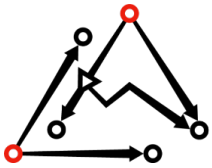
Security Assessment of Least Authority's Gridsync Application and Tahoe LAFS Android Application



Least Authority
PRIVACY MATTERS

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Include Security (IncludeSec)

IncludeSec brings together some of the best information security talent from around the world. The team is composed of security experts in every aspect of consumer and enterprise technology, from low-level hardware and operating systems to the latest cutting-edge web and mobile applications. More information about the company can be found at www.IncludeSecurity.com.

Assessment Objectives

The objective of this assessment was to identify and confirm potential security vulnerabilities within targets in-scope of the SOW. The team assigned a qualitative risk ranking to each finding. Recommendations were provided for remediation steps which Least Authority could implement to secure its applications and systems.

Scope and Methodology

Include Security performed a security assessment of Least Authority's Gridsync application and Tahoe LAFS Android application. The assessment team performed an eight(8) work day effort spanning from May 17th 2021 – May 31st 2021 using a Standard Grey Box assessment methodology which included a security review of all the components described in a manner consistent with the original Statement of Work (SOW).

Findings Overview

IncludeSec identified 6 categories of findings. There were 0 deemed to be "Critical-Risk," 0 deemed to be "High-Risk," 1 deemed to be "Medium-Risk," and 3 deemed to be "Low-Risk," which pose some tangible security risk. Additionally, 2 "Informational" level findings were identified that do not immediately pose a security risk.

IncludeSec encourages Least Authority to redefine the stated risk categorizations internally in a manner that incorporates internal knowledge regarding business model, customer risk, and mitigation environmental factors. As a third party consulting company our advice should highly influence, but not dictate how our clients manage risk.

Next Steps

IncludeSec advises Least Authority to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further security concerns from being introduced into future release cycles.

Being that these two applications assessed are open source, and that the IncludeSec team's efforts in this assessment were a standard assessment and not an exhaustive assessment this report should be seen as an excellent first step in security assurance for end users and customers. Ongoing security assessments, ongoing DevSecOp tooling, bug bounty programs and inviting the security community to file issues on the projects in Github may be other high value steps in future improving the applications.

ASSESMENT INTRODUCTION

The assessment team performed an audit of the **Gridsync desktop** and **Tahoe LAFS Android** applications. Gridsync is a free, open-source solution that allows users to install, configure, and use the Tahoe-LAFS decentralized file store to securely sync files between devices on all major operating systems. The Gridsync application was created to overcome some of Tahoe-LAFS usability issues, making it easier for non-technical users to get up and running. The assessment was time-boxed for workdays to focus on the mobile application and five work days on the desktop application. This work effort included source code review, dynamic analysis, and reporting for both. The assessment team focused on identifying security and privacy concerns in various areas including, but not limited to, cryptographic issues, data leakage, insecure file permissions, and denial of service issues.

At a high level, Gridsync helps automate the setup of the Tahoe-LAFS system on a user's computer. Once downloaded and configured to access a storage grid, a user will create a key encrypted with a password, which is referred to as the root capability. As long as the user has this key, they can access their files on other systems as well. Files added will be encrypted and split up among various nodes that make up the storage grid the user is connected to. Currently, the mobile application must be running on a device on the same local network as the user's laptop configured to access the storage grid. The Gridsync desktop application is used to pair the devices and allow the mobile application access to the storage grid.

Overall, the Gridsync desktop and Android applications proved notably resilient to many common types of attacks. The cryptographic design appeared to be correct, with no observed weaknesses related to encrypted file storage. The Tahoe-LAFS architecture itself does not utilize traditional user accounts, so vulnerabilities related to authorization boundaries were not applicable to Gridsync. The assessment team also looked for common client-side attack vectors, such as privilege escalation and insecure file permissions, but was unsuccessful in identifying any findings in these vulnerability classes. As a result of these positive security practices, most of the findings in this assessment were related to Denial-of-Service (DoS) attacks, as well as Low-risk configuration issues within the Android application.

RISK CATEGORIZATIONS

At the conclusion of the assessment, Include Security categorized findings into five levels of perceived security risk: Critical, High, Medium, Low, or Informational. **The risk categorizations below are guidelines that IncludeSec understands reflect best practices in the security industry and may differ from a client's internal perceived risk. Additionally, all risk is viewed as "location agnostic" as if the system in question was deployed on the Internet. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances.**

Critical-Risk findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

High-Risk findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

Medium-Risk findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

Low-Risk findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration, and outdated patches or policies.

Informational findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application. Any informational findings for which the assessment team perceived a direct security risk, were also reported in the spirit of full disclosure but were considered to be out of scope of the engagement.

The findings represented in this report are listed by a risk rated short name (e.g., C1, H2, M3, L4, and I5) and finding title. Each finding may include if applicable: Title, Description, Impact, Reproduction (evidence necessary to reproduce findings), Recommended Remediation, and References.

CRITICAL-RISK FINDINGS

No “Critical-Risk” findings were identified during the course of the engagement.

HIGH-RISK FINDINGS

No “High-Risk” findings were identified during the course of the engagement.

MEDIUM-RISK FINDINGS

M1: [Wormhole] Gridsync Vulnerable to Denial of Service Attacks

Description:

The assessment team found it was possible to prevent a user from creating an invite code by opening a large number of connections on the **Tahoe LAFS Wormhole** relay server using the **send** command. The relay server became temporarily unresponsive under high load, causing an error message to be displayed in the **Gridsync** client when attempting to generate an invite code.

Additionally, a malicious actor could flood the server with **receive** requests for all channels, preventing legitimate users from connecting to the wormhole.

Note that susceptibility to Denial of Service attacks is a known and documented limitation of Magic Wormhole (see References section for more information).

Impact:

A malicious actor could target the **Tahoe LAFS Wormhole** relay server, preventing legitimate users from generating invite codes.

Reproduction:

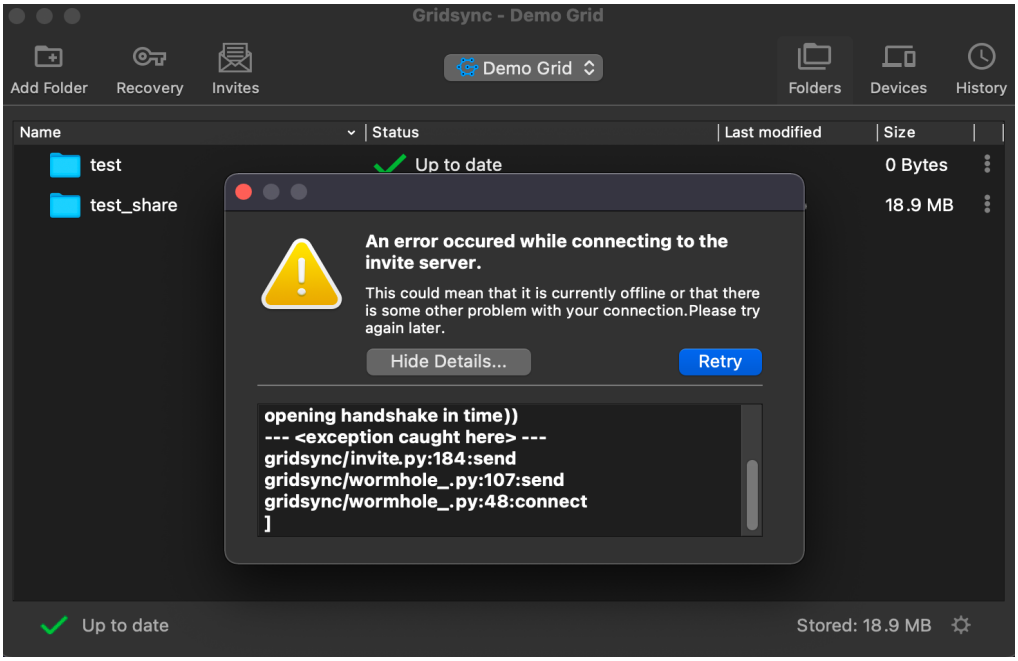
The following was tested from a Linux virtual machine as a proof of concept:

1. Install the magic wormhole tool via pip
2. Execute the command shown below these steps.
3. With the command running, attempt to generate a new invite code.
4. The error message below should be displayed.

Command:

```
for x in {1..1000}; do wormhole --appid tahoe-lafs.org/invite --relay-url ws://wormhole.tahoe-lafs.org:4000/v1 send test.txt &; done
```

Note that this proof of concept caused stability issues within the Linux virtual machine.



Recommended Remediation:

The assessment team recommends implementing rate limiting on Wormhole relay servers to prevent Denial of Service.

References:

Magic Wormhole: Known Vulnerabilities
GitHub Issue #107: server DOS attack

LOW-RISK FINDINGS

L1: [Android] Application Configured to Support Unencrypted HTTP

Description:

The assessment team found the **usesCleartextTraffic** flag set to **true** within the **AndroidManifest.xml** file. This attribute indicates whether the app intends to use cleartext network traffic, such as HTTP. The default value for apps which target API level 27 or lower is “true”. Apps which target API level 28 or higher default to “false”.

When this attribute is set to “false”, platform components such as HTTP, FTP, **DownloadManager**, and **MediaPlayer** refuse application requests to establish cleartext connections.

Impact:

When cleartext HTTP is used, an attacker who is suitably positioned on the network could intercept and modify this traffic to attack the user and obtain confidential information as it travels between the client and server. Note that no unencrypted HTTP traffic was observed during the assessment, so the assessment team considers this finding to be Low risk.

Reproduction:

The following snippet from **AndroidManifest.xml** shows that the application is configured to support cleartext HTTP, rather than requiring HTTPS:

```
android:usesCleartextTraffic="true"
```

Recommended Remediation:

The assessment team recommends converting all communications to encrypted channels, and modifying the application configuration to set **usesCleartextTraffic** to **false**.

References:

Android Developer: android:usesCleartextTraffic

L2: [Android] Application Allowed Android Backups

Description:

The Android application was configured to allow backups via the **android:allowBackup** attribute. This attribute specifies whether application data can be backed up and restored by a user who has enabled USB debugging.

Impact:

An attacker with physical access to the device would be able to back up the application's data via the Android Debug Bridge **adb**, which could allow the attacker to extract any confidential data contained within the application, such as personal files and data the user has shared with the storage grid.

Reproduction:

The following snippet from **AndroidManifest.xml** shows that backups were enabled for the application:

```
android:allowBackup="true"
```

Recommended Remediation:

The assessment team recommends setting **allowBackup** to **false** to prevent attackers from creating and analyzing backups.

References:

Android Developers: Data backup overview
OWASP: Testing Backups for Sensitive Data

L3: Client-Side Denial of Service via Malicious QR Code

Description:

The assessment team found a consistent way to crash and corrupt the application, forcing the user to delete and reinstall the application to continue using it.

Impact:

An attacker could exploit this behavior to corrupt the application, causing Denial of Service (DoS) conditions for Least Authority end users. This could result in increased support costs and loss of trust in the brand. Note that this attack would require an attacker to supply a malicious URL code to a user during the device registration phase, significantly reducing the attack window.

Reproduction:

The mobile application used a QR Code generated by the desktop application. After reading this QR code the mobile application generated a Shared Preferences file (**org_tahoe_lafs_shared_preferences.xml**) with the following format:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="SCANNER_URL">https://172.20.10.2:62973/2MyaLHVKU5ALNrkZFt9LmFs7mLhsW8TW</string>
    <string
name="SCANNER_TOKEN">LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJoekNDQVN5Z0F3SUJBZ0lVSXdEK09raldtOFhVc0dDRXBINjlxaW
VOVE1Jd0NnWUlLb1pJemowRUF3SXcKSGpFY01Cb0dBMVVFQXd3VE1UY3lMakl3TGpFd0xqSXVhVzUyWVd4cFpEQWdGdzB5TVRBMU1UZ3hPVEEwTVRaY
QpHQTh5TVRJeE1EUXlOREU1TURReE5sb3dIakVjTUJvR0ExVUVBd3dUTVRjeUxqSXdMakV3TGpJdWFXNTJZV3hwClpEQlpNQk1HQnlxR1NNNDlBZ0VH
Q0NxR1NNNDlBd0VIQTBJQUJOTENiZnA1aEtiRzM5TytxRkQ2L1F2M2cxUFMKUUFsaVdOVnAyOS8rNkdBMy9pZVM0T2phbVFvRm9DMGpVRTV2UjNhSUF
NSXBVNUFnYnpzOFBicHBmL2VqUmpCRQpNREVHQTFVZEVRUXFNQ2lDRXpFM01pNHlNQzR4TUM0eUxtbHVkbUZzYVdTQ0N6RTNNaTR5TUM0eE1DNHlod1
NzCkZBb0NNQThHQTFVZEV3UUlNQVlCQWY4Q0FRQXdDZ1lJS29aSXpqMEVBd0lEU1FBd1JnSWhBTjJsUDJGazI3eGwKdjVjRG5qUEYzclh4NnpaTWxid
ExkeVZVbU9sbWt1N2xBaUVBOWI3ZUhLSk9rNXBuUjNCWlA2ZVoydUtDWHlDTQozSk1uc2pIeVR1U0tCUU09Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0t
LS0K</string>
</map>
```

The assessment team found two different ways to crash the mobile application, and in both cases, the only way to continue using the application was to fully uninstall and reinstall it.

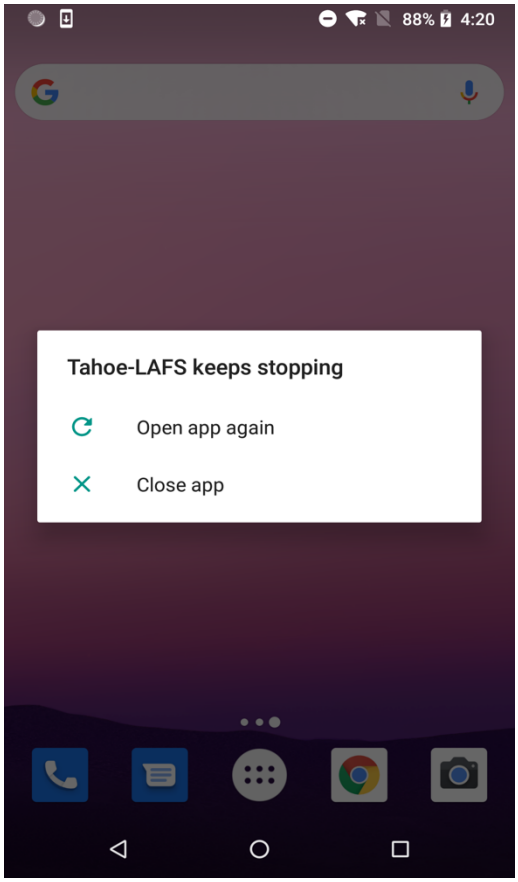
The mobile application required a valid URL, such as https://example.com or http://example.com, in the **SCANNER_URL** value. When it received an empty string, or one which did not start with **http** or **https**, it crashed. A missing or invalid **SCANNER_TOKEN** value can also cause a crash.

The assessment team crafted payloads to generate application errors. The following QR Code includes the https://includesecurity.com payload.

On reading this QR Code, the application crashed because “SCANNER_TOKEN” is empty.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="SCANNER_URL">https://includesecurity</string>
    <string name="SCANNER_TOKEN"></string>
</map>
```

The following screenshot shows the Android error message notifying the user that the application has crashed multiple times:



The following error was logged after reading the above QR Code:

```
D/ScannerFragment$addCallbacks: Scanned value is = https://includesecurity
I/zygote64: Do full code cache collection, code=109KB, data=93KB
I/zygote64: After code cache collection, code=90KB, data=62KB
D/MagicFolderFragment: Scanned url = https://includesecurity
D/ExtensionsKt: Full parse URL = https://includesecurity
D/AndroidRuntime: Shutting down VM
E/AndroidRuntime: FATAL EXCEPTION: main
    Process: org.tahoe.lafs, PID: 31557
    java.lang.RuntimeException: Unable to start activity
ComponentInfo{org.tahoe.lafs/org.tahoe.lafs.ui.home.HomeActivity}: java.lang.IllegalArgumentException: expected
non-empty set of trusted certificates
        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2778)
        at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2856)
        at android.app.ActivityThread.-wrap11(Unknown Source:0)
        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1589)
        at android.os.Handler.dispatchMessage(Handler.java:106)
        at android.os.Looper.loop(Looper.java:164)
        at android.app.ActivityThread.main(ActivityThread.java:6494)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:438)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:807)
     Caused by: java.lang.IllegalArgumentException: expected non-empty set of trusted certificates
        at org.tahoe.lafs.network.base.CustomTrustClient.trustManagerForCertificates(CustomTrustClient.kt:62)
        at org.tahoe.lafs.network.base.CustomTrustClient.<init>(CustomTrustClient.kt:22)
        at org.tahoe.lafs.network.base.BaseHttpClient$httpClient$2.invoke(BaseHttpClient.kt:39)
        at org.tahoe.lafs.network.base.BaseHttpClient$httpClient$2.invoke(BaseHttpClient.kt:32)
        at kotlin.SynchronizedLazyImpl.getValue(LazyJVM.kt:74)
        at org.tahoe.lafs.network.base.BaseHttpClient.getHttpClient(Unknown Source:2)
        at org.tahoe.lafs.network.base.BaseHttpClient.access$getHttpClient$p(BaseHttpClient.kt:32)
        at org.tahoe.lafs.network.base.BaseHttpClient$retrofitClient$2.invoke(BaseHttpClient.kt:57)
        at org.tahoe.lafs.network.base.BaseHttpClient$retrofitClient$2.invoke(BaseHttpClient.kt:32)
        at kotlin.SynchronizedLazyImpl.getValue(LazyJVM.kt:74)
        at org.tahoe.lafs.network.base.BaseHttpClient.getRetrofitClient(Unknown Source:2)
        at org.tahoe.lafs.di.module.ServiceModule.providesGridSyncApiService(ServiceModule.kt:29)
        at
org.tahoe.lafs.di.module.ServiceModule_ProvidesGridSyncApiServiceFactory.providesGridSyncApiService(ServiceModule_P
rovidesGridSyncApiServiceFactory.java:33)
        at
org.tahoe.lafs.DaggerTahoeApplication_HiltComponents_SingletonC.gridSyncApiService(DaggerTahoeApplication_HiltCompo
nents_SingletonC.java:182)
        at
org.tahoe.lafs.DaggerTahoeApplication_HiltComponents_SingletonC.gridSyncApiRepository(DaggerTahoeApplication_HiltCo
mponents_SingletonC.java:210)
        at
org.tahoe.lafs.DaggerTahoeApplication_HiltComponents_SingletonC.access$2000(DaggerTahoeApplication_HiltComponents_S
ingletonC.java:70)
        at
org.tahoe.lafs.DaggerTahoeApplication_HiltComponents_SingletonC$SwitchingProvider.get(DaggerTahoeApplication_HiltCo
mponents_SingletonC.java:592)
        at
org.tahoe.lafs.ui.viewmodel.GetFileStructureViewModel_AssistedFactory.create(GetFileStructureViewModel_AssistedFact
ory.java:27)
        at
org.tahoe.lafs.ui.viewmodel.GetFileStructureViewModel_AssistedFactory.create(GetFileStructureViewModel_AssistedFact
ory.java:12)
        at androidx.hilt.lifecycle.HiltViewModelFactory.create(HiltViewModelFactory.java:76)
        at androidx.lifecycle.AbstractSavedStateViewModelFactory.create(AbstractSavedStateViewModelFactory.java:69)
        at androidx.lifecycle.ViewModelProvider.get(ViewModelProvider.java:185)
        at androidx.lifecycle.ViewModelProvider.get(ViewModelProvider.java:150)
        at androidx.lifecycle.ViewModelLazy.getValue(ViewModelProvider.kt:54)
        at androidx.lifecycle.ViewModelLazy.getValue(ViewModelProvider.kt:41)
        at org.tahoe.lafs.ui.home.MagicFolderFragment.getGetFileStructureViewModel(Unknown Source:2)
        at org.tahoe.lafs.ui.home.MagicFolderFragment.initListeners(MagicFolderFragment.kt:105)
        at org.tahoe.lafs.ui.home.MagicFolderFragment.onViewCreated(MagicFolderFragment.kt:51)
        at androidx.fragment.app.FragmentStateManager.createView(FragmentStateManager.java:332)
        at androidx.fragment.app.FragmentManager.moveToState(FragmentManager.java:1199)
E/AndroidRuntime:     at androidx.fragment.app.FragmentManager.addAddedFragments(FragmentManager.java:2236)
        at androidx.fragment.app.FragmentManager.executeOpsTogether(FragmentManager.java:2009)
```

```
        at androidx.fragment.app.FragmentManager.removeRedundantOperationsAndExecute(FragmentManager.java:1965)
        at androidx.fragment.app.FragmentManager.execPendingActions(FragmentManager.java:1861)
        at androidx.fragment.app.FragmentManager.dispatchStateChange(FragmentManager.java:2641)
        at androidx.fragment.app.FragmentManager.dispatchActivityCreated(FragmentManager.java:2589)
        at androidx.fragment.app.Fragment.performActivityCreated(Fragment.java:2723)
        at androidx.fragment.app.FragmentStateManager.activityCreated(FragmentStateManager.java:346)
        at androidx.fragment.app.FragmentManager.moveToState(FragmentManager.java:1200)
        at androidx.fragment.app.FragmentManager.moveToState(FragmentManager.java:1368)
        at androidx.fragment.app.FragmentManager.moveFragmentToExpectedState(FragmentManager.java:1446)
        at androidx.fragment.app.FragmentManager.moveToState(FragmentManager.java:1509)
        at androidx.fragment.app.FragmentManager.dispatchStateChange(FragmentManager.java:2637)
        at androidx.fragment.app.FragmentManager.dispatchActivityCreated(FragmentManager.java:2589)
        at androidx.fragment.app.FragmentController.dispatchActivityCreated(FragmentController.java:247)
        at androidx.fragment.app.FragmentActivity.onStart(FragmentActivity.java:541)
        at androidx.appcompat.app.AppCompatActivity.onStart(AppCompatActivity.java:210)
        at android.app.Instrumentation.callActivityOnStart(Instrumentation.java:1334)
        at android.app.Activity.performStart(Activity.java:7019)
        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2741)
               ... 9 more
```

Recommended Remediation:

The assessment team recommends reviewing the QR Code parser and ensuring that values are validated and that exceptions are gracefully handled before attempting to use them.

To further protect against a corrupt Shared Preferences file, the application could detect and either delete or replace an invalid file when the “Link Device” process occurs.

References:

Android Developers Error Exception
Android: OWASP MSTG: Testing Exception Handling MSTG-CODE-6 and MSTG-CODE-7

INFORMATIONAL FINDINGS

I1: Gridsync Application Denial of Service via Import Recovery Key Functionality

Description:

The **Gridsync** application was found to be vulnerable to Denial of Service (DoS) attacks. When a user imports a recovery key file containing only numeric characters, or when they attempt to import a directory instead of a file using this functionality, the application crashed with an unhandled exception.

Impact:

A malicious actor could share an improperly formatted recovery key with a legitimate user to crash their application. This could interrupt various application actions such as uploading or downloading files to/from the storage grid.

Reproduction:

First Instance – DoS via Recovery Key Containing Only Numeric Characters

1. Create a file containing only numeric characters:
   echo 123 > test
2. Launch **Gridsync** (adding **—debug** when launching via Terminal displays exception information).
3. Select **Recovery Import Recovery Key**.
4. Browse to the **test** file and **Open** it for import.
5. The application crashes.

The **Gridsync** application crashed when a user attempted to import a file which contains only numeric characters, such as **123**. That value was treated as valid JSON so it did not trigger the **except** statement in the **_parse_content()** function of **recover.py**. The following snippet shows the stack trace after the crash:

```
Traceback (most recent call last):
  File "gridsync/gui/main_window.py", line 355, in import_recovery_key
  File "gridsync/gui/welcome.py", line 408, in on_restore_link_activated
  File "gridsync/recovery.py", line 224, in do_import
  File "gridsync/recovery.py", line 209, in _load_from_file
  File "gridsync/recovery.py", line 199, in _parse_content
TypeError: RecoveryKeyImporter.done[dict].emit(): argument 1 has unexpected type 'int'
Abort trap: 6
```

Second Instance – DoS via Importing Directory (MacOS)

1. On a Mac open Terminal and create a directory with the .app extension:
   mkdir test.app
2. Launch **Gridsync** (adding **—debug** when launching via Terminal displays exception information).
3. Select **Recovery Import Recovery Key**.
4. Browse to **test.app**, select it, and click **Open**.
5. The application crashes.

The assessment team also found that the **Gridsync** application crashed when a user attempted to import a directory instead of a file containing a recovery key. This is possible on macOS, where certain directories appear to be files, allowing them to be selected for import. For example applications are actually directories with the .app extension, containing application code. The following snippet shows the error when attempting to import a directory:

```
2021-05-28 11:24:18,218 DEBUG _load_from_file Loading /Users/<REDACTED>/Desktop/test.app...
Traceback (most recent call last):
  File "gridsync/recovery.py", line 204, in _load_from_file
IsADirectoryError: [Errno 21] Is a directory: '/Users/<REDACTED>/Desktop/test.app'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "gridsync/gui/main_window.py", line 355, in import_recovery_key
  File "gridsync/gui/welcome.py", line 408, in on_restore_link_activated
  File "gridsync/recovery.py", line 224, in do_import
  File "gridsync/recovery.py", line 207, in _load_from_file
  File "gridsync/msg.py", line 17, in error
TypeError: arguments did not match any overloaded call:
  QMessageBox(parent: QWidget = None): argument 1 has unexpected type 'RecoveryKeyImporter'
  QMessageBox(QMessageBox.Icon, str, str, buttons: Union[QMessageBox.StandardButtons, QMessageBox.StandardButton] =
QMessageBox.NoButton, parent: QWidget = None, flags: Union[Qt.WindowFlags, Qt.WindowType]
= Qt.Dialog|Qt.MSWindowsFixedSizeDialogHint): argument 1 has unexpected type 'RecoveryKeyImporter'
Abort trap: 6
```

Recommended Remediation:

The assessment team recommends adding more data validity checks and exception handling to prevent the application from crashing when attempting to import an invalid recovery key.

The application also attempted to load very large files, which can lead to resource exhaustion and denial of service. Refusing to load unreasonably large recovery key files would avoid this issue.

References:

OWASP: Error Handling

I2: Gridsync Application Crash via Voucher Code Containing non-ASCII Characters

Description:

The assessment team was able to crash the **Gridsync** application by entering a voucher code containing non-ASCII characters.

Impact:

An attacker could leverage this by sending or posting an voucher code with non-ASCII characters and tricking a targeted user into entering it into the application. This could interrupt application actions such as uploading and downloading files to and from the storage grid.

Reproduction:

1. Launch the **Gridsync** application from the command line with the **—debug** flag.
2. Join the **HRO Cloud Grid**, which requires entry of a voucher once joined.
3. Tap **Use voucher code**.
4. Enter a voucher containing non-ASCII characters.
5. Observe that the application crashes.

The crash occurred in the **is_valid()** function of the **voucher.py** module, on line 39 when the voucher was decoded:

```
def is_valid(code: str, checksum_length: int = 2) -> bool:
    code = dehyphenate(code)
    try:
        decoded = base64.b32decode(code)
    except binascii.Error:
        return False
    b = decoded[:-checksum_length]
    checksum = decoded[-checksum_length:]
    if checksum == get_checksum(b):
        return True
    return False
```

The **except** statement is executed and returns false when an invalid base-32 ASCII character is found. However processing a non-ASCII character crashed the application:

```
Traceback (most recent call last):
  File "base64.py", line 37, in _bytes_from_decode_data
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-15: ordinal not in range(128)

During handling of the above exception, another exception occurred. An example voucher that will cause a crash and
display the following output from the --debug option is *AAAA-AAAA-AAAA-AAAŁ*

Traceback (most recent call last):
  File "gridsync/gui/voucher.py", line 52, in on_return_pressed
  File "gridsync/voucher.py", line 39, in is_valid
  File "base64.py", line 203, in b32decode
  File "base64.py", line 39, in _bytes_from_decode_data
ValueError: string argument should contain only ASCII characters
Abort trap: 6
```

Recommended Remediation:

The assessment team recommends adding additional exception handling to detect and gracefully handle invalid voucher codes.

References:

OWASP: Error Handling

APPENDICES

OWASP Mobile Top 10

This matrix is provided to understand where findings in the mobile app might be aligned to in this vulnerability category enumeration, and also to provide a high-level plan of attack. Security concerns past these ten were explored, but this gives an understanding of a base plan of attack.

Category	Description	Assessment Observations
M1. Improper Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.	No findings were identified related to improper platform usage.
M2. Insecure Data Storage	This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.	The assessment team found that the application is using the Shared Prefs file, in order to store a base64 encoded with a X509v3 certificate used to establish communication with the desktop app.
M3. Insecure Communications	This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.	The assessment team found use of the “usesCleartextTraffic” flag within the AndroidManifest
M4. Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: • Failing to identify the user at all when that should be required • Failure to maintain the user's identity when it is required • Weaknesses in session management	The assessment team found that the application crashed repeatedly after manipulating QR code information (authentication process with the desktop app).
M5. Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for findings where cryptography was attempted, but it wasn't done correctly.	No findings related to “insufficient” cryptography were identified.
M6. Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication findings (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.	No findings related to “insecure” authorization were identified.

M7. Client Code Quality	This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.	No findings related to client code quality were identified.
M8. Code Tampering	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.	No findings related to code tampering were identified.
M9. Reverse Engineering	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.	The application implements Certificate Pinning, but does not implement any additional protection against static manipulation or decompilation. Note that this may be intentional as the application source code is freely available.
M10. Extraneous Functionality	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.	No findings related to extraneous functionality were identified.