BLU-MAN-006-R01
Allegato 08-R00
Aggiornamento del 08/09/2026

Flutter

Italgas S.p.A - Informazioni Interne

INDICE

1 INTRODUZIONE AL DOCUMENTO 3
1.1 Contesto 3
1.2 Scopo 3
1.3 Struttura del Documento 3
2 ACCESS CONTROL & PRIVILEGE MANAGEMENT 5
2.1 Secured Storage 5
2.2 Autenticazione locale 5
3 CODE OBFUSCATION 6
3.1 Code Obfuscation in Flutter 6
4 INFORMAZIONI CONFIDENZIALI 6
4.1 Protezione degli snapshot 6
5 ROOT DETECTION 6
5.1 Root Detection in Flutter 7
6 CRITTOGRAFIA 8
6.1 SSL Pinning 8
7 AGGIORNAMENTO DELLE DIPENDENZE 10
8 Link utili 10

1 INTRODUZIONE AL DOCUMENTO

1.1 Contesto

Questo documento è uno degli allegati al Manuale Operativo Sviluppo Sicuro e si colloca all'interno di un articolato sistema di linee guida volte a garantire la sicurezza nello sviluppo dei prodotti commissionati da Italgas. L'obiettivo è assicurare che tali prodotti siano solidi e resistenti alle minacce informatiche. È stato redatto un documento generale che affronta l'argomento in maniera trasversale senza entrare nei dettagli specifici di ciascuna tecnologia. Il presente documento, invece, approfondisce le stesse tematiche trattate nel documento generale, focalizzandosi e contestualizzandole sulla tecnologia Flutter.

1.2 Scopo

Questo documento ha lo scopo di illustrare gli aspetti di sicurezza informatica da tenere in considerazione quando si sviluppa del codice applicativo. La sicurezza del software è un requisito fondamentale per garantire la protezione delle informazioni, la privacy degli utenti e il corretto funzionamento dell'applicazione finale.
In questo documento, le tematiche affrontate si concentrano sulle possibili problematiche di sicurezza che affliggono le odierne applicazioni. Nello specifico, si fa riferimento alla tecnologia Flutter. Per ogni aspetto di sicurezza viene fornita una descrizione, degli esempi, le corrispettive contromisure e gli accorgimenti da adottare al fine di scongiurare la presenza di vulnerabilità o bug all'interno dell'applicazione finale. Per ogni tematica trattata, l'obiettivo è quello di fornire agli sviluppatori e agli architetti una lista di problematiche di sicurezza che possono emergere come conseguenza di una programmazione non orientata agli aspetti di sicurezza.

1.3 Struttura del Documento

Nel presente documento verranno trattate diverse tematiche riguardanti il linguaggio Flutter, partendo da un *livello di astrazione* più alto andando man mano scendendo verso aspetti più pratici e tecnici. Gli argomenti discussi riguardano prettamente il linguaggio e le sue peculiarità. Pertanto, si rimanda al documento di linee guida generali per lo sviluppo sicuro che sviscera i problemi in maniera più astratta senza scendere nel dettaglio del linguaggio.

Viene successivamente proposto un capitolo relativo all'**Access Control**, presentando diversi scenari che pongono l'attenzione sull'importanza di attuare adeguati meccanismi di *controllo degli accessi* e sulle problematiche di sicurezza che può comportare una mancata o scorretta implementazione di questi.

Il concetto di Access Control viene quindi esteso da quello di **Privilege Management**. La *gestione dei privilegi* è un aspetto fortemente correlato a quello del *controllo degli accessi*: implementare una solida gestione dei privilegi significa ridurre il rischio di abusi e attacchi informatici, garantendo che solo gli utenti autorizzati possano eseguire operazioni sensibili.

La **Code Obfuscation**, traducibile come offuscamento del codice, è il processo di modifica del file binario di un'applicazione al fine di renderlo più difficile da comprendere per gli esseri umani. Difatti, tecniche come ad esempio il Reverse Engineering possono esporre chiavi API, classi, nomi di funzioni e tutte le stringhe e in generale rendere di facile comprensione le logiche di funzionamento dell'applicativo. Queste informazioni potrebbero consentire ad utenti malintenzionati di accedere ai dati sensibili.

Le **Informazioni Confidenziali**, il cui valore deve essere conosciuto solo e solamente dalle persone o programmi autorizzati, devono essere protette durante tutta la fase di vita del software. Questo include i dati in uso e a riposo, oltre che quando in transito. È per questo motivo che non è sufficiente scrivere il segreto direttamente nei file sorgente o utilizzare algoritmi di cifratura robustissimi solo in alcuni passaggi, in quanto è necessario che le informazioni confidenziali restino tali sempre.

La **Root Detection**, invece, è una tecnica presente in Flutter che permette di verificare Runtime i permessi dell'utente che ha avviato il processo. Questo per evitare che un utente privilegiato, chiamato solitamente

root, possa avviare l'applicativo con tali permessi. È una pratica comune per garantire la sicurezza dell'applicativo e dei dati con cui questo lavora.

La crittografia è alla base di molte delle operazioni che un software si trova ad eseguire durante il suo ciclo di vita, permettendo di implementare comunicazioni sicure e proteggere i dati da accessi non autorizzati.
La **Gestione della Crittografia** costituisce un aspetto fondamentale da tenere in considerazione quando si valuta il livello di sicurezza di un software: l'utilizzo di tecniche crittografiche deprecate e non più sicure o una gestione impropria delle chiavi crittografiche possono vanificare le più robuste misure di sicurezza di un sistema.

Infine, per quanto concerne l'**Utilizzo di Codice di Terze Parti,** si rimanda al documento di linee guida generali per lo sviluppo sicuro, essendo le best practice in materia valide trasversalmente a tutte le tecnologie.

Italgas S.p.A - Informazioni Interne

2 ACCESS CONTROL & PRIVILEGE MANAGEMENT

Il controllo degli accessi e la gestione dei privilegi assicurano la confidenzialità dei dati, e che le operazioni su questi siano definite da una rigorosa policy. Se quest'ultima è troppo lasca, o se il sistema presenta debolezze che ne minano la robustezza, c'è il serio rischio di incorrere in vulnerabilità quali information disclosure a individui non autorizzati o privilege escalation.

2.1 Secured Storage

Le "Shared Preferences" sono uno dei meccanismi più comuni per memorizzare in modo persistente semplici coppie chiave-valore all'interno di un'applicazione Flutter. Questi dati vengono salvati in una struttura anch'essa di tipo chiave-valore, e possono essere recuperati anche dopo che l'app è stata chiusa e riavviata. Ad esempio, sono spesso utilizzate per memorizzare credenziali di accesso o token di autenticazione. Viene mostrato un esempio.

```
saveStringValue () async
 {
  prefs = await SharedPreferences.getInstance();
  prefs.setString("username", "franco");
 }
```

Tuttavia, il loro utilizzo comporta alcuni rischi. Se si consulta il Readme del plugin shared_preferences in pub.dev, non si troverà alcuna garanzia che le scritture per questo saranno effettivamente persistenti su disco.

Le Shared Preferences o SQLite possono rappresentare un altro problema per i dati critici degli utenti. Poiché i dati non sono crittografati, possono essere aperti su qualsiasi altro dispositivo, causando fughe di dati.

Per garantire la crittografia si può utilizzare il plugin flutter_secure_storage, che utilizza Keychain per iOS e AESEncryption per Android. Il link alla guida ufficiale è riportato nel paragrafo *"Link Utili"*.

2.2 Autenticazione locale

L'autenticazione locale, a differenza della classica autenticazione tramite la combinazione di username e password ad un servizio remoto (solitamente basato su un'architettura Client-Server), si svolge interamente sul dispositivo utente.

Ovviamente non si riferisce all'autenticazione al servizio fornito, che altrimenti dovrebbe esporre al client le credenziali da convalidare (rendendo inutile in primo luogo l'autenticazione), ma fa riferimento all'autenticazione successiva allo sblocco dello schermo per confermare l'identità dell'utilizzatore del dispositivo. Questa va implementata per passaggi critici, come conferme di pagamento, mediante autenticazione biometrica in ambiente iOS (Touch ID o codice di sblocco) e API per le impronte digitali in Android (introdotte con And. v6.0).

Il package local_auth, la cui esaustiva guida ufficiale è riportata nel paragrafo "*Link Utili*" in coda al capitolo, risponde all'esigenza.

3 CODE OBFUSCATION

L'offuscamento del codice è il processo di modifica del file binario di un'applicazione al fine di renderlo più difficile da comprendere per gli esseri umani. Difatti, tecniche come ad esempio il Reverse Engineering possono esporre chiavi API, classi, nomi di funzioni e tutte le stringhe e in generale rendere di facile comprensione le logiche di funzionamento dell'applicativo. Queste informazioni potrebbero consentire ad utenti malintenzionati di accedere ai dati sensibili.

3.1 Code Obfuscation in Flutter

In Flutter è possibile utilizzare tecniche per offuscare le applicazioni. A tal fine, è necessario compilare la versione rilasciata dell'applicativo utilizzando il flag --obfuscate, combinato con il flag --split-debug-info, come mostrato di seguito:

```
flutter build apk –obfuscate –split-debug-info=/<project-name>/<directory>
```

Cosi facendo viene generato un file simbolico, che può essere usato per deoffuscare lo stack trace, usando flutter symbolize, come mostrato di seguito:

```
flutter symbolize -i <stack trace file> -d /out/android/app.android-arm64.symbols
```

L'offuscamento del codice in Flutter è attualmente supportato per le build di rilascio su Android, iOS e MacOS.

4 INFORMAZIONI CONFIDENZIALI

Tutte quelle informazioni che sono classificate come confidenziali, le credenziali di accesso ad un sistema informatico su tutte, devono essere protette in maniera adeguata. Disporre di un forte algoritmo di autenticazione e verifica dell'identità potrebbe essere vano se le credenziali sono facilmente violabili

4.1 Protezione degli snapshot

Quando un'applicazione mobile è in background, viene visualizzato automaticamente uno snapshot dell'ultimo stato dell'app. Sebbene sia utile passare da un'app all'altra, non è auspicabile mostrare dati sensibili dell'utente. Questo è particolarmente vero in caso di applicazioni che contengono dati sensibili dell'utente, app bancarie, pagamenti e così via. Al fine di risolvere la problematica si suggerisce di usare uno specifico plugin di sicurezza i cui dettagli sono riportati nei link utili.

5 ROOT DETECTION

In ambiente Android, ma non solo, è pratica comune ottenere i permessi di amministratore sul dispositivo, pur essendo questo comportamento spesso in violazione delle policy sottoscritte con il vendor del prodotto. I permessi di amministratore ampliano quelli che sono i normali privilegi utente, permettendo di avere il

controllo totale delle risorse hardware e software del dispositivo, ampliando la superficie d'attacco ed esponendo l'intero sistema.

5.1 Root Detection in Flutter

Per evitare che l'applicativo Flutter venga eseguito dall'utente con i permessi di amministratore, in Android come in iOS, esiste una libreria chiamata flutter_jailbreak_detection che permette di verificare se l'applicativo è stato lanciato da un'utenza privilegiata o meno.

In base all'utilizzo o meno dei privilegi di amministratore l'applicativo dovrebbe essere per design in grado di rispondere, ad esempio, impedendo l'accesso all'applicazione ed evitando alcuna interazione con i dati da essa gestiti.

6 CRITTOGRAFIA

Gli algoritmi crittografici servono a diversi scopi, tra cui la cifratura e la decifratura dei dati, la firma di questi per verificarne l'integrità o che provengano da una fonte attendibile, la confidenzialità di una comunicazione ed il non ripudio.

I moderni algoritmi crittografici si basano interamente sulla segretezza delle loro chiavi per la sicurezza; gli algoritmi stessi sono ben noti e studiati con attenzione. Questo principio è noto comunemente come il **Principio di Kerckhoffs**.

6.1 SSL Pinning

La tecnica chiamata SSL Pinning si pone come obiettivo il fissare tutte le richieste di rete ad un ben definito dominio. Per questo motivo, il certificato di quest'ultimo è incluso (embedded) all'interno dell'applicativo stesso ed è verificato lato client per ogni richiesta inviata al server.

Nota: Il certificato usato dev'essere strutturato in formato **P**rivacy **E**nhanced **M**ail (PEM). Ne viene riportato un esempio.

```
-----BEGIN CERTIFICATE-----
... Certificate Content ...
-----END CERTIFICATE-----
```

Per prima cosa, è necessario aggiungere il package http come dipendenza all'interno del file di progetto pubspec.yaml

```
dependencies:
  http: 1.2.2
```

Per includere il certificato della CA è necessario includerlo nel folder *'assets'* del progetto, all'interno di un file *ca_certificate.pem*. Va poi aggiunto al file pubspec.yaml il seguente contenuto

```
flutter:
  assets:
    - assets/ca_certificate.pem
```

Infine, è necessario creare un custom HttpClient che adotti un Security Context con il certificato specificato precedentemente. Viene riportato un esempio di come è possibile implementarlo.

```
import 'dart:io';
import 'package:flutter/services.dart' show rootBundle;

Future<HttpClient> createHttpClientWithPinnedCertificate() async {
  final SecurityContext securityContext = SecurityContext();
  final String certificate = await rootBundle.loadString('assets/ca_certificate.pem');
  securityContext.setTrustedCertificatesBytes(certificate.codeUnits);
  return HttpClient(context: securityContext);
}
```

In questa funzione, creiamo un nuovo SecurityContext e carichiamo il contenuto del certificato CA usando rootBundle.loadString() dal package flutter/services.dart. Si impostano poi i certificati attendibili per il SecurityContext, utilizzando il metodo setTrustedCertificatesBytes().

Infine, creiamo un'istanza di HttpClient passando il SecurityContext al costruttore e restituendolo.

Per fare richieste HTTP con il pinning del certificato, dobbiamo usare l'HttpClient personalizzato creato nel passo precedente. Invece di usare le funzioni predefinite del pacchetto http, creeremo un'istanza http.IOClient con il nostro HttpClient personalizzato. Ecco un esempio di richiesta GET:

```
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:http/io_client.dart';

Future<void> main() async {
  final HttpClient httpClient = await createHttpClientWithPinnedCertificate();
  final http.IOClient ioClient = http.IOClient(httpClient);
  final response = await ioClient.get(Uri.parse('https://example.com'));
  if (response.statusCode == 200) {
    print(jsonDecode(response.body));
  } else {
    print('Failed to load data. StatusCode: ${response.statusCode}');
  }
  // Chiusura dell'oggetto ioClient
  ioClient.close();
}
```

Nell'esempio appena discusso, si crea un'istanza di HttpClient personalizzata, chiamando la funzione createHttpClientWithPinnedCertificate(). Quindi, si crea un http.IOClient passando l'istanza di HttpClient al suo costruttore.

È quindi possibile usare l'ioClient per fare richieste HTTP. In questo caso, si manda una richiesta GET a https://example.com. Si controlla allora il codice di stato della risposta e si stampa il corpo della risposta decodificato se la richiesta ha avuto successo. In caso contrario, viene stampato un messaggio di errore che indica il fallimento.

Nota: È altresì importante chiudere ioClient con il metodo close(). Questo per liberare le risorse di sistema ed evitare la saturazione della memoria.

Per ulteriori informazioni, in particolare nell'utilizzo del package http, si rimanda alla guida ufficiale linkata in coda al capitolo.

7 AGGIORNAMENTO DELLE DIPENDENZE

Essendo Flutter un software development kit per le UI rilasciato recentemente da Google, nel 2017, viene costantemente aggiornato al fine di correggere i bug di sicurezza riscontrati nelle versioni precedenti. Per tale ragione si suggerisce di evitare di far riferimento a versioni specifiche per le dipendenze e di assicurarsi di controllare periodicamente se le dipendenze hanno avuto aggiornamenti di sicurezza. Assicurarsi si usare la versione stabile più aggiornata di Flutter e delle sue dipendenze.

8 Link utili

- **Secured Storage**
  - shared_preferences | Flutter package (pub.dev)
  - flutter_secure_storage | Flutter package (pub.dev)
- **Autenticazione locale**

local_auth | Flutter package (pub.dev)

- **Code Obfuscation in Flutter**:
  - Obfuscate Dart code | Flutter
- **Protezione degli snapshot**
  - secure_application | Flutter package (pub.dev)
- **Root Detection in Flutter**
  - flutter_jailbreak_detection | Flutter package (pub.dev)
- **SSL Pinning**
  - http | Dart package (pub.dev)