

Implementasi *Apktool* Untuk Memanipulasi *SmaliCode* Pada Aplikasi *Android*

Irvan Wijaya¹, Suliswaningsih^{2*}

^{1,2}Universitas Amikom Purwokerto, Indonesia

¹irfanwijayasardam@gmail.com, ²suliswani@amikompurwokerto.ac.id



Histori Artikel:

Diajukan: 12 Januari 2024

Disetujui: 15 Mei 2024

Dipublikasi: 31 Mei 2024

Kata Kunci:

Android; *SmaliCode*;
Manipulation; *APKTool*;
Jadx

Digital Transformation Technology (Digitech) is an
Creative Commons License This
work is licensed under a
Creative Commons Attribution-
NonCommercial 4.0
International (CC BY-NC 4.0).

Abstrak

Keamanan aplikasi *Android* menunjukkan peran esensial dalam era digital yang dipenuhi oleh penggunaan perangkat mobile yang meluas. Dalam konteks ini, penerapan *APKTool* untuk memanipulasi *Smali Code* pada aplikasi *Android* memperoleh signifikansi sebagai strategi utama dalam menjaga keamanan aplikasi. Proses analisis aplikasi dengan menggunakan *jadx* dan dekompilasi *APK* menggunakan *APKTool* memberikan wawasan mendalam mengenai struktur dan logika fundamental, memungkinkan identifikasi potensi kerentanan yang perlu diatasi. Manipulasi *Smali Code* menjadi elemen integral dalam menghadapi dan menanggulangi celah keamanan, menjadikan aplikasi lebih resisten terhadap potensi serangan. Pendekatan ini bukan hanya sebagai respons terhadap ancaman keamanan kontemporer, tetapi juga sebagai upaya proaktif untuk memastikan bahwa aplikasi dapat terus berkembang seiring dengan evolusi tuntutan keamanan dalam lingkungan digital yang dinamis. Oleh karena itu, keamanan aplikasi *Android* tidak sekadar menjadi prasyarat, melainkan fondasi yang sangat penting untuk menyediakan pengalaman pengguna yang aman dan dapat dipercaya.

PENDAHULUAN

Android sebagai sistem operasi mobile paling populer, telah mengalami perkembangan pesat dalam integrasi teknologi di kehidupan sehari-hari. Namun, seiring dengan kemajuan tersebut, tantangan keamanan semakin kompleks, terutama dalam melindungi aplikasi yang menjalankan ekosistem *Android* (Qiu et al., 2020). Keamanan menjadi aspek kritis yang harus diperhatikan, mengingat informasi sensitif yang disimpan dan diproses oleh aplikasi tersebut. Aplikasi berbasis *Android* dan beberapa layanan sistem menggunakan *Android Runtime* (ART). Sebelum ART, lingkungan runtime yang digunakan adalah Dalvik. Keduanya, ART dan Dalvik, diciptakan untuk proyek-proyek terkait aplikasi *Android*. ART mengeksekusi format Dalvik Executable (DEX) dan spesifikasi bytecode. Aspek lainnya adalah manajemen memori dan manajemen daya karena aplikasi berbasis *Android* berjalan pada perangkat bertenaga baterai dengan memori terbatas. Oleh karena itu, sistem operasi *Android* dirancang agar sumber daya apa pun dapat dikelola dengan baik (Senanayake et al., 2021).

Penting untuk mencermati alat-alat yang mendukung analisis dan manipulasi *SmaliCode*, dan dalam konteks ini, *APKTool* muncul sebagai alat yang memainkan peran sentral. *APKTool* memungkinkan dekompilasi dan analisis aplikasi *Android*, termasuk memanipulasi *SmaliCode* untuk keperluan keamanan. Oleh karena itu, implementasi *APKTool* menjadi kunci dalam merespons dan mengatasi tantangan keamanan yang berkembang dalam lingkungan *Android* (Pratama & Daniswara, 2020).

Reverse engineering adalah suatu metode yang melibatkan transformasi source code menjadi model desain yang digunakan dalam rekayasa ulang sistem. Fokus utama dari reverse engineering adalah menciptakan sudut pandang alternatif terhadap suatu sistem, mengembalikan informasi yang hilang, mendeteksi kesalahan dalam sistem, membuat abstraksi sistem yang lebih tinggi, dan memudahkan proses reverse engineering (Mulyani et al., 2022).

Dalam konteks *Android*, reverse engineering seringkali berkaitan dengan analisis dan pemahaman aplikasi *Android* serta sistemnya. Salah satu aspek penting dalam reverse engineering *Android* adalah penggunaan *Smali Code*, yang merupakan representasi bahasa bytecode DEX (Dalvik Executable) dalam bentuk teks (Bostani & Moonsamy, 2024). *Smali Code* digunakan untuk memahami dan memodifikasi aplikasi *Android*, serta membongkar struktur dan logika yang terkandung di dalamnya. Sehingga, reverse engineering tidak hanya menjadi alat untuk memahami perangkat lunak, tetapi juga menjadi kunci untuk mengoptimalkan dan memodifikasi aplikasi *Android* sesuai kebutuhan.

Penelitian ini bertujuan untuk menjelajahi dan mengevaluasi implementasi *APKTool* untuk memanipulasi *SmaliCode* pada aplikasi *Android*. Dengan memfokuskan perhatian pada *Android* sebagai platform, keamanan sebagai isu utama, *SmaliCode* sebagai komponen penting, dan *APKTool* sebagai alat implementasi, penelitian ini diharapkan dapat memberikan kontribusi signifikan dalam upaya meningkatkan keamanan aplikasi *Android* secara menyeluruh.

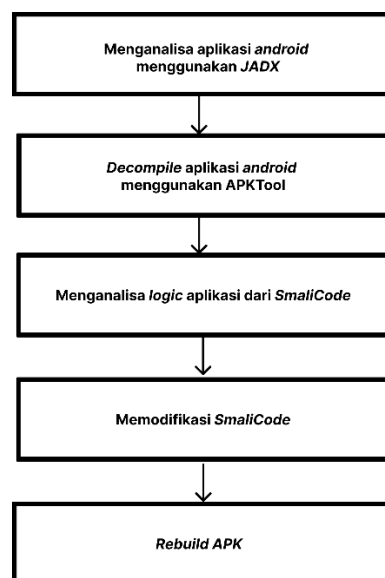
STUDI LITERATUR

Meningkatnya jumlah pengguna ponsel *Android* seiring waktu juga diiringi dengan peningkatan serentak dalam jumlah aplikasi malware mobile yang melakukan aktivitas berbahaya, seperti penggunaan tidak sah terhadap informasi pribadi pengguna, pengiriman pesan SMS berbahaya, dan ekstraksi detail kontak pengguna. Dampaknya tidak hanya terbatas pada pengguna individu, namun juga merugikan organisasi melalui perolehan data pribadi dan rahasia yang tidak sah. Oleh karena itu, klasifikasi dan identifikasi malware menjadi isu kritis dalam konteks keamanan *Android*. Meskipun aplikasi *Android* telah tersebar luas, pengguna umumnya kurang menyadari potensi infeksi malware. Mekanisme izin yang ada, yang seharusnya menyampaikan informasi akses yang diperlukan oleh aplikasi selama instalasi, terbukti kurang efektif dalam melindungi pengguna dari potensi bahaya (Senanayake et al., 2021).

Reverse engineering merupakan metode yang melibatkan transformasi source code menjadi model desain yang digunakan dalam rekayasa ulang system (Arisiah et al., 2023). Dalam konteks *Android*, reverse engineering berkaitan dengan analisis dan pemahaman aplikasi *Android* serta sistemnya. Penggunaan *Smali Code*, representasi bahasa bytecode DEX dalam bentuk teks, menjadi aspek penting dalam reverse engineering *Android*. *Smali Code* digunakan untuk memahami, memodifikasi, dan membongkar struktur serta logika yang terkandung dalam aplikasi *Android* (Dahdouh et al., 2019). Dengan demikian, reverse engineering tidak hanya menjadi alat untuk memahami perangkat lunak, tetapi juga kunci untuk mengoptimalkan dan memodifikasi aplikasi *Android* sesuai kebutuhan.

METODE

Pada penelitian ini terdapat beberapa langkah yang akan dilakukan dalam mengimplementasikan metodologi analisis keamanan aplikasi *Android*. Pertama, aplikasi *Android* akan dianalisis menggunakan JADX guna memahami struktur dan logika dasarnya (Widiyasono et al., 2022). Selanjutnya, *APK* aplikasi akan di-decompile menggunakan *APKTool* untuk mengakses *SmaliCode*. Langkah ketiga melibatkan analisis mendalam terhadap logika aplikasi yang terkandung dalam *SmaliCode*. Setelah identifikasi potensi kerentanan, langkah keempat melibatkan modifikasi langsung pada *SmaliCode* untuk memodifikasi perilaku aplikasi sesuai kebutuhan penelitian. Proses kelima mencakup pembangunan kembali (rebuild) file *APK* menggunakan *APKTool* untuk memastikan integrasi perubahan yang telah diimplementasikan. Dengan serangkaian langkah ini, penelitian ini bertujuan untuk menyelidiki dan memitigasi potensi risiko keamanan pada aplikasi *Android*. Setelah *APK* di *rebuild*, maka *APK* perlu dilakukan *Signing* menggunakan *jarsigner* (Zhang et al., 2021). Alur penelitian dapat dilihat pada Gambar 1:

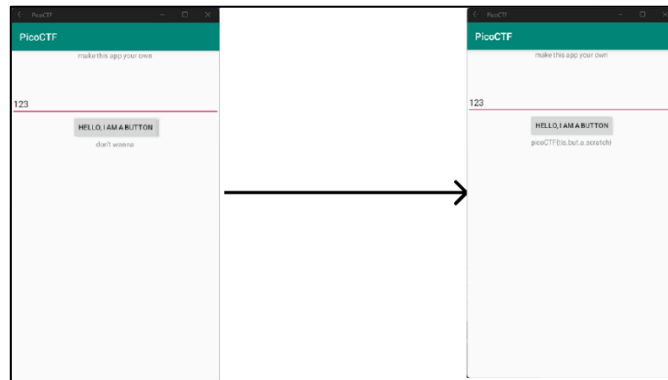


Gambar 1 Alur Penelitian

HASIL

Penelitian ini mencapai pencapaian signifikan dengan menghasilkan sebuah aplikasi yang mengalami perubahan logika melalui modifikasi pada *SmaliCode*. Sebelumnya, ketika diisi dengan angka 123, aplikasi menampilkan teks "don't wanna", namun setelah proses penggantian *SmaliCode* menggunakan *APKTool*, aplikasi kini menampilkan teks yang dirahasiakan. Keberhasilan metode penggantian *SmaliCode* dengan *APKTool* tidak hanya termanifestasi dalam perubahan teks semata, tetapi juga dalam kemampuan mengamankan informasi sensitif yang sebelumnya terpapar. Hasil penelitian ini memperkuat konsep bahwa implementasi *APKTool* dapat

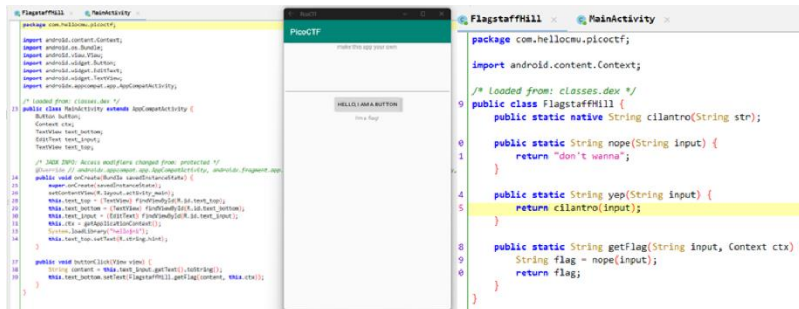
menjadi instrumen efektif dalam memanipulasi *SmaliCode* pada aplikasi *Android*, membuka peluang untuk peningkatan keamanan dan kontrol atas perilaku aplikasi yang kritis.



Gambar 2 Hasil Manipulasi

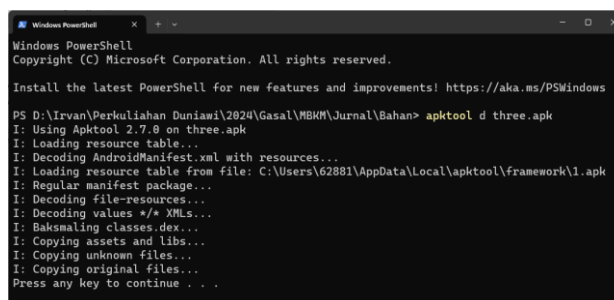
PEMBAHASAN

Dalam analisis menggunakan *JADX*, terdapat temuan bahwa pada suatu aplikasi *Android*, ketika tombol tertentu diklik, proses eksekusi mengarah kepada fungsi *'getFlag'* yang terdapat dalam kelas *'FlagStaffHill'*. Fenomena ini mencerminkan sebuah logika program yang kritis, di mana tombol klik tersebut memicu jalannya operasi atau aksi yang tertanam dalam metode *'getFlag'* pada kelas yang disebutkan. Hal ini dapat memberikan petunjuk signifikan terkait dengan fungsionalitas dan logika aplikasi, dan dari sudut pandang keamanan, mungkin memerlukan perhatian khusus terkait kebijakan akses dan kontrol atas penggunaan fungsi tersebut agar tidak disalahgunakan atau dieksploitasi oleh pihak yang tidak bertanggung jawab. Analisis lebih lanjut pada *SmaliCode* dan struktur aplikasi perlu dilakukan untuk memahami secara mendalam dampak dan implikasi dari eksekusi fungsi *'getFlag'* tersebut.



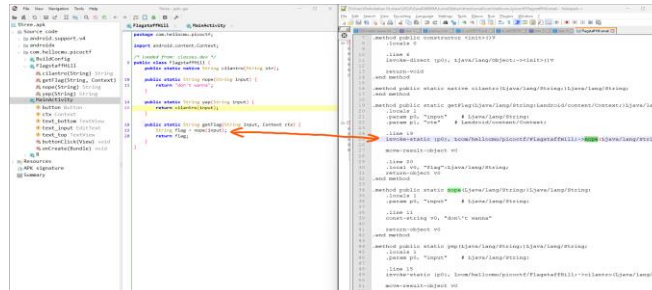
Gambar 1 Proses Analisis Menggunakan JADX

Pada tahap dekompile menggunakan *APKTool*, proses dimulai dengan men-decompile file *APK* sehingga menghasilkan *Smali Code* yang mencakup representasi teks dari *bytecode Java* yang terdapat dalam aplikasi *Android*. Setelah proses dekompile selesai, *Smali Code* dari setiap kelas dalam aplikasi dapat dianalisis lebih lanjut. Melalui penerjemahan kode sumber yang telah didekompile, peneliti dapat memahami logika dan struktur aplikasi secara mendalam. Fokus analisis pada *Smali Code* menjadi krusial, karena hal ini memungkinkan identifikasi potensi kerentanan atau celah keamanan yang mungkin terdapat dalam aplikasi (Anand et al., 2023). Dengan demikian, tahap ini menjadi fondasi untuk langkah-langkah berikutnya dalam manipulasi dan pengujian keamanan pada aplikasi *Android*.



Gambar 2 Proses Dekompilasi menggunakan *APKTool*

Setelah proses dekompilasi selesai, hasilnya berupa *Smali Code* dari berbagai kelas dalam aplikasi, termasuk kelas *FlagStaffHill* yang menjadi fokus analisis. Pada tahap ini, pemahaman terhadap logika aplikasi semakin terperinci, khususnya dalam fungsi *getFlag* yang memanggil fungsi *Nope*. Melalui analisis *Smali Code*, terungkap bahwa fungsi *Nope* selalu memberikan nilai kembalian "don't wanna". Dengan pemahaman ini, penelitian berlanjut ke tahap modifikasi *Smali Code* sesuai dengan logika yang diinginkan.

Gambar 3 Proses Analisa *Smali Code*

Dalam kasus ini, modifikasi dilakukan pada fungsi *getFlag*, yang awalnya mengeksekusi fungsi *Nope*, menjadi mengeksekusi fungsi *Yep*. Dengan perubahan ini, diharAPkan aplikasi akan mengembalikan data yang sesuai dengan logika yang diinginkan oleh peneliti. Tahap ini menunjukkan bagaimana pemahaman terhadap *Smali Code* memungkinkan peneliti untuk mengidentifikasi, menganalisis, dan memodifikasi logika aplikasi sesuai kebutuhan penelitian.

```
.end method
.method public static getFlag(Ljava/lang/String;Landroid/content/Context;)Ljava/lang/String;
    .locals 1
    .param p0, "input"    # Ljava/lang/String;
    .param p1, "ctx"      # Landroid/content/Context;

    .line 19
    invoke-static {p0}, Lcom/hellomum/picoctf/FlagstaffHill;->get(Ljava/lang/String;)Ljava/lang/String;
    move-result-object v0

    .line 20
    local v0, "flag":Ljava/lang/String;
    return-object v0
.end method

.method public static nope(Ljava/lang/String;)Ljava/lang/String;
    .locals 1
    .param p0, "input"    # Ljava/lang/String;

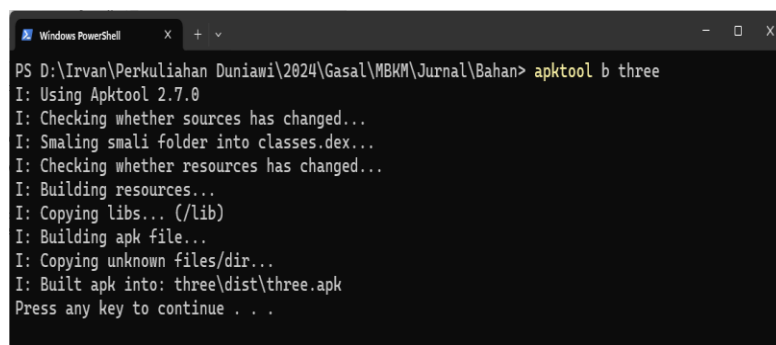
    .line 11
    const-string v0, "don't wanna"
    return-object v0
.end method

.method public static yep(Ljava/lang/String;)Ljava/lang/String;
    .locals 1
    .param p0, "input"    # Ljava/lang/String;

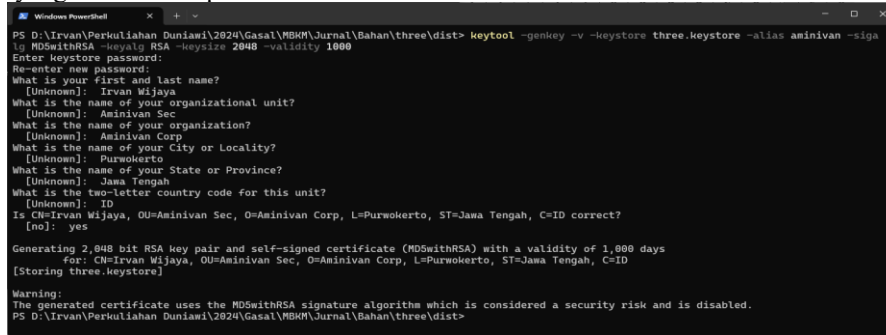
    .line 15
    invoke-static {p0}, Lcom/hellomum/picoctf/FlagstaffHill;->vclantro(Ljava/lang/String;)Ljava/lang/String;
    move-result-object v0
    return-object v0
.end method
```

Gambar 4 Proses manipulasi *Smali Code*

Saat ini, setelah berhasil mengganti fungsi *getFlag* untuk mengeksekusi fungsi *Yep*, langkah terakhir dalam proses manipulasi aplikasi *Android* adalah melakukan pembangunan ulang (*rebuild*) *APK* menggunakan *APKTool*. Tahap ini krusial untuk mengintegrasikan perubahan yang telah diterAPkan pada *Smali Code* ke dalam struktur aplikasi secara keseluruhan. Dengan me-rebuild *APK*, hasil akhirnya akan menjadi sebuah aplikasi yang telah dimodifikasi sesuai dengan logika yang diinginkan. Proses ini memastikan bahwa perubahan yang telah dilakukan pada level *Smali Code* dapat diaplikasikan secara efektif ke dalam *APK* sehingga aplikasi yang dihasilkan tetap berjalan dengan fungsi yang telah dimodifikasi (Zhao et al., 2020). Dengan demikian, tahap rebuild *APK* menjadi penutup yang esensial dalam rangkaian proses manipulasi *Smali Code* menggunakan *APKTool* dalam upaya pengujian dan analisis keamanan aplikasi *Android*.

Gambar 5 Proses Rebuild *APK* menggunakan *APKTool*

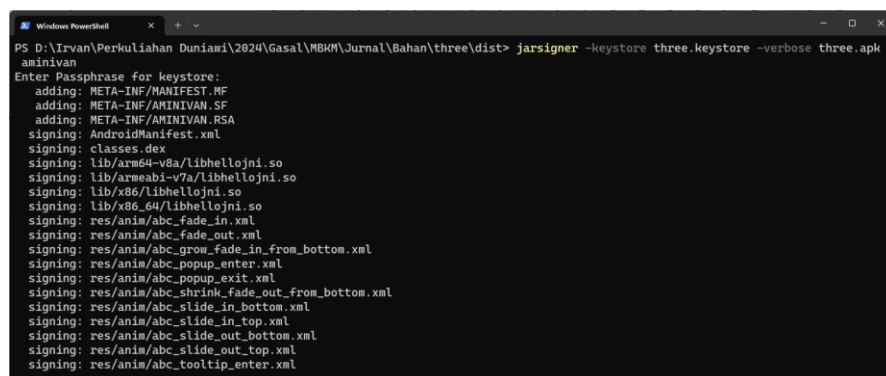
Setelah menyelesaikan proses dekompilasi menggunakan *APKTool*, langkah selanjutnya adalah melakukan penandatanganan (signing) kembali *APK* yang telah dimodifikasi, menggunakan alat seperti Jarsigner. Tindakan ini diperlukan untuk memvalidasi integritas dan otentikasi aplikasi yang telah mengalami perubahan, sekaligus menjamin bahwa aplikasi tersebut berasal dari sumber yang sah (Yoshida et al., 2019). Dengan memanfaatkan Jarsigner, kita memberikan cap digital pada *APK* yang dihasilkan, memastikan bahwa setiap perubahan yang dilakukan pada *Smali Code* atau bagian lain dari aplikasi dapat diakui dan diterima sebagai hasil modifikasi yang sah (Sharma et al., n.d.). Proses penandatanganan *APK* dengan menggunakan Jarsigner menjadi langkah penting untuk memastikan keberhasilan implementasi perubahan dan modifikasi, serta mengukuhkan integritas keamanan aplikasi *Android* yang telah dimanipulasi.



```
PS D:\Irvan\Perkuliah Duniawi\2024\Gasal\MBKM\Jurnal\Bahan\three\dist> keytool -genkey -v -keystore three.keystore -alias aminivan -sigalg MD5withRSA -keyalg RSA -keysize 2048 -validity 1000
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Irvan Wijaya
What is the name of your organizational unit?
[Unknown]: Aminivan Sec
What is the name of your organization?
[Unknown]: Aminivan Corp
What is the name of your City or Locality?
[Unknown]: Purwokerto
What is the name of your State or Province?
[Unknown]: Jawa Tengah
What is the two-letter country code for this unit?
[Unknown]: ID
Is CN=Irvan Wijaya, OU=Aminivan Sec, O=Aminivan Corp, L=Purwokerto, ST=Jawa Tengah, C=ID correct?
[no]: yes
Generating 2,048 bit RSA key pair and self-signed certificate (MD5withRSA) with a validity of 1,000 days
for: CN=Irvan Wijaya, OU=Aminivan Sec, O=Aminivan Corp, L=Purwokerto, ST=Jawa Tengah, C=ID
[Storing three.keystore]
Warning:
The generated certificate uses the MD5withRSA signature algorithm which is considered a security risk and is disabled.
PS D:\Irvan\Perkuliah Duniawi\2024\Gasal\MBKM\Jurnal\Bahan\three\dist>
```

Gambar 6 Membuat sertifikat untuk aplikasi

Tujuan utama dari Jarsigner adalah memastikan integritas dan otentikasi JAR file dalam lingkungan Java. Dengan menambahkan cap digital menggunakan kunci privat, Jarsigner memberikan tanda yang dapat diverifikasi untuk memvalidasi bahwa file tersebut tidak mengalami perubahan yang tidak sah (Sihag et al., 2021). Proses ini juga memberikan autentikasi terhadap sumber JAR file, melindunginya dari modifikasi yang tidak sah, serta mencegah repudiasi dengan menetapkan keterkaitan antara file dan sumbernya.



```
PS D:\Irvan\Perkuliah Duniawi\2024\Gasal\MBKM\Jurnal\Bahan\three\dist> jarsigner -keystore three.keystore -verbose three.apk aminivan
Enter passphrase for keystore:
adding: META-INF/MANIFEST.MF
adding: META-INF/AMINIVAN.SF
adding: META-INF/AMINIVAN.RSA
signing: AndroidManifest.xml
signing: classes.dex
signing: lib/arm64-v8a/libhellojni.so
signing: lib/armeabi-v7a/libhellojni.so
signing: lib/x86/libhellojni.so
signing: lib/x86_64/libhellojni.so
signing: res/anim/abc_fade_in.xml
signing: res/anim/abc_fade_out.xml
signing: res/anim/abc_grow_fade_in_from_bottom.xml
signing: res/anim/abc_popup_enter.xml
signing: res/anim/abc_popup_exit.xml
signing: res/anim/abc_shrink_fade_out_from_bottom.xml
signing: res/anim/abc_slide_in_bottom.xml
signing: res/anim/abc_slide_in_top.xml
signing: res/anim/abc_slide_out_bottom.xml
signing: res/anim/abc_slide_out_top.xml
signing: res/anim/abc_tooltip_enter.xml
```

Gambar 7 Proses Signing

Setelah proses penandatanganan menggunakan Jarsigner, aplikasi yang telah mengalami manipulasi pada *Smali Code* dapat diinstal dan dijalankan pada perangkat *Android*. Tanda digital yang ditambahkan memastikan bahwa integritas dan otentikasi aplikasi tetap terjaga, sehingga sistem dapat mengenali dan menerima aplikasi tersebut sebagai hasil modifikasi yang sah. Dengan demikian, penandatanganan kembali (re-signing) aplikasi menjadi langkah penting dalam memastikan keberhasilan implementasi perubahan logika pada aplikasi *Android*, sehingga aplikasi dapat berfungsi sesuai dengan logika yang telah dimanipulasi.

KESIMPULAN

Secara keseluruhan, manipulasi *Smali Code* menggunakan alat seperti *APKTool* dan proses penandatanganan kembali dengan Jarsigner memiliki dampak signifikan dalam konteks pengujian dan analisis keamanan aplikasi *Android*. Proses dekompilasi dengan *APKTool* memberikan akses ke *Smali Code*, memungkinkan identifikasi logika aplikasi secara mendalam. Melalui analisis *Smali Code*, perubahan logika aplikasi dapat diidentifikasi dan dimodifikasi sesuai kebutuhan. Langkah krusial berikutnya adalah menggunakan Jarsigner untuk memberikan tanda digital pada *APK* yang dimodifikasi, memastikan keabsahan dan keamanan dari perubahan tersebut. Dengan demikian, aplikasi hasil modifikasi dapat diinstal dan berjalan di perangkat *Android* dengan mempertahankan integritas dan otentikasinya. Keseluruhan proses ini memberikan wawasan

penting terhadap kerentanan keamanan, memungkinkan pemahaman yang lebih baik tentang cara melindungi aplikasi *Android* dari potensi risiko keamanan.

REFERENSI

- Anand, A., Singh, J. P., Khan, R. S., Kumari, A., & Mishra, D. (2023). *Android Malware Detection using LSTM with Smali Codes*.
- Arisiah, M. A., Kusnaty, A., & Caesaron, D. (2023). Penerapan Reverse Engineering Dan Simulasi Dalam Perancangan Drum Ponton Untuk Pembakaran Tempurung Kelapa. *EProceedings of Engineering*, 10(2).
- Bostani, H., & Moonsamy, V. (2024). Evadroid: A practical evasion attack on machine learning for black-box android malware detection. *Computers & Security*, 139, 103676.
- Dahdouh, M., Ayman, M., Khawatmi, S., & Bouchi, A. (2019). Design an Optimized Compiler to Enhance Performance of Android Applications. *International Journal of Computer Applications*, 975, 8887.
- Mulyani, H., Nugraha, M., & others. (2022). Perancangan Sistem Informasi Institutional Repository Politeknik Enjinering Indoroma. *Technomedia Journal*, 6(2 Februari), 152–162.
- Pratama, I. P. A. E., & Daniswara, R. R. (2020). Pengujian dan Analisa Reverse Engineering Pada Platform Android (Studi Kasus: Tebak_Gambar.apk). *JTT (Jurnal Teknologi Terpadu)*, 8(2), 69–76.
- Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., & Xiang, Y. (2020). A survey of android malware detection with deep neural models. *ACM Computing Surveys (CSUR)*, 53(6), 1–36.
- Senanayake, J., Kalutarage, H., & Al-Kadri, M. O. (2021). Android mobile malware detection using machine learning: A systematic review. *Electronics*, 10(13), 1606.
- Sharma, P., Lepcha, C., Bhutia, S. T., & Sharma, A. (n.d.). *CASE STUDY EXPLOIT OF ANDROID DEVICES USING PAYLOAD INJECTED APK*.
- Sihag, V., Vardhan, M., & Singh, P. (2021). A survey of android application and malware hardening. *Computer Science Review*, 39, 100365.
- Widiyasono, N., Mubarak, H., & MF, A. F. (2022). Analisis Malware Ahmyth pada Platform Android Menggunakan Metode Reverse Engineering. *Generation Journal*, 6(2), 114–123.
- Yoshida, K., Imai, H., Serizawa, N., Mori, T., & Kanaoka, A. (2019). Understanding the origins of weak cryptographic algorithms used for signing android apps. *Journal of Information Processing*, 27, 593–602.
- Zhang, X., Breiting, F., Luechinger, E., & O'Shaughnessy, S. (2021). Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations. *Forensic Science International: Digital Investigation*, 39, 301285.
- Zhao, Y., Tang, Z., Ye, G., Peng, D., Fang, D., Chen, X., & Wang, Z. (2020). Compile-time code virtualization for android applications. *Computers & Security*, 94, 101821.