

Using r2 to Uncover Mobile App Weaknesses

Carlos Holguera @ r2con
24 October 2025





Carlos
Holguera

OWASP

Mobile Application Security (MAS)
Flagship Project Co-leader

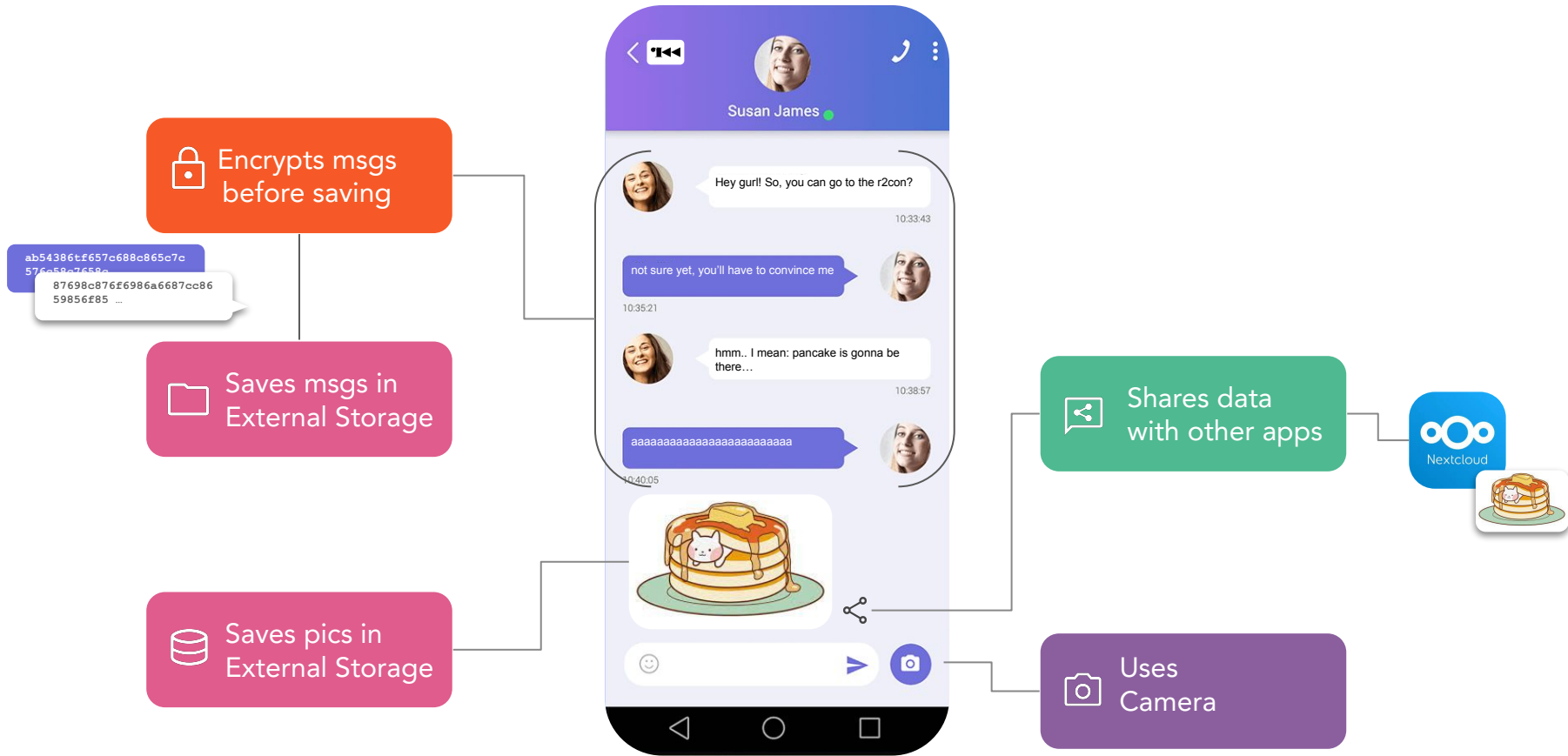
[carlos-holguera](#)



NowSecure

Principal Mobile Security
Research Engineer

[@grepharder](#)



High-level
Less Specific

MASVS

Example

Insecure Crypto is bad

MASWE

Example

The app uses weak
Encryption Algorithms

MASTG

Example

Test for `Cipher` objects
using DES

Low-level
More specific

MASVS-CRYPTO-1

The app employs current strong cryptography and uses it according to industry best practices.



L1

L2

Weakness: Cryptographically Weak Pseudo-Random Number Generator (PRNG)

Test 1: Insecure Random API Usage

Demo 1: Common Uses of Insecure Random APIs

Sample Code

Test Script

SAST Rule *

Output

Test 2: Non-random Sources Usage

Tests

The screenshot shows the OWASP Mobile Application Security (MAS) website. The header includes the OWASP logo, navigation links (Home, MASRG (Beta), MASRG, MASVS, MAS Checklist, MAS Crackmes, News, Talks, Contribute, Donate, Connect with Us), a search bar, and the OWASP logo with version information (OWASP/masrg-0202, 0.1.0, 0.1.0, 0.1.0).

The left sidebar contains a navigation menu with categories: MASRG, MASVS-STORAGE, MASVS-CRYPTO, MASVS-PRIVACY, Demos (v2 Beta), Techniques, Tools, and Apps.

The main content area displays the MASRG-0202: References to APIs and Permissions for Accessing External Storage page. The page title is "MASRG-0202: References to APIs and Permissions for Accessing External Storage". Below the title is a "Content in BETA" banner.

The page content includes:

- Overview:** This test uses static analysis to look for uses of APIs allowing an app to write to locations that are shared with other apps (e.g., [Testing Local Storage for Sensitive Data](#)) such as the external storage APIs or the MediaStore API as well as the relevant Android manifest storage-related permissions. This static test is great for identifying all code locations where the app is writing data to shared storage. However, it does not provide the actual data being written, and in some cases, the actual path in the device storage where the data is being written. Therefore, it is recommended to combine this test with others that take a dynamic approach, as this will provide a more complete view of the data being written to shared storage.
- Steps:**
 - Reverse engineer the app ([Decompiling Java Code](#)).
 - Run a static analysis ([Static Analysis on Android](#)) tool on the reverse engineered app targeting calls to any external storage APIs and Android manifest storage permissions.The static analysis tool should be able to identify all possible APIs and permissions used to write to shared storage, such as `getExternalStoragePublicDirectory`, `getExternalStorageDirectory`, `getExternalFilesDir`, `MediaStore`, `WRITE_EXTERNAL_STORAGE`, and `MANAGE_EXTERNAL_STORAGE`. See the [Android documentation](#) for more information on these APIs and permissions.
- Observation:** The output should contain a list of APIs and storage-related permissions used to write to shared storage and their code locations.
- Evaluation:**

The test case fails if:

 - the app has the proper permissions declared in the Android manifest (e.g. `WRITE_EXTERNAL_STORAGE`, `MANAGE_EXTERNAL_STORAGE`, etc.)
 - and the data being written to shared storage is sensitive and not encrypted.To determine the latter, you may need to carefully review the reversed code ([Reviewing Decompiled Java Code](#)) and/or combine this test with others that take a dynamic approach, as this will provide a more complete view of the data being written to shared storage.
- References:**
 - [Manage all files on a storage device](#)
 - [Access media files from shared storage](#)
- Demos:**
 - [MASRG-DEMO-0003: App Writing to External Storage without Scoped Storage Restrictions](#)
 - [MASRG-DEMO-0005: App Writing to External Storage via the MediaStore API](#)
 - [MASRG-DEMO-0004: App Writing to External Storage with Scoped Storage Restrictions](#)

What are we testing and why?

Steps to test

Output of test

How to evaluate?

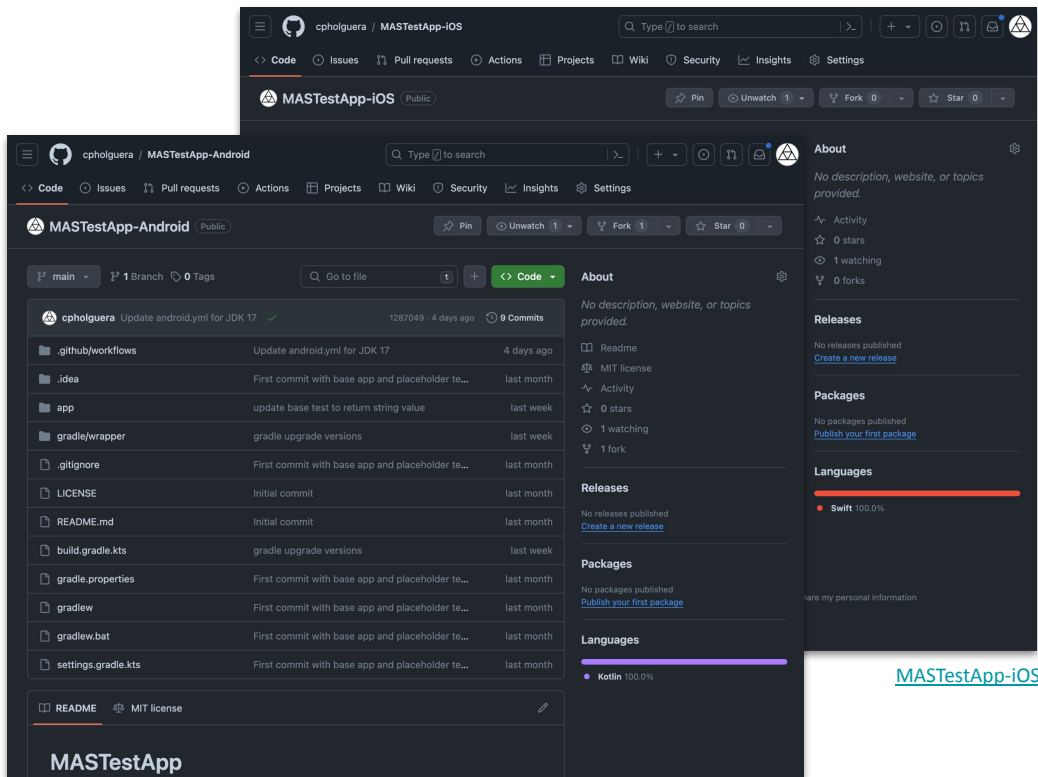
Links to demos



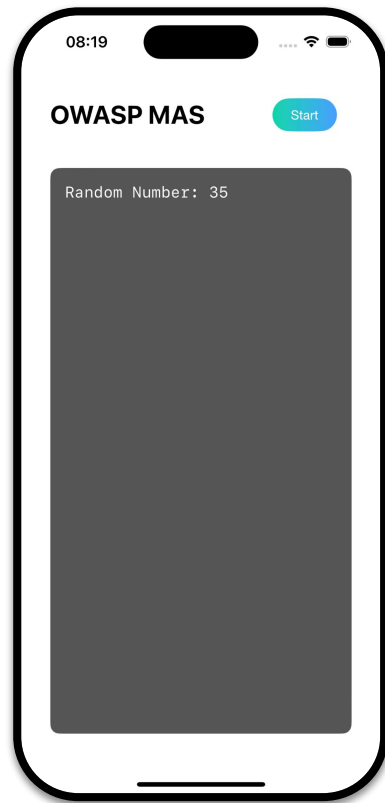
MAS Test Apps



MASTestApps

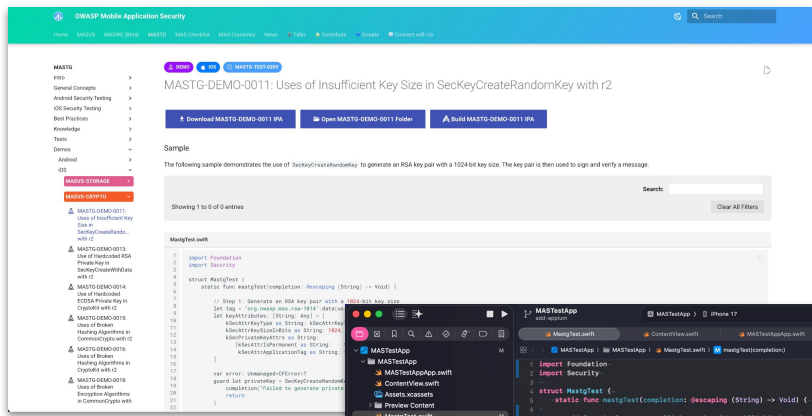


[MASTestApp-iOS](#)

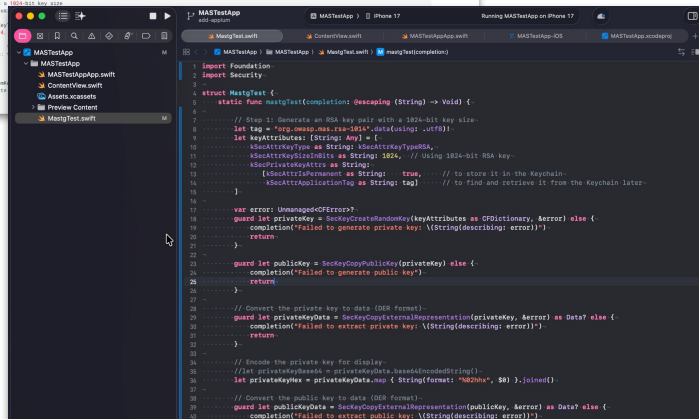


MASTestApp-Android

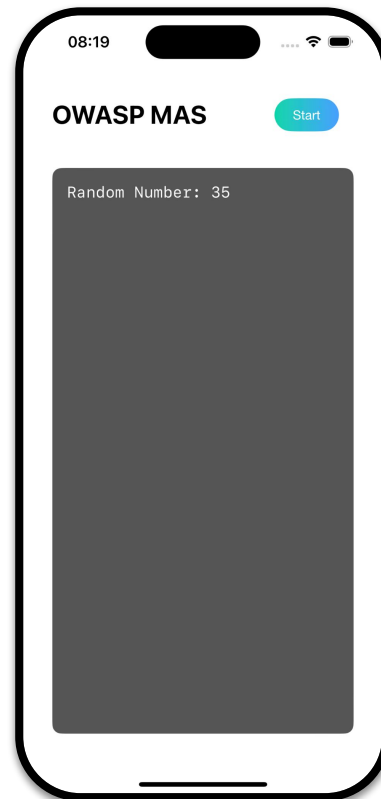
MASTestApps



Copy



Paste



Run



Demos:
MASWE vulnerable
code samples as
downloadable apps

MASVS-CRYPTO



MASTG-DEMO-0011

Uses of Insufficient Key Size in
SecKeyCreateRandomKey with
r2

[TEST](#) | [DEMO](#)

14:03

OWASP MAS

Start

Original: This is a
sample text

Private Key (Hex):

```
3082025c02010002818100c1c2888
a2ea9327a8974e8767e25bea4a211
37c2922c947a445621dfb7f1c3f-
ba744afeb3717cb1c8d6a2d83f137
166adf22b83ce-
b3f47b0bf6a31ae6273d-
ba35f344c4f8b-
d5a5e94088429eae22f-
f73b248ea4f6ad-
b6d2d22f937ed75aa39d92306102b
38b76f1b2e3868b386814880f300f
9b3d72d6b-
d8a8c3b8424b780cd020301000102
81804f38e6f2f8d91d5694d544270
eb50819f059f1de903d-
c207abe13bc0694964c48443c559f
b10b36f5d32c8a166030c1b6e5897
d3f0ab-
d2f8258b99fc9f2c97e4a233810d8
32eeb8cb1949a999ed996e173f-
b588f6acb1f2b94a976290ebd-
cd5433b2ad46b006a8ae4d515e1d-
f2f2732e29a68b702e-
f3ae39bf70a1671b2e26d1024100e
```

MASVS-CRYPTO



MASTG-DEMO-0015

Uses of Broken Hashing
Algorithms in CommonCrypto
with r2

[TEST](#) | [DEMO](#)





 @OWASP_MAS

 @grepharder

 carlos-holguera

mas.owasp.org/contact





References

<https://mas.owasp.org/MASTG/>
<https://mas.owasp.org/MASWE/>

<https://mas.owasp.org/contributing/>
<https://mas.owasp.org/contact/>
<https://mas.owasp.org/news/>

<https://github.com/OWASP/owasp-mastg/>
<https://github.com/cpholguera/MASTestApp-Android/>
<https://github.com/cpholguera/MASTestApp-iOS/>

<https://github.com/radareorg/radare2-mcp>