Ghera
A Repository of Android App Vulnerability Benchmarks

Joydeep Mitra    Venkatesh-Prasad Ranganath
Mobile Apps are Ubiquitous

Mobile apps are an integral part of our everyday life

- Banking and Commerce
- Social networking (or is it social life?)
- Two-factor verification / OTP
Mobile App Security is Relevant

Vulnerabilities in mobile apps can affect users’ safety and security

- Information leak
- Denial of Service
- Privileged Access
Ways to Secure Mobile Ecosystems

- Keep malicious apps out of the ecosystems
- Educate users
- Improve developer awareness about vulnerabilities
- Secure/Harden mobile apps
Our Focus

• Keep malicious apps out of the ecosystems
• Educate users
• **Improve developer awareness about vulnerabilities**
• **Secure/Harden mobile apps**
  • A methodology to build secure apps
Our Initial Goals

1. Learn and understand known vulnerabilities in Android apps
2. Establish a baseline to evaluate our methodology
Our First Steps

• Study existing literature about vulnerabilities in Android apps
  • Including secure coding prescriptions, e.g., SEI, Symantec

• Study existing vulnerability benchmarks and repositories
  • DroidBench: A micro-benchmark suite to assess the stability of taint-analysis tools for Android
  • AndroZoo: A huge collection of Android apps from several apps stores
  • PlayDrone: Another collection of Android apps from Google Play
Our Observations

• **Too Specific**
  - Information leaks that can be detected via taint analysis
  - Information leaks stemming from ICC

• **Too Vague**
  - Which version of Android is affected?
  - Is the app vulnerable? If so, how? Is it due to a known vulnerability? Which one?
  - Are all known vulnerabilities captured?
Ghera

SQLite-SQLInjection-Lean/

Summary

Apps that do not use selectionArgs to construct select, update, and delete queries are vulnerable to SQL injection attacks.

Versions of Android affected

Tested on Android 4.4 - 7.1

Description of the vulnerability and the corresponding exploit

Android allows apps to save data in a local database called SQLite. SQL queries are used to interact with the SQLite database.

Issue: If the app does not parameterize the queries it uses to interact with the database then it is vulnerable to SQL injection attacks.

Example: We demonstrate this benchmark with the help of two apps Benign and Malicious. Benign/MyContentProvider uses a non-parameterized check if a particular username/password is in the database or not. Benign/MyContentProvider is also exported. Malicious uses a specially crafted query to retrieve all the username/password pairs in the table.

Steps to build the sample apps and to exploit the vulnerability

1. List targets:

   $ android list targets
Ghera

Each benchmark

- captures a vulnerability and a corresponding exploit
  - An vulnerable/benign app (source + APK)
  - An exploitive/malicious app (source + APK)
- is tested across Android versions 4.4 - 7.1
- includes information about the vulnerability
- includes instructions to build and use the benchmark
- includes references to the source of vulnerabilities
Average size of apps is ~275 Java + XML + Gradle LoC (including tests)

Average size of apks is ~1.5MB

Benchmarks are based on vulnerabilities
- reported in literatures and
- discovered by studying of Android APIs
Looking back

• There was very little to no guidance for creating benchmarks

• We did a retrospection
  • identify the characteristics we considered in Ghera
  • identify the reasons for these characteristics
Desirable Characteristics of Vulnerability Repository

1. **Open**: The repository is open to the community both for consumption and contribution

2. **Comprehensive**: The repository has benchmarks that account for (almost) all known vulnerabilities of the target environment
Desirable Characteristics of Vulnerability Benchmark

1. **Ready-To-Use:** The benchmark is composed of artifacts that can be used as is to reproduce the vulnerability.

2. **Easy-To-Use:** The benchmark is easy to set up and reproduce the vulnerability.

3. **Well Documented:** The benchmark is accompanied by relevant documentation.

4. **Version Specific:** The benchmark is associated only with the versions of the framework in which the contained vulnerability can be reproduced.

5. **Tool & Technique agnostic:** The benchmark is agnostic to tools and techniques and how they detect vulnerabilities.
5. **Authentic:** If the benchmark claims to contain vulnerability X, then it truly contains vulnerability X.

6. **Dual:** The benchmark contains both the vulnerability and a corresponding exploit (dual).

7. **Feature Specific:** If the benchmark uses only features F of a framework to create vulnerability X, then the benchmark does not contain other features of the framework that can be used to create X.

8. **Contextual:** The benchmark capturing vulnerability X in a context C is distinct from benchmarks capturing X in other contexts.
So, how does Ghera measure up?

- Ghera may not be comprehensive
- Ghera has only lean benchmarks
- No hard evidence to confirm Ghera is feature-specific and contextual
Current & Future Work

1. Add new benchmarks to capture new vulnerabilities
2. Extend Ghera with fat benchmarks corresponding to existing lean benchmarks
3. Use Ghera to evaluate existing vulnerability detection tools and malware detection tools
4. Use Ghera to evaluate our solution/methodology
Takeaways

Call for Actions

Ghera is a growing repository of Android app vulnerability benchmarks

- Contribute to Ghera
- Use Ghera in your Android security related efforts

With Ghera, we identified few desirable characteristics of vulnerability benchmarks and repositories

- Critique/Extend/Fix them
- Consider them when creating benchmarks and repositories

https://bitbucket.org/secure-it-i/android-app-vulnerability-benchmarks