Crittografia nelle applicazioni Android

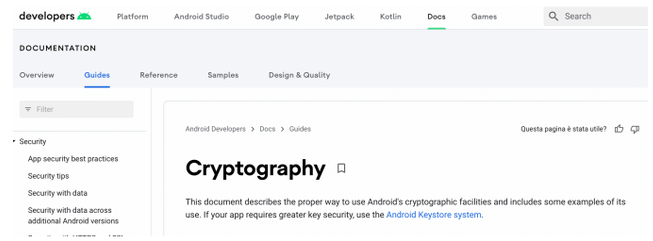
Best practies, vulnerabilità e analisi statica

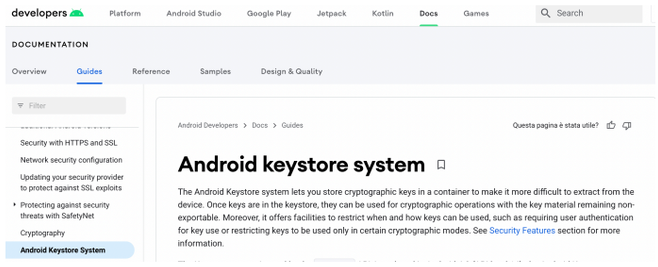
Matteo Maraniello M63001110

Introduzione

Top 10 Mobile Risks - Final List 2016

- M1: Improper Platform Usage
- M2: Insecure Data Storage
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorization
- M7: Client Code Quality
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality





Ormai la crittografia parte integrante di molte app, per messaggi, password, file multimediali. OWASP vulnerabilità, android developers best practices, base func crypto, e avanzata keystore.

Best Practies - Android Developer

Read/Write file

```
val fileToWrite = "my_sensitive_data.txt"
val encryptedFile = EncryptedFile.Builder(
    File(DIRECTORY, fileToWrite),
    applicationContext,
    mainKeyAlias,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()
```

Crittazione messaggio

```
byte[] plaintext = ...;
KeyGenerator keygen = KeyGenerator.getInstance("AES");
keygen.init(256);
SecretKey key = keygen.generateKey();
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] ciphertext = cipher.doFinal(plaintext);
byte[] iv = cipher.getIV();
```

Digest

```
byte[] message = ...;
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] digest = md.digest(message);
```

Firma digitale

```
byte[] message = ...;
PrivateKey key = ...;
Signature s = Signature.getInstance("SHA256withECDSA");
s.initSign(key);
s.update(message);
byte[] signature = s.sign();
```

```
byte[] message = ...;
byte[] signature = ...;
PublicKey key = ...;
Signature s = Signature.getInstance("SHA256withECDSA");
s.initVerify(key);
s.update(message);
boolean valid = s.verify(signature);
```

Per leggere/scrivere=>encrypted file; mess con cipher; per digest; come firma digitale e verifica

Best Practies - Raccomandazioni

Non consigliato Bouncy Castle Algorithm (problemi di compatibilità)

```
Cipher.getInstance("AES/CBC/PKCS7PADDING", "BC");
// OR
Cipher.getInstance("AES/CBC/PKCS7PADDING", Security.getProvider("BC"));
```

Uso di IV puramente randomici

```
SecretKey key = ...;
Cipher cipher = Cipher.getInstance("PBEWITHSHA256AND256BITAES-CBC-BC");
byte[] iv = new byte[16];
new SecureRandom().nextBytes(iv);
cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(iv));
```

Best Practies - Android Keystore

Gestore per le chiavi: il loro ciclo di vita, proteggendole da attività malevole e da un uso inappropriato di app terze.

```
KeyStore ks = KeyStore.getInstance("AndroidKeyStore");
ks.load(null);
Enumeration<String> aliases = ks.aliases();
```

```
KeyStore ks = KeyStore.getInstance("AndroidKeyStore");
ks.load(null);
KeyStore.Entry entry = ks.getEntry(alias, null);
if (!(entry instanceof PrivateKeyEntry)) {
    Log.w(TAG, "Not an instance of a PrivateKeyEntry");
    return null;
}
Signature s = Signature.getInstance("SHA256withECDSA");
s.initSign(((PrivateKeyEntry) entry).getPrivateKey());
s.update(data);
byte[] signature = s.sign();
```

Vedere certificati importati, uso delle chiavi tramite keystore; esistono anche funzioni avanzate per usare chiavi ad impronte biometriche (impronte), e se ancora + sicurezza, uso di HW dedicato

26 Rule

Columbia University (2020):

26 best practies per un buon uso delle strutture crittografiche nella programmazione di app mobile Android.

ID	Rule Description
R-01	Don't use broken hash functions (`SHA1`, `MD2`, `MD5`, ..)
R-02	Don't use broken encryption alg. (`RC2`, `DES`, `IDEA` ..)
R-03	Don't use the operation mode `ECB` with $>$ 1 data block
R-04	Don't use the operation mode `CBC` (client/server scenarios)
R-05	Don't use a static (= constant) key for encryption
R-06	Don't use a "badly-derived" key for encryption
R-07	Don't use a static (= constant) initialization vector (IV)
R-08	Don't use a "badly-derived" initialization vector (IV)
R-09	Don't reuse the initialization vector (IV) and key pairs
R-10	Don't use a static (= constant) salt for key derivation
R-11	Don't use a short salt ($<$ 64 bits) for key derivation
R-12	Don't use the same salt for different purposes
R-13	Don't use $<$ 1000 iterations for kev derivation
R-14	Don't use a weak password (score $<$ 3)
R-15	Don't use a NIST-black-listed password
R-16	Don't reuse a password multiple times
R-17	Don't use a static (= constant) seed for PRNG
R-18	Don't use an unsafe PRNG (java.util.Random)
R-19	Don't use a short key ($<$ 2048 bits) for RSA
R-20	Don't use the textbook (raw) algorithm for RSA
R-21	Don't use the padding `PKCS1-v1.5` for RSA
R-22	Don't use HTTP URL connections (use HTTPS)
R-23	Don't use a static (= constant) password for store
R-24	Don't verify host names in SSL in trivial ways
R-25	Don't verify certificates in SSL in trivial ways
R-26	Don't manually change the hostname verifier

Un altro buon punto di partenza: pubblicato una ricerca, dove, oltre a esporre un SW ideato da loro per una analisi dinamica CRYLOGGER, ha stilato 26 regole.

26 Rule - Approfondimento

Non usare hash functions e algoritmi di crittografia deprecati

R-01	Don't use broken hash functions (`SHA1`, `MD2`, `MD5`, ..)
R-02	Don't use broken encryption alg. (`RC2`, `DES`, `IDEA` ..)

MD5->sconsigliato dal 2012, Flame Malware

SHA1->deprecato dal NIST nel 2011 per problemi di collisioni

=>SHA2 SHA512

DES->deprecato per la facilità tramite attacchi a forza bruta di ottenere la chiave

=>AES

Tra queste regole, spiccano quelle abbastanza intuitive di non usare algoritmi di crittografia o algoritmi di hash che sono stati rotti, o che portino collisioni (R-01, R02->consiglio usare SHA2 digital, SHA512 storage pw) [flame: forgiava un certificato che aveva come CA Microsoft]; anche 3DES deprecato dal 2023

26 Rule - Approfondimento

Non usare valori costanti per la generazione di chiavi o IV, con funzioni sicure secondo la crittografia

R-05	Don't use a static (= constant) key for encryption
R-06	Don't use a "badly-derived" key for encryption
R-07	Don't use a static (= constant) initialization vector (IV)
R-08	Don't use a "badly-derived" initialization vector (IV)
R-09	Don't reuse the initialization vector (IV) and key pairs
R-10	Don't use a static (= constant) salt for key derivation
R-11	Don't use a short salt (< 64 bits) for key derivation
R-12	Don't use the same salt for different purposes

Uso della funzione *java.util.Random*-> non abbiamo numeri realmente casuali

=> java.security.SecureRandom (output non deterministico e non prodotto da funzione lineare, più bit [128], non uso del clock per il seed)

R-18	Don't use an unsafe PRNG (java.util.Random)

Random tramite una funzione matematica nota

Vulnerabilità

Lo studio della Columbia, basato su 1780 App Android provenienti dal Play Store, è stata effettuato basandosi proprio sulle 26 rule.

Risultati: tutte almeno una regola violata!

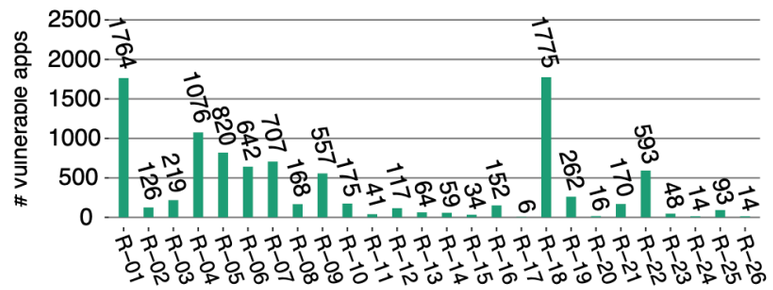


Fig. 7. Number of vulnerable Android apps for each crypto rule. We analyzed 1780 Android apps with *CRYLOGGER* configured to generate 30k random events with Monkey. We downloaded the apps from the official Google Play Store. The dataset of apps was collected between September and October 2019.

App dal play store; altri, R 18 non secure random; R 1 non hash sicuro; R 5 hardcoded key

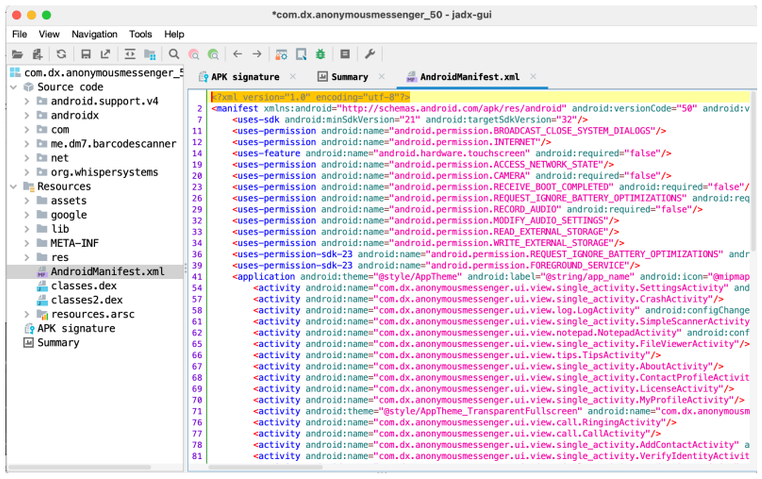
Analisi statica

Tool per analizzare APK android

Apktool

dex2jar

jadx-gui

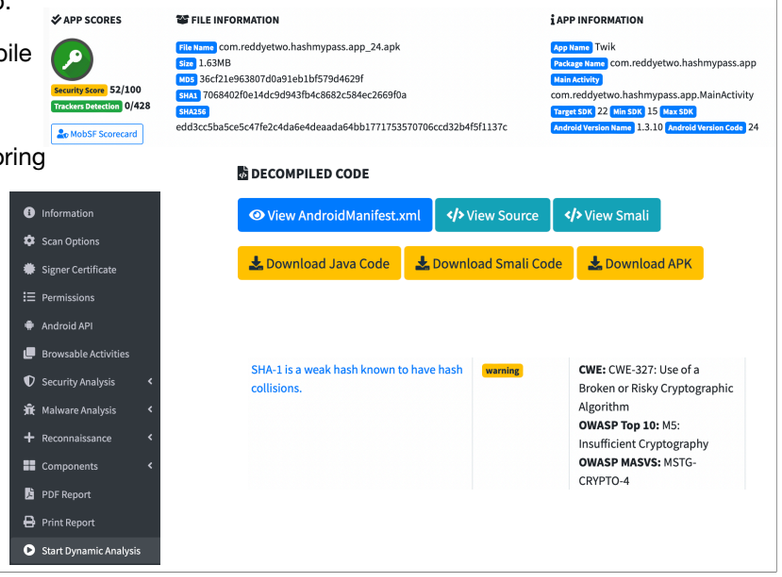


Statica=>solo codice senza eseguirlo, sia bytecode (.smali) o codice (.java)
1->spacchettare e rendere leggibili i file e il manifest 2->convertire file dei to jar, reverse eng 3->gui grafica per leggere apk o jar e fare ricerca per parole

Mobile Security Framework (MobSF)

Framework all-in-one, open source e gratuito:

- Conversione files dell'apk in formato leggibile
- Reverse Engineering dei file dex compilati
- Valutazioni preliminari sulla sicurezza e scoring
- Report da Manifest, codice, activity…

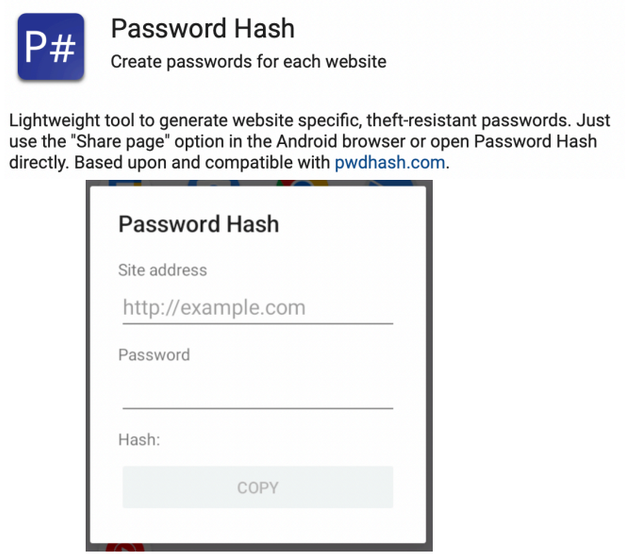


Come espressioni regolari o stringhe problemi di uso di determinate funzioni (MD5); cerca tali modelli. Ma non possiamo gestire tutti i problemi di sicurezza in questo modo, ma di aiuto. Tramite docker, si è installato MobSF da git-hub->andremo ad analizzare applicazioni free open source, da F-Droid, leggendo la signature, vedendo il codice

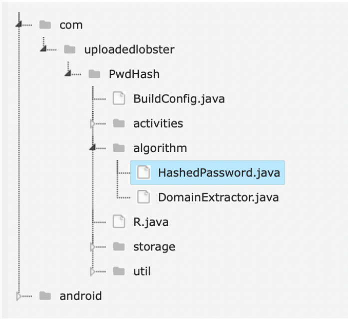
App analizzate-PasswordHash

PasswordHash:

App che permette di “creare” password tramite hash del url del sito e una password



Risultati-PasswordHash



SIGNER CERTIFICATE

```
APK is signed
v1 signature: True
v2 signature: False
v3 signature: False
Found 1 unique certificates
Subject: C=UK, ST=ORG, L=ORG, O=fdroid.org, OU=FDroid, CN=FDroid
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2012-07-21 17:10:49+00:00
Valid To: 2039-12-07 17:10:49+00:00
Issuer: C=UK, ST=ORG, L=ORG, O=fdroid.org, OU=FDroid, CN=FDroid
Serial Number: 0x500ae299
Hash Algorithm: sha1
md5: 16666dee66f74e0976e65a5ddf870700
sha1: b39266316ace38a0dd70ebabfa24d0ff1ff17952
sha256: f6c007b1b160c2928b1f14bd22ebcb97ee518503154ee44642eaea130af89939
sha512: 0b2b760cf944b3ff35e6558a7d6000aed477c787c60ebb7d8a2a0dcadbed7355bbe769a31f55f4bf0d5abcce3aa8e8c77
```

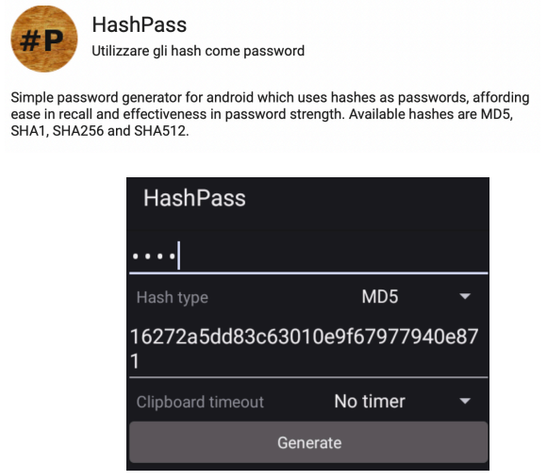
Certificato: come si può ben vedere, è siglato con ***SHA1***, deprecato, come anche le pw vengono generate usando ***HMAC_MD5***! BAD

```java
    private static byte[] createHmacMD5(String str, String str2) {
        if (str == null || str.equals("")) {
            throw new IllegalArgumentException("key must not be null or empty");
        } else if (str2 != null) {
            byte[] encodeStringToBytes = encodeStringToBytes(str);
            byte[] encodeStringToBytes2 = encodeStringToBytes(str2);
            try {
                Mac mac = Mac.getInstance(HMAC_MD5);
                mac.init(new SecretKeySpec(encodeStringToBytes, HMAC_MD5));
                return mac.doFinal(encodeStringToBytes2);
            } catch (InvalidKeyException e) {
                Log.e(HashedPassword.class.getName(), "Invalid secret key.", e);
                return new byte[0];
            } catch (NoSuchAlgorithmException e2) {
                Log.e(HashedPassword.class.getName(), "HMAC_MD5 algorithm not supported on this platform.", e2);
                return new byte[0];
            }
        } else {
            throw new IllegalArgumentException("data must not be null");
        }
    }
```

App Analizzate-HashPass

HashPass:

App che permette di “creare” hash da usare come password a partire da una stringa, a scelta tra più algoritmi di hash



Risultati-HashPass

Come si notava anche nella descrizione dell'app, permette us o di più hash, ***MD5, SHA1***, SHA256, SHA512, ma primi due deprecati! BAD

```
@Override // android.view.View.OnClickListener
public void onClick(View v) {
    if (v != this.generate || this.input.getText().toString().length() == 0) {
        Toast.makeText(this, "input must not be empty", 0).show();
        return;
    }
    ClipboardManager clipboard = (ClipboardManager) getSystemService("clipboard");
    if (this.hash_choice.equals(getResources().getString(R.string.md5)) || this.hash_choice.equals(getResources().getString(R.string.sha1))
        this.output_str = createHash(this.input.getText().toString(), this.hash_choice);
        this.output.setText(this.output_str);
        clipboard.setText(this.output_str);
        Toast.makeText(this, "hash copied to clipboard", 0).show();
        if (this.timeout != 0) {
            PostHash posthash = new PostHash();
            posthash.execute(Integer.valueOf(this.timeout));
        }
    }
}
```

```
private String createHash(String input, String type) {
    try {
        byte[] bytesOfMessage = input.getBytes("UTF-8");
        try {
            MessageDigest md = MessageDigest.getInstance(type);
            byte[] thedigest = md.digest(bytesOfMessage);
            char[] HEX_CHARS = "0123456789abcdef".toCharArray();
            StringBuilder sb = new StringBuilder(thedigest.length * 2);
            for (byte b : thedigest) {
                sb.append(HEX_CHARS[(b & 240) >> 4]);
                sb.append(HEX_CHARS[b & 15]);
            }
            return sb.toString();
        } catch (NoSuchAlgorithmException nsae) {
            nsae.printStackTrace();
            return null;
        }
    } catch (UnsupportedEncodingException uee) {
        uee.printStackTrace();
    }
}
```

App Analizzate-Messanger Anonimo

Messanger Anonimo:

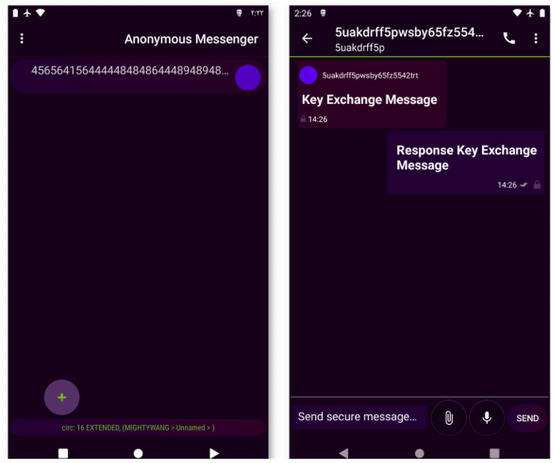
App che permette di instaurare connessioni totalmente private tramite tor.
Ci si registra con user e password, per poi scambiarsi una chiave tramite QR Code e comunicare anche con file

Messenger anonimo

Messenger FOSS "peer to peer" privato, anonimo e sicuro che utilizza Tor.

Novità nella versione 0.8.14

Fixed alert overlay style.
Design and performance changes.
Added no password mode when starting a new account.
Saved vertical space with new message design.
Changes in verify identity design.
Fixed country route filter.
Removed strong password restrictions.
Updated Tor to 0.4.6.8.
Changed main menu to the right side.



Risultati-Messanger Anonimo

Uso di SHA1! BAD

```
public static byte[] secretToKey(byte[] secret, byte[] specifier) {
    try {
        MessageDigest messageDigest = MessageDigest.getInstance("SHA-1");
        int i = specifier[8] & 255;
        int i2 = ((i & 15) + 16) << ((i >> 4) + 6);
        int length = secret.length + 8;
        byte[] bArr = new byte[length];
        System.arraycopy(specifier, 0, bArr, 0, 8);
        System.arraycopy(secret, 0, bArr, 8, secret.length);
        while (i2 > 0) {
            if (i2 >= length) {
                messageDigest.update(bArr);
                i2 -= length;
            } else {
                messageDigest.update(bArr, 0, i2);
                i2 = 0;
            }
        }
        byte[] bArr2 = new byte[29];
        System.arraycopy(messageDigest.digest(), 0, bArr2, 9, 20);
        System.arraycopy(specifier, 0, bArr2, 0, 9);
        return bArr2;
    } catch (NoSuchAlgorithmException unused) {
        throw new RuntimeException("Can't run without sha-1.");
    }
}
```

Uso non sicuro crittograficamente di SHA1! (per generare numeri non randomici) BAD

```
public static int generateRegistrationId(boolean extendedRange) {
    try {
        SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");
        if (extendedRange) {
            return secureRandom.nextInt(2147483646) + 1;
        }
        return secureRandom.nextInt(16380) + 1;
    } catch (NoSuchAlgorithmException e) {
        throw new AssertionError(e);
    }
}
```

Tor package sicuri per le connessioni:
net.sf.freehaven.tor OK

```
public static Socket getCallSocket(String OnionAddress, DxApplication app, String type) {
    try {
        Socket socks5SocketConnection = socks5SocketConnection(OnionAddress, 5780, "127.0.0.1", app.getTorSocket().getOnionProxyManager().getIPv4LocalHostSocksPort());
        DataOutputStream dataOutputStream = new DataOutputStream(socks5SocketConnection.getOutputStream());
        DataInputStream dataInputStream = new DataInputStream(socks5SocketConnection.getInputStream());
        dataOutputStream.writeUTF(NotificationCompat.CATEGORY_CALL);
        dataOutputStream.flush();
        if (dataInputStream.readUTF().contains("ok")) {
            dataOutputStream.writeUTF(app.getHostname());
            dataOutputStream.flush();
            if (dataInputStream.readUTF().contains("ok")) {
                dataOutputStream.writeUTF(type);
                dataOutputStream.flush();
                return socks5SocketConnection;
            }
        }
    } catch (Exception unused) {
    }
    return null;
}

public static boolean sendMedia(String onion, DxApplication app, String msg, byte[] media, boolean isProfileImage) {
    Socket socket = null;
    try {
        Socket socks5SocketConnection = socks5SocketConnection(onion, 5780, "127.0.0.1", app.getTorSocket().getOnionProxyManager().getIPv4LocalHostSocksPort());
        DataOutputStream dataOutputStream = new DataOutputStream(socks5SocketConnection.getOutputStream());
        DataInputStream dataInputStream = new DataInputStream(socks5SocketConnection.getInputStream());
        if (isProfileImage) {
            dataOutputStream.writeUTF("profile_image");
        } else {
            dataOutputStream.writeUTF("media");
        }
```

AES/GCM/NoPadding per criptare e decrittare i files OK

```
if (dataInputStream.readUTF().contains("ok")) {
    byte[] sha256 = app.getSha256();
    Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
    app.sendNotificationWithProgress(app.getString(R.string.sending_file), app
    int i5 = 0;
```

Come conserva le chiavi, uso dei *KeyStore* OK

```
SenderKeyRecord loadSenderKey = this.senderKeyStore.loadSenderKey(this.senderKeyId);
if (!loadSenderKey.isEmpty()) {
    SenderKeyMessage senderKeyMessage = new SenderKeyMessage(senderKeyMessageBytes);
    SenderKeyState senderKeyState = loadSenderKey.getSenderKeyState(senderKeyMessage.getKeyId());
    senderKeyMessage.verifySignature(senderKeyState.getSigningKeyPublic());
    SenderMessageKey senderKey = getSenderKey(senderKeyState, senderKeyMessage.getIteration());
    plainText = getPlainText(senderKey.getIv(), senderKey.getCipherKey(), senderKeyMessage.getCipherText());
    callback.handlePlaintext(plainText);
    this.senderKeyStore.storeSenderKey(this.senderKeyId, loadSenderKey);
```

Uso non consigliato di *cipher.init*
(parametri variabili stanziate) BAD

```
int i3 = 2;
int i4 = 0;
try {
```

```
i2.read(bArr, i4, 12);
cipher.init(i3, new SecretKeySpec(sha256, "AES"), new GCMParameterSpec(128, bArr));
byte[] bArr2 = new byte[4];
```

Uso di AES/CBC/PKC5Padding per decrittare e criptare i messaggi di testo: sconsigliato se usato nelle interazioni con i database ma non è questo il caso. BAD--OK

```
private byte[] getCipherText(byte[] iv, byte[] key, byte[] plaintext) {
    try {
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(1, new SecretKeySpec(key, "AES"), ivParameterSpec);
        return cipher.doFinal(plaintext);
```

```
private byte[] getPlainText(byte[] iv, byte[] key, byte[] ciphertext) throws InvalidMessageExceptio
    Object e;
    Throwable e2;
    try {
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(2, new SecretKeySpec(key, "AES"), ivParameterSpec);
        return cipher.doFinal(ciphertext);
```

Problema legato alla CBC mode e il padding PKC5 e 7, se errore di padding, return error message e attaccante con delle prove può arrivare al plaintext senza la chiave

App Analizzate-Photok

Photok:

App che permette di nascondere file multimediali in una cassaforte, protetta da password e da crittazione

Photok

Encrypt your photos on your device and keep them safe from others.

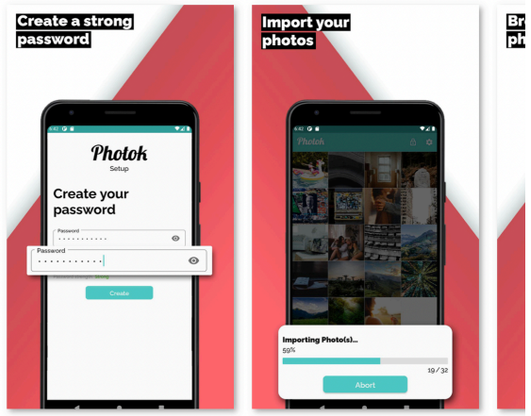
Photok is a free Photo-Safe. It stores your photos encrypted on your device and hides them from others.
It uses technologies like, AES-256 encryption standard or bcrypt, to keep your photos secure.
Photok is completely free, open source, and contains no ads.

Features

- import photos from your gallery
- fully functional in app gallery
- export photos back to your gallery
- change your password
- create and Restore backups
- share to Photok
- hide app icon

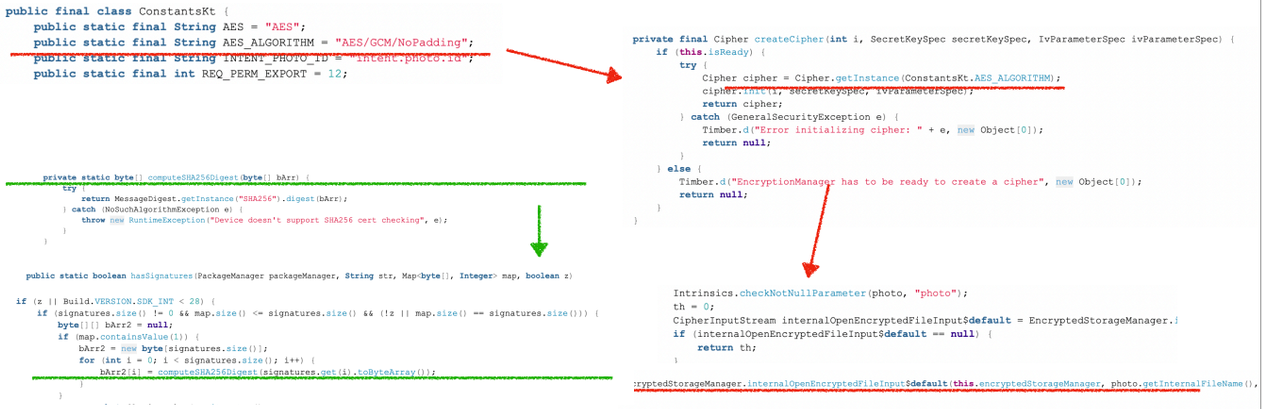
Benefits

- hide your sensitive photos from others
- protect your photos from data theft



Risultati-Photok

AES: crittazione e decrittazione file OK
SHA256: signatures per ogni file OK



App complessa, tanti file, niente trovato

Compass

App che permette di nascondere file multimediali in una cassaforte, protetta da password e da crittazione

Compass Vault - Hide Photos & Videos

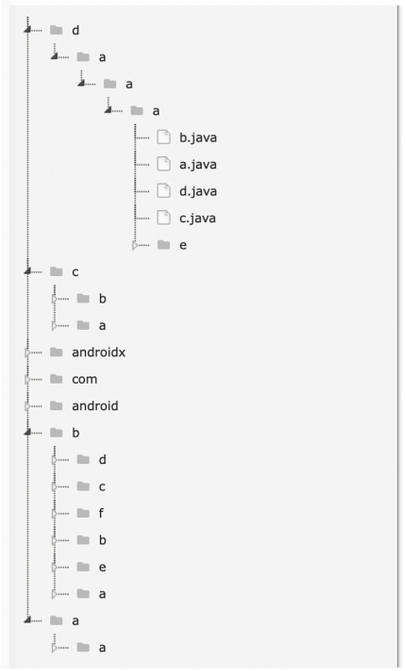
1.2.6 for Android

0 Reviews | 0 Posts

MizzOraninlky

Download APK (4.3 MB) Versions

MobSF con il suo report, presenta criticità, come uso di SHA1, MD5, PKCS7, ma come si nota dall'albero dei file, è veramente difficile districarsi.



Conclusioni

1-interessante analizzare app mobile; 2analizzato molte app, tutte almeno un problema crypto; 3-come detto, analisi statica, problema che non vedeva, 2a app vista, tool non nota; o anche al contrario, hardcoded!!