Demystifying SMALI Analysis for Penetration Testing - Custom Log Messages

บทนำ (Overview)

บทความนี้จะแนะนำขั้นตอนการเพิ่ม Debug Log ใน SMALI ตั้งแต่การค้นหาหน้าจอ การระบุ Method การแก้ไขไฟล์ SMALI จนถึงการทดสอบ Debug Log ที่ได้ ทั้งนี้เพื่อเป็นพื้นฐานว่าสามารถตรวจสอบและวิเคราะห์ปที่เกิดขึ้นภายในแอปได้ได้อย่างง่ายขึ้น บทความนี้ต่อจากบทความ Getting Started with SMALI for Penetration Testing ที่อธิบายถึงพื้นฐานและหลักการของการ Decompile และ SMALI

Warunyou Sunpachit และ Boonperm Mark
Cybersecurity consultant

1. การเพิ่ม Debug Log ใน SMALI

- ใช้ `apktool` เพื่อแปลงแอป Android ของเป็น smali code:
    1. `-f` (หรือ `--force`): เป็นพารามิเตอร์ที่บังคับใช้ หากมีการใช้จะบังคับให้ Apktool ลบโฟลเดอร์ปลายทาง (output directory) หากมีอยู่แล้ว และทำการถอดแปลงใหม่
    2. `-o output`: คือพารามิเตอร์ที่ใช้ระบุโฟลเดอร์ปลายทางที่ต้องการให้ Apktool บันทึกโค้ดและโครงสร้างทรัพยากรที่ถอดแปลงลงใน
    3. --use-aapt2`: เป็นพารามิเตอร์ที่บังคับใช้เพื่อระบุให้ Apktool ใช้ AAPT2 (Android Asset Packaging Tool 2) ในการประมวลผลทรัพยากรของ APK แทน AAPT1 ซึ่งเป็นเวอร์ชันเก่าของ AAPT.

```
apktool d your_app.apk -f -o output_folder --use-aapt2
```

ตัวอย่าง

```
λ apktool d permcheck_ctf2023.apk -f -o output --use-aapt2
I: Using Apktool 2.8.1 on permcheck_ctf2023.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file:
C:\Users\Asus\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

- เปิด SMALI code ที่ต้องการแก้ไขโดยใช้โปรแกรมแก้ไขข้อความ (text editor) และค้นหาส่วนที่ต้องการเพิ่ม debug log ลงไป สำหรับการเพิ่ม log ใน SMALI code สามารถใช้คำสั่ง `invoke-static` เพื่อเรียกใช้ Log.d() หรือ Log.i() ได้ เช่น:

```
const-string v0, "MyDebugTag"
const-string v1, "Debug Message"
invoke-static {v0, v1}, Landroid/util/Log;-
>d(Ljava/lang/String;Ljava/lang/String;)I
```

- เมื่อแก้ไขเสร็จสิ้นให้ทำการ Build กลับเป็น APK เหมือนเดิมโดยใช้ apktool
    1. `output_folder`: ระบุโฟลเดอร์ที่มีโครงสร้างของแอปพลิเคชัน Android ที่ได้แก้ไขแล้วโดยใช้ `apktool` ในขั้นตอนก่อนหน้านี้ โดยทั่วไปแล้ว, `output_folder` คือโฟลเดอร์ที่สร้างขึ้นจากการใช้คำสั่ง `apktool d` เพื่อแปลง APK เป็นโครงสร้างแฟ้มและเนื้อหาของแอปพลิเคชัน.
    2. `-o output.apk`: ระบุชื่อและตำแหน่งที่ต้องการให้ APK ที่สร้างด้วยคำสั่งนี้ถูกบันทึก ในที่นี้ `output.apk` คือชื่อไฟล์ APK ที่จะสร้าง

```
apktool b output_folder -o output.apk --use-aapt2
```

ตัวอย่าง

```
λ apktool b output -o output.apk --use-aapt2
I: Using Apktool 2.8.1
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Copying libs... (/kotlin)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk into: output.apk
```

- Signed App ให้เรียบร้อยเพื่อให้สามารถติดตั้งบนเครื่อง Android ได้ โดยสามารถดาวน์โหลด Uber Apk Signer และใช้คำสั่งดังนี้

```
java -jar uber-apk-signer.jar --apks /path/to/apks
```

ตัวอย่าง

```
λ java -jar uber-apk-signer-1.3.0.jar --apks output.apk
source:
        C:\Users\Asus\Desktop\Tool
binary-lib/windows-33_0_2/libwinpthread-1.dll
C:\Users\Asus\AppData\Local\Temp\uapksigner-12247413487323163228
zipalign location: BUILT_IN
        C:\Users\Asus\AppData\Local\Temp\uapksigner-12247413487323163228\win-
zipalign_33_0_2.exe8916485263224414034.tmp
keystore:
        [0] 54e0fa34 C:\Users\Asus\.android\debug.keystore (DEBUG_ANDROID_FOLDER)

01. output.apk

        SIGN
        file: C:\Users\Asus\Desktop\Tool\output.apk (17.61 MiB)
        checksum: d4b44af0c3655eae904b2582e0ae84bbaea2d3f3b3ca836aeab2ec3688bd043
(sha256)
        - zipalign success
        - sign success

        VERIFY
        file: C:\Users\Asus\Desktop\Tool\output-aligned-debugSigned.apk (17.7
MiB)
        checksum:
a6b82440337fff3885eedaf532675663aa4b598038239bfc47914fad988aad15 (sha256)
        - zipalign verified
        - signature verified [v3]
                Subject: C=US, O=Android, CN=Android Debug
                SHA256:
b51b05fa5b3f67200ed67c9361cf1c56d257586e34839cb4671848cf49ce6192 / SHA256withRSA
                Expires: Sat Aug 27 21:20:11 ICT 2596

[Sat Sep 23 11:21:01 ICT 2023][v1.3.0]
Successfully processed 1 APKs and 0 errors in 0.84 seconds.
```

2. ค้นหาหน้าจอการใช้งานปัจจุบัน

- การค้นหาหน้าจอการใช้งานปัจจุบันเป็นขั้นตอนสำคัญในกระบวนการวิเคราะห์แอปและการทำงานของ Smali ในแอป Android ช่วยในการวิเคราะห์และแก้ไข Smali code ของแอปนั้น ๆ ให้ตรงกับความต้องการ ก่อนอื่นใช้คำสั่งจามข้างล่างตรวจสอบว่าได้สิทธิ Root เพื่อเข้าถึงระบบระบบ Android

```
adb shell
su
id
```

- คำอธิบายเพิ่มเติม

1. `adb shell` : คำสั่งนี้ใช้ในการเปิดหน้าจอคอมมานด์ไลน์เชลล์ของอุปกรณ์ Android ซึ่งช่วยให้สามารถป้อนคำสั่งแบบเชลล์ที่สามารถทำงานในระบบปฏิบัติการ Android ได้โดยตรง เป็นเครื่องมือที่มีประโยชน์ในการดำเนินการหลายอย่างเช่นดูไฟล์และโฟลเดอร์ในอุปกรณ์ รันแอพพลิเคชันเป็นผู้ใช้ root (superuser) และอื่น ๆ
2. `su` : คำสั่ง `su` ถูกใช้ในการเปลี่ยนสิทธิของผู้ใช้ให้เป็น root หรือ superuser
3. `id` : คำสั่ง `id` แสดงข้อมูลเกี่ยวกับบัญชีผู้ใช้ที่กำลังใช้งานเชลล์ รวมถึง UID (User ID) และ GID (Group ID) ของผู้ใช้นั้น คำสั่งนี้ช่วยให้รู้ว่ากำลังใช้งานเชลล์ด้วยสิทธิพิเศษหรือไม่ ถ้าเห็น UID 0 ในผลลัพธ์ แสดงว่ากำลังใช้งานเป็น superuser หรือ root และมีสิทธิพิเศษในการจัดการระบบ Android ในระดับสูงสุด

ตัวอย่าง

```
λ adb shell
emu64xa:/ $ su
emu64xa:/ # id
uid=0(root) gid=0(root)
groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),107
8(ext_data_rw),1079(ext_obb_rw),3001(net_bt_admin),3002(net_bt),3003(inet),3006(n
et_bw_stats),3009(readproc),3011(uhid),3012(readtracefs) context=u:r:su:s0
```

- การแสดงข้อมูลเกี่ยวกับสถานะของหน้าจอและแอปพลิเคชันที่กำลังโฟกัสในอุปกรณ์ Android โดยใช้คำสั่ง `dumpsys`

```
dumpsys window | grep -E 'mCurrentFocus|mFocusedApp'
```

- `dumpsys` : คำสั่ง `dumpsys` เป็นคำสั่งที่ใช้ในระบบปฏิบัติการ Android เพื่อดูและรายงานข้อมูลเกี่ยวกับการทำงานและสถานะของระบบ Android ได้โดยละเอียด
- `dumpsys window` เป็นคำสั่งที่ใช้ในระบบปฏิบัติการ Android เพื่อดูข้อมูลและสถานะที่เกี่ยวข้องกับหน้าจอและการจัดการหน้าจอของอุปกรณ์ Android ได้โดยละเอียด
- `|` : สัญลักษณ์แทนการนำเอาผลลัพธ์จากคำสั่งที่อยู่ด้านซ้ายของ `|` มาส่งให้กับคำสั่งที่อยู่ด้านขวา เป็นการเชื่อมคำสั่งสองคำสั่งเข้าด้วยกัน.
- `grep -E 'mCurrentFocus|mFocusedApp'` : คำสั่ง `grep` ใช้ในการค้นหาและกรองข้อมูลจากผลลัพธ์ที่เป็นข้อความ ตรงกับรูปแบบที่ระบุใน `'mCurrentFocus|mFocusedApp'`

ตัวอย่าง

```
emu64xa:/ # dumpsys window | grep -E 'mCurrentFocus|mFocusedApp'
  mCurrentFocus=Window{a6e2f21 u0
com.itselectlab.permcheck/com.itselectlab.permcheck.MainActivity}
  mFocusedApp=ActivityRecord{4a318c5 u0 com.itselectlab.permcheck/.MainActivity
t23}
```

- คำอธิบายเพิ่มเติม

  ผลลัพธ์ที่จะได้จากคำสั่งนี้คือข้อมูลเกี่ยวกับ `mCurrentFocus` และ `mFocusedApp` ซึ่งระบุหน้าจอและแอปพลิเคชันที่กำลังมีการโฟกัสอยู่ในขณะนั้นบนอุปกรณ์ Android ของ ข้อมูลนี้สามารถช่วยให้ตรวจสอบหน้าจอหรือแอปพลิเคชันที่มีการโฟกัสในระหว่างการใช้งานอุปกรณ์ Android ของได้

`mCurrentFocus=Window{a6e2f21 u0 com.itselectlab.permcheck/com.itselectlab.permcheck.MainActivity}`:

1. `mCurrentFocus` คือตัวแปรที่เก็บข้อมูลเกี่ยวกับหน้าจอที่กำลังโฟกัสอยู่ในขณะนี้.
2. window{a6e2f21 u0 com.itselectlab.permcheck/com.itselectlab.permcheck.MainActivity} คือข้อมูลของหน้าจอปัจจุบันที่กำลังโฟกัสอยู่:
   - `a6e2f21` คือ ID ของหน้าจอ (Window ID).
   - `u0` คือ User ID ของผู้ใช้
   - `com.itselectlab.permcheck/com.itselectlab.permcheck.MainActivity` คือชื่อแพ็กเกจ (package name) และชื่อคลาส (class name) ของ Activity ที่กำลังโฟกัสอยู่ ในที่นี้คือแอปพลิเคชัน `com.itselectlab.permcheck` และคลาส `MainActivity`

`mFocusedApp=ActivityRecord{4a318c5 u0 com.itselectlab.permcheck/.MainActivity t23}`:

1. `mFocusedApp` คือตัวแปรที่เก็บข้อมูลเกี่ยวกับแอปพลิเคชันที่กำลังมีการโฟกัสอยู่ในขณะนี้.
2. `activityRecord{4a318c5 u0 com.itselectlab.permcheck/.MainActivity t23}` คือข้อมูลของแอปพลิเคชันที่กำลังโฟกัสอยู่:
   - `4a318c5` คือ ID ของ Activity
   - `u0` คือ User ID ของผู้ใช้
   - `com.itselectlab.permcheck/.MainActivity` คือชื่อแพ็กเกจและคลาสของ Activity ที่กำลังโฟกัสอยู่ ในที่นี้คือแอปพลิเคชัน `com.itselectlab.permcheck` และคลาส `MainActivity``
   - `` `t23 `` คือข้อมูลเพิ่มเติมเกี่ยวกับการจัดการแอปพลิเคชัน (App Task) โดย `t23` คือ Task ID ของแอปพลิเคชันที่กำลังโฟกัสอยู่.

- ถ้าเราลองเปลี่ยนหน้าจอจะพบว่า Activity จะเปลี่ยนไป จากตัวอย่างเช่น `AppActivitiesActivity`

```
emu64xa:/ $ dumpsys window | grep -E 'mCurrentFocus|mFocusedApp'
  mCurrentFocus=Window{f7f5fcd u0
com.itselectlab.permcheck/com.itselectlab.permcheck.AppActivitiesActivity}
  mFocusedApp=ActivityRecord{aad9b91 u0
com.itselectlab.permcheck/.AppActivitiesActivity t7}
emu64xa:/ $
```

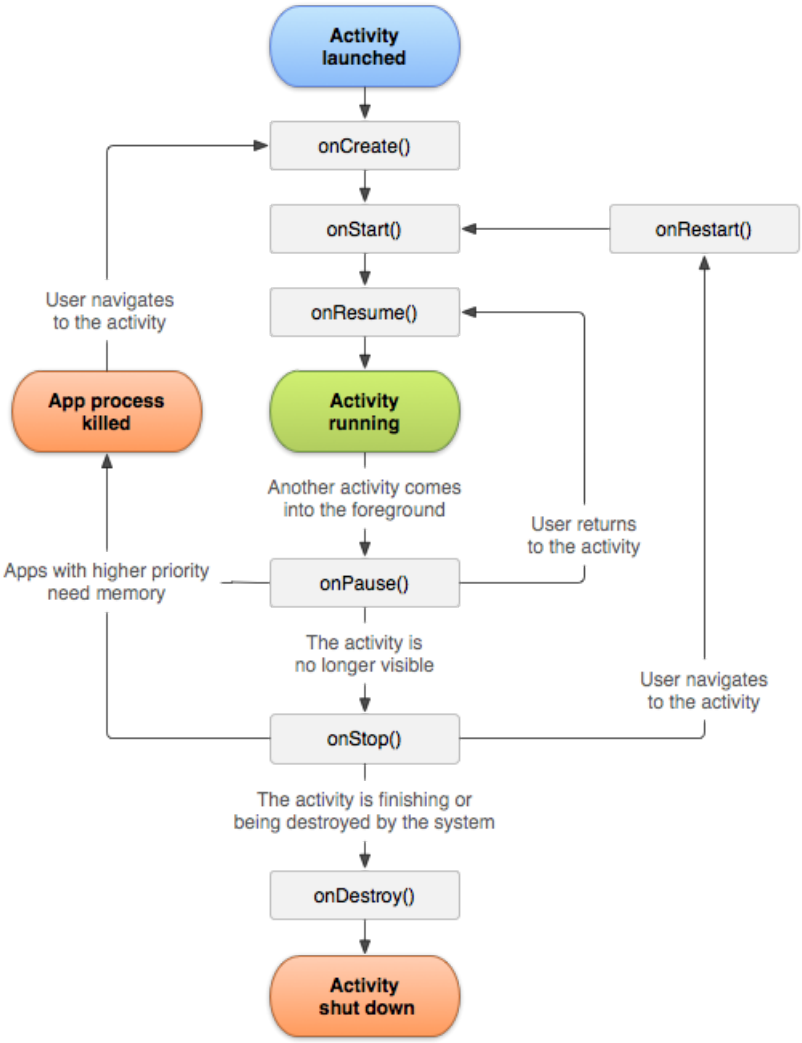
3. ค้นหา Method ที่จะใส่ Debug Log

- ส่วนที่เหมาะที่สุดในการเพิ่มโค้ด Debug Log คือภายในเมทอด (method) เช่น `onCreate()`, `onStart()`, `onResume()`, `onClick()` เป็นต้น โดยขึ้นอยู่กับจุดที่ต้องการตรวจสอบการทำงานของแอคทิวิตี้นั้นๆ โดยแต่ละ Method มีความสำคัญต่างกันดังนี้

1. `onCreate()`: ทำงานครั้งเดียวเมื่อ Activity ถูกสร้างขึ้นและใช้ในการตั้งค่าพื้นฐานและกำหนดเนื้อหาหรือ layout ของ Activity
2. `onStart()`: เริ่มต้นการทำงานที่ต้องการเมื่อ Activity กำลังจะแสดงบนหน้าจอ แต่ยังไม่มีการทำงานกับผู้ใช้

3. `onResume()` : ทำงานทุกครั้งที่ Activity แสดงบนหน้าจอและพร้อมให้ผู้ใช้ทำงาน
4. `onClick()` : ทำงานเมื่อผู้ใช้คลิกหรือทำการกระทำบนองค์ประกอบในหน้าจอ โดยเฉพาะเมื่อผู้ใช้ทำการกระทำ

- อ้างอิงรูปจาก [Activity-lifecycle concepts]



- `onCreate()`, `onStart()`, `onResume()`, และ `onClick()` เป็นเมทอด (methods) ที่มักถูกเขียนในภาษา Java หรือ Kotlin เมื่อพัฒนาแอปพลิเคชัน Android โดยตรง แต่ในไฟล์ Smali ซึ่งเป็นภาษาแอสเซมบลี การเขียน method จะมีรูปแบบที่แตกต่างกัน ดังนี้

1. `onCreate()` ใน Smali สำหรับ Activity อาจมีลักษณะประมาณนี้:

```
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 0

    .prologue #ไม่มีผลกระทบต่อการทำงานของ method แต่ช่วยบอกส่วนเริ่มต้นของ code
    .line 10 #ช่วยระบุบรรทัดของ code เมื่อแปลงกลับไปเป็น Java หรือ Kotlin
    invoke-super {p0, p1}, Landroid/app/Activity;-
>onCreate(Landroid/os/Bundle;)V

    .line 11
    return-void
.end method
```

2. `onStart()` ใน Smali สำหรับ Activity อาจมีลักษณะประมาณนี้:

```
.method protected onStart()V
    .locals 0

    .prologue
    .line 10
    invoke-super {p0}, Landroid/app/Activity;->onStart()V

    .line 11
    return-void
.end method
```

3. `onResume()` ใน Smali สำหรับ Activity อาจมีลักษณะประมาณนี้:

```
.method protected onResume()V
    .locals 0

    .prologue
    .line 10
    invoke-super {p0}, Landroid/app/Activity;->onResume()V

    .line 11
    return-void
.end method
```

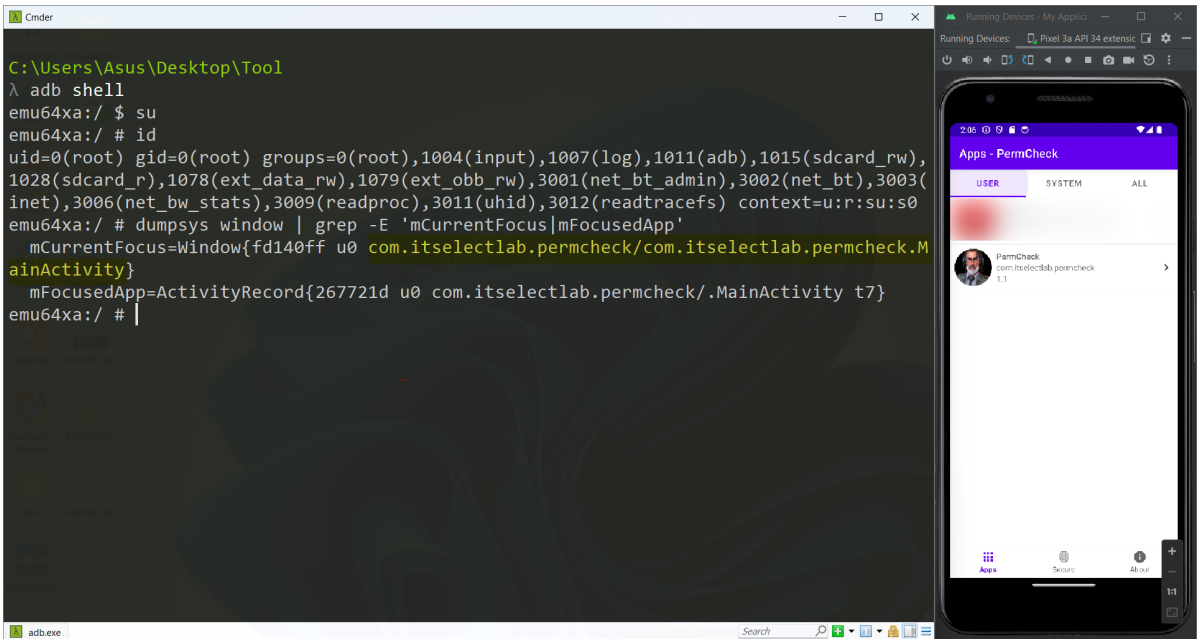
4. `onClick()` ใน Smali จะขึ้นอยู่กับโค้ดที่ต้องการที่จะเรียกเมื่อมีการคลิก ตัวอย่างเบื้องต้นของการเขียน method `onClick()` สำหรับปุ่ม (Button) ในหน้าจอ:

```
.method public onClick(Landroid/view/View;)V
    .locals 1

    .prologue
    .line 10

    .line 11
    const v0, 0x7f080001

    .line 12
    invoke-static {p0, v0}, Landroid/widget/Toast;-
>makeText(Landroid/content/Context;I)Landroid/widget/Toast;

    move-result-object v0

    invoke-virtual {v0}, Landroid/widget/Toast;->show()V

    .line 13
    return-void
.end method
```

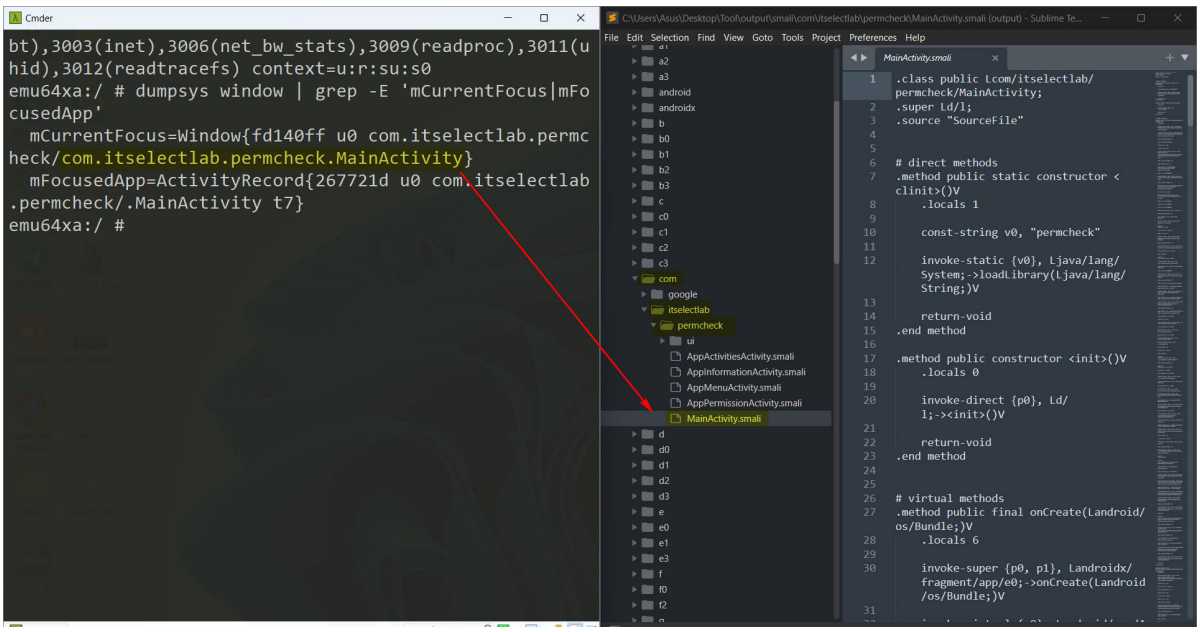
4. แก้ไข SAMLI ด้วย App จริง

- ค้นหาหน้าจอการใช้งานปัจจุบัน โดยใช้คำสั่ง

```
dumpsys window | grep -E 'mCurrentFocus|mFocusedApp
```



- ทำการค้นหาไฟล์ SMALI จาก Activity ที่ค้นพบ



- ค้นหา Method ชื่อ `onCreate()`

```
# virtual methods
.method public final onCreate(Landroid/os/Bundle;)V
    .locals 6

    invoke-super {p0, p1}, Landroidx/fragment/app/e0;-
>onCreate(Landroid/os/Bundle;)V

    --SNIP--

.end method
```

- ใส่ Debug ลงไปในบรรทัดที่ต้องการ

```
const-string v0, "MyDebugTag"
const-string v1, "Debug Message"
invoke-static {v0, v1}, Landroid/util/Log;-
>d(Ljava/lang/String;Ljava/lang/String;)I
```

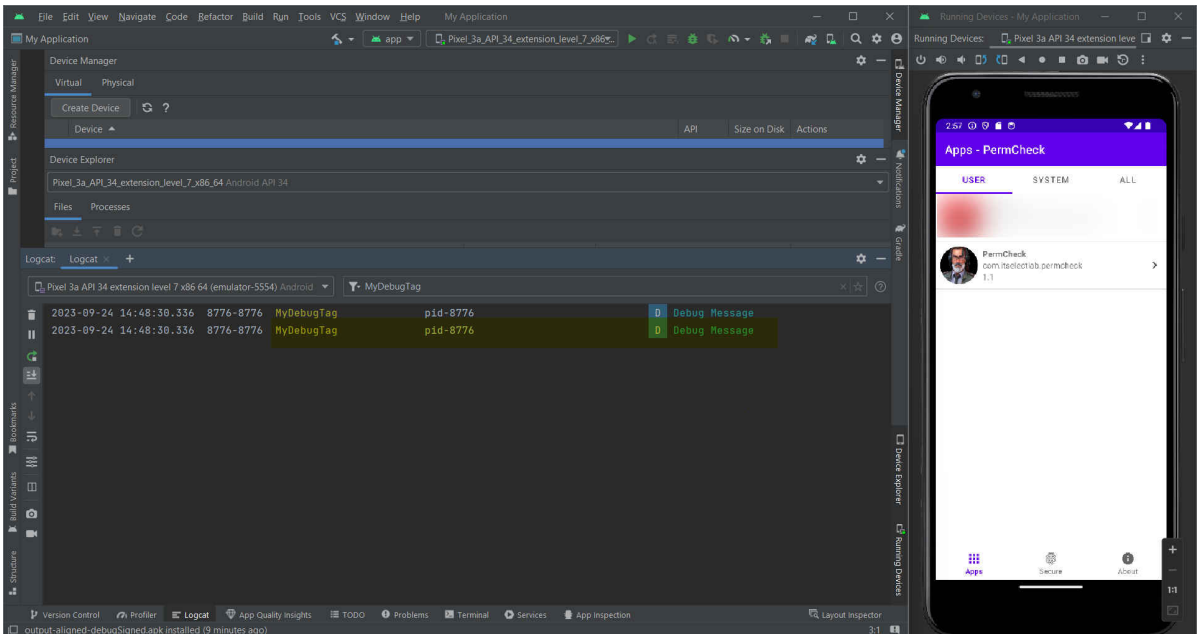
- ทดลองใส่แทรกบรรทัดแต่สิ่งที่ต้องระมัดระวังคือค่า `.locals 8` และ `const-string v6` และ `const-string v7`

    1. `.locals 8` หมายถึงว่าเมทอดหรือบล็อกของโค้ดนั้นจะใช้ตัวแปรทั้งหมด 8 ตัว ในการเก็บข้อมูลต่าง ๆ ภายในบล็อก แต่เดิมคือ 6 เราพยายามสร้างใหม่อีก 2 รวมเป็น 8
    2. `const-string v6` และ `const-string v7` พยายามค้นหาใน Method ที่เรากำลังตรวจสอบไม่มีการใช้หมายเลขดังกล่าวเพื่อไม่ให้ซ้อนทับกันหรือผิดเพี้ยนไปเป็นผลทำให้ Application เปิดใช้งานไม่ได้

```
# virtual methods
.method public final onCreate(Landroid/os/Bundle;)V
    .locals 8

    invoke-super {p0, p1}, Landroidx/fragment/app/e0;-
>onCreate(Landroid/os/Bundle;)V

    invoke-virtual {p0}, Landroid/app/Activity;-
>getLayoutInflater()Landroid/view/LayoutInflater;

    move-result-object p1

    const v0, 0x7f0b0020

    const-string v6, "MyDebugTag"

    const-string v7, "Debug Message"

    invoke-static {v6, v7}, Landroid/util/Log;-
>d(Ljava/lang/String;Ljava/lang/String;)I

    const/4 v1, 0x0
```

5. ทำการ Build ติดตั้งและทดสอบ

- เมื่อแก้ไขเสร็จสิ้นให้ทำการ Build กลับเป็น APK เหมือนเดิมโดยใช้ apktool และ Signed App ให้เรียบร้อย เพื่อให้สามารถติดตั้งบนเครื่อง Android ได้ จากนั้นตรวจสอบ App ที่ Log Cat ว่ามีข้อมูลตามที่เขียนไว้หรือไม่โดยใช้ Android Studio



6. อ้างอิง (References)

- https://apktool.org/
- https://github.com/patrickfav/uber-apk-signer/releases
- https://blog.itselectlab.com/?p=14387
- https://developer.android.com/guide/components/activities/activity-lifecycle
- https://blog.itselectlab.com/?p=15476