UNIVERSITÀ
DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA,
ELETTRONICA E SISTEMISTICA

Corso di Laurea in Ingegneria Informatica

Analisi delle vulnerabilità di un'applicazione per il trasporto pubblico

Relatore:

Prof.
Gianluigi Folino

Candidato:

Domenico Femia
Mat. 236916

ANNO ACCADEMICO 25/26

Ringraziamenti

Ladies and gentlemen, we did it! Finalmente ci siamo, con un po' di ritardo e qualche intoppo lungo la via, ma ce l'abbiamo fatta. Se sto (o se state) leggendo queste righe, vuol dire che il giorno della laurea è arrivato, segnando la fine di un percorso, e che percorso! Lungo questa strada ci sono stati numerosi cambi di rotta, ma verso la fine possiamo dire che abbiamo trovato un buon punto di assestamento.

Per non rendere vano tutto questo sforzo ci sono numerose persone da ringraziare. In primis la mia famiglia, che tra mille preoccupazioni e un sostegno inesauribile mi ha portato alla fine di questo percorso tutt'altro che facile. Grazie Papà e Mamma; senza di voi non sarebbe stato possibile ottenere tutto ciò. Un ringraziamento enorme va a mia sorella Roberta, la mia complice numero uno dal primissimo giorno di università. Senza la 'principespessa', arrivare a questo traguardo tutti integri sarebbe stato semplicemente impossibile.

Vi sono stati altri complici in questa avventura, ovviamente: chi più vicino e chi più lontano, chi a Torino, chi a Cosenza e chi giù nella "piana". Un grandissimo grazie ai miei amici, i campioni delle superiori e i campioni dell'università, che sono diventati come una seconda famiglia. Grazie per esserci stati, tra lo studio matto e disperato, lo svago e per tutto ciò al di fuori delle questioni universitarie. Se siamo arrivati qui sani e salvi, è anche merito vostro.

Un ringraziamento speciale anche ad Hacklab e a Stefano; molto probabilmente, senza la loro spinta, questa tesi non sarebbe stata così speciale a livello di contenuti e di dialettica come lo è attualmente!

Ultimo, ma non per importanza, un grazie a me stesso. Per aver tenuto duro e per essere arrivato finalmente a questo grandissimo traguardo, cosa decisamente non da poco!

Indice

Capitolo 1

Introduzione

Nel contesto odierno, in cui gli utenti fanno un uso sempre maggiore di servizi digitali accessibili attraverso le relative applicazioni (*client*), cresce di pari passo l'importanza della sicurezza informatica nel proteggere la mole di dati personali o sensibili che transitano attraverso questi servizi. Uno degli aspetti più complessi quando si parla di sicurezza delle applicazioni è proprio il metro di giudizio che viene utilizzato per quest'ultima; esiste, di fatto, una malriposta *presunzione di sicurezza* dei prodotti software, che decade unicamente in seguito a un'analisi che ne dimostri l'insicurezza. A mettere in discussione questa concezione è stato Bruce Schneier, crittografo di fama mondiale, con questa riflessione:

> "One of the biggest conceptual problems we have is that something is believed secure until demonstrated otherwise. We need to reverse that: everything should be believed insecure until demonstrated otherwise."[1]

Questa prospettiva ribalta completamente i requisiti necessari affinché un'app possa essere considerata sicura; applicare questo principio vuol dire, infatti, per gli sviluppatori, adottare un approccio *Security by Design*, mettendo al centro la sicurezza della propria applicazione fin dalle prime fasi di sviluppo e predisporsi in modo adeguato a segnalazioni esterne di problematiche che possano impattarla, già a partire dal primo rilascio.
Lo scopo di questo lavoro di tesi è presentare un'analisi di sicurezza approfondita (*Security Assessment* e *Penetration Testing*) su un'applicazione mobile reale sviluppata per la gestione di un servizio di trasporto pubblico a chiamata. L'obiettivo è valutare la sicurezza dell'intera infrastruttura, sia analizzando la parte client che la parte backend. Questo avviene effettuando analisi statiche e dinamiche sul codice, individuando problemi di configurazione e difetti di progettazione, per poi verificare l'aderenza alle *best practice* del settore.

1.1 Motivazioni

La scelta di incentrare questo elaborato su un sistema esistente di mobilità locale è stata motivata, in primo luogo, dalla mia personale attitudine generale verso il reverse engineering e dalla volontà di comprendere nel dettaglio i sistemi informatici e il loro operato, sia da un più specifico interesse per l'area *Information Security* (InfoSec).
Essendo io stesso uno studente e un assiduo fruitore del servizio di mobilità pubblica in questione, mi è sembrato importante analizzare la sicurezza della piattaforma al fine di rimuovere le forti perplessità che l'esperienza d'uso dell'applicativo, in qualità di utente, mi aveva lasciato fin dall'inizio e di proporre possibili miglioramenti a beneficio di tutti gli utenti finali.
Un'ulteriore motivazione che ha reso significativa l'analisi dell'app è stata la volontà da parte dell'associazione Hacklab Cosenza, di cui faccio parte, di integrare all'interno della nuova versione del tabellone digitale (*digital signage*) installato presso le pensiline dell'Università le informazioni sulla posizione in tempo reale dei mezzi in servizio disponibili all'interno dell'app oggetto di analisi.

1.2 Struttura della Tesi

Il lavoro è organizzato come segue:
Nel Capitolo 2 vengono presentate le principali tecnologie, i componenti essenziali e i concetti teorici utilizzati per la realizzazione di un'applicazione mobile moderna, con particolare attenzione alla creazione di un sistema che implementa un servizio di trasporto a chiamata. Vengono descritte la classica architettura client-server e le peculiarità dei framework adottati.
Nel Capitolo 3 viene descritto il sistema in esame e viene introdotta la metodologia di analisi applicata, con l'illustrazione delle fasi di analisi statica e dinamica del codice. I risultati vengono confrontati prendendo a riferimento le linee guida per il testing e la valutazione della sicurezza del framework OWASP MASVS.
Nel Capitolo 4 il focus si sposta sul backend dell'applicazione, del quale viene documentata la fase di ricerca delle risorse a esso collegate, degli endpoint e delle interfacce presenti. Vengono evidenziate le gravi problematiche riscontrate sulla validazione degli input, sulla configurazione dei servizi, sulla gestione dei permessi utente e sulle scelte progettuali dell'implementazione delle RESTful API presenti nel backend.
Nel Capitolo 5 l'attenzione si sposta sull'analisi del client mobile del sistema. In questa sezione vengono discusse alcune buone pratiche implementate nel-

l'applicazione, come l'offuscamento del codice e l'utilizzo del *certificate pinning* per la protezione del traffico di rete, alle quali si contrappongono alcune scelte problematiche quali il salvataggio in chiaro (*insecure local storage*) dei dati dell'utente e la gestione non corretta della procedura di logout.
Nel Capitolo 6 lo studio assume una prospettiva più aziendale. Vengono discussi gli aspetti legati alla struttura organizzativa in ambito di sicurezza, affrontando la questione della *Vulnerability Disclosure*, effettuata utilizzando come fonte di riferimento lo standard ISO, dove vengono illustrate le best practice per la ricezione e la gestione di segnalazioni da ricercatori esterni su problematiche di sicurezza.
Infine, nel Capitolo 7 vengono presentate le conclusioni del lavoro e alcune possibili direzioni per sviluppi futuri.

Capitolo 2

Tecnologie Utilizzate

In questa parte vengono presentate le principali tecnologie utilizzate per lo sviluppo e la gestione di un sistema software per un servizio di trasporto a chiamata oggetto del presente lavoro. Questo tipo di applicazioni sono strutturate seguendo un'architettura a due livelli: una parte client, costituita da un'app per dispositivi mobili con la quale gli utenti finali, sia guidatori che passeggeri, si interfacciano con il sistema; una parte backend che gestisce l'autenticazione degli utenti, le prenotazioni e i relativi cambiamenti di stato, nonché la navigazione per le varie fasi del viaggio. A supporto di questi componenti possiamo trovare anche una serie di servizi infrastrutturali, tra cui un motore di routing geografico e interfacce di gestione amministrativa, utilizzando tecnologie già presenti sul mercato.

2.1 Frontend

2.1.1 React Native

React Native è un framework open source sviluppato inizialmente da Meta (allora nota come Facebook) e rilasciato nel 2015, attualmente gestito dalla Linux Foundation, React è progettato per semplificare la creazione di applicazioni mobili multipiattaforma a partire da un'unica base di codice scritta in JavaScript o TypeScript [2]. Attraverso l'utilizzo di meta-elementi, definiti tramite il linguaggio JavaScript, il framework li traduce in componenti nativi per il sistema operativo di destinazione (Android o iOS). Il codice JavaScript di tutti questi moduli viene impacchettato in un unico file denominato `index.android.bundle`, destinato al sistema operativo Android; è questo file ad essere poi inserito all'interno del pacchetto APK, per poi essere eseguito a runtime dal motore JavaScript dell'applicazione.

2.1.2 Hermes

Hermes è un motore JavaScript open source sviluppato da Meta, ottimizzato specificatamente per eseguire le applicazioni React Native [3]. A differenza dei motori JavaScript tradizionali, Hermes non esegue il codice o lo compila *Just In Time*, ma genera un bytecode del codice *Ahead Of Time*; il file `index.android.bundle` in questo caso contiene proprio il bytecode derivato dall'elaborazione.
Questo approccio porta numerosi vantaggi, tra cui un aumento delle prestazioni, una riduzione del consumo di memoria e una dimensione ridotta dell'applicativo finale. Diversamente dalla sua controparte Just In Time, il bytecode Hermes risulta molto difficile da analizzare e leggere, obbligando gli analisti all'utilizzo di sistemi di decompilazione, come discusso anche nella Sezione 3.4.

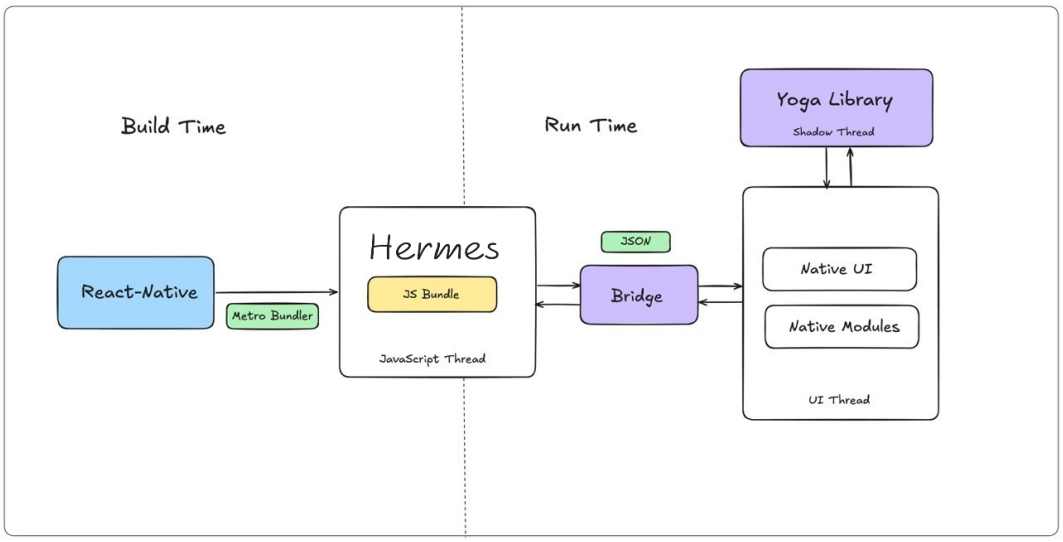

Figura 2.1: Rappresentazione flussi Build Time e Run Time nell'architettura a Bridge (rielaborazione propria da [4]).

2.2 Backend

2.2.1 Django

Django è un framework web per lo sviluppo di backend in Python, pubblicato nel 2005. Il suo scopo principale è permettere, grazie a tutta una serie di strumenti integrati la prototipazione rapida di applicazioni web. L'aspetto più rilevante agli scopi di questa tesi è che nei componenti definiti di Django vengono implementate misure di sicurezza contro le vulnerabilità web più comuni. In particolare, possiamo menzionare varie mitigazioni riguardanti attacchi come Cross-Site Request Forgery (CSRF), SQL Injection, Cross-Site Scripting (XSS) e Clickjacking; l'inclusione in modo predefinito di queste con-

tromisure permette di ridurre i rischi di sicurezza. L'architettura di Django si basa sul pattern Model-View-Template (MVT), una variante del pattern Model-View-Controller (MVC).

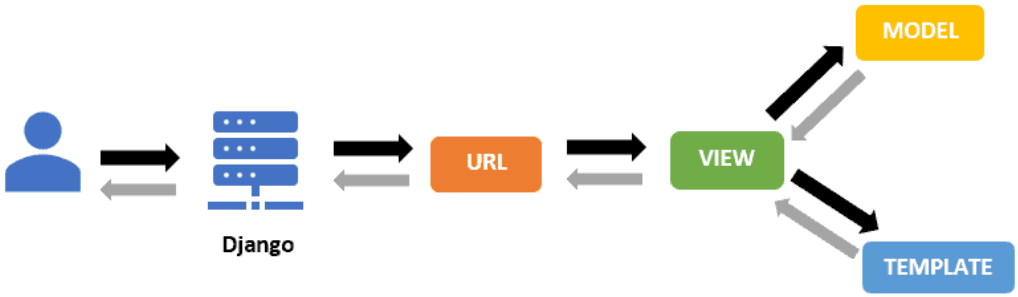

Figura 2.2: Schema dell'architettura Model–View–Template che caratterizza il framework Django, e che mostra l'interazione tra Model, View e Template nella gestione delle richieste dell'utente [5].

Il Model serve per interfacciarsi con il database e gestire la logica di business legata ai dati; la View viene utilizzata per elaborare le richieste HTTP ricevute, delegando le modifiche sui dati al Model e la rappresentazione al Template; il Template presenta i dati nel formato previsto, HTML per un pagina Web e JSON nel caso di una risposta ad API RESTful.

2.2.2 Django REST Framework

Django REST Framework è una libreria che estende le funzionalità di Django permettendo lo sviluppo di API web RESTful. Per impostazione predefinita l'ambiente di sviluppo è configurato per generare contenuti utili solo a browser web, come per esempio pagine HTML, fogli di stile CSS e script JavaScript. Django REST Framework permette di creare servizi che rispondono a chiamate HTTP con dati in formato JSON, che è il formato standard nelle applicazioni moderne che fanno uso di API RESTful. L'utilizzo di questo toolkit risulta particolarmente importante nelle architetture moderne poiché l'accesso ai dati avviene da molteplici sistemi; in larga parte si parla di applicazioni destinate a dispositivi mobili. Esso quindi svolge il ruolo di layer API tra il client e il resto del codice backend di Django, gestendo tutta la parte di verifica dei dati, l'autenticazione degli utenti e molto altro.

2.2.3 Django Channels e Daphne

Django è molto efficace nel gestire le comunicazioni che fanno uso del protocollo HTTP, che è lo standard nelle applicazioni web. Tuttavia, il protocollo non è sempre efficace per comunicazioni di dimensioni ridotte e con elevata

frequenza di invio dei dati, perché ciò comporterebbe un eccessivo overhead introdotto dagli header HTTP, i quali finirebbero per pesare più del payload vero e proprio della comunicazione (si pensi, per esempio, all'invio di dati per la geolocalizzazione in tempo reale dei veicoli). Per risolvere queste problematiche in comunicazioni di questo genere si fa uso del protocollo WebSocket, che permette di instaurare una connessione persistente e di scambiare dati con un overhead ridotto rispetto alle chiamate HTTP. Poiché Django non presenta il supporto predefinito a comunicazioni con questo protocollo, il supporto viene aggiunto con le librerie Daphne e Django Channels.

2.2.4 drf-yasg

drf-yasg è una libreria Python il cui scopo principale è generare automaticamente la documentazione delle API RESTful a partire dalle view definite con Django REST Framework.
La libreria rilascia una descrizione delle API utilizzando lo standard OpenAPI [6], una specifica ampiamente utilizzata nel panorama dei servizi web. Tale standard utilizza il formato JSON per descrivere in modo dettagliato e formale gli endpoint disponibili, i metodi HTTP supportati, i parametri richiesti e il formato dei dati accettati o restituiti dalle API.
Oltre a creare il file *openapi.json*, che non sarebbe altro che il file contenente la descrizione completa delle API, drf-yasg permette di esplorare facilmente la documentazione tramite due interfacce web. In particolare, la documentazione può essere visionata utilizzando Swagger, all'endpoint */swagger/*, oppure tramite ReDoc, all'endpoint */redoc/*. Queste interfacce permettono di consultare e testare le API in modo rapido e accessibile, senza avere accesso diretto al codice sorgente originale.

2.2.5 Autenticazione JWT

Il sistema di autenticazione è implementato tramite l'utilizzo del JSON Web Token (JWT), uno standard definito dalla Internet Engineering Task Force con la RFC 7519 e successivi aggiornamenti. Il token è codificato utilizzando Base64 e ha una struttura in tre sezione con il punto come carattere separatore: l'header, che contiene il tipo di token e l'algoritmo utilizzato per la firma, il payload, che include l'identificativo dell'utente, la data di scadenza del token e la data di rilascio del token, e l'ultima parte, che contiene la firma calcolata con l'algoritmo specificato nell'header.
Nel sistema di trasporto a chiamata analizzato, l'algoritmo utilizzato per la firma è *HS256* (HMAC-SHA256), dove viene usata una chiave simmetrica sia per la firma che per la verifica di quest'ultima, la durata degli *access token*

è di 15 minuti mentre quella dei *refresh token* è di 31 giorni. L'intero flusso è gestito in automatico utilizzando la libreria `Simple JWT` di Django REST Framework.

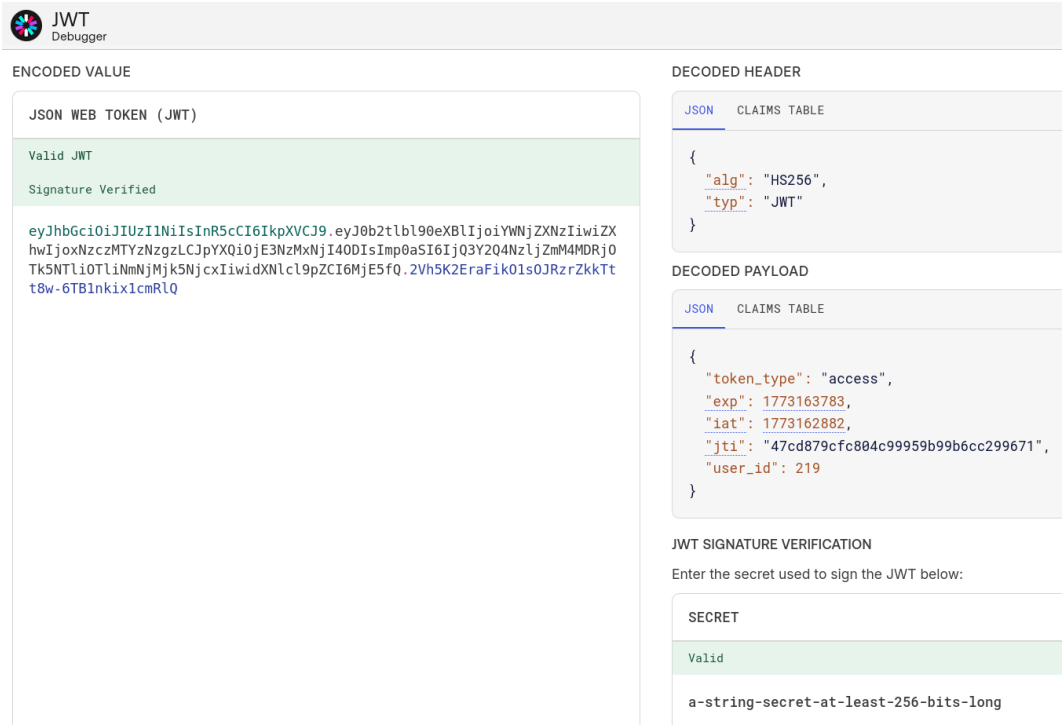

Figura 2.3: Esempio di struttura e decodifica di un JSON Web Token tramite il debugger di jwt.io [7].

2.3 Database

2.3.1 PostgreSQL

PostgreSQL è un Database Management System open source sviluppato come evoluzione del progetto Ingres dell'University of California, Berkeley. È noto per la sua affidabilità, robustezza e conformità agli standard SQL. Uno dei punti di forza di PostgreSQL è il suo supporto a strutture dati avanzate e la possibilità di espandere le sue funzionalità tramite estensioni. In particolare, nei progetti di trasporto condiviso si fa uso di una struttura dati complessa chiamata JSONField, che permette di salvare dati in formato JSON. Il tutto è integrabile con Django tramite il suo Object-Relational Mapping (ORM), che permette di effettuare operazioni di lettura e scrittura all'interno del database, semplificando notevolmente l'interazione del backend con il database.

2.3.2 SQLite

SQLite è una libreria scritta in C che implementa un database relazionale leggero e autonomo, nel quale tutti i dati sono archiviati in un singolo file locale, senza necessità di comunicazioni di rete verso un database server dedicato [8]. È largamente utilizzato in contesti simili come database di recupero (failover): in caso di malfunzionamento del database principale PostgreSQL, framework come Django possono essere impostati per reindirizzare automaticamente delle operazioni al file *db.sqlite3*, mitigando la perdita di dati e garantendo il continuo funzionamento del servizio.

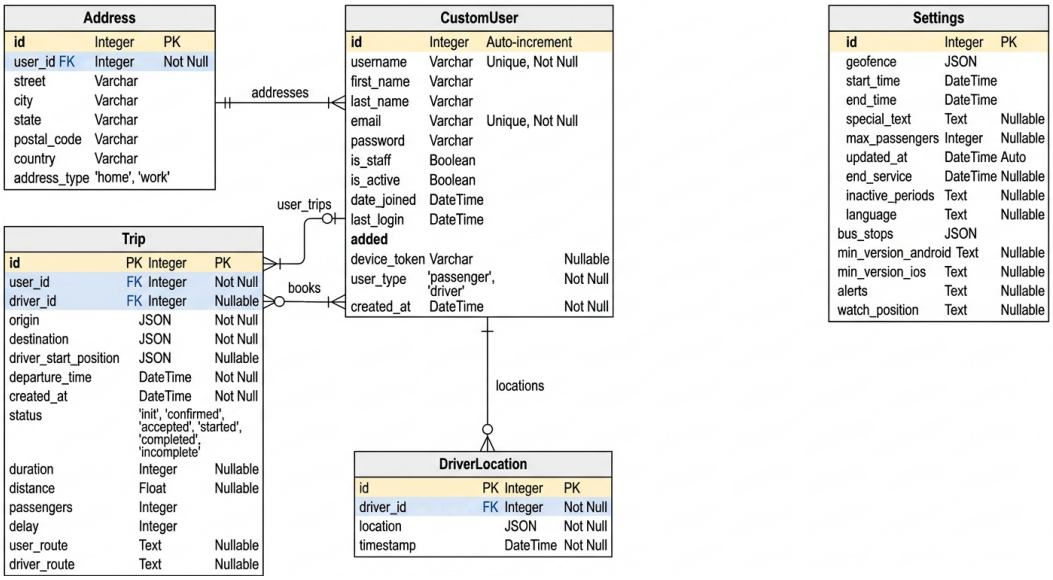

Figura 2.4: Schema Entità–Relazione del database per l'applicazione presa in esame, ottenuto tramite l'accesso al backend descritto nel Capitolo 4. Il diagramma rappresenta la struttura logica dei dati e le relazioni tra le entità del sistema.

2.4 Infrastruttura Server

2.4.1 Apache HTTP Server

Apache HTTP Server è un server web open source sviluppato dalla Apache Software Foundation e rilasciato nel 1995. Esso non è altro che un demone che viene eseguito sul server in background, e che resta in ascolto delle richieste HTTP provenienti dai client e risponde fornendo le risorse corrispondenti. Quando interrogato, Apache può restituire direttamente le varie risorse, come ad esempio file statici o pagine HTML, oppure agire da *reverse proxy* e inoltrare la richiesta al backend Django REST Framework. Nella configurazione in produzione riscontrata nell'applicazione in esame, Apache è stato utilizzato

per servire gli asset statici e ridurre il carico sul backend Django, migliorando di conseguenza l'efficienza del sistema.

2.4.2 Plesk

Plesk è un pannello di controllo che permette di amministrare un server web e i suoi vari servizi, in modo facile e tramite un'interfaccia grafica. Grazie a Plesk si possono gestire e configurare componenti quali il server web Apache HTTP Server, database PostgreSQL, servizi di posta elettronica e domini. Largamente diffuso, esso è utilizzato per facilitare la gestione e la configurazione dei servizi necessari per il funzionamento dell'applicazione.

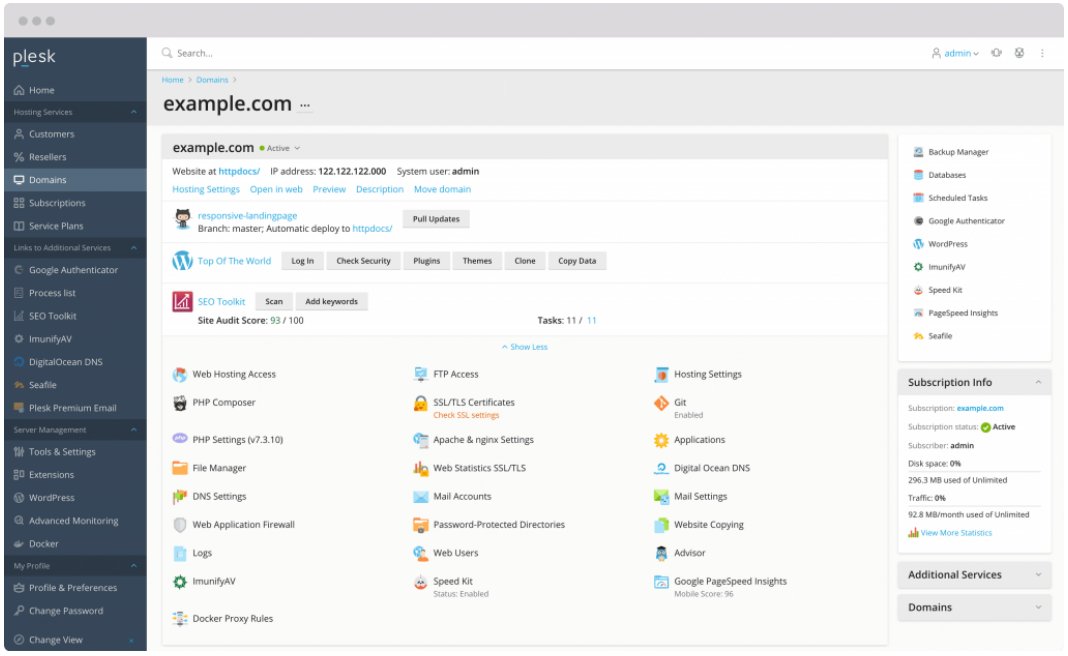

Figura 2.5: Interfaccia grafica del pannello di controllo Plesk utilizzata per amministrare i servizi dell'infrastruttura server.

2.5 Servizi Complementari

2.5.1 Firebase Cloud Messaging

Firebase è un servizio offerto da Google per facilitare lo sviluppo di applicazioni, fornendo strumenti per la gestione dei dati in tempo reale, sia per la memorizzazione sia per la sincronizzazione. Nel dominio delle applicazioni mobile si utilizza Firebase Cloud Messaging (FCM) per la gestione delle notifiche push in modo affidabile al client. In particolare, FCM viene utilizzato per informare gli utenti sui cambiamenti di stato delle prenotazioni, ad esempio la conferma o l'accettazione da parte di un guidatore, oppure per notificare all'autista la

presenza di una nuova prenotazione. Generalmente Firebase viene integrato nel client in modo tale da ricevere notifiche anche quando l'applicazione non è in primo piano.

2.5.2 Valhalla

Valhalla è un motore di routing, sviluppato inizialmente da Mapzen e attualmente mantenuto e sviluppato dalla comunità, creato per funzionare con i dati cartografici derivanti da OpenStreetMap (OSM), una piattaforma collaborativa di dati geografici. In scenari legati ai trasporti, motori come Valhalla sono impiegati come sistema per il calcolo del tragitto ottimale tra due o più punti spaziali, permettendo di derivare in modo preciso la distanza totale del percorso e il tempo necessario al tragitto. Evitando l'utilizzo di un'istanza pubblica, ma self-hostando la soluzione avvalendosi delle risorse già presenti sui propri server è possibile abbattere i tempi di risposta e avere pieno controllo sul sistema stesso.

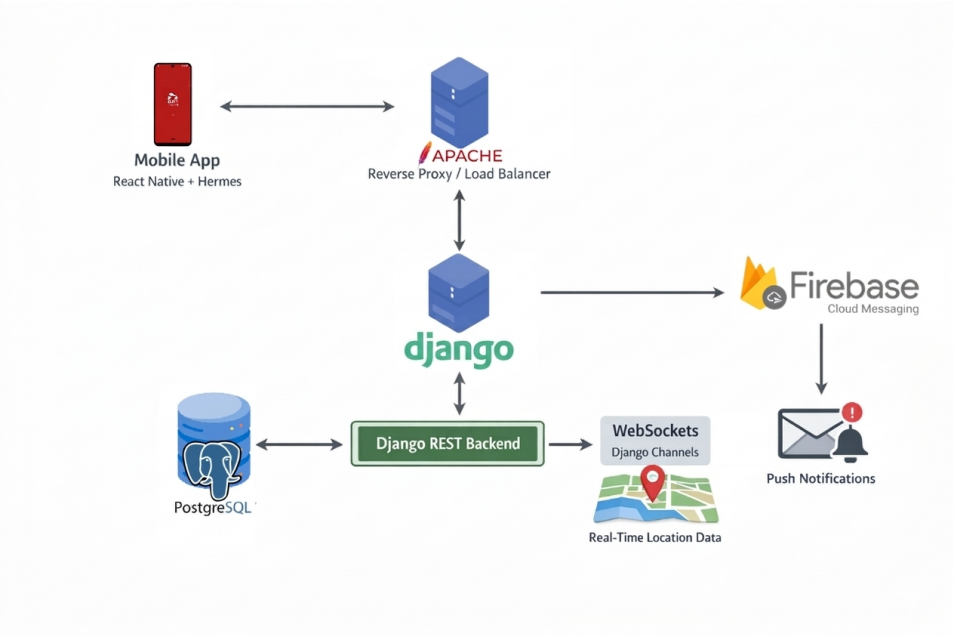

Figura 2.6: Diagramma architetturale dell'applicazione di sharing mobility presa in esame, con le principali componenti del sistema e le interazioni tra client mobile, backend, database e servizi esterni.

Capitolo 3

Analisi del Sistema

In questa tesi, come caso di studio di cybersecurity, si è deciso di analizzare un'applicazione esistente nell'ambito della mobilità condivisa. L'obiettivo è individuare eventuali problematiche di sicurezza nel sistema e nell'infrastruttura interconnessa ad esso, per poi valutare il livello complessivo di sicurezza.
L'analisi è stata effettuata completamente in modalità black box, emulando il comportamento di un utente esterno che non ha accesso a conoscenze esclusivamente disponibili al personale dell'azienda. Questa modalità si è avvalsa di tecniche di reverse engineering, analisi statica e dinamica degli applicativi e delle comunicazioni effettuate, rendendo possibile comprendere il funzionamento del sistema e poi individuare le relative criticità.
Nelle seguenti sezioni verranno descritte le principali funzionalità e le metodologie utilizzate per effettuare lo studio del sistema.

3.1 Descrizione Servizio

L'applicazione implementa un servizio di mobilità condivisa che permette all'utente di prenotare un viaggio indicando un punto di partenza e un punto di arrivo; successivamente, un operatore (il conducente della navetta in quel momento) riceve le suddette richieste e ne modifica lo stato attraverso un'interfaccia specifica per il suo ruolo. In particolare, la richiesta "confermata" passa allo stato di "accettata" in seguito alla presa in carico della prenotazione da parte dell'autista; quando il fruitore sale a bordo e ha inizio il viaggio, l'autista segna la richiesta come "in corso"; infine, a viaggio ultimato la richiesta è "terminata". L'utente può monitorare lo stato della richiesta tramite l'applicazione, assieme alla posizione in tempo reale della navetta, quest'ultima disponibile però solo dopo che la sua richiesta è stata accettata. Sono presenti anche delle funzioni aggiuntive nell'app, per esempio la consultazione

dello storico delle prenotazioni e il salvataggio degli indirizzi preferiti al fine di velocizzare le successive prenotazioni.

3.2 Restrizioni di Utilizzo

L'applicazione non è utilizzabile da chiunque, ma solo da un pubblico ristretto. Infatti, la registrazione dal client permette solo la creazione di utenti in possesso di e-mail di domini istituzionali certificati, rispettivamente `@studenti.dominio.it` per gli studenti e `@dominio.it` per il personale dell'università. Un altro vincolo molto importante imposto a livello client è quello di poter prenotare solo in determinati orari, precisamente la fascia oraria compresa tra le 20:00 e le 23:45, solo in una specifica area geografica delimitata con una geo-fence e con un massimo di quattordici passeggeri.

3.3 Metodologia di Analisi

L'analisi è stata effettuata interamente in modalità black box, cioè senza la conoscenza del sistema e delle sue componenti. L'azienda ha autorizzato lo studio, ma non ha fornito alcun aiuto interno per l'analisi vera e propria dell'applicativo, e tutte le informazioni raccolte sono state ottenute in autonomia. L'approccio esterno è stato scelto con l'obiettivo di comprendere la superficie di attacco che può avere un utente malintenzionato, per rendere l'analisi il più verosimile possibile e garantire la corretta valutazione della sicurezza del sistema.

3.4 Analisi Statica

Sono stati analizzati il codice e i binari dell'applicazione al fine di ottenere una maggiore conoscenza del sistema e di individuare cattive pratiche di sicurezza al suo interno. Per quanto riguarda il frontend, nonostante il codice all'interno del file `index.android.bundle` fosse stato compilato, è stato possibile decompilarlo utilizzando `hermes-dec`, ottenendo un codice offuscato che, sebbene non fosse immediatamente comprensibile nella logica di business, è risultato molto utile per ottenere informazioni sulle API quali il dominio, gli endpoint e il corpo delle varie richieste, ma anche informazioni più generali come lo stack utilizzato per la realizzazione del client, incluse le relative dipendenze. Per il backend, solo dopo un'analisi dinamica è stato possibile ottenere il codice sorgente e effettuare delle analisi manuali e automatiche con strumenti quali `bandit`, per il codice, e `safety`, per le dipendenze utilizzate dall'app.

3.5 Analisi Dinamica

L'analisi dinamica è stata svolta effettuando ingegneria inversa del traffico prodotto dal client ufficiale e con l'ausilio di ambienti di test e di sviluppo API come `Postman`, congiuntamente ad alcuni script Python realizzati allo scopo.

3.5.1 Configurazione Analisi

Per analizzare il traffico, si è fatto uso di un telefono Android con una custom rom installata che consentisse di accedere alla riga di comando del telefono con i permessi di amministratore. Nello specifico, si è utilizzato uno Xiaomi Mi 10T Lite con il bootloader sbloccato e LineageOS 22 installato, e un PC con sistema operativo Linux (Fedora 42). PC e telefono sono stati collegati alla stessa sotto-rete, realizzando visibilità reciproca al Layer 2. Il PC è stato configurato per intercettare le chiamate HTTPS tra l'applicazione e il backend grazie a `mitmproxy`, il quale fungeva da proxy in modalità Man in the Middle. Per rendere possibile la decrittazione del traffico HTTPS non è stato sufficiente aggiungere il certificato self-signed di mitmproxy al Certificate Store centralizzato di Android in quanto l'applicazione mobile fa uso del `Certificate Pinning`, ignorando dunque tutti i certificati non presenti in un'*allow list*; il telefono è stato dunque collegato al PC tramite USB, usando ADB con i permessi di root, e si è proceduto a iniettare a runtime il toolkit di analisi dinamica Frida [9] per permettere di intercettare l'inizializzazione di `SSLContext`, sostituendo il TrustManager risultante con una variante contenente il certificato di mitmproxy. In alternativa, si sarebbe potuto modificare l'APK dell'app per aggiungere al suo interno Frida Gadget, ma per via della minore complessità di implementazione e della disponibilità di un dispositivo adatto è stata scelta la prima soluzione.

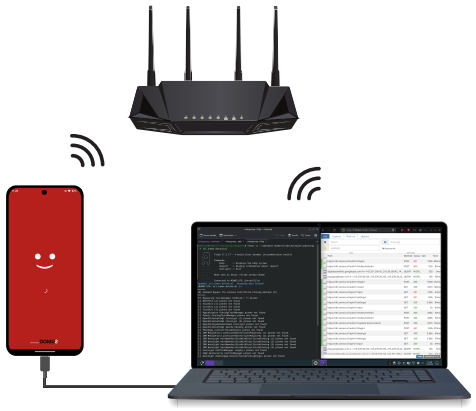

Figura 3.1: Schema della configurazione utilizzata per l'analisi dinamica dell'applicazione.

3.6 Linee Guida OWASP

Per l'analisi di sicurezza è stato deciso di adottare come riferimento principale OWASP (Open Worldwide Application Security Project) per le metriche di sicurezza, un'organizzazione no-profit riconosciuta a livello mondiale che si occupa di sicurezza delle applicazioni. Questa scelta è stata fatta per valutare le vulnerabilità non solo in termini pratici, ma anche rispetto agli standard e alle buone pratiche suggerite dalla comunità. Infatti, OWASP rende accessibili liberamente numerose risorse, strumenti e linee guida utili per l'implementazione delle migliori pratiche durante lo sviluppo, con l'obiettivo di migliorare la sicurezza del software trasversalmente a tutta l'industria. Durante tutta l'analisi si farà riferimento a tutte le specifiche rilasciate pubblicamente dal progetto per rendere l'attività di analisi molto più rigorosa e per una precisa tassonomia delle problematiche riscontrate.

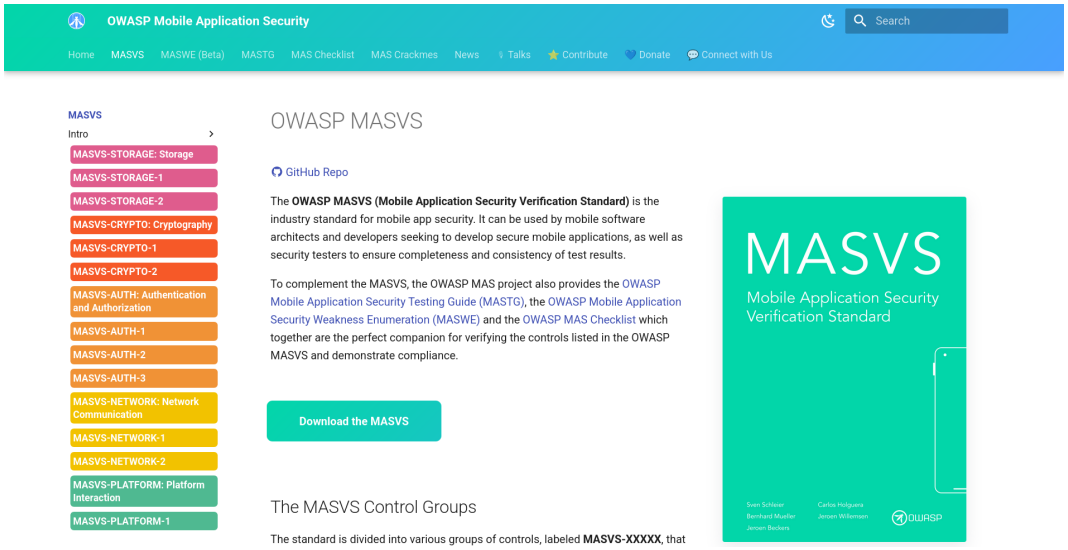

Figura 3.2: Schermata introduttiva del Mobile Application Security Verification Standard (MASVS) pubblicato da OWASP, che definisce linee guida e requisiti di sicurezza per lo sviluppo e la verifica delle applicazioni mobile.

Capitolo 4

Analisi Lato Server

4.1 Raccolta Informazioni

Da una decompilazione iniziale dell'APK presente sul Google Play Store, è stato possibile ottenere il dominio del backend, e un analisi seguente delle stringhe presenti nel binario React ha permesso di trovare anche alcuni endpoint. L'ottenimento di queste due informazioni ha dato inizio a varie analisi dinamiche. Non è stato possibile riscontrare nel codice informazioni riguardo al body atteso da tali endpoint. Tuttavia, il semplice invio di qualsiasi richiesta, anche malformata, verso gli endpoint reperiti dalla decompilazione produceva una pagina di errore del Django REST Framework corredata da informazioni complete sui metodi supportati e i campi richiesti, rendendo più facile il lavoro di ingegneria inversa delle API.

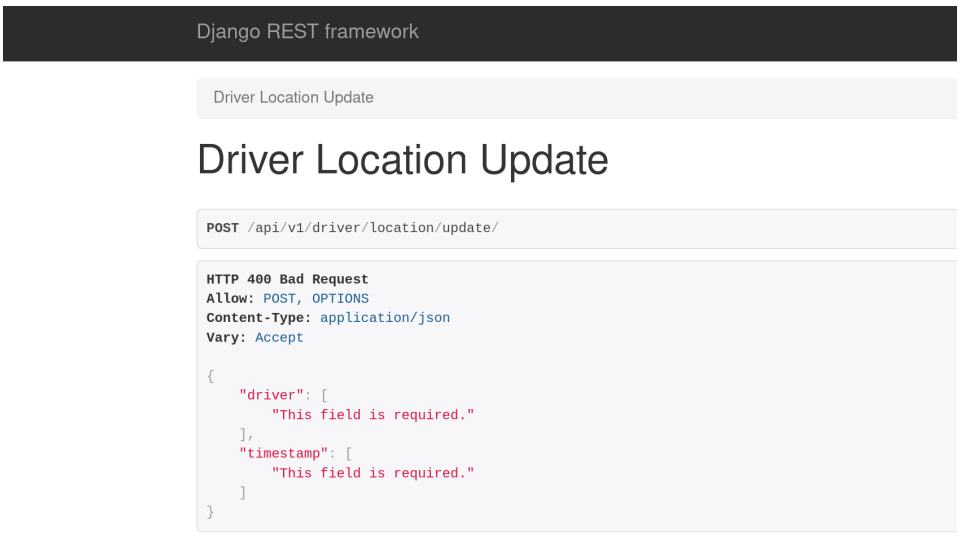

Figura 4.1: Schermata di errore di Django REST Framework che evidenzia i campi obbligatori per la richiesta.

Oltre a queste evidenze, è emerso un altro problema di configurazione; gli

endpoint della documentazione, anch'esso tipico di un'istanza Django in modalità debug; infatti gli endpoint della documentazione Swagger e Redoc, che dovrebbero essere accessibili solo agli sviluppatori, sono stati resi pubblicamente reperibili. Con l'aiuto del codice decompilato, dell'analisi del traffico dinamico dell'app e delle pagine di errore di DRF, è stato possibile ricostruire fedelmente la documentazione su tutti gli endpoint. Inoltre, ciò ha permesso di enumerare anche gli endpoint del tutto inutilizzati dal client, come ad esempio `/auth/v1/users/`, che è accessibile solo agli utenti con privilegi di amministratore.

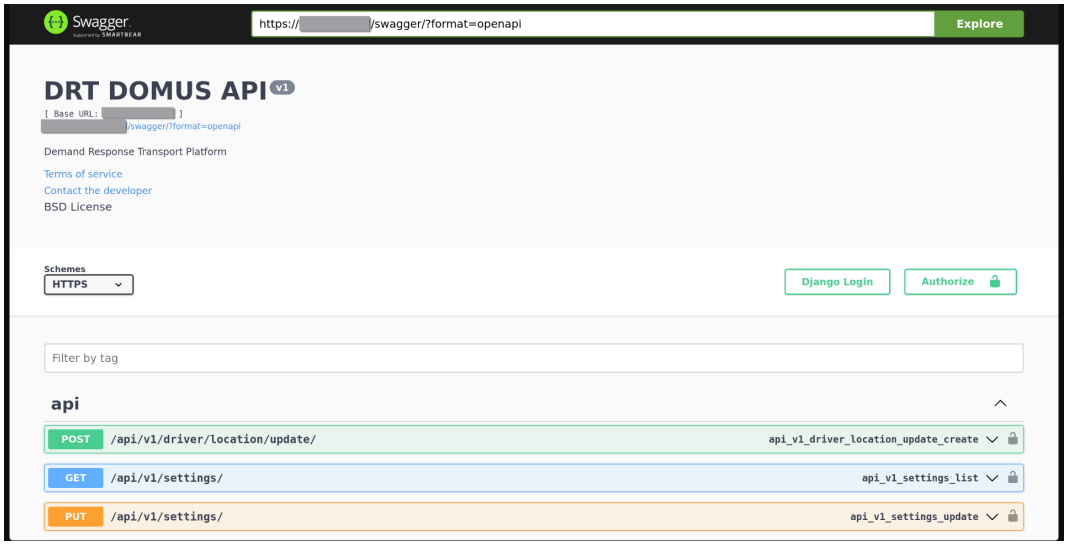

Figura 4.2: Schermata della documentazione pubblica Swagger, che mostra la struttura degli endpoint disponibili all'analisi. Le informazioni sensibili sono state oscurate per motivi di sicurezza.

Ultimata la documentazione degli endpoint, si è passati ad analizzare il comportamento del backend.

4.2 Analisi dei Controlli Lato Server

Effettuando chiamate di test verso il backend, è stata determinata la necessità di un'autenticazione obbligatoria per tutti gli endpoint, fatta eccezione per quelli che instaurano l'autenticazione stessa (`/login`) o creano un nuovo account (`/signup`).
Durante l'utilizzo dell'applicazione si è osservato che la semplice modifica dell'orario del dispositivo consentiva di raggirare la limitazione temporale prevista per la prenotazione, poiché non era prevista nessuna validazione sia lato client sia lato server dell'orario effettivo della stessa. Questa scoperta ha permesso di focalizzare l'analisi sulla presenza di una reale applicazione dei vincoli decisi

dal vendor. Il primo vincolo analizzato è uno dei più importanti imposti dal client ovvero la creazione di account con indirizzo e-mail afferente ai soli domini dell'università. Questo è infatti l'unico meccanismo previsto per bloccare l'accesso al servizio a persone al di fuori del contesto universitario. Tuttavia, dall'analisi è emerso che questa verifica non è effettuata anche lato backend. Di conseguenza, chiunque sia a conoscenza dell'endpoint signup e in possesso di un indirizzo e-mail valido può creare un account, bypassando completamente il vincolo. La mancanza del controllo sul dominio dell'e-mail rappresenta una `A04:2021 -- Insecure Design` [10] (Progettazione non sicura), poiché nella progettazione del sistema non è stato previsto un controllo fondamentale, cioè quello della registrazione esclusiva degli utenti.

4.2.1 Gestione delle Prenotazioni

La problematica precedentemente descritta è stata riscontrata molteplici volte nell'applicazione, ma affligge principalmente gli endpoint coinvolti nella gestione delle prenotazioni, che approfondiremo in questo capitolo. Il funzionamento è analogo a quello descritto nella Sezione 3.1, ma in questo caso verrà analizzato, più nello specifico, il flusso di richieste HTTP effettuate durante una prenotazione per creare, confermare e aggiornare lo stato di un viaggio e come il backend gestisce il tutto.
Quando un utente crea una prenotazione, la prima operazione che compie è inserire un punto di partenza e un punto di arrivo, che il client inserirà in una richiesta PUT all'endpoint `/trips`, il quale risponderà con l'oggetto prenotazione creato nel database, con stato iniziale `init`. Successivamente, quando l'utente conferma il viaggio, specificando anche il numero di persone, il client farà una richiesta PATCH verso l'endpoint `/trip/[id]`, inviando il nuovo stato della prenotazione, cioè `confirmed`. A questo punto, l'autista in servizio potrà decidere di aggiornare lo stato della richiesta, prendendo in carico il viaggio (lo stato passa ad `accepted`) o marcando il viaggio come in corso (lo stato passa ad `started`). Infine, l'autista segna il viaggio come concluso e lo stato della prenotazione passa a `completed`.

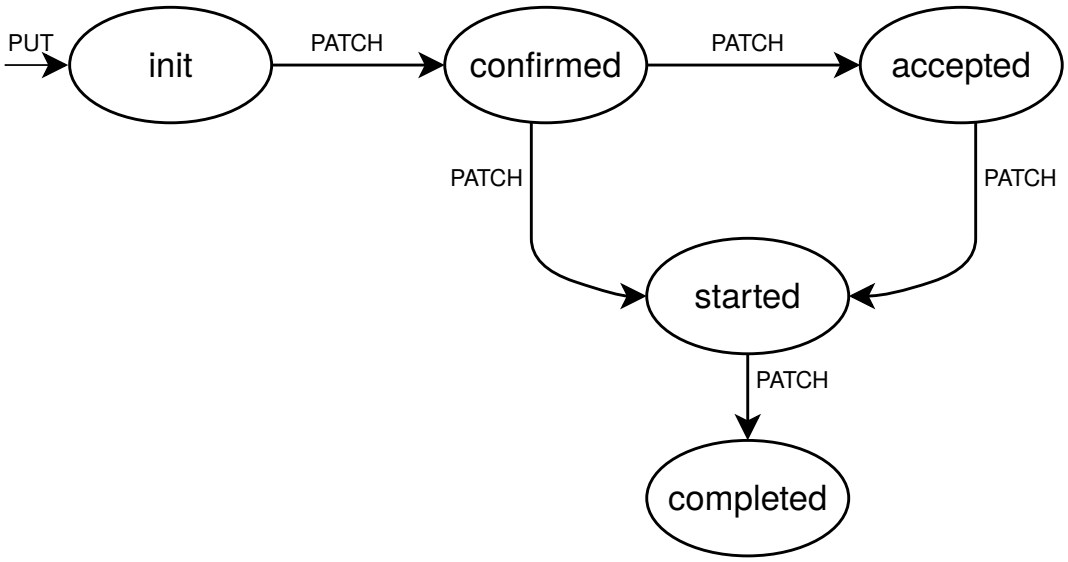

Figura 4.3: Diagramma delle transizioni di stato di una prenotazione, che illustra l'evoluzione del processo dalla creazione alla conclusione del servizio.

Tuttavia, analizzando questa operatività del backend, è emerso che non esiste una logica di controllo server-side delle transizioni di stato effettuate tramite richieste PATCH; è possibile anzi modificare una prenotazione saltando arbitrariamente gli stati intermedi o tornando indietro (ad esempio passando direttamente da `init` a `completed` o viceversa). Inoltre, lo stesso endpoint accetta modifiche da qualsiasi account, anche privo del ruolo guidatore, il che rende possibile a chiunque abbia un account sulla piattaforma di manipolare i dati presenti nell'oggetto prenotazione, inclusi gli stati (`accepted`, `started`, `completed`) che dovrebbero essere riservati solo agli autisti.

Accesso alle Prenotazioni

Un'altra funzionalità importante dell'app è il recupero delle prenotazioni. Queste, infatti, non vengono memorizzate sul telefono su cui l'applicazione è in esecuzione, ma vengono ottenute tramite una richiesta GET ad ogni avvio dell'app o ogni volta che si accede alla schermata principale. La richiesta può essere effettuata a due endpoint differenti. Il primo di essi è `/trips`, che presenta un comportamento diverso a seconda del ruolo dell'utente che lo richiama: infatti, l'utente normale ottiene lo storico delle proprie prenotazioni, mentre un guidatore ottiene l'insieme di tutte le prenotazioni effettuate da tutti gli utenti presenti nel database, in una risposta priva di paginazione poiché del tutto non supportata dalle API.
Dai test effettuati è stato osservato uno storico delle prenotazioni in formato JSON di circa 6,9 MB (peso misurato al momento della stesura). Questo comporta un notevole rallentamento sia in fase di scaricamento che in fase

di elaborazione, da ricondurre al tempo necessario per costruire gli oggetti in memoria.
L'effetto congiunto dell'assenza di alcun limite sulle chiamate che si possono fare alle API (rate limit) e della possibilità da parte di un attaccante di riempire il database prenotazioni con prenotazioni fittizie (anche a nome di altri utenti come vedremo nella Sezione 4.3.3), rende possibile l'aumento non controllato della dimensione dello storico scaricato automaticamente da qualsiasi utente dell'applicazione realizzando dunque un attacco Denial of Service [11].
Il secondo endpoint, destinato al recupero delle singole prenotazioni è `/trip/[id]`, che però non effettua alcun controllo che l'id specificato sia relativo a una prenotazione dell'utente che richiama l'endpoint. La conseguenza è che è possibile accedere alle prenotazioni di altri utenti della piattaforma semplicemente incrementando o decrementando l'id a partire da un id noto. Questo approccio costituisce un secondo metodo con cui è possibile ottenere l'intero storico delle prenotazioni. La criticità appena descritta, unita al mancato controllo di stato delle prenotazioni, crea un problema di tipo *Insecure Direct Object References* (IDOR) sia in lettura che in scrittura.

4.2.2 Aggiornamenti in Tempo Reale

Oltre alla gestione delle prenotazioni tramite endpoint REST, l'applicazione fa uso anche di un canale in tempo reale per la condivisione della posizione della navetta, che si attiva durante il cambiamento di stato che avviene quando l'autista accetta la prenotazione. Il client del guidatore invia periodicamente la posizione della navetta per mezzo di una chiamata POST all'endpoint `/driver/location/update/` e inviando un JSON così costruito:

```
{
    "driver_id": 856,
    "location": {
        "lat": 39.3538334,
        "lon": 16.2313784
    },
    "timestamp": "2025-09-26␣21:31:47"
}
```

Figura 4.4: Esempio di payload JSON inviato dal backend tramite connessione WebSocket per notificare ai client la posizione aggiornata della navetta.

Il backend ottiene questi dati e li reindirizza senza alcuna modifica al WebSocket apposito `/ws/driver/[id]`. L'accesso al canale richiede che il client

conosca l'identificativo del guidatore, che è possibile ottenere semplicemente recuperando la prenotazione poiché il campo viene popolato dal client in possesso al guidatore.
Analizzando il meccanismo, è emerso che la possibilità di collegarsi al WebSocket non è vincolata a nessun sistema di autenticazione o autorizzazione, e dunque per accedervi è necessaria solo la semplice conoscenza dell'endpoint e dell'identificativo del guidatore. Inoltre, il canale non è impostato in sola lettura e quindi chiunque sia collegato ha la possibilità di inviare messaggi arbitrari, incluse posizioni fittizie che tutti i client in ascolto non hanno alcuna possibilità di verificare nella loro correttezza. A peggiorare la situazione contribuisce la condivisione dello stesso account da parte di tutti gli autisti, cosa che comporta una mancata rotazione e una più facile riutilizzazione dell'id. Infine, la frequenza di invio di posizioni reali da parte del client dell'autista è estremamente dipendente dalla schermata che egli sta visualizzando al momento, il che rende più semplice per un attaccante (che non ha il limite delle schermata) sovraccaricare il sistema con posizioni fasulle.

4.2.3 Gestione delle Impostazioni dell'App

L'applicazione, ad ogni avvio, fa una chiamata GET all'endpoint `/settings`, il quale restituisce tutte le restrizioni che il client deve imporre, già illustrate nella Sezione 3.2, oltre ad altri parametri, quali, ad esempio, la versione minima supportata di Android e iOS. Analizzando le API nella documentazione Swagger, è stato scoperto che l'endpoint sopra riportato accettava anche il metodo PUT, il quale richiede un token per essere usato, ma per il quale non è previsto alcun controllo specifico sul ruolo dell'account che ne fa utilizzo. Di conseguenza chiunque sia a conoscenza dell'endpoint, e sia in grado di generare un token valido (Sezione 4.3.3) sarebbe in grado di modificare le impostazioni di base dell'intero applicativo e rendere il servizio inutilizzabile per tutti, ad esempio inserendo un massimo di zero passeggeri per prenotazione oppure fissando una geo-fence molto ristretta.

4.2.4 Mancata Rotazione del Refresh Token

Il sistema usa un meccanismo di autenticazione basato su JSON Web Token (JWT). Alla prima autenticazione, il client dell'utente invia le credenziali di accesso (e-mail e password) e riceve in risposta dal server due token; l'*access token*, utilizzato dal client per autenticarsi al backend, e il *refresh token*, utilizzato per richiedere un nuovo access token alla sua scadenza, rispettivamente della durata di quindici minuti e trentuno giorni. È emerso, durante le analisi, che il backend restituisce esclusivamente l'access token al rinnovo del token di

accesso, senza rigenerare e invalidare il refresh token. Questo viola le specifiche OWASP Session Management Cheat Sheet [12], le quali prevedono anche la rotazione del token di refresh durante la transizione da uno stato non autenticato (access token scaduto) a uno stato autenticato (access token rinnovato). Questa assenza di rotazione del refresh token, unita alla memorizzazione in chiaro analizzata nel Capitolo 5, permette a un attaccante che entri in possesso dei token attualmente validi (ad esempio mediante l'accesso fisico al client) di mantenere l'accesso al servizio per almeno trentuno giorni.

4.3 Configurazioni e Implementazioni Insicure

4.3.1 Django Debug

Come anticipato nella Sezione 4.1, l'istanza in produzione di Django REST API è in modalità debug; tale modalità è valida per tutto il framework Django. È stato infatti possibile inviare al backend una richiesta appositamente malformata per poi ottenere di conseguenza una pagina di errore dal server contenente un output di stack trace. All'interno della stack trace è stato possibile visionare informazioni sul funzionamento dell'intera applicazione molto utili per un attaccante. Si potevano leggere, per esempio, alcuni percorsi relativi all'organizzazione dei file sul server e dettagli sulle credenziali di accesso al server di posta e al database quali la lunghezza delle password. Inoltre, venivano specificati dati del sistema come la versione di Python, l'elenco dei moduli usati e molto altro. Le informazioni più importanti scoperte sono il nome di alcuni file con i relativi percorsi e l'url per accedere ai file statici `/static`, che come vedremo nella sezione successiva, presentano anche una problematica di sicurezza.

Figura 4.5: Stack trace restituita dal framework Django quando l'applicazione è eseguita in modalità *DEBUG*. Le informazioni sensibili sono state oscurate per motivi di sicurezza.

4.3.2 Gestione Insicura dei Percorsi

L'istanza di Django in modalità debug è risultata essere vulnerabile a un attacco denominato path traversal [13]. L'uso della sequenza double-dot (..) come componente della URL delle richieste, unitamente alla conoscenza dei nomi dei file e del loro percorso ricavabili dalla stack trace, ha permesso di navigare a ritroso nel filesystem, ottenendo l'accesso a tutti i file presenti nell'ambiente Python. Né Apache (nelle sue direttive di Location, Directory, Files o Proxy) nè Django (con apposite `BASE_DIR`) sono stati configurati per impedire l'uso della componente double-dot.
Il risultato è stata l'esfiltrazione della quasi totalità del codice sorgente del backend, ma anche di vari file di configurazione e documenti JSON, ottenendo una conoscenza più avanzata sul funzionamento dell'applicazione.

4.3.3 Compromissione dei Segreti Applicativi

Durante l'analisi del codice ottenuto tramite attacco path traversal, analizzato nella sezione precedente, sono stati individuati numerosi segreti all'interno dei file esfiltrati, tra cui alcuni direttamente cablati (*hardcoded*) nel codice. Tra questi segreti si possono elencare, per esempio, informazioni per la connessione al database dell'applicativo incluse le credenziali, la chiave privata per la firma dei token di accesso creati dal backend e il token Firebase utilizzato per l'invio delle notifiche in tempo reale.

Chiave Privata JWT

Come evidenziato in precedenza, è stato possibile rinvenire dal file `settings.py` la chiave privata utilizzata dal backend per firmare i JSON Web Token (JWT) di accesso.
L'analisi del traffico ha consentito di visionare la struttura dei token emessi dal server, già descritta nella Sezione 2.2.5. Ciò ha permesso la generazione di token validi per qualsiasi account e la conseguente possibilità di impersonificazione di qualsiasi utente. Eseguendo la generazione di un token per l'utente con id 1, partendo dall'assunto che si tratta di un id tipicamente assegnato a un utente privilegiato, è stato possibile ottenere un token con permessi di amministratore del backend RESTful API e di conseguenza richiamare le API riservate esclusivamente al ruolo di amministratore. Tra questi endpoint figura `/auth/v1/users/`, che restituisce una lista paginata di tutti gli account ordinati per id, la quale riporta dati personali degli utenti (nome, cognome, indirizzo e-mail, identificativo dell'account, data di iscrizione) ma non dati di autenticazione, quali le password.

Una parziale ma importante mitigazione è il fatto che i token generati risultano validi solo per le API e non possono essere utilizzati per accedere alla sezione admin dell'interfaccia web di Django, la quale richiede espressamente un nome utente e una password per creare una sessione. Questa scoperta ha reso possibile la creazione di token per account guidatore che, viste le peculiarità del ruolo, ha consentito un grande approfondimento della conoscenza sull'intero sistema.

Chiave Privata Firebase

È stata rinvenuta tra i file di configurazione la chiave del servizio Firebase, utilizzata dal sistema per autenticarsi con il servizio Google. Tale informazione potrebbe essere sfruttata da un utente malintenzionato per inviare massivamente notifiche push ai dispositivi con il client installato, causando i tipici disturbi all'utenza connessi allo spam.

Server PostgreSQL

All'interno della configurazione erano presenti anche le credenziali di accesso al database PostgreSQL, comprensive dell'host e dei parametri di connessione. Il servizio risulta raggiungibile per chiunque sulla rete internet, ma le richieste vengono accettate dal server solo se l'indirizzo IP del chiamante è presente nella lista degli autorizzati all'accesso, mitigando parzialmente il problema degli accessi non autorizzati. Rimane però la gestione non corretta delle credenziali, che dovrebbero invece essere rimosse dal codice e trattate come variabili d'ambiente.

Server E-mail

Infine, sono state trovate anche le impostazioni di collegamento al server SMTP utilizzato dall'applicazione per l'operazione di registrazione e reimpostazione delle password. A differenza del database, quest'ultimo non presentava alcun vincolo sulla provenienza del chiamante, il che, grazie alle credenziali ottenute, ha permesso l'invio di e-mail utilizzando il dominio legittimo del servizio; ciò potrebbe facilmente favorire campagne di phishing. A scopo di verifica e documentazione, è stato inviato un messaggio a un indirizzo di posta temporaneo per confermare il funzionamento delle credenziali. Un aspetto cruciale della configurazione del server e-mail è che quest'ultimo è stato impostato per permettere solo l'invio e non il download delle e-mail, il che vanifica un possibile attacco per il reset della password di un account amministratore tramite lo scaricamento (dalla cartella posta inviata del server) dell'e-mail contenen-

te il link di reset generato da una procedura di recupero iniziata dallo stesso attaccante.

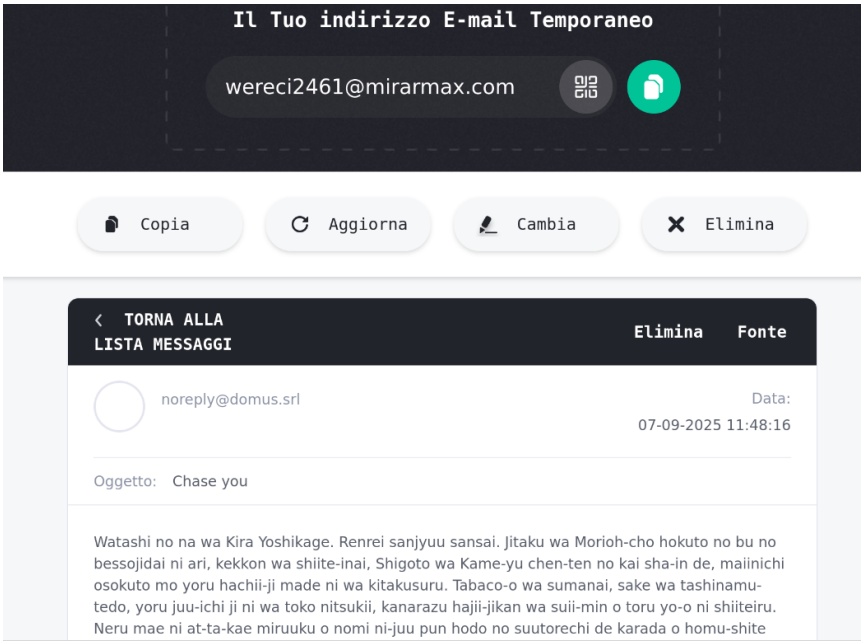

Figura 4.6: Schermata dell'invio di un'e-mail di test a un indirizzo temporaneo tramite le credenziali SMTP ottenute dai file di configurazione.

4.3.4 Meccanismo di Reset Password

Il meccanismo di reset della password deve essere considerato, a livello di sicurezza, come un secondo meccanismo di autenticazione. Un sistema di reimpostazione della password che non preserva i controlli previsti dal meccanismo di login principale costituisce un vettore di attacco. Nei paragrafi precedenti è stato mostrato come ottenere numerose informazioni sensibili, le quali combinate tra di loro, riducono notevolmente la difficoltà di generare un link di reset valido.

```
def _make_hash_value(self, user, timestamp):
  login_timestamp = (
      ""
      if user.last_login is None
      else user.last_login.replace(microsecond=0, tzinfo=None))
  email_field = user.get_email_field_name()
  email = getattr(user, email_field, "") or ""
  return
   ↪  f"{user.pk}{user.password}{login_timestamp}{timestamp}{email}"
```

Figura 4.7: Implementazione del metodo _make_hash_value utilizzato per la generazione del token di reset password.

Come è possibile vedere nell'estratto del codice Django preposto alla creazione di un link di reset valido in Figura 4.7, risultano necessari i seguenti elementi:

- identificativo dell'utente (user id);
- chiave privata utilizzata per la firma del token;
- timestamp dell'ultimo login dell'utente;
- timestamp di creazione della richiesta di reset;
- indirizzo e-mail dell'utente;
- hash della password memorizzato nel database.

Nei precedenti capitoli sono stati già rinvenuti l'identificativo dell'utente, la chiave privata e l'indirizzo e-mail; il timestamp di creazione della richiesta è noto a priori dall'utente che effettua la richiesta di reset della password. Pertanto, a frapporsi alla creazione di link validi restano solamente l'hash della password e il timestamp dell'ultimo login. L'accesso al database per l'ottenimento dell'hash delle password, come visto in precedenza, è limitato solo da un filtro per indirizzo IP; tuttavia, non può essere considerato un controllo sufficiente per l'accesso a un'informazione critica quale l'hash delle password.

Riduzione dello spazio di ricerca del timestamp

Il valore del campo `last_login` può essere facilmente ottenuto tramite una stima, a causa dell'implementazione scorretta del sistema di rinnovo dei token utente; il client, infatti, non riesce a rinnovare il token di accesso, limitando la durata della sessione a quella del token del primo accesso, cioè quindici minuti. Tale comportamento non si verifica solo per gli utenti passeggeri, ma anche per i guidatori; il tutto, combinato con il fatto che ciò accade anche durante gli orari di esercizio del servizio, permette a un attaccante che si trova fisicamente nella navetta di osservare il momento in cui l'autista effettua il login e di restringere l'intervallo temporale di ricerca a pochi secondi prima e dopo l'evento osservato. La finestra in cui questo attacco brute force può essere effettuato, però, non è sempre la stessa: se si osserva un login durante l'orario di normale funzionamento del servizio (20:00-23:45) la finestra si limita ai meri quindici minuti di durata del token di accesso; tuttavia, se l'ultimo login viene osservato in prossimità della fine del servizio la finestra temporale si allarga fino ad estendersi dalle 23:45 (orario di fine servizio) alle 20:00 (orario di inizio servizio) del giorno successivo, ampliando notevolmente il tempo disponibile per la ricerca.

Capitolo 5

Analisi Lato Client

5.1 Protezione del Codice

A differenza delle applicazioni native Android, sviluppate in Kotlin o Java, il codice React Native non è immediatamente disponibile. Nelle app tradizionali, infatti, è molto facile risalire al codice sorgente originale decompilando i file contenenti il bytecode smali ottenuti dall'APK dell'app. Con l'adozione di Hermes nel 2021 come motore JavaScript sono state introdotte numerose migliorie non solo a livello prestazionale, ma anche a livello di protezione del codice applicativo; Hermes, infatti, non si limita a compilare il codice in bytecode nativo, ma di default offusca il codice rendendo così più difficile l'ottenimento del codice originale e quindi la comprensione della logica di business. L'applicazione in analisi ne beneficia, poiché il codice risulta totalmente offuscato.

5.2 Gestione Storage Locale

Analizzando la cartella di archiviazione locale dell'applicazione è stato possibile trovare alcune problematiche nella gestione di informazioni sensibili. Nel percorso `/data/data/package_name/files/mmkv` è stato rinvenuto, infatti, il file `mmkv.default`, contenente in chiaro molti dati personali riguardanti l'utente, come ad esempio nome, cognome, e-mail, ruolo e token di autenticazione. Il salvataggio in chiaro di queste informazioni rappresenta un rischio identificato da OWASP con il nome MASWE-0006: Sensitive Data Stored Unencrypted in Private Storage Locations (Dati sensibili memorizzati in chiaro in aree di archiviazione privata) [14]. Questa problematica implica l'esposizione di dati sensibili ad accessi non autorizzati. OWASP raccomanda l'utilizzo di meccanismi di cifratura che fanno uso di `Android Keystore` e `iOS Keychain` per il salvataggio delle chiavi crittografiche. Il rischio è parzialmente mitigato dal fatto che l'applicazione venga rilasciata al pubblico con

la flag `android:allowBackup="false"` e senza la flag `android:debuggable`; tuttavia queste flag non risolvono completamente il problema ma lo rendono più difficile da sfruttare per un attaccante. È stato rinvenuto, tra i file dell'applicazione, il file `RN_KEYCHAIN.xml`, che come suggerisce il nome dovrebbe essere associato alla libreria React Native Keychain, la quale consente il salvataggio sicuro dei dati; questo file non sembra però effettivamente utilizzato ed è probabile che i dati al suo interno vengano duplicati all'interno del file `mmkv.default`.

5.3 Gestione delle Sessioni

Durante il monitoraggio del traffico di rete è stato possibile individuare un problema riguardante la gestione dei logout. Sebbene l'applicazione cancelli i dati presenti nel file `mmkv.default` e reindirizzi l'utente alla schermata di login, l'operazione è puramente locale e non è seguita da nessuna chiamata equivalente al backend, il che permette di utilizzare il token anche dopo aver effettuato il logout. Questo vìola le linee guida OWASP Mobile App Authentication Architectures [15], le quali affermano che all'uscita del servizio l'applicazione deve eliminare i dati salvati localmente dell'utente e invalidare il token sul backend remoto.

5.4 Protezione delle Comunicazioni

Tutte le comunicazioni tra l'applicazione e il server avvengono utilizzando il protocollo HTTPS, il quale garantisce la non accessibilità dei dati in transito a un eventuale attaccante passivo sulla rete. L'uso di HTTPS è un comportamento predefinito della libreria utilizzata per le comunicazioni di rete (OkHttp). Tuttavia, l'app implementa anche un altro controllo, non presente di default, che è il Certificate Pinning. Questo meccanismo consente al client di accettare solo uno o più certificati specifici del server preventivamente censiti nell'applicazione, rifiutando qualsiasi altro certificato emesso da altre autorità di certificazione, inclusi quelli già presenti a livello di sistema. Tale configurazione riduce drasticamente il rischio di attacchi Man in the Middle, ma al contempo, ha reso più complessa l'analisi dinamica dell'app.

Capitolo 6

Considerazioni sull'Organizzazione

L'analisi tecnica perde parte del suo valore se, nel momento in cui i risultati vengono consegnati, non esiste una struttura organizzativa ben definita per risolvere e gestire le problematiche di sicurezza riscontrate. Scoprire i problemi di sicurezza non rappresenta altro che il primo passo di un processo molto più grande volto a effettuare dei veri e propri miglioramenti sulla sicurezza del sistema. In assenza di tali procedure di segnalazione e correzione delle vulnerabilità identificate, i problemi individuati potrebbero ritorcersi contro l'azienda stessa; qualora tali vulnerabilità venissero rese pubbliche prima di essere state risolte potrebbero minare ulteriormente la sicurezza del sistema. Questo è un problema che non riguarda solamente i software creati dalle grandi aziende, ma risulta particolarmente calzante per l'applicazione in analisi, proveniente da un vendor minore. Un esempio rilevante è rappresentato dal presente lavoro di tesi: le dinamiche di divulgazione delle vulnerabilità non sono state discusse preventivamente a causa della mancanza di una vera e propria procedura formale all'interno dell'organizzazione dell'azienda in questione.

6.1 Quadro Normativo

Per inquadrare bene il problema si sono utilizzati come punto di riferimento gli standard internazionali sulla gestione delle vulnerabilità, cioè lo standard *ISO/IEC 29147 (Vulnerability Disclosure)*, che definisce i criteri con cui un'azienda deve recepire segnalazioni di vulnerabilità e le metodologie con cui i ricercatori devono divulgare in modo corretto queste informazioni, nonché lo standard *ISO/IEC 30111 (Vulnerability Handling Processes)*, che esplica il flusso di processi organizzativi interni da attuare dopo aver ricevuto una segnalazione di una vulnerabilità.

6.2 Criticità Canale di Comunicazioni

Dall'analisi effettuata è emersa una mancata soddisfazione dei requisiti previsti dalla norma ISO/IEC 29147, la quale afferma che devono essere predisposti dei canali di comunicazione appositi per questo tipo di segnalazioni; attualmente l'unico contatto disponibile per un eventuale ricercatore è l'indirizzo *info@dominio.srl*, che è un indirizzo e-mail generico utilizzato per tutte le comunicazioni. Questa scelta porta con sé numerose problematiche, tra cui il rischio di smarrimento delle informazioni, dovuto alla loro dispersione tra comunicazioni non rilevanti, e la mancata riservatezza e integrità, in quanto i messaggi possono essere intercettati o modificati, nonché visionati da personale non qualificato alla gestione di questo genere di segnalazioni.
Inoltre, sul sito web non è presente alcuna sezione riguardante la sicurezza, né indicazioni su come un ricercatore di sicurezza dovrebbe effettuare una segnalazione.
Un'ulteriore buona pratica è la pubblicazione di un file `security.txt`, un documento standardizzato definito dalla RFC 9116, che consente a un eventuale ricercatore di sicurezza di reperire facilmente le modalità per segnalare le vulnerabilità. All'interno di questo file possiamo trovare i contatti specifici per le segnalazioni di sicurezza, eventuali chiavi PGP per comunicazioni cifrate, le politiche di divulgazione, se presenti, e programmi di bug bounty.

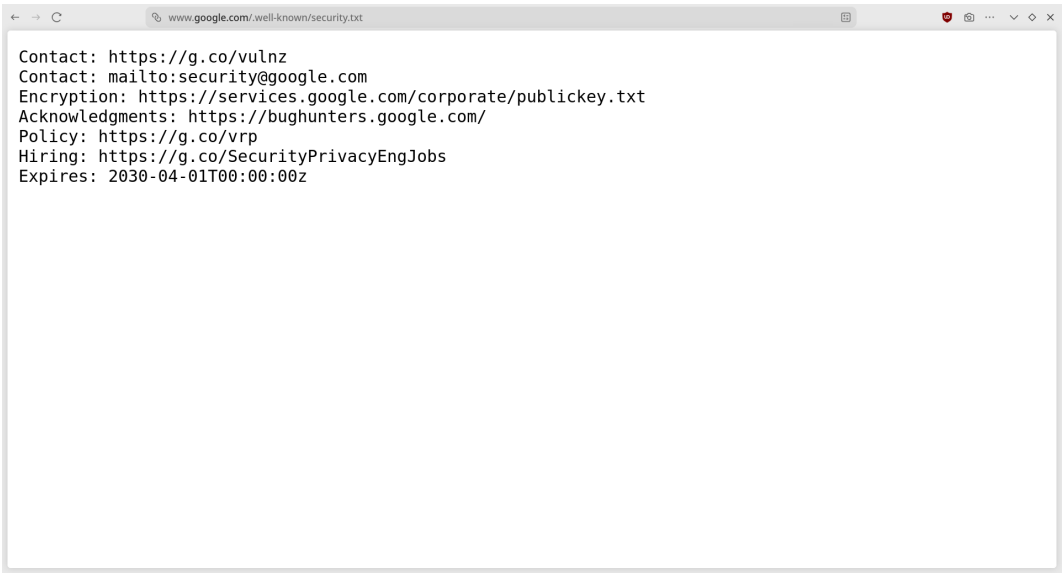

Figura 6.1: File security.txt pubblicato da Google, contenente informazioni su come segnalare vulnerabilità di sicurezza e i contatti dedicati per la divulgazione.

6.3 Workflow Interno

Le problematiche non si limitano solo alla fase di ricezione, ma sono presenti anche nella gestione delle vulnerabilità. Lo standard ISO/IEC 30111 afferma che, una volta ricevuta una segnalazione, si deve attuare un processo ben strutturato nelle seguenti fasi:

- **Triage**: valutazione della gravità e della validità della vulnerabilità.
- **Investigazione**: analisi tecnica approfondita per individuare la causa del problema.
- **Risoluzione**: sviluppo e distribuzione di una correzione della debolezza.

6.4 Conclusioni sull'Organizzazione

In conclusione, l'implementazione degli standard ISO/IEC 29147 e ISO/IEC 30111 non rappresenta solo l'adeguamento a degli standard di conformità, ma anche una scelta dell'azienda per consentire di trasformare una potenziale falla di sicurezza in un processo di miglioramento del prodotto software utilizzando un flusso di lavoro ben specificato e prevedibile.
Nel caso in analisi, l'assenza di tali pratiche evidenzia un basso livello di maturità organizzativa nella gestione della sicurezza. L'introduzione di procedure formali, canali dedicati e policy di divulgazione rappresenta un passo fondamentale per ridurre i rischi e rafforzare la fiducia, propria e del pubblico nel sistema sviluppato.

Capitolo 7

Conclusioni

L'analisi effettuata sull'applicazione ha fatto emergere diverse problematiche significative riguardo alla sicurezza del sistema. Le vulnerabilità sono molteplici e spaziano dalla mancanza di controlli sui dati scambiati lato server all'assenza di controlli sui ruoli e sui permessi dell'utente, passando per configurazioni insicure dei componenti software e per una memorizzazione non corretta dei dati dell'utente.
Lo studio del sistema dal punto di vista di un utente esterno ha permesso di fare una stima realistica e accurata della superficie di attacco esposta a potenziali attaccanti. Dai risultati ottenuti, è stato poi possibile evidenziare la necessità di integrare la sicurezza sin dalle fasi iniziali dello sviluppo dell'applicazione, cosa che non risolve solo i problemi nella sfera della sicurezza, ma migliora anche l'affidabilità e la robustezza complessiva del sistema.
Il processo ha permesso di generare due risultati utili: da un lato ha fornito all'azienda sviluppatrice del sistema delle indicazioni tecniche e metodologiche su come gestire e risolvere le falle di sicurezza; dall'altro lato ha rappresentato un'importante attività formativa sui temi trattati. L'esperienza ha consentito di approfondire gli exploit e i vettori di attacco più comuni, oltre all'acquisizione di conoscenze sull'utilizzo di strumenti sia per l'analisi dinamica sia per l'analisi statica e capacità di ragionamento e pensiero laterale applicato alla ricerca di vulnerabilità e superfici di attacco meno ovvie.

Bibliografia

[1] Bruce Schneier. Hacking drug pumps. Post sul blog "Schneier on Security", 2015. Ultimo accesso: 5 Settembre 2025.

[2] React Foundation. React native · learn once, write anywhere. Sito ufficiale React Native, 2025. Ultimo accesso: 7 Marzo 2026.

[3] React Foundation. Using hermes · react native. Documentazione ufficiale React per il motore Hermes, 2025. Ultimo accesso: 7 Marzo 2026.

[4] Manmeet Singh. React native explained: From the legacy bridge to the new architecture, 2026. Ultimo accesso: 10 Marzo 2026.

[5] Rays. Python django framework best tutorial, 2026. Ultimo accesso: 11 Marzo 2026.

[6] Swagger. Openapi specification - version 3.1.0 — swagger. Documentazione online Swagger, 2026. Ultimo accesso: 03 Aprile 2026.

[7] Auth0. Jwt.io - json web token introduction, 2026. Ultimo accesso: 10 Marzo 2026.

[8] SQLite Project. Sqlite home page. Sito ufficiale SQLite, 2025. Ultimo accesso: 8 Marzo 2026.

[9] Frida. Frida - a world-class dynamic instrumentation toolkit. Sito ufficiale Frida, 2026. Ultimo accesso: 03 Aprile 2026.

[10] OWASP Foundation. A04 insecure design - owasp top 10:2021. Documentazione online OWASP, 2021. Ultimo accesso: 20 Settembre 2025.

[11] OWASP Foundation. Denial of service — owasp foundation. Documentazione online OWASP, 2026. Ultimo accesso: 03 Aprile 2026.

[12] OWASP Foundation. Session management cheat sheet - owasp cheat sheet series. Documentazione online OWASP, 2025. Ultimo accesso: 27 Febbraio 2026.

[13] OWASP Foundation. Path traversal — owasp foundation. Documentazione online OWASP, 2026. Ultimo accesso: 03 Aprile 2026.

[14] OWASP Foundation. Maswe-0006: Sensitive data stored unencrypted in private storage locations - owasp mobile application security. Documentazione online OWASP, 2025. Ultimo accesso: 16 Ottobre 2025.

[15] OWASP Foundation. Mobile app authentication architectures - owasp mobile application security. Documentazione online OWASP, 2025. Ultimo accesso: 18 Ottobre 2025.