Writing MAS Weaknesses & Tests

Reference examples can be found in:

- https://github.com/OWASP/owasp-mastg/blob/master/weaknesses/
- https://github.com/OWASP/owasp-mastg/tree/master/tests-beta
- https://github.com/OWASP/owasp-mastg/tree/master/demos

- The OWASP MAS team -

Weaknesses

Weaknesses are OS-agnostic which has the advantage of not having to duplicate the weakness description.

Weaknesses must be located under weaknesses/ and the corresponding MASVS category. Their file names are the weakness' IDs.

Example:

```
% ls -1 -F weaknesses/MASVS-CRYPTO/

MASWE-0009.md
MASWE-0010.md
MASWE-0011.md
MASWE-0012.md
...
```

Markdown: Metadata

title

- Clarity: Is the title easily understood? Does it clearly state what the weakness is?
- Specificity: Does the title precisely describe what is being tested? Does it avoid vague or generic terminology?
- Consistency: Does the title follow a similar structure and style as other titles in the dataset?

Tip: you may take inspiration from related CWEs. The title of a weakness should be similar to a CWE title.

platform

The mobile platform. Can be many of: ios, android. Prefer generic weaknesses as much as possible (applicable to both Android and iOS). Use mobile platform-specific weaknesses sparingly.

Example:

```
platform: ["android", "ios"]
```

profiles

A weakness can map to one or more MAS profiles.

Example:

```
profiles: ["L1", "L2"]
```

mappings

A weakness must map to at least one MASVS v2 control:
https://github.com/OWASP/owasp-masvs/tree/v2.1.0/controls

If applicable, it must include the mapping to the MASVS v1:
https://github.com/OWASP/owasp-masvs/tree/v1.5.0/Document

Optional mappings can be added for

- CWE: https://cwe.mitre.org/index.html
- Android Risks: https://developer.android.com/privacy-and-security/risks
- Android Core App Quality (Privacy & security): https://developer.android.com/docs/quality-guidelines/core-app-quality#sc

Example:

```
mappings:
  masvs-v1: [MSTG-CRYPTO-6]
  masvs-v2: [MASVS-CRYPTO-1]
  cwe: [338, 337]
  android-risks:
   - https://developer.android.com/privacy-and-security/risks/weak-p
     rng
  android-core-app-quality: [SC-C1]
```

Observed Examples

Add examples from CVEs, blog posts, and other sources such as hackerone.

```
observed_examples:
- https://nvd.nist.gov/vuln/detail/CVE-2013-6386
- https://nvd.nist.gov/vuln/detail/CVE-2013-6386
- https://nvd.nist.gov/vuln/detail/CVE-2006-3419
- https://nvd.nist.gov/vuln/detail/CVE-2008-4102
```

Markdown: Body

Overview

- Should clearly and concisely describe the weakness without specifying the impact (which has its separate section)
- Be as generic as possible: avoid specific Android, iOS API names, terms, etc., similar to CWE (unless it's a platform-specific weakness).

Example:

```
## Overview

A [pseudorandom number generator
(PRNG)](https://en.wikipedia.org/wiki/Pseudorandom_number_generator)
algorithm generates sequences based on a seed that may be
predictable. Common implementations are not cryptographically secure.
For example, they typically use a linear congruential formula,
allowing an attacker to predict future outputs, given enough observed
outputs. Therefore, it is not suitable for security-critical
applications or protecting sensitive data.
```

Impact

See examples of Impact here:

- https://developer.android.com/topic/security/risks/sticky-broadcast#impact
- https://developer.android.com/topic/security/risks/unsafe-trustmanager#impact

It should contain a bullet point list with short explanations.

Example:

```
## Impact

- **Bypass Protection Mechanism**: Using a non-cryptographically
secure PRNG in a security context, such as authentication, poses
significant risks. An attacker could potentially guess the generated
numbers and gain access to privileged data or functionality.
Predicting or regenerating random numbers can lead to encryption
breaches, compromise sensitive user information, or enable user
impersonation.
```

Idea: use a collection of common impact patterns as CWE does so that people can refer to it and reuse the terms. This will be useful for filtering weaknesses as well.

Technical Impact	Description	Mobile App Example
Modify Memory	Unauthorized alteration of system memory	An attacker modifies a game's memory to change the score.
Read Memory	Unauthorized reading of system memory	An attacker reads memory to find passwords or cryptographic keys.

Modify Files or Directories	Unauthorized alteration of public or system files or directories	A malicious app modifies system files to gain elevated privileges.
Read Files or Directories	Unauthorized reading of public or system files or directories	A malicious app reads private user data, like photos or documents.
Modify Application Data	Unauthorized alteration of application private data	An attacker alters the data in a banking app to change account balances.
Read Application Data	Unauthorized reading of application private data	An attacker reads sensitive information from a messaging app's data.
DoS	Excessive use of resources	An app excessively uses device storage, memory, leaving no resources for other apps or causes the app to intermittently crash or freeze leaving it unresponsive.
Execute Unauthorized Code or Commands	Execution of unauthorized commands	An app allows the execution of arbitrary code when processing malicious content.
Gain Privileges or Assume Identity	Unauthorized escalation of privileges or identity theft	A malicious app gains root access and impersonates the user to send premium SMS messages.

Bypass Protection Mechanism	Circumventing security mechanisms	A malicious app exploits a vulnerability to bypass permissions and access protected APIs.

Modes of Introduction

This section provides information about how and when the weakness may be introduced, including a short description. Multiple introduction points can be provided if they exist.

They typically indicate areas to be tested.

Example:

```
## Modes of Introduction

- **System Logs**: The application may log sensitive data to the system log, which can be accessed by other applications on the device (in old OS versions or compromised devices or if they hold the appropriate permissions).

- **App Logs**: The application may log sensitive data to a file in the application's data directory, which can be accessed by any application on the device if the device is rooted.
```

Mitigations

The mitigation section should include generic recommendations that explain common concepts on how the weakness can be eliminated or at least mitigated.

Note: Do not include platform-specific mitigations here. Use the "mitigations:" key in the tests metadata to link to an existing mitigation in the mitigations/ directory. This section will automatically list any platform-specific mitigations that are linked in tests.

In these separate mitigation files: Where available, use links to the Android and iOS developer documentation for more detailed instructions, including "*good code*" examples.

Example:

```
## Mitigations

For security relevant contexts, use cryptographically secure random numbers.
```

In general, it is strongly recommended not to use any random function in a deterministic way, even if it's a secure one, especially those involving hardcoded seed values (which are vulnerable to exposure by decompilation).

Refer to the [RFC 1750 - Randomness Recommendations for Security](https://www.ietf.org/rfc/rfc1750.txt) and the [OWASP Cryptographic Storage Cheat Sheet - Secure Random Number Generation](https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html#secure-random-number-generation) for more information and recommendations on random number generation.

Tests

Tests are platform-specific and must be located under tests-beta/android/ or tests-beta/ios/, within the corresponding MASVS category. Their file names are the test IDs.

Tests have an overview and contain Steps which produce outputs called observations: after following the Steps you come up with an Observation which you will Evaluate.

Example:

```
% ls -1 -F tests-beta/android/MASVS-CRYPTO/
MASTG-TEST-0204.md
MASTG-TEST-0205.md
```

Markdown: Metadata

title

Test titles should be concise and clearly state the purpose of the test.

In some cases, the test name and the weakness may have the same title, but typically the tests will test different aspects of a weakness (as defined in "Modes of introduction"), so the titles need to reflect that.

Avoid including Android or iOS in the titles unless absolutely necessary, as in "Insecure use of the Android Protected Confirmation API".

Please ensure that the titles follow a similar structure and style to other titles in the dataset.

Conventions

- **Static**: “References to…” (semgrep/r2)
- **Dynamic**: “Runtime Use …” (frida)

We currently consider some exceptions to this convention for “dynamic tests” where it would feel forced to start it with "Runtime ...". For example, when the test is based on using tools like adb (e.g. to perform a local backup), performing file system snapshots, etc.

Examples include:

- MASTG-TEST-0207
- MASTG-TEST-0216
- MASTG-TEST-0263

platform

The mobile platform. Can be one of: ios, android.

id

The ID of the test

weakness

The MASWE weakness ID the test is referencing.

type

A test can have one or more types.

Supported types: static, dynamic, network, manual.

Example:

```
type: [static]
```

mitigations

Use mitigations in the test metadata to add platform specific mitigations or best practices. Our automation will create a “Mitigations” section automatically.

You can create new best practice files under best-practices/ and use them as mitigations.

Example:

```
mitigations: [MASTG-BEST-0001]
```

This will create a link to https://mas.owasp.org/MASTG/best-practices/MASTG-BEST-0001/

prerequisites

Link to any prerequisites needed for executing or evaluating the test.

Existing prerequisites are in the prerequisites/ folder. Create new ones if required.

For example, for evaluating the results of the test you may need to have identified the sensitive data relevant to your app.

Example:

```
prerequisites:
- identify-sensitive-data
- identify-security-relevant-contexts
```

Markdown: Body

Overview

The overview of a test is platform-specific and acts as an extension of the weakness overview for the particular area covered by the test. It may mention specific APIs and platform features.

Example:

```
## Overview

Android apps sometimes use insecure pseudorandom number generators
(PRNGs) such as `java.util.Random`, which is essentially a linear
congruential generator. This type of PRNG generates a predictable
sequence of numbers for any given seed value, making the sequence
reproducible and insecure for cryptographic use. In particular,
`java.util.Random` and `Math.random()` ([the
latter](https://franklinta.com/2014/08/31/predicting-the-next-math-ra
ndom-in-java/) simply calling `nextDouble()` on a static
`java.util.Random` instance) produce identical number sequences when
initialized with the same seed across all Java implementations.
```

Steps

A test must include one or more steps that can be static, dynamic, both. Previously in the MASTG we were forcing one or the other and sometimes mention that you can do both. But usually the steps will be mixed.

For example, to "check app notifications"

1. method trace for related APIs (dynamic)
2. use the app (manual)
3. RE to understand use or use backtraces & more hooking (static)
4. taint analysis using known/self-defined values and letting them get to the notification (dynamic)
5. grep the method trace or integrate "grep" in a frida script (static/dynamic)

Example:

```
## Steps
```

```
1. Run a [static
analysis](/MASTG/techniques/android/MASTG-TECH-0014.md) tool on the
app and look for insecure random APIs.
```

Always link to existing techniques (or create new ones if they don't exist yet) to prevent duplication or repetition of content. In this example: techniques/android/MASTG-TECH-0014.md

Observation

This is the output you get after executing all steps. It serves as evidence.

Examples: method trace for specific APIs, network traffic trace filtered in some way, hooking events containing sensitive data (indicating which APIs handle that data).

It should start with "The output should contain …".

Example:

```
## Observation

The output should contain a **list of locations where insecure random
APIs are used**.
```

Evaluation

Using the observation as input, the evaluation tells you how to evaluate it and must explicitly describe what makes the test fail.

It should start with "The test case fails if …"

Example:

```
## Evaluation

The test case fails if you can find random numbers generated using
those APIs that are used in security-relevant contexts.
```

Demos

A collection of demos (demonstrative examples) of the test that include working code samples and test scripts to ensure reproducibility and reliability.

Demos live in `demos/android/` or `demos/ios/` under the corresponding MASVS category folder. Each demo has its own folder named using its ID and contains:

- Markdown file: `MASTG-DEMO-xxx.md`
- Code samples (e.g. .kt, .swift, .xml, .plist)
- Testing code (e.g. sh, py)
- Output files (e.g. txt, sarif)

Language: The samples are written in **Kotlin** or **Swift**, depending on the platform. In some cases, the samples will also include configuration files such as AndroidManifest.xml or Info.plist.

Decompiled Code: If the sample can be decompiled, the decompiled code is also provided in the demo (e.g as a Java file on Android: `MastgTest_reversed.java`). This is useful for understanding the code in the context of the application.

The **demos MUST WORK**. See Code Samples.

Demos are required to be **fully self-contained** and should **not rely on external resources or dependencies**. This ensures that the demos can be run independently and that the results are reproducible. They must be proven to work on the provided sample applications and must be tested thoroughly before being included in the MASTG.

Don't create demos for outdated OS versions that aren't supported by the MASTG. The MASTestApp is meant to always be up to date and aligned with the versions supported by the MASTG, so as to avoid additional maintenance of the MASTestApp. However, you can include demos showcasing the "good case" in the metadata using `kind: pass` in certain cases where it can be helpful or educational. This is permitted as long as the demos work with the current version of the MASTestApp.

Please specify the mobile platform version, IDE and version, device.

Example:

```
% ls -1 -F demos/android/MASVS-CRYPTO/MASTG-DEMO-0007
```

```
MASTG-DEMO-0007.md
MastgTest.kt
MastgTest_reversed.java
output.txt
run.sh*
```

Markdown: Metadata

title

The title should concisely express what the demo is about.

Example:

```
title: Common Uses of Insecure Random APIs
```

platform

The mobile platform. Can be one of: ios, android.

tools

Tools used in the demo.

They must be referenced using their IDs from https://mas.owasp.org/MASTG/tools/

Example:

```
tools: [MASTG-TOOL-0031]
```

code

The language in which the samples are written.

Example:

```
code: [java]
```

Markdown: Body

Sample

Shortly describe the sample and specify the exact sample files used using this notation:

Single file:

```
{{ MastgTest.kt }}
```

Multi-file rendered in tabs:

```
{{ MastgTest.kt # MastgTest_reversed.java }}
```

Example:

```
### Sample

The snippet below shows sample code that sends sensitive data over
the network using the `HttpURLConnection` class. The data is sent to
`https://httpbin.org/post` which is a dummy endpoint that returns the
data it receives.

{{ MastgTest.kt # MastgTest_reversed.java }}
```

Steps

A concise writeup following all steps from the test and including the relevant placeholders for testing code (e.g. SAST rules, run.sh files).

Example:

```
### Steps

Let's run our semgrep rule against the sample code.

{{ ../../../../rules/mastg-android-non-random-use.yaml }}

{{ run.sh }}
```

Observation

A concise description of the observation for this specific demo including the relevant placeholders for output files (e.g. output.txt).

Example:

```
### Observation

The rule has identified some instances in the code file where a
non-random source is used. The specified line numbers can be located
in the original code for further investigation and remediation.

{{ output.txt }}
```

Evaluation

A concise explanation about how you applied the test “Evaluation” section to this specific demo. For example, if lines are present explain each line.

Example:

```
### Evaluation

Review each of the reported instances.

- Line 12 seems to be used to generate random numbers for security
purposes, in this case for generating authentication tokens.
- Line 17 is part of the function `get_random`. Review any calls to
this function to ensure that the random number is not used in a
security-relevant context.
- Line 27 is part of the password generation function which is a
security-critical operation.

Note that line 37 did not trigger the rule because the random number
is generated using `SecureRandom` which is a secure random number
generator.
```

Code Samples

Code samples for demos **must be created using one of our test apps** to ensure consistency across demos and facilitate the review process:

- https://github.com/cpholguera/MASTestApp-Android
- https://github.com/cpholguera/MASTestApp-iOS

Simply clone the repository and follow the instructions to run the apps on your local machine. You **must use these apps to validate the demos** before submitting them to the MASTG.

File

Must be a modified version of the original files in the apps' repos:

- Android: `app/src/main/java/org/owasp/mastestapp/MastgTest.kt`
- iOS: `MASTestApp/MastgTest.swift`

When working on a new demo you **must include the whole file** with the original name in the demo folder.

Summary

Must contain a summary as a comment.

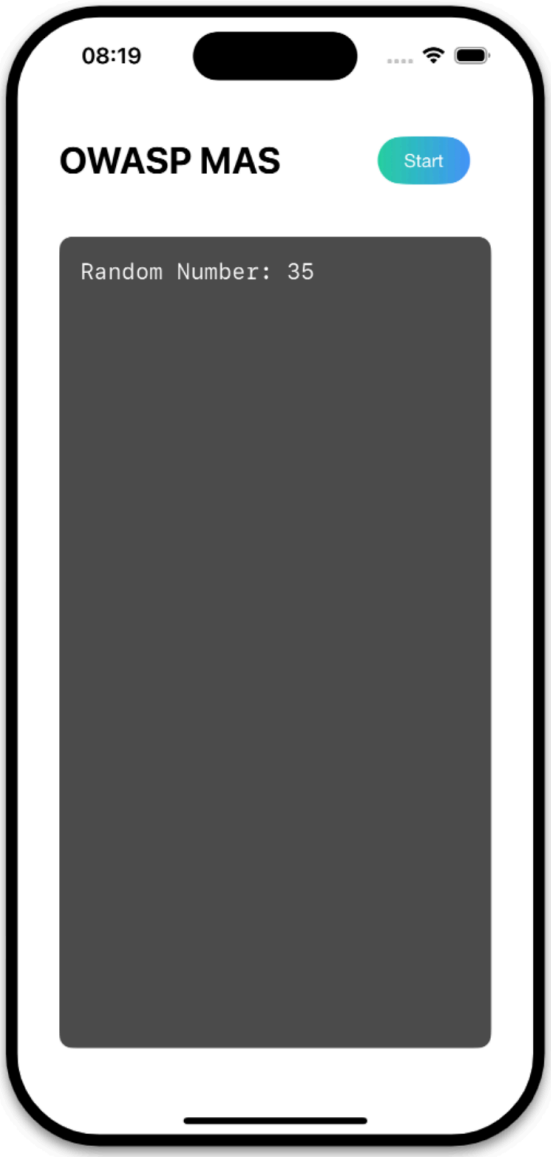
Example:

```
// SUMMARY: This sample demonstrates different common ways of
insecurely generating random numbers in Java.
```

Logic

The file must include code that demonstrates the addressed weakness.
The provided default `MastgTest.kt` and `MastgTest.swift` contain some basic logic that will return a string to the UI. If possible try to return some meaningful string.

For example, if you create a random number you can return it; or if you write files to the external storage you can return a list of file paths so that the user of the app can read them. You can also use that string to display some meaningful errors.



Returning an random number on an iOS demo

Returning an error string on an Android demo

Fail/Pass

Must contain comments indicating fail/pass and the test alias. This way we're able to validate that the output is correct (e.g. the code contains 3 failures of `MASTG-TEST-0204`). We can easily parse and count the comments and we can do the same in the output.

Each FAIL/PASS comment must include the test Id and an explanation of why it files/passes.

Example:

```
// FAIL: [MASTG-TEST-0204] The app insecurely uses random numbers for
generating authentication tokens.
return r.nextDouble();

// PASS: [MASTG-TEST-0204] The app uses a secure random number
generator.
return number.nextInt(21);
```

run.sh

Every test that can be automated must contain a run.sh file

Static

Static tests must work using the **reverse-engineered code**. The app's repos contain scripts or indications to obtain the reversed files.

Example: semgrep

```
NO_COLOR=true semgrep -c
../../../../rules/mastg-android-insecure-random-use.yaml
./MastgTest_reversed.java --text -o output.txt
```

Dynamic

Example: frida-trace

```
frida-trace -U -f com.google.android.youtube --runtime=v8 -j
'*!*certificate*/isu' > output.txt
```

Example: frida

```
frida -U sg.vp.owasp_mobile.omtg_android -l hook_edittext.js >
output.txt
```

Networking

Example: mitmproxy

```
mitmdump -s mitm_sensitive_logger.py
```

Rules & Scripts

SAST rules (and potentially also DAST scripts) live in https://github.com/OWASP/owasp-mastg/tree/master/rules. They can be referenced and reused by the demos.

Semgrep rules

https://semgrep.dev/docs/getting-started/quickstart/
https://semgrep.dev/learn
https://academy.semgrep.dev/courses/secure-guardrails

Tip: use https://semgrep.dev/playground/new for experimentation.

They must be named `mastg-<name of the test>.yaml` and follow valid syntax according to https://semgrep.dev/docs/writing-rules/rule-syntax/

- **id**: same as the file name
- **severity**:
  - WARNING
  - ERROR
- **languages**: e.g. java
- **metadata**: must include summary
  - summary: Short description of the rule.
  - original_source: you may use rules from sources on the internet be sure to check that the license allows this and always link to the original source here. Modify the rule if needed and the license allows for it.
- **message**: must start with the MASVS control ID and concisely explain what the rule is reporting.
- **patterns**: see https://semgrep.dev/docs/writing-rules/pattern-syntax/

Do not include authors in the semgrep rules. If it was copied from some other place, **include the link to the original source**. Since many people will potentially contribute to the rule as part of the MASTG work, the authors will be calculated using git.

Example:

```
rules:
  - id: mastg-android-insecure-random-use
    severity: WARNING
    languages:
      - java
```

```
    metadata:
      summary: This rule looks for common patterns including classes
and methods.
    message: "[MASVS-CRYPTO-1] The application makes use of an
insecure random number generator."

    pattern-either:
        - patterns:
            - pattern-inside: $M(...){ ... }
            - pattern-either:
                - pattern: Math.random(...)
                - pattern: (java.util.Random $X).$Y(...)
```

Frida Scripting

Frida: TBD
Frida-trace: TBD

Demos must use Frida 17 and above, see
https://mas.owasp.org/MASTG/tools/generic/MASTG-TOOL-0031/#frida-17

mitmproxy Scripting

TBD

radare2 Scripting

TBD

We're currently using .r2 scripts which will be updated to python r2pipe scripts at some point:

https://book.rada.re/scripting/r2pipe.html

Best Practices

https://mas.owasp.org/MASTG/best-practices/

https://github.com/OWASP/owasp-mastg/tree/master/best-practices

Best practices must be linked to MASTG tests using the `best-practices:` key.

They must have official references which may include the MASTG as long as it contains further references to e.g. Google/Apple docs or other official sources.

Best Practices should contain:

- what's the recommendation
- why is that good
- any caveats or considerations (e.g. "it's good to have it but remember it can be bypassed this way", etc.)