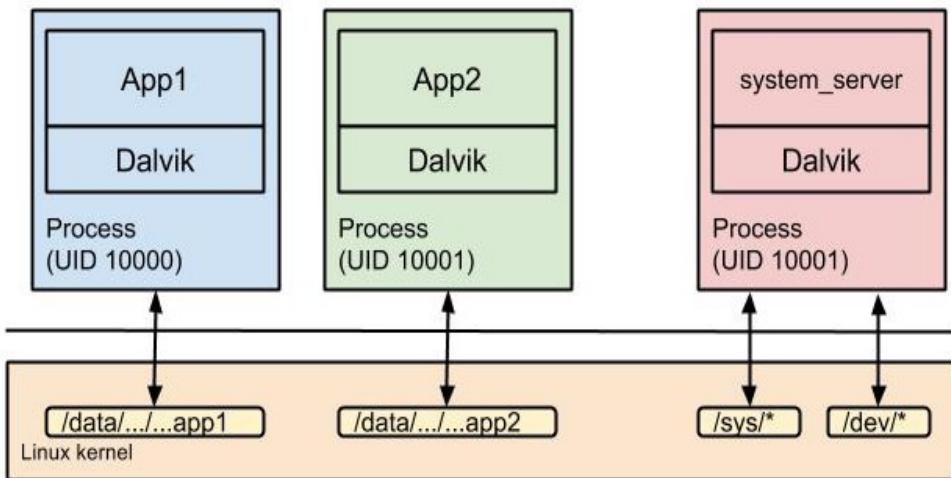


Figura 3: Sandbox delle applicazioni

### 3.1.2 Permessi

Uno dei punti di forza dell’architettura di sicurezza Android è che nessuna applicazione, di default, ha la possibilità di eseguire operazioni che potrebbero impattare negativamente su altre app, il sistema operativo, o l’utente. Dato che difficilmente un’applicazione avrà a disposizione internamente tutte le risorse necessarie, per conseguire i propri obiettivi dovranno essere utilizzate anche risorse esterne, come canali per comunicare con altre applicazioni, ottenere l’accesso ad Internet, o, ad esempio, accedere alla geolocalizzazione del dispositivo.

I permessi sono la risposta a questo bisogno: attraverso la dichiarazione delle attività dell’applicazione e delle risorse esterne richieste in un file di manifesto, è possibile conoscerne i requisiti. Il file xml in questione è l’“AndroidManifest”, documento di configurazione obbligatorio all’interno di ogni progetto che descrive la struttura del progetto ed i requisiti richiesti, nella forma di stringhe che prendono il nome di “permessi”.

I permessi che non pongono rischi alla privacy o che non interferiscono con le operazioni del dispositivo sono concessi automaticamente dal sistema; per i permessi considerati pericolosi (permesso che potrebbe ledere alla privacy dell’utente o alle operazioni del device) è invece richiesto l’intervento diretto dell’utente, che dovrà verificare personalmente se accettare o meno tali permessi.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.valdioveliu.myapplication" > <!-- the application package -->

    <!-- The list of permissions -->
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>

    <!-- The application -->

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="My Application" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

Figura 4: AndroidManifest.xml (in giallo le stringhe riportanti i permessi)

**Nota:** a seconda della versione del sistema operativo potrebbe variare la lista di permessi considerati pericolosi

### 3.1.3 Autenticazione dell’utente

Al primo avvio del dispositivo dopo il ripristino dei dati di fabbrica, viene richiesto all’utente di immettere un codice PIN / pattern / password a sua scelta, atto a riconoscerlo in avvii futuri. Questa registrazione iniziale crea un Security Identifier (SID) utente strettamente legato alla password scelta, in modo che questo possa funzionare da identificatore per l’utente.

Questo SID verrà inoltre usato per le operazioni di cifratura eseguite durante la sessione dall’utente.

### **3.1.4 Controllo della firma dell'applicazione**

Ogni applicazione designata per essere eseguita su Android deve essere in possesso di una firma digitale atta a riconoscere gli sviluppatori ed a garantire l'integrità del file APK (file contenente il codice dell'applicazione da installare).

Questo per due ragioni: da una parte se il riconoscimento della firma ha successo, si ha la certezza che il file d'installazione non sia stato modificato da fonti esterne; dall'altra questa verifica obbligatoria permette ad Android di ritenere responsabili gli sviluppatori originali per il comportamento dell'applicazione.

Su Android, la firma delle applicazioni è il primo passo per posizionare un'applicazione nella sua sandbox. Il certificato dell'applicazione firmato definisce quale utente è associato a quale applicazione; diverse applicazioni vengono eseguite con ID utente diversi. La firma delle applicazioni garantisce che un'applicazione non possa accedere ad altre applicazioni se non tramite Inter Process Communication (IPC) ben definito.

Al momento dell'installazione dell'applicazione, il Package Manager verifica che il file APK sia stato firmato correttamente con il certificato incluso all'interno. Se il certificato (o, più precisamente, la chiave pubblica nel certificato) corrisponde alla chiave usata per firmare qualsiasi altro APK sul dispositivo, il nuovo APK ha la possibilità di specificare nel proprio manifest che condividerà il proprio UID con ogni altro APK firmato similmente.

### **3.1.5 Chiavi crittografiche**

La necessità di dover trasmettere un messaggio attraverso canali non sicuri rende necessario celarne il contenuto attraverso appositi algoritmi di cifratura, per nasconderne il contenuto a terze parti potenzialmente ostili; in questo modo, anche se il messaggio venisse intercettato, un attaccante non sarebbe in grado di comprenderne il significato. Sono stati studiati molti algoritmi di cifratura in grado di raggiungere questo scopo, ma non è sicuro mantenere tutte le informazioni all'interno dell'algoritmo stesso; qualora l'attaccante sia a conoscenza dell'algoritmo utilizzato, questo sarebbe in grado di decifrare il contenuto del messaggio. È necessario quindi separare le parti logiche di questo procedimento, mantenendo il processo di cifratura nell'algoritmo ed aggiungendo una componente, chiamata “*chiave*”, che a seconda del suo valore, sia in grado di modificare

interamente il risultato di tale operazione (figura 5).

La chiave è quindi l'unica informazione che deve essere effettivamente tenuta nascosta; anche se l'attaccante avesse a disposizione l'algoritmo di cifratura, senza la chiave giusta non sarebbe in grado di decifrare il contenuto del messaggio.



Figura 5: Schema concettuale delle operazioni di cifratura

Per far ciò, Android mette a disposizione il sistema “Keystore”, un contenitore separato dai processi in esecuzione, con lo scopo di proteggere le chiavi e informazioni sensibili da un uso non autorizzato.

Quando un'applicazione esegue operazioni di crittografia utilizzando una chiave assegnata all'Android Keystore, il testo in chiaro, testo cifrato e messaggi da firmare vengono inviati a un processo di sistema che esegue le operazioni di crittografia in Trusted Execution Environment (TEE), ambiente sicuro dove solo il sistema operativo ha accesso. In questo modo se l'applicazione venisse compromessa, questa non sarebbe in grado di estrarne e riutilizzarne la chiave.

È possibile ottenere anche un grado di sicurezza aggiuntivo: per evitare la diffusione delle chiavi nel caso di manomissione del sistema Android, l'hardware dei dispositivi più recenti mette a disposizione un TEE direttamente su di un chip indipendente dal sistema operativo, dove è possibile conservare le chiavi ed eseguire operazioni di cifratura, senza che un attaccante possa ottenere informazioni dalla memoria del dispositivo.

### 3.1.6 Verified Boot

“Verifiedboot” è il processo che assicura all'utente, l'integrità del software presente sul dispositivo o la sua eventuale manomissione. A tale scopo, vengono conservate in un'area di memoria protetta le informazioni cifrate sulle partizioni, relative ad uno stato di avvio sicuro del dispositivo. Queste informazioni sono rappresentate nella forma di

hash, un valore che rappresenta in maniera univoca l'argomento di partenza, sui dati veri e propri o molto spesso sui metadati relativi alla partizione.

All'avvio del dispositivo, si controlla la legittimità di queste informazioni confrontandone la firma crittografica con la propria chiave. Se i valori contenuti combaciano con i valori hash delle partizioni attuali del dispositivo, si ha la certezza che il codice del dispositivo non sia stato sottoposto a modifiche esterne e che si stia caricando una versione legittima di Android.

## 4 Android Things

---

### 4.1 Introduzione

Dopo tre anni di sviluppo, nel maggio del 2018 Google ha rilasciato la prima versione di Android Things [8], sistema operativo dedicato al mondo dell’Internet degli Oggetti che si pone l’obiettivo di arginare le note falle di sicurezza presenti nei dispositivi IoT<sup>13</sup>. Pensato per operare in perfetta sincronia con l’omonima versione mobile e con i servizi Google sviluppati, Android Things si pone l’obiettivo di diventare il nuovo standard industriale per dispositivi IoT; coprendo le lacune di sicurezza dei dispositivi precedenti, si fornisce agli sviluppatori l’occasione di concentrarsi interamente sulla propria app, il cui sviluppo è pressoché invariato rispetto alla versione mobile, e sulla sua implementazione nel device attraverso una comoda console web, dove si gestisce la parte di update, di testing e di rilevazione delle statistiche.

### 4.2 Devices

Android Things (A.T.) è pensato per operare su piattaforme a single-board computer (sia da 32 che da 64 bit) dalla memoria limitata (sebbene la memoria minima richiesta da A.T. sia 512 MB). Google certifica la compatibilità con i System-on-Module (SoM) sviluppati con i propri partner commerciali e fornisce i Board Support Package (BSP) necessari per l’integrazione di A.T. con la parte hardware.



Figura 6: Single-board computer “*RaspberryPi 3 Model B*”

---

<sup>13</sup> <https://developer.android.com/things/get-started>

Attualmente sono disponibili solamente due piattaforme hardware a single-board computer per creazione di prototipi e testing [9]:

- “NXP Pico i.MX7D” (ARM Cortex A7, 32-bit)
- “RaspberryPi 3 Model B” (ARM Cortex A53, 64-bit) [Figura 6]

In futuro è previsto il rilascio di nuove piattaforme che consentiranno la creazione ed il rilascio di devices IoT commerciali nel mercato. Rientrano nella lista le seguenti board:

- NXP i.Mx8M (ARM Cortex A53, 64-bit + QC7000Lite GPU)
- Qualcomm SDA212 (ARM Cortex A7, 32-bit + QC Adreno 304 GPU)
- Qualcomm SDA624 (ARM Cortex A54, 64-bit + QC Adreno 506 GPU)
- MediaTek MT8516 (ARM Cortex A35, 64-bit)

Il design modulare dei SoM permette agli sviluppatori di iniziare la progettazione della propria applicazione su una board riservata al testing, per poi trasferirla senza bisogno di modifiche sulle versioni commerciali per il rilascio sul mercato del prodotto finale.

### 4.3 Gestione dei dispositivi

Attraverso l'infrastruttura Weave (rappresentata in figura 7), Google si fa completamente carico della gestione delle comunicazioni tra l'interfaccia d'accesso per l'utente, i servizi Google usati, il device Android Things e la compagnia responsabile della produzione del device.

Per la produzione di nuovi dispositivi, Google mette a disposizione la console di sviluppo Android Things<sup>14</sup>, strumento dalle molteplici funzionalità per la creazione di un nuovo prodotto, la gestione dell'applicazione per il device SoM scelto, il flashing del dispositivo, la visualizzazione delle prestazioni e la distribuzione degli aggiornamenti. All'utente viene invece fornito, attraverso Weave, una comoda applicazione mobile per poter avviare, visualizzare e comunicare con tutti i suoi dispositivi in possesso ed in qualunque posizione del globo.

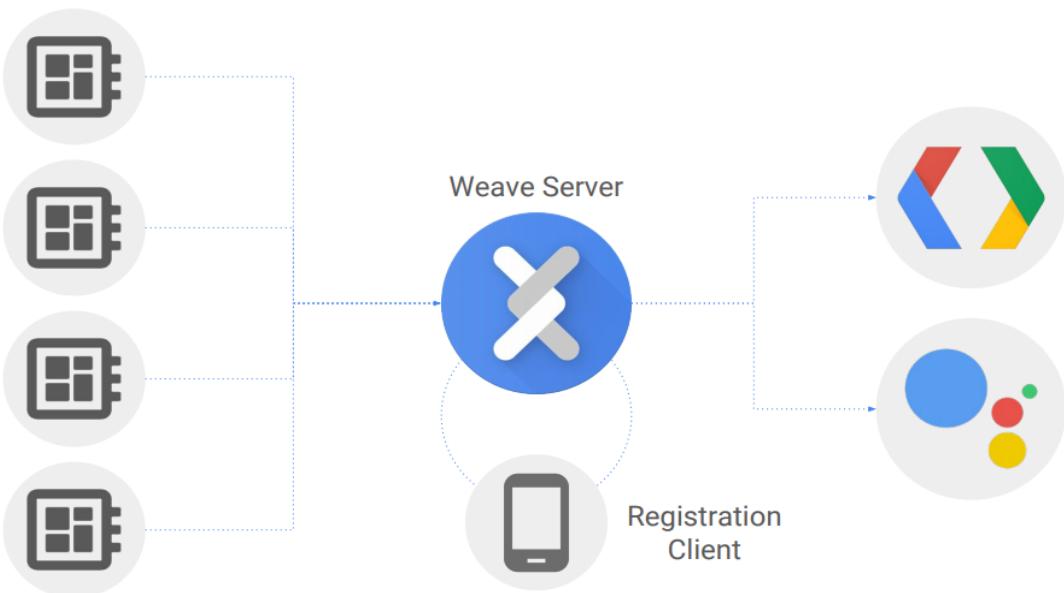


Figura 7: Schema di comunicazione *Google Weave*.  
A sinistra sono rappresentati i devices IoT, a destra i servizi Google.

<sup>14</sup> <https://partner.android.com/things/console/>

## 4.4 Architettura

Il sistema operativo Android Things si basa sull'architettura della sua controparte Android [sezione 3], ripresentando tutte quelle caratteristiche che l'hanno reso famoso per il mondo mobile; le differenze che lo contraddistinguono, oltre al fatto di essere un Sistema Operativo (SO) dedicato agli oggetti e non ai cellulari, si basano soprattutto sulla forte autonomia dall'utente, caratteristica del mondo IoT.

Nelle figure 8 e 9 è possibile osservare una prima differenza tra le due architetture; nell'ecosistema IoT, l'utente non ha più il ruolo di controllore, dove necessita di pieno controllo del dispositivo, ma è relegato ad un ruolo più esterno, fonte di dati ed input con cui il device interagisce: lo schermo diventa quindi un optional, mentre solamente l'applicazione principale (designata “home activity”) viene lanciata all'avvio del device.

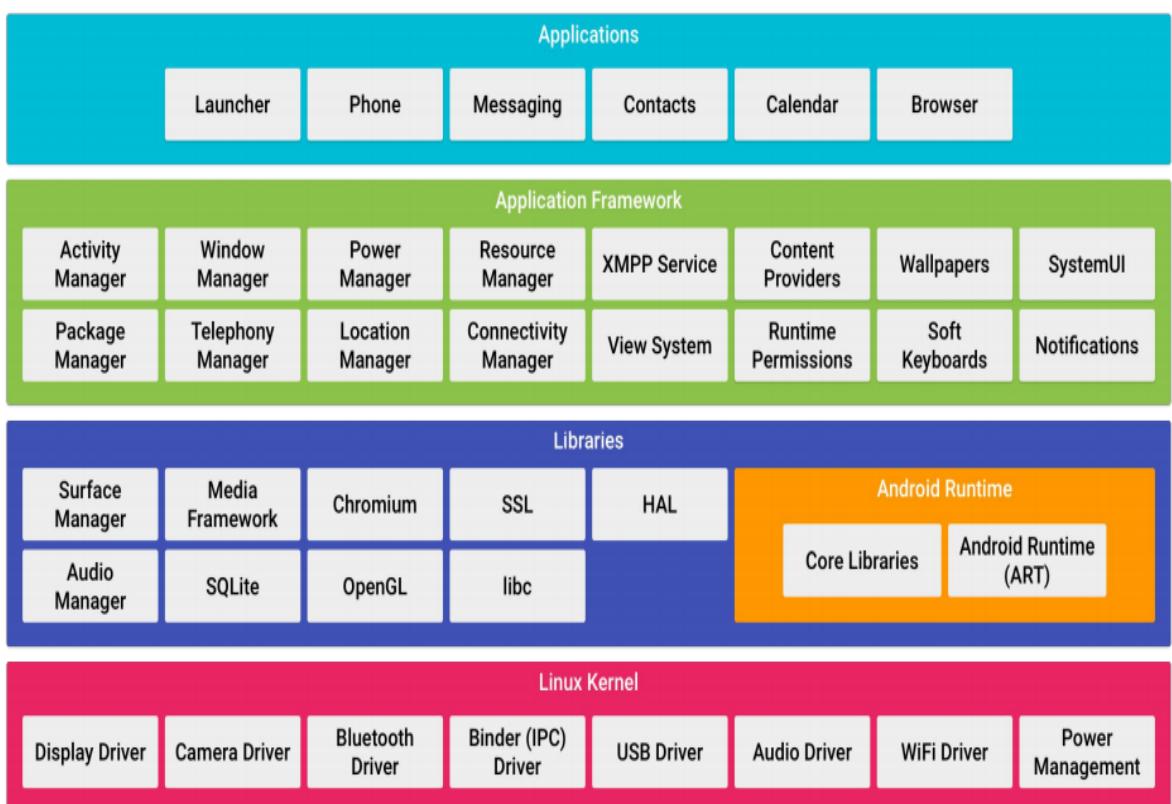


Figura 8: Architettura Android

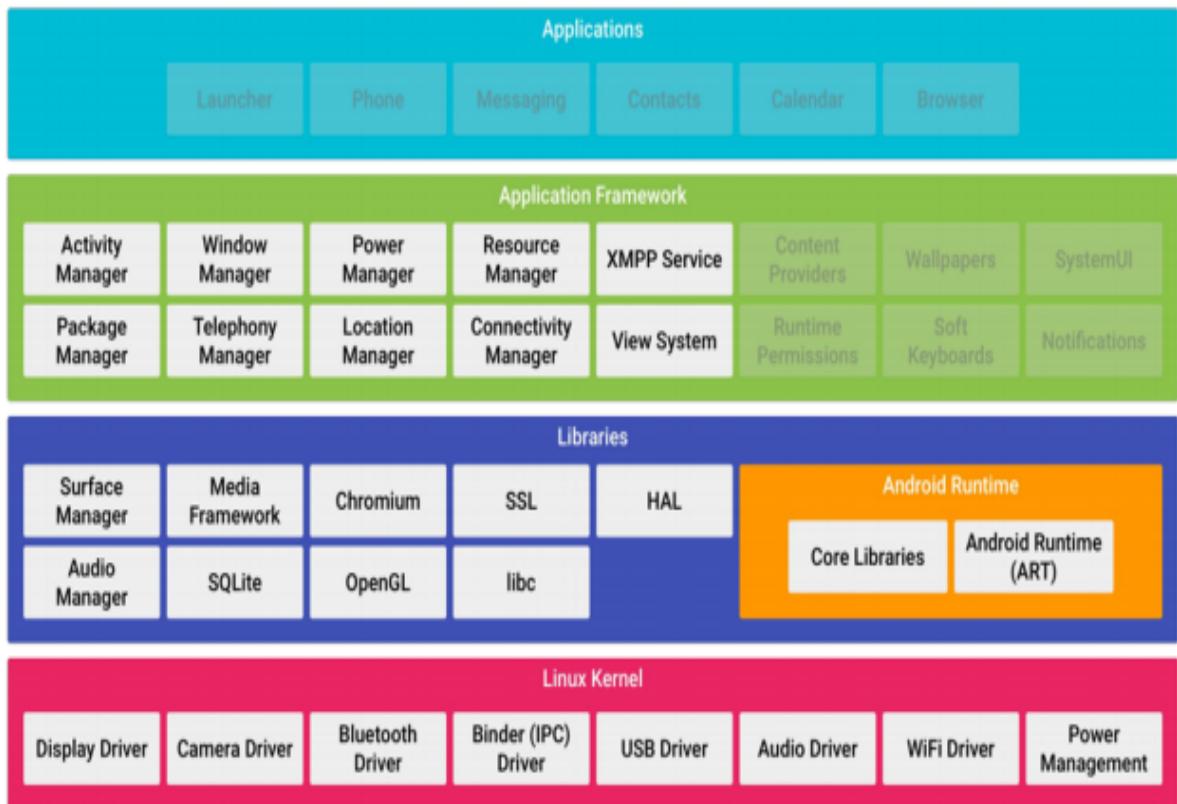


Figura 9: Architettura Android Things a confronto con Android

Sono quindi state rimosse tutte le applicazioni di sistema e le seguenti API legate alle utilities per l'utente (in generale legate ad input o autenticazione) [10]:

- NotificationManager
- KeyguardManager
- WallpaperManager
- SpeechRecognizer
- FingerprintManager
- NfcManager
- SmsManager
- TelephonyManager

- UsbAccessory
- WifiAwareManager
- AppWidgetManager
- AutofillManager
- BackupManager
- CompanionDeviceManager
- Activity Picture-in-picture
- PrintManager
- SipManager

Inoltre, per il riadattamento della sicurezza Android nel mondo IoT (sezione 4.5), sono stati rimossi i permessi runtime, accettati dall'utente durante l'esecuzione dell'applicazione. Sono ancora presenti molte API legate al mondo mobile, come Telephony manager, ma se ne sconsiglia fortemente il loro utilizzo<sup>15</sup>:

- CalendarContract
- ContactsContract
- DocumentsContract
- DownloadManager
- MediaStore
- Settings
- Telephony
- UserDictionary
- VoicemailContract

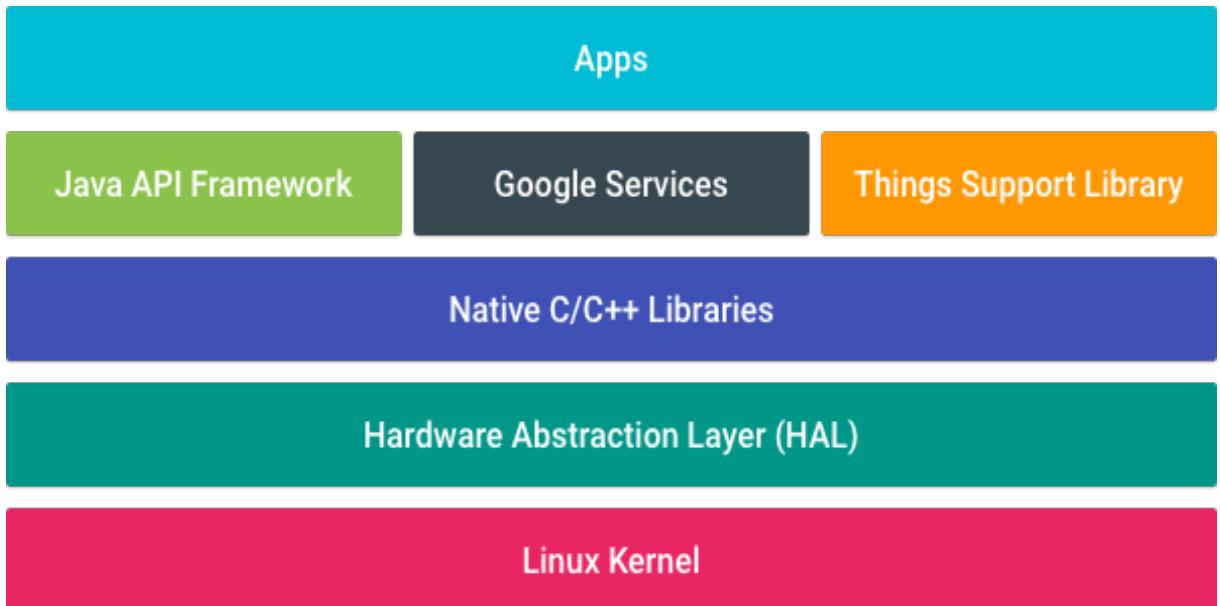


Figura 10: Architettura per dispositivi Android Things

La *figura 10* riporta all'interno quanto rappresentato in figura y, con alcune aggiunte nel livello Application Framework:

### *Google Services*

Nonostante queste siano un sottoinsieme delle API già presenti in Android, A.T. è pensato per operare in armonia con i servizi Google, come *Firebase*, *Cloud* e *TensorFlow*. Viene quindi sottolineato quest'aspetto nello schema.

### *Android Things Support Library*

Sono state aggiunte API per la comunicazione con sensori ed altri dispositivi secondo i più recenti standard industriali<sup>15</sup>. Attualmente sono supportate le seguenti interfacce di comunicazione: GPIO, PWM, I2C, SPI, UART.

---

<sup>15</sup> <https://developer.android.com/things/get-started/platform-differences#intents>

<sup>16</sup> <https://developer.android.com/things/sdk/pio>

Sono presenti API nuove o aggiornate per gestire la comunicazione con altri dispositivi:

- *Bluetooth*. Versione rivista della omonima API, con interfaccia semplificata per permettere un accoppiamento ed uno scambio di dati tra reti di devices-

- *LoWPAN* (LOw-power Wireless Personal Area Network).

Estensione dell'API per gestire la comunicazione WiFi, permette di creare, scansioneare ed unirsi a reti mesh a bassa potenza, come la rete “*Thread*” [11]. Questa tipologia di rete mesh IPv6 fornisce un'infrastruttura di comunicazione resiliente e sicura, senza alcun punto di fallimento [12] .

È inoltre presente l’”User Driver API” <sup>17</sup>, che permette all’utente di creare ed utilizzare i propri driver, attraverso l’iniezione di listener nell’applicazione di eventi hardware.

Un’altra differenza che contraddistingue Android Things si trova nel livello applicativo. Con l’aumento della capacità di calcolo e della memoria, i cellulari sono passati dall’essere strumenti di chiamata, a strumenti multiruolo con un numero elevato di applicazioni, ognuna con competenze diverse: messaggistica, navigazione stradale, gestione delle risorse, fino ad arrivare ad app di svago. Con un paio di click si può accedere ad un mercato centrale, il Play Store, dove ognuno può caricare e vendere qualsiasi tipo di applicazione, con il requisito di rispettare gli standard di sicurezza Android.

Nel mondo IoT, ogni device è pensato e caratterizzato per rispettare un’unica determinata funzione, ed i meccanismi di personalizzazione vengono abbandonati a favore di una maggiore sicurezza. Android Things rimuove dal livello applicativo ogni applicazione di sistema, ora inutili, ed al momento dell’avvio del dispositivo viene quindi lanciata l’applicazione principale, definita “**applicazione home**”.

Non vi è più spazio per un “acquisto di un’applicazione”, in quanto ora l’applicazione corrisponde al device stesso. In questa nuova ottica, costruire un device A.T. significa sviluppare la home activity. In A.T. non è quindi presente uno store centrale per condividere applicazioni, in quanto queste sono applicazioni OEM altamente specializzate e prodotte direttamente dall’azienda responsabile (o il loro sviluppo viene affidato su contratto a compagnie terze).

---

<sup>17</sup> <https://developer.android.com/things/sdk/drivers>

## 4.5 Sicurezza

Anche la sicurezza è stata riadattata al mondo IoT, in un’ottica dove il device si astiene il più possibile dal controllo dell’utente diventando il più possibile autosufficiente.

A differenza di quanto accade nel mondo mobile, Android Things è un sistema operativo chiuso, non lasciando la possibilità di modificarne liberamente le caratteristiche di sistema. A livello di gestione, allo sviluppatore viene lasciata la scelta di quale versione implementare, quando eseguire un upgrade disponibile e di come gestire gli aggiornamenti attraverso la console di sviluppo.

Nonostante le differenze che contraddistinguono A.T. con Android, si mantengono inalterati alcuni meccanismi di sicurezza, anche se a volte meno significativi, in modo da ridurre al minimo il numero possibile di superfici di attacco.

### 4.5.1 Sandbox delle applicazioni

Si ripropone il meccanismo d’isolamento di applicazioni e risorse, eseguendo ogni processo attraverso ART, sotto un determinato UID; sono lasciati inalterati i Controlli di Accesso Obbligatori (MAC) attraverso SELinux, insieme a tutti i filtri delle kernel syscall.

### 4.5.2 Privilegi

I programmi che vengono eseguiti operano sotto il principio del *minor privilegio possibile* (se non hanno bisogno di un permesso, allora non lo hanno a disposizione). In questo modo, qualora il dispositivo venisse infettato, non sarebbe in grado di accedere a risorse di cui non dispone l’autorizzazione.

Sono stati però disabilitati i *permessi in runtime*; al primo avvio, sarà richiesto all’utente l’accettazione “in blocco” di tutti i permessi richiesti dall’applicazione. È possibile però replicarne il meccanismo, attraverso una seconda applicazione dedicata solamente all’accesso di una particolare risorsa; l’applicazione principale non sarà quindi in possesso di quel determinato permesso, e per leggere o scrivere sull’obiettivo, dovrà agirvi indirettamente attraverso le regole stabilite nell’applicazione specializzata. Non viene richiesta comunque nessuna conferma all’utente, ma nel caso che la prima applicazione venisse infettata, non potrebbe accedere direttamente a quella risorsa.

#### **4.5.3 Cifratura e Keystore**

Sono presenti tutti gli algoritmi, i meccanismi di cifratura ed il keystore presenti in Android.

Non viene effettuata una cifratura completa del disco, in quanto i devices potrebbero aver bisogno di riavvii automatici. Tutte le informazioni sensibili devono essere quindi cifrate nella memoria flash dall'applicazione stessa, con una chiave conservata nell'hardware backedkey attraverso la Keystore API.

#### **4.5.4 Verified Boot Hash**

Anche in A.T. è presente il meccanismo di VerifiedBoot del dispositivo, dove si usano le firme crittografiche per verificare il codice e tutte le partizioni di memoria dietro alle operazioni di avvio.

Questo meccanismo viene esteso per ottenere informazioni in fase di attestazione: ad ogni stage del boot del dispositivo, il valore hash dato dai metadati rappresentanti le partizioni di memoria interessate, viene mixato nell'hash del livello di boot precedente. In questa maniera, al termine del processo di avvio, si ottiene un valore hash che riesce a rappresentare l'esatto software attualmente eseguito nel sistema, sia a livello di sistema operativo, sia a livello di applicazione. Qualora il dispositivo venisse infettato con successo, un confronto di questo valore hash con quello aspettato permette all'utente di individuare rapidamente l'anomalia, e di avviare tempestivamente le contromisure necessarie.

#### **4.5.5 Anti Rollback Attack**

Il “Rollback Attack” si basa sul portare il sistema ad una versione precedente, in modo da sfruttarne le vulnerabilità presenti e portare il dispositivo nuovamente in uno stato insicuro ed attaccabile.

Viene quindi evitato questo exploit accettando che, sia l'utente che lo sviluppatore, possano decidere quando effettuare un aggiornamento di sistema. Una volta che questo ha avuto successo, si preclude in futuro ogni suo possibile downgrade. Questo viene riproposto anche per il meccanismo di rotazione delle chiavi di sicurezza, dove una volta che viene generata una nuova chiave, non è più possibile utilizzare quella precedente.

#### 4.5.6 Certificazione

Non essendo presente un market centrale ed essendoci una notevole complessità organizzativa dietro alla creazione di un prodotto, si rende necessaria la presenza di una catena di fiducia responsabile della convalidazione di ogni componente hardware e software.

Attraverso la console di sviluppo AndroidThings, Google riesce ad assumere la figura di intermediario tra produttori e consumatori, imponendosi come certificate authority (CA) di fiducia e sviluppando una catena di certificati digitali (in figura 11) che riesce a garantire il prodotto nella sua interezza.

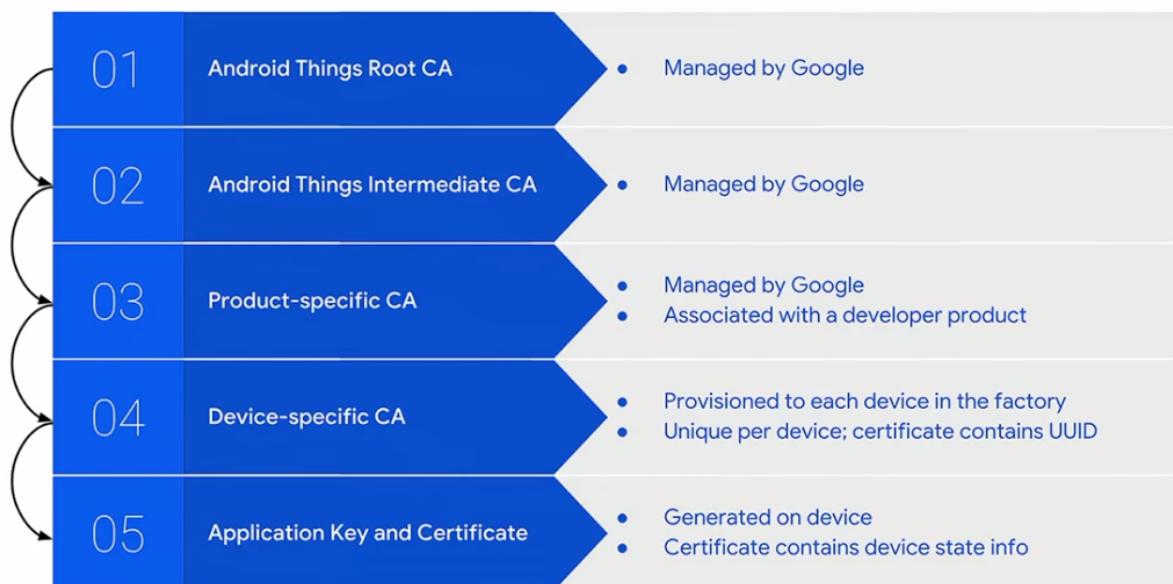


Figura 11: Catena di certificati di un device Android Things

- Android Things Root CA e intermediate CA, entrambi gestiti da Google, sono l'autorità per ogni certificazione di Android Things.
- Product-specific CA, riguarda la console di sviluppo AndroidThings, dove si legano le informazioni sull'azienda e sul prodotto con il proprio intermediate-level CA.
- Device Specific CA, unico per ogni dispositivo, viene generato a partire dal suo UUID.

- Application Key and Certificate. Generate dall'applicazione a partire dal Device Specific CA, e memorizzati in TEE.

#### 4.5.7 Attestazione remota

L'attestazione remota<sup>18</sup> è il processo che permette all'utente di interrogare il device, per autenticarne la provenienza e conoscerne lo stato attuale.

Per evitare il coinvolgimento diretto di una delle parti, si utilizza la catena di certificati digitali del dispositivo per garantire l'anonimità del processo. Come mostrato in figura 12, l'utente ha quindi modo di verificare tutte le caratteristiche del prodotto.

(Tutte le operazioni sono effettuate attraverso una cifratura a chiave pubblica, in modo che le informazioni scambiate possano essere lette solamente dai programmi coinvolti)

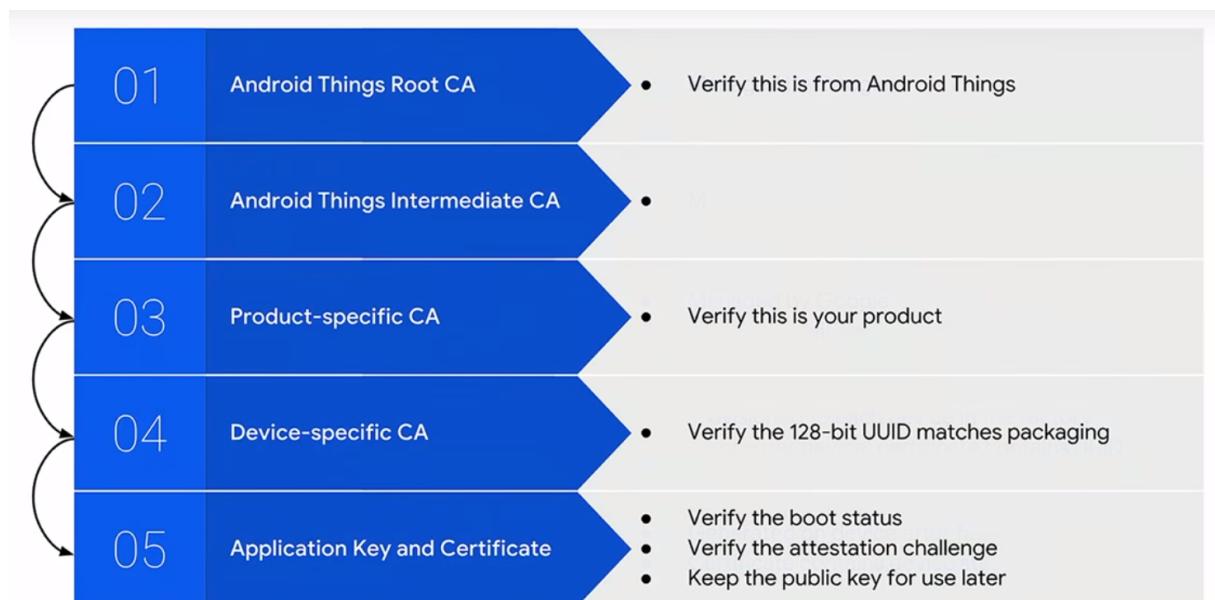


Figura 12: Significati dietro alle operazioni di verifica dei certificati

<sup>18</sup> [https://it.wikipedia.org/wiki/Direct\\_Anonymous\\_Attestation](https://it.wikipedia.org/wiki/Direct_Anonymous_Attestation)

#### **4.5.8 Nessuna comunicazione con il sistema operativo**

Nella modalità release il sistema operativo chiude ogni suo canale di comunicazione con l'esterno, incluse le porte Android Debug Bridge (ADB), per evitare manomissioni da terze parti. Ogni comunicazione, inclusi gli aggiornamenti di sicurezza, dovranno prima passare attraverso l'applicazione principale, responsabile del funzionamento del device.

Nella modalità testing per facilitare le operazioni, sarà sempre possibile leggere i file di LOG ed interagire direttamente con il sistema operativo, attraverso la porta ADB.

## 4.6 Aggiornamenti

Gli aggiornamenti OTA per il device<sup>19</sup>, gestiti dall'apposita interfaccia nella console di sviluppo AndroidThings, si dividono in due categorie: aggiornamenti dell'applicazione (sviluppata dall'azienda responsabile del device) ed aggiornamenti di sistema (sviluppati da Google e divisi internamente tra aggiornamenti maggiori del sistema, aggiornamenti minori sulle API ed aggiornamenti sulla sicurezza [figura 13]).



Figura 13: Informazioni sulla versione A.T.

Per ogni nuovo SoM supportato, Google assicura il rilascio di aggiornamenti per un periodo minimo di tre anni per ogni maggiore versione A.T.<sup>20</sup>.

Nuove versioni potrebbero però interferire con le applicazioni installate sul dispositivo; viene quindi lasciata libertà alle aziende responsabili su quando e con che modalità rilasciare questi aggiornamenti, in modo da poter effettuare una fase di testing, prima del rilascio dell'update ai propri clienti.

Ma mentre sono le compagnie a decidere quale versione di A.T. utilizzare, ci si aspetta che queste non ignorino gli updates minori sulla sicurezza. A discrezione dello sviluppatore, sarà quindi possibile automatizzarne il rilascio a tutti i dispositivi connessi. Google si riserva inoltre la facoltà di forzare questi aggiornamenti qualora il prodotto non venga più seguito dall'azienda.

Per il testing sono a disposizione numerosi canali completamente personalizzabili, attraverso i quali è possibile fornire versioni di build diverse per sviluppo, beta testing e produzione. In questo modo, mentre i dispositivi rilasciati nel mercato saranno tutti

<sup>19</sup> <https://developer.android.com/things/console/update>

<sup>20</sup> Google I/O '18

parte del canale “release”, l’azienda potrà testare prima gli effetti dell’aggiornamento sui propri device nel canale “beta”.

Questi aggiornamenti vengono così distribuiti in una determinata finestra temporale ai devices appartenenti al canale selezionato; qualora la compagnia responsabile noti delle incompatibilità con il nuovo aggiornamento, durante questa fase le sarà possibile tornare alla versione precedente.

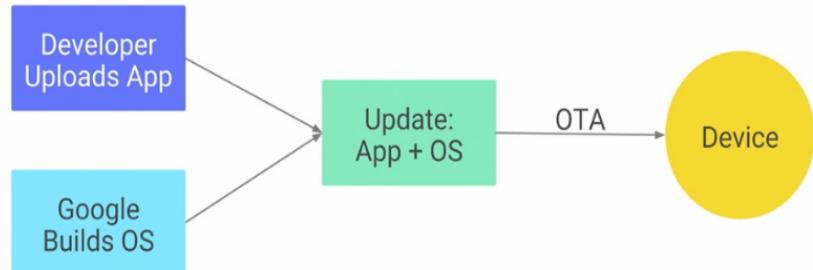


Figura 14: Schema degli aggiornamenti Android Things

A livello di dispositivo, l’API “Device Updates” consente ad un’applicazione di controllare e monitorare la gestione degli aggiornamenti software OTA, definendo nel manifesto la politica d’aggiornamento (se eseguire il check, check ed aggiornamento, check con aggiornamento e reboot automatico).

Nella classe Update Service verrà invece descritto il canale a cui il dispositivo si dovrà collegare per la ricezione degli aggiornamenti. Per ovvie ragioni, un dispositivo potrà iscriversi ad un solo canale alla volta.

Una volta che verrà rilevata la disponibilità di un aggiornamento, a seconda della politica specificata si procederà con il suo download (figura 15).

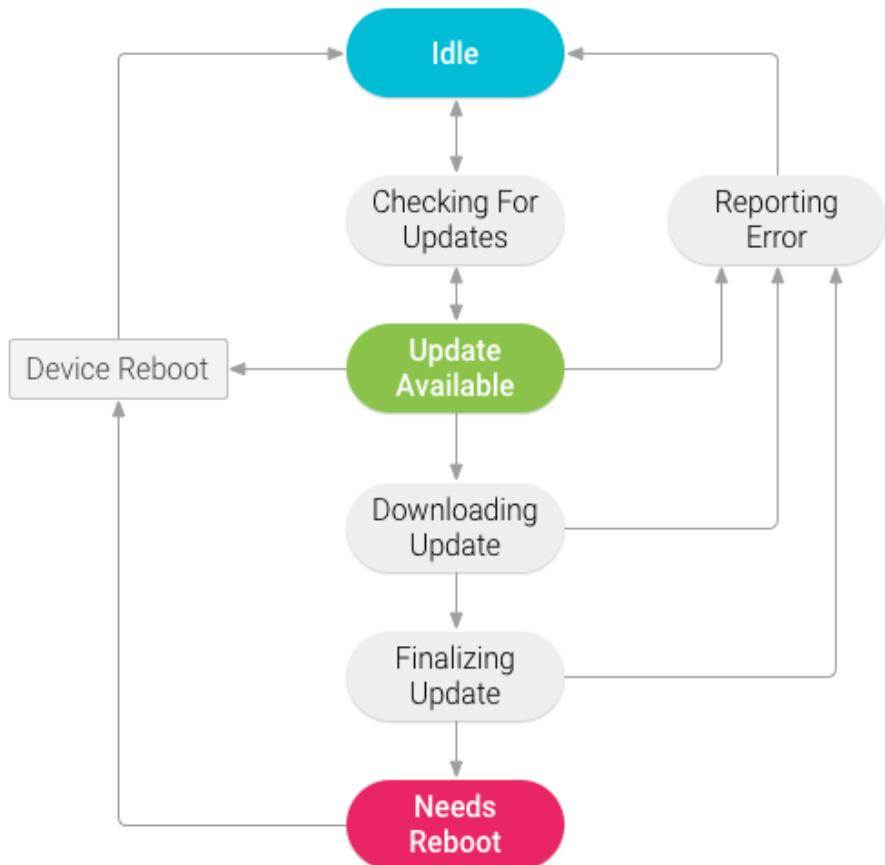


Figura 15: Diagramma di flusso della gestione degli aggiornamenti del dispositivo

Come è possibile vedere in figura 16, nel device sono presenti due partizioni principali di memoria, chiamate AB, che ripropongono le medesime categorie d'informazione. Durante il normale avvio del dispositivo, si utilizzano le immagini contenute in una sola partizione alla volta, lasciando l'altra in uno stato “idle” con una copia di backup delle immagini.

Solamente con il download dell'aggiornamento si andrà ad attivare la partizione inutilizzata: lì verrà installato in background il contenuto dell'update, così che il device possa continuare a svolgere la sua applicazione principale; dopo il riavvio in SecureBoot con le immagini contenute nella seconda partizione, il device potrà quindi godere dei nuovi aggiornamenti.

In caso di errore, il dispositivo potrà sempre tornare ad utilizzare la vecchia partizione di memoria, ritornando operativo in breve tempo.

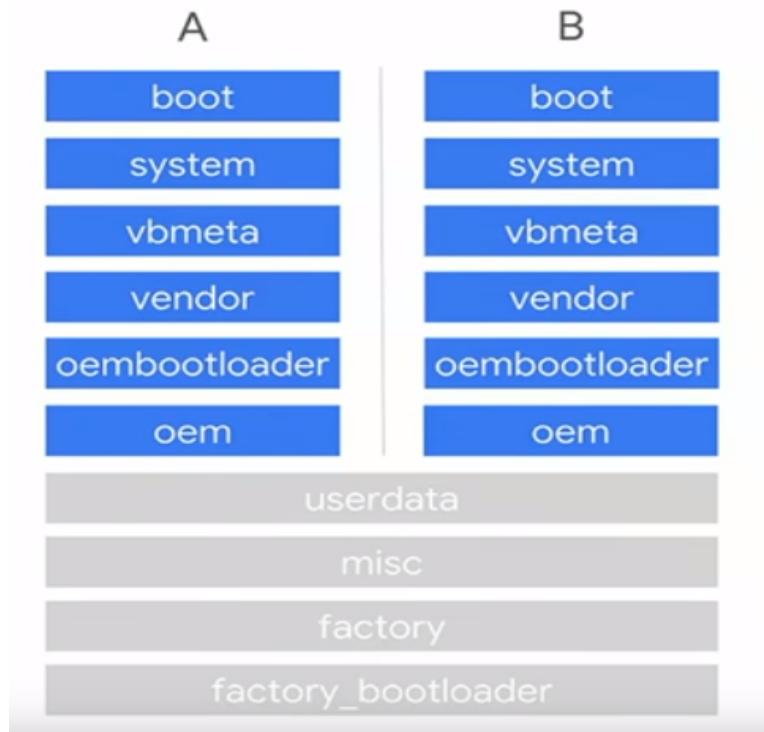


Figura 16: Partizione di memoria AB

Dato che la memoria del device è limitata, la dimensione delle partizioni di memoria è decisa al momento in cui si esegue l'operazione di flashing del dispositivo; i devices potranno eseguire gli aggiornamenti OTA solamente di build che dispongano delle medesime dimensioni di partizionamento. Viene quindi consigliato di lasciare un certo buffer, per permettere una crescita futura.

## 4.7 Funzionamento

Android Thing è pensato per lavorare in sincronia con la controparte mobile per la gestione del dispositivo, e con i numerosi servizi Google per assicurarsi un'infrastruttura di supporto stabile e sicura. Non è presente però una situazione di monopolio da parte di Google; le scelte di implementazione rimangono totalmente nelle mani degli sviluppatori, liberi di costruire ed usufruire di qualsiasi servizio presente in rete.

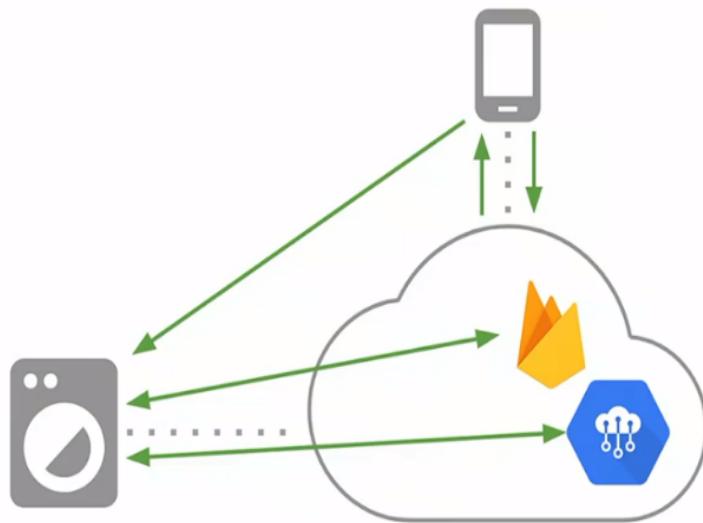


Figura 17: Rappresentazione dell'ecosistema telefono-device-web

Ogni device dovrà quindi rispondere a delle necessità fondamentali, quali:

- Inizializzazione e registrazione del proprietario
- Gestione delle impostazioni di accesso e sicurezza
- Comunicazione con servizi esterni
- Attestazione del proprio stato
- Ritorno ad uno stato “senza proprietario” con eliminazione dei dati sensibili raccolti

Non è detto però che i device AT dispongano di interfacce in grado di permettere una comunicazione con l'utente; le porte adb, utilizzate dai dispositivi Android per permettere la comunicazione con un dispositivo esterno, non sono disponibili una volta che si effettua il deploy del dispositivo. Per questo sono necessarie operazioni di autenticazione a due vie particolari, tra telefono e device, al termine delle quali devono essere garantite al device tutte le informazioni necessarie per interagire con l'esterno.

Il primo passo da eseguire sarà quindi l'inizializzazione del dispositivo attraverso il primo accesso dell'utente.

Grazie al supporto dell'API "Nearby Connection"<sup>21</sup>, che fornisce un layer astratto per la connessione con devices nelle vicinanze a prescindere se questo richieda una connessione bluetooth o wifi, ed alla infrastruttura "Weave", sarà possibile rilevare attraverso il proprio smartphone i dispositivi Android Things registrati a canali pubblici nelle vicinanze.

L'utente potrà quindi effettuare il primo accesso comodamente da cellulare, impostandosi come proprietario ed inviando tutti i dati richiesti dal device per la navigazione online. Sarà quindi necessario specificare il punto d'accesso attraverso il quale il dispositivo dovrà connettersi, ma molto più importante, stabilire una prima autenticazione tra utente e device in modo che il dispositivo possa usufruire correttamente dei servizi in rete.

La modalità principale con cui vengono eseguite operazioni di autenticazione è attraverso la verifica del "bearer token", file codificato con cui poter rappresentare l'identità dell'utente, senza doverne riportare informazioni sensibili (già immesse in un primo login).

L'utente potrà quindi autenticarsi con il servizio in rete attraverso una classica interfaccia di login, ricevendo così il proprio token d'autenticazione; successivamente, attraverso il collegamento diretto con il device, sarà possibile trasferire le autorizzazioni interessate al proprio dispositivo, che sarà ora in grado di comunicare con i servizi web sotto l'identità dell'utente.

---

<sup>21</sup> <https://developers.google.com/nearby/connections/overview>

## 4.8 Sicurezza dei due sistemi

Sono stati testati vari scenari di sicurezza a livello applicativo nella versione 1.0.21 di Android Things, utilizzando un Raspberry Pi 3 model B.

Con l’eccezione della parte di gestione dell’interfaccia utente (assente in Android Things), non sono state rilevate anomalie nel confronto tra applicazioni A.T. ed applicazioni Android.

È possibile quindi applicare i controlli di sicurezza già consolidati nel mondo mobile [13] a livello statico e dinamico.

Note sulle API:

È suggerito astenersi dall’usare API provenienti dal mondo mobile, ma mantenute in Android Thing. Tali API sono incentrate su un dispositivo a stretto contatto con l’utente, come la gestione dei relativi contatti, la gestione del download dal web, la gestione di file multimediali etc.

Tra queste rientra anche il content provider, pensato per gestire l’accesso alle risorse esterne di un’applicazione, ma durante i test eseguiti non sono state rilevate grosse differenze con la versione Android. Rimane però consigliato un’accesso diretto alle risorse, oppure un accesso tramite una seconda applicazione di supporto.

L’API Bluetooth presenta alcuni problemi, dove “devices con input utente differenti potrebbero non essere in grado di interagire tra loro” <sup>22</sup>.

Un warning sulle API relative alla gestione delle comunicazioni con le periferiche, una porta in uscita conserva il suo ultimo valore in uscita anche dopo che il metodo close() di chiusura è chiamato. Se ciò si rivela un problema, è quindi necessario riportare i settaggi al valore di default manualmente, attraverso gli appositi metodi.

---

<sup>22</sup> <https://developer.android.com/things/sdk/apis/bluetooth>

## PUNTI DI FORZA

I maggiori punti di forza di Android Things sono certamente il riproporre uno sviluppo delle applicazioni con linguaggio Java e modalità identiche allo sviluppo Android, già consolidato da tempo, ed il fornire un’infrastruttura collaudata e nascosta in grado di mantenere una comunicazione sicura tra produttori e qualsiasi dispositivo registrato. Riproponendo le top 10 OWASP IoT vulnerabilities [sezione 2], si nota che Google prende carico di molti problemi che affliggono l’Internet degli Oggetti (figura 18 a pagina seguente), assumendosi la responsabilità di fornire una piattaforma hardware stabile e sicura e lasciando le compagnie produttrici di devices libere di concentrarsi sulla parte applicativa, snellendo così il processo produttivo.

## PROBLEMI PRESENTI

La compatibilità con Android, uno dei principali punti di forza, arriva ad un costo. I requisiti minimi di sistema di Android Things sono piuttosto alti se paragonati ad altri sistemi operativi per SoM tradizionali, come “mbed Os” [14].

Davvero un frigorifero intelligente ha bisogno di un processore quad-core e di 512 MB di Ram?

Certamente una maggiore potenza di calcolo fa sempre comodo, ma dato che questi dispositivi saranno integrati nel Web, un’alternativa potrebbe essere quella di delegare elaborazioni di dati particolarmente onerose ad altri devices più performanti o a sistemi più modulari, specializzati in quel campo.

Inoltre molte delle API e delle features importanti sono ancora in fase di testing o sviluppo, ed attualmente non sono presenti piattaforme SoM con le quali rilasciare il prodotto nel mercato. La stessa ricerca delle applicazioni risulta ostacolata, non essendoci un mercato ufficiale dove condividerle (l’unica possibilità di condivisione attuale è attraverso la condivisione con un proprio contatto gmail).

<b>1</b>	<b>Weak, Guessable, or Hardcoded Passwords</b> Use of easily bruteforced, publicly available, or unchangeable credentials, including backdoors in firmware or client software that grants unauthorized access to deployed systems.	
<b>2</b>	<b>Insecure Network Services</b> Unneeded or insecure network services running on the device itself, especially those exposed to the internet, that compromise the confidentiality, integrity/authenticity, or availability of information or allow unauthorized remote control...	
<b>3</b>	<b>Insecure Ecosystem Interfaces</b> Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering.	
<b>4</b>	<b>Secure Update Mechanism</b>	
<b>5</b>	<b>Components</b>	
<b>6</b>	<b>Insufficient Privacy Protection</b> User's personal information stored on the device or in the ecosystem that is used insecurely, improperly, or without permission.	
<b>7</b>	<b>Insecure Data Transfer and Storage</b> Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing.	
<b>8</b>	<b>Device Management</b>	
<b>9</b>	<b>Default Settings</b>	
<b>10</b>	<b>Physical Hardening</b>	

Figura 18: Top 10 OWASP IoT vulnerabilities, parti prese a carico da Android Things.

## 5 Analisi di sicurezza

---

### 5.1 Introduzione

Con il termine “analisi di sicurezza” si fa riferimento ad un insieme molto numeroso di tecniche e verifiche per attestare la sicurezza di un’applicazione, ognuna con un obiettivo ed una metodologia differente. Queste tecniche vengono quindi divise in due gruppi logici: le tecniche di analisi statica, che agiscono sul codice sorgente dell’applicazione, e le tecniche di analisi dinamica, che operano durante l’avvio e l’esecuzione dell’applicazione da testare.

In questo capitolo vengono descritte le analisi utili ad assestarsi lo stato di sicurezza di un’applicazione mobile, spiegandone per ognuna il procedimento e gli obiettivi; successivamente ci si sofferma sul framework OWASP, framework di analisi approvato dalla comunità scientifica, con lo scopo di riadattarlo a delle analisi di dispositivi IoT.

### 5.2 Analisi statica

Le tecniche di analisi statica fanno riferimento all’analisi del codice sorgente o del bytecode dell’applicazione, per determinare la presenza (o la mancanza) di eventuali proprietà del programma senza eseguire l’applicazione in sè [15].

Uno dei limiti dell’analisi statica è quello di analizzare completamente il codice dell’applicazione, incluse le parti che potrebbero non venire effettivamente eseguite, o vulnerabilità individuabili solamente durante l’esecuzione dell’applicazione. Anche se la parola d’ordine è “automatizzazione”, quest’analisi tende a produrre molti falsi positivi e falsi negativi. Al termine di queste operazioni sarà quindi richiesta una corretta interpretazione dei risultati, fornendo una falsa sensazione che tutto sia stato controllato a sviluppatori meno esperti.

Rientrano in questa categoria le:

- Analisi del manifest applicativo

La lettura automatizzata del file manifest permette di rilevare la struttura dell’applicazione ed i permessi richiesti da essa, al fine di avere un’idea generale della tipologia delle operazioni eseguite.

- Analisi dei permessi

Viene analizzato il codice dell'applicazione per stilare una lista di tutti i permessi potenzialmente richiesti a runtime, per verificare che non vi siano permessi dichiarati in eccesso (potenziali superfici d'attacco sfruttabili in caso d'attacco) o che non tutti i privilegi vengano dichiarati (quindi nascosti all'utente).

- Analisi di pattern di vulnerabilità

Vengono confrontati frammenti di codice dell'applicazione con dei pattern già frequentemente riscontrati in applicazioni vulnerabili, stabilendone la somiglianza (procedimento attuato ad esempio durante le scansioni malaware, per stabilire se l'applicazione contiene un virus al suo interno); se questa corrispondenza supera una certa soglia, la porzione di codice viene classificata anch'essa come vulnerabile.

- Analisi delle librerie

Vengono analizzate ed identificate le librerie utilizzate dall'applicazione al fine di riconoscere se queste contengono al loro interno vulnerabilità note, se risultano essere obsolete o addirittura eliminate in determinate versioni del sistema operativo bersaglio.

- Analisi del livello di offuscamento

Questo tipo di analisi serve a stabilire quanto un'applicazione risulta essere resistente ad operazioni di reverse engineering, stabilendone un punteggio di difficoltà chiamato "livello di offuscamento". Questo rappresenta la difficoltà di lettura e riconoscimento dell'algoritmo implementato dall'applicazione; maggiore è il livello, più difficoltà avrà un attaccante ad effettuare il reverse engineering dell'applicazione ed a modificarne gli algoritmi per riadattarli alle proprie esigenze [16].

### 5.3 Analisi dinamica

Mentre l’analisi statica tratta la lettura ed il riconoscimento di pattern all’interno del codice, l’analisi dinamica riguarda il testing e la valutazione del programma eseguendo il contenuto in tempo reale con l’obiettivo di riconoscerne errori durante il suo avvio, analizzandone le prestazioni senza essere a conoscenza di come il software è effettivamente composto [17].

Attraverso il testing automatico del programma in tutti gli scenari per cui è pensato, l’analisi dinamica elimina il bisogno di ricreare artificialmente probabili situazioni in grado di produrre errori, analizzando la memoria interna dell’applicazione ed immettendo automaticamente input sensibili nelle stesse modalità di immissione pensate per l’utente. Vengono così registrati gli stati interessanti dell’applicazione e gli output prodotti, evidenziandone le anomalie attraverso una parte di reasoning senza il bisogno della supervisione dello sviluppatore.

Lo studio a runtime, oltre ad individuarne vulnerabilità altrimenti nascoste a livello di codice o considerate altrimenti falsi negativi, può essere eseguito anche senza avere accesso diretto al codice sorgente dell’applicazione. Quest’approccio non è però privo di difetti: il limite principale di quest’analisi è quello di stimolare automaticamente l’applicazione in modalità “black box”, ovvero senza conoscenza a priori dei suoi stati, risultando complicata un’esplorazione esaustiva dell’applicazione. La principale conseguenza è quella di una falsa sensazione di completezza nella parte di testing, oltre al grande numero di falsi positivi e falsi negativi prodotti e senza la possibilità di rintracciare gli errori nell’esatta posizione all’interno del codice sorgente.

Le principali informazioni di interesse per l’analisi dinamica del dispositivo si possono riassumere in quattro punti:

- Comunicazioni network in entrata ed in uscita

Al fine di proteggere le informazioni trasmesse dal dispositivo, risulta necessario che le comunicazioni vengano effettuate tramite protocolli SSL, in modo che il traffico possa essere letto chiaramente solamente dall’emittente e dal ricevente. Qualora il traffico venga trasmesso in chiaro (come ad esempio usando il protocollo http), non vi è nulla che impedisca a terze parti in ascolto (o coinvolte nelle comunicazioni) di leggere i messaggi trasmessi, ottenendo o nel peggio dei casi compromettendo dati sensibili.

Direzionando il traffico del dispositivo attraverso un proxy, è possibile osservare i dati trasmessi dal dispositivo. Senza certificati, un proxy riesce a simulare un attacco “man in the middle”, dove un generico attaccante si mette in ascolto del flusso di dati del dispositivo. Ne segue che tutte le intercettazioni in chiaro e lettura dei relativi payload da parte del proxy, risultano in una chiara superficie di attacco a cui il dispositivo potrebbe essere soggetto una volta lanciato sul mercato.

Molti proxy, quando il messaggio inviato risulta illeggibile, rifiutano di continuare la trasmissione, scartando il suddetto messaggio. Qualora si utilizzi un protocollo di comunicazione criptato (ad esempio https), per riuscire a leggere i dati in chiaro sarà necessario installare i relativi permessi all’interno del dispositivo.

- Metodi richiamati da librerie esterne con relativi input e risultati in uscita

Mentre il codice del programma è scritto dallo sviluppatore, l’utilizzo di librerie esterne potrebbe portare all’esecuzione di comandi o azioni non programmate/-nascoste in alto livello all’utente.

Attraverso la cattura di tali metodi è possibile creare una lista di tutte le azioni effettivamente eseguite dal dispositivo. A grandi linee si riuscirà a generare un report riportante tutte le azioni effettivamente eseguite dal dispositivo, con relativi input e output in questione, informando così lo sviluppatore che le validerà o meno.

- Lettura/scrittura di informazioni in memoria

Non è solo il come i dati del dispositivo vengono trasmessi, ma anche il come questi vengono effettivamente salvati e riportati nella memoria del dispositivo risulta essere di vitale importanza, sia per la corretta anonimizzazione di dati sensibili, sia per una corretta protezione e mantenimento dei dati salvati. Mappando i files interessati dall’applicazione ed analizzando i metodi per la scrittura e lettura delle tuple dei databases, una volta terminata l’applicazione è possibile (sempre avendo a disposizione i privilegi di amministratore) andare a verificare l’effettiva correttezza dei dati memorizzati e delle operazioni di lettura effettuate, con la verifica delle relative impostazioni di sicurezza.

- Utilizzo di Log files

Con il termine “log” si indica la scrittura di un messaggio testuale in un’apposita area di memoria durante l’esecuzione del programma. Questo permette di riportare e memorizzare gli errori, warning o messaggi informativi generati durante l’esecuzione dell’applicazione in modo da poter analizzarne con maggiore precisione quanto sia effettivamente accaduto.

## 5.4 Mobile Security Analysis

Per verificare la sicurezza dell’applicazione, vengono tipicamente combinate le due metodologie di analisi sopra descritte; in questo modo si ottengono informazioni sulla sua struttura, sui permessi richiesti e librerie esterne utilizzate tramite tecniche di analisi statica sul codice dell’applicazione, testandolo successivamente in real time ed ottenendo un resoconto delle operazioni eseguite e dei risultati prodotti tramite analisi dinamica.

<b>ANALISI STATICÀ</b>	<b>ANALISI DINAMICA</b>
Lettura del manifest	Controllo input
Permessi dichiarati	Analisi output
Bontà del codice	Trasmissione dati
API dichiarate	File creati/letti/scritti
Riconoscimento pattern	Scrittura/lettura database
Riconoscimento offuscamento	API utilizzate
Rilevamento Oggetti	Permessi utilizzati
	Operazioni effettuate
	Correttezza memoria RAM

Figura 19: Obiettivi delle analisi combinate

Le analisi effettuate (riportate in figura 19) forniscono una panoramica dell’applicazione specificata, riportando tutte le informazioni interessanti sul singolo funzionamento e

fornendo allo sviluppatore un chiaro quadro della situazione del dispositivo.

#### 5.4.1 OWASP Mobile Security Project

L'*Open Web Application Security Project* (OWASP)<sup>23</sup> è un'organizzazione non affiliata e senza scopo di lucro che si pone l'obiettivo di migliorare la sicurezza software. L'OWASP cura uno specifico progetto dedicato alla sicurezza mobile chiamato "OWASP Mobile Security Project" che rappresenta un punto di riferimento della comunità di sicurezza sia industriale che accademica.

Per aiutare gli sviluppatori a rendere le proprie applicazioni più sicure, l'OWASP ha definito una serie di best practice di sicurezza e di controlli da effettuare nelle applicazioni mobile, pubblicando due apposite guide, la *Mobile Security Testing Guide* (MSTG [13]) e il *Mobile Application Security Verification Standard* (MASVS [18])

In queste guide sono riportate le pratiche di programmazione da seguire, le tecniche di analisi e gli strumenti necessari per garantire la sicurezza in vari punti caratteristici dell'applicazione, come:

- **Conservazione dei dati**

Viene testata la sicurezza dell'archivio dati, verificando se terze parti possono averne accesso o se informazioni sensibili sono diffuse senza autorizzazione all'esterno, non sono state completamente eliminate dalla memoria volatile del sistema o se sono presenti degli exploit per recuperarne il contenuto in modi più indiretti (ad esempio attraverso la cache della tastiera)

- **Tecniche crittografiche**

Vengono visionati gli algoritmi di criptografia utilizzati, evitando tecniche obsolete o superate, verificando inoltre che il generatore di numeri casuali (indispensabile per la maggior parte di queste tecniche) produca risultati non prevedibili o replicabili. Viene inoltre sollecitato l'utilizzo di Android Keystore per conservare le chiavi crittografiche in un ambiente protetto e non accessibile dall'esterno.

---

<sup>23</sup> <https://www.owasp.org>

- **Autenticazione locale**

Viene testato il sistema di conferma delle credenziali utilizzato, sia nella forma di password/PIN, sia nella forma di conferma biometrica, al fine di procedere con un uso corretto e non manomissibile di tali operazioni.

- **Comunicazioni network**

Vengono testate le comunicazioni del dispositivo, confermando la verifica dell'identità dell'endpoint e testando la sicurezza del protocollo di comunicazione utilizzato. Viene testato infine il pinning dei certificati e la loro corretta memorizzazione.

- **Interfaccia utente, permessi, struttura interna e persistenza degli oggetti**

Vengono studiati e testati i permessi richiesti dall'applicazione, le tecniche di IPC (Inter Process Communication) utilizzate e la loro sicurezza, l'esecuzione di codice JavaScript, la persistenza degli oggetti Java e la possibilità di esporre le loro informazioni attraverso l'analisi della memoria volatile o tramite l'uso di librerie non sicure.

- **Qualità del codice e debugging**

Viene verificato che l'applicazione sia correttamente firmata e che il codice non contenga tracce che potrebbero fornire accesso alle operazioni di sicurezza effettuate (come password riportate all'interno di commenti). Viene verificato che i messaggi verbali di debugging vengano completamente eliminati dal codice, e che le impostazioni di sicurezza del dispositivo vengano riattivate al termine del test dell'applicazione. Vengono inoltre analizzate le librerie di terze parti, per assicurarsi che non abbiano all'interno alcuna vulnerabilità sfruttabile da un attaccante.

- **Anti reverse engineering ed offuscamento del codice**

Viene effettuato un reverse engineering dell'applicazione per assicurarsi che tutte le informazioni trovate non possano essere utilizzate per fini malevoli o per ottenere maggiori privilegi non acconsentiti all'utente. Viene inoltre verifica-

to l'efficacia dell'offuscamento del codice effettuato, assicurandosi che il codice sorgente dell'applicazione non sia facilmente riproducibile e manomissibile.



Figura 20: Rappresentazione delle analisi eseguite nell'OWASP framework

Un'applicazione in grado di completare con successo tutte le analisi nei rispettivi campi è da considerarsi robusta o “sicura” contro eventuali manomissioni da terze parti. Tutte queste verifiche costituiscono quindi lo scheletro dei moderni strumenti di analisi di sicurezza, attraverso i quali è possibile conoscere a quali superfici di attacco l'applicazione risulta vulnerabile.

### 5.4.2 Approver

Sviluppato da Talos [19] e basato sul framework OWASP, Approver [20] è uno strumento per analizzare la struttura ed il funzionamento di applicazioni mobile per determinarne il loro stato di sicurezza. Vengono utilizzate le tecniche di analisi statica e dinamica precedentemente elencate nella metodologia per verificare la presenza di vulnerabilità e superfici d'attacco, fornendo una panoramica allo sviluppatore sui comportamenti dell'applicazione analizzata e suggerendogli come rimediare a tali problemi.

Data la possibilità di eseguire diverse analisi in parallelo, Approver è sviluppato in una piattaforma a microservizi, gestiti dal modulo centrale “Orchestrator” (figura 21)

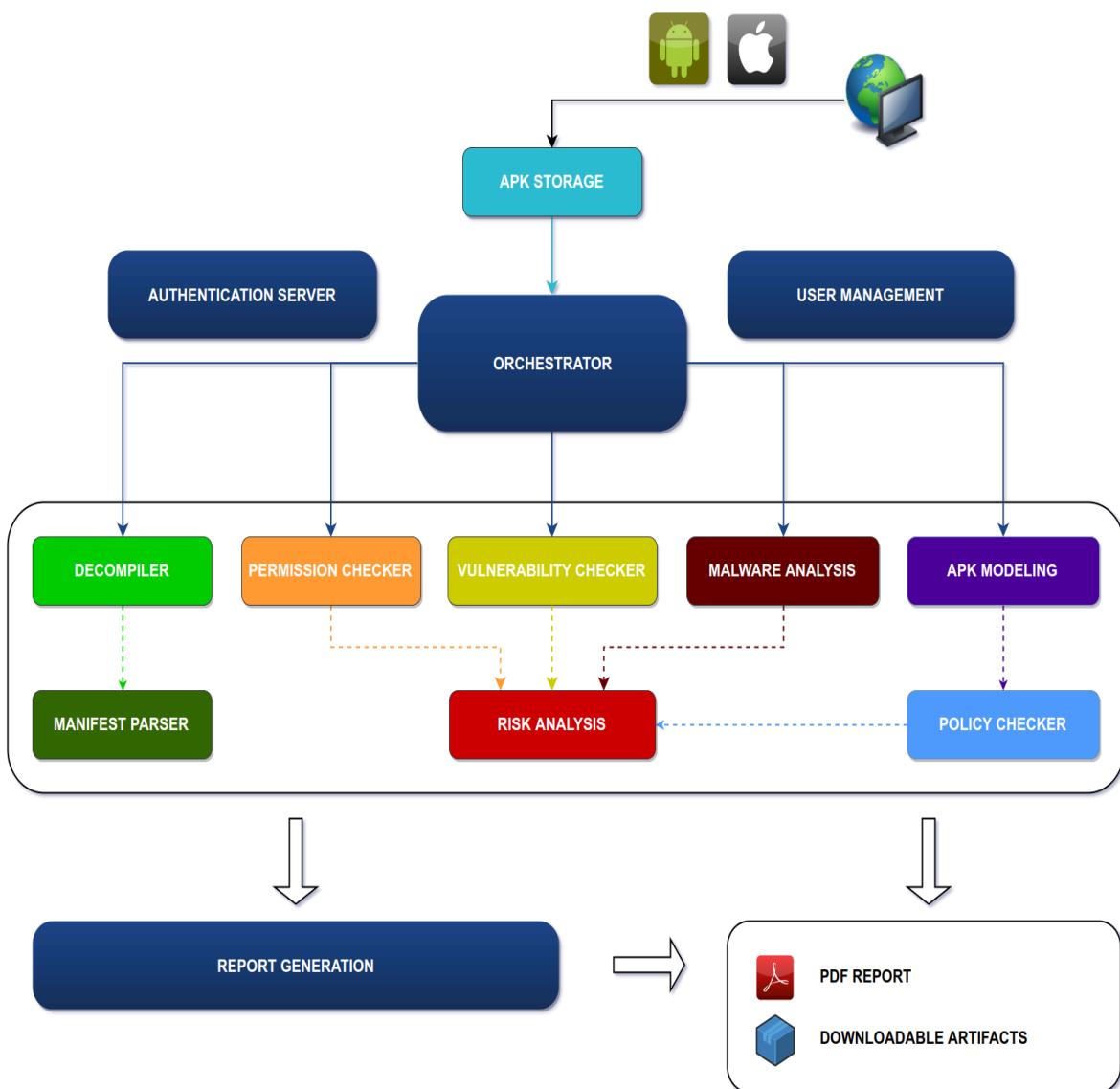


Figura 21: Schema Approver

Ricevuta la copia dell'apk da analizzare, il modulo centrale “Orchestrator” comunicherà con i vari servizi per iniziare ad analizzarla sia staticamente (attraverso la decompilazione del Dalvik bytecode in un linguaggio intermedio chiamato Smali [21], molto simile al codice sorgente Java), sia dinamicamente (attraverso l'installazione dell'applicazione in una macchina virtuale Android).

Al termine delle analisi, i risultati vengono inviati al modulo “Risk Analysis” per valutare le possibili superfici d'attacco rilevate, stabilendone un punteggio di pericolosità. Viene infine steso un report con tutte le informazioni raccolte durante il processo di analisi, attraverso il quale lo sviluppatore potrà conoscere quali settori della sua applicazione necessitano di aggiustamenti.

## 5.5 IoT App Security Analysis

Data l'elevata somiglianza tra il sistema operativo mobile "*Android*" ed il nuovo sistema operativo "*Android Things*", è possibile riadattare il framework OWASP per consentire di conoscere lo stato della sicurezza dei propri devices IoT.

Gli stessi meccanismi di analisi applicabili su applicazioni mobile permettono quindi di conoscere lo stato della sicurezza di un'applicazione AT. Le differenze principali si manifestano a livello di API richiamabili: un esempio è rappresentato dal fatto che Android Things non presenta di per sé un'interfaccia utente standard; tutte le analisi ad essa relativa non sono quindi applicabili in questo contesto. Inoltre sono rivisitati i meccanismi di autenticazione utente, spostati da uno scenario locale (password/ pin/ riconoscimento biometrico) ad uno scenario di dispositivi multipli interconnessi tra loro.

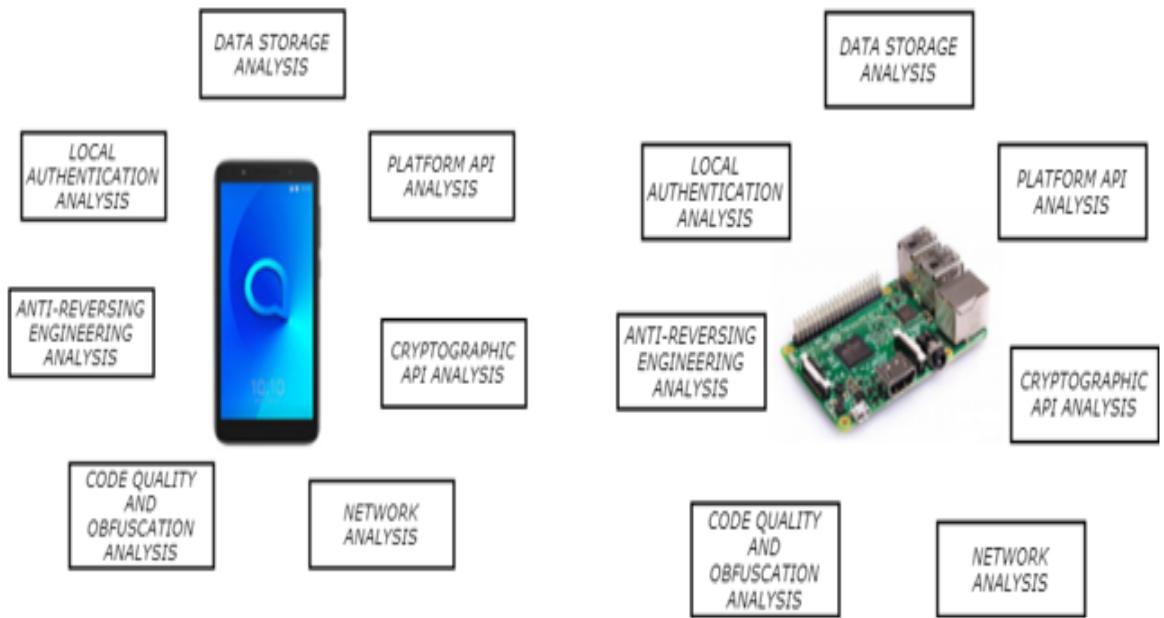


Figura 22: Analisi OWASP eseguite su Android ed Android Things

Ma è proprio l'interazione tra diversi dispositivi che pone una forte limitazione nelle analisi di sicurezza IoT: procedendo con la sola rivisitazione del framework OWASP si riesce a valutare lo stato di sicurezza della singola app, ma non si ha l'opportunità di analizzarne il comportamento in relazione ai dispositivi circostanti. Lo scopo ultimo dei dispositivi IoT non è quello di fornire un'elaborazione indipendente dei dati, ma è quello di rimanere connessi ad un'infrastruttura più grande, ricevendo e trasmettendo nuovi dati per poterli analizzare e rielaborare in un servizio molto più complesso.

L'esecuzione di una determinato caso d'uso può richiedere quindi un carico di lavoro ridistribuito su dispositivi multipli, dove questi condividono tra loro lo stato, informazioni, comandi e punti di accesso; si viene quindi a generare un **ecosistema**, una rete operante al di sopra dei singoli dispositivi da cui è composta, mantenendone le sue parti indipendenti ma al tempo stesso altamente interconnesse tra loro, fornendo all'utente un'astrazione di quanto effettivamente succede attraverso un'interfaccia di accesso mobile e riuscendo a generare risultati che vanno ben al di là della somma delle singole azioni.

Ma l'unione di più dispositivi eterogenei tra loro aumenta sensibilmente la complessità per assestarsi lo stato di sicurezza dell'intero sistema: l'analisi indipendente dei singoli

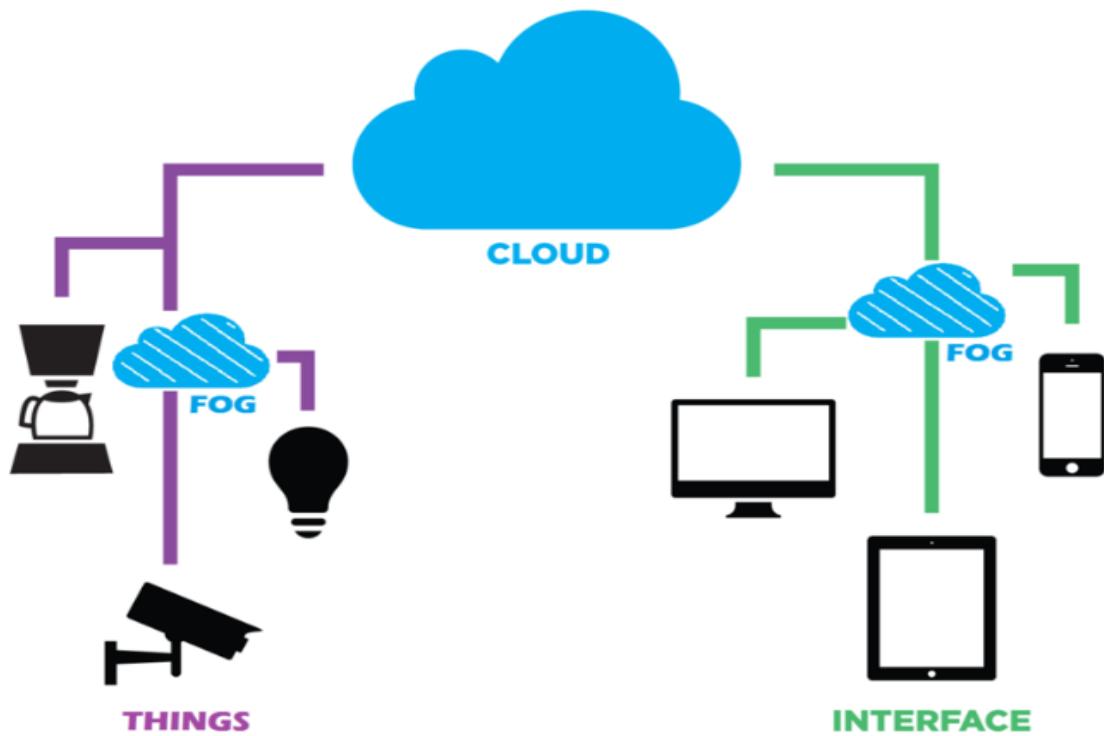


Figura 23: Ecosistema di dispositivi IoT

dispositivi, per quanto utile, non riesce da sola a fornire una panoramica completa sul funzionamento e sullo stato generale dell'ecosistema; un device IoT programmato per mantenere la sua temperatura interna in un determinato range potrebbe lavorare perfettamente qualora esegua autonomamente i compiti assegnati, ma sviluppare comportamenti inattesi qualora questo entri in contatto con informazioni anomale ma apparentemente innocue provenienti da altri dispositivi. L'unione di più dispositivi considerati sicuri nella loro individualità non garantisce una totale sicurezza dell'infrastruttura generale.

Dato che le applicazioni mobile, target ideali per l'implementazione d'interfacce d'accesso agli ecosistemi IoT, rappresentano anche il principale vettore d'attacco per interferire con i devices, durante questo lavoro di tesi è stato proposto e sviluppato una nuova metodologia di analisi in grado di valutare lo stato di sicurezza sia dei singoli dispositivi, sia dell'ecosistema nella sua interezza; osservando le reazioni sotto questi due punti di vista, è possibile ricondurre le superfici di attacco rilevate dalle analisi alla loro causa scatenante, per determinare così il caso d'uso vulnerabile.

## 6 Metodologia

---

### 6.1 Introduzione

Benché siano disponibili online strumenti per analizzare la sicurezza delle applicazioni mobile (riadattabili per l'analisi di applicazioni IoT), attualmente non esistono strumenti in grado di testare l'interazione in tempo reale tra una generica interfaccia di controllo utente (rappresentata qui da un'applicazione mobile) ed un device IoT online. L'obiettivo finale di questa tesi consiste nella definizione di una metodologia in grado di eseguire un'analisi di sicurezza dell'ecosistema di applicazioni mobile-IoT. L'analisi dell'ecosistema, oltre a considerare i singoli ambienti (device + applicazione) singolarmente, ha l'obiettivo di valutare la loro interazione, in modo da riuscire a collegare i risultati ottenuti dalle singole applicazioni e ricreare quanto avvenuto a livello di ecosistema.

La metodologia sviluppata in questo lavoro di tesi si basa quindi sulla cultura già consolidata di tecniche di analisi statica e di analisi dinamica del mondo mobile, estese e riadattate per il mondo IoT; dato che non esistono software in grado di compiere attualmente simili analisi congiunte, si sono sviluppate delle tecniche in grado di poter supportare osservazioni multiple di dispositivi, evidenziando le anomalie di sicurezza rilevate e permettendo una correlazione tra queste ed il relativo segnale scatenante.

A seguito della definizione di tale metologia, è stato sviluppato anche un framework di analisi per applicazioni Android e Android Thigs, chiamato **AppIoTTE (App-IoT Testing Environment)**, come verrà descritto nei capitoli successivi.

## 6.2 Analisi dell'Ecosistema Applicativo Mobile-IoT

La metodologia di analisi dell'ecosistema mobile-iot ha l'obiettivo di analizzare come le singole parti che compongono l'ecosistema interagiscono tra loro, stimolando le applicazioni che costituiscono i diversi punti dell'infrastruttura ed osservando come queste interazioni si manifestano sui dispositivi e si ripercuotano negli aspetti della loro sicurezza.

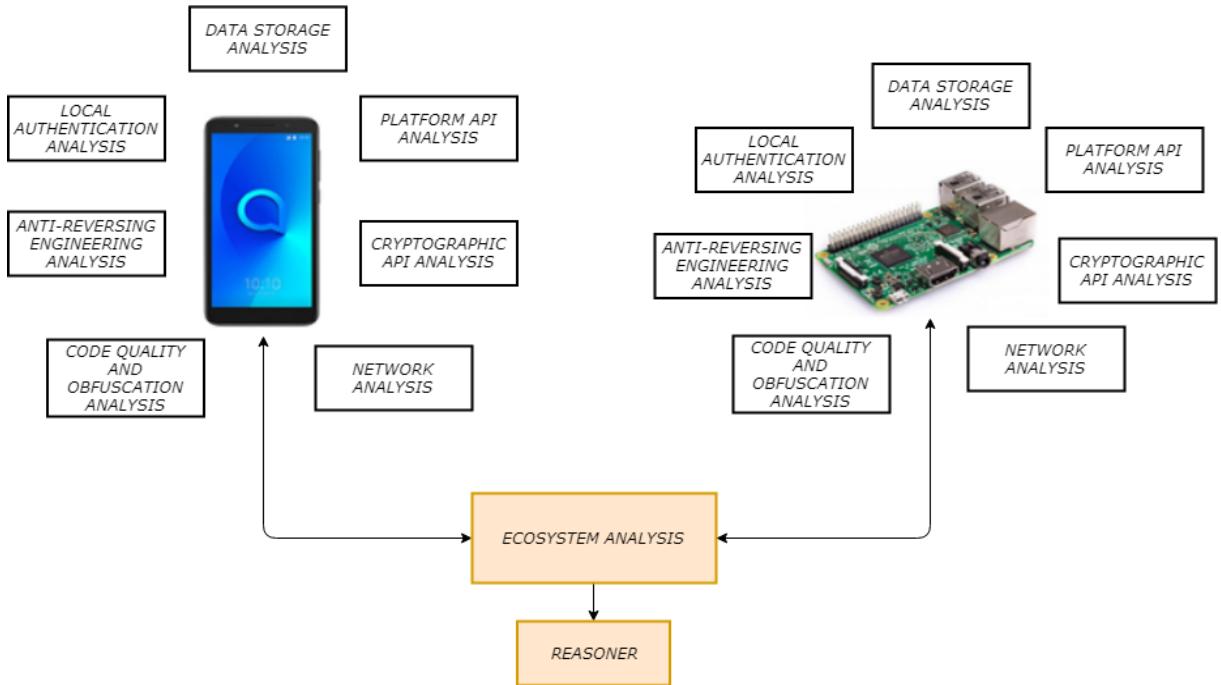


Figura 24: Unione delle analisi dei singoli dispositivi con l'ecosistema

La metodologia è formata da una serie di moduli di analisi statica e dinamica, derivati dagli stessi controlli OWASP per l'analisi individuale dei dispositivi, che consentono di analizzare proprietà di sicurezza come:

- Trasmissioni cifrate tra dispositivi
- Controllo delle credenziali dell'utente e dell'interfaccia di accesso
- Librerie utilizzate autenticate e sicure
- Eventi generati durante l'esecuzione del sistema

Il suo punto di forza è costituito dalla parte di analisi integrata dell'ecosistema: in questa fase di analisi dinamica sia l'applicazione mobile che l'applicazione IoT vengono eseguite all'interno di un ambiente di test integrato, dove tutti i dispositivi interessati nel processo sono dispiegati ed online, stimolandoli in maniera congiunta per osservarne gli effetti sull'intero sistema; vengono così confrontati gli input inviati da un dispositivo di **controllo** (impulsi inviati generalmente dagli utenti e rappresentati in esempio da un'applicazione smartphone) con gli effetti che questo scatena nei dispositivi IoT **sotto osservazione** (in figura il dispositivo IoT). Questi effetti vengono registrati ed analizzati per determinare se l'elaborazione di questo input presenta o meno vulnerabilità nella sicurezza di altri devices. L'ecosystem analysis permette in questo modo non solo di verificare lo stato di sicurezza a runtime delle singole app ma anche di individuare vulnerabilità complesse, derivanti dall'interazione dei due sistemi come vedremo nei capitoli successivi.

### 6.2.1 Funzionamento generale

L'analisi di sicurezza dell'ecosistema viene eseguita in due fasi distinte: I) fase individuale e II) fase combinata.

Nella fase individuale le applicazioni mobile ed IoT vengono analizzate staticamente, quindi vengono caricate nell'ambiente di test e vengono stimolate dinamicamente in maniera separata e isolate. Questa parte permette lo studio del comportamento di sicurezza delle due applicazioni in maniera indipendente dal contesto.

Una volta analizzati in maniera indipendente i vari dispositivi, si procede con la fase combinata, rimuovendo l'isolamento e abilitando tutte le comunicazioni tra le due applicazioni poste rispettivamente sul dispositivo mobile e su quello IoT. A questo punto il sistema di stimolazione dell'ecosistema provvede a stimolare automaticamente lo smartphone attraverso la sua apposita interfaccia utente (questo dispositivo viene considerato il **controllore** del processo); viene quindi osservata la reazione dell'intero ecosistema, registrando il comportamento e tutte le azioni avvenute nel secondo dispositivo (chiamato quindi dispositivo di **osservazione**)

Terminate le analisi, vengono confrontati i risultati con quanto ottenuto in precedenza dal reasoner interno al modulo "Ecosystem analysis", con il compito di separare quanto

osservato in seguito allo stimolo del controller, da quanto osservato autonomamente nel secondo dispositivo. Una volta stilate queste corrispondenze, il reasoner proseguirà con l'identificare tutte le anomalie, cercandone di identificare i problemi, le possibili cause sia interne che nell'input del controllore, fornendone se possibile una soluzione.

Il flusso di funzionamento generale delle analisi di un dispositivo di controllo mobile ed un dispositivo di osservazione IoT è descritto in seguito.

---

**Algorithm 1:** App-IoT Testing Environment

---

**Input** : phone, device  
**Input** : phoneApp, iotApp  
**Input** : proxyAddress, proxyCertificate  
**Output**: data, correlationList

```
1 phoneApp = installAPK (phone, phoneApp);
2 deviceApp = installAPK(device, iotApp);
3 proxy = startProxy(proxyAddress);
4 installProxyCert (phone, proxyCertificate);
5 installProxyCert (device, proxyCertificate);
6 installProbes(phone);
7 installProbes(device);

8 phone.startMonitoring();
9 phone.startApplication( phoneApp );
10 phoneIsolatedAnalysisResult = phone.getAnalysisResults();
11 phone.endMonitoring();
12 phone.uninstallAPK( phoneApp );

13 device.startMonitoring();
14 device.startApplication( deviceApp );
15 deviceIsolatedAnalysisResult = device.getAnalysisResults();
16 device.endMonitoring();
17 device.uninstallAPK( deviceApp );

18 phoneApp = installAPK (phone, phoneApp);
19 deviceApp = installAPK(device, iotApp);
20 phone.prepareEcosystemAnalysis();
21 device.prepareEcosystemAnalysis();

22 while not phone.readyToStart() or not device.readyToStart() do
23   | wait( 10 );
24 end

25 phone.startMonitoring();
26 device.startMonitoring();
27 phone.startApplication( phoneApp );
28 device.startApplication( deviceApp );
29 phone.sendAutomaticInputs();

30 phoneAnalysisResult = phone.getAnalysisResults();
31 deviceAnalysisResult = device.getAnalysisResults();

32 stimuliReaction= clearFromSelfEvents (deviceIsolatedAnalysisResult, deviceAnalysisResult);
33 listAnomalies = findAnomalies ( stimuliReaction );
34 correlationList = [];
35 foreach anomaly in listAnomalies do
36   | causedBy = findRelativePhoneInput ( phoneAnalysisResult.getInputs(), anomaly);
37   | correlationList.add(anomaly, causedBy);
38 end
39 return (phoneIsolatedAnalysisResult, deviceIsolatedAnalysisResult, correlationList);
```

---

Terminate le operazioni di stimolazione ed osservazione dell’ecosistema, ci si pone l’obiettivo di correlare i risultati ottenuti dalle applicazioni mobile e IoT in maniera da individuare delle relazioni di causalità tra comando generato e vulnerabilità riscontrata. Per questo motivo il framework, tramite un modulo di reasoning, dovrà quindi analizzare i dati raccolti e determinare una correlazione tra l’anomalia scatenata e lo stimolo inviato.

In questa metodologia sono stati studiati e proposti differenti algoritmi per raggiungere un tale risultato, che verranno descritti nel prosegoo di questo capitolo.

### 6.2.2 Algoritmo di correlazione "Naive"

L’algoritmo “naive” cerca di stabilire una corrispondenza tra lo stimolo e le rispettive anomalie, basandosi su due principi base:

- I segnali inviati dalla board vengono processati in un tempo tale da essere completati entro l’invio del segnale successivo.
- Si riesce ad identificare con precisione ogni segnale inviato dallo stimolo e le anomalie generate da esso vengono completamente separate dalle anomalie generate dai processi interni ed indipendenti della board.

Questi due punti rappresentano una situazione ideale di monitoraggio, dove si riconoscono con chiarezza i segnali inviati in seguito allo stimolo dello smartphone rispetto a quelli generati autonomamente dai processi interni dell’applicazione IoT.

Prese in ingresso le liste dei segnali inviati dal dispositivo mobile e delle anomalie generate dal dispositivo IoT, l’algoritmo per stabilire la corrispondenza segnale-anomalia viene riassunto nella pagina successiva.

---

**Algorithm 2:** Algoritmo "Naive"

---

**Input** : signalList, anomaliesList  
**Output**: correlationList

```
1 position = 0;
2 signal = signalList[position];
3 pastSignal = null;
4 listOfCorrelations = new List();
5 foreach anomaly in anomaliesList do
6     findingSignal = true;
7     while findingSignal is true do
8         //Confronto l'orario dell'anomalia con quello del segnale
9         timeDiff = getTime(anomaly) - getTime(signal);
10        if timeDiff < 0 then
11            //il segnale è arrivato dopo l'anomalia. La correlo con il segnale precedente
12            findingSignal = false;
13        end
14        else
15            Position = position + 1;
16            pastSignal = signal;
17            if (position < signalList.length() then
18                signal = signalList[ position ];
19            end
20            else
21                //Letta l'intera lista dei segnali
22                //Correlo l'anomalia con l'ultimo segnale rilevato
23                findingSignal = false;
24            end
25        end
26    end
27 end
```

---

Quest’algoritmo scorre la lista delle anomalie e la lista dei segnali in parallelo, andando per ogni anomalia X a verificare se il segnale Y è stato inviato precedentemente o successivamente. Una volta identificato un segnale Y inviato in un orario maggiore rispetto all’anomalia X, il segnale Y-1 viene identificato come sua causa scatenante e si memorizza la coppia (X,Y-1)

Di seguito si analizzano le possibili situazioni che si potrebbero verificare durante le analisi dell’ambiente IoT.

Questi scenari sono composti da:

- Segnali dal dispositivo di controllo mobile

In seguito alla stimolazione del dispositivo di controllo, è possibile che questo tenti di comunicare con il mondo esterno tramite la rete network a cui è connessa o tramite wifi. Queste comunicazioni vengono intercettate, riportate in una lista, ed ognuna di esse viene generalmente riferita come "segnale" inviato all'esterno. Rientrano in questa categoria i segnali A e B

- Anomalie dal dispositivo di osservazione IoT

Una volta ricevuto dal dispositivo di osservazione, è possibile che il segnale porti all’esecuzione di vari processi, i quali vengono analizzati e viene stilata una lista delle anomalie di sicurezza prodotte.

Negli scenari, le anomalie a1,a2 sono causate dall’invio del segnale A; le anomalie b1,b2,b3 sono causate dall’invio del segnale B

- Disturbi nel dispositivo di controllo (o rumori)

Sono segnali inviati dal dispositivo di controllo, ma non indirizzati al dispositivo di osservazione. La loro presenza costituisce un disturbo interno al processo, in quanto possono portare a false correlazioni.

Rientrano in questa categoria i disturbi E

- Disturbi nel dispositivo di osservazione (o rumori)

Sono anomalie generate da processi autonomi del dispositivo e non causati da parti esterne. Rappresentano il mancato filtraggio tra quanto osservato nella fase "offline" e nella fase "online" delle analisi del dispositivo. Vengono chiamati "disturbi" in quanto, sebbene rappresentino delle anomalie di sicurezza, sono estranee al processo di correlazione e riportate per errore dai moduli di filtraggio.

Rientrano in questa categoria i disturbi e1

### SCENARIO 1: ASSENZA DI RUMORE

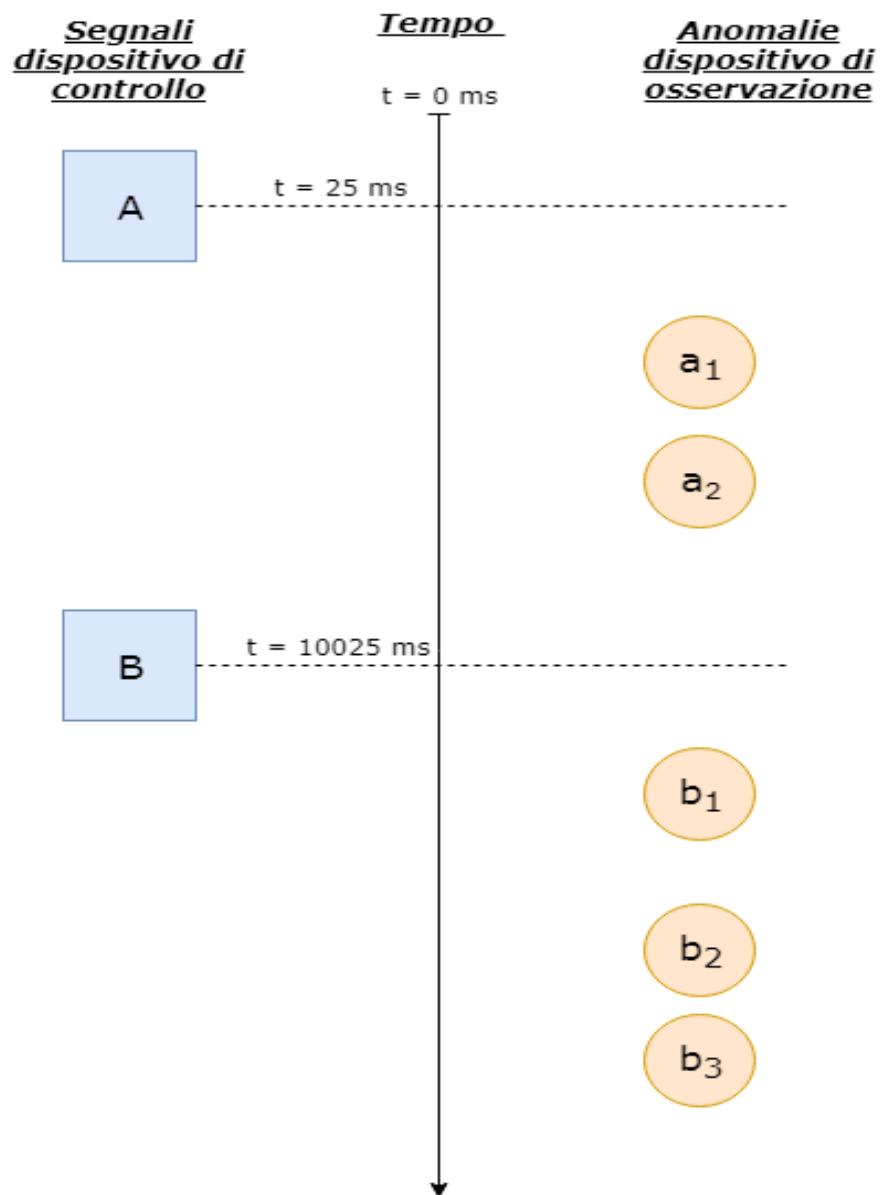


Figura 25: Scenario 1: algoritmo naive senza disturbi

Il primo scenario rappresenta una situazione ideale, in assenza di rumore sia all'interno del dispositivo di controllo, sia all'interno del dispositivo di osservazione. Il dispositivo di controllo è stato soggetto allo stimolo A, e ciò ha causato l'invio di un segnale al dispositivo di osservazione; per una latenza nella connessione molto bassa ed un costo computazionale ridotto per l'elaborazione della richiesta, il dispositivo di osservazione è riuscito a completare le operazioni richieste in un intervallo di tempo ridotto, ma effettuando operazioni a1 e a2 che potrebbero ledere alla sicurezza del dispositivo. La finestra di tempo relativa alla stimolazione B può quindi concentrarsi sulla cattura delle proprie anomalie b1, b2 e b3, senza preoccuparsi di quanto avvenuto in precedenza.

#### RISULTATI DELL'ALGORITMO NAIVE – SCENARIO 1

SEGNALE A: generate anomalie a1, a2

SEGNALE B: generate anomalie b1, b2, b3

## SCENARIO 2: PRESENZA DI RUMORE

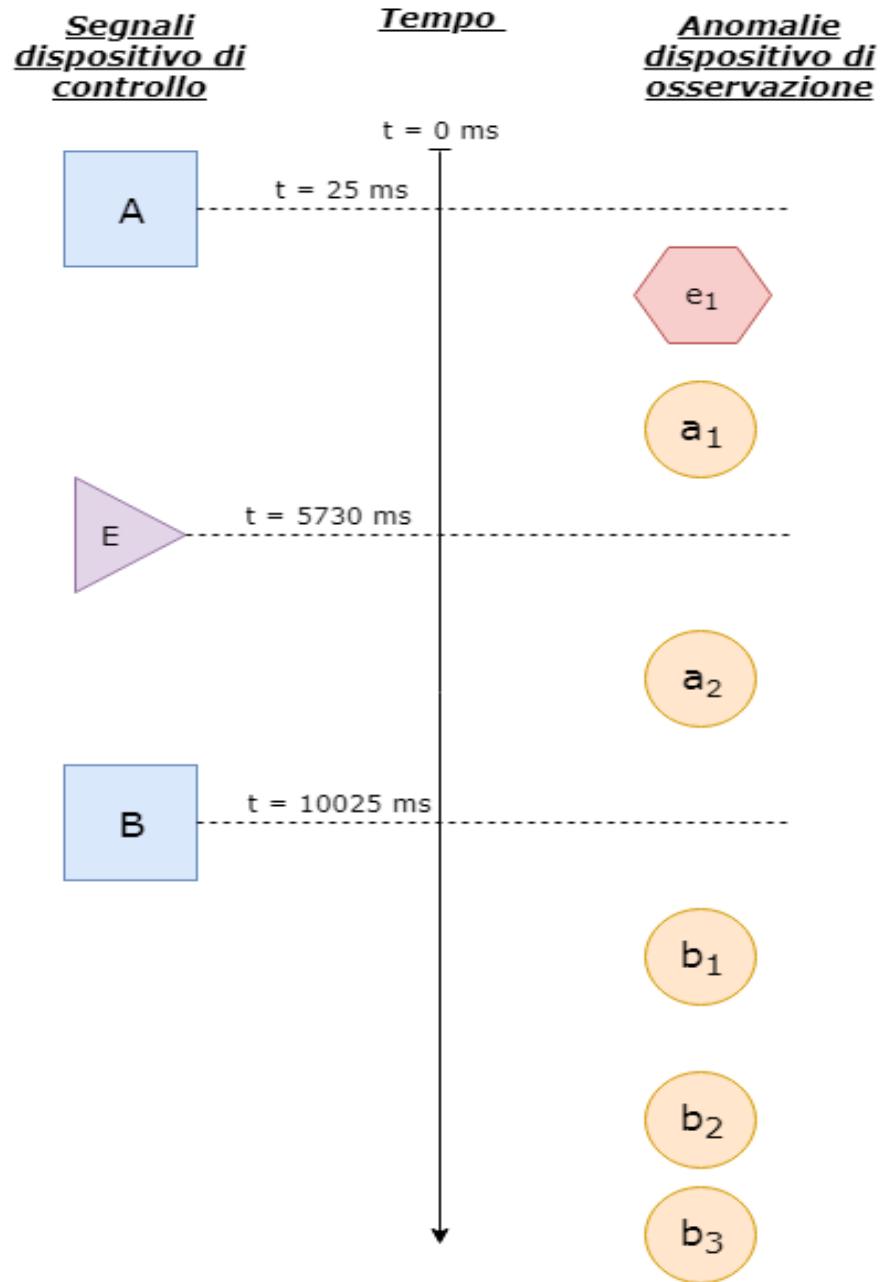


Figura 26: Scenario 2: algoritmo naive con distubi

Nel secondo scenario si considera la presenza di rumore, sia tra i segnali inviati dal dispositivo di controllo (comunicazioni con terze parti non interessate nel processo), che tra le anomalie rilevate nel dispositivo (esecuzione di codice non rilevato durante le precedenti operazioni di osservazione). Già in questo scenario si può notare come l'algoritmo naïve non riesca a riconoscere gli stimoli effettuati dai segnali autonomi inviati. L'algoritmo affiderà quindi al rumore “E” la responsabilità dell'anomalia a2, verificatasi durante lo svolgimento di codice pertinente allo stimolo A.

#### RISULTATI DELL'ALGORITMO NAIVE, SCENARIO 2

SEGNALE A: generate anomalie e1, a1

SEGNALE E: generate anomalie a2

SEGNALE B: generate anomalie b1, b2, b3

SCENARIO 3: PRESENZA DI RUMORE E RITARDI NELL'ESECUZIONE DEI PROCESSI

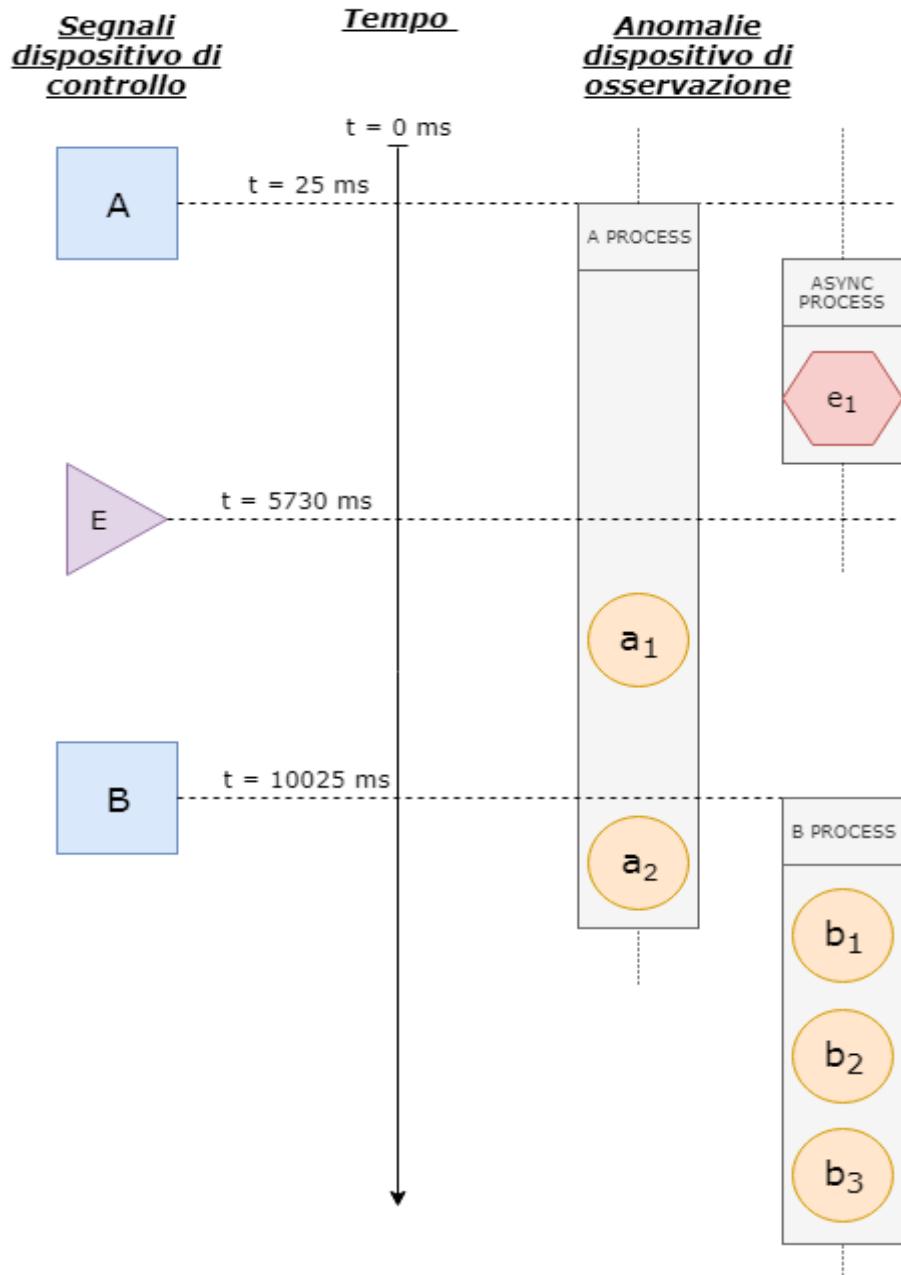


Figura 27: Scenario 3: algoritmo naive con distubi e ritardi

Come nel secondo scenario, sono presenti rumori sia durante la stimolazione, sia durante l’elaborazione degli stimoli. Ma questa volta l’elaborazione del segnale A è stata ritardata per motivi sconosciuti, o ha richiesto più tempo rispetto alla frequenza di stimolazione del dispositivo di controllo. L’algoritmo naive non riesce ad elaborare un simile scenario, e fornirà come causa scatenante dell’anomalia lo stimolo più vicino.

### RISULTATI DELL’ALGORITMO NAIVE, SCENARIO 3

SEGNALE A: generate anomalie e1

SEGNALE E: generate anomalie a1

SEGNALE B: generate anomalie a2, b1, b2, b3

L’algoritmo naive riesce ad effettuare una corretta correlazione solamente qualora non si verifichino segnali/processi autonomi precedentemente non rilevati nei vari dispositivi (valutati come disturbi). Per riuscire a comprendere applicazioni che potrebbero richiedere tempi di elaborazione più lunghi o comunicazioni con terze parti molto frequenti, è necessario adottare nuove strategie.

#### 6.2.3 Algoritmo di correlazione “Repetitions”

Per poter superare le limitazioni dell’algoritmo naive, nell’ambito della tesi è stato definito un nuovo algoritmo, denominato “Repetitions”, al fine di riuscire a determinare una forte correlazione tra anomalie e segnali anche in presenza di forti disturbi. L’idea di base è quella di stimolare l’ecosistema e raccogliere quanti più dati possibili. Una volta rilevate delle anomalie nella sicurezza, si effettua una nuova installazione pulita dell’applicazione da testare nel dispositivo di osservazione, e si ripete l’ultima stimolazione effettuata; i risultati ottenuti vengono quindi confrontati con quelli rilevati in precedenza, stabilendo una correlazione basandosi sulle anomalie che si sono ripetute; ottenuti i risultati, questi vengono filtrati dalla lista delle anomalie, per non poter più venire correlati in seguito, procedendo con l’analisi del segnale successivo.

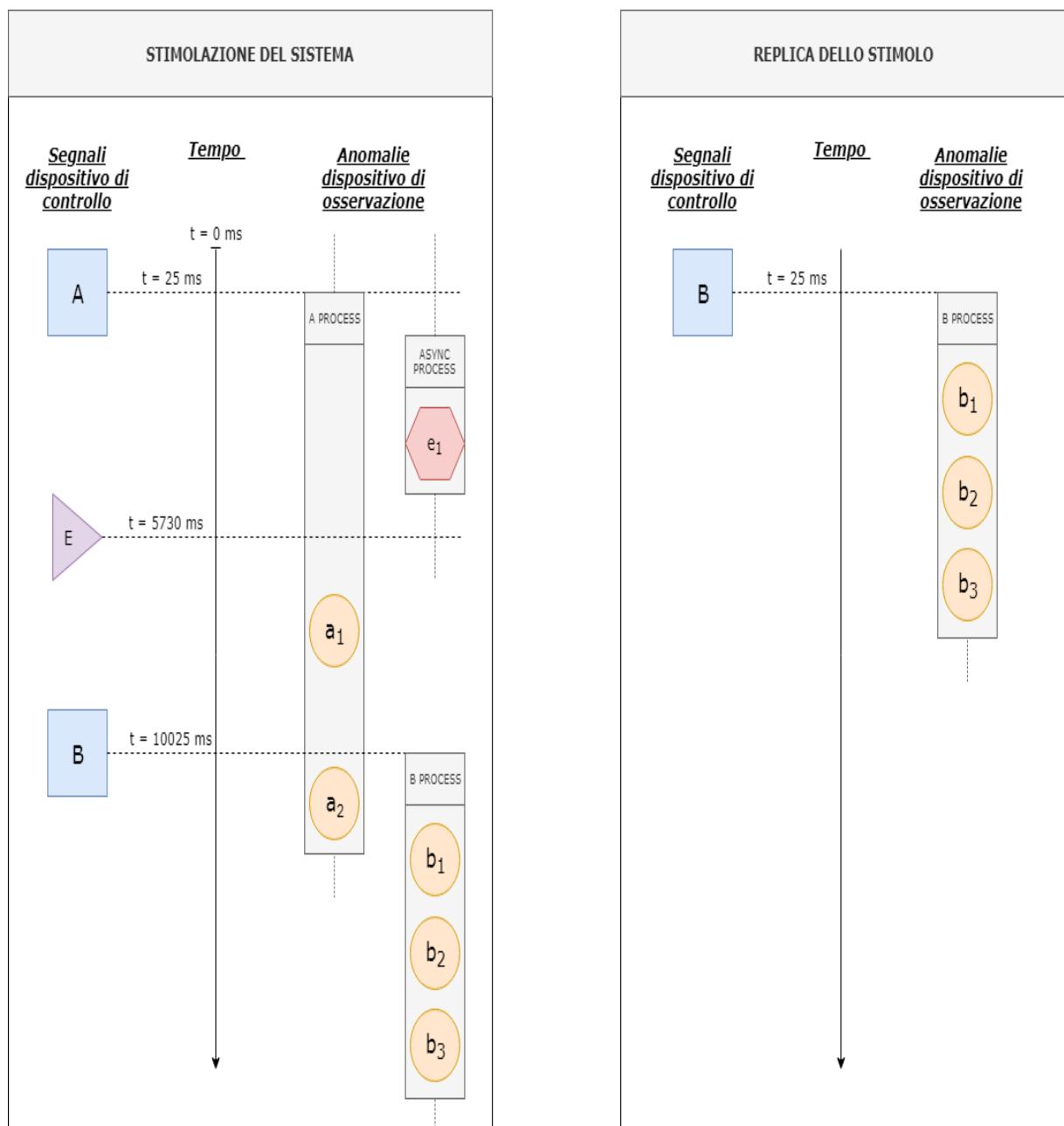


Figura 28: Analisi dell'ecosistema, con ripetizione del segnale B (destra)

## RISULTATI DELL'ALGORITMO REPETITIONS SUL SEGNALE B

SEGNALE B(1): generate anomalie a2, b1, b2, b3

SEGNALE B(2): generate anomalie b1, b2, b3

Tra i due casi, il segnale B risulta sempre il fattore scatenante delle anomalie b1, b2, b3

Anche durante la ripetizione dello stimolo potrebbero venire rilevati dei disturbi; a meno che questi sviluppino le medesime anomalie verificate durante la prima osservazione, questo procedimento porta alla loro eliminazione dalla lista delle correlazioni con il segnale.

Tuttavia, uno dei punti negativi di quest'algoritmo è che non sempre è possibile replicare quanto inviato senza prima ripetere quanto accaduto prima: basti solo pensare al meccanismo di log-off ad un sistema informatico: senza prima aver effettuato un login, non è possibile replicare tale azione. Considerando quanto studiato prima, per intraprendere e replicare le azioni scatenate dal segnale B, potrebbe essere necessario analizzare nuovamente il segnale A; l'analisi di un segnale in maniera indipendente dal contesto potrebbe non essere sempre possibile.

Senza conoscere il contesto, per poter osservare alcuni segnali una seconda volta potrebbe essere necessario studiare le combinazioni dei segnali inviati precedentemente, allungando così di molto il processo di testing e potenzialmente arrivando a replicare l'esatta situazione verificatasi nello scenario 1, vanificando gli sforzi fatti.

Lo si è quindi riadattato per permettere un'analisi sulla ripetizione dei segnali in una stessa sessione di analisi senza dover interrompere le stimolazioni eseguite, dotandolo di una logica interna in grado di escludere dal processo di correlazione anomalie e segnali di disturbo.

---

**Algorithm 3:** Algoritmo "Naive"

---

**Input** : signalList, anomaliesList  
**Output:** correlationList

```
1 listGroupCorrelations = new List();
2 noMoreGroups = false;
3 while not noMoreGroups do
4     //ottengo il prossimo segnale da correlare
5     signalGroup = findGoodGroupUserSignal( signalList );
6     if signalGroup is not null then
7         relativeAnomaliesList = new relativeAnomaliesList();
8         for signal in signalGroup do
9             time = signal.getTime();
10            relativeAnomalies = getAnomaliesUnderTime (anomaliesList, time);
11            relativeAnomaliesList.add( relativeAnomalies );
12        end
13        anomaliesCausedBySignal = findCommonElements( relativeAnomaliesList );
14        baseSignal = signalGroup.baseSignal();
15        listGroupCorrelations.add( anomaliesCausedBySignal, baseSignal );
16        clearAnomaliesFromList( anomaliesList, anomaliesCausedBySignal );
17        clearSignalFromList( signalList, baseSignal );
18    end
19    else
20        noMoreGroups = true;
21    end
22 end
23 simpleCorrelation = naiveAlgorithm (signalList, anomaliesList);
24 completeCorrelations = [listGroupCorrelations, simpleCorrelation];
25 Return completeCorrelations;
```

---

Di seguito si illustra il funzionamento nello scenario dove A e B rappresentano gli stimoli dell'ecosistema, dove non è stato possibile ripeterli in un modo continuo. E rappresenta un segnale autonomo, mentre F rappresenta un disturbo.

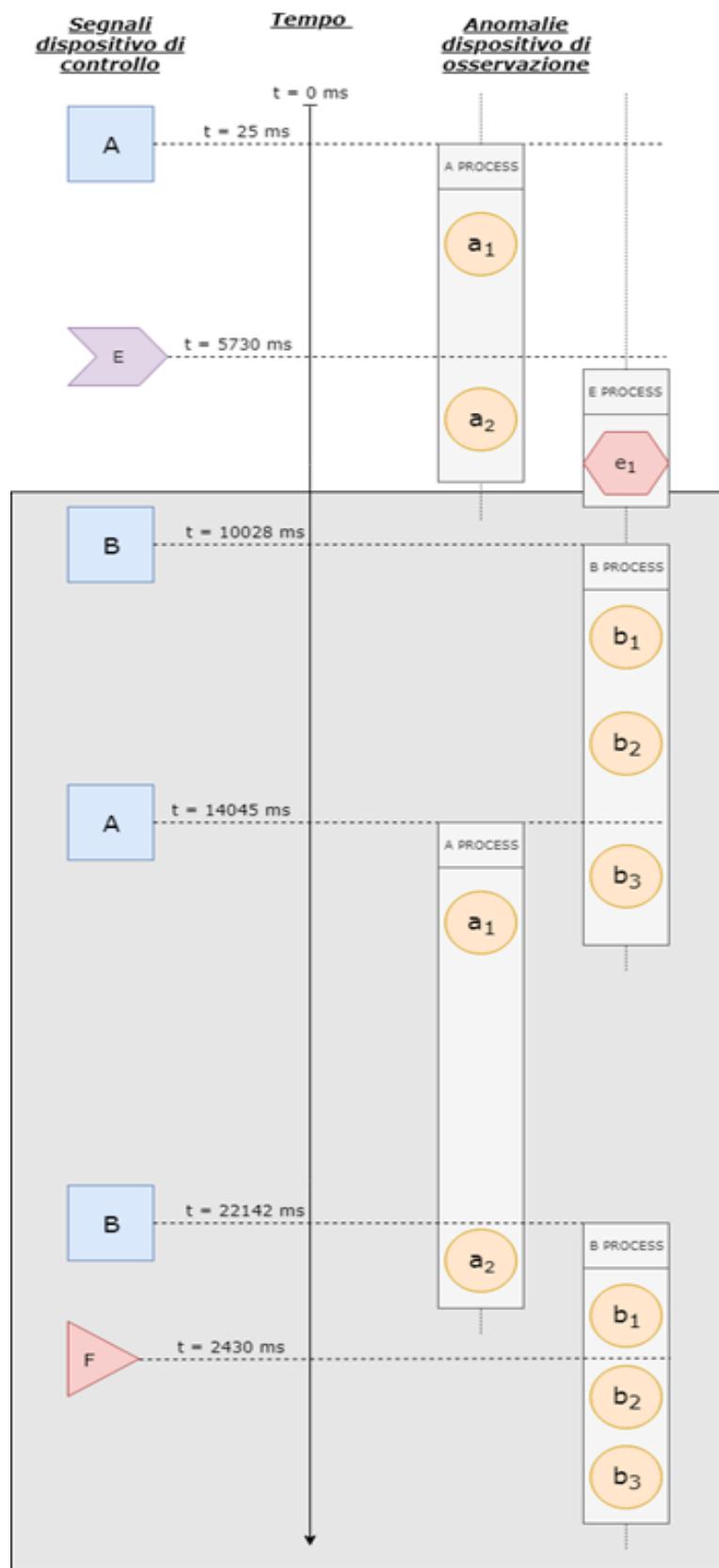


Figura 29: Analisi di uno scenario eterogeneo. In grigio, i segnali B analizzati

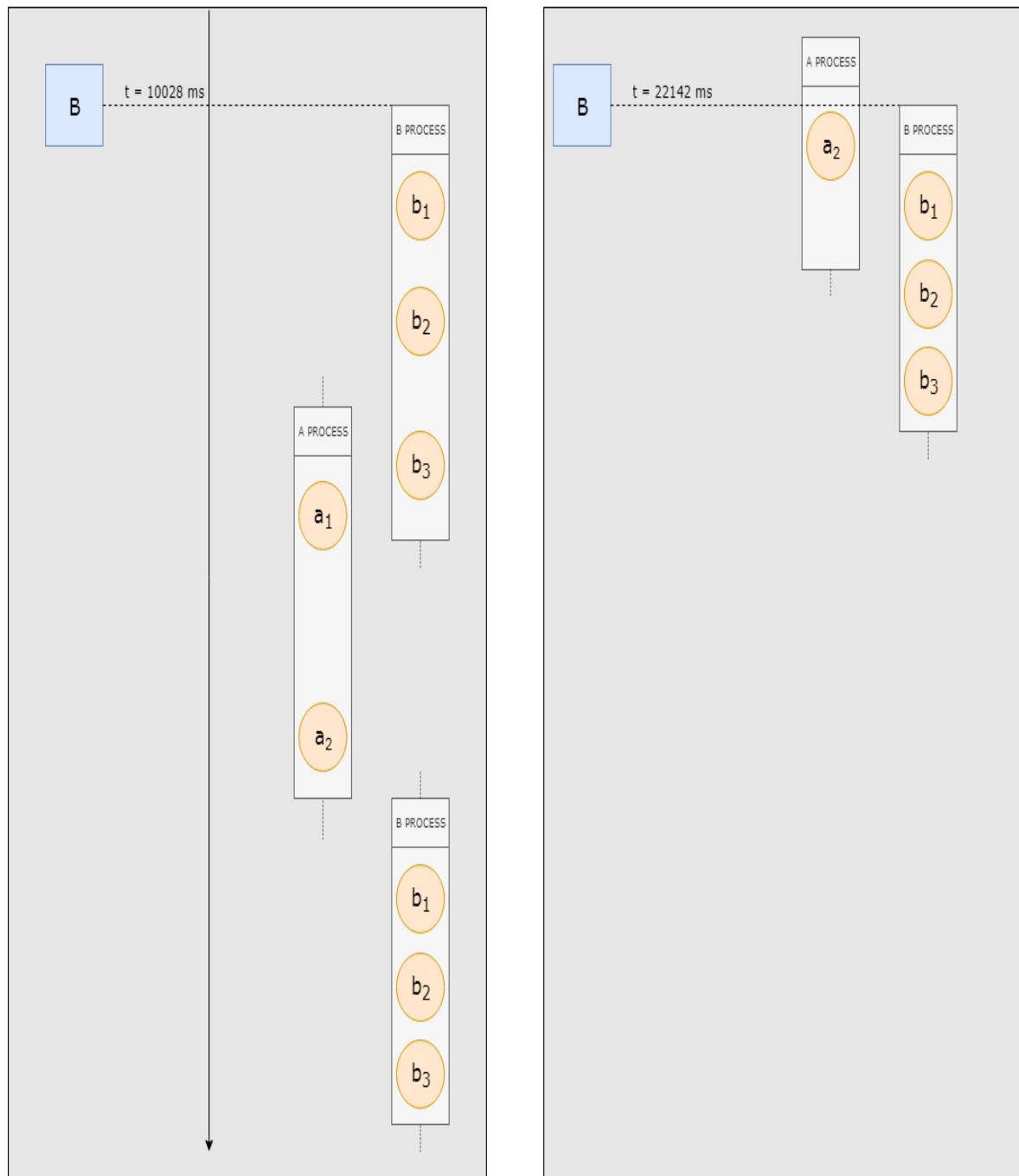


Figura 30: Analisi dei vari segnali B, con le relative anomalie sottostanti

Tenendo in considerazione che il primo stimolo vede sotto di sé anche le risposte del secondo, l'algoritmo è in grado di riconoscere che le due parti hanno in comune le anomalie b1, b2, b3, ripetute più di  $[n/2]+1$  volte, dove n è il numero di volte in cui il segnale viene ripetuto.

L'anomalia a2 non viene riconosciuta come causata dal segnale B: nonostante appaia nelle due liste, questa viene riconosciuta come la medesima e condivisa da entrambe; non soddisfando così il numero minimo di ripetizioni, questa viene scartata dalla lista. Per ragioni simili, l'anomalia a1 viene riconosciuta dall'algoritmo come “non correlata al segnale”, e scartata dal processo.

Una volta trovate le possibili correlazioni tra segnale B ed anomalia, quanto trovato non viene più considerato per agevolare il procedimento di correlazione con i segnali successivi. Si passa quindi al segnale A, dove vengono così riconosciute le ripetizioni delle anomalie a1 ed a2 (precedentemente visualizzate sotto B).

#### RISULTATI ALGORITMO REPETITIONS:

SEGNALE B: anomalie b1, b2, b3

SEGNALE A: anomalie a1, a2

#### 6.2.4 Differenze tra gli algoritmi

Mentre l'algoritmo naive permette di riconoscere le cause generate dallo stimolo in quanto singolo, l'algoritmo repetitions valorizza invece le correlazioni con le anomalie visualizzate dalla maggioranza della stessa tipologia di segnali.

Ma la ripetizione dello stesso stimolo potrebbe non portare all'esecuzione della stessa parte di codice; si pensi ad un processo che richieda l'accesso a risorse protette, non condivisibili con altri processi: finché le operazioni su di esse non sono terminate, non potranno essere avviati ulteriori processi interessati a quella risorsa. Lo stesso segnale potrebbe essere condiviso da più casi d'uso, e causare così differenti anomalie. L'algoritmo repetitions, utile per sopprimere eventuali disturbi generati nei dispositivi, in questo scenario è causa della mancata correlazione con le anomalie scatenate dal segnale.

le principale, ignorate perché non ripetute dallo stesso segnale inviato successivamente.

Se il punto di forza dell'algoritmo naive è quello di dare importanza al singolo segnale, ed il punto di forza dell'algoritmo repetitions è quello di ottenere una forte correlazione segnale-anomalia tramite ripetizione, si propone di unire queste due strategie in un unico procedimento in grado di cogliere i punti di forza di entrambe.

### 6.2.5 Algoritmo combinations

Quest'algoritmo si basa sull'unione dell'algoritmo naive, in grado di dare importanza alle anomalie generate dal singolo segnale, con l'algoritmo repetitions, in grado di fornire una forte correlazione tra il segnale ripetuto e le anomalie causate.

I presupposti su cui si basa sono i seguenti:

- Uno stimolo non può essere correlato con anomalie generate prima del suo invio.
- Le anomalie rilevate vicino agli orari di ricezione del segnale sono probabilmente correlate ad esso.
- Se un sistema risulta essere molto performante, il tempo richiesto per svolgere una determinata operazione potrebbe essere sempre minore di un intervallo T.
- Non è presente alcuna correlazione diretta nell'invio degli stimoli. Gli stimoli non sono quindi correlati tra loro.
- Uno stimolo potrebbe causare operazioni nel dispositivo di controllo ancora in fase di elaborazione durante l'invio dello stimolo successivo.
- Non è possibile sapere a priori il carico computazionale ed i requisiti nel dispositivo di osservazione richiesti in seguito all'invio di un segnale.

Questi presupposti, oltre definire i limiti e le modalità di esecuzione presenti in quest’approccio, permettono di stabilire una gerarchia d’importanza nelle correlazioni rilevabili tra i diversi dispositivi:

1. Correlazioni temporali.

Se un’anomalia viene generata sempre con un determinato intervallo di tempo dall’invio di un segnale, allora vi è una correlazione molto forte tra i due.

2. Correlazioni di ripetizione.

Se un’anomalia viene generata frequentemente in seguito all’invio di un determinato segnale, allora vi è una forte correlazione tra i due.

3. Correlazioni tra finestre temporali limitrofe.

Se un’anomalia è presente in un intervallo di tempo vicino all’invio di un segnale, allora quell’anomalia potrebbe essere causata da esso ed i due sono debolmente correlati.

Il rilevare una forte correlazione permette quindi di filtrare quanto trovato dalla lista delle anomalie osservate, procedendo le analisi e testando via via forme di correlazioni inferiori. I primi due punti si basano sull’algoritmo repetitions, che richiede l’invio al dispositivo di controllo di una stessa stimolazione più volte.

Come primo passo, viene analizzata ogni ripetizione di uno stesso segnale partendo dall’ultimo inviate, cercando anomalie verificatesi sempre ad una stessa distanza temporale; in seguito alla loro individuazione, viene quindi stabilita una correlazione segnale-anomalia di tipo (1).

Una volta analizzato ogni segnale e ripulito quanto trovato dalla lista di anomalie da analizzare, si analizzano nuovamente le ripetizioni di uno stesso segnale, questa volta cercando di individuare tramite l’algoritmo repetitions anomalie verificatesi molto frequentemente in seguito ad un loro invio. Qualora un’anomalia si sia verificata più del 50% delle volte in seguito ad un determinato stimolo, queste informazioni vengono salvate per stabilire una correlazione di tipo (2).

Terminate queste analisi, si procede con lo stabilire correlazioni deboli tra le anomalie rimaste ed i relativi segnali in prossimità, attraverso l’algoritmo naive.

In seguito vengono riportate i risultati dei vari algoritmi di correlazione nei vari scenari.

I segnali causano anomalie distinte tra loro, ognuna osservata prima dell'invio della stimolazione seguente

CASO OTTIMO	Nessun disturbo	Disturbi dal dispositivo di controllo	Disturbi dal dispositivo di osservazione
Alg.Naive	Correlazione perfetta	Correlazione anomalia con disturbi	Correlazione disturbo con segnale
Alg.Repetitions	Correlazione perfetta	Invariato	Invariato
Combinazione degli algoritmi	Correlazione perfetta	Invariato	Correlazione disturbo con segnale

I segnali sono soggetti a forte ritardo, tali da scatenare anomalie durante l'invio di segnali distinti successivi

SEGNALI CON RITARDO	Nessun disturbo	Disturbi dal dispositivo di controllo	Disturbi dal dispositivo di osservazione
Alg.Naive	Correlazione tra anomalia e segnale successivo	Correlazione anomalia con disturbi/segnali successivi	Correlazione disturbo con segnale
Alg.Repetitions	Correlazione perfetta, purchè la maggioranza delle anomalie si verifichino durante il tempo d'analisi del gruppo	Invariato	Invariato
Combinazione degli algoritmi	Correlazione perfetta, purchè la maggioranza delle anomalie si verifichino durante il tempo d'analisi del gruppo	Invariato	Correlazione disturbo con segnale

Figura 31: Comportamento dei vari algoritmi di correlazione nei possibili scenari (parte 1)

I segnali possono generare risultati differenti a seconda della tempistica con cui sono inviati. Ad esempio il primo segnale A darà inizio ad un processo di elaborazione, ma il secondo segnale A darà avvio ad un segnale di errore e di attesa, finché il processo iniziato non è terminato

STESO SEGNALE CON EFFETTI DIFFERENTI	Nessun disturbo	Disturbi dal dispositivo di controllo	Disturbi dal dispositivo di osservazione
Alg.Naive	Correlazione perfetta	Correlazione anomalia con disturbo	Correlazione disturbo con segnale
Alg.Repetitions	Vengono rilevate solo le anomalie scatenate dalla maggioranza dei segnali. Le anomalie causate dall'elaborazione principale vengono ignorate	Invariato	Invariato
Alg.Repetitions	Vengono rilevate solo le anomalie scatenate dalla maggioranza dei segnali. Le anomalie causate dall'elaborazione principale vengono attribuite al segnale più vicino	Se un'anomalia non viene ripetuta ed è presente un disturbo nelle vicinanze, i due vengono correlati	Correlazione disturbo con segnale

Figura 32: Comportamento dei vari algoritmi di correlazione nei possibili scenari (parte 2)

Come si evince dalle tabelle A, B, C (figure 31 e 32), l'unione dei due differenti approcci nell'algoritmo Combintions permette quindi di ridare importanza al singolo segnale, permettendo di stabilire correlazioni anche con segnali autonomi generati da processi aperiodici del dispositivo. Tuttavia ciò ha un costo: le anomalie generate spontaneamente dal dispositivo di osservazione, prima soppresse dall'algoritmo repetitions, ora sono correlabili debolmente con i segnali inviati in loro prossimità. Se questo reintroduce disturbi lato dispositivo IoT, presentando una correlazione falsa positiva, dall'altra attraverso l'osservazione in un ambiente “offline” del dispositivo si possono rilevare le anomalie generate autonomamente dal device, eliminandole successivamente durante le analisi dell'intero ecosistema.

Un altro punto degno di nota è che sebbene il processo di correlazione possa portare alla generazione di falsi positivi e falsi negativi, i processi per stabilire se si sono verificate anomalie nella sicurezza sono generalmente deterministic, dotati quindi di un'elevata precisione. Vi potranno essere falsi collegamenti tra anomalie e stimolazioni dell'ecosistema, ma le suddette anomalie rappresentano comunque degli indizi allo sviluppatore su ciò che bisogna correggere nel dispositivo.

## 7 AppIoTTE: un framework per l’analisi integrata

---

A seguito della definizione della metodologia di analisi dell’ecosistema di applicazioni mobile-IoT, durante il lavoro di tesi è stato sviluppato un framework, chiamato AppIoTTE, che consente il testing e la verifica di un ecosistema basato sulle tecnologie Android e Android Things.

In questo capitolo vengono discussi gli strumenti e le tecnologie utilizzate, illustrando in seguito l’implementazione completa del sistema.

Come accennato, in rete sono a disposizione numerosi strumenti in grado di analizzare applicazioni Android per verificarne la sicurezza. Un esempio è rappresentato da “Approver”, piattaforma basata su microservizi sviluppata da Talos, spin-off dell’Università di Genova, che propone le tecniche di analisi statica e dinamica per fornire all’utente un riassunto sullo stato della sicurezza della propria applicazione.

A differenza dello sviluppo “monolitico” di un programma (dove tutto il codice è contenuto in un’unica applicazione), un’architettura a microservizi si basa sullo sviluppo di unità logiche indipendenti tra loro e comunicanti tramite apposite API basate su protocollo HTTP. Ogni microservizio, realizzabile in un linguaggio di programmazione differente, è responsabile di una propria area di competenza con input e output ben definiti. Questa indipendenza tra i vari moduli, oltre a permettere una rapida parallelizzazione delle attività durante l’esecuzione del programma, facilita le operazioni di manutenzione o sostituzione che occorrono durante l’intero arco di vita del software. Si è quindi deciso di proseguire su questa linea, riadattando ed espandendo i servizi di analisi proposti da Approver per l’analisi di applicazioni Android Things e creando nuovi moduli necessari per l’analisi ambientale IoT (realizzati in Python 3.6.3).

## 7.1 AppIoTTE

Partendo da Approver come base, sono stati sviluppati nuovi moduli analoghi per l'analisi di applicazioni Android Things.

Data la grande somiglianza tra Android Things e Android nella struttura delle applicazioni, è stato possibile utilizzare molte delle tecniche di analisi statica presenti all'interno di Approver, applicando alcuni adattamenti al contesto IoT.

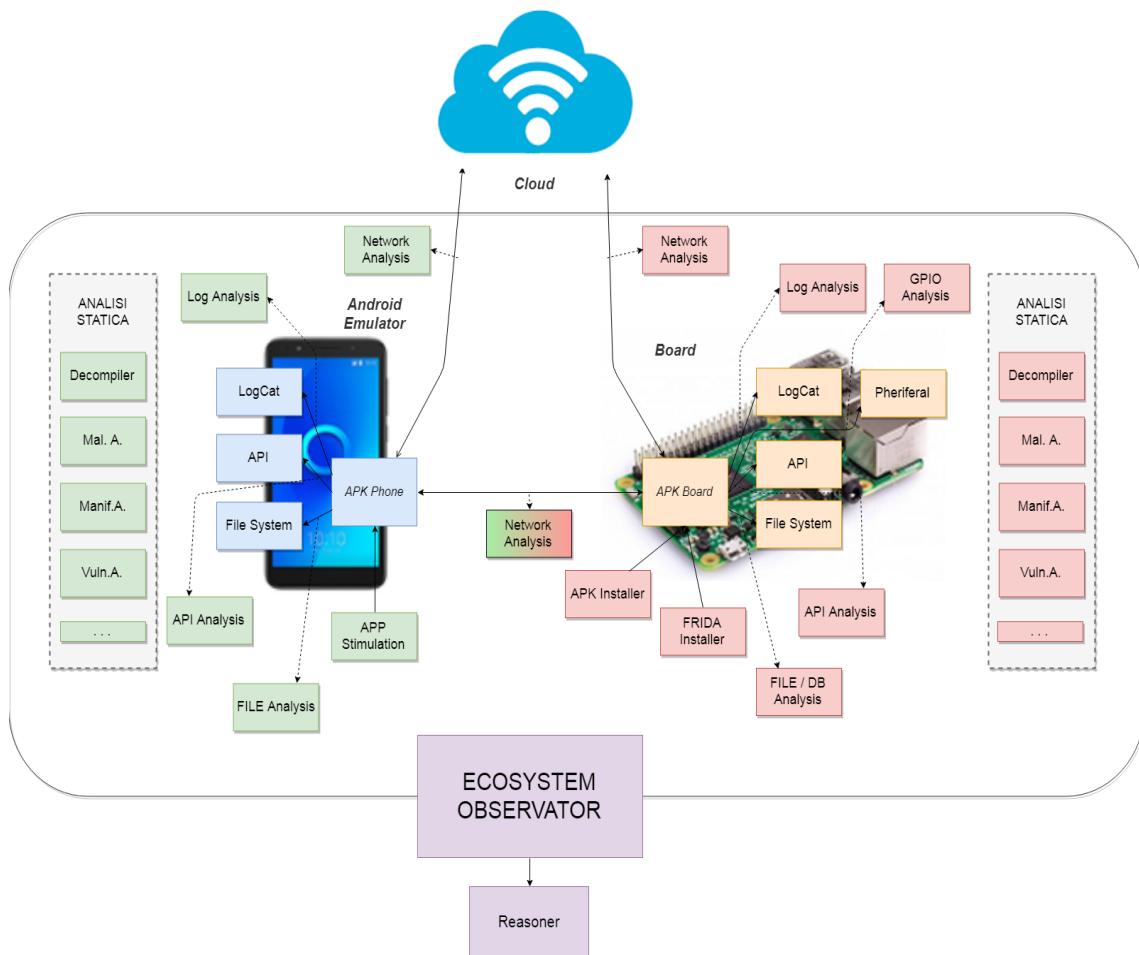


Figura 33: Panoramica dei moduli AppIoTTE per l'analisi mobile (verde) ed IoT (rosso)

## 7.2 Analisi statica

I seguenti moduli di analisi statica sono stati ripresi da Approver, per fornire una panoramica a 360 gradi sul codice dell'applicazione analizzata:

- Decompiler

Questo modulo prende in ingresso il file contenente l'applicazione e lo traduce in linguaggio smali; Da questo formato è possibile eseguire le analisi come se si stesse lavorando sul codice sorgente dell'applicazione.

- Manifest Parser

Questo modulo prende in ingresso il file manifest dell'applicazione, analizzandolo per estrarne informazioni utili come la struttura dell'applicazione, la sua versione, i permessi dichiarati, ecc.

- Permission Checker

Ha il compito di individuare quali sono i permessi dichiarati e quali sono effettivamente utilizzati dall'applicazione, attraverso un'analisi congiunta del file manifest e del codice dell'applicazione.

- String and API Analysis

Questo modulo prende in ingresso l'archivio zip con le risorse estratte dall'applicazione (artefatto reso disponibile dal Decompiler) e analizza il codice decompilato in formato smali contenuto all'interno. La funzione di questo modulo è duplice: il codice decompilato viene analizzato per individuare al suo interno tutte le stringhe *hard-coded* e allo stesso tempo vengono catalogate tutte le chiamate ad API Android rilevanti ai fini della sicurezza. Le stringhe "hard-coded" rappresentano un rischio per la sicurezza in quanto potrebbero contenere informazioni sensibili utilizzate in fase di sviluppo e che non sono state rimosse dalla versione finale dell'applicazione (ad esempio credenziali di test, chiavi per API di terze parti ecc.). La lista di chiamate ad API Android può invece essere interessante per indirizzare l'analista in caso di ispezione manuale del codice .

- Vulnerability Checker

Ha la funzione di analizzare il codice decompilato e determinare la presenza di eventuali vulnerabilità o pattern di codice che si possono tradurre in rischi per la sicurezza dell'applicazione.

- Malware Analysis

Ha il compito di identificare l'eventuale presenza di codice malevolo all'interno dell'applicazione. Questa operazione viene eseguita appoggiandosi al servizio esterno "*Jotti*", che fornisce alcune API per la scansione di file da parte di 15+ motori anti-malware

- APK Modeling

Questo modulo ha la funzione di analizzare il codice dell'applicazione ed estrarre da esso un *Inter-procedural Control Flow Graph* (IPCFG) rappresentante tutti i possibili percorsi di esecuzione dell'applicazione. Tale modello viene poi semplificato tenendo conto delle sole operazioni rilevanti ai fini della sicurezza.

- Risk Analysis

Questo modulo prende in ingresso i risultati ottenuti da diversi moduli e li combina per calcolare un valore di rischio di sicurezza per l'applicazione oggetto di analisi. In particolare, il valore di rischio complessivo è dato dal massimo valore assegnato.

Per rendere i moduli aggiornati con le funzionalità IoT offerte da Android Things, alla lista delle API interessanti sono state aggiunte il controllo delle periferiche, le comunicazioni LoWPAN e l'User Driver API.

## 7.3 Analisi dinamica

### 7.3.1 Dispositivi per il test

Nelle analisi dinamiche è richiesto di testare in tempo reale sia l'applicazione mobile, sia l'applicazione IoT; ma a differenza delle analisi statiche, per far ciò è necessario avere un device dove poter eseguire i test.

Per il mondo mobile sono a disposizione online numerosi emulatori virtuali con i quali poter ricreare l'esecuzione di uno smartphone: attraverso l'uso di "VirtualBox" [22], noto emulatore virtuale, si ricrea un'immagine del sistema operativo Android con tutti gli strumenti richiesti già installati ed a disposizione, pronto per testare l'applicazione mobile bersaglio.

Non sono però attualmente disponibili emulatori Android Things: per eseguire un'analisi dinamica dell'applicazione AT da analizzare, AppIoTTE richiede di essere collegato attraverso ADB (Android Debug Bridge) con una board dove è presente una versione di AT adibita al testing. AppIoTTE controllerà la possibilità di ottenere i privilegi di amministratore (necessari per installare tutti gli strumenti richiesti), procedendo in caso di conferma con l'installazione dell'applicazione da analizzare e degli strumenti attraverso i moduli Apk e Frida Installer.

### 7.3.2 Frida

Frida [23] è un toolkit che permette di iniettare frammenti di codice Javascript personalizzati nelle librerie native Android. Viene iniettato il V8 engine di Google nel processo di destinazione, dove il codice Javascript può così venir eseguito con pieno accesso alla memoria, funzioni di hooking e chiamate di funzioni native all'interno del processo, creando inoltre un canale di comunicazione bidirezionale usato per trasmettere informazioni tra la propria console ed il codice javascript iniettato (figura 34).

Per permettere una simile alterazione della normale procedura di esecuzione dell'applicazione, il modulo Frida si occuperà di installare i certificati necessari e di avviare il server Frida all'interno del dispositivo, specificando nel codice Javascript tutti i metodi, le chiamate e gli oggetti da agganciare e da monitorare.

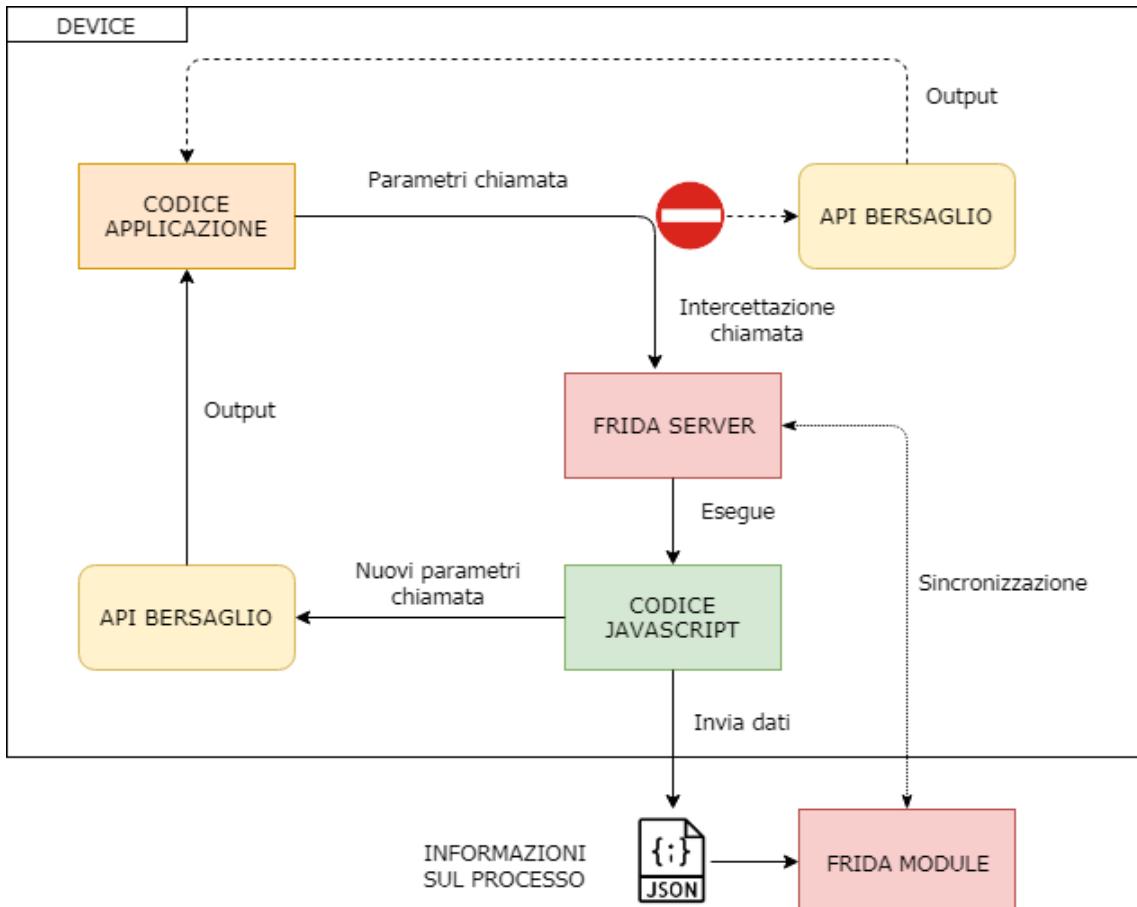


Figura 34: Esempio di una cattura di chiamata API con Frida

Durante l'esecuzione dell'applicazione nel dispositivo, il modulo Frida riceverà quindi informazioni sugli eventi accaduti tramite comunicazione Json. Sono ritenute informazioni interessanti l'API interessa, il metodo richiamato, gli argomenti passati, i valori di ritorno e quando è stato rilevato l'evento. Il codice javascript necessario per l'intercettazione viene riassunto brevemente in figura sottostante.

---

#### Algorithm 4: Frida Hook

---

```

1 ...
2 JAVA.API.METHOD.OVERLOAD(arg).(values) {
3     returnValue = call. 'JAVA'.API.METHOD.OVERLOAD(arg);
4     SendProcessInfo(Date, API, Method, Overloads, values);
5     Return (returnValue);
6 }

```

---

### 7.3.3 Analisi network

Per poter controllare le comunicazioni effettuate sia dal dispositivo mobile che dalla board IoT, vi è bisogno di direzionare il suo traffico di rete attraverso un proxy<sup>24</sup>, strumento atto a servire da intermediario tra il device e l'esterno. Per far ciò si è utilizzato Mitmproxy [24], software open-source modulare e facilmente configurabile che permette di intercettare e memorizzare il traffico di rete.

Il modulo di network, tramite tecniche di Man in the Middle, è in grado rilevare le seguenti configurazioni:

- Traffico non cifrato

Viene avviata l'applicazione di testing, ma senza i certificati richiesti per autenticare Mitmproxy, osservando come il sistema reagisce ad una simile intrusione; solamente protocolli non sicuri come http dovrebbero essere visualizzabili dal proxy, e direzionati alla destinazione. Ogni risultato ottenuto dal proxy deve essere considerato quindi come vulnerabilità nella sicurezza.

- Traffico cifrato ma con una cattiva configurazione

Similmente al punto precedente, l'applicazione viene avviata senza disporre dei certificati d'autenticazione richiesti. Nonostante l'uso di comunicazioni cifrate, il proxy è comunque in grado di intercettare il messaggio, sintomo di una cattiva configurazione o metodi di crittografia inefficienti.

- Traffico cifrato, ma senza ssl pinning

viene lanciata nuovamente l'applicazione con i certificati richiesti per autenticare il proxy. In questa parte il proxy dovrebbe essere in grado di visualizzare il contenuto di ogni trasmissione, criptata o meno, con l'eccezione di comunicazioni facenti uso di ssl pinning.

- Traffico cifrato con ssl pinning

SSL pinning è una tecnica usata lato client per evitare attacchi Man in the Middle, attraverso la validazione dei certificati server anche in seguito all'SSL handshaking. Gli sviluppatori incorporano una lista di certificati fidati nell'applicazione,

---

<sup>24</sup> <https://it.wikipedia.org/wiki/Proxy>

e li usano per comparare i certificati del server durante la sua esecuzione. Se viene rilevata una differenza, la connessione viene rifiutata, non permettendo l'invio di ulteriori dati a quell'indirizzo. Questo assicura che il dispositivo comunichi solamente con server dedicati e fidati. L'intercettazione di simili messaggi da parte del proxy comporta quindi l'interruzione della comunicazione, andando ad influire sul funzionamento dell'applicazione; per evitare ciò, in questa fase il proxy viene quindi disabilitato.

In alcune fasi dell'analisi, per poter ispezionare il traffico di rete cifrato passante per il proxy, il modulo deve ricorrere a tecniche di SSL inspection che richiedono l'installazione di certificati direttamente sul dispositivo, sia mobile che IoT.

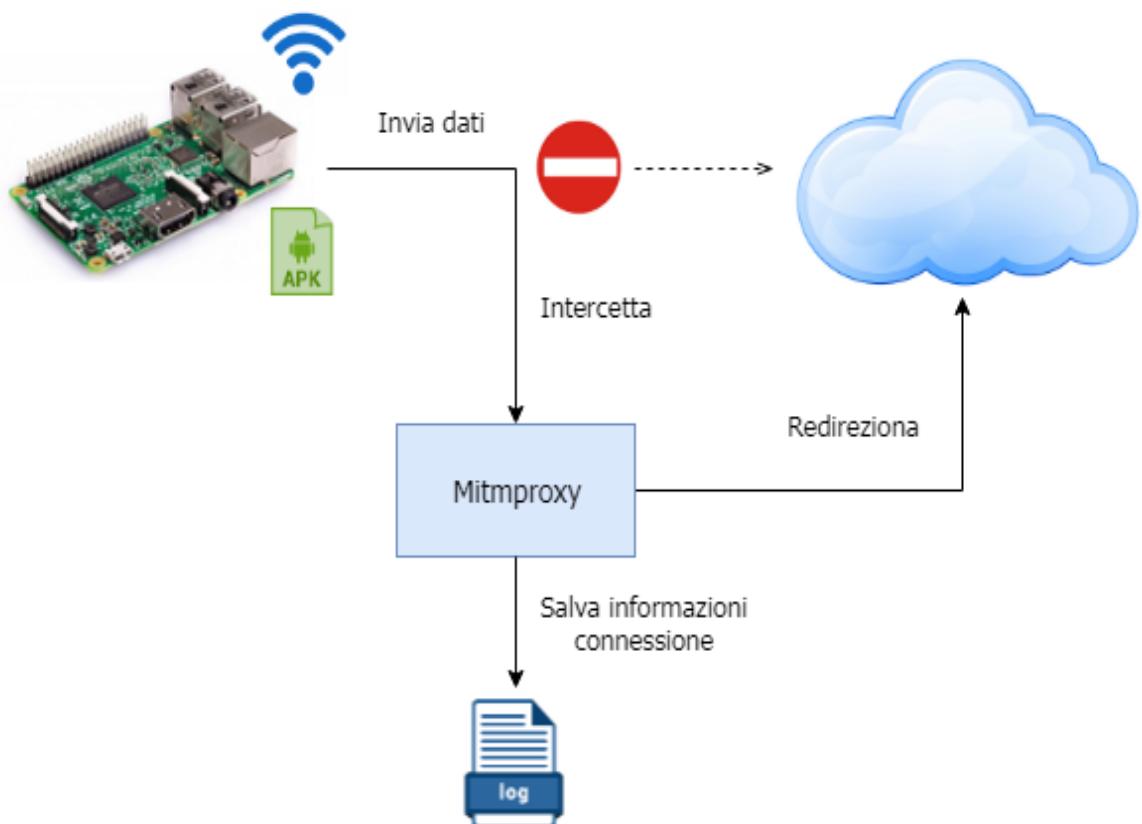


Figura 35: Funzionamento generale di un proxy

### 7.3.4 Moduli implementati

Attraverso tools come Frida e Mitmproxy, sono stati sviluppati vari moduli che permettono di conoscere l'attività dell'applicazione in esecuzione. Basandosi sui meccanismi di analisi già implementati da Approver per Android, i nuovi moduli si concentrano sulle analisi di applicazioni Android Things:

- FILE/DB Analysis

È la parte relativa al salvataggio in memoria delle informazioni elaborate dal dispositivo. Si occupa di tracciare la creazione di nuovi files (nonché la loro lettura, scrittura o cancellazione), inviando alla console informazioni come percorso del file, modalità d'esecuzione, contenuto trasmesso e l'ora (in millisecondi) in cui è stata effettuata l'operazione.

Inoltre, viene registrata ogni richiesta di elaborazione di una query ad un database MySQL. Dato che la query contiene tutte le informazioni su cosa è stato eseguito, viene quindi memorizzata ed inviata alla console insieme all'ora in millisecondi.

- API Analysis

Si occupa dell'intercettazione di tutte le chiamate API native di Android Things. Da questa lista di API, vengono estratti i relativi metodi in modo da agganciare le loro chiamate con Frida. Una volta intercettata la chiamata, vengono memorizzati gli argomenti trasmessi e viene eseguito il codice richiamato, salvandone il risultato. È così possibile inviare alla console l'API, il metodo, gli argomenti passati ed il risultato ottenuto dal processo, oltre che all'ora in cui è stato richiamato.

Riguardo alle API per la comunicazione con le periferiche esterne, si è deciso di concentrare gli studi sulle periferiche GPIO, soffermandosi sulle azioni eseguibili. Per le altre tipologie di comunicazione, è sempre possibile utilizzare questo modulo generico per osservarne i comportamenti.

- GPIO Analysis

Simile al modulo per l'analisi delle API, ma è stato dedicato un modulo apposito per studiare come il device comunica con le periferiche esterne, riportando quali porte vengono aperte e quali segnali vengono inviati/ricevuti dall'esterno, insieme

me al loro orario.

- **LOG Analysis**

Ha il compito di catturare ogni messaggio di Log generato dal sistema, riportare l'ora in millisecondi ed inviare il contenuto del messaggio alla console.

- **Network Analysis**

Si occupa dell'intercettazione di tutte le comunicazioni network, attraverso tre diversi stage: senza certificati installati, con certificati installati, proxy disabilitato per permettere lo ssl pinning.

Il contenuto trasmesso da questi moduli viene raccolto e catalogato in appositi buffer, per poi creare un report generale ed avere una visione globale di quanto successo durante l'esecuzione dell'applicazione.

## 7.4 Architettura

Il framework AppIoTTE, attraverso l'unione di tutte le componenti descritte finora, permette di monitorare il funzionamento delle singole applicazioni smartphone/IoT in tempo reale. Lo studio dell'intero ecosistema è invece fornito attraverso la correlazione dei risultati ottenuti, al fine di stabilire una corrispondenza tra stimolo ed effetti causati.

È stata quindi implementata l'architettura generale descritta in figura 33 con i vari moduli che la compongono. Per semplificare la visione dell'implementazione, l'architettura è stata suddivisa in macromoduli logici, come il modulo “analisi interna”, che raggruppa tutti i moduli di analisi specializzati nell'analizzare l'interazione dell'applicazione con il sistema operativo Android Things.

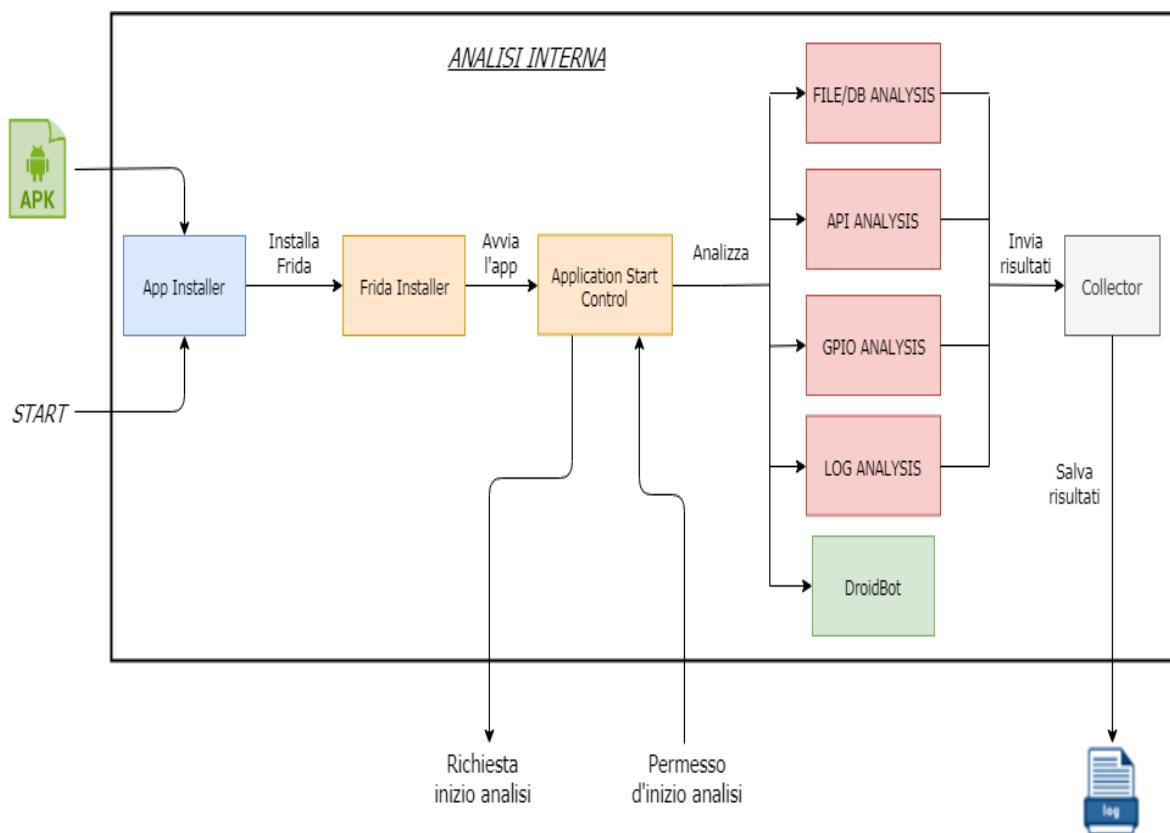


Figura 36: Composizione del modulo “Analisi Interna”

Il macromodulo descritto in figura 36 prende in ingresso un segnale di inizio analisi, insieme ad altri argomenti come l’apk dell’applicazione, l’indirizzo IP della board dove installare l’applicazione e dove salvare le informazioni raccolte. All’inizio del procedimento, il modulo “*App Installer*” esegue un’installazione pulita dell’applicazione in esame nella board specificata, proseguendo poi con l’installazione del tool Frida nel modulo successivo, il “*Frida Installer*”, con i permessi richiesti per garantirne il corretto funzionamento.

Al termine delle installazioni, il modulo “*Application Start Control*” ha il duplice compito di avviare Frida all’interno della Board, caricare in esso il file Javascript contenente la logica dietro tutte le intercettazioni, e di coordinarsi con gli altri membri dell’architettura. Una volta terminata l’inizializzazione di Frida, si comunica con gli altri moduli che si è pronti per iniziare le analisi; una volta ricevuto il permesso, si avvia l’applicazione e si analizza il suo funzionamento. Il modulo “*Droidbot*” ha il compito di stimolare automaticamente l’applicazione, generando nuovi use case da analizzare [25].

I moduli “*FILE/DB ANALYSIS*”, “*API ANALYSIS*”, “*GPIO ANALYSIS*” e “*LOG ANALYSIS*” hanno il compito di raccogliere e catalogare tutte le informazioni inviate da Frida, inviando quanto ottenuto al modulo “*Collector*” che provvederà a salvare le informazioni dove precedentemente specificato.

All’analisi interna viene unita l’analisi network (figura 37), che ha il compito di monitorare le comunicazioni effettuate dal dispositivo. Il modulo “*Proxy*” avvierà quindi Mitmproxy e programmerà la board per redirigere in esso tutto il suo traffico. I risultati sono quindi raccolti e memorizzati nel percorso specificato nel modulo “*Collector*”, che deciderà inoltre della necessità di continuare o meno l’analisi a seconda che siano richiesti maggiori privilegi di accesso per monitorare al meglio le comunicazioni.

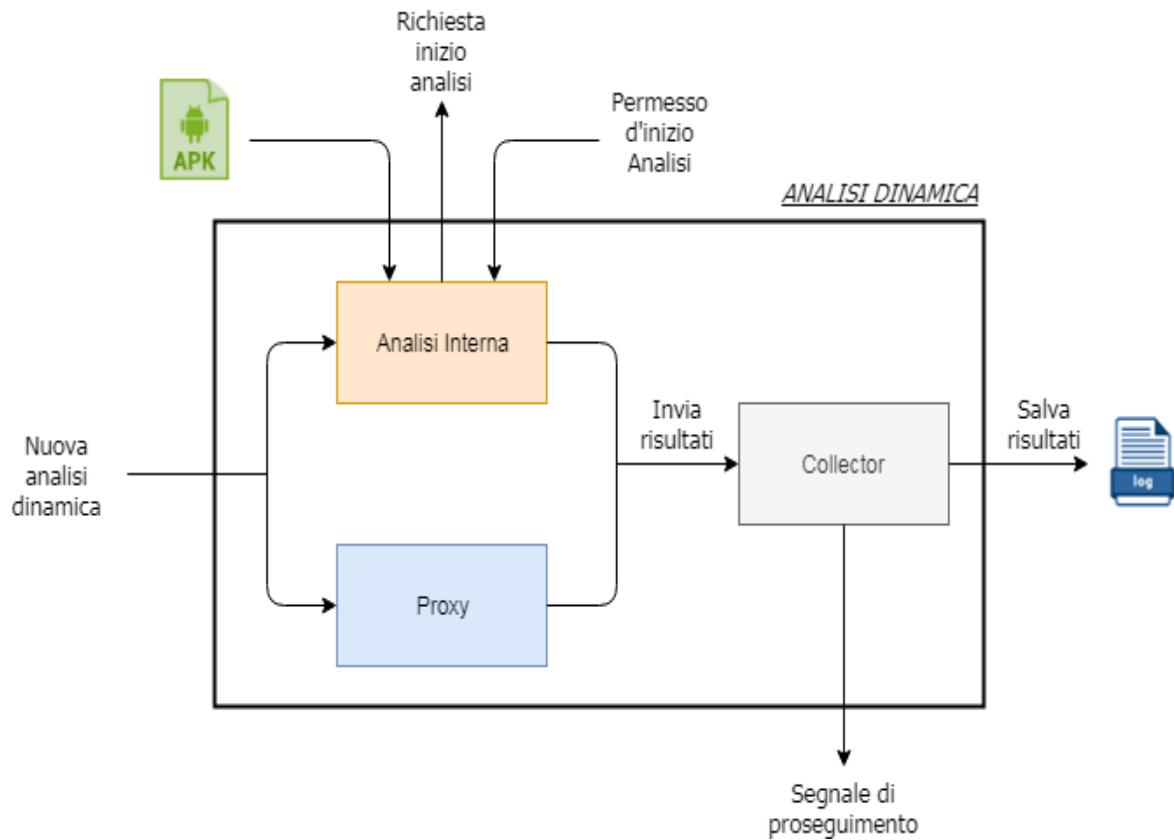


Figura 37: Composizione del macromodulo “Analisi Dinamica”

Senza l'installazione dei certificati, il proxy riesce ad intercettare solamente comunicazioni non criptate. Per questo, una volta accertato che l'applicazione riesca ad identificare la presenza di terze parti nei canali di comunicazione, si passa allo step successivo, ripetendo le analisi in seguito all'installazione dei certificati richiesti.

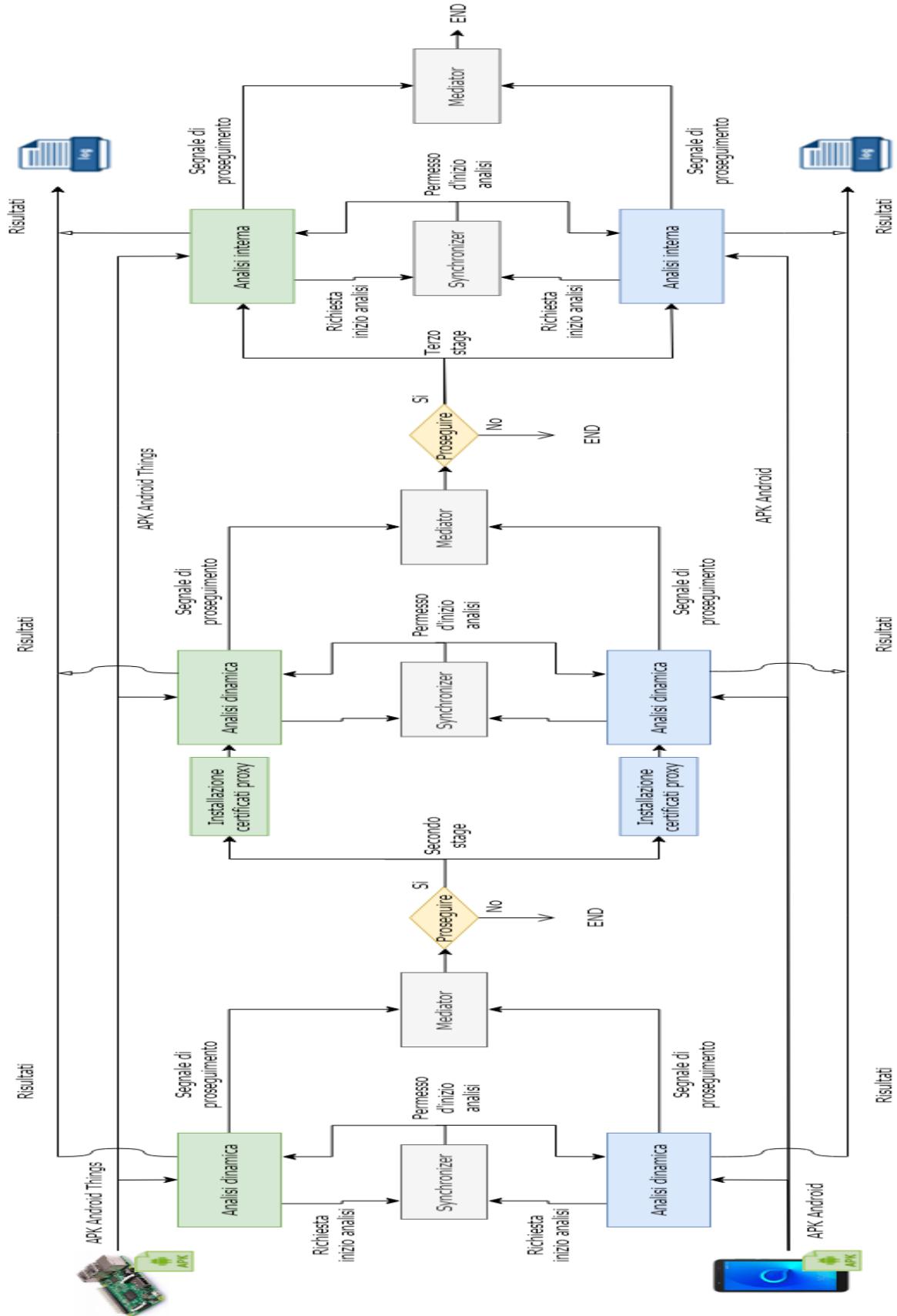


Figura 38: Architettura dell’analisi congiunta tra mobile e dispositivo IoT

La figura 38 riporta quindi l'intero processo di analisi congiunta tra board (in verde) e cellulare (in blu). I moduli “*Synchronizer*” e “*Mediator*” sono stati realizzati con lo scopo di mettere in comunicazione i processi di analisi svolti in parallelo ed in contemporanea. Posto in ascolto, il modulo *Synchronizer* rimane in attesa dei messaggi di richiesta di analisi inviati dai moduli di analisi dinamica; solamente quando entrambi i moduli hanno inviato con successo queste richieste, viene inviato il permesso di iniziare le analisi, in modo da sincronizzare i due processi e permettere ai due dispositivi di comunicare tra loro.

Una volta terminate le analisi, i rispettivi moduli di analisi della board e del cellulare decidono se è nel loro interesse continuare le analisi, procedendo allo step successivo, o fermarsi (in quanto gli step successivi non creerebbero nuove informazioni). Queste decisioni vengono inviate al modulo “*Mediator*”, che ha il compito di tenersi in attesa delle due risposte ed elaborare la strategia da seguire: solamente quando entrambi i processi di analisi sono giunti alla conclusione dell'inutilità di andare oltre, il processo generale di analisi dell'infrastruttura ha termine; in caso contrario, si procede allo step successivo, installando i certificati proxy nei due dispositivi o proseguendo con le analisi senza inserire proxy nel processo.

Si noti che la figura 38 mostra il processo di analisi congiunta tra i due dispositivi. Qualora sia richiesta l'analisi di un dispositivo singolo, è necessario rimuovere i moduli di analisi della controparte (in verde o in blu), permettendo ai moduli “*Synchronizer*” e “*Mediator*” di effettuare comunque la loro risposta anche in mancanza del secondo input.

Al termine dell'analisi generale dell'ecosistema IoT, si hanno a disposizione tutti i resoconti dell'attività delle applicazioni in merito alle varie casistiche. Si procede quindi all'elaborazione di questi dati per tracciare un assestamento della sicurezza attraverso il reasoner.

## 7.5 Reasoner

### 7.5.1 Schema

Una volta ottenuti i dati risultanti dalle analisi dei dispositivi, questi vengono inviati al “*reasoner*” per proseguire nella loro elaborazione e verificare la presenza di vulnerabilità della sicurezza (figura 39)

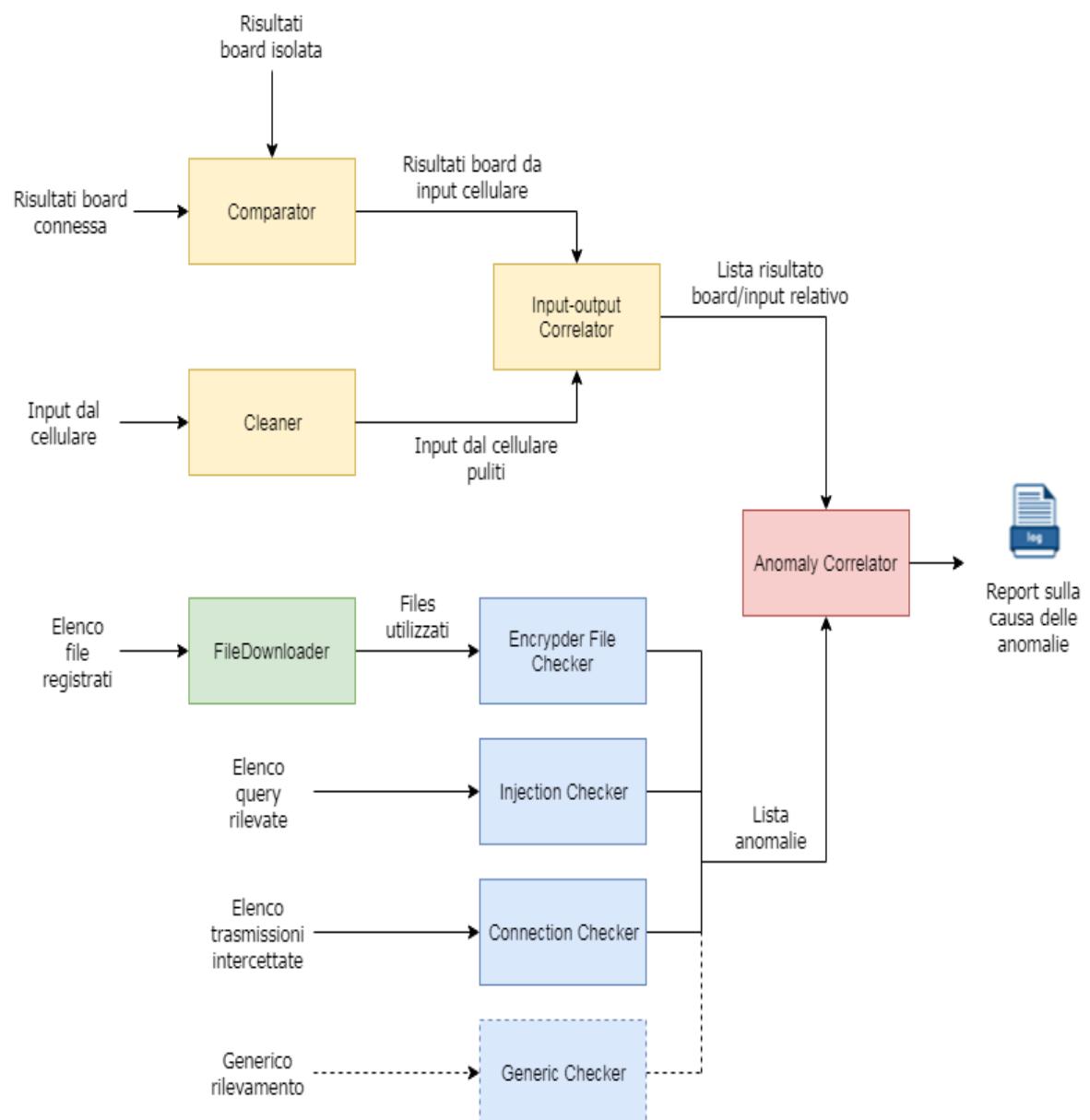


Figura 39: Schema interno del reasoner AppIoTTE

I compiti del reasoner sono due:

- 1) elaborare le informazioni ricevute in merito all'esecuzione delle applicazioni mobile e IoT e 2) Gestire le informazioni derivanti dalle analisi congiunte del dispositivo di controllo e di osservazione (in figura 39 rappresentati rispettivamente da un cellulare Android e da un dispositivo AT)

### 7.5.2 Elaborazione informazioni

Tutte le informazioni relative all'uso di file, database, api, connessioni network ed azioni eseguite dal dispositivo vengono analizzate per individuare possibili vulnerabilità nella sicurezza. Nel prototipo sono stati implementati:

L' "*Encrypter File Checker*", dove grazie ai privilegi di amministratore si scaricano tutti i files che sono stati utilizzati dall'applicazione, verificando se sono stati criptati o meno tramite una misura della loro entropia.

L' "*Injection Checker*", dove si verifica che non sia stato possibile ottenere informazioni riservate (o non accessibili dal database interno dell'applicazione) tramite iniezioni dirette di stringhe SQL dall'interfaccia utente o intercettando il traffico di rete.

Il "*Connection Checker*", dove si analizza il traffico di rete del dispositivo per verificare che tutte le comunicazioni siano correttamente criptate e che rispettino i normali protocolli di sicurezza.

Le informazioni rimanenti vengono analizzate tramite moduli generici, per determinare se queste possano rappresentare una superficie di attacco sfruttabile da un attaccante. Tutte le vulnerabilità di sicurezza vengono quindi riportate in una lista; qualora le informazioni siano state raccolte durante l'analisi congiunta con più dispositivi, vengono correlate con l'input relativo nel modulo successivo.

### 7.5.3 Filtraggio dati e correlazione

Vengono analizzate le stimolazioni dell’ecosistema e vengono riportate le comunicazioni inviate dal dispositivo di controllo, filtrate nel modulo “*Cleaner*” dai disturbi noti che potrebbero sorgere (come ad esempio il controllo della connettività da parte dei servizi Google). In parallelo vengono analizzate le informazioni raccolte durante le analisi del dispositivo di osservazione, confrontando i rilevamenti ottenuti durante la stimolazione dell’ecosistema con quelli osservati precedentemente, dove il dispositivo di osservazione non era in grado di connettersi al resto dell’infrastruttura. Il modulo “*Comparator*” ha quindi il compito di stabilire quali dei comportamenti sono imputabili all’esecuzione autonoma del device, e quali sono stati causati da dispositivi esterni, riportando questi ultimi in una lista.

Gli input inviati dal dispositivo di controllo e le azioni causate da device esterni rilevate nel dispositivo di osservazione sono successivamente inviate all “*Input-Output Correlator*”; questo modulo ha il computo, come suggerisce il nome, di correlare tra loro questi elementi applicando gli algoritmi analizzati precedentemente nella metodologia. Stabilite le correlazioni tra input e cause, ogni azione catalogata come “anomalia” viene ricondotta all’input scatenante nel modulo “*Anomaly Correlator*”, riportando un resoconto dettagliato dello stato della sicurezza del dispositivo di osservazione allo sviluppatore.

### 7.5.4 Algoritmo di correlazione utilizzato

Sono state effettuate varie prove per testare l’efficacia dei vari algoritmi e determinare quale implementare. Dato che l’obiettivo del framework è analizzare applicazioni generiche, senza porre dei limiti, vi sono un’infinità di combinazioni con cui i dispositivi di controllo e di osservazione potrebbero interagire tra loro, passando da un’esecuzione breve ed efficiente scatenata da un determinato input utente a situazioni dove l’elaborazione è smistata su più dispositivi, lavoranti in maniera congiunta e dipendente.

Si è deciso quindi di analizzare un caso pessimo, dove:

- L’ecosistema è soggetto a numerosi stimoli ripetuti tra loro non necessariamente in maniera sequenziale

- Le risposte a questi segnali avvengono in tempi molto lunghi e potenzialmente ancora elaborati durante l'invio di uno stimolo successivo
- Sono presenti dei disturbi su entrambi i dispositivi (segnali inviati a terze parti dal dispositivo di controllo, processi autonomi precedentemente non rilevati nel dispositivo di osservazione)
- Sono presenti segnali generati autonomamente dal dispositivo di controllo, che hanno ripercussioni sul dispositivo di osservazione

Nello scenario analizzato erano presenti:

- 16 stimoli dell'ecosistema, ognuno ripetuto due o tre volte e non sempre lo stesso stimolo causa le stesse reazioni nel sistema
- 44 anomalie causate da relativi stimoli. A volte un'anomalia relativa allo stimolo X può apparire durante l'invio dello stimolo Y
- 2 segnali asincroni, provenienti dal dispositivo di controllo in maniera automatica e scatenanti ognuno due anomalie nel dispositivo di osservazione
- 3 segnali di disturbo nel dispositivo di controllo.
- 3 anomalie di disturbo nel dispositivo di controllo, causate in maniera automatica e non correlate ad alcun segnale

Di seguito si riportano i risultati:

ALGORITMO	ANOMALIE CORRELATE CORRETTAMENTE	ANOMALIE CORRELATE IN MODO ERRATO	ANOMALIE NON CORRELATE	ANOMALIE DI DISTURBO FALSAMENTE CORRELATE
Naive	24	10	0	3
Repetitions	24	3	7	0
Combinations	30	4	0	3

ALGORITMO	ACCURATEZZA	PRECISIONE
Naive	64%	70%
Repetitions	72%	71%
Combinations	81%	93%

Figura 40: Risultati dell’analisi di un caso pessimo

Come è possibile osservare dalla figura 40, i risultati ottenuti combaciano con quanto studiato nella metodologia: singolarmente gli algoritmi naive e repetitions risultano insufficienti per stabilire una buona correlazione tra anomalia e segnale, il primo producendo un elevato numero di false correlazioni ed il secondo ignorando le anomalie ripetute un numero insufficiente di volte (producendo così un elevato numero di mancate correlazioni, spesso risultate ovvie applicando l’algoritmo naive).

L’unione dei due algoritmi risulta essere la scelta migliore, mostrando un buon grado di accuratezza e di precisione nelle operazioni di correlazione, merito dell’unione dei rispettivi punti di forza ereditati. Sebbene l’algoritmo erediti una mancanza di riconoscimento delle anomalie create da processi autonomi e quindi non correlate, que-

st’aspetto è mitigato dalle operazioni preliminari di comparazione tra il dispositivo di osservazione connesso e non connesso alla sua infrastruttura.

Per queste ragioni, si è quindi deciso di implementare nel reasoner l’algoritmo **“Combinations”**.

Al termine delle operazioni di correlazione, vengono forniti in uscita tutti i report contenenti l’assestamento dello stato di sicurezza dei vari dispositivi, elencando le vulnerabilità di sicurezza rilevate, le cause scatenanti e le correlazioni con altre operazioni effettuate in diversi dispositivi.

## 8 Casi d'uso

---

In questo capitolo viene mostrato come è possibile utilizzare la piattaforma AppIoT-TE per testare lo stato di sicurezza dell'ecosistema mobile-IoT. In dettaglio, vengono descritti i passi operativi necessari per avviare le analisi sincronizzate tra il controller smartphone (rappresentato da un'applicazione Android) ed il device IoT (rappresentato da un'applicazione Android Thing), mostrando i risultati prodotti dall'osservazione del sistema.

### 8.1 Preparazione all'avvio

#### 8.1.1 Analisi Mobile

Il prototipo è stato inizialmente realizzato partendo da un deploy in locale dei servizi di analisi dinamica di Approver. È quindi richiesto all'avvio la preparazione di una macchina virtuale dove poter riprodurre l'immagine di un sistema operativo Android già pronto per l'analisi.

Selezionando all'interno della directory del programma il percorso relativo all'analisi mobile, si avvia l'ambiente attraverso i comandi `./setup_environment.sh`

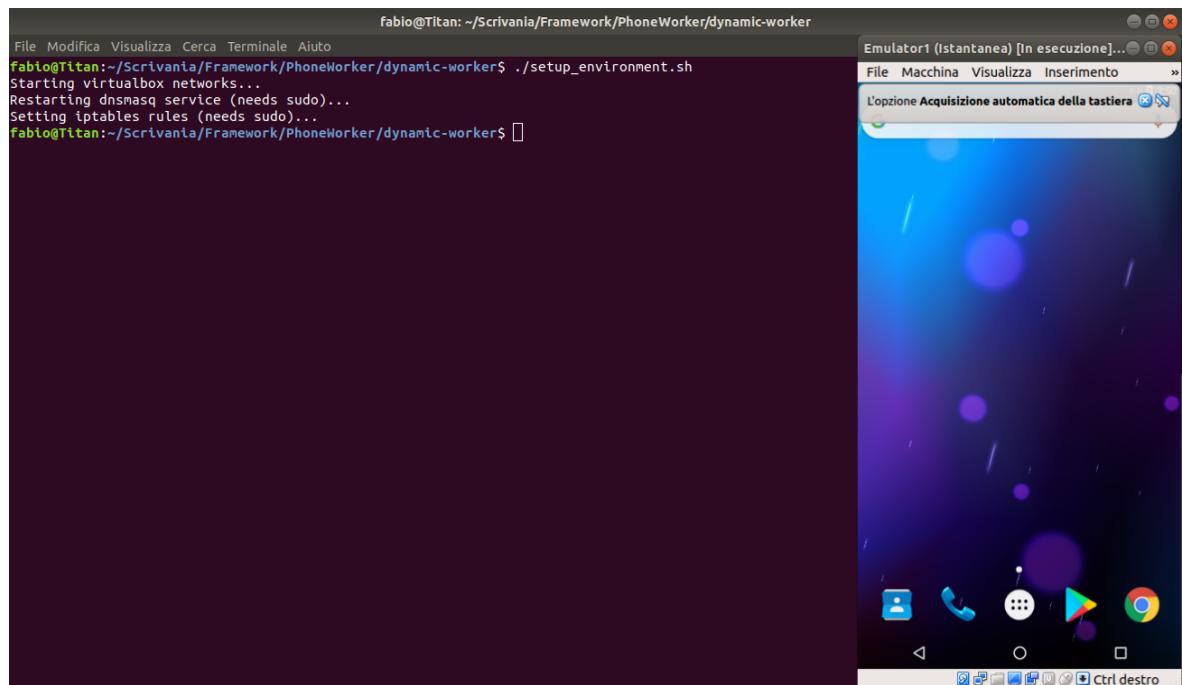
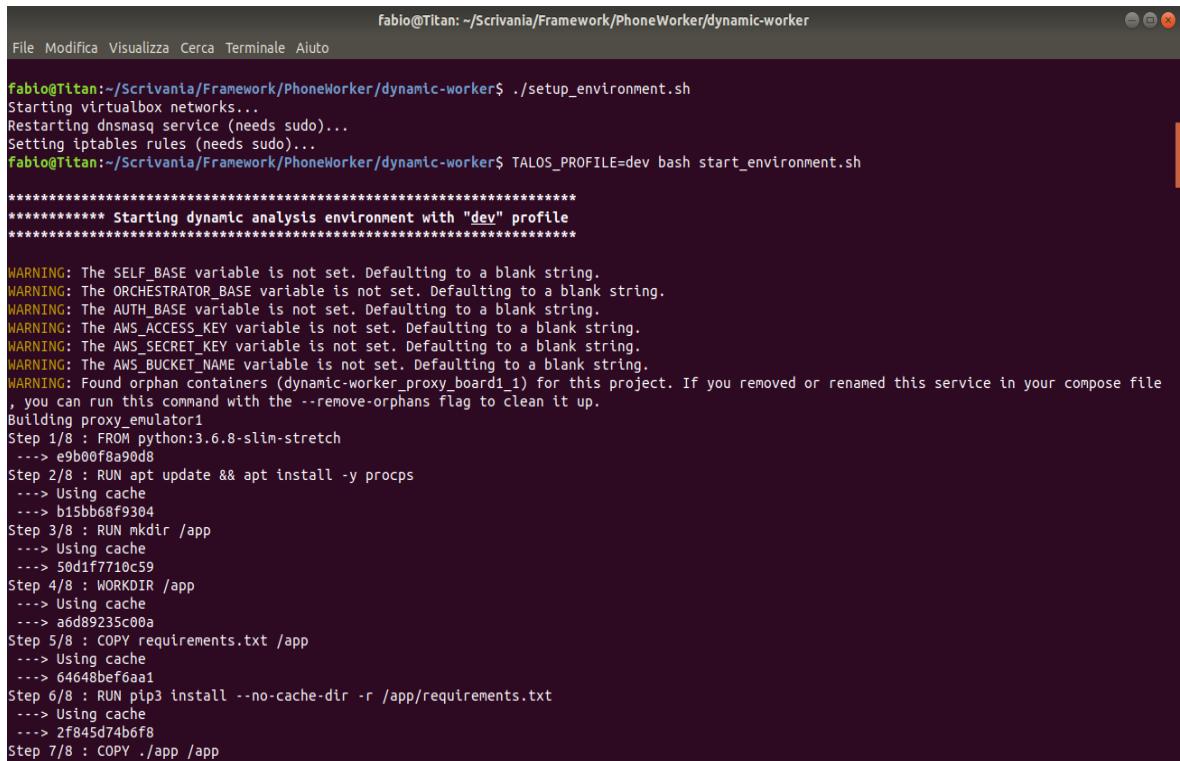


Figura 41: Comando `./setup_environment.sh` a console

e `TALOS-PROFILE=dev bash start_environment.sh`



```
fabio@Titan: ~/Scrivania/Framework/PhoneWorker/dynamic-worker
File Modifica Visualizza Cerca Terminale Aiuto
fabio@Titan:~/Scrivania/Framework/PhoneWorker/dynamic-worker$ ./setup_environment.sh
Starting virtualbox networks...
Restarting dnsmasq service (needs sudo)...
Setting iptables rules (needs sudo)...
fabio@Titan:~/Scrivania/Framework/PhoneWorker/dynamic-worker$ TALOS_PROFILE=dev bash start_environment.sh
*****
***** Starting dynamic analysis environment with "dev" profile
*****
WARNING: The SELF_BASE variable is not set. Defaulting to a blank string.
WARNING: The ORCHESTRATOR_BASE variable is not set. Defaulting to a blank string.
WARNING: The AUTH_BASE variable is not set. Defaulting to a blank string.
WARNING: The AWS_ACCESS_KEY variable is not set. Defaulting to a blank string.
WARNING: The AWS_SECRET_KEY variable is not set. Defaulting to a blank string.
WARNING: The AWS_BUCKET_NAME variable is not set. Defaulting to a blank string.
WARNING: Found orphan containers (dynamic-worker_proxy_board1_1) for this project. If you removed or renamed this service in your compose file , you can run this command with the --remove-orphans flag to clean it up.
Building proxy_emulator1
Step 1/8 : FROM python:3.6.8-slim-stretch
--> e9b0f8a90d8
Step 2/8 : RUN apt update && apt install -y procps
--> Using cache
--> b15bb08f9304
Step 3/8 : RUN mkdir /app
--> Using cache
--> 50df1f7710c59
Step 4/8 : WORKDIR /app
--> Using cache
--> a6d89235c00a
Step 5/8 : COPY requirements.txt /app
--> Using cache
--> 04648bef6aa1
Step 6/8 : RUN pip3 install --no-cache-dir -r /app/requirements.txt
--> Using cache
--> 2f845d74b6f8
Step 7/8 : COPY ./app /app
```

Figura 42: Comando `TALOS-PROFILE=dev bash start_environment.sh` a console

Dato che il software di analisi è stato originariamente pensato per operare come una piattaforma a microservizi, è richiesto l'avvio di un server locale per gestire la parte di analisi dei dati, all'interno del percorso dove è contenuto il file apk da analizzare. Il comando è "`Python3 -m http.server 9999`"

Il server di analisi è così pronto a gestire tutte le richieste di analisi, sottoposte attraverso un comando curl specificando il valore MD5 assegnato e l'apk da analizzare.

```
Curl -X POST -vk https://localhost/apks -H"content-type:application/json" -d'{
"apk": {
"md5": MD5_VALUE,
"url": "http://172.17.0.1:9999/" + APK
}
}'
```

### 8.1.2 Analisi dell'ecosistema mobile-IoT

Una volta preparato il server in parallelo per l'analisi di applicazioni Mobile, per avviare le analisi dell'ecosistema IoT è necessario avviare il modulo “orchestrator” che si occuperà di coordinare ogni modulo interessato nel processo. Attraverso l'apposita interfaccia di comando verrà richiesto di inserire l'indirizzo IP della board Android Things (testing mode) su cui si vogliono effettuare le analisi. Una volta verificata la connessione, si potrà inserire il percorso dell'Apk A.T. per iniziare il processo di osservazione in maniera totalmente automatica; ogni aspetto di sincronizzazione tra mondo mobile e mondo IoT verrà gestito autonomamente dal programma, seguendo i processi logici descritti nei capitoli precedenti.

La stimolazione dell'ecosistema è normalmente effettuata in automatico stimolando sistematicamente punti dell'interfaccia di accesso; non è però preclusa all'utente la possibilità di interagire durante il processo, potendo andare ad effettuare personalmente determinati stimoli nell'applicazione.

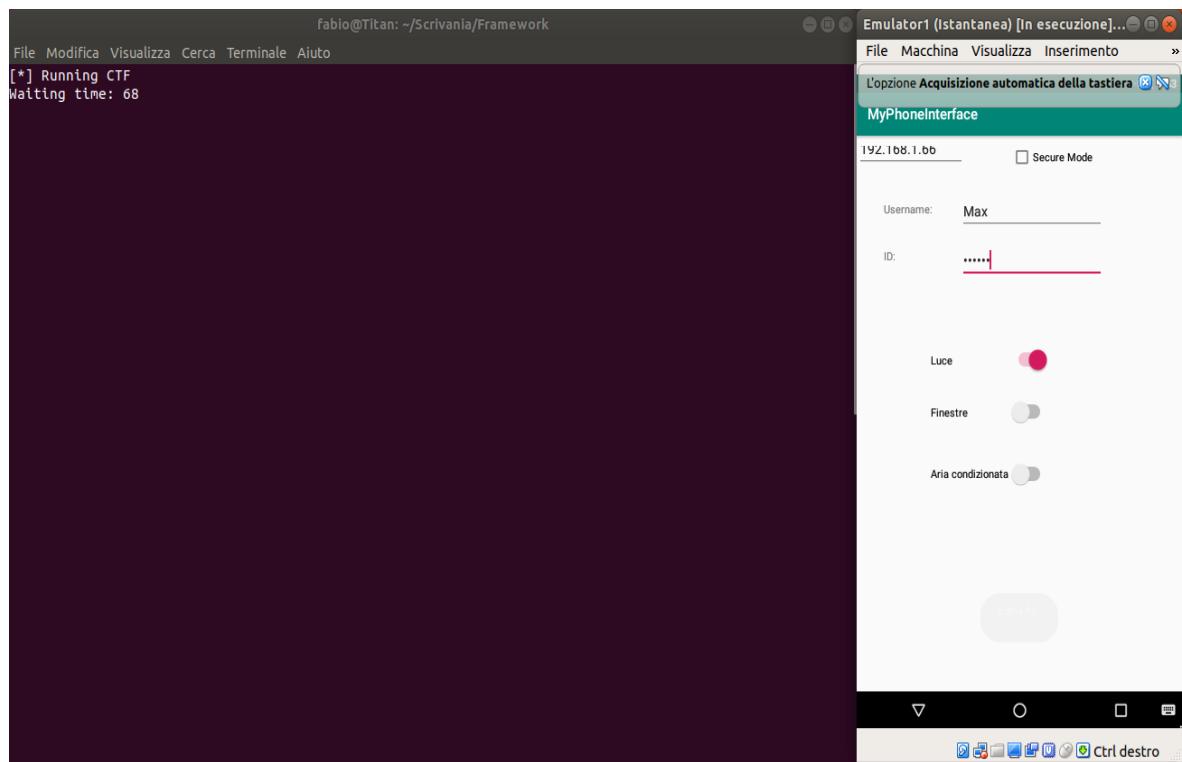


Figura 43: Avvio delle analisi IoT tramite AppIoTTE

Al termine delle analisi il sistema fornisce allo sviluppatore tutte le informazioni relative ai dispositivi presenti nell'ecosistema, con particolare enfasi per il dispositivo di

osservzione scelto. Tutti i dati saranno quindi rappresentati verbalmente nell’interfaccia di comando, con la possibilità di salvarli in un file esterno per un eventuale loro esportazione, ricollegandoli così alle informazioni prodotte durante i processi di analisi statica.

## 8.2 Test

Per mostrare il funzionamento del framework AppIoTTE, prendendo spunto dagli esempi forniti da Google è stato sviluppato un tipico sistema IoT per gestire alcune delle funzionalità della propria Smart Home, direttamente dal proprio cellulare.

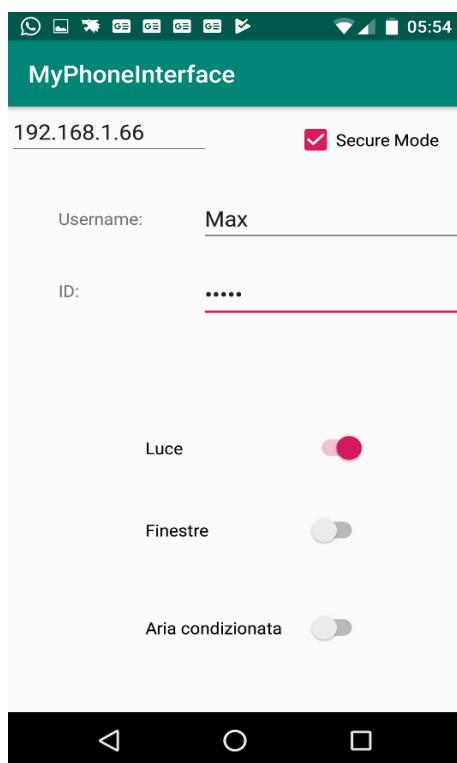


Figura 44: Applicazione di controllo della centralina IoT

È stata quindi realizzata un’applicazione per device Android Things per ricalcare il funzionamento di una centralina smarthome, con l’obiettivo di gestire i vari aspetti tecnologici della propria abitazione (come ad esempio accendere le luci, gestire le finestre o azionare l’aria condizionata) [figura 44]. L’utente può accedere a queste funzionalità tramite un applicazione creata per cellulari Android, dove immettendo il proprio username ed un proprio identificatore, l’utente potrà gestire il tutto a distanza.

Alla pressione di uno degli switch, le informazioni vengono mandate alla centralina, che verificherà che l'utente sia registrato nel proprio database interno ed autorizzato a compiere queste azioni.

In caso affermativo, vengono aperte in uscita le relative porte GPIO, comunicando così l'inizio di quel determinato processo.

Tutti i cambiamenti eseguiti vengono salvati in un file interno alla centralina, in modo da tenere traccia di quanto avvenuto nella propria abitazione

Per poter mostrare con più chiarezza le comunicazioni avvenute tra i due dispositivi, si è deciso di stabilire una comunicazione diretta tra i due, tramite l'indirizzo IP dei dispositivi ed il passaggio delle informazioni tramite metodo GET. Normalmente si sarebbe optato per un trasferimento delle informazioni tramite l'uso di real time database disponibili online, ma ciò non influisce sul funzionamento del framework AppIoTTE, in quanto non sono (per ora) presenti post-elaborazioni sul contenuto di per sé dei segnali inviati.

Sono state avviate quindi le analisi individuali sulle due applicazioni. Dato che il caso d'uso relativo alla gestione della propria smarthome è condiviso tra i due dispositivi, con risorse allocate in posizioni differenti e non accessibili se non connessi tra loro, le analisi dinamiche hanno potuto osservare solamente una frazione del loro funzionamento, non rilevando vulnerabilità di sicurezza di alcun genere.

Durante lo sviluppo di un prodotto informatico è richiesto di analizzare anche il funzionamento condiviso delle varie applicazioni ed ogni possibile caso d'uso. Le analisi, oltre a richiedere aree di competenza diverse tra loro, devono quindi essere effettuate confrontando i dati ottenuti dall'osservazione in parallelo dei dispositivi, correlandoli quindi tra loro per poter unire logicamente quanto sia successo a livello d'infrastruttura.

Questo lavoro viene automatizzato dal prototipo di AppIoTTE sviluppato: sottoponendogli le applicazioni da analizzare, si è stati in grado di individuare i problemi di sicurezza che le affliggevano, eliminando così le superfici di attacco sfruttabili da un'attaccante.

Analizzando il sistema smarthome, la prima vulnerabilità riguardava il controllo delle credenziali dell'utente: normalmente le informazioni contenute all'interno del database presente nel dispositivo IoT sono accessibili solamente agli amministratori, e visualizzabili parzialmente ed attraverso modalità ben definite agli utenti certificati per poter

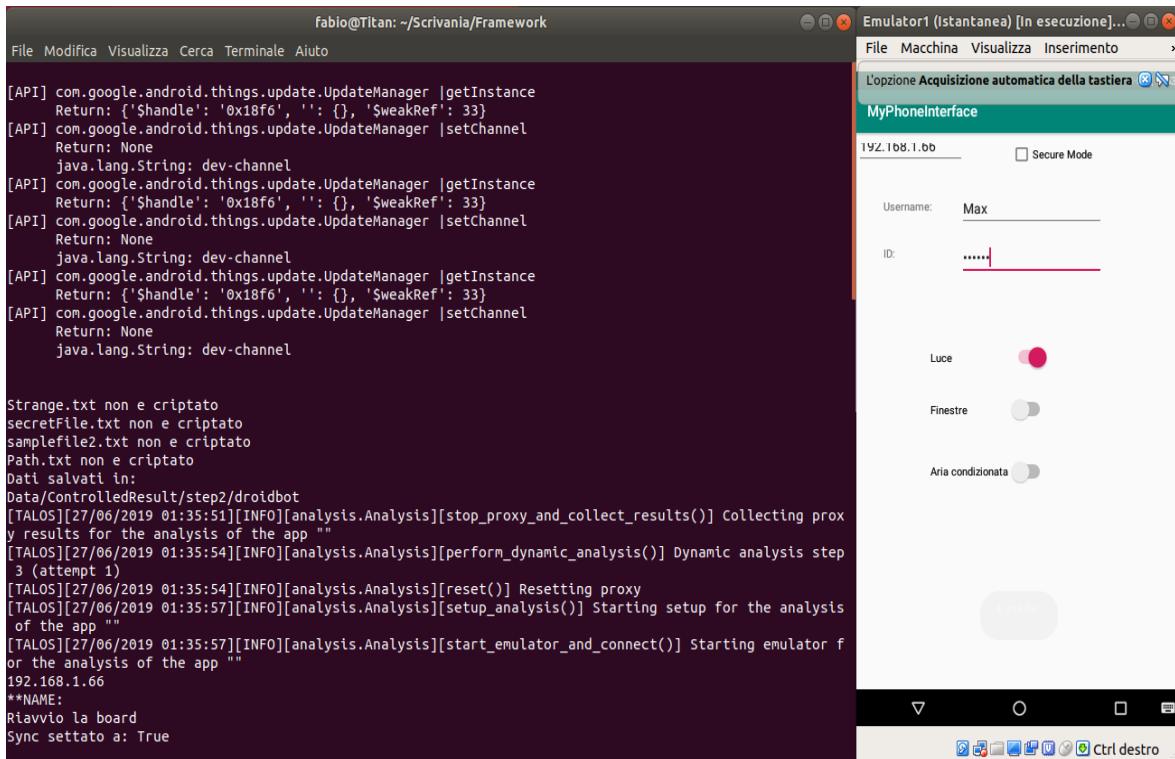


Figura 45: Analisi dell'ecosistema smartHome

proteggere dati ritenuti sensibili. Durante la stimolazione dell'ambiente è stato però notato che, nonostante il dispositivo mobile effettuasse operazioni di sanitizzazione e le comunicazioni tra i dispositivi fossero effettuate con il massimo livello di sicurezza, i tentativi di introdurre delle time-based sql injection nel database avevano dato esito positivo.

Sono quindi seguite le operazioni di correlazione per determinare se l'iniezione di sql nel database fosse correlato con un particolare segnale.

Le operazioni di correlazione hanno rilevato i segnali che hanno scatenato le anomalie nel database, e si è potuto notare che queste erano tutte riconducibili alla richiesta di apertura delle finestre dal cellulare, introducendo nel campo ID (rappresentato dal campo password nel segnale) l'iniezione di comandi sql. Si è quindi trovato un errore nelle operazioni di sanitizzazione effettuate durante quel particolare caso d'uso, risolvibile effettuando operazioni di controllo più serrate sulle query generate dal sistema.

Una seconda vulnerabilità è stata riscontrata in merito alle comunicazioni con ter-

```

fabio@Titan:~/Scrivania/Reasoner$ python3 legame-segnale-anomalia.py
['1561504479955', 'SQLiteDatabase |SELECT * FROM Users WHERE (Username="123456" AND Password=22-IF(''5''='5\',SLEEP(15),0));-- -');
['1561504493263', 'SQLiteDatabase |SELECT * FROM Users WHERE (Username="6" AND Password=14-IF(''1''='1\',SLEEP(15),0));-- -');
['1561504521382', 'SQLiteDatabase |SELECT * FROM Users WHERE (Username="10" AND Password=14-IF(''1''='1\',SLEEP(15),0));-- -');
['1561504536847', 'SQLiteDatabase |SELECT * FROM Users WHERE (Username="12" AND Password=14-IF(''1''='1\',SLEEP(15),0));-- -');
['1561504563227', 'SQLiteDatabase |SELECT * FROM Users WHERE (Username="15" AND Password=14-IF(''1''='1\',SLEEP(15),0));-- -');

-----
-----Correlazione con DB anomaly-----

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=window&username=15&password=14-IF('1'='1',SLEEP(15),0));--%20-&secureConnection=YES |
-----> anomalia SQLiteDatabase |SELECT * FROM Users WHERE (Username="15" AND Password=14-IF('1'='1\',SLEEP(15),0));-- - );

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=window&username=12&password=14-IF('1'='1\',SLEEP(15),0));--%20-&secureConnection=YES |
-----> anomalia SQLiteDatabase |SELECT * FROM Users WHERE (Username="12" AND Password=14-IF('1'='1\',SLEEP(15),0));-- - );

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=window&username=10&password=14-IF('1'='1\',SLEEP(15),0));--%20-&secureConnection=YES |
-----> anomalia SQLiteDatabase |SELECT * FROM Users WHERE (Username="10" AND Password=14-IF('1'='1\',SLEEP(15),0));-- - );

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=window&username=6&password=14-IF('1'='1\',SLEEP(15),0));--%20-&secureConnection=NO |
-----> anomalia SQLiteDatabase |SELECT * FROM Users WHERE (Username="6" AND Password=14-IF('1'='1\',SLEEP(15),0));-- - );

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=window&username=123456&password=22-IF('5' = '5', SLEEP(15), 0)); -- &secureConnection=NO |
-----> anomalia SQLiteDatabase |SELECT * FROM Users WHERE (Username="123456" AND Password=22-IF('5'='5\',SLEEP(15),0));-- - );

```

Figura 46: Risultati correlazione DB

ze parti. La centralina, oltre a salvare i dati raccolti su un file interno non codificato, effettuava comunicazioni attraverso protocolli non sicuri con un altro dispositivo IoT, inviando nella trasmissione le informazioni sensibili dell’utente.

Questo comportamento, precedentemente non rilevato durante le analisi individuali, si è scoperto essere causato dall’assenza della modalità sicura inviando il comando di accendere le luci da cellulare; è stato quindi risolto imponendo un protocollo di comunicazione sicura per ogni trasmissione tra i dispositivi.

```

fabio@Titan:~/Scrivania/Reasoner$ python3 legame-segnale-anomalia.py
['1561504455882', '[192.168.1.20] http://192.168.1.20:5284/?switch=true&username=2&password=2']
['1561504466345', '[192.168.1.20] http://192.168.1.20:5284/?switch=false&username=3&password=123456']
['1561504474267', '[192.168.1.20] http://192.168.1.20:5284/?switch=true&username=4&password=123456']
['1561504509910', '[192.168.1.20] http://192.168.1.20:5284/?switch=false&username=8&password=123456']
['1561504515973', '[192.168.1.20] http://192.168.1.20:5284/?switch=true&username=9&password=123456']

-----
-----Correlazione con unsecure communications anomaly-----

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=light&status=true&username=9&password=123456&secureConnection=NO |
-----> anomalia [192.168.1.20] http://192.168.1.20:5284/?switch=true&username=9&password=123456

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=light&status=false&username=8&password=123456&secureConnection=NO |
-----> anomalia [192.168.1.20] http://192.168.1.20:5284/?switch=false&username=8&password=123456

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=light&status=true&username=4&password=123456&secureConnection=NO |
-----> anomalia [192.168.1.20] http://192.168.1.20:5284/?switch=true&username=4&password=123456

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=light&status=false&username=3&password=123456&secureConnection=NO |
-----> anomalia [192.168.1.20] http://192.168.1.20:5284/?switch=false&username=3&password=123456

CORRELAZIONE --> segnale [192.168.1.66] https://192.168.1.66:5283/?instance=light&status=true&username=2&password=2&secureConnection=NO |
-----> anomalia [192.168.1.20] http://192.168.1.20:5284/?switch=true&username=2&password=2

```

Figura 47: Risultati correlazione comunicazioni non protette

Nel complesso, i risultati ottenuti sono stati soddisfacenti, riuscendo ad individuare tutte le vulnerabilità di sicurezza di cui le applicazioni erano soggette.

A posteriori, le operazioni di correlazione hanno mostrato una percentuale di accuratezza del 91%, merito il fatto che i processi istanziati nei dispositivi non richiedevano un grande onere computazionale, venendo terminati in tempi molto brevi.

Il restante 9% di accuratezza persa non è stato attribuito ad errori nel processo di correlazione, ma dal mancato filtraggio di alcune operazioni effettuate in maniera autonoma dalla centralina (non causate da alcuno stimolo da cellulare).

Tutte le informazioni sono quindi raccolte in appositi files, catalogati per settore, pronti per essere analizzati dallo sviluppatore per riconoscere quanto è successo durante i test eseguiti.

## 9 Conclusioni e Sviluppi futuri

---

L'obiettivo della tesi è stato quello di studiare dal punto di vista della sicurezza un ecosistema complesso come quello mobile-IoT, focalizzandosi sul layer applicativo.

Inizialmente la metodologia era basata sulle tecniche di analisi proposte e consolidate nel framework OWASP; i risultati prodotti dalle analisi dei singoli dispositivi IoT, a prima vista molto promettenti, si sono rivelati falsati dalla mancanza di uno studio globale, condiviso a livello di ecosistema. La vera sfida è stata quindi lo sviluppo di una metodologia che permetesse l'unione dei risultati delle analisi di dispositivi differenti, separando quanto effettivamente causato dal device in sè dalle azioni generate da stimoli esterni. Dato che un'applicazione può avere infiniti comportamenti differenti, a seconda del proprio stato e dalle azioni che lo hanno preceduto (e complice la carenza di applicazioni IoT su cui basare l'apprendimento) non è stato possibile addestrare il reasoner attraverso tecniche di machine learning. Sono stati quindi sviluppati algoritmi che permettessero di ottenere risultati analoghi, partendo dall'osservazione del comportamento dei dispositivi prima separati dall'ecosistema, e poi connessi. Si è quindi implementato quanto studiato in un framework, chiamato AppIoTTE, in grado di unire tecnologie eterogenee ma molto simili come Android ed Android Things. I risultati ottenuti hanno potuto confermare la bontà di quest'approccio, rilevando superfici di attacco scoperte e segnalando il processo che l'ha generata con una percentuale di accuratezza superiore all'81%, aumentando notevolmente qualora i dispositivi si rivellino essere molto performanti.

## Sviluppi Futuri

Questo lavoro di Tesi lascia spazio a diversi sviluppi futuri. I più importanti sono certamente la raffinazione degli algoritmi di reasoning ed un miglioramento delle operazioni di filtraggio, in modo da migliorare il confronto tra quanto avvenuto autonomamente nel dispositivo IoT e quanto causato in seguito allo stimolo dell'ecosistema; il miglioramento di questi due punti mira ad aumentare la percentuale di accuratezza ottenuta dalle operazioni di correlazione, chiave fondamentale per lo studio dell'ambiente IoT.

Molto interessante sarebbe l'unione dei diversi Inter-procedural Control Flow Graph generati dalle analisi statiche sui diversi dispositivi, che potrebbe portare ad un aumento della precisione del reasoner tramite l'eliminazione delle anomalie generate spontaneamente dai devices.

L'abbandono del supporto ad Android Things da parte di Google costringe però a rivisitare le modalità di analisi, riadattandole per un sistema operativo Linux, il più diffuso tra i devices IoT attuali, prima di poter rilasciare AppIoTTE come strumento completo di analisi.

## Riferimenti bibliografici

---

- [1] I. K. Chin Yin Leong, “Internet of Things (IoT) for Dynamic Change Management in Mass Customization,” 2017.
- [2] K. A. Chris McLaren, Gordon Archibald, “Security and the IoT ecosystem ,” 2016.
- [3] I. Skerrett, “IoT Developer Survey Result.” [Online]. Available: <https://ianskerrett.wordpress.com/2017/04/19/iot-developer-trends-2017-edition/>
- [4] “Owasp IoT Top 10 vulnerabilities.” [Online]. Available: [https://www.owasp.org/index.php/OWASP\\_Internet\\_of\\_Things\\_Project](https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project)
- [5] O. A. A.-R. S. Y. J. Jacob Wurm, Khoa Hoang, “Security Analysis on Consumer and Industrial IoT Devices,” *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016.
- [6] L. P. P. C. Zoran Cekerevac, Zdenek Dvorak, “Internet Of Things and the Man-In-The-Middle Attacks – security and economic risks ,” *IEEE International Workshop on Factory Communication Systems*, 2017. [Online]. Available: [http://mest.meste.org/MEST\\_2\\_2017/10\\_03.pdf](http://mest.meste.org/MEST_2_2017/10_03.pdf)
- [7] O. A. A.-R. S. Y. J. Jacob Wurm, Khoa Hoang, “Security Analysis on Consumer and Industrial IoT Devices.”
- [8] “Android Things,” 2018. [Online]. Available: <https://partner.android.com/things/console/?pli=1#/tools>
- [9] “Android Things, Supported hardware,” 27/7/2018. [Online]. Available: <https://developer.android.com/things/hardware>
- [10] “Differences between Android Things and Android,” 27/7/2018. [Online]. Available: <https://developer.android.com/things/get-started/platform-differences#apis>
- [11] T. Group, “Thread Stack Fundamental,” 13/7/2015. [Online]. Available: [https://www.threadgroup.org/Portals/0/documents/support/ThreadOverview\\_633\\_2.pdf](https://www.threadgroup.org/Portals/0/documents/support/ThreadOverview_633_2.pdf)

- [12] T. G. of the Hong Kong Special Administrative Region, “IPv6 SECURITY,” May 2011. [Online]. Available: <https://www.infosec.gov.hk/english/technical/files/ipv6s.pdf>
- [13] O. Group, *Mobile Security Testing Guide (MSTG)*, 09/17/2018. [Online]. Available: <https://github.com/OWASP/owasp-mstg/releases>
- [14] G. Sims, “What is Android Things? – Gary explains,” January 13, 2017. [Online]. Available: <https://www.androidauthority.com/what-is-android-things-gary-explains-740572/>
- [15] O. Group, “Static Code Analysis.” [Online]. Available: [https://www.owasp.org/index.php/Static\\_Code\\_Analysis](https://www.owasp.org/index.php/Static_Code_Analysis)
- [16] Y. C. Vaibhav Rastogi and X. Jiang, “Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks,” *IEEE Transactions on Information Forensics and Security*, vol. 9, 2014. [Online]. Available: <http://pages.cs.wisc.edu/~vrastogi/static/papers/rcj14.pdf>
- [17] F. D. T. Ankita Kapratwar and M. Stamp, “Static and Dynamic Analysis of Android Malware .” [Online]. Available: [https://www.researchgate.net/publication/314521542\\_Static\\_and\\_Dynamic\\_Analysis\\_of\\_Android\\_Malware](https://www.researchgate.net/publication/314521542_Static_and_Dynamic_Analysis_of_Android_Malware)
- [18] O. Group, *OWASP Mobile Application Security Verification Standard (MASVS)*. [Online]. Available: <https://github.com/OWASP/owasp-masvs>
- [19] T. srls, “<https://www.talos-sec.com>.” [Online]. Available: <https://www.talos-sec.com>
- [20] Talos S.r.l.s. (2016) Approver. [Online]. Available: <http://www.talos-security.com/>
- [21] B.Gruver, “Smali – Assembler/Disassembler for the dex format.” [Online]. Available: <http://github.com/JesusFreke/smali/>
- [22] Virtualbox. [Online]. Available: <https://www.virtualbox.org/>
- [23] Frida. [Online]. Available: <https://www.frida.re/docs/home/>
- [24] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors, “mitmproxy: A free and open source interactive HTTPS proxy,” 2010–, [Version 4.0]. [Online]. Available: <https://mitmproxy.org/>

- [25] Y. Li, Z. Yang, Y. Guo, and X. Chen, “DroidBot: A lightweight UI-guided testinput generator for android,” *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, 2017. [Online]. Available: [https://www.researchgate.net/publication/314521542\\_Static\\_and\\_Dynamic\\_Analysis\\_of\\_Android\\_Malware](https://www.researchgate.net/publication/314521542_Static_and_Dynamic_Analysis_of_Android_Malware)

## Ringraziamenti

---

Dopo dieci lunghi e intensi mesi, finalmente il giorno è arrivato: scrivere queste frasi di ringraziamento è il tocco finale della mia tesi. È stato un periodo di profondo apprendimento, non solo a livello scientifico, ma anche personale. Vorrei quindi spendere due parole di ringraziamento nei confronti di tutte le persone che mi hanno sostenuto e aiutato durante questo periodo.

Prima di tutto, vorrei ringraziare il mio relatore, il Prof. Alessio Merlo, ed il mio correlatore, il Dott. Luca Verderame, per tutto l'aiuto che mi hanno fornito in questi mesi. Mi avete fornito tutti gli strumenti di cui avevo bisogno per intraprendere la strada giusta e portare a compimento la mia tesi. Senza di voi tutto questo non sarebbe stato possibile.

Un ringraziamento particolare ad i miei amici e colleghi universitari Davide, Fuzio, Mirko, Martina e Gabriel, persone stupende che mi sento veramente fortunato di avere conosciuto. Ci siamo sempre sostenuti a vicenda, nella buona e nella cattiva sorte, sia durante le fatiche e lo sconforto che hanno caratterizzato il nostro percorso, sia nei momenti di gioia e soddisfazione fino al raggiungimento del traguardo. Grazie per aver condiviso con me questi fantastici cinque anni.

Vorrei infine ringraziare mia madre, mio padre e Roberto, i miei punti di riferimento che mi hanno sostenuto sia economicamente che emotivamente durante questi anni, permettendomi di percorrere e concludere questo cammino. Grazie per avermi supportato, consigliato e guidato, per aver sempre creduto in me sia durante i successi, sia durante le difficoltà. Senza di voi tutto questo non sarebbe stato possibile.

Grazie a tutti voi!