FYEO

Security Code Review of Censo Vault Android

Censo

December 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

EXECUTIVE SUMMARY

Overview

Censo engaged FYEO Inc. to perform a Security Code Review of Censo Vault Android.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on November 07 - December 05, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-CENSO-AND-01 – Google JWT Sub field is used for generating symmetric key for Approver key encryption
- FYEO-CENSO-AND-02 – The approver key may not be removed from the file system
- FYEO-CENSO-AND-03 – To generate a symmetric key, the hash function sha256 is used
- FYEO-CENSO-AND-04 – Custom implementation of Shamir's Secret Sharing Scheme
- FYEO-CENSO-AND-05 – Custom implementation of TOTP generator
- FYEO-CENSO-AND-06 – Unnecessary permissions have been added to the Owner and Guardian applications
- FYEO-CENSO-AND-07 – Applications allow to create backup
- FYEO-CENSO-AND-08 – Applications trust all pre-installed CAs

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of Censo Vault Android. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/Censo-Inc/vault-android with the commit hash cd578fb81b9247a17eb75f6d47d346679364137e.

Files included in the code review

```
vault-android/
├── guardian/
│   └── src/
│       └── main/
│           └── java/
│               └── co/
│                   └── censo/
│                       └── guardian/
│                           ├── di/
│                           │   └── AppModule.kt
│                           ├── presentation/
│                           │   ├── components/
│                           │   │   ├── ApproverCodeVerification.kt
│                           │   │   └── GuardianTopBar.kt
│                           │   ├── home/
│                           │   │   ├── GuardianHomeScreen.kt
│                           │   │   ├── GuardianHomeState.kt
│                           │   │   └── GuardianHomeViewModel.kt
│                           │   ├── GuardianColors.kt
│                           │   └── Screen.kt
│                           ├── service/
│                           │   └── MessagingService.kt
│                           ├── ui/
│                           │   └── theme/
│                           │       ├── Color.kt
│                           │       ├── Theme.kt
│                           │       └── Type.kt
│                           ├── GuardianApplication.kt
│                           └── MainActivity.kt
└── owner/
    └── src/
        └── main/
            └── java/
                └── co/
```

Files included in the code review

```
└── censo/
    └── vault/
        ├── data/
        │   ├── repository/
        │   │   └── FacetecRepository.kt
        │   └── FaceVerificationStatus.kt
        ├── di/
        │   └── AppModule.kt
        ├── presentation/
        │   ├── access_approval/
        │   │   ├── components/
        │   │   │   ├── AnotherDeviceAccessScreen.kt
        │   │   │   ├── ApproveAccessUI.kt
        │   │   │   ├── ApprovedUI.kt
        │   │   │   └── SelectApproverUI.kt
        │   │   ├── AccessApprovalScreen.kt
        │   │   ├── AccessApprovalState.kt
        │   │   └── AccessApprovalViewModel.kt
        │   ├── access_seed_phrases/
        │   │   ├── components/
        │   │   │   ├── AccessPhrasesTopBar.kt
        │   │   │   ├── ReadyToAccessPhrase.kt
        │   │   │   ├── SelectPhraseUI.kt
        │   │   │   └── ViewAccessPhraseUI.kt
        │   │   ├── AccessSeedPhrasesScreen.kt
        │   │   ├── AccessSeedPhrasesState.kt
        │   │   └── AccessSeedPhrasesViewModel.kt
        │   ├── components/
        │   │   ├── owner_information/
        │   │   │   ├── OwnerInformationField.kt
        │   │   │   ├── OwnerInformationRow.kt
        │   │   │   └── VerifyCode.kt
        │   │   ├── recovery/
        │   │   │   └── AccessPhrasesScreen.kt
        │   │   ├── security_plan/
        │   │   │   ├── SecurityPlanComponents.kt
        │   │   │   └── SecurityPlanScreens.kt
        │   │   ├── vault/
        │   │   │   ├── UnlockedVaultScreen.kt
        │   │   │   └── VaultComponents.kt
        │   │   ├── ActivateApproversComponents.kt
        │   │   ├── PhraseWords.kt
        │   │   ├── PushNotificationDialog.kt
```

Files included in the code review

```
│   │   ├── VaultButton.kt
│   │   └── YesNoDialog.kt
│   ├── enter_phrase/
│   │   ├── components/
│   │   │   ├── AddPhraseLabelUI.kt
│   │   │   ├── EditPhraseWordUI.kt
│   │   │   ├── EnterSeedPhraseMainScreen.kt
│   │   │   ├── PastePhraseUI.kt
│   │   │   ├── PhraseEntry.kt
│   │   │   ├── ReviewSeedPhraseUI.kt
│   │   │   └── ViewPhraseWordUI.kt
│   │   ├── EnterPhraseScreen.kt
│   │   ├── EnterPhraseState.kt
│   │   └── EnterPhraseViewModel.kt
│   ├── facetec_auth/
│   │   ├── FacetecAuth.kt
│   │   ├── FacetecAuthState.kt
│   │   └── FacetecAuthViewModel.kt
│   ├── initial_plan_setup/
│   │   ├── InitialPlanSetupScreen.kt
│   │   ├── InitialPlanSetupState.kt
│   │   └── InitialPlanSetupViewModel.kt
│   ├── lock_screen/
│   │   ├── components/
│   │   │   └── LockEngagedUI.kt
│   │   ├── LockScreen.kt
│   │   ├── LockScreenState.kt
│   │   └── LockScreenViewModel.kt
│   ├── main/
│   │   ├── ApproversHomeScreen.kt
│   │   ├── BottomNavBar.kt
│   │   ├── MainVaultScreen.kt
│   │   ├── PhraseHomeScreen.kt
│   │   ├── SettingsHomeScreen.kt
│   │   ├── VaultHomeScreen.kt
│   │   ├── VaultScreenState.kt
│   │   ├── VaultScreenViewModel.kt
│   │   └── VaultTopBar.kt
│   ├── plan_setup/
│   │   ├── components/
│   │   │   ├── ActivateApproverUI.kt
│   │   │   ├── AddAlternateApproverUI.kt
│   │   │   ├── AddTrustedApproversUI.kt
```

Files included in the code review

```
│   │   │   ├── ApproverNicknameUI.kt
│   │   │   ├── GetLiveWithApproverUI.kt
│   │   │   └── SavedAndShardedUI.kt
│   │   ├── PlanSetupScreen.kt
│   │   ├── PlanSetupState.kt
│   │   └── PlanSetupViewModel.kt
│   ├── welcome/
│   │   ├── WelcomeScreen.kt
│   │   ├── WelcomeState.kt
│   │   └── WelcomeViewModel.kt
│   ├── Screen.kt
│   └── VaultColors.kt
├── service/
│   └── MessagingService.kt
├── ui/
│   └── theme/
│       ├── Color.kt
│       ├── Theme.kt
│       └── Type.kt
├── util/
│   ├── NavUtils.kt
│   └── TestTag.kt
├── MainActivity.kt
└── VaultApplication.kt
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Code Review of Censo Vault Android, we discovered:

- 1 finding with HIGH severity rating.
- 2 findings with MEDIUM severity rating.
- 3 findings with LOW severity rating.
- 2 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

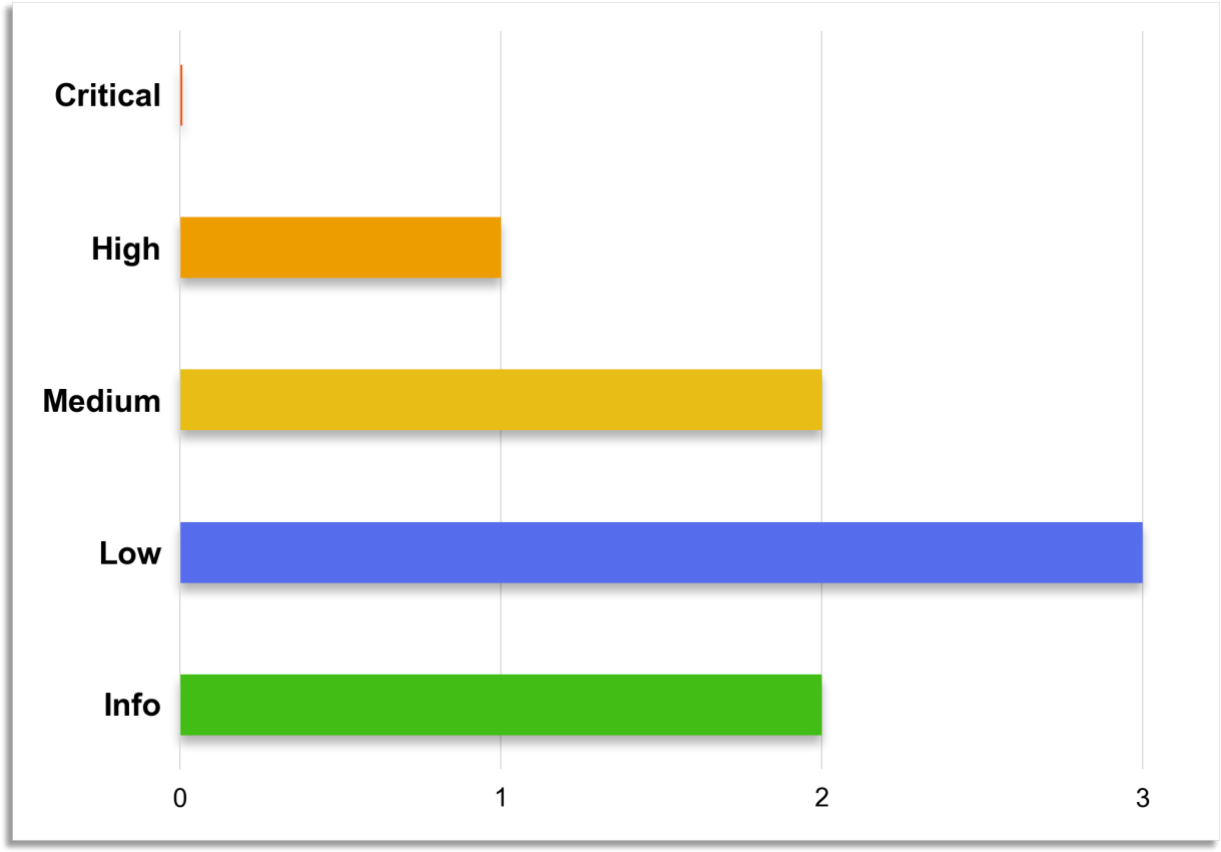


Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-CENSO-AND-01	**High**	Google JWT Sub field is used for generating symmetric key for Approver key encryption
FYEO-CENSO-AND-02	**Medium**	The approver key may not be removed from the file system
FYEO-CENSO-AND-03	**Medium**	To generate a symmetric key, the hash function sha256 is used
FYEO-CENSO-AND-04	**Low**	Custom implementation of Shamir's Secret Sharing Scheme
FYEO-CENSO-AND-05	**Low**	Custom implementation of TOTP generator
FYEO-CENSO-AND-06	**Low**	Unnecessary permissions have been added to the Owner and Guardian applications
FYEO-CENSO-AND-07	**Informational**	Applications allow to create backup
FYEO-CENSO-AND-08	**Informational**	Applications trust all pre-installed CAs

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

CONCLUSION

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

TECHNICAL FINDINGS

GENERAL OBSERVATIONS

This code review is part of the security audit of the Censo Seed Phrase Manager system.

The main functionality of the system is to provide the ability to securely store a seed phrase for crypto wallets. The system allows multiple participants to be responsible for storing a seed phrase. In particular, the supported schemes are:

- 1:1 - a scheme where the owner of the seed phrase acts as an approver,
- 2:2 - a scheme with the owner and one approver, and
- 2:3 - a scheme with the owner and two approvers, in which confirmation of one of the approvers is sufficient.

This audit covers the Vault and Guardian Android applications, which are the components of the Censo Seed Phrase Manager system. Vault is the main application for storing a seed phrase. Guardian application is an application for approvers.

Applications are developed for Android SDK API 33 and higher. This solution allows you to use security improvements, but at the same time limits the number of supported devices.

In our thorough examination of the code base, we dedicated significant attention to both logical and common errors.

The code is implementing the functionality as specified in the documentation and no real discrepancies from the documentation were identified during the audit.

However High priority issues were identified in the code in conjunction with the encryption of the verifiers' keys. All of these have been remediated.

The code also contains custom implementations of cryptographic algorithms, in particular the Shamir secret sharing scheme, this implementation has been reviewed as implemented in the code but no deep analysis has been performed of any third party libraries and dependencies.

GOOGLE JWT SUB FIELD IS USED FOR GENERATING SYMMETRIC KEY FOR APPROVER KEY ENCRYPTION

Finding ID: FYEO-CENSO-AND-01
Severity: **High**
Status: **Remediated**

Description

The symmetric key for encrypting the approver key is generated from the value of the Sub field of the Google JWT token.

File name: shared/src/main/java/co/censo/shared/data/repository/KeyRepository.kt
Line number: 92-95

```
val encryptedKey = SymmetricEncryption().encrypt(
    retrieveSavedDeviceId().sha256digest(),
    key.bigInt().toByteArrayNoSign()
)
```

retrieveSavedDeviceId() calls SecurePreferences.retrieveDeviceKeyId() method:

File name: shared/src/main/java/co/censo/shared/data/storage/SecurePreferences.kt
Line number: 142-143

```
override fun retrieveDeviceKeyId() =
    sharedPrefs.getString(DEVICE_KEY, "") ?: ""
```

If we look at the DeviceID saving method, we will see the following chain:

File name: shared/src/main/java/co/censo/shared/data/storage/SecurePreferences.kt
Line number: 136-140

```
override fun saveDeviceKeyId(id: String) {
    val editor = sharedPrefs.edit()
    editor.putString(DEVICE_KEY, id)
    editor.apply()
}
```

File name: shared/src/main/java/co/censo/shared/data/repository/KeyRepository.kt
Line number: 55-57

```
override fun setSavedDeviceId(id: String) {
    storage.saveDeviceKeyId(id)
}
```

File name: shared/src/main/java/co/censo/shared/data/repository/KeyRepository.kt
Line number: 50-53

```
override fun createAndSaveKeyWithId(id: String) {
    KeystoreHelper().getOrCreateDeviceKey(id)
    setSavedDeviceId(id)
}
```

And createAndSaveKeyWithId(id: String) is called in signInUser(jwt: String?), where id comes from verifyToken() method:

File name: shared/src/main/java/co/censo/shared/presentation/entrance/EntranceViewModel.kt
Line number: 198-203

```
val idToken = try {
    ownerRepository.verifyToken(jwt)
} catch (e: Exception) {
    googleAuthFailure(GoogleAuthError.FailedToVerifyId(e))
    return@launch
}
```

File name: shared/src/main/java/co/censo/shared/presentation/entrance/EntranceViewModel.kt
Line number: 211-219

```
if (!keyRepository.hasKeyWithId(idToken)) {
    try {
        keyRepository.createAndSaveKeyWithId(idToken)
    } catch (e: Exception) {
        googleAuthFailure(GoogleAuthError.FailedToCreateKeyWithId(e))
    }
} else {
    keyRepository.setSavedDeviceId(idToken)
}
```

And verifyToken() returns subject field in payload

File name: shared/src/main/java/co/censo/shared/util/AuthUtil.kt
Line number: 111-120

```
override fun verifyToken(token: String): String? {
    val verifier = GoogleIdTokenVerifier.Builder(
        NetHttpTransport(), GsonFactory()
    )
    .setAudience(BuildConfig.GOOGLE_AUTH_CLIENT_IDS.toList())
    .build()

    val verifiedIdToken: GoogleIdToken? = verifier.verify(token)
    return verifiedIdToken?.payload?.subject
}
```

For Google authentication, the .sub claim is unique to the user account and remains the same across different applications that the user might authorize. In other words, it is global per user, not per app, which helps in identifying the user across different applications. this means that any third party where the user have authenticated will have access to the sub that is used as the key derivement for this encryption. rendering the encryption more or less useless against an determined attacker.

Severity and Impact Summary

Public information is used to derive the symmetric key. This allows third parties to obtain the key, thereby significantly increasing the risk of compromising the approval key.

Recommendation

We recommend reconsidering the approach to generating a symmetric key for encrypting the approver key and eliminating the use of publicly available data as source material for the key.

References

https://developers.google.com/identity/openid-connect/openid-connect#an-id-tokens-payload

THE APPROVER KEY MAY NOT BE REMOVED FROM THE FILE SYSTEM

Finding ID: FYEO-CENSO-AND-02
Severity: **Medium**
Status: **Remediated**

Description

Before uploading the encrypted approver key to Google Drive, the key is temporarily saved in the external directory of the application. Once uploaded, the encrypted key file should be deleted, but this may not happen if an exception was thrown earlier.

Proof of Issue

File name: shared/src/main/java/co/censo/shared/data/storage/CloudStorage.kt
Line number: 69 - 95

```
 try {
     val fileMetaData = com.google.api.services.drive.model.File()
     fileMetaData.name = fileName

     val mediaContent = FileContent(FILE_MIME_TYPE, localFile)

     val uploadedFile = driveService.files().create(fileMetaData, mediaContent)
         .setFields(ID_FIELD)
         .execute()

     //Delete the temp local file
     localFile.delete()
     projectLog(message = "File upload process finished")
     return if (uploadedFile != null && uploadedFile.id != null ) {
         projectLog(message = "File uploaded with id: ${uploadedFile.id}")
         Resource.Success(Unit)
     } else {
         projectLog(message = "File upload failed")
         Resource.Error()
     }
 } catch (e: GoogleJsonResponseException) {
    projectLog(message = "File upload failed, with google json response exception:
$e")
    return Resource.Error(exception = e)
 } catch (e: Exception) {
     projectLog(message = "File upload failed, with exception: $e")
     return Resource.Error(exception = e)
 }
```

If any of the calls up to the

```
localFile.delete()
```

result in an exception, the encrypted approver key will remain on the file system and can be retrieved without elevated privileges.

Although the attacker will still need to decrypt the key, key presence in external storage increases the chances of compromising the system.

Severity and Impact Summary

An attacker can obtain a copy of the encrypted approver key.

Recommendation

It is recommended to add key deletion regardless of whether the file upload to Google Drive is successful or not.

TO GENERATE A SYMMETRIC KEY, THE HASH FUNCTION SHA256 IS USED

Finding ID: FYEO-CENSO-AND-03
Severity: **Medium**
Status: **Remediated**

Description

The key to encrypt the approver(guardian) key is the sha256 hash of the sub field in the Google JWT token.

Proof of Issue

File name: shared/src/main/java/co/censo/shared/data/repository/KeyRepository.kt
Line number: 92-95

```
val encryptedKey = SymmetricEncryption().encrypt(
    retrieveSavedDeviceId().sha256digest(),
    key.bigInt().toByteArrayNoSign()
)
```

Using sha256() to directly generate a symmetric key is generally not recommended for secure key generation. While SHA-256 produces a fixed-length hash value from an input, it doesn't provide the necessary properties for generating secure symmetric keys. Keys generated in this way would not be truly random or unpredictable.

Severity and Impact Summary

Symmetric key generation is simplified, which increases the chances of decrypting the Approver key.

Recommendation

We recommend reading the Recommendation for Cryptographic Key Generation and considering the possibility of generating a symmetric key through the KDF function. As a KDF, you may consider modern sponge-based functions, for example, the Keccak function family.

References

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133r2.pdf

https://keccak.team/obsolete/Keccak-main-2.1.pdf

CUSTOM IMPLEMENTATION OF SHAMIR'S SECRET SHARING SCHEME

Finding ID: FYEO-CENSO-AND-04
Severity: **Low**
Status: **Open**

Description

The Owner application uses a custom implementation of Shamir's secret sharing scheme to split the Intermediate key and recover it.

Proof of Issue

File name: shared/src/main/java/co/censo/shared/data/cryptography/SecretSharer.kt
Line number: 166-193

```
class SecretSharer(val secret: BigInteger, val threshold: Int, val participants:
List<BigInteger>, val order: BigInteger = ORDER) {
    val shards : List<Point> = getShares(participants, threshold, secret)

    fun vandermonde(participants: List<BigInteger>, threshold: Int) =
SecretSharerUtils.vandermonde(participants, threshold, order)
    fun dotProduct(matrix: Matrix, vector: Vector) =
SecretSharerUtils.dotProduct(matrix, vector, order)
    fun invertLUP(lu: Matrix, p: Vector) = SecretSharerUtils.invertLUP(lu, p, order)
    fun decomposeLUP(matrix: Matrix) = SecretSharerUtils.decomposeLUP(matrix, order)
    fun addShards(shares: List<BigInteger>, weights: List<BigInteger>) =
SecretSharerUtils.addShares(shares, weights, order)
    fun recoverSecret(shares: List<Point>) = SecretSharerUtils.recoverSecret(shares,
order)

    fun getShares(participants: List<BigInteger>, threshold: Int, secret: BigInteger):
List<Point> {
        if (threshold > participants.size) {
            throw Exception("secret would be irrecoverable.")
        }
        val vec = Vector(threshold) { if (it == 0) secret else
SecretSharerUtils.randomFieldElement(order) }
        val rowShares = dotProduct(vandermonde(participants, threshold), vec)
        return participants.mapIndexed { i, p ->
            Point(p, rowShares[i])
        }
    }

    fun getReshares(newParticipants: List<BigInteger>, newThreshold: Int):
List<List<Point>> {
        // generate the reshares from the first threshold participants
        return participants.take(threshold).indices.map { i ->
            getShares(newParticipants, newThreshold, shards[i].y)
        }
    }
}
```

It is common practice to use well-established and widely accepted implementations of cryptographic algorithms. In the case of Shamir's scheme, such implementations are indeed not available for the Android platform. However, custom implementation of cryptographic algorithms carries significant risks for the security of the system as a whole. In addition, cryptography is a rapidly evolving field. New attacks and vulnerabilities are discovered, and cryptographic standards are updated accordingly. Maintaining a custom cryptographic implementation requires constant vigilance to stay abreast of the latest developments, which can be a resource-intensive task.

Severity and Impact Summary

A custom implementation of the cryptographic algorithm is used to split and recover the Intermediate key. If there are vulnerabilities in this algorithm, the risk of system compromise increases significantly.

CUSTOM IMPLEMENTATION OF TOTP GENERATOR

Finding ID: FYEO-CENSO-AND-05
Severity: **Low**
Status: **Open**

Description

A custom TOTP implementation is used to generate TOTP.

Proof of Issue

File name: shared/src/main/java/co/censo/shared/data/cryptography/CryptoUtils.kt
Line number: 17-57

```
object TotpGenerator {
    const val CODE_LENGTH = 6
    const val CODE_EXPIRATION = 60L

    fun generateSecret(): String {
        val alphaChars = ('0'..'9').toList().toTypedArray() +
('a'..'z').toList().toTypedArray() + ('A'..'Z').toList().toTypedArray()
        return Base32()
            .encodeAsString((1..10).map { alphaChars.random().toChar()
}.toMutableList().joinToString("").toByteArray())
    }

    fun generateCode(secret: String, counter: Long): String {
        // convert counter to long and insert into bytearray
        val payload: ByteArray = ByteBuffer.allocate(8).putLong(0, counter).array()
        val hash = generateHash(secret, payload)
        val truncatedHash = truncateHash(hash)
        // generate code by computing the hash as integer mod 1000000
        val code = ByteBuffer.wrap(truncatedHash).int % 10.0.pow(CODE_LENGTH).toInt()
        // pad code to correct length, could be too small
        return code.toString().padStart(CODE_LENGTH, '0')
    }

    private fun generateHash(secret: String, payload: ByteArray): ByteArray {
        val key = Base32().decode(secret)
        val mac = Mac.getInstance("HmacSHA512")
        mac.init(SecretKeySpec(key, "RAW"))
        return mac.doFinal(payload)
    }

    private fun truncateHash(hash: ByteArray): ByteArray {
        // last nibble of hash
        val offset = hash.last().and(0x0F).toInt()
        // get 4 bytes of the hash starting at the offset
        val truncatedHash = ByteArray(4)
        for (i in 0..3) {
            truncatedHash[i] = hash[offset + i]
        }
        // remove most significant bit
```

```
        truncatedHash[0] = truncatedHash[0].and(0x7F)
        return truncatedHash
    }
}
```

Custom implementation of TOTP generation, as well as other cryptographic security mechanisms, is not recommended, as it may carry potential threats to compromise the application's security system. Even if the TOTP generator in accordance with all standard requirements, maintaining it in accordance with security requirements will require additional effort, which may complicate the support of the product.

Severity and Impact Summary

Custom implementation of security mechanisms carries potential system security risks. In this case the implementation is minimalistic and conforms to the specifications. The team has chosen to implement their on version to assure cross platform compatibility in the application.

Recommendation

We suggest considering the possibility of using standard libraries to generate TOTP. For example, Google Authenticator.

References

- https://github.com/google/google-authenticator#google-authenticator-opensource
- https://datatracker.ietf.org/doc/html/rfc6238

UNNECESSARY PERMISSIONS HAVE BEEN ADDED TO THE OWNER AND GUARDIAN APPLICATIONS

Finding ID: FYEO-CENSO-AND-06
Severity: **Low**
Status: **Remediated**

Description

Several permissions from the "dangerous" category have been added to the Owner and Guardian applications. These permissions are not required for the applications to work. Specifically:

- android.permission.GET_ACCOUNTS
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.MANAGE_EXTERNAL_STORAGE

Proof of Issue

File name: owner/src/main/AndroidManifest.xml

Line number: 8 - 11

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE" />
```

File name: guardian/src/main/AndroidManifest.xml
Line number: 16 - 19

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE" />
```

If an app has more permissions than needed, it increases the surface area for potential security vulnerabilities. A malicious actor who exploits a vulnerability in the app can do more damage with unnecessary permissions.

Also, requesting extra permissions may have the following negative consequences:

- Making it more challenging to pass checks when publishing an application on Google Play
- Decreasing user trust in the application

Severity and Impact Summary

Increases the surface area for potential security vulnerabilities

Recommendation

We can assume that android.permission.WRITE_EXTERNAL_STORAGE and android.permission.READ_EXTERNAL_STORAGE were given to the application because the application saves the file with an encrypted symmetric key to the file system (shared/src/main/java/co/censo/shared/data/storage/CloudStorage.kt):

```
val fileDir = context.getExternalFilesDir(null)
```

But the getExternalFilesDir function does not need these permissions starting from API level 19 (Android 4). Moreover, starting from API level 30 (Android 11), these permissions have no effect at all, and the applications under study are built exclusively for Android 13.

Regarding android.permission.MANAGE_EXTERNAL_STORAGE: this permission gives read and write access to all files within shared storage. This is usually needed for File managers or other applications that need extended access to files on the file system. There was no need for this kind of privilege for the Owners and Guardian applications. In addition, it is important to remember that the use of this permission is limited by Google Play policy.

The android.permission.GET_ACCOUNTS permission was previously required to work with the Google API, but GoogleSignInApi, which is used in the application, does not require direct access to on-device capabilities.

Based on the above, we recommend that you consider removing these permissions from the manifest.

References

- https://developer.android.com/reference/android/Manifest.permission#WRITE_EXTERNAL_STORAGE
- https://developer.android.com/reference/android/Manifest.permission#READ_EXTERNAL_STORAGE
- https://support.google.com/googleplay/android-developer/answer/10467955
- https://developer.android.com/training/data-storage#scoped-storage

APPLICATIONS ALLOW TO CREATE BACKUP

Finding ID: FYEO-CENSO-AND-07
Severity: **Informational**
Status: **Remediated**

Description

Owner and Guardian applications allow to create backup. By default, Android always allows app backups unless otherwise specified (android:allowBackup="false"). At the same time, the applications save only SharedPreferences, but they are encrypted with a key from the Android Keystore, which cannot be transferred to another device. Therefore, the ability to create a backup may be unnecessary.

Proof of Issue

File name: owner/src/main/AndroidManifest.xml
Line number: 22-32

```
android:name=".VaultApplication"
android:dataExtractionRules="@xml/data_extraction_rules"
android:fullBackupContent="@xml/backup_rules"
android:windowSoftInputMode="adjustResize"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.Vault"
tools:targetApi="31"
android:networkSecurityConfig="@xml/network_security_config">
```

File name: guardian/src/main/AndroidManifest.xml
Line number: 14-20

```
android:name=".GuardianApplication"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.Guardian"
android:networkSecurityConfig="@xml/network_security_config">
```

As you can see, there is no explicit disabling of backup in the Application section, which means it is enabled by default.

Severity and Impact Summary

At this stage, backup does not pose any security risks. This item is for informational purposes only.

Recommendation

Since the backup function for this application is most likely unnecessary, we recommend disabling the ability to create backups

References

https://developer.android.com/guide/topics/data/autobackup#EnablingAutoBackup

APPLICATIONS TRUST ALL PRE-INSTALLED CAS

Finding ID: FYEO-CENSO-AND-08
Severity: **Informational**
Status: **Open**

Description

Owner and Guardian applications trust all system pre-installed certificates. System certificates are typically managed by the operating system, and relying on them can simplify the management of certificate authorities and reduce the potential for security vulnerabilities. But if any of these CAs were to issue a fraudulent certificate, the app would be at risk from an on-path attacker.

Proof of Issue

File name: owner/src/main/res/xml/network_security_config.xml
Line number: 10-14

```
<base-config cleartextTrafficPermitted="false">
    <trust-anchors>
        <certificates src="system" />
    </trust-anchors>
</base-config>
```

File name: guardian/src/main/res/xml/network_security_config.xml
Line number: 10-14

```
<base-config cleartextTrafficPermitted="false">
    <trust-anchors>
        <certificates src="system" />
    </trust-anchors>
</base-config>
```

Since the application transmits sensitive data over the network, it may be worth considering options to reduce the attack surface and add additional security measures.

These options include:

- Limit the set of trusted certificates
- Using the certification pinning mechanism

Severity and Impact Summary

This item is advisory in nature and is intended to inform about the possibility of reducing the attack surface on the communication channel between the application and the server.

Recommendation

We recommend considering the following options:

- Limit the set of trusted certificates
- Using the certification pinning mechanism

References

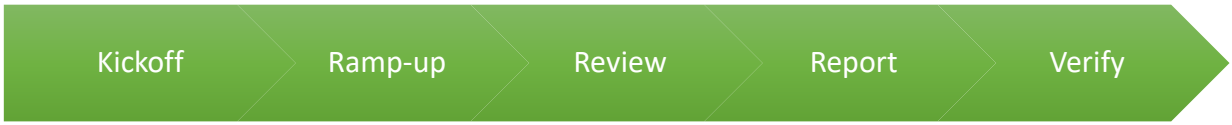
https://developer.android.com/privacy-and-security/security-config

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow



KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations