App iShredder
MASA L1
2025-02-26

Summary of the application

Target of Evaluation	**App Version: 7.1.3**
Model and/or type reference	**iShredder**
Other identification of the product	**com.projectstar.ishredder.android.standard**
Features	**Tools & Utilities**
Manufacturer	**Protectstar Inc.**
Test Method Requested	**Security evaluation based on limited set of evaluation procedures from OWASP Mobile Application Security Verification Standard established by ADA.**
Validation Type	**Level 1 - Verified Self**
Validated By	**Jose María Santos López – Cybersecurity Engineer**
Platform	**Android**
Date of Issue	**2025-02-26**

DEKRA Testing and Certification guarantees the reliability of the data presented in this report, which is the result of the measurements and the tests performed to the item under test on the date and under the conditions stated on the report and, it is based on the knowledge and technical facilities available at DEKRA Testing and Certification at the time of performance of the test.

DEKRA Testing and Certification is liable to the client for the maintenance of the confidentiality of all information related to the item under test and the results of the test. The results presented in this Test Report apply only to the particular item under test established in this document.

Nomenclature

Below you can find the verification type considered for MASA L1 as well as the description of each of them.

Verification Type

- **Self Declare:** developer provides Yes/No response to verify they meet the requirement
- **Documentation:** developer provides artifacts / evidence to verify they meet the requirement
- **NMI:** lab provides output from static analysis for developers to respond to
- **Scan Verified:** lab performs static analysis against fully automatable requirements to assign a Pass / Fail for each requirement
    - For scan failures the developer is expected to resolve the issue or provide a written justification explaining how they meet the requirement including supporting evidence such as scan output, screenshots or supporting documentation.

Summary Table

Requirement	Description	Validation Type	Status	Dev Action
MSTG-STORAGE-1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys	Self Declare	Pass	N
MSTG-STORAGE-2	No sensitive data should be stored outside of the app container or system credential storage facilities.	Self Declare	Pass	N
MSTG-STORAGE-3	No sensitive data is written to application logs.	NMI	Pass	N
MSTG-STORAGE-5	The keyboard cache is disabled on text inputs that process sensitive	NMI	Pass	N

	data.			
MSTG-STORAGE-7	No sensitive data, such as passwords or pins, is exposed through the user interface.	NMI	Pass	N
MSTG-STORAGE-12	The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.	Self Declare	Pass	N
MSTG-CRYPTO-1	The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	Scan Verified	Pass	N
MSTG-CRYPTO-2	The app uses proven implementations of cryptographic primitives.	Scan Verified	Pass	N
MSTG-CRYPTO-3	The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.	Scan Verified	Pass	N
MSTG-CRYPTO-4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	Scan Verified	Pass	N

MSTG-CRYPTO-5	The app does not re-use the same cryptographic key for multiple purposes.	Self Declare	Pass	N
MSTG-CRYPTO-6	All random values are generated using a sufficiently secure random number generator.	Scan Verified	Pass	N
MSTG-AUTH-1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	Self Declare	Pass	N
MSTG-AUTH-2	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user`s credentials.	Self Declare	Pass	N
MSTG-AUTH-3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.	Self Declare	Pass	N
MSTG-AUTH-4	The remote endpoint terminates the existing session when the user logs	Self Declare	Pass	N

	out.			
MSTG-AUTH-5	A password policy exists and is enforced at the remote endpoint.	Self Declare	Pass	N
MSTG-AUTH-6	The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.	Self Declare	Pass	N
MSTG-AUTH-7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.	Self Declare	Pass	N
MSTG-NETWORK-1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	Scan Verified	Pass	N
MSTG-NETWORK-2	The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.	Scan Verified	Pass	N
MSTG-NETWORK-3	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established.	Scan Verified	Pass	N

MSTG-PLATFORM-1	The app only requests the minimum set of permissions necessary.	NMI	Pass	N
MSTG-PLATFORM-2	All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.	Scan Verified	Pass	N
MSTG-PLATFORM-3	The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.	NMI	Pass	N
MSTG-PLATFORM-4	The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.	Self Declare	Pass	N
MSTG-CODE-1	The app is signed and provisioned with a valid certificate, of which the private key is properly protected.	Scan Verified	Pass	N
MSTG-CODE-2	The app has been built in release mode, with settings appropriate for a release build (e.g.	Scan Verified	Pass	N

	non-debuggable).			
MSTG-CODE-3	Debugging symbols have been removed from native binaries.	Scan Verified	No run	N
MSTG-CODE-4	Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.	Scan Verified	Pass	N
MSTG-CODE-5	All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.	Self Declare	Pass	N
MSTG-CODE-9	Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.	Scan Verified	Pass	N

Self-Declare

MSTG-STORAGE-1

System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys

Does your app (or library/SDK) use cryptographic keys to encrypt all data that may be considered sensitive, such as PII?
A. Yes
B. **No**

No sensitive data is not stored in credential storage facilities. They are stored in regular storaged facilities (Shared preferences) that have restricted access on unrooted devices. They get deleted when the app gets deleted.
Does your app (or library/SDK) use Android Keystore API to store user credentials?
A. Yes
B. **No**

Does your app (or library/SDK) use Android Keystore API to store cryptographic keys?
A. Yes
B. **No**

MSTG-STORAGE-2

No sensitive data should be stored outside of the app container or system credential storage facilities.

Does your application or library/SDK exclusively store sensitive data within the app container or use system credential storage facilities?
A. **Yes**
B. No

By default, Android SharedPreferences are stored within your app's private internal storage, specifically in the app container (typically at /data/data/<your.package.name>/shared_prefs). This means they aren't stored in a publicly accessible location outside your app's sandbox, ensuring that other apps cannot access them unless the device is rooted or the app explicitly allows external access.

MSTG-STORAGE-12

The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.

Do you educate users about the types of personally identifiable information (PII) it processes, as well as security best practices they should follow when using the app?

A. **Yes**
B. No

Does your app (or library/SDK) provide a clear and easily accessible link to your privacy policy within the app?
A. **Yes**
B. No

MSTG-CRYPTO-5

The app does not re-use the same cryptographic key for multiple purposes.

Do you have mechanisms in place to audit and verify that each cryptographic key is used exclusively for its designated purpose within the app (or library/SDK) ?
A. Yes
B. **No**

Because we do not use cryptographic key
Do you verify that each cryptographic key in your app (or library/SDK) is assigned a single, specific purpose to avoid key reuse?
A. Yes
B. **No**

MSTG-AUTH-1

If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.

Does your app (or library/SDK) implement authentication at the remote endpoint?
A. **Yes**
B. No
C. N/A (app does not have authentication)

Is used for MY.PROTECTSTAR (https://my.protectstar.com) user license management only

MSTG-AUTH-2

If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user`s credentials.

Does your app`s (or library/SDK) remote endpoint use unique and unpredictable session identifiers to authenticate client requests without transmitting the user's credentials?

A. **Yes**
B. No
C. N/A (app does not have authentication)

MSTG-AUTH-3

If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.

If stateless token-based authentication is utilized by your application, does your app server (or library/SDK) provide a token protected using a secure algorithm such as HMAC-SHA256?

A. **Yes**
B. No
C. N/A (app does not have authentication or app does not use stateless token-based authentication)

Yes, we do HMAC-SHA256

MSTG-AUTH-4

The remote endpoint terminates the existing session when the user logs out.

Does your app (or library/SDK) terminate the existing session at the remote endpoint when the user logs out?

A. **Yes**
B. No
C. N/A (app does not have authentication)

MSTG-AUTH-5

A password policy exists and is enforced at the remote endpoint.

Does your app (or library/SDK) prevent users from setting passwords they have already used before at the remote endpoint?

A. **Yes**
B. No
C. N/A (app does not have authentication)

Does your app (or library/SDK) enforce a password policy (e.g., minimum length, complexity) on the server/backend side?

A. **Yes**
B. No
C. N/A (app does not have authentication or password is never sent to server (e.g. federated auth or zero-knowledge password proof))

MSTG-AUTH-6

The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.

Does your app (or library/SDK) implement a mechanism at the remote endpoint to protect against the submission of credentials an excessive number of times?

A. **Yes**
B. No
C. N/A (app does not have authentication or password is never sent to server (e.g. federated auth or zero-knowledge password proof))

MSTG-AUTH-7

Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.

Does your app (or library/SDK) implement a mechanism at the remote endpoint (server-side) to automatically invalidate user sessions after a predefined period of inactivity?

A. **Yes**
B. No

C. N/A (app does not have authentication)

Do access tokens used for authentication and authorization within your app (or library/SDK) have a set expiration time, after which the server will reject them as invalid?

A. **Yes**
B. No
C. N/A (app does not have authentication)

MSTG-PLATFORM-4

The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.

Does your app (or library/SDK) expose sensitive functionality through inter-process communication (IPC) mechanisms?

A. Yes
B. **No**

Does your app (or library/SDK) have security measures like access controls, data validation, and encryption in place to protect sensitive functionality exposed through IPC mechanisms in your app or library/SDK?

A. **Yes**
B. No
C. N/A (app does not have IPC mechanisms)

MSTG-CODE-5

All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.

Does your app (or library/SDK) use third party libraries?

A. **Yes**
B. No

1. Provide a list of 3P libraries to labs

implementation 'net.dongliu:apk-parser:2.6.10' implementation 'com.zsoltsafrany:needle:1.0.0' implementation 'com.github.kenglxn.QRGen:android:3.0.1' implementation 'com.sothree.slidinguppanel:library:3.4.0' implementation 'com.ogaclejapan.smarttablayout:library:2.0.0@aar' implementation 'com.github.sevar83:indeterminate-checkbox:1.0.5@aar' implementation 'com.futuremind.recyclerfastscroll:fastscroll:0.2.5' implementation 'com.github.bumptech.glide:glide:4.14.2' implementation 'com.android.volley:volley:1.2.1' implementation 'com.squareup.okhttp3:okhttp:4.11.0'

NMI

MSTG-STORAGE-3

No sensitive data is written to application logs.

We would like to clarify the following points:

1. **No Sensitive User Data Logged**
   Our application does not write sensitive data (e.g., user credentials, PII, or license keys) to the logs. Any text input by the user (such as license keys) is strictly handled internally and never recorded in device logs.
2. **Referenced Classes Are Third-Party / Google Library Code**
   The classes listed (CoordinatorLayout.java, PreferenceGroup.java, various Glide engine files, etc.) are from standard Android libraries or the Glide caching library (https://github.com/bumptech/glide). We have **not** modified these libraries to log sensitive data. Glide primarily manages image loading and caching, and we are unaware of any logging of private user inputs in this process.
3. **Compliance with MSTG-STORAGE-3**
   We confirm adherence to MSTG-STORAGE-3, ensuring that no confidential or user-sensitive information is exposed through logs. Our logging is restricted to operational or debugging statements that do not contain personal or license data.

If you require further information or have any follow-up queries, please let us know.

MSTG-STORAGE-5

The keyboard cache is disabled on text inputs that process sensitive data.

The file in question, myps_activity_activate_slider.xml, contains an EditText field where users can enter a license key.

1. **Nature of the License Key**

- The entered license key is **not** personally identifiable information (PII) like a password or credit card number.
- It serves only to activate the product and does not represent a user credential (i.e., it does not grant access to private user data or accounts).

2. **Keyboard Caching / Autofill**
   - Although we do not consider license keys as personal or confidential data, we understand the best practice of **disabling keyboard suggestions** or autofill for any input that might be interpreted as sensitive.
   - To address any concerns, we can add attributes such as android:inputType="textNoSuggestions" or android:importantForAutofill="no" to ensure that the OS keyboard does not cache this entry.
3. **Compliance with MSTG-STORAGE-5**
   - We confirm that no other fields in our application store or expose user-sensitive data via the keyboard cache.
   - For completeness and in line with security best practices, we will update the license-entry field to explicitly **disable keyboard suggestions** so there is zero risk of the license code being stored in a dictionary or autocomplete cache.

MSTG-STORAGE-7

No sensitive data, such as passwords or pins, is exposed through the user interface.

The referenced layout file (myps_activity_activate_slider.xml) contains an EditText field for entering a license key.

1. **License Keys vs. Sensitive User Data**
   - The license key is **not** a password, PIN, or personally identifiable information (PII).
   - It does not grant access to personal user data or any user account and is therefore not considered confidential data in the same sense as passwords or PINs.
2. **No Exposure of Confidential Information**
   - We do not display any user credentials, session tokens, or other data that could compromise user security.
   - The text field is used solely for product activation, and its contents do not reveal sensitive details about the user.
3. **Compliance with MSTG-STORAGE-7**
   - Since the license key is not a form of sensitive user data (e.g., password or PIN), it does not violate any requirement about exposing sensitive fields in the UI.
   - Other UI elements in the application likewise do not present sensitive user information.

Should you require any additional clarification or have any further questions, please let us know.

MSTG-PLATFORM-1

The app only requests the minimum set of permissions necessary.

We would like to clarify:

1. **Optional Permissions**
   The additional permissions requested in our AndroidManifest (e.g., access to SMS/MMS or call logs) are **optional** and only required if the user explicitly chooses to erase those specific data types.
2. **User Choice & Granular Access**
   Our application does not automatically use these permissions. We prompt the user for consent **only** when they opt to shred SMS, MMS, or call logs. If the user does not wish to erase these data types, the permissions are neither granted nor used.
3. **Adherence to Principle of Least Privilege**
   By requesting permissions on an as-needed basis, we ensure that the app does not unnecessarily access any data or functionality. Therefore, we are in compliance with the "minimum set of permissions" guideline.

Should you have any further questions or need additional details, please let us know.

MSTG-PLATFORM-3

The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.

We would like to clarify the following:

1. **Standard Schemes Only**
   The <data android:scheme="sms"/>, <data android:scheme="smsto"/>, <data android:scheme="mms"/>, and <data android:scheme="mmsto"/> entries in our AndroidManifest are **standard Android schemes** for handling SMS/MMS URIs. They are not custom or proprietary schemes (like myapp://...) that could invoke privileged app functionality.
2. **No Sensitive Functionality Exposed**
   These intent filters allow the app to interact with SMS and MMS links in a standard, user-initiated context. We do **not** export any privileged or sensitive operations. The app does not silently send messages or perform actions without proper user interaction or permissions.
3. **Proper Protections**
   Since these are standard schemes and do not provide direct access to internal APIs or confidential features, no additional protection (e.g., authentication tokens) is required. The declared intent filters simply register our app for basic messaging-related URIs, as is common practice on Android.

We hope this clarifies the matter. Should you require any further details, please feel free to let us know.

Scan Verified

In this section you can find all the results based on the automatic evaluation, for the test cases in PASS it can be seen as informative, for the results marked as Fail you need to fix the issues or provide some feedback or justification in the respective area.

FAIL

PASS

MSTG-CRYPTO-1

The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.

We would like to clarify the following points:

1. **The flagged "Hardcoded Keys" are actually overwrite patterns** , not cryptographic keys.
   - These patterns are part of our secure data erasure logic, where we overwrite storage areas with specific byte sequences (or randomly generated ones) to securely delete data.
   - They are **not** used for encrypting or decrypting information.
2. **We do use random data for certain passes, generated via SecureRandom** , and **no fixed seed** is employed.
   - Whenever random data is required, it is indeed produced by calling SecureRandom (or ThreadLocalRandom), ensuring high entropy.
   - There is no reliance on any static or embedded secret for cryptographic purposes.
3. **These fixed patterns are deliberate** and serve only to ensure multiple overwrite passes using known byte sequences.
   - Such patterns are commonly used in data-sanitization methods and do not compromise security.
   - The presence of these patterns in the code may have been mistakenly flagged as "keys."

We hope this explanation clarifies why the reported "Hardcoded Keys" do not represent an actual security risk. If you need further technical details or have additional questions, please let us know.
After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-2

The app uses proven implementations of cryptographic primitives.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-3

The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-4

The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CRYPTO-6

All random values are generated using a sufficiently secure random number generator.

After further analysis the application seems to comply successfully with the requirement.

MSTG-NETWORK-1

Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.

After further analysis the application seems to comply successfully with the requirement.

MSTG-NETWORK-2

The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.

After further analysis the application seems to comply successfully with the requirement.

MSTG-NETWORK-3

The app verifies the X.509 certificate of the remote endpoint when the secure channel is established.

After further analysis the application seems to comply successfully with the requirement.

MSTG-PLATFORM-2

All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

We would like to clarify:

1. **Google-Provided Library Code**
   The classes you referenced (java_source/i1/i.java, java_source/y3/l.java, java_source/y3/o.java) appear to be part of a Google-provided (open-source) library. We have not altered or customized the underlying logic in these files.
2. **Input Validation in Our App Code**
   Although the flagged code comes from a standard library, our own application code ensures that **any user-supplied or external data** is validated and sanitized **before** it is passed along to these library functions. Consequently, we do not feed untrusted or potentially malicious input into sensitive routines.
3. **Adherence to MSTG-PLATFORM-2**
   We confirm that we follow a defensive approach, validating all incoming data—be it from UI inputs, intents, URLs, or network sources. This approach reduces the risk of injection attacks or other security breaches. Our integration of Google's library does not bypass any of these checks.
4. **Regular Updates and Security Monitoring**
   We monitor and update our dependencies (including Google libraries) to ensure we always have the latest security patches and best practices in place.

We hope this clarifies how we address input validation and sanitization. If you have any further questions, please let us know.
After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-1

The app is signed and provisioned with a valid certificate, of which the private key is properly protected.

After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-2

The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).

After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-4

Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.

We would like to clarify the following:

1. **Referenced Library (Glide):**
   The code snippet in question is part of the open-source Glide library, which we use for efficient image loading and caching. Glide helps us handle large image lists while ensuring smooth scrolling and optimized performance.
2. **StrictMode Check (No Debug Backdoor):**
   The specific code that you have flagged is related to a StrictMode policy check. This check ensures that no network requests occur on the main (UI) thread, thereby preventing performance issues (e.g., ANRs) rather than serving as a “debugging backdoor.” There are no verbose logs or hidden test routines in production related to this code.
3. **No Debugging or Verbose Logging in Production:**
   We confirm that we do **not** include any additional debug or developer assistance code, hidden settings, or backdoors in our production build. The Glide library code mentioned does not log sensitive information and only helps ensure stable performance and smooth user experience.

In conclusion, we comply with your requirement that “Debugging code and developer assistance code have been removed, and the app does not log verbose errors or debugging messages.” If you need further details, please let us know.
After further analysis the application seems to comply successfully with the requirement.

MSTG-CODE-9

Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.

After further analysis the application seems to comply successfully with the requirement.

Recommendations