# Getting Started with SMALI for Penetration Testing

## บทนำ (Overview)

การทดสอบความปลอดภัยของแอปพลิเคชัน Android เป็นกระบวนการสำคัญที่ช่วยให้นักทดสอบความปลอดภัยค้นพบข้อบกพร่องและปัญหาเกี่ยวกับความปลอดภัยในแอปพลิเคชัน มีข้อผิดพลาดหรือช่องโหว่ที่อาจเปิดโอกาสให้ผู้ไม่หวังดีโจมตีแอปพลิเคชันหรือเข้าถึงข้อมูลส่วนบุคคลของผู้ใช้ การทดสอบความปลอดภัยช่วยให้พัฒนาแอปพลิเคชันอย่างมั่นคงและปลอดภัยต่อผู้ใช้เป็นอย่างมาก

หนึ่งในเครื่องมือที่นักทดสอบความปลอดภัยสามารถใช้ในการวิเคราะห์และทดสอบแอปพลิเคชัน Android คือ SMALI ภาษานี้ช่วยให้นักทดสอบความปลอดภัยสามารถอ่านและแก้ไขโค้ดของแอปพลิเคชัน Android อย่างละเอียด แม้ว่าโค้ดแอปพลิเคชันจะถูกคอมไพล์ในรูปแบบของไฟล์ DEX ซึ่งไม่ใช่ภาษา Java หรือ Kotlin แต่ SMALI ช่วยให้นักทดสอบความปลอดภัยสามารถวิเคราะห์โค้ดและค้นหาช่องโหว่หรือปัญหาความปลอดภัยได้อย่างมีประสิทธิภาพ

Warunyou Sunpachit และ Boonperm Mark
Cybersecurity consultant

## 1. อะไรคือ SMALI (What is SMALI ?)

SMALI ย่อมาจาก "Simple Machine Assembly Language Interpreter" เป็นภาษาโปรแกรมระดับต่ำ (low-level programming language) ที่ใช้ในการแสดงรหัสไบนารีของแอป Android (bytecode) ภายในไฟล์ `.dex` โดยมีโครงสร้างที่ซับซ้อนและรหัสที่ยากต่อการอ่านและเข้าใจในรูปแบบข้อความ (assembly-like) สามารถใช้เครื่องมือ Text Editor ทั่ว ๆ ไปเพื่อแก้ไขและปรับแต่งรหัส SMALI ได้ง่าย ๆ โดย SMALI ช่วยให้สามารถระบุข้อผิดพลาดหรือช่องโหว่และประเมินความปลอดภัยของแอปพลิเคชัน Android ผ่านกระบวนการวิเคราะห์ย้อนกลับ (Reverse Engineering)

# 2. ไฟล์ .dex อยู่ที่ไหน (Where is .dex ?)

ไฟล์ .dex (Dalvik Executable) คือไฟล์ที่มีรหัสไบนารีของแอปพลิเคชัน Android และมีหน้าที่เก็บโค้ดและข้อมูลที่ใช้ในการทำงานของแอปนั้น ๆ ไฟล์ .dex นั้นสำคัญสำหรับการรันแอปบนระบบ Android โดยทั่วไป ไฟล์ .dex สามารถพบได้ในโครงสร้างของแอป Android แต่มันอาจมีได้หลายไฟล์ .dex ขึ้นอยู่กับขนาดและโครงสร้างของแอป. ปกติแล้วไฟล์ .dex จะอยู่ในภายในไฟล์ APK ของแอปชื่อ `classes.dex` เมื่อใช้ ApkTool JADX ไฟล์ .dex จะถูกถอดรหัสและแปลงเป็นรหัส หรือ SMALI หรือ Java ตามลำดับ และบันทึกในโฟลเดอร์ที่สร้างขึ้นในกระบวนการถอดรหัสแอป โดยทั่วไปโครงสร้างไฟล์ มีลักษณะดังต่อไปนี้

```
myapp.apk
|
|-- META-INF/
|   |-- CERT.RSA
|   |-- CERT.SF
|   |-- MANIFEST.MF
|
|-- assets/
|
|-- res/
|
|-- AndroidManifest.xml
|
|-- classes.dex
|
|-- lib/
|   |-- armeabi-v7a/
|   |   |-- libmylibrary.so
|   |
|   |-- x86/
|       |-- libmylibrary.so
|
|-- resources.arsc
```

# 3. หลักการ (Principles)

## 3.1 การถอดรหัส APK (Decompiling the APK)

การถอดรหัส APK (Decompiling the APK) เป็นกระบวนการที่ใช้เครื่องมือ ApkTool เพื่อแยกไฟล์ APK ออกเป็นซอร์สโค้ดของแอปพลิเคชัน เป็นขั้นตอนที่จะช่วยให้สามารถดูและแก้ไขรหัสของแอปพลิเคชันได้

1. **ดาวน์โหลดและติดตั้ง ApkTool**: จำเป็นต้องดาวน์โหลดและติดตั้ง ApkTool ลงในคอมพิวเตอร์ของก่อนที่จะเริ่มกระบวนการถอดรหัส APK

   https://apktool.org/docs/install

2. **ใช้ ApkTool ถอดรหัส APK**: เมื่อมี ApkTool ติดตั้งแล้ว ให้เปิดหน้าต่างเทอร์มินัล (Command Prompt หรือ Terminal) และใช้คำสั่ง ApkTool เพื่อถอดรหัส APK ในคำสั่งนี้ ต้องแทนที่ "ชื่อไฟล์.apk" ด้วยชื่อของไฟล์ APK ที่ต้องการถอดรหัส เมื่อคำสั่งถูกป้อนและเริ่มการทำงาน ApkTool จะเริ่มแยกไฟล์ APK ออกเป็นซอร์สโค้ด.

   ```
   apktool d ชื่อไฟล์.apk
   ```

3. **สร้างไดเรกทอรีและรหัส SMALI**: ขั้นตอนนี้ ApkTool จะสร้างไดเรกทอรีใหม่ที่มีทรัพยากรของแอปและรหัส SMALI ของแอป SMALI เป็นภาษาที่ใช้ในการเขียนรหัสของแอป Android สามารถแก้ไขรหัส SMALI เพื่อปรับแต่งแอปตามที่ต้องการ.

## 3.2 ทำความเข้าใจกับรหัส SMALI (Understanding SMALI Code)

1. **คลาสและเมท็อด (Class และ Method)**: ใน SMALI คลาสเป็นกลุ่มของรหัสที่รวมกันเพื่อทำงานบางอย่างในแอปของ เราสามารถคิดเสมือนว่าคลาสเป็นกล่องที่มีรหัสภายใน และภายในคลาสนั้นจะมีเมท็อด (Method) ซึ่งเป็นกลุ่มของคำสั่งที่ทำงานเฉพาะหน้าที่หนึ่ง เช่น เปิดแอปหรือทำงานกับข้อมูล.
2. **คำสั่ง (Instructions)**: รหัส SMALI ประกอบด้วยคำสั่งต่าง ๆ ที่ใช้ในการทำงาน บางคำสั่งเป็นคำสั่งเบื้องต้น เช่น "move" ที่ใช้ในการย้ายข้อมูล และบางคำสั่งเป็นคำสั่งเชิงขั้นสูงเช่น "invoke-virtual" ที่ใช้ในการเรียกเมท็อด.
3. **ที่อยู่ (Registers)**: SMALI ใช้ที่อยู่เพื่อเก็บข้อมูลและส่งข้อมูลระหว่างคำสั่ง แบบเทียบเคียงได้กับตัวแปรในภาษาโปรแกรมอื่น ๆ ที่ใช้ในการจัดเก็บข้อมูลชั่วคราว
4. **การส่งค่าและการคืนค่า (Arguments and Return Values)**: เมื่อเราเรียกใช้เมท็อดหรือฟังก์ชัน แสดงด้วยคำสั่ง "invoke-xxx" เราจะส่งอาร์กิวเมนต์ (arguments) เข้าไปในเมท็อด และมันอาจคืนค่า (return value) กลับมาให้เราเมื่อเสร็จสิ้น.
5. **โครงสร้างควบคุม (Control Structures)**: SMALI รองรับโครงสร้างควบคุมเชิงพื้นฐาน เช่น การเงื่อนไข (if-else), การวนซ้ำ (loops), และการกระโดดไปยังเมท็อดอื่น (jumps).
6. **การอ้างอิงอ็อบเจ็กต์และฟิลด์ (Object References and Fields)**: แอป Android มักจะมีการใช้อ็อบเจ็กต์เพื่อจัดการข้อมูล และ SMALI รองรับการอ้างอิงอ็อบเจ็กต์และฟิลด์ของอ็อบเจ็กต์นั้น.
7. **การเรียกใช้คลาสอื่น (Class Invocation)**: แอป Android สามารถเรียกใช้คลาสอื่น ๆ เพื่อทำงานร่วมกัน และใน SMALI, เราใช้คำสั่ง "new-instance" เพื่อสร้างอ็อบเจ็กต์ของคลาสใหม่และ "invoke-direct" เพื่อเรียกใช้เมท็อดในคลาสนั้น.

## 3.3 วิเคราะห์รหัส SMALI (Analyzing SMALI Code)

การวิเคราะห์รหัส SMALI เป็นกระบวนการที่ทำให้เราสามารถเข้าใจโค้ดของแอป Android และค้นหาข้อมูลที่น่าสนใจหรือความช่องโหว่ได้ ต่อไปนี้คือขั้นตอนและเนื้อหาสำคัญที่สามารถช่วยในการวิเคราะห์รหัส SMALI:

1. **ค้นหาและอ่านเมท็อด (Methods)**: เริ่มจากการค้นหาเมท็อดที่น่าสนใจภายในคลาสที่กำลังวิเคราะห์ ใน SMALI เมท็อดจะถูกเรียกด้วยคำสั่ง "invoke-xxx" ดังนั้นค้นหาคำสั่งเหล่านี้เพื่อหาเมท็อดที่น่าสนใจ
2. **ตรวจสอบการจัดการข้อมูล (Data Handling)**: สังเกตว่าโค้ด SMALI มีการจัดการข้อมูลอย่างไร และตรวจสอบว่าข้อมูลสำคัญถูกเข้ารหัสหรือถูกเข้ารหัสอย่างไร เช่น การเก็บข้อมูลส่วนตัวหรือรหัสผ่าน
3. **ค้นหาการตรวจสอบสิทธิ์ (Permissions Check)**: หากแอปมีการใช้สิทธิ์เพื่อเข้าถึงฟังก์ชันหรือข้อมูลบางอย่าง ค้นหาโค้ดที่เกี่ยวข้องกับการตรวจสอบสิทธิ์นี้ เช่น การตรวจสอบการอนุญาตในการเข้าถึงกล้องหรือตำแหน่ง
4. **ตรวจสอบช่องโหว่ด้านความปลอดภัย (Security Vulnerabilities)**: ค้นหาโค้ดที่อาจทำให้เกิดความปลอดภัยเสี่ยง เช่น การรับข้อมูลจากผู้ใช้โดยไม่ตรวจสอบ หรือการใช้ข้อมูลที่ไม่ปลอดภัย
5. **ใช้เครื่องมืออื่น ๆ ช่วย**: มีเครื่องมือพิเศษที่ออกแบบมาเพื่อช่วยในการวิเคราะห์ความปลอดภัย โดยเฉพาะเครื่องมือสำหรับค้นหาช่องโหว่ด้านความปลอดภัยและการตรวจสอบสิทธิ์ ตัวอย่างเช่น `Mobile Security Framework` หรือ `QARK (Quick Android Review Kit)`

## 3.4 ทำการเปลี่ยนแปลงคำสั่ง (Making  Changes)

การแก้ไขโค้ด SMALI ที่ต้องการเปลี่ยนแปลง สามารถใช้ Text Editor เพื่อเปิดและแก้ไขไฟล์ SMALI ได้เหมือนกับการแก้ไขข้อความปกติ โดยการ เปลี่ยนแปลงโค้ดตามที่ต้องการ เช่น เพิ่มฟังก์ชันใหม่หรือแก้ไขฟังก์ชันที่มีอยู่แล้ว ตัวอย่างการแก้ไขรหัส SMALI เช่น

1. เปิดไฟล์ SMALI: ใช้ Text Editor เพื่อเปิดไฟล์ SMALI ที่ต้องการแก้ไข เช่น `MainActivity.smali`.
2. ค้นหาการแสดงข้อความ: ค้นหาโค้ด SMALI ที่เกี่ยวข้องกับการแสดงข้อความ "Hello, Android!" โดยอาจจะมีคำสั่งที่เรียกใช้ `invoke-static` เพื่อแสดงข้อความ. เช่น:

```
invoke-static {p0}, Landroid/widget/Toast;-
>makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/
Toast;
```

3. แก้ไขข้อความ: เปลี่ยนข้อความที่อยู่ในพารามิเตอร์ของ `makeText` เป็นข้อความใหม่ที่ต้องการแสดง เช่น:

```
const-string v0, "สวัสดี, Android!"
invoke-static {p0, v0, v1}, Landroid/widget/Toast;-
>makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/
Toast;
```

## 3.5 Recompiling the APK (การคอมไพล์ APK ใหม่)

หลังจากที่ทำการเปลี่ยนแปลงแล้ว ก็ถึงเวลาคอมไพล์ APK ใหม่โดยใช้ ApkTool

1. **สร้าง APK ใหม่ (Recompile)**: เมื่อทำการแก้ไขรหัสเสร็จสิ้น สามารถใช้ ApkTool ในการสร้าง APK ใหม่จากซอร์สโค้ดที่แก้ไขแล้ว โดยใช้คำสั่ง ในคำสั่งนี้ ต้องแทนที่ "ชื่อไดเรกทอรี" ด้วยชื่อของไดเรกทอรีที่มีซอร์สโค้ดและการแก้ไขของ.

```
apktool b ชื่อไดเรกทอรี
```

2. **รับไฟล์ APK ที่สร้างใหม่**: หลังจากที่สร้าง APK ใหม่เรียบร้อย สามารถหาไฟล์ APK ใหม่นี้ในไดเรกทอรี "dist" ที่ถูกสร้างขึ้นโดย ApkTool.

## 3.6 Testing and Validating (ติดตั้งและทดสอบ APK)

ขั้นตอน "ติดตั้งและทดสอบ APK" เป็นขั้นตอนที่สำคัญในกระบวนการการแก้ไขแอปพลิเคชัน Android และการตรวจสอบว่าการเปลี่ยนแปลงที่ทำทำงานตามที่คาดหวัง

1. **ติดตั้ง APK ที่แก้ไข**: เมื่อได้คอมไพล์แอปพลิเคชัน Android ใหม่หลังจากที่แก้ไขรหัส SMALI สามารถติดตั้งไฟล์ APK ที่ได้รับการดัดแปลงล่าสุดบนอุปกรณ์ Android หรือโปรแกรมจำลอง Android เช่น Android Emulator.
2. **ทดสอบแอปพลิเคชัน**: เปิดแอปพลิเคชันบนอุปกรณ์ Android หรือโปรแกรมจำลอง และทดสอบฟังก์ชันและความปลอดภัยของมัน ตรวจสอบว่าการแก้ไขที่ทำทำงานถูกต้องหรือไม่ และว่าไม่มีข้อผิดพลาดใด ๆ เกิดขึ้น.

# 4. คำสั่งพื้นฐานของ SMALI (Basic SMALI Instructions)

## 4.1 ตัวแปลและค่าคงที่

- **const**: ใช้ในการกำหนดค่าคงที่และตัวแปร

```
const v0, 0x7f040000
```

`v0` คือชื่อของตัวแปร

`0x7f040000` คือค่าคงที่หรือค่าตัวแปรที่ถูกกำหนดในรหัส SMALI โดยใช้รูปแบบเลขฐาน 16 (Hexadecimal)

สำหรับตัวอย่างนี้ `v0` ถูกกำหนดค่าเป็น `0x7f040000` ซึ่งหมายถึงว่า `v0` จะเก็บค่า `0x7f040000` ในการทำงานของโค้ดที่มีการอ้างอิงถึง `v0` ในภายหลัง ตัวแปร `v0` นี้อาจถูกใช้ในการส่งค่าไปยังเมทอดอื่น ๆ หรือในการดำเนินการคำนวณต่าง ๆ ในรหัส SMALI ของแอปพลิเคชัน Android

- คำสั่ง `const` ใน SMALI ใช้ในการกำหนดค่าคงที่ (constant) และส่วนมากจะใช้ในการกำหนดค่าที่เป็นตัวเลข (integer) หรือค่าที่เป็นอินสแตนซ์ (instance) ของคลาส ไม่สามารถใช้ในการกำหนดค่าเป็นสตริง (string) หรือชนิดข้อมูลอื่น ๆ ได้โดยตรงด้วยคำสั่ง `const` เช่น:

```
const v0, 123        ; กำหนดค่าคงที่เป็น integer 123 ให้กับ v0
```

- ถ้าต้องการกำหนดค่าเป็นสตริงใน SMALI จะใช้คำสั่ง `const-string`

```
const-string v0, "Hello, Android!"   ; กำหนดค่าคงที่เป็นสตริง "Hello, Android!" ให้กับ v0
```

## 4.2 การย้ายค่าจากหนึ่งรีจิสเตอร์ไปยังรีจิสเตอร์อีกอันหนึ่ง (Move)

- คำสั่ง `move` ใช้เพื่อย้ายค่าจากรีจิสเตอร์หนึ่งไปยังรีจิสเตอร์อีกอันหนึ่ง ซึ่งค่าที่จะถูกย้ายนั้นอาจเป็นค่าจากการคำนวณหรือจากการโหลดค่าจากหน่วยความจำ คำสั่ง `move` จะมีรูปแบบเช่น:
  1. `destination_register`: รีจิสเตอร์ที่ค่าจะถูกย้ายไปยัง
  2. `source_register`: รีจิสเตอร์ที่ค่าจะถูกย้ายมาจาก

```
move destination_register, source_register
```

- ตัวอย่างการใช้คำสั่ง `move`:

`move v0, v1` หมายถึงการย้ายค่าที่อยู่ในรีจิสเตอร์ `v1` ไปยังรีจิสเตอร์ `v0` นั่นหมายความว่าค่าที่เคยอยู่ใน `v1` จะถูกคัดลอกและเก็บไว้ใน `v0` โดยที่ `v1` ไม่มีค่าแล้วหลังจากคำสั่ง `move` ถูกดำเนินการ.

```
move v0, v1
```

## 4.3 การดำเนินการทางคณิตศาสตร์ (Mathematical Operations)

- "Arithmetic Operations" ซึ่งเป็นกระบวนการหรือการกระทำที่เกี่ยวข้องกับการคำนวณค่าตัวเลขโดยใช้ตัวดำเนินการทางคณิตศาสตร์ เช่น การบวก (+), การลบ (-), การคูณ (*), และการหาร (/) เพื่อสร้างผลลัพธ์ทางคณิตศาสตร์ในรูปของค่าตัวเลข เช่น add , sub , mul , และ div
- **คำสั่ง `add` (Addition)**: คำสั่งนี้ใช้ในการบวกค่าสองค่าเข้าด้วยกัน แล้วเก็บผลลัพธ์ในรีจิสเตอร์ที่ระบุ ตัวอย่าง:

```
add v0, v1, v2  ; v0 = v1 + v2
```

- **คำสั่ง `sub` (Subtraction)**: คำสั่งนี้ใช้ในการลบค่าสองค่าจากกัน แล้วเก็บผลลัพธ์ในรีจิสเตอร์ที่ระบุ

```
sub v0, v1, v2  ; v0 = v1 - v2
```

- **คำสั่ง `mul` (Multiplication)**: คำสั่งนี้ใช้ในการคูณค่าสองค่าเข้าด้วยกัน แล้วเก็บผลลัพธ์ในรีจิสเตอร์ที่ระบุ ตัวอย่าง:

```
mul v0, v1, v2  ; v0 = v1 * v2
```

- **คำสั่ง `div` (Division)**: คำสั่งนี้ใช้ในการหารค่าในรีจิสเตอร์หนึ่งด้วยค่าในรีจิสเตอร์อีกอันหนึ่ง แล้วเก็บผลลัพธ์ในรีจิสเตอร์ที่ระบุ ตัวอย่าง:

```
div v0, v1, v2  ; v0 = v1 / v2
```

## 4.4 การกระโดดไปคำสั่งบรรทัดอื่น ๆ โดยใช้ เงื่อนไข (IF condition)

- คำสั่ง `if` ใช้สำหรับควบคุมการกระโดดไปยังโค้ดที่ต่างกันขึ้นอยู่กับเงื่อนไขที่กำหนด โครงสร้างพื้นฐานของ `if` ใน Smali คือ:

```
if-<เงื่อนไข> :cond_0
```

- `<เงื่อนไข>` คือ เงื่อนไขที่ต้องการตรวจสอบ เช่น `eqz` (เท่ากับศูนย์), `nez` (ไม่เท่ากับศูนย์), `lt` (น้อยกว่า), `ge` (มากกว่าหรือเท่ากับ), เป็นต้น.
- `:cond_0` คือ ป้ายกำกับ (label) ที่จะใช้ในการระบุส่วนของโค้ดที่ต้องกระโดดไป.
- ตัวอย่างการใช้คำสั่ง `if`

```
const v0, 5
if-eqz v0, :cond_0

const-string v1, "เงื่อนไขเป็นจริง"
invoke-static {v1}, Landroid/util/Log;->i(Ljava/lang/String;)I

:cond_0
const-string v2, "เสร็จสิ้น"
invoke-static {v2}, Landroid/util/Log;->i(Ljava/lang/String;)I
```

1. `const v0, 5` กำหนดค่า 5 ในที่เก็บ `v0`.

2. `if-eqz v0, :cond_0` ตรวจสอบว่า `v0` เป็นศูนย์หรือไม่ ถ้าเงื่อนไขเป็นจริง (v0 เป็น 0) จะกระโดดไปยังป้ายกำกับ `:cond_0`.
3. ถ้าเงื่อนไขเป็นเท็จ (v0 ไม่เป็น 0) คำสั่งในบล็อกที่ตามหลัง `if` จะถูกข้ามไปเลย และโปรแกรมจะทำงานต่อที่คำสั่ง `const-string v2, "เสร็จสิ้น"`
4. ดังนั้น, ในกรณีนี้ถ้า `v0` มีค่าเป็น 0 เราจะเห็นการล็อก "เงื่อนไขเป็นจริง" บน Logcat แต่ถ้า `v0` ไม่เป็น 0 เราจะเห็นการล็อก "เสร็จสิ้น" แทน.

## 4.5 ความคิดเห็น (Comments)

- ความคิดเห็นนี้จะไม่มีผลต่อการทำงานของโค้ดและจะถูกข้ามไปเมื่อ Smali ถูกคอมไพล์เป็น bytecode. ในตัวอย่างนี้, การใช้ความคิดเห็นช่วยให้ผู้อ่านโค้ดเข้าใจว่าแต่ละส่วนของโค้ดทำงานอย่างไรและสามารถอธิบายหรือเน้นข้อมูลสำคัญในโค้ดได้อย่างชัดเจน.

```
# นี่คือความคิดเห็นเกี่ยวกับการกำหนดค่าตัวแปร
const v0, 5

# นี่คือความคิดเห็นเกี่ยวกับการตรวจสอบเงื่อนไข
if-eqz v0, :cond_0

# นี่คือความคิดเห็นเกี่ยวกับการล็อกข้อมูล
const-string v1, "เงื่อนไขเป็นจริง"
invoke-static {v1}, Landroid/util/Log;->i(Ljava/lang/String;)I

:cond_0
# นี่คือความคิดเห็นเกี่ยวกับการล็อกข้อมูลเมื่อเงื่อนไขเป็นเท็จ
const-string v2, "เสร็จสิ้น"
invoke-static {v2}, Landroid/util/Log;->i(Ljava/lang/String;)I
```

## 4.6 การนิยามเมธอด (Method Definitions)

- `.method`: เริ่มการนิยามเมธอดด้วย `.method` โดยระบุชื่อของเมธอด และรายละเอียดอื่น ๆ เช่น การส่งพารามิเตอร์และประเภทของเมธอด (public, private, static, เป็นต้น). ตัวอย่าง:

```
.method public static myMethod(IILjava/lang/String;)Ljava/lang/String;
```

- `.registers`: ระบุจำนวนของที่เก็บ (registers) ที่ใช้ในเมธอด. เป็นขั้นตอนที่จำเป็นเพื่อกำหนดจำนวนที่เก็บที่จะใช้ในการเก็บค่าตัวแปรและคำสั่งอื่น ๆ ในเมธอด. ตัวอย่าง:

```
registers 4
```

- ชุดคำสั่งในเมธอดจะตามหลัง `.registers` คำสั่งเหล่านี้อาจเป็นการประมวลผลข้อมูลหรือการเรียกใช้เมธอดอื่น ๆ ในคลาสหรือคลาสอื่น ๆ. ตัวอย่าง:

```
const v0, 0x7f040000
invoke-virtual {p0, v0}, Landroid/widget/TextView;->setText(I)V
```

- `.end method`: สิ้นสุดการนิยามเมธอด แสดงว่าเนื้อหาของเมธอดได้สิ้นสุดลง. ตัวอย่าง:

```
.end method
```

## 4.7 ส่งคืนค่าจากเมธอด (Return)

- คำสั่ง `return` ในภาษา Smali ใช้สำหรับส่งคืนค่าจากเมธอด (method) และออกจากเมธอดนั้นๆ โดยค่าที่ส่งคืนจะอยู่ในรีจิสเตอร์ที่ระบุในคำสั่ง `return` และประเภทของค่าที่ส่งคืนจะตรงกับประเภทที่ระบุในการนิยามของเมธอด (method signature) ในบางครั้งค่าที่ส่งคืนจะเป็นค่าคงที่ (constant) หรือค่าจากการดำเนินการทางคณิตศาสตร์ที่ได้รับในโค้ดของเมธอดนั้น ๆ
- ตัวอย่างการใช้คำสั่ง `return` ใน Smali เพื่อส่งคืนค่าจากเมธอดที่มีชื่อว่า `addTwoIntegers` ซึ่งรับพารามิเตอร์สองจำนวนเต็มและส่งคืนผลลัพธ์เป็นจำนวนเต็มดังนี้:

```
.method public static addTwoIntegers(II)I
    .locals 2

    # Load parameters into registers
    .param p0, "a"
    .param p1, "b"

    # Add the values in v0 and v1
    add-int v0, p0, p1

    # Return the result
    return v0
.end method
```

- ในตัวอย่างนี้ เมธอด `addTwoIntegers` รับค่าพารามิเตอร์สองตัว (a และ b) และทำการบวกกัน ผลลัพธ์ถูกเก็บในรีจิสเตอร์ `v0` และสุดท้ายส่งคืนค่านี้โดยใช้คำสั่ง `return v0` ซึ่งหมายถึงการส่งคืนค่าจากเมธอดโดยใช้ค่าที่อยู่ในรีจิสเตอร์ `v0` เป็นค่าผลลัพธ์ของการบวก
- `addTwoIntegers(II)` ในที่นี้เป็นส่วนหนึ่งของการนิยามของเมธอด (method signature) ในภาษา Smali และมีความหมายดังนี้:
    1. `addTwoIntegers`: นี่คือชื่อของเมธอด (method name) ซึ่งในที่นี้เมธอดชื่อว่า `addTwoIntegers``
    2. `(II)`: นี่คือส่วนของลายเซ็นเมธอด (method signature) ซึ่งระบุประเภทของพารามิเตอร์และประเภทของค่าที่เมธอดนี้จะส่งคืน:
        - `(II)`: มีวงเล็บเปิดและปิดรอบสองครั้ง แสดงถึงการรับพารามิเตอร์สองตัว
        - `I`: แสดงถึงประเภทของค่าที่เมธอดนี้จะส่งคืน ในที่นี้คือจำนวนเต็ม (integer)
- คำสั่ง `add-int` ในภาษา Smali ใช้สำหรับการดำเนินการบวกสองค่าจำนวนเต็มและเก็บผลลัพธ์ในรีจิสเตอร์ที่ระบุ โดยรีจิสเตอร์ที่จะเก็บผลลัพธ์จะถูกระบุในคำสั่ง `add-int` เอง
    1. `add-int`: เป็นคำสั่งบวกจำนวนเต็ม
    2. `v0`: คือรีจิสเตอร์ที่จะเก็บผลลัพธ์
    3. `v1`: คือรีจิสเตอร์ที่เก็บค่าจำนวนเต็มตัวแรกที่จะนำมาบวก
    4. `v2`: คือรีจิสเตอร์ที่เก็บค่าจำนวนเต็มตัวที่สองที่จะนำมาบวก

## 4.8 ตัวกำหนดการเข้าถึง (Access Modifiers)

- ตัวกำหนดการเข้าถึง (Access Modifiers) ใช้สำหรับกำหนดระดับการเข้าถึงหรือการเปิดเผยของสมาชิกในคลาส หรือเมธอดของคลาส เมื่อเรากำหนดการเข้าถึงแล้วจะระบุความเป็นส่วนตัวหรือเปิดเผยของสมาชิกนั้น ๆ ในแอปพลิเคชัน Android. ดังนี้คือตัวกำหนดการเข้าถึงที่ใช้งานได้ใน Smali:
    1. `public`: กำหนดให้สมาชิกหรือเมธอดเปิดเผยแก่ทุกคลาส และสามารถเข้าถึงจากทุกที่ในแอปพลิเคชัน Android.
    2. `private`: กำหนดให้สมาชิกหรือเมธอดเป็นส่วนตัวและสามารถเข้าถึงได้เฉพาะในคลาสที่สมาชิกหรือเมธอดนั้นถูกประกาศ.
    3. `protected`: กำหนดให้สมาชิกหรือเมธอดสามารถเข้าถึงได้ในคลาสที่ประกาศสมาชิกหรือเมธอดนี้ และในคลาสที่สืบทอดจากคลาสที่ประกาศ.
    4. `static`: กำหนดให้เมธอดเป็นสถิตและเรียกใช้โดยไม่ต้องสร้างอ็อบเจกต์ของคลาส.
    5. `final`: กำหนดให้เมธอดหรือสมาชิกเป็นค่าคงที่และไม่สามารถแก้ไขค่าได้.
- ตัวกำหนดการเข้าถึงจะถูกใช้ใน Smali โดยการเรียงตามความสำคัญจาก `private` ไปจนถึง `public`. ตัวกำหนดการเข้าถึงจะถูกกำหนดหลังชื่อเมธอดหรือสมาชิกเช่น:

```
.method public myMethod()V
```

## 4.9 การเรียกใช้งานเมทอด (Invoke Method)

- ใช้คำสั่ง `invoke-...` เพื่อเรียกใช้เมธอด คำสั่งที่แน่นอนขึ้นอยู่กับระดับการเข้าถึงของเมธอดและประเภทการคืนค่า เช่น:
    - `invoke-static`: เรียกใช้เมธอดแบบสแตติก
    - `invoke-virtual`: เรียกใช้เมธอดตัวอย่างบนอ็อบเจกต์
    - `invoke-direct`: เรียกใช้คอนสตรักเตอร์หรือเมธอดส่วนตัว (private)
    - `invoke-interface`: เรียกใช้เมธอดบนอินเทอร์เฟซ
- คำสั่ง `invoke-static` ใน Android SMALI เป็นคำสั่งที่ใช้ในการเรียกเมทอด (method) ภายในคลาส (class) โดยไม่ต้องสร้างอินสแตนซ์ (instance) ของคลาสนั้น ๆ ก่อน คือเรียกเมทอดที่เป็น static จากคลาสโดยตรง รูปแบบของคำสั่ง `invoke-static` ใน SMALI มักจะมีรูปแบบดังนี้:

```
invoke-static {parameter_list}, class_name->method_name(return_type)
```

    1. `parameter_list`: รายการพารามิเตอร์ที่ถูกส่งให้กับเมทอด ถ้าไม่มีพารามิเตอร์ จะเว้นว่างไว้ หรือจะใช้ `{}` เป็นสัญลักษณ์แทนรายการพารามิเตอร์ว่าง.
    2. `class_name`: ชื่อของคลาสที่มีเมทอดที่ต้องการเรียกใช้ (โดยใช้เครื่องหมาย `/` แทน `.` เพื่อแบ่งชื่อแพ็คเกจ และคลาส).
    3. `method_name`: ชื่อของเมทอดที่ต้องการเรียกใช้.
    4. return_type `: ประเภทของค่าที่เมทอดนี้จะส่งคืน (return) หากไม่มีการส่งคืน จะใช้ V แทน (void).`

    ในตัวอย่างนี้, เราใช้ `invoke-static` เพื่อเรียกเมทอด `addTwoNumbers` จากคลาส `MyClass` โดยส่งพารามิเตอร์ `v0` และ `v1` ไปให้เมทอด และเรากำหนดให้คืนค่าประเภท integer (`I`). `ตัวอย่างการใช้งาน` invoke-static` ใน SMALI:

```
invoke-static {v0, v1}, Lcom/example/MyClass;->addTwoNumbers(II)I
```

## 5. อุปกรณ์ที่จำเป็น (Getting Set Up)

ก่อนที่เราจะเริ่ม จะต้องมีเครื่องมือดังต่อไปนี้:

- Android Studio: สภาพแวดล้อมการพัฒนาแบบรวม (IDE) สำหรับการเขียนโปรแกรมบนเครื่อง Android
- Java Development Kit (JDK): ตรวจสอบให้แน่ใจว่าได้ติดตั้ง JDK ล่าสุด
- ApkTool: ApkTool เป็นสิ่งจำเป็นสำหรับการถอดรหัสและคอมไพล์ไฟล์ Android APK

## 6. อ้างอิง (References)


- https://github.com/skylot/jadx
- https://apktool.org/docs/install
- https://github.com/MobSF/Mobile-Security-Framework-MobSF
- https://github.com/linkedin/qark
- https://blog.itselectlab.com/?p=15476