An Effective Android Code Coverage Tool

Aleksandr Pilgun
SnT, University of Luxembourg
aleksandr.pilgun@uni.lu

Olga Gadyatskaya
SnT, University of Luxembourg
olga.gadyatskaya@uni.lu

Stanislav Dashevskyi
SnT, University of Luxembourg
stanislav.dashevskyi@uni.lu

Yury Zhauniarovich
Qatar Computing Research Institute, HBKU
yzhauniarovich@hbku.edu.qa

Artsiom Kushniarou
SnT, University of Luxembourg
artsiom.kushniarou@uni.lu

https://github.com/pilgun/acvtool

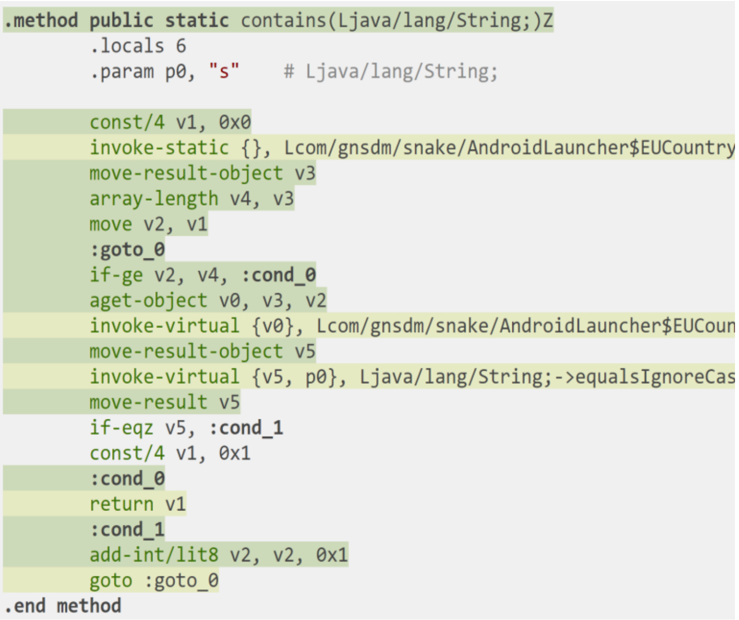
Context

The deluge of Android apps from third-party developers calls for sophisticated security testing and analysis techniques to inspect suspicious apps without accessing their source code. Code coverage is an important metric used in these techniques to evaluate their effectiveness, and even as a fitness function to help achieving better results in evolutionary and fuzzy approaches.

Existing tools [1-4] for measuring code coverage over the bytecode of Android apps have the following limitations:

— coarse granularity
— low instrumentation success rate
— limited empirical evaluation

Smali Report



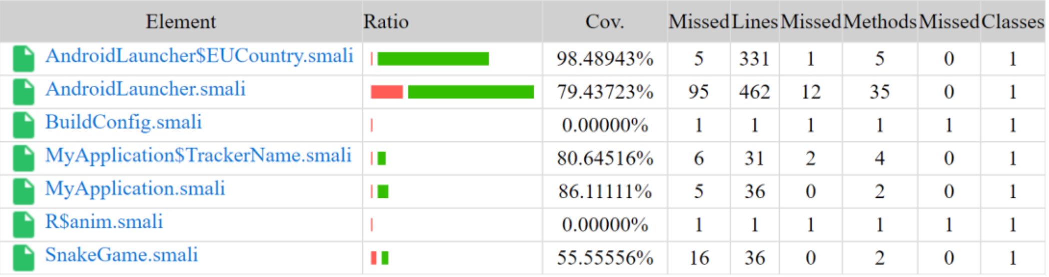
```
.method public static contains(Ljava/lang/String;)Z
        .locals 6
        .param p0, "s"    # Ljava/lang/String;

        const/4 v1, 0x0
        invoke-static {}, Lcom/gnsdm/snake/AndroidLauncher$EUCountry
        move-result-object v3
        array-length v4, v3
        move v2, v1
        :goto_0
        if-ge v2, v4, :cond_0
        aget-object v0, v3, v2
        invoke-virtual {v0}, Lcom/gnsdm/snake/AndroidLauncher$EUCoun
        move-result-object v5
        invoke-virtual {v5, p0}, Ljava/lang/String;->equalsIgnoreCas
        move-result v5
        if-eqz v5, :cond_1
        const/4 v1, 0x1
        :cond_0
        return v1
        :cond_1
        add-int/lit8 v2, v2, 0x1
        goto :goto_0
.end method
```

Approach

ACVTool allows to measure and analyze the degree to which the code of a closed-source Android app is executed during testing, and to collect crash reports occurred during this process. The tool instruments an app and measures code coverage at instruction, method and class granularities.

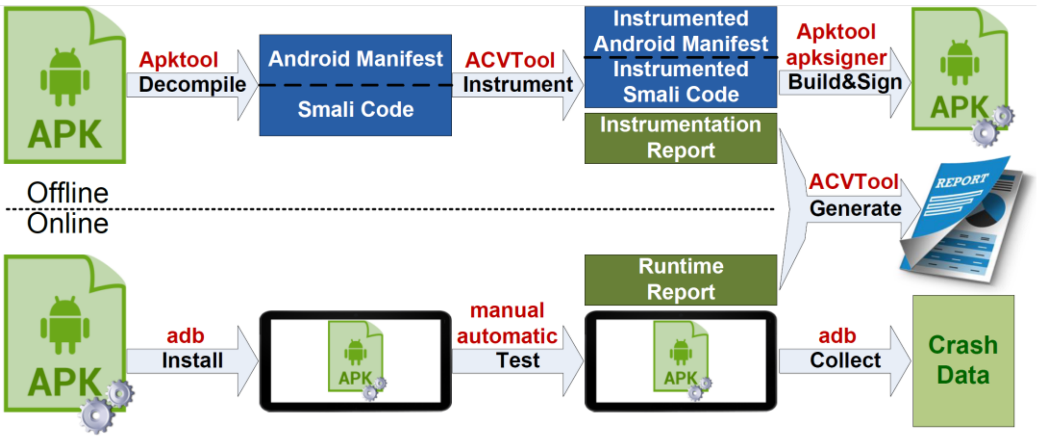
ACVTool produces detailed coverage reports that are convenient for either visual inspections (html), or automatic processing (xml). Our tool also collects crash reports that facilitate the analysis of faults within apps.



Element	Ratio	Cov.	Missed	Lines	Missed	Methods	Missed	Classes
AndroidLauncher$EUCountry.smali	🟥🟩	98.48943%	5	331	1	5	0	1
AndroidLauncher.smali	🟥🟩	79.43723%	95	462	12	35	0	1
BuildConfig.smali	🟥	0.00000%	1	1	1	1	1	1
MyApplication$TrackerName.smali	🟥🟩	80.64516%	6	31	2	4	0	1
MyApplication.smali	🟥🟩	86.11111%	5	36	0	2	0	1
R$anim.smali	🟥	0.00000%	1	1	1	1	1	1
SnakeGame.smali	🟥🟩	55.55556%	16	36	0	2	0	1

ACVTool code coverage report example

Design



Evaluation

We have extensively tested ACVTool on real-life third party applications. The sample consists of 448 runnable applications from F-Droid and 398 randomly selected Google Play applications targeted to the Android API 22+.

Parameter	F-Droid benchmark	Google Play benchmark	Total
Total # selected apps	448	398	846
Average apk size	3.1MB	11.1MB	6.8MB
Instrumented apps	444 (99.1%)	382 (95.9%)	97.6%
Healthy instrumented apps	440 (98.2%)	380 (95.4%)	96.9%
Avg. instrumentation time (total per app)	24.7 sec	49.6 sec	36.2 sec

Conclusion: total ACVTool success rate is 96.9% with average instrumentation time 36 seconds on our dataset.

Conclusions

- We offer to Android security testing community a novel tool for black-box code coverage measurement of Android applications.
- We have significantly improved the *smali* instrumentation technique and consequently our instrumentation success rate is 96.9%, compared with 36% in Huang et al. [2] and 65% in Zhauniarovich et al. [4].
- ACVTool is an open source tool currently available at **https://github.com/pilgun/acvtool**.

References

[1] ELLA. 2016. A Tool for Binary Instrumentation of Android Apps, https://github. com/saswatanand/ella.

[2] C. Huang, C. Chiu, C. Lin, and H. Tzeng. 2015. Code Coverage Measurement for Android Dynamic Analysis Tools. In Proc. of Mobile Services (MS). IEEE, 209–216.

[3] J. Liu, T. Wu, X. Deng, J. Yan, and J. Zhang. 2017. InsDal: A safe and extensible instrumentation tool on Dalvik byte-code for Android applications. In Proc. of SANER. IEEE, 502–506.

[4] Y. Zhauniarovich, A. Philippov, O. Gadyatskaya, B. Crispo, and F. Massacci. 2015. Towards black box testing of Android apps. In Proc. of SAW at ARES. IEEE, 501–510.