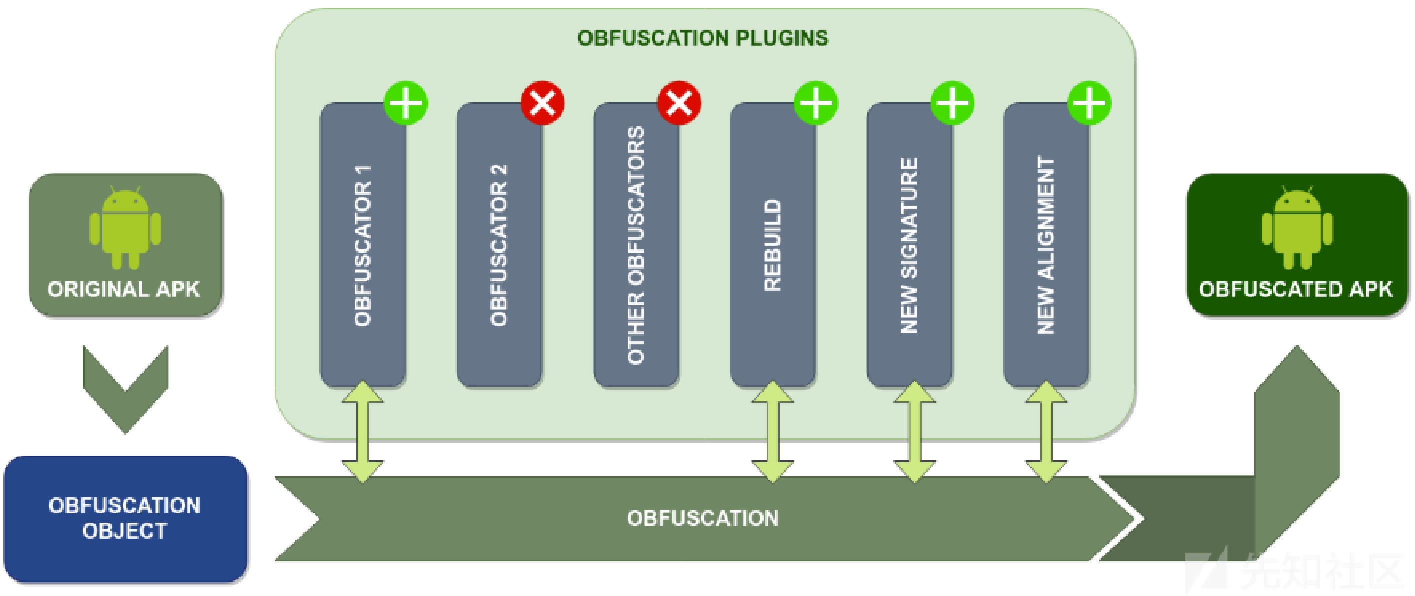
apk加固工具探究系列——Obfuscapk

houjingyi / 2019-10-01 09:19:11 / 浏览数 5249 安全技术 移动安全 顶(0) 踩(0)

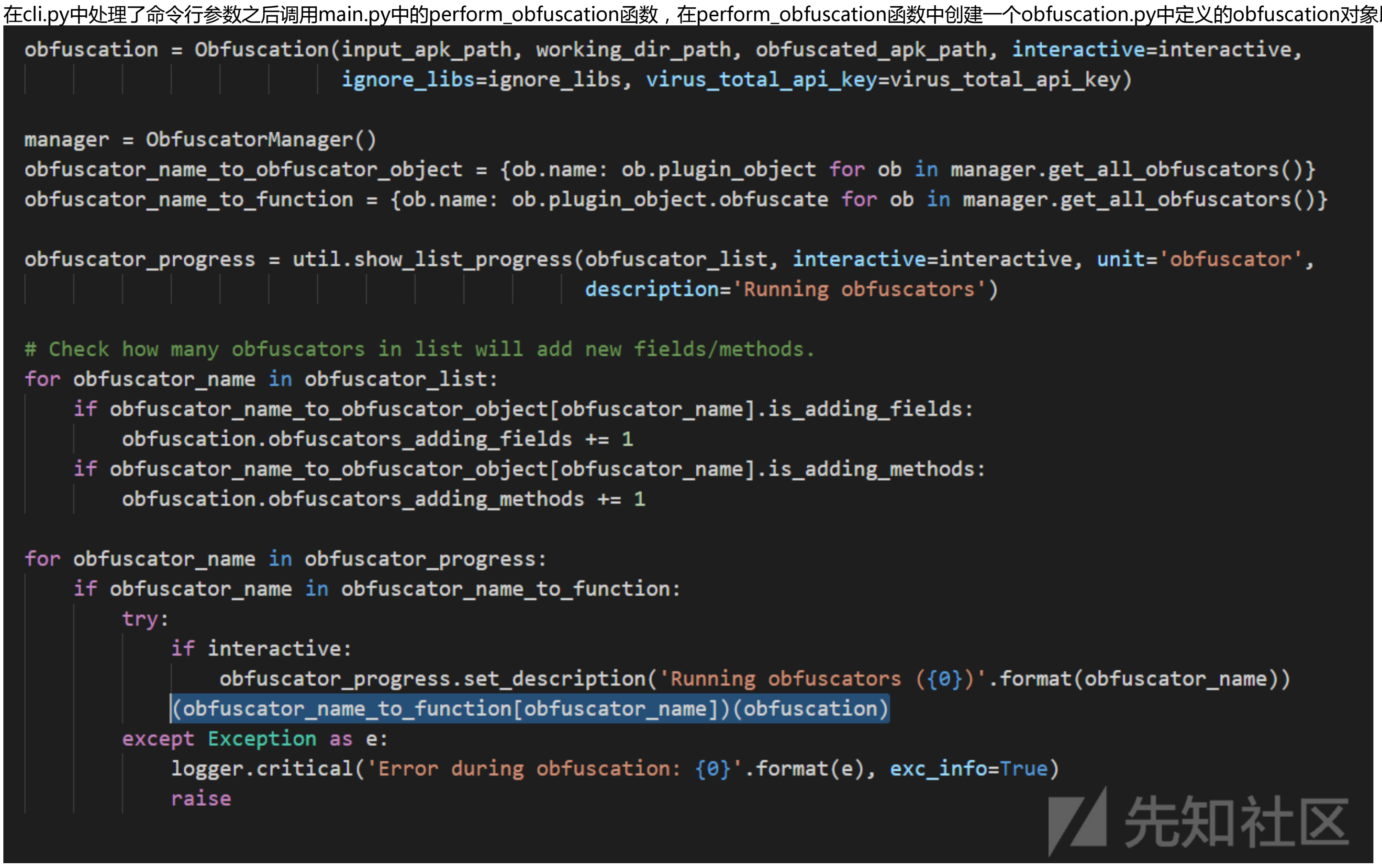
前言



Obfuscapk是一个python实现的apk混淆工具，使用插件系统构建，被设计为模块化且易于扩展。每个obfuscator都是一个从抽象基类(obfuscator_category.py)继承的插件
使用新的obfuscator对该工具进行扩展非常简单：在src/obfuscapk/obfuscators目录中添加实现混淆技术的源代码和插件元数据文件(obfuscator-name.obfuscator)即可。

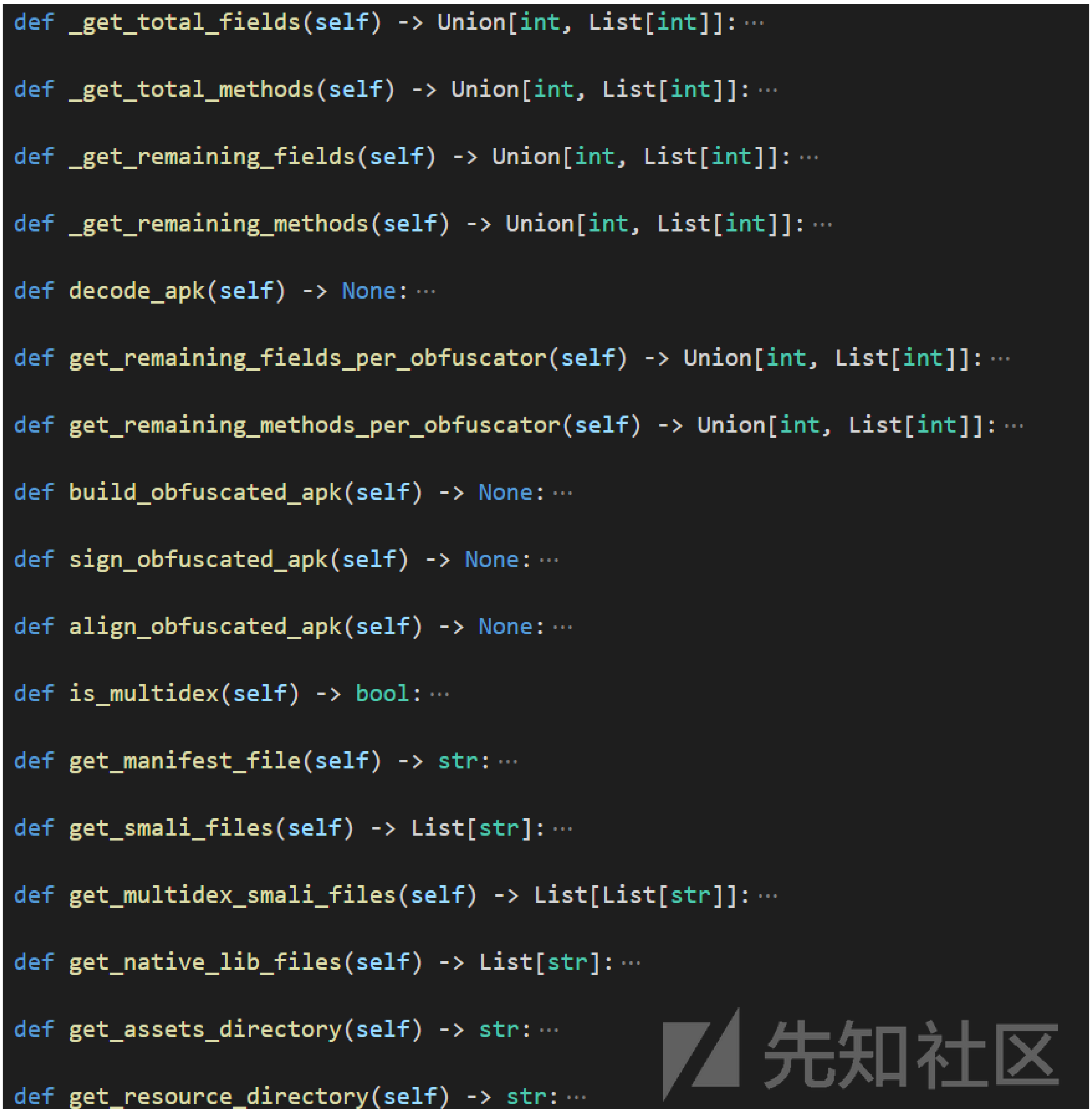
代码分析

在cli.py中处理了命令行参数之后调用main.py中的perform_obfuscation函数，在perform_obfuscation函数中创建一个obfuscation.py中定义的obfuscation对象以存储所



```python
obfuscation = Obfuscation(input_apk_path, working_dir_path, obfuscated_apk_path, interactive=interactive,
                          ignore_libs=ignore_libs, virus_total_api_key=virus_total_api_key)

manager = ObfuscatorManager()
obfuscator_name_to_obfuscator_object = {ob.name: ob.plugin_object for ob in manager.get_all_obfuscators()}
obfuscator_name_to_function = {ob.name: ob.plugin_object.obfuscate for ob in manager.get_all_obfuscators()}

obfuscator_progress = util.show_list_progress(obfuscator_list, interactive=interactive, unit='obfuscator',
                                              description='Running obfuscators')

# Check how many obfuscators in list will add new fields/methods.
for obfuscator_name in obfuscator_list:
    if obfuscator_name_to_obfuscator_object[obfuscator_name].is_adding_fields:
        obfuscation.obfuscators_adding_fields += 1
    if obfuscator_name_to_obfuscator_object[obfuscator_name].is_adding_methods:
        obfuscation.obfuscators_adding_methods += 1

for obfuscator_name in obfuscator_progress:
    if obfuscator_name in obfuscator_name_to_function:
        try:
            if interactive:
                obfuscator_progress.set_description('Running obfuscators ({0})'.format(obfuscator_name))
            (obfuscator_name_to_function[obfuscator_name])(obfuscation)
        except Exception as e:
            logger.critical('Error during obfuscation: {0}'.format(e), exc_info=True)
            raise
```

obfuscation对象中调用decode_apk函数，其中调用apktool对原始apk文件进行反编译，得到AndroidManifest.xml\resource文件\assets文件\so文件等等信息，对smali

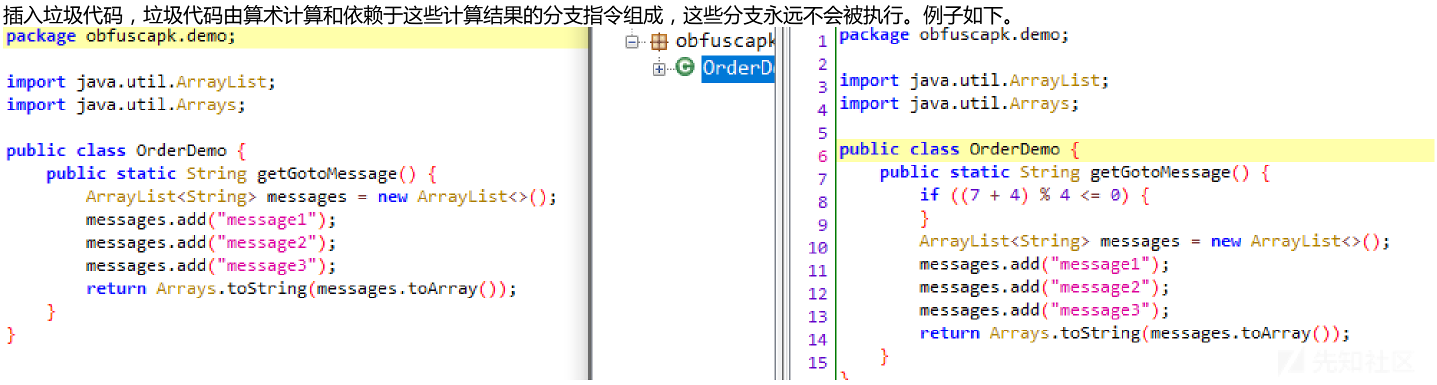


```python
def _get_total_fields(self) -> Union[int, List[int]]: ···

def _get_total_methods(self) -> Union[int, List[int]]: ···

def _get_remaining_fields(self) -> Union[int, List[int]]: ···

def _get_remaining_methods(self) -> Union[int, List[int]]: ···

def decode_apk(self) -> None: ···

def get_remaining_fields_per_obfuscator(self) -> Union[int, List[int]]: ···

def get_remaining_methods_per_obfuscator(self) -> Union[int, List[int]]: ···

def build_obfuscated_apk(self) -> None: ···

def sign_obfuscated_apk(self) -> None: ···

def align_obfuscated_apk(self) -> None: ···

def is_multidex(self) -> bool: ···

def get_manifest_file(self) -> str: ···

def get_smali_files(self) -> List[str]: ···

def get_multidex_smali_files(self) -> List[List[str]]: ···

def get_native_lib_files(self) -> List[str]: ···

def get_assets_directory(self) -> str: ···

def get_resource_directory(self) -> str: ···
```

接下来我们来看每个obfuscator的实现。NewAlignment，NewSignature，Rebuild分别用来重新对齐，重新签名和重新构建；VirusTotal用来将混淆前和混淆后的apk发

ArithmeticBranch

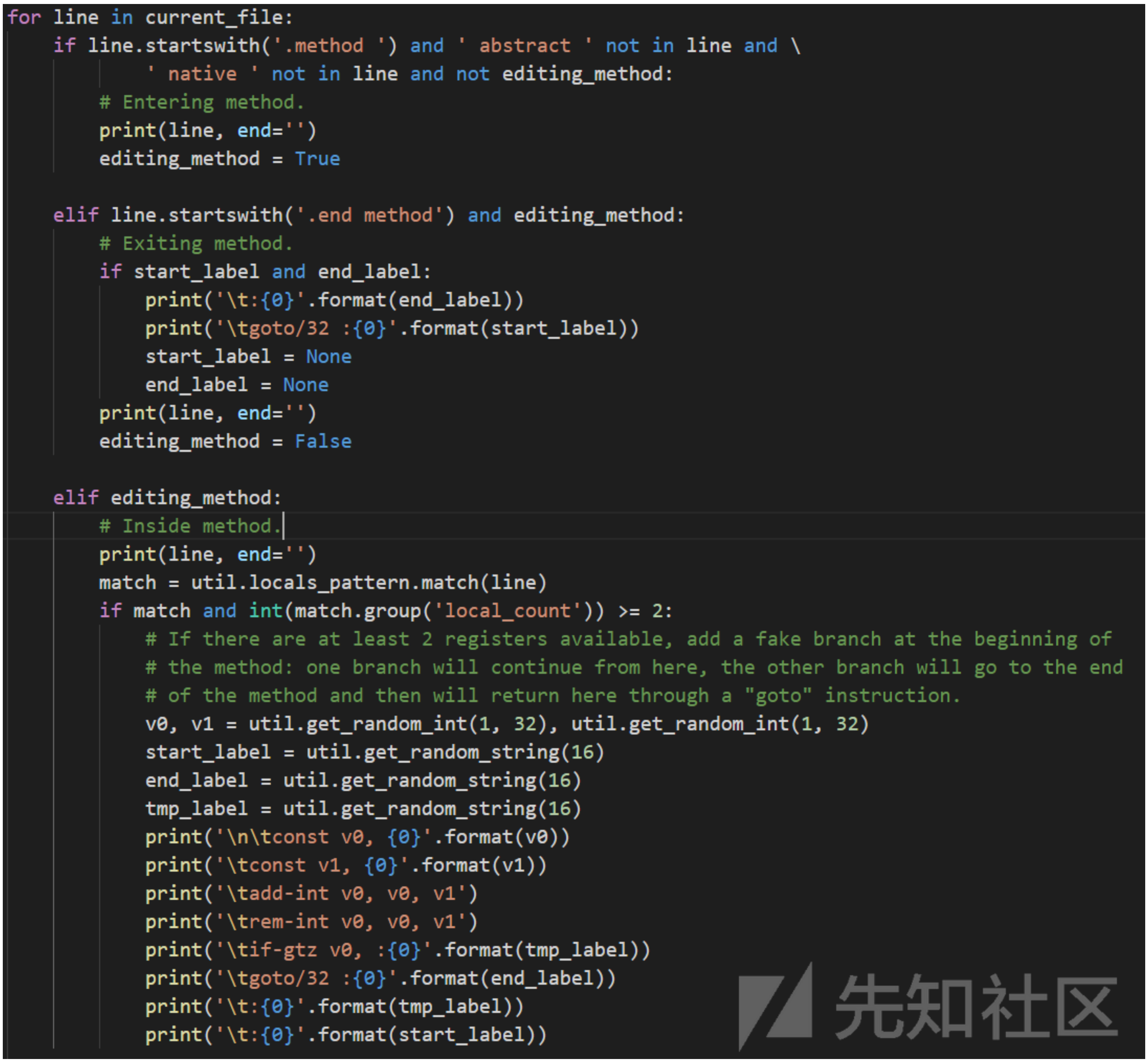
插入垃圾代码，垃圾代码由算术计算和依赖于这些计算结果的分支指令组成，这些分支永远不会被执行。例子如下。



```java
package obfuscapk.demo;

import java.util.ArrayList;
import java.util.Arrays;

public class OrderDemo {
    public static String getGotoMessage() {
        ArrayList<String> messages = new ArrayList<>();
        messages.add("message1");
        messages.add("message2");
        messages.add("message3");
        return Arrays.toString(messages.toArray());
    }
}
```

```java
package obfuscapk.demo;

import java.util.ArrayList;
import java.util.Arrays;

public class OrderDemo {
    public static String getGotoMessage() {
        if ((7 + 4) % 4 <= 0) {
        }
        ArrayList<String> messages = new ArrayList<>();
        messages.add("message1");
        messages.add("message2");
        messages.add("message3");
        return Arrays.toString(messages.toArray());
    }
```

如果一个方法使用了两个及以上的寄存器就添加一个形式如(a+b)%b的条件，如果大于等于0继续执行下面的代码，如果小于0(不会发生)跳到method的结尾，method结尾

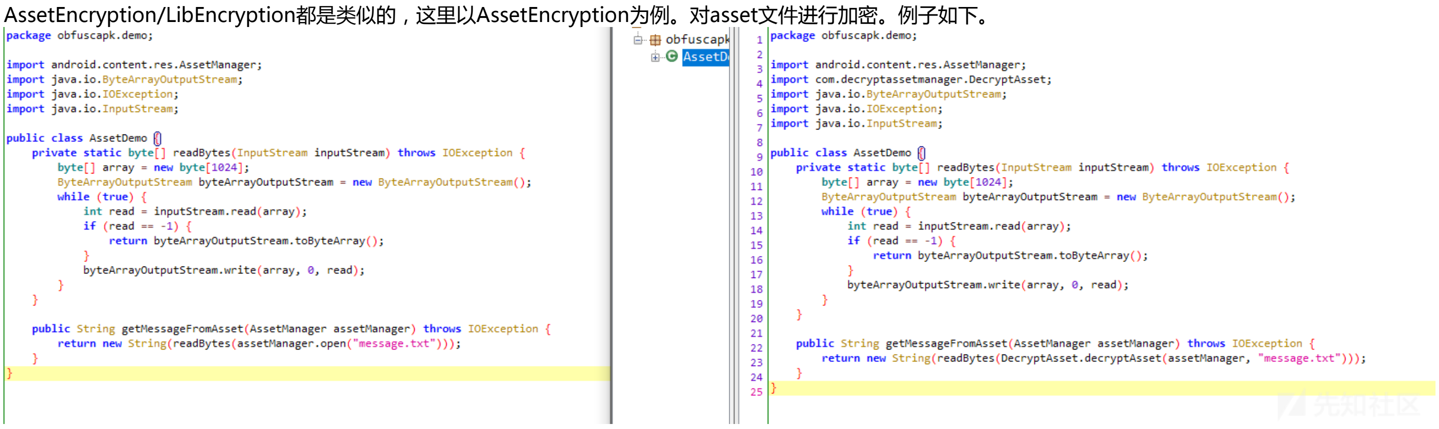


```python
for line in current_file:
    if line.startswith('.method ') and ' abstract ' not in line and \
            ' native ' not in line and not editing_method:
        # Entering method.
        print(line, end='')
        editing_method = True

    elif line.startswith('.end method') and editing_method:
        # Exiting method.
        if start_label and end_label:
            print('\t:{0}'.format(end_label))
            print('\tgoto/32 :{0}'.format(start_label))
            start_label = None
            end_label = None
        print(line, end='')
        editing_method = False

    elif editing_method:
        # Inside method.
        print(line, end='')
        match = util.locals_pattern.match(line)
        if match and int(match.group('local_count')) >= 2:
            # If there are at least 2 registers available, add a fake branch at the beginning of
            # the method: one branch will continue from here, the other branch will go to the end
            # of the method and then will return here through a "goto" instruction.
            v0, v1 = util.get_random_int(1, 32), util.get_random_int(1, 32)
            start_label = util.get_random_string(16)
            end_label = util.get_random_string(16)
            tmp_label = util.get_random_string(16)
            print('\n\tconst v0, {0}'.format(v0))
            print('\tconst v1, {0}'.format(v1))
            print('\tadd-int v0, v0, v1')
            print('\trem-int v0, v0, v1')
            print('\tif-gtz v0, :{0}'.format(tmp_label))
            print('\tgoto/32 :{0}'.format(end_label))
            print('\t:{0}'.format(tmp_label))
            print('\t:{0}'.format(start_label))
```

虽然看上去效果比较鸡肋，但是可以进一步做得更复杂。

AssetEncryption/LibEncryption

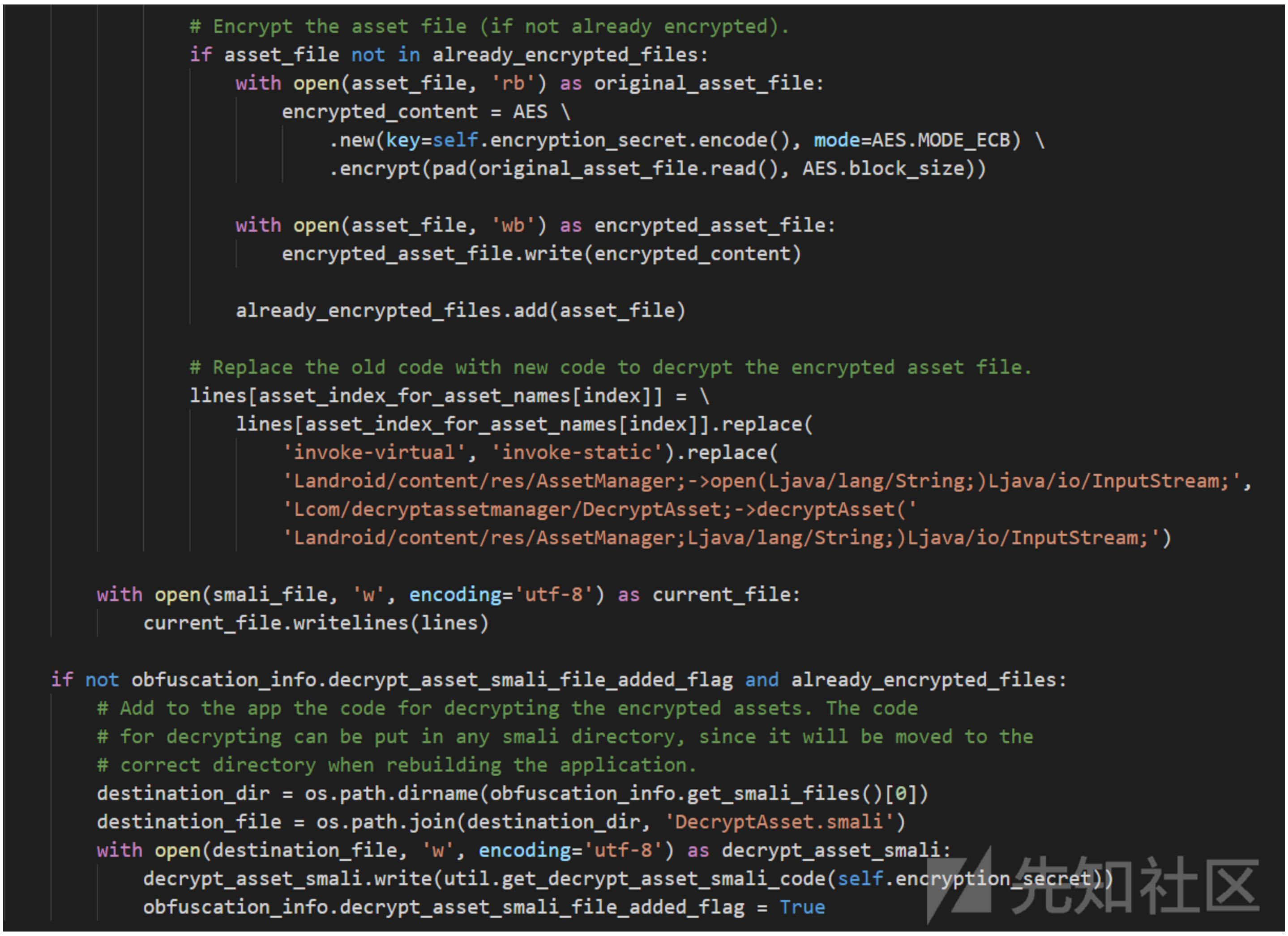
AssetEncryption/LibEncryption都是类似的，这里以AssetEncryption为例。对asset文件进行加密。例子如下。



```java
package obfuscapk.demo;

import android.content.res.AssetManager;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class AssetDemo {
    private static byte[] readBytes(InputStream inputStream) throws IOException {
        byte[] array = new byte[1024];
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        while (true) {
            int read = inputStream.read(array);
            if (read == -1) {
                return byteArrayOutputStream.toByteArray();
            }
            byteArrayOutputStream.write(array, 0, read);
        }
    }

    public String getMessageFromAsset(AssetManager assetManager) throws IOException {
        return new String(readBytes(assetManager.open("message.txt")));
    }
}
```

```java
package obfuscapk.demo;

import android.content.res.AssetManager;
import com.decryptassetmanager.DecryptAsset;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class AssetDemo {
    private static byte[] readBytes(InputStream inputStream) throws IOException {
        byte[] array = new byte[1024];
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        while (true) {
            int read = inputStream.read(array);
            if (read == -1) {
                return byteArrayOutputStream.toByteArray();
            }
            byteArrayOutputStream.write(array, 0, read);
        }
    }

    public String getMessageFromAsset(AssetManager assetManager) throws IOException {
        return new String(readBytes(DecryptAsset.decryptAsset(assetManager, "message.txt")));
    }
}
```

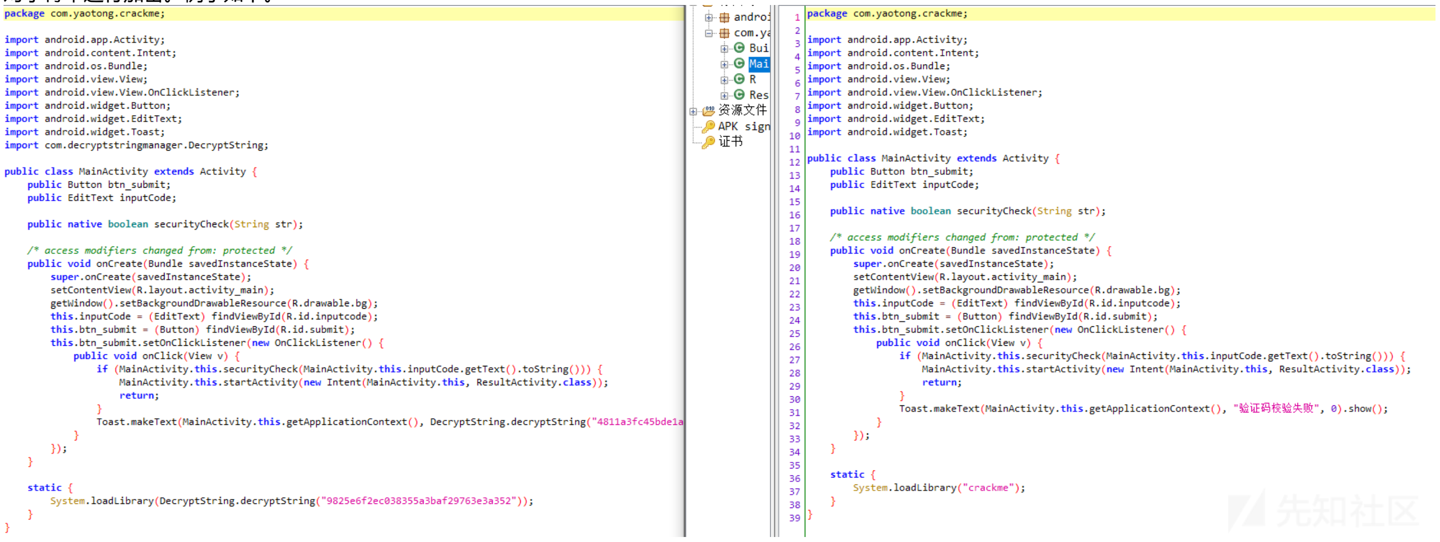
如果调用了assetManager.open函数打开asset文件就对asset文件进行AES加密，同时把assetManager.open函数替换成自己的解密函数，如果进行了加密并且没有添加存



```python
                # Encrypt the asset file (if not already encrypted).
                if asset_file not in already_encrypted_files:
                    with open(asset_file, 'rb') as original_asset_file:
                        encrypted_content = AES \
                            .new(key=self.encryption_secret.encode(), mode=AES.MODE_ECB) \
                            .encrypt(pad(original_asset_file.read(), AES.block_size))

                    with open(asset_file, 'wb') as encrypted_asset_file:
                        encrypted_asset_file.write(encrypted_content)

                    already_encrypted_files.add(asset_file)

                # Replace the old code with new code to decrypt the encrypted asset file.
                lines[asset_index_for_asset_names[index]] = \
                    lines[asset_index_for_asset_names[index]].replace(
                        'invoke-virtual', 'invoke-static').replace(
                        'Landroid/content/res/AssetManager;->open(Ljava/lang/String;)Ljava/io/InputStream;',
                        'Lcom/decryptassetmanager/DecryptAsset;->decryptAsset('
                        'Landroid/content/res/AssetManager;Ljava/lang/String;)Ljava/io/InputStream;')

        with open(smali_file, 'w', encoding='utf-8') as current_file:
            current_file.writelines(lines)

    if not obfuscation_info.decrypt_asset_smali_file_added_flag and already_encrypted_files:
        # Add to the app the code for decrypting the encrypted assets. The code
        # for decrypting can be put in any smali directory, since it will be moved to the
        # correct directory when rebuilding the application.
        destination_dir = os.path.dirname(obfuscation_info.get_smali_files()[0])
        destination_file = os.path.join(destination_dir, 'DecryptAsset.smali')
        with open(destination_file, 'w', encoding='utf-8') as decrypt_asset_smali:
            decrypt_asset_smali.write(util.get_decrypt_asset_smali_code(self.encryption_secret))
            obfuscation_info.decrypt_asset_smali_file_added_flag = True
```

ConstStringEncryption

对字符串进行加密。例子如下。

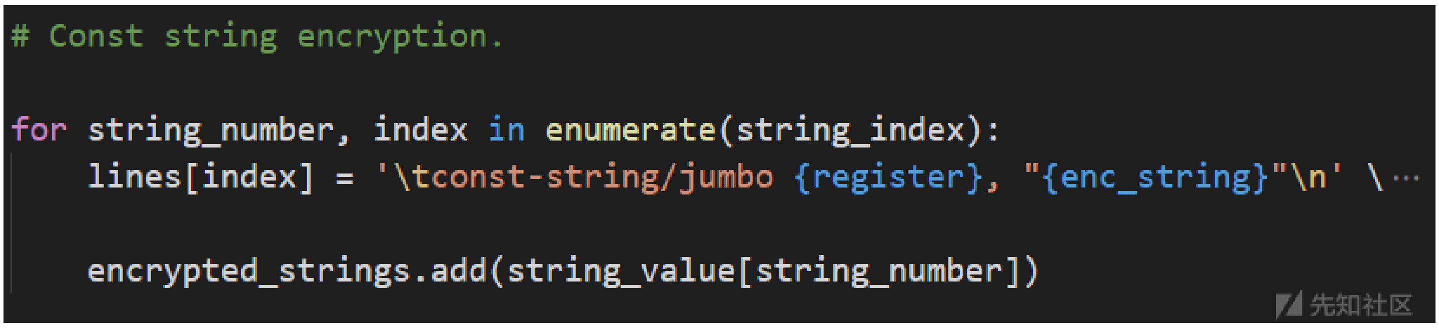


```java
package com.yaotong.crackme;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import com.decryptstringmanager.DecryptString;

public class MainActivity extends Activity {
    public Button btn_submit;
    public EditText inputCode;

    public native boolean securityCheck(String str);

    /* access modifiers changed from: protected */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getWindow().setBackgroundDrawableResource(R.drawable.bg);
        this.inputCode = (EditText) findViewById(R.id.inputcode);
        this.btn_submit = (Button) findViewById(R.id.submit);
        this.btn_submit.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                if (MainActivity.this.securityCheck(MainActivity.this.inputCode.getText().toString())) {
                    MainActivity.this.startActivity(new Intent(MainActivity.this, ResultActivity.class));
                    return;
                }
                Toast.makeText(MainActivity.this.getApplicationContext(), DecryptString.decryptString("4811a3fc45bde1a
            }
        });
    }

    static {
        System.loadLibrary(DecryptString.decryptString("9825e6f2ec038355a3baf29763e3a352"));
    }
}
```

```java
package com.yaotong.crackme;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
    public Button btn_submit;
    public EditText inputCode;

    public native boolean securityCheck(String str);

    /* access modifiers changed from: protected */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getWindow().setBackgroundDrawableResource(R.drawable.bg);
        this.inputCode = (EditText) findViewById(R.id.inputcode);
        this.btn_submit = (Button) findViewById(R.id.submit);
        this.btn_submit.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                if (MainActivity.this.securityCheck(MainActivity.this.inputCode.getText().toString())) {
                    MainActivity.this.startActivity(new Intent(MainActivity.this, ResultActivity.class));
                    return;
                }
                Toast.makeText(MainActivity.this.getApplicationContext(), "验证码校验失败", 0).show();
            }
        });
    }

    static {
        System.loadLibrary("crackme");
    }
}
```

将`const-string register, plaintext`中的`plaintext`加密成`ciphertext`，然后将其替换成下面三行代码(接下来的代码中[]中为变量名)。

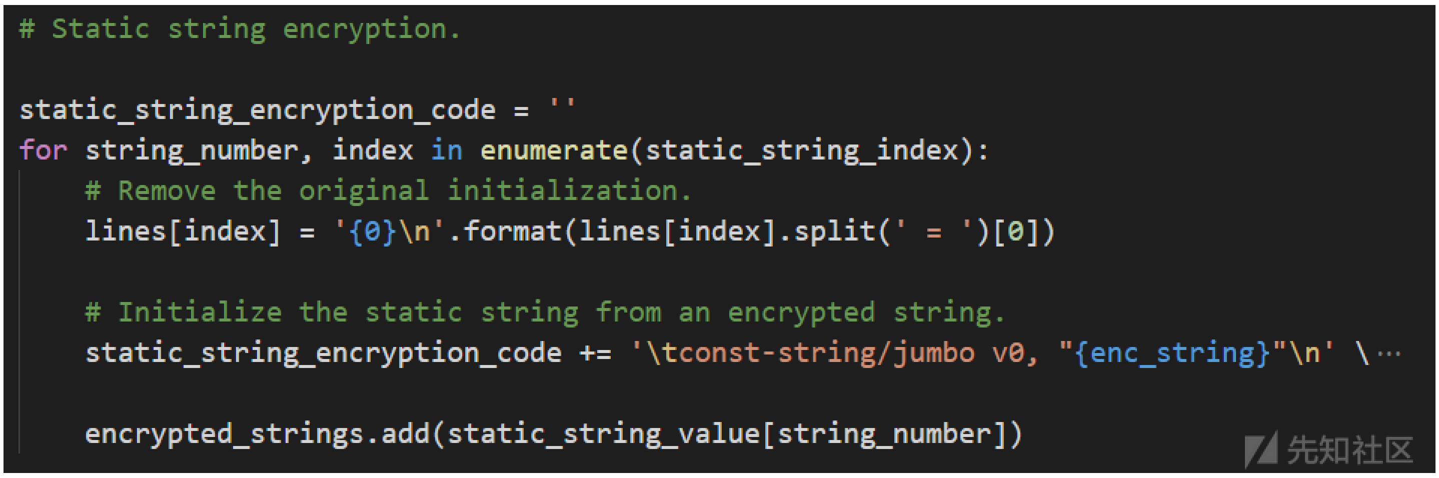
```
const-string/jumbo [register], [ciphertext]
invoke-static {[register]}, Lcom/decryptstringmanager/DecryptString;->decryptString(Ljava/lang/String;)Ljava/lang/String;
move-result-object [register]
```



```python
# Const string encryption.

for string_number, index in enumerate(string_index):
    lines[index] = '\tconst-string/jumbo {register}, "{enc_string}"\n' \…

    encrypted_strings.add(string_value[string_number])
```

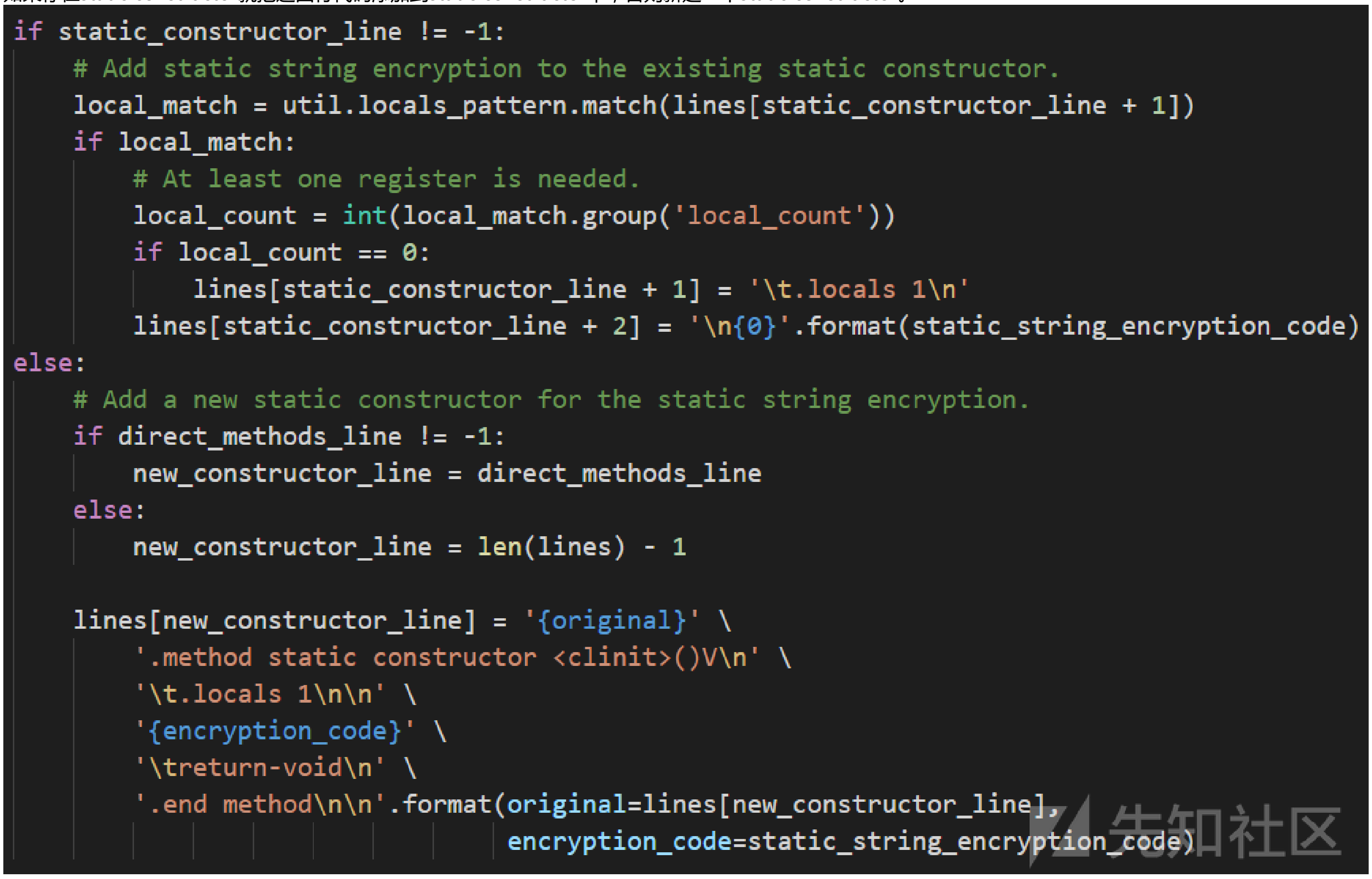
将.field (optional) static (optional) string_name:Ljava/lang/String; = plaintext中的plaintext加密成ciphertext，将其替换成.field (optional) static (optional) string_name:Ljava/lang/String;，然后增加下面四行代码。

```
const-string/jumbo v0, [ciphertext]
invoke-static {v0}, Lcom/decryptstringmanager/DecryptString;->decryptString(Ljava/lang/String;)Ljava/lang/String;
move-result-object v0
sput-object v0, [class_name]->[string_name]:Ljava/lang/String;
```



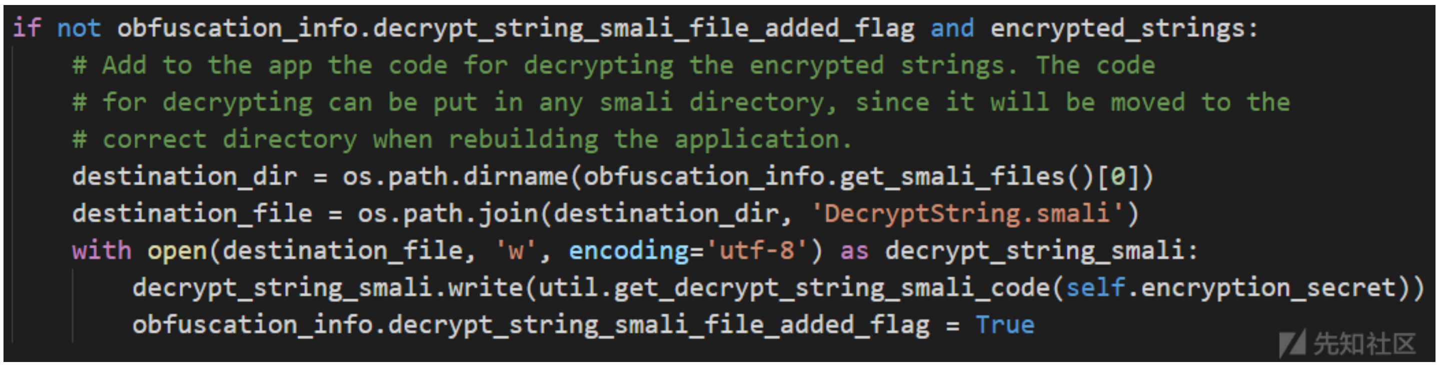
```python
# Static string encryption.

static_string_encryption_code = ''
for string_number, index in enumerate(static_string_index):
    # Remove the original initialization.
    lines[index] = '{0}\n'.format(lines[index].split(' = ')[0])

    # Initialize the static string from an encrypted string.
    static_string_encryption_code += '\tconst-string/jumbo v0, "{enc_string}"\n' \…

    encrypted_strings.add(static_string_value[string_number])
```

如果存在static constructor就把这四行代码添加到static constructor中，否则新建一个static constructor。



```python
if static_constructor_line != -1:
    # Add static string encryption to the existing static constructor.
    local_match = util.locals_pattern.match(lines[static_constructor_line + 1])
    if local_match:
        # At least one register is needed.
        local_count = int(local_match.group('local_count'))
        if local_count == 0:
            lines[static_constructor_line + 1] = '\t.locals 1\n'
        lines[static_constructor_line + 2] = '\n{0}'.format(static_string_encryption_code)
else:
    # Add a new static constructor for the static string encryption.
    if direct_methods_line != -1:
        new_constructor_line = direct_methods_line
    else:
        new_constructor_line = len(lines) - 1

    lines[new_constructor_line] = '{original}' \
        '.method static constructor <clinit>()V\n' \
        '\t.locals 1\n\n' \
        '{encryption_code}' \
        '\treturn-void\n' \
        '.end method\n\n'.format(original=lines[new_constructor_line],
                                 encryption_code=static_string_encryption_code)
```

同样如果进行了加密并且没有添加存在解密函数的smali文件就添加。

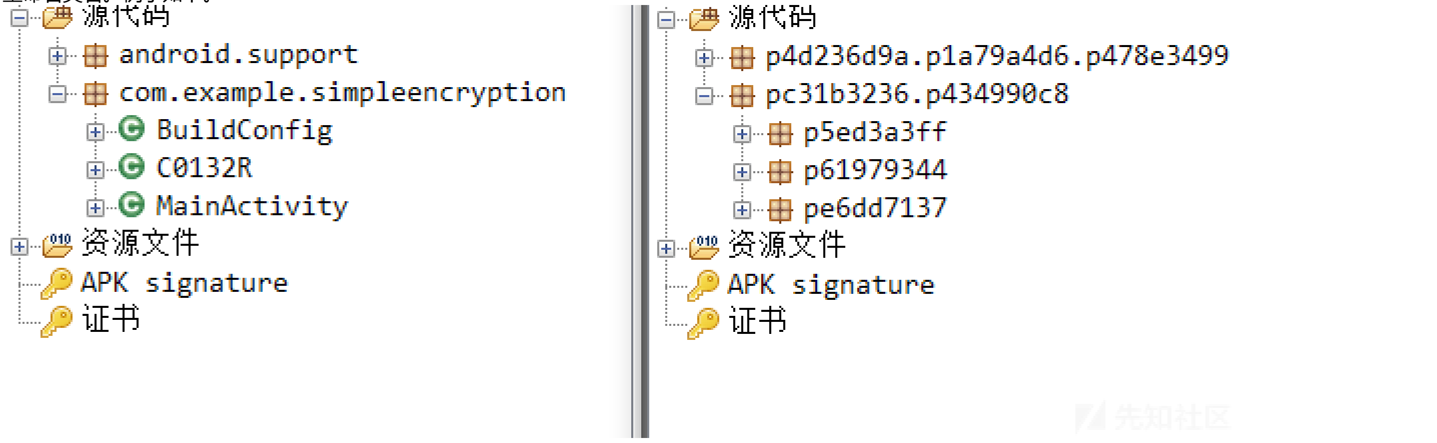


```python
if not obfuscation_info.decrypt_string_smali_file_added_flag and encrypted_strings:
    # Add to the app the code for decrypting the encrypted strings. The code
    # for decrypting can be put in any smali directory, since it will be moved to the
    # correct directory when rebuilding the application.
    destination_dir = os.path.dirname(obfuscation_info.get_smali_files()[0])
    destination_file = os.path.join(destination_dir, 'DecryptString.smali')
    with open(destination_file, 'w', encoding='utf-8') as decrypt_string_smali:
        decrypt_string_smali.write(util.get_decrypt_string_smali_code(self.encryption_secret))
        obfuscation_info.decrypt_string_smali_file_added_flag = True
```

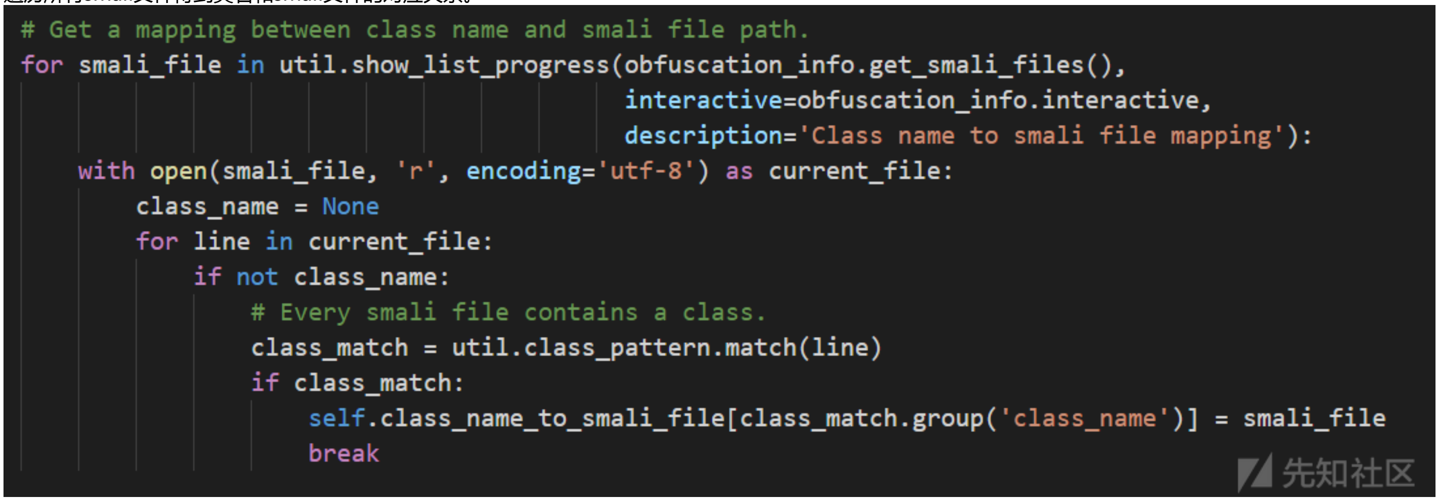
类似的ResStringEncryption可以对资源文件中的字符串加密，这里就不再分析了。

ClassRename

重命名类名。例子如下。

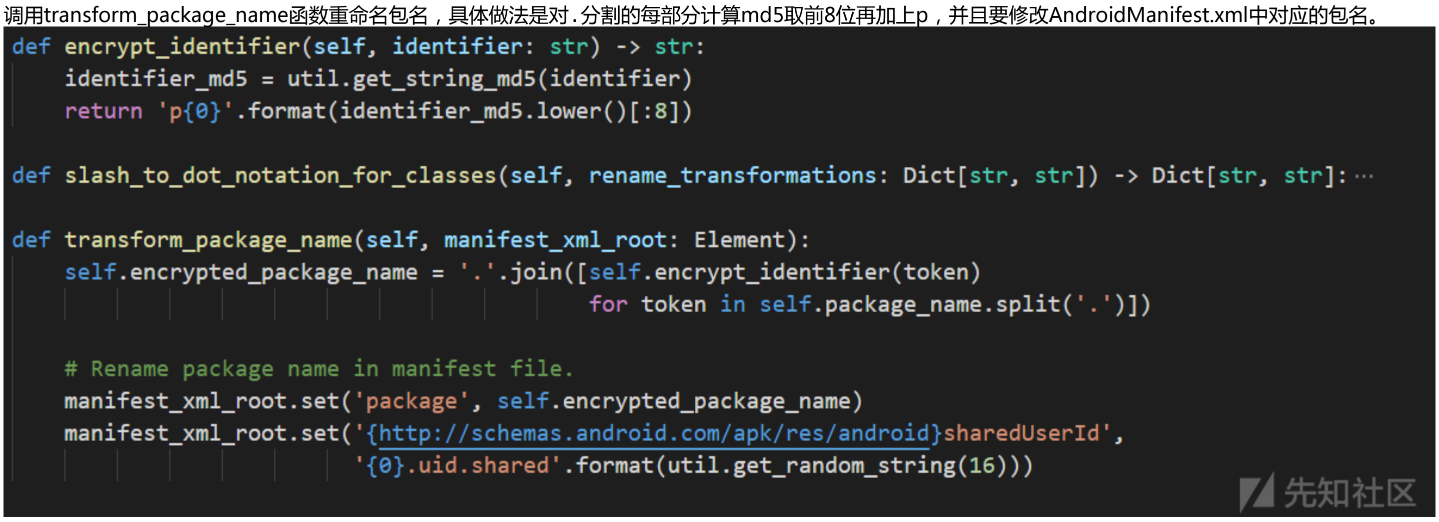


遍历所有smali文件得到类名和smali文件的对应关系。



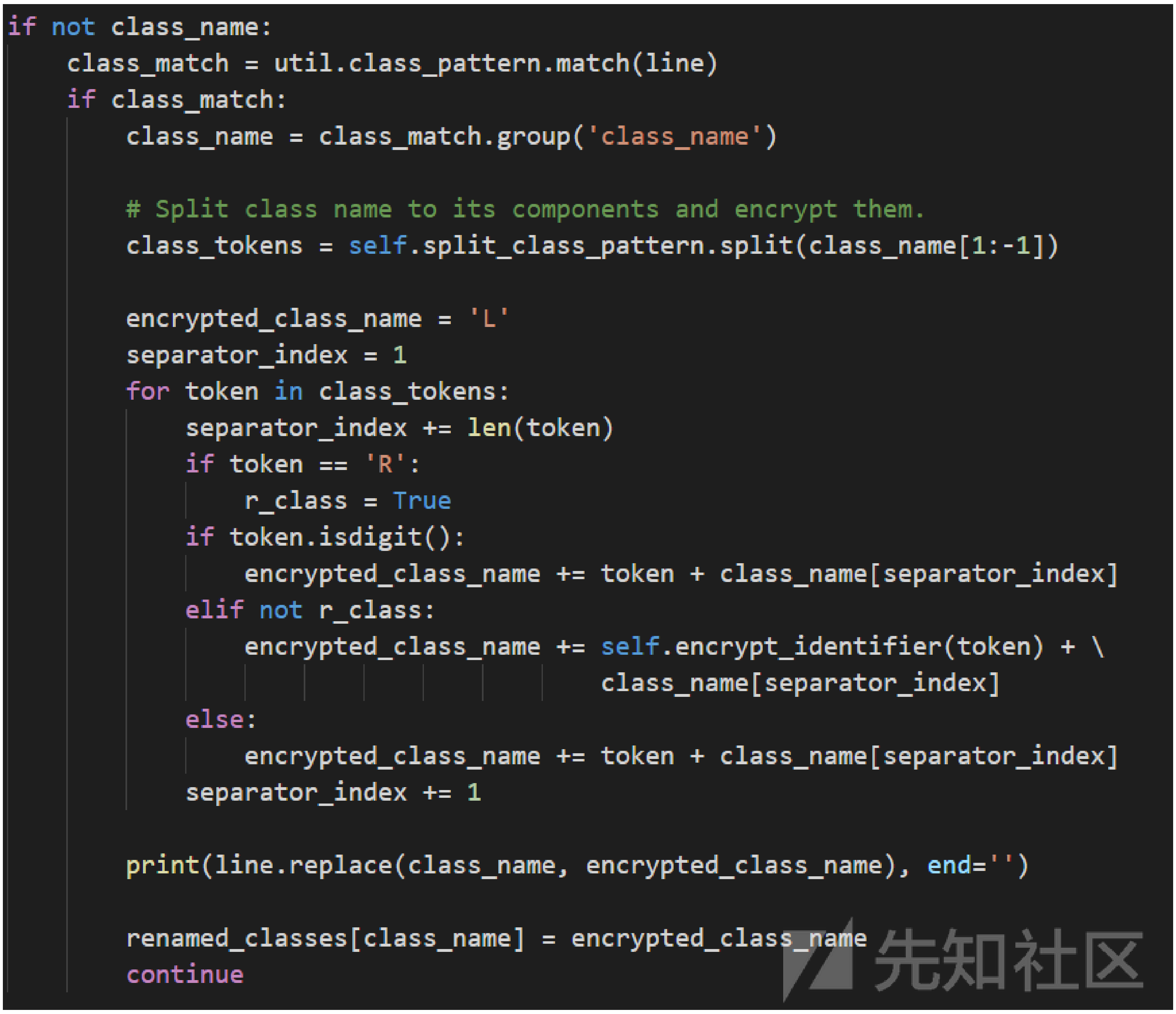
```python
# Get a mapping between class name and smali file path.
for smali_file in util.show_list_progress(obfuscation_info.get_smali_files(),
                                          interactive=obfuscation_info.interactive,
                                          description='Class name to smali file mapping'):
    with open(smali_file, 'r', encoding='utf-8') as current_file:
        class_name = None
        for line in current_file:
            if not class_name:
                # Every smali file contains a class.
                class_match = util.class_pattern.match(line)
                if class_match:
                    self.class_name_to_smali_file[class_match.group('class_name')] = smali_file
                    break
```

调用transform_package_name函数重命名包名，具体做法是对.分割的每部分计算md5取前8位再加上p，并且要修改AndroidManifest.xml中对应的包名。



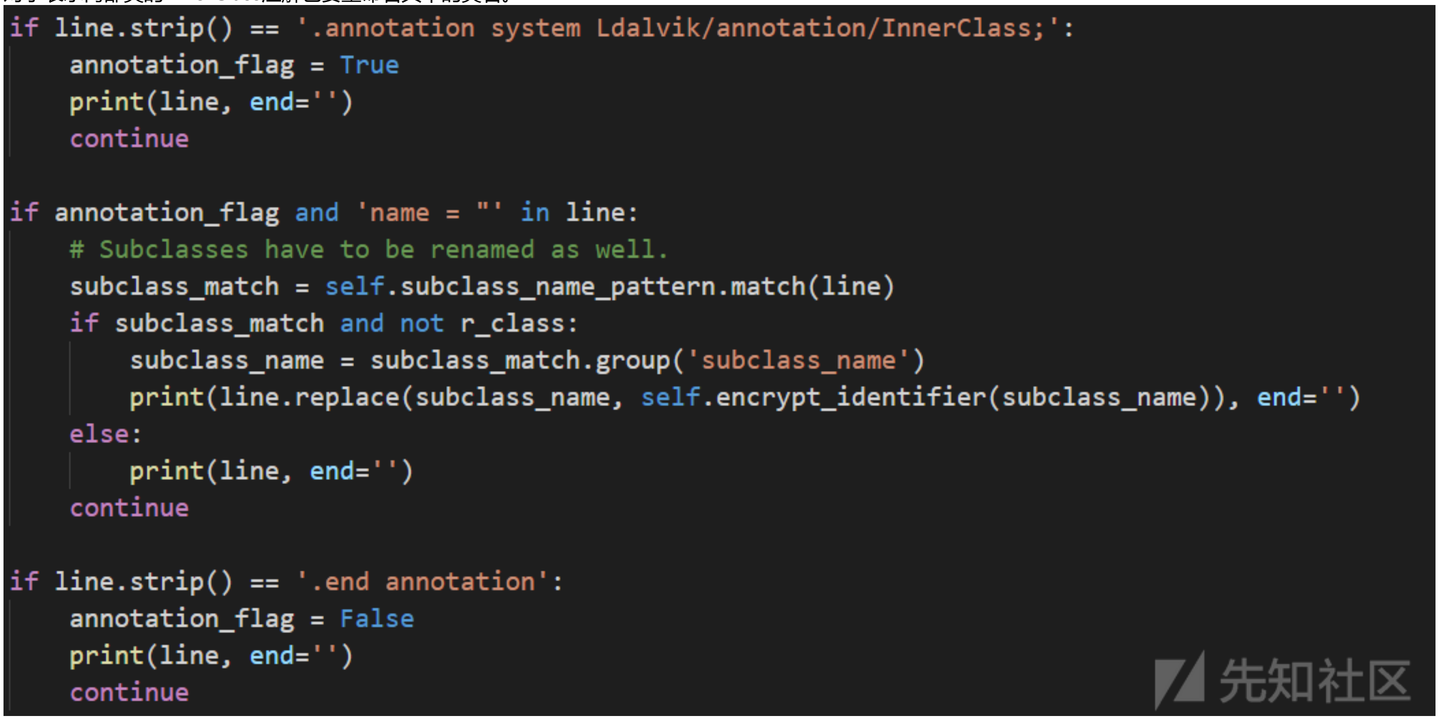
```python
def encrypt_identifier(self, identifier: str) -> str:
    identifier_md5 = util.get_string_md5(identifier)
    return 'p{0}'.format(identifier_md5.lower()[:8])

def slash_to_dot_notation_for_classes(self, rename_transformations: Dict[str, str]) -> Dict[str, str]:···

def transform_package_name(self, manifest_xml_root: Element):
    self.encrypted_package_name = '.'.join([self.encrypt_identifier(token)
                                            for token in self.package_name.split('.')])

    # Rename package name in manifest file.
    manifest_xml_root.set('package', self.encrypted_package_name)
    manifest_xml_root.set('{http://schemas.android.com/apk/res/android}sharedUserId',
                          '{0}.uid.shared'.format(util.get_random_string(16)))
```

调用rename_class_declarations函数对类名的定义重命名，对以/和$分割的每部分如果不是数字并且不是R类用和前面同样的方法重命名。



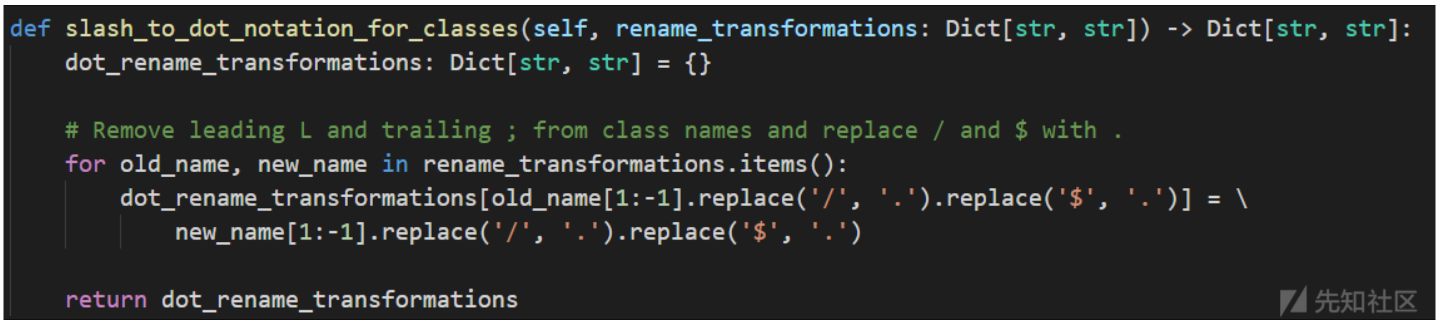
```python
if not class_name:
    class_match = util.class_pattern.match(line)
    if class_match:
        class_name = class_match.group('class_name')

        # Split class name to its components and encrypt them.
        class_tokens = self.split_class_pattern.split(class_name[1:-1])

        encrypted_class_name = 'L'
        separator_index = 1
        for token in class_tokens:
            separator_index += len(token)
            if token == 'R':
                r_class = True
            if token.isdigit():
                encrypted_class_name += token + class_name[separator_index]
            elif not r_class:
                encrypted_class_name += self.encrypt_identifier(token) + \
                                        class_name[separator_index]
            else:
                encrypted_class_name += token + class_name[separator_index]
            separator_index += 1

        print(line.replace(class_name, encrypted_class_name), end='')

        renamed_classes[class_name] = encrypted_class_name
        continue
```

对于表示内部类的InnerClass注解也要重命名其中的类名。



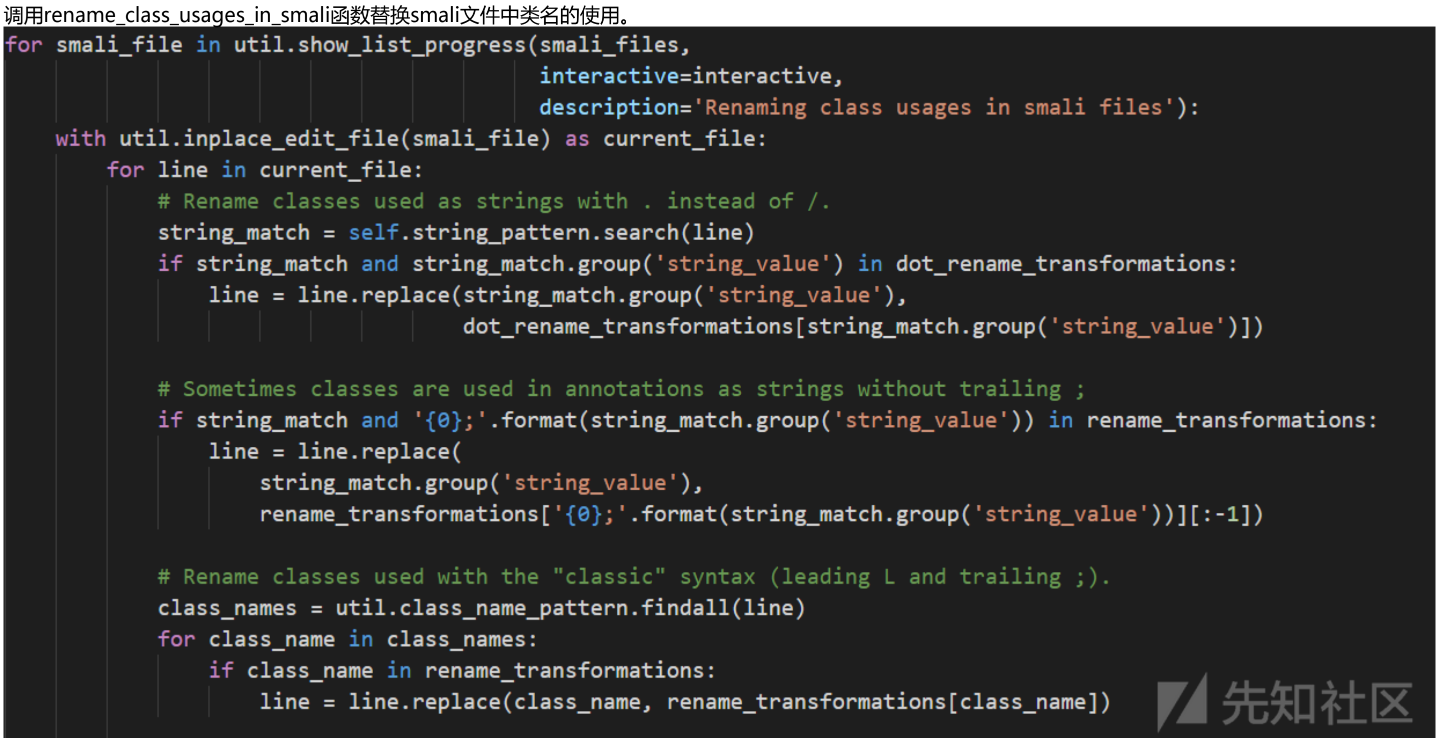
```python
if line.strip() == '.annotation system Ldalvik/annotation/InnerClass;':
    annotation_flag = True
    print(line, end='')
    continue

if annotation_flag and 'name = "' in line:
    # Subclasses have to be renamed as well.
    subclass_match = self.subclass_name_pattern.match(line)
    if subclass_match and not r_class:
        subclass_name = subclass_match.group('subclass_name')
        print(line.replace(subclass_name, self.encrypt_identifier(subclass_name)), end='')
    else:
        print(line, end='')
    continue

if line.strip() == '.end annotation':
    annotation_flag = False
    print(line, end='')
    continue
```

rename_class_declarations函数返回重命名前后类名的对应关系rename_transformations。接下来会调用slash_to_dot_notation_for_classes函数对rename_transforma



```python
def slash_to_dot_notation_for_classes(self, rename_transformations: Dict[str, str]) -> Dict[str, str]:
    dot_rename_transformations: Dict[str, str] = {}

    # Remove leading L and trailing ; from class names and replace / and $ with .
    for old_name, new_name in rename_transformations.items():
        dot_rename_transformations[old_name[1:-1].replace('/', '.').replace('$', '.')] = \
            new_name[1:-1].replace('/', '.').replace('$', '.')

    return dot_rename_transformations
```

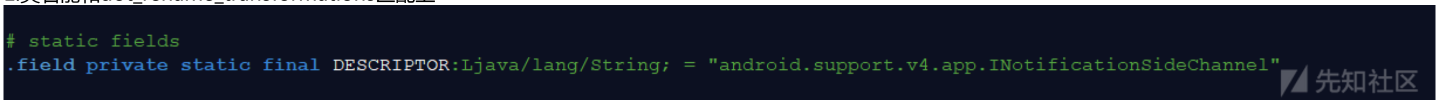
调用rename_class_usages_in_smali函数替换smali文件中类名的使用。



```python
for smali_file in util.show_list_progress(smali_files,
                                          interactive=interactive,
                                          description='Renaming class usages in smali files'):
    with util.inplace_edit_file(smali_file) as current_file:
        for line in current_file:
            # Rename classes used as strings with . instead of /.
            string_match = self.string_pattern.search(line)
            if string_match and string_match.group('string_value') in dot_rename_transformations:
                line = line.replace(string_match.group('string_value'),
                                    dot_rename_transformations[string_match.group('string_value')])

            # Sometimes classes are used in annotations as strings without trailing ;
            if string_match and '{0};'.format(string_match.group('string_value')) in rename_transformations:
                line = line.replace(
                    string_match.group('string_value'),
                    rename_transformations['{0};'.format(string_match.group('string_value'))][:-1])

            # Rename classes used with the "classic" syntax (leading L and trailing ;).
            class_names = util.class_name_pattern.findall(line)
            for class_name in class_names:
                if class_name in rename_transformations:
                    line = line.replace(class_name, rename_transformations[class_name])
```

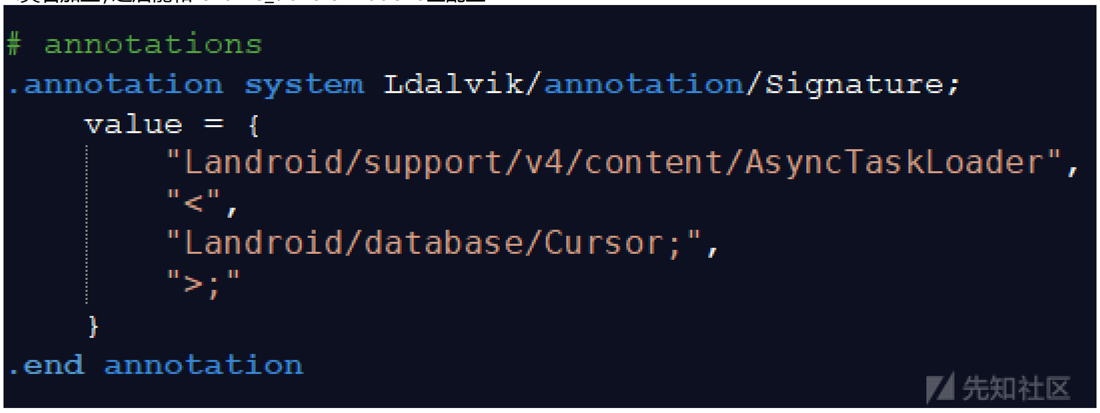
考虑了以下几种情况：

1.类名能和dot_rename_transformations匹配上



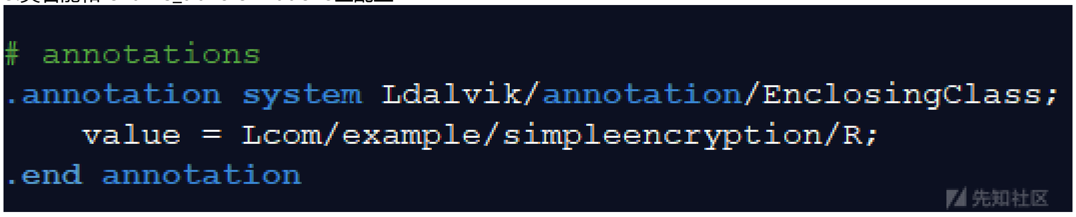
```
# static fields
.field private static final DESCRIPTOR:Ljava/lang/String; = "android.support.v4.app.INotificationSideChannel"
```

2.类名加上;之后能和rename_transformations匹配上



```
# annotations
.annotation system Ldalvik/annotation/Signature;
    value = {
        "Landroid/support/v4/content/AsyncTaskLoader",
        "<",
        "Landroid/database/Cursor;",
        ">;"
    }
.end annotation
```

3.类名能和rename_transformations匹配上



```
# annotations
.annotation system Ldalvik/annotation/EnclosingClass;
    value = Lcom/example/simpleencryption/R;
.end annotation
```

调用rename_class_usages_in_xml函数对xml文件中的类名进行替换。获取所有layout目录下的xml文件和AndroidManifest.xml文件。

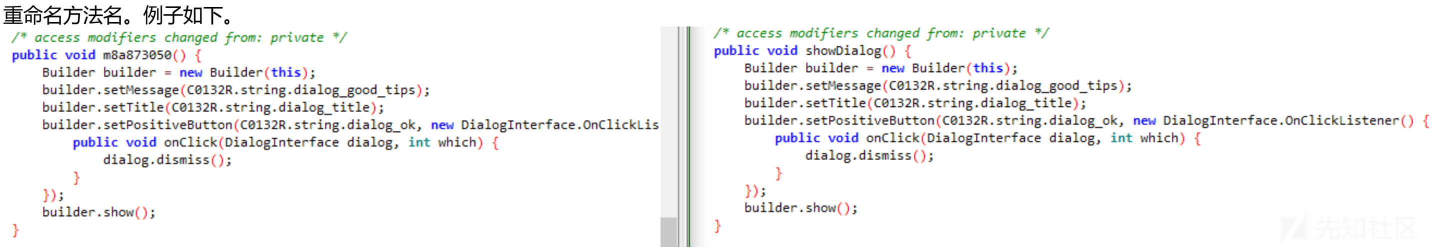
```python
xml_files: Set[str] = set(
    os.path.join(root, file_name)
    for root, dir_names, file_names in os.walk(obfuscation_info.get_resource_directory())
    for file_name in file_names if file_name.endswith('.xml') and 'layout' in root  # Only layout files.
)
xml_files.add(obfuscation_info.get_manifest_file())
```

替换时要从最长的到最短的替换，防止发生只替换了一部分的情况。还要替换没有包名的Activity名(AndroidManifest.xml中的String Chunk)。

```python
def rename_class_usages_in_xml(self, xml_files: List[str], rename_transformations: dict,
                               interactive: bool = False):
    dot_rename_transformations = self.slash_to_dot_notation_for_classes(rename_transformations)

    # Add package name.
    dot_rename_transformations[self.package_name] = self.encrypted_package_name

    for xml_file in util.show_list_progress(xml_files,
                                            interactive=interactive,
                                            description='Renaming class usages in xml files'):
        with open(xml_file, 'r', encoding='utf-8') as current_file:
            file_content = current_file.read()

        # Replace strings from longest to shortest (to avoid replacing partial strings).
        for old_name in sorted(dot_rename_transformations, reverse=True, key=lambda x: len(x)):
            file_content = file_content.replace(old_name, dot_rename_transformations[old_name])

            # Activity without package name (".ActivityName")
            if '"{0}"'.format(old_name.replace(self.package_name, '')) in file_content:
                file_content = file_content.replace(
                    '"{0}"'.format(old_name.replace(self.package_name, '')),
                    '"{0}"'.format(dot_rename_transformations[old_name].replace(self.encrypted_package_name, '')))

        with open(xml_file, 'w', encoding='utf-8') as current_file:
            current_file.write(file_content)
```

MethodRename

重命名方法名。例子如下。



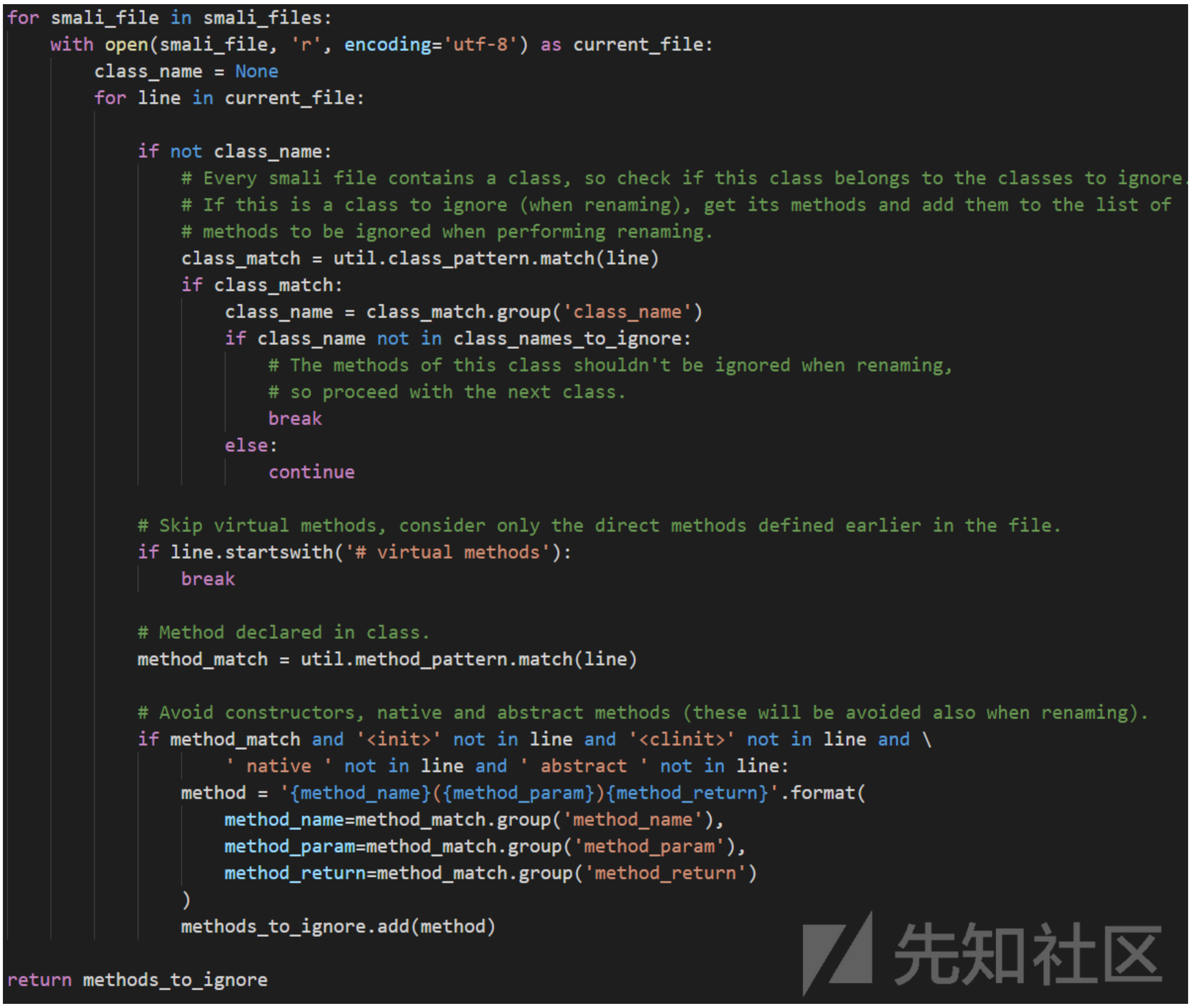
```java
/* access modifiers changed from: private */
public void m8a873050() {
    Builder builder = new Builder(this);
    builder.setMessage(C0132R.string.dialog_good_tips);
    builder.setTitle(C0132R.string.dialog_title);
    builder.setPositiveButton(C0132R.string.dialog_ok, new DialogInterface.OnClickLis
        public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        }
    });
    builder.show();
}
```

```java
/* access modifiers changed from: private */
public void showDialog() {
    Builder builder = new Builder(this);
    builder.setMessage(C0132R.string.dialog_good_tips);
    builder.setTitle(C0132R.string.dialog_title);
    builder.setPositiveButton(C0132R.string.dialog_ok, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        }
    });
    builder.show();
}
```

读取Obfuscapk\src\obfuscapk\resources目录下的android_class_names_api_27.txt文件得到android系统中的类名，然后读取smali文件中的.super得到apk中用到的父

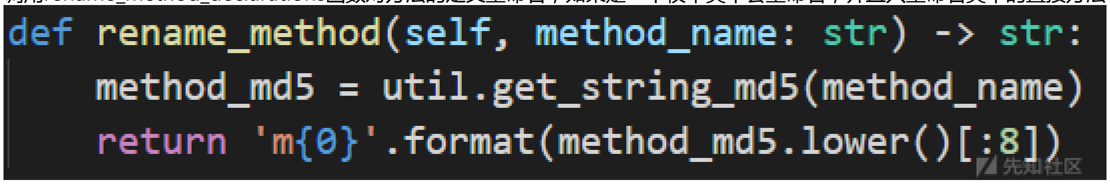
```python
android_class_names: Set[str] = set(util.get_android_class_names())
parent_class_names: Set[str] = self.get_parent_class_names(obfuscation_info.get_smali_files())

# Methods in parent classes belonging to the Android framework should be ignored when renaming.
classes_to_ignore: Set[str] = parent_class_names.intersection(android_class_names)
```

调用get_methods_to_ignore函数读取smali文件包含的类，检查这个类是否属于应该忽略的类。如果这是一个应该忽略的类，获取它的方法并添加到重命名时要忽略的方法

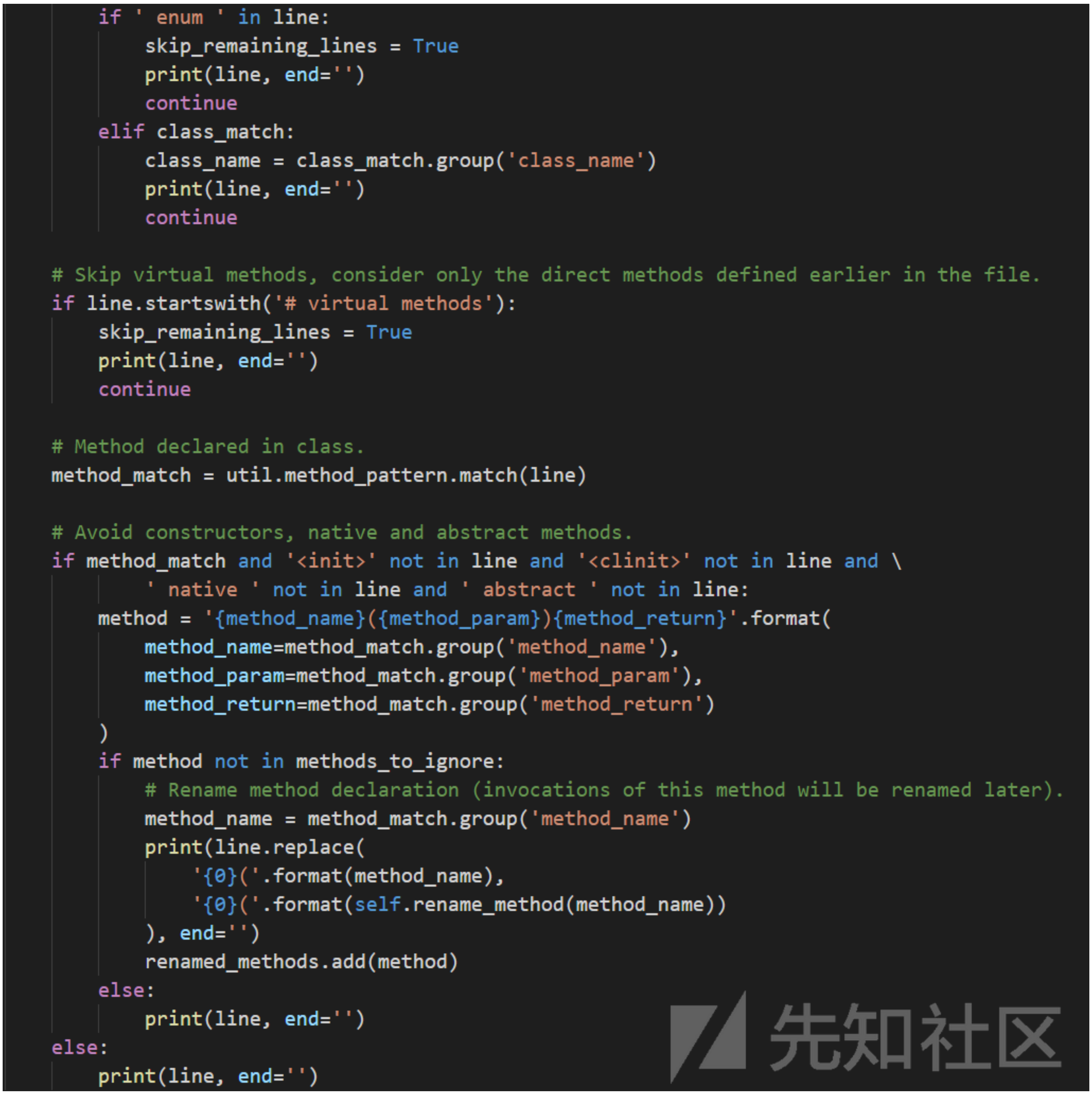


```python
for smali_file in smali_files:
    with open(smali_file, 'r', encoding='utf-8') as current_file:
        class_name = None
        for line in current_file:

            if not class_name:
                # Every smali file contains a class, so check if this class belongs to the classes to ignore.
                # If this is a class to ignore (when renaming), get its methods and add them to the list of
                # methods to be ignored when performing renaming.
                class_match = util.class_pattern.match(line)
                if class_match:
                    class_name = class_match.group('class_name')
                    if class_name not in class_names_to_ignore:
                        # The methods of this class shouldn't be ignored when renaming,
                        # so proceed with the next class.
                        break
                    else:
                        continue

            # Skip virtual methods, consider only the direct methods defined earlier in the file.
            if line.startswith('# virtual methods'):
                break

            # Method declared in class.
            method_match = util.method_pattern.match(line)

            # Avoid constructors, native and abstract methods (these will be avoided also when renaming).
            if method_match and '<init>' not in line and '<clinit>' not in line and \
                    ' native ' not in line and ' abstract ' not in line:
                method = '{method_name}({method_param}){method_return}'.format(
                    method_name=method_match.group('method_name'),
                    method_param=method_match.group('method_param'),
                    method_return=method_match.group('method_return')
                )
                methods_to_ignore.add(method)

return methods_to_ignore
```

调用rename_method_declarations函数对方法的定义重命名，如果是一个枚举类不会重命名，并且只重命名类中的直接方法中除了构造方法，native方法和抽象方法的不在

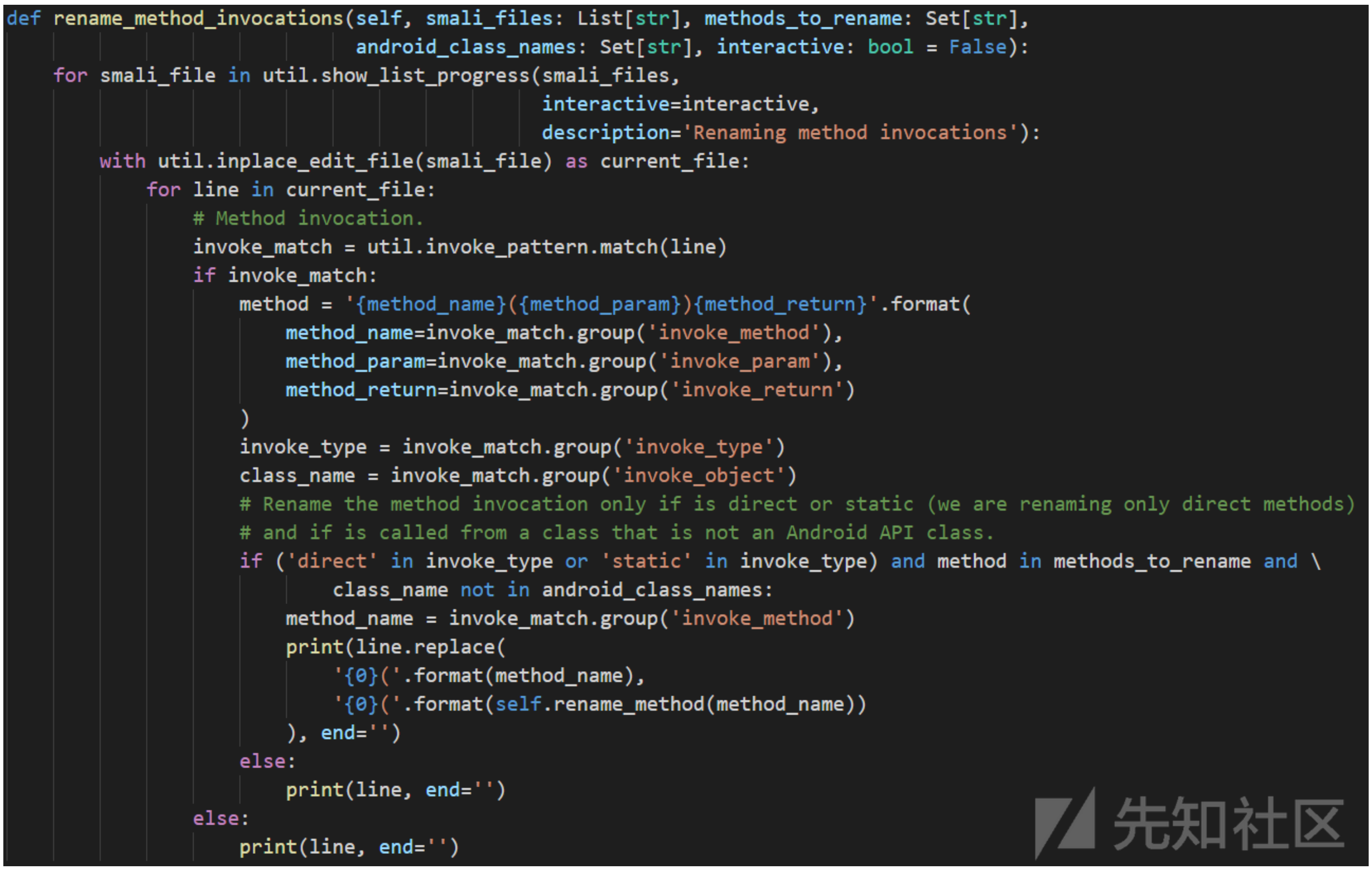


```python
def rename_method(self, method_name: str) -> str:
    method_md5 = util.get_string_md5(method_name)
    return 'm{0}'.format(method_md5.lower()[:8])
```



```
    if ' enum ' in line:
        skip_remaining_lines = True
        print(line, end='')
        continue
    elif class_match:
        class_name = class_match.group('class_name')
        print(line, end='')
        continue

# Skip virtual methods, consider only the direct methods defined earlier in the file.
if line.startswith('# virtual methods'):
    skip_remaining_lines = True
    print(line, end='')
    continue

# Method declared in class.
method_match = util.method_pattern.match(line)

# Avoid constructors, native and abstract methods.
if method_match and '<init>' not in line and '<clinit>' not in line and \
        ' native ' not in line and ' abstract ' not in line:
    method = '{method_name}({method_param}){method_return}'.format(
        method_name=method_match.group('method_name'),
        method_param=method_match.group('method_param'),
        method_return=method_match.group('method_return')
    )
    if method not in methods_to_ignore:
        # Rename method declaration (invocations of this method will be renamed later).
        method_name = method_match.group('method_name')
        print(line.replace(
            '{0}('.format(method_name),
            '{0}('.format(self.rename_method(method_name))
        ), end='')
        renamed_methods.add(method)
    else:
        print(line, end='')
else:
    print(line, end='')
```

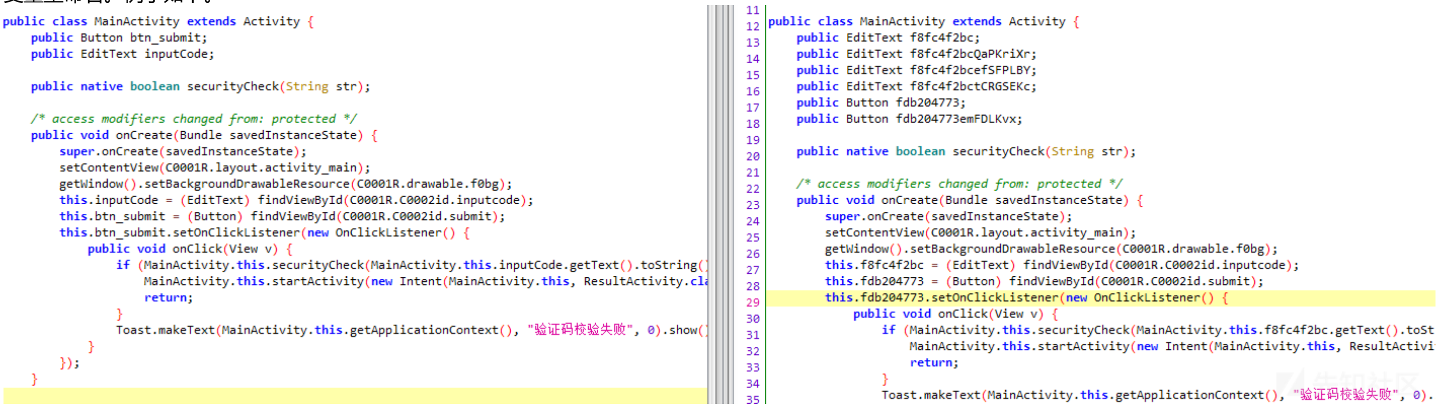
调用rename_method_invocations函数对方法的调用重命名，如果调用的是直接方法或者静态方法并且方法在renamed_methods中并且不是在android系统中的类中被调



```python
def rename_method_invocations(self, smali_files: List[str], methods_to_rename: Set[str],
                              android_class_names: Set[str], interactive: bool = False):
    for smali_file in util.show_list_progress(smali_files,
                                              interactive=interactive,
                                              description='Renaming method invocations'):
        with util.inplace_edit_file(smali_file) as current_file:
            for line in current_file:
                # Method invocation.
                invoke_match = util.invoke_pattern.match(line)
                if invoke_match:
                    method = '{method_name}({method_param}){method_return}'.format(
                        method_name=invoke_match.group('invoke_method'),
                        method_param=invoke_match.group('invoke_param'),
                        method_return=invoke_match.group('invoke_return')
                    )
                    invoke_type = invoke_match.group('invoke_type')
                    class_name = invoke_match.group('invoke_object')
                    # Rename the method invocation only if is direct or static (we are renaming only direct methods)
                    # and if is called from a class that is not an Android API class.
                    if ('direct' in invoke_type or 'static' in invoke_type) and method in methods_to_rename and \
                            class_name not in android_class_names:
                        method_name = invoke_match.group('invoke_method')
                        print(line.replace(
                            '{0}('.format(method_name),
                            '{0}('.format(self.rename_method(method_name))
                        ), end='')
                    else:
                        print(line, end='')
                else:
                    print(line, end='')
```

FieldRename

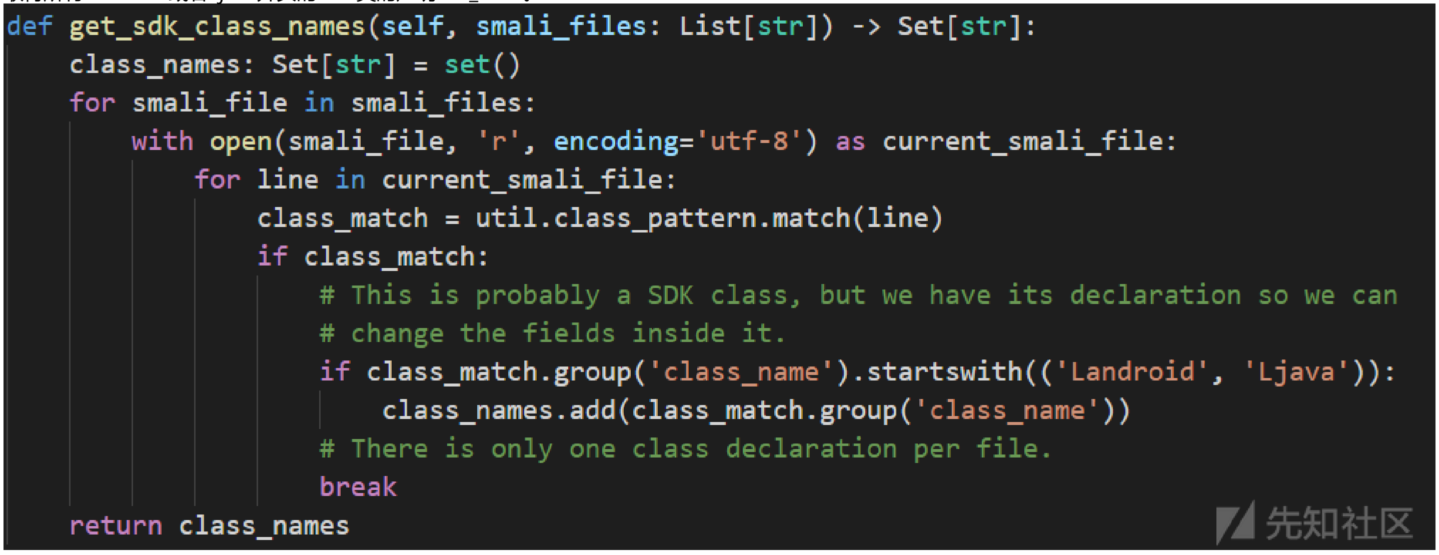
变量重命名。例子如下。



```java
public class MainActivity extends Activity {
    public Button btn_submit;
    public EditText inputCode;

    public native boolean securityCheck(String str);

    /* access modifiers changed from: protected */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(C0001R.layout.activity_main);
        getWindow().setBackgroundDrawableResource(C0001R.drawable.f0bg);
        this.inputCode = (EditText) findViewById(C0001R.C0002id.inputcode);
        this.btn_submit = (Button) findViewById(C0001R.C0002id.submit);
        this.btn_submit.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                if (MainActivity.this.securityCheck(MainActivity.this.inputCode.getText().toString(
                    MainActivity.this.startActivity(new Intent(MainActivity.this, ResultActivity.cla
                    return;
                }
                Toast.makeText(MainActivity.this.getApplicationContext(), "验证码校验失败", 0).show(
            }
        });
    }
```

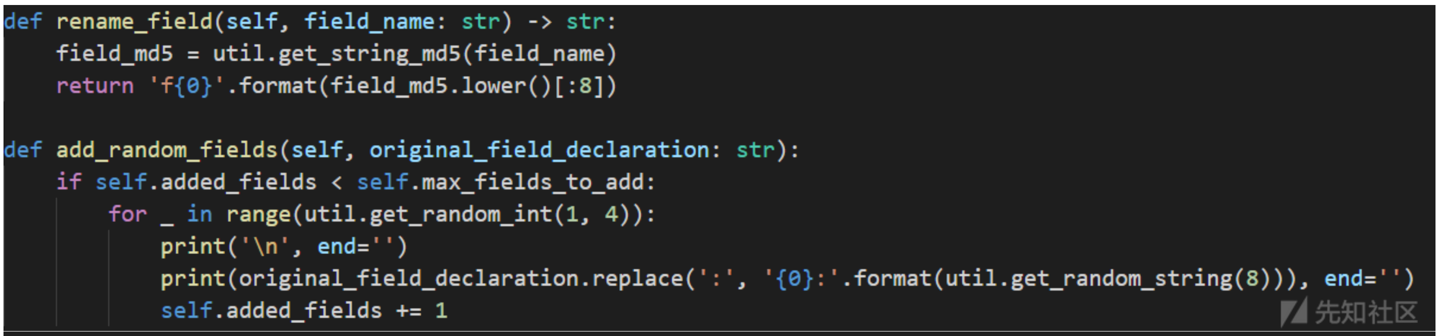
```java
public class MainActivity extends Activity {
    public EditText f8fc4f2bc;
    public EditText f8fc4f2bcQaPKriXr;
    public EditText f8fc4f2bcefSFPLBY;
    public EditText f8fc4f2bctCRGSEKc;
    public Button fdb204773;
    public Button fdb204773emFDLKvx;

    public native boolean securityCheck(String str);

    /* access modifiers changed from: protected */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(C0001R.layout.activity_main);
        getWindow().setBackgroundDrawableResource(C0001R.drawable.f0bg);
        this.f8fc4f2bc = (EditText) findViewById(C0001R.C0002id.inputcode);
        this.fdb204773 = (Button) findViewById(C0001R.C0002id.submit);
        this.fdb204773.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                if (MainActivity.this.securityCheck(MainActivity.this.f8fc4f2bc.getText().toSt
                    MainActivity.this.startActivity(new Intent(MainActivity.this, ResultActivi
                    return;
                }
                Toast.makeText(MainActivity.this.getApplicationContext(), "验证码校验失败", 0).
```

取得所有Landroid或者Ljava开头的SDK类的声明sdk_class。

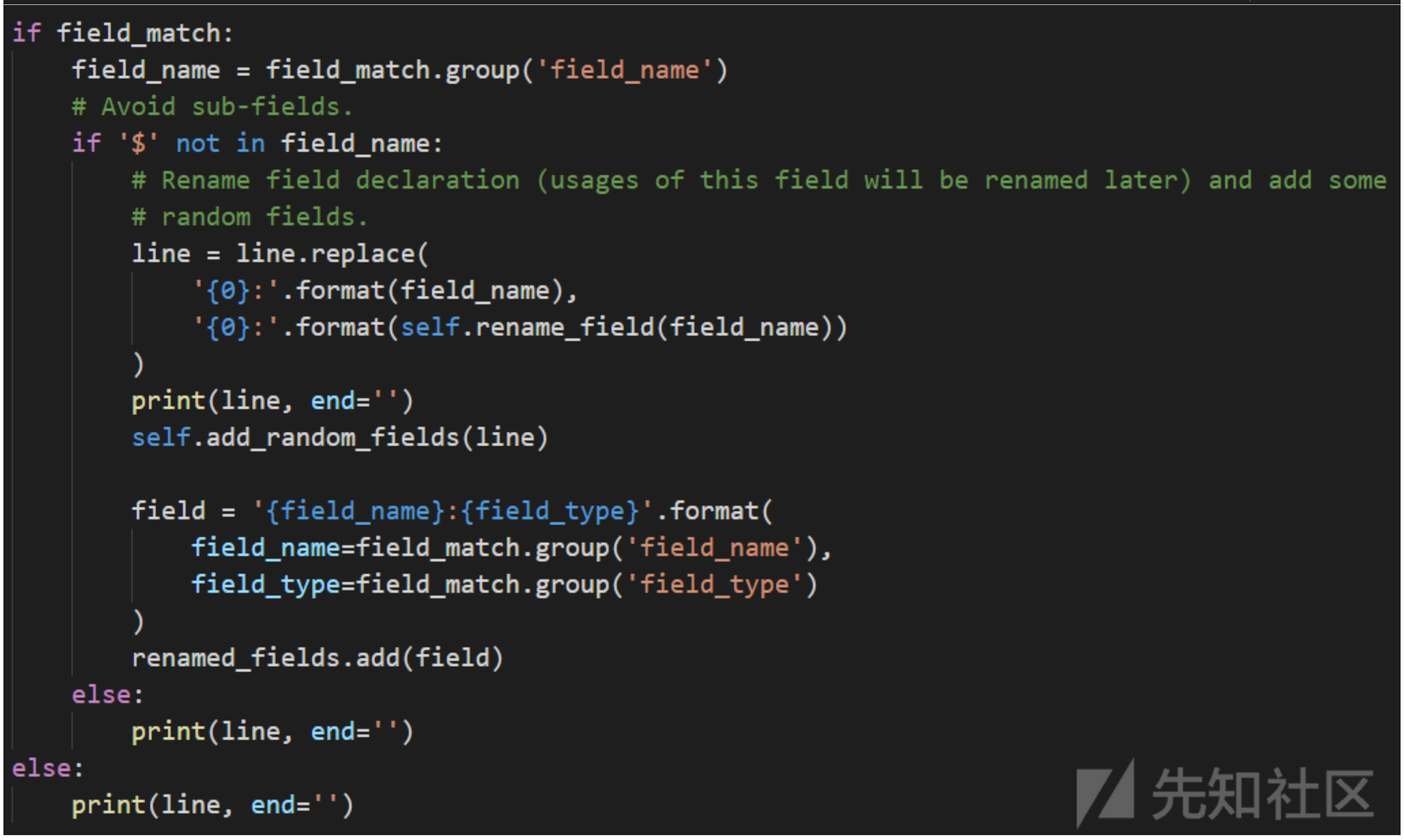


```python
def get_sdk_class_names(self, smali_files: List[str]) -> Set[str]:
    class_names: Set[str] = set()
    for smali_file in smali_files:
        with open(smali_file, 'r', encoding='utf-8') as current_smali_file:
            for line in current_smali_file:
                class_match = util.class_pattern.match(line)
                if class_match:
                    # This is probably a SDK class, but we have its declaration so we can
                    # change the fields inside it.
                    if class_match.group('class_name').startswith(('Landroid', 'Ljava')):
                        class_names.add(class_match.group('class_name'))
                    # There is only one class declaration per file.
                    break
    return class_names
```

判断是不是multidex，如果是的话要分别处理每个dex，分别调用rename_field_declarations函数进行对变量的定义重命名，并且每次重命名时都会调用add_random_fiel

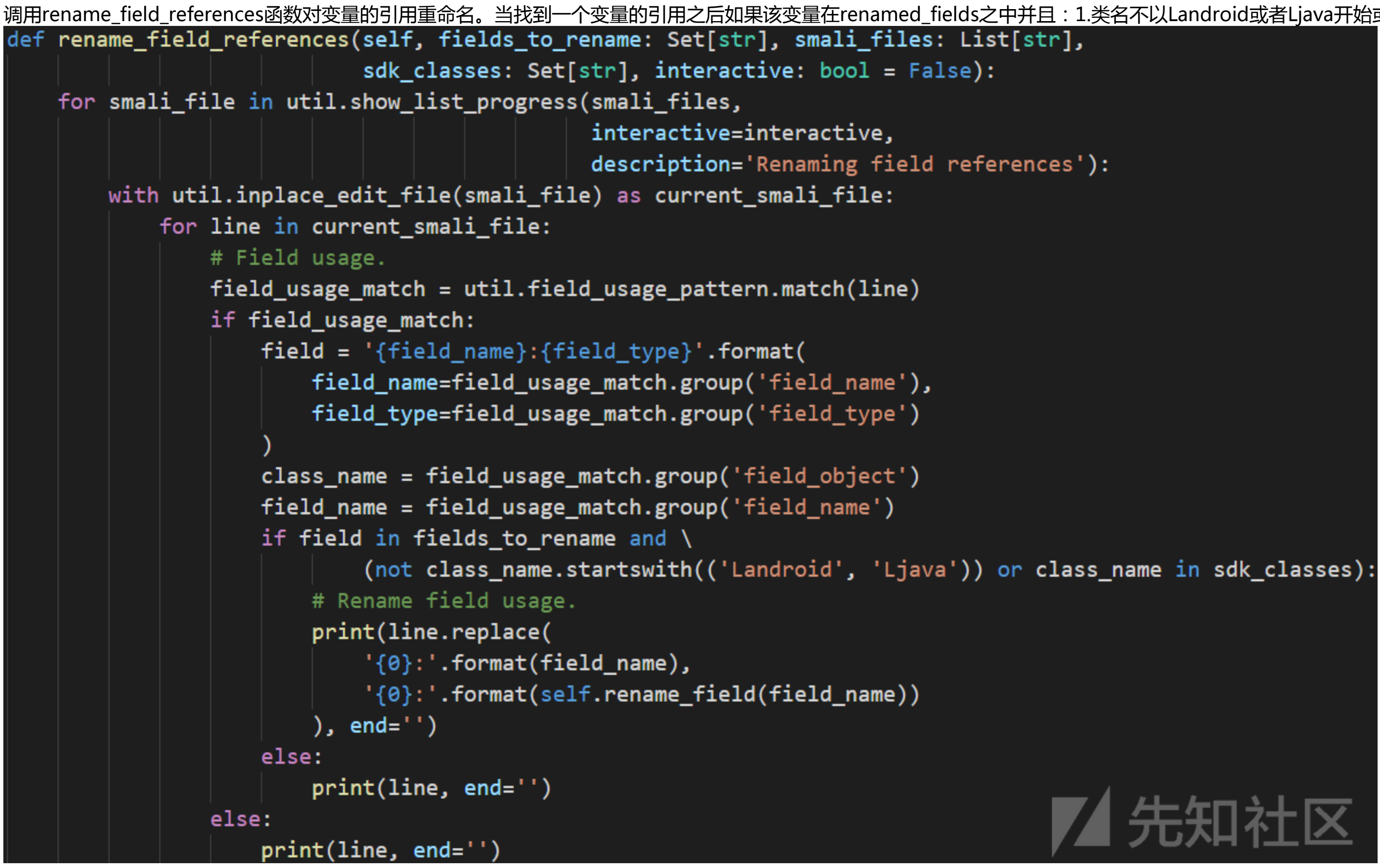


```
def rename_field(self, field_name: str) -> str:
    field_md5 = util.get_string_md5(field_name)
    return 'f{0}'.format(field_md5.lower()[:8])

def add_random_fields(self, original_field_declaration: str):
    if self.added_fields < self.max_fields_to_add:
        for _ in range(util.get_random_int(1, 4)):
            print('\n', end='')
            print(original_field_declaration.replace(':', '{0}:'.format(util.get_random_string(8))), end='')
            self.added_fields += 1
```



```
if field_match:
    field_name = field_match.group('field_name')
    # Avoid sub-fields.
    if '$' not in field_name:
        # Rename field declaration (usages of this field will be renamed later) and add some
        # random fields.
        line = line.replace(
            '{0}:'.format(field_name),
            '{0}:'.format(self.rename_field(field_name))
        )
        print(line, end='')
        self.add_random_fields(line)

        field = '{field_name}:{field_type}'.format(
            field_name=field_match.group('field_name'),
            field_type=field_match.group('field_type')
        )
        renamed_fields.add(field)
    else:
        print(line, end='')
else:
    print(line, end='')
```

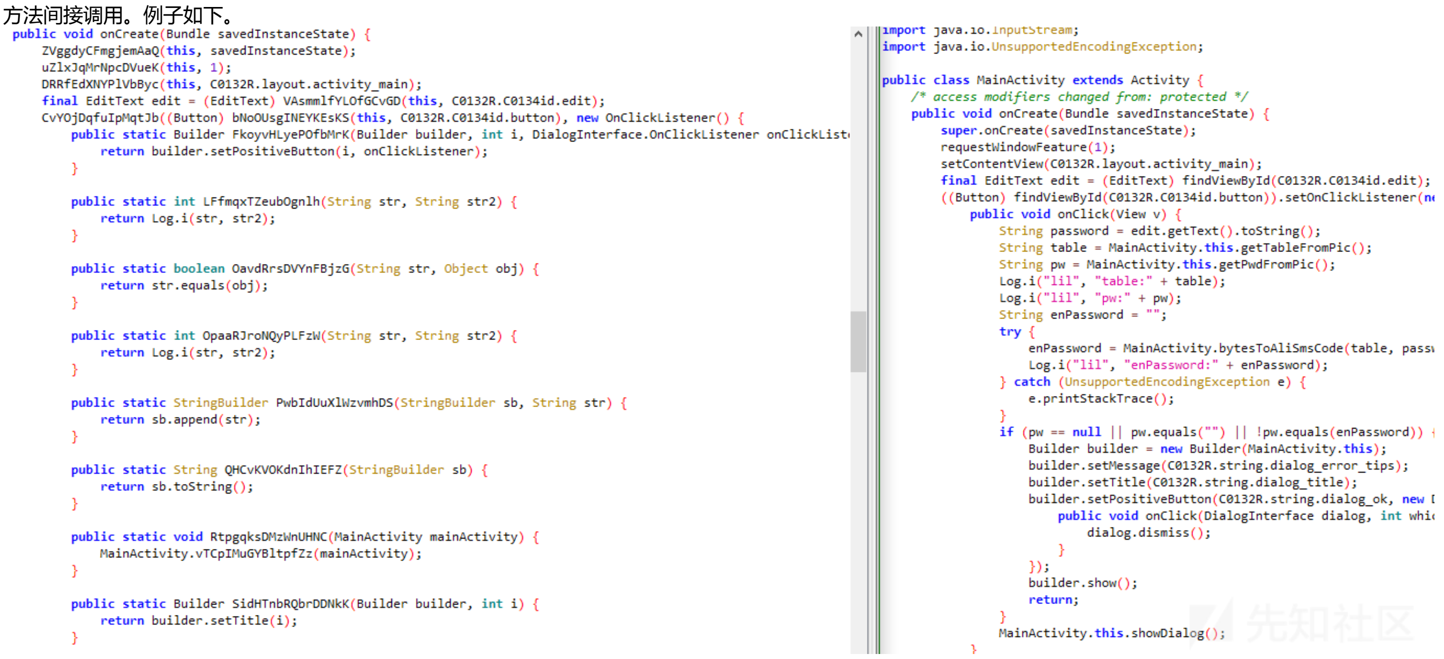
调用rename_field_references函数对变量的引用重命名。当找到一个变量的引用之后如果该变量在renamed_fields之中并且：1.类名不以Landroid或者Ljava开始或者2.类名



```
def rename_field_references(self, fields_to_rename: Set[str], smali_files: List[str],
                            sdk_classes: Set[str], interactive: bool = False):
    for smali_file in util.show_list_progress(smali_files,
                                              interactive=interactive,
                                              description='Renaming field references'):
        with util.inplace_edit_file(smali_file) as current_smali_file:
            for line in current_smali_file:
                # Field usage.
                field_usage_match = util.field_usage_pattern.match(line)
                if field_usage_match:
                    field = '{field_name}:{field_type}'.format(
                        field_name=field_usage_match.group('field_name'),
                        field_type=field_usage_match.group('field_type')
                    )
                    class_name = field_usage_match.group('field_object')
                    field_name = field_usage_match.group('field_name')
                    if field in fields_to_rename and \
                            (not class_name.startswith(('Landroid', 'Ljava')) or class_name in sdk_classes):
                        # Rename field usage.
                        print(line.replace(
                            '{0}:'.format(field_name),
                            '{0}:'.format(self.rename_field(field_name))
                        ), end='')
                    else:
                        print(line, end='')
                else:
                    print(line, end='')
```

CallIndirection

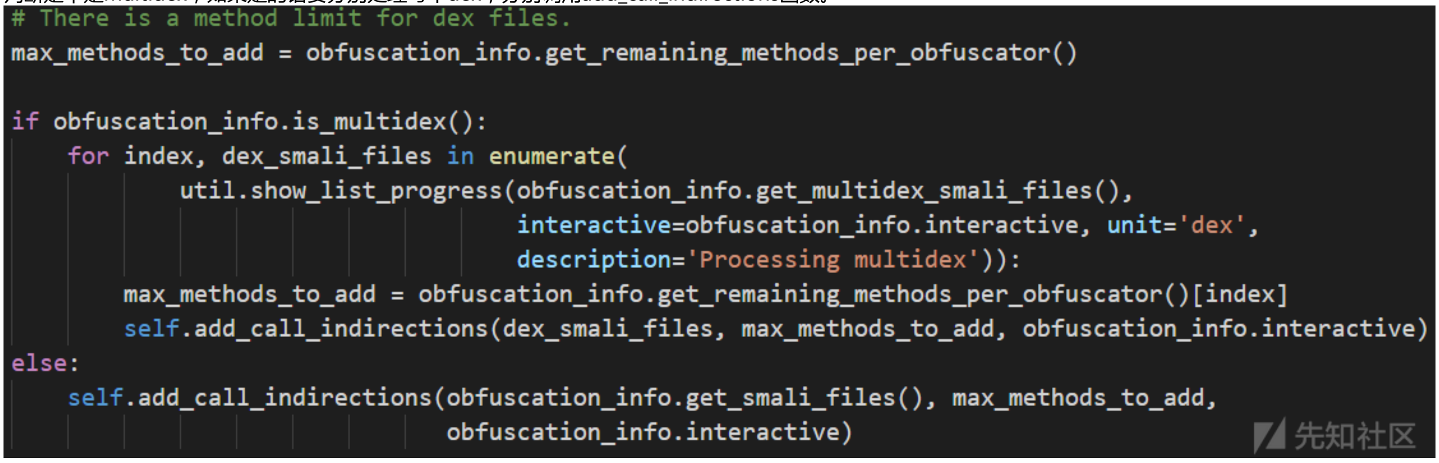
方法间接调用。例子如下。



```
public void onCreate(Bundle savedInstanceState) {
    ZVggdyCFmgjemAaQ(this, savedInstanceState);
    uZlxJqMrNpcDVueK(this, 1);
    DRRfEdXNYPlVbByc(this, C0132R.layout.activity_main);
    final EditText edit = (EditText) VAsmmlfYLOfGCvGD(this, C0132R.C0134id.edit);
    CvYOjDqfuIpMqtJb((Button) bNoOUsgINEYKEsKS(this, C0132R.C0134id.button), new OnClickListener() {
        public static Builder FkoyvHLyePOfbMrK(Builder builder, int i, DialogInterface.OnClickListener onClickListe
            return builder.setPositiveButton(i, onClickListener);
        }

        public static int LFfmqxTZeubOgnlh(String str, String str2) {
            return Log.i(str, str2);
        }

        public static boolean OavdRrsDVYnFBjzG(String str, Object obj) {
            return str.equals(obj);
        }

        public static int OpaaRJroNQyPLFzW(String str, String str2) {
            return Log.i(str, str2);
        }

        public static StringBuilder PwbIdUuXlWzvmhDS(StringBuilder sb, String str) {
            return sb.append(str);
        }

        public static String QHCvKVOKdnIhIEFZ(StringBuilder sb) {
            return sb.toString();
        }

        public static void RtpgqksDMzWnUHNC(MainActivity mainActivity) {
            MainActivity.vTCpIMuGYBltpfZz(mainActivity);
        }

        public static Builder SidHTnbRQbrDDNkK(Builder builder, int i) {
            return builder.setTitle(i);
        }
```

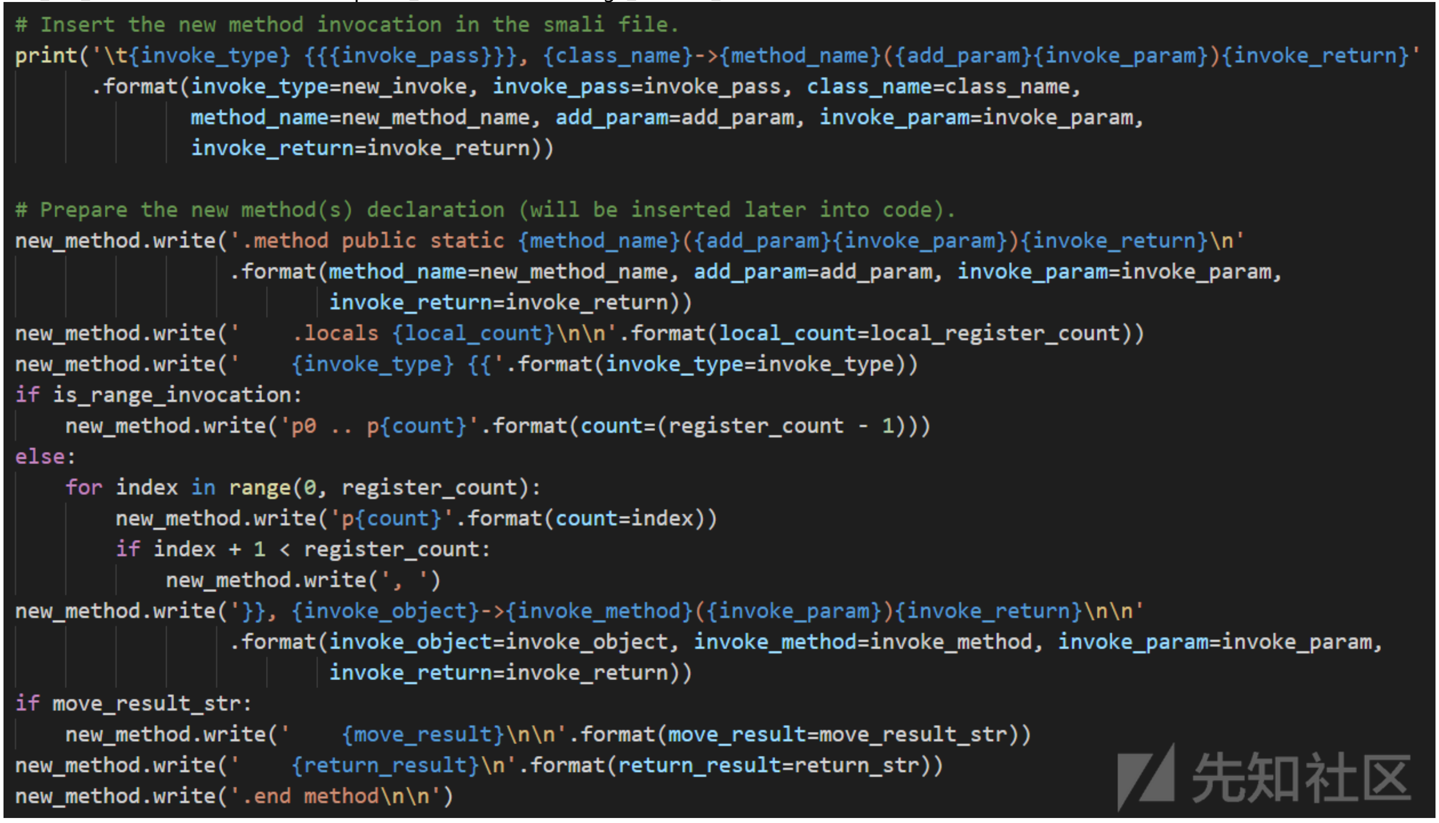
```
import java.io.InputStream;
import java.io.UnsupportedEncodingException;

public class MainActivity extends Activity {
    /* access modifiers changed from: protected */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(1);
        setContentView(C0132R.layout.activity_main);
        final EditText edit = (EditText) findViewById(C0132R.C0134id.edit);
        ((Button) findViewById(C0132R.C0134id.button)).setOnClickListener(ne
            public void onClick(View v) {
                String password = edit.getText().toString();
                String table = MainActivity.this.getTableFromPic();
                String pw = MainActivity.this.getPwdFromPic();
                Log.i("lil", "table:" + table);
                Log.i("lil", "pw:" + pw);
                String enPassword = "";
                try {
                    enPassword = MainActivity.bytesToAliSmsCode(table, passw
                    Log.i("lil", "enPassword:" + enPassword);
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
                if (pw == null || pw.equals("") || !pw.equals(enPassword)) {
                    Builder builder = new Builder(MainActivity.this);
                    builder.setMessage(C0132R.string.dialog_error_tips);
                    builder.setTitle(C0132R.string.dialog_title);
                    builder.setPositiveButton(C0132R.string.dialog_ok, new D
                        public void onClick(DialogInterface dialog, int whic
                            dialog.dismiss();
                        }
                    });
                    builder.show();
                    return;
                }
                MainActivity.this.showDialog();
            }
```

判断是不是multidex，如果是的话要分别处理每个dex，分别调用add_call_indirections函数。



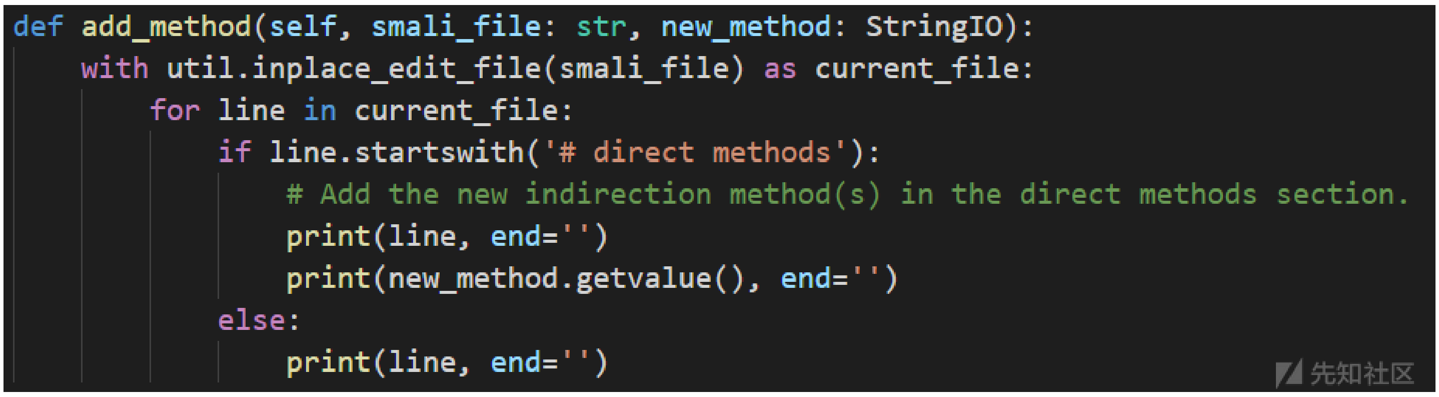
```
# There is a method limit for dex files.
max_methods_to_add = obfuscation_info.get_remaining_methods_per_obfuscator()

if obfuscation_info.is_multidex():
    for index, dex_smali_files in enumerate(
            util.show_list_progress(obfuscation_info.get_multidex_smali_files(),
                                    interactive=obfuscation_info.interactive, unit='dex',
                                    description='Processing multidex')):
        max_methods_to_add = obfuscation_info.get_remaining_methods_per_obfuscator()[index]
        self.add_call_indirections(dex_smali_files, max_methods_to_add, obfuscation_info.interactive)
else:
    self.add_call_indirections(obfuscation_info.get_smali_files(), max_methods_to_add,
                               obfuscation_info.interactive)
```

add_call_indirections函数中首先调用update_method函数->change_method_call函数将代码中调用原来的方法改成调用新增的方法，并准备好新增的方法的声明，新增



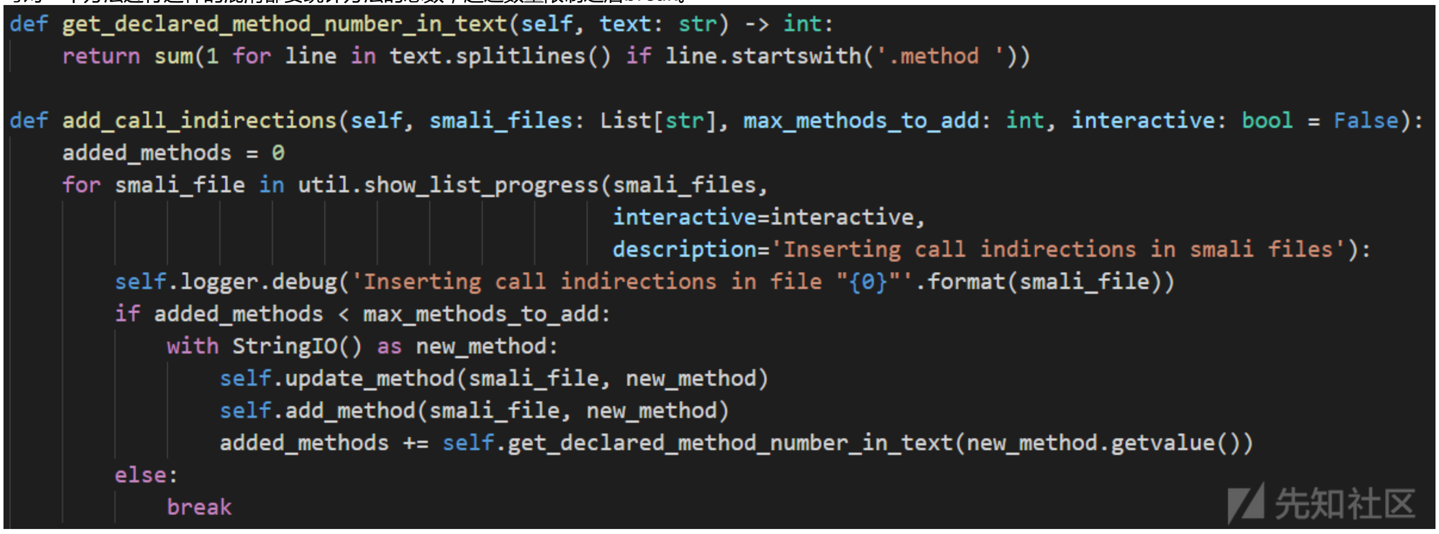
```
# Insert the new method invocation in the smali file.
print('\t{invoke_type} {{{invoke_pass}}}, {class_name}->{method_name}({add_param}{invoke_param}){invoke_return}'
      .format(invoke_type=new_invoke, invoke_pass=invoke_pass, class_name=class_name,
              method_name=new_method_name, add_param=add_param, invoke_param=invoke_param,
              invoke_return=invoke_return))

# Prepare the new method(s) declaration (will be inserted later into code).
new_method.write('.method public static {method_name}({add_param}{invoke_param}){invoke_return}\n'
                 .format(method_name=new_method_name, add_param=add_param, invoke_param=invoke_param,
                         invoke_return=invoke_return))
new_method.write('    .locals {local_count}\n\n'.format(local_count=local_register_count))
new_method.write('    {invoke_type} {{'.format(invoke_type=invoke_type))
if is_range_invocation:
    new_method.write('p0 .. p{count}'.format(count=(register_count - 1)))
else:
    for index in range(0, register_count):
        new_method.write('p{count}'.format(count=index))
        if index + 1 < register_count:
            new_method.write(', ')
new_method.write('}}, {invoke_object}->{invoke_method}({invoke_param}){invoke_return}\n\n'
                 .format(invoke_object=invoke_object, invoke_method=invoke_method, invoke_param=invoke_param,
                         invoke_return=invoke_return))
if move_result_str:
    new_method.write('    {move_result}\n\n'.format(move_result=move_result_str))
new_method.write('    {return_result}\n'.format(return_result=return_str))
new_method.write('.end method\n\n')
```

调用add_method函数将新增的方法的声明添加到`# direct methods`之后。



```python
def add_method(self, smali_file: str, new_method: StringIO):
    with util.inplace_edit_file(smali_file) as current_file:
        for line in current_file:
            if line.startswith('# direct methods'):
                # Add the new indirection method(s) in the direct methods section.
                print(line, end='')
                print(new_method.getvalue(), end='')
            else:
                print(line, end='')
```

每对一个方法进行这样的混淆都要统计方法的总数，超过数量限制之后break。



```python
def get_declared_method_number_in_text(self, text: str) -> int:
    return sum(1 for line in text.splitlines() if line.startswith('.method '))

def add_call_indirections(self, smali_files: List[str], max_methods_to_add: int, interactive: bool = False):
    added_methods = 0
    for smali_file in util.show_list_progress(smali_files,
                                              interactive=interactive,
                                              description='Inserting call indirections in smali files'):
        self.logger.debug('Inserting call indirections in file "{0}"'.format(smali_file))
        if added_methods < max_methods_to_add:
            with StringIO() as new_method:
                self.update_method(smali_file, new_method)
                self.add_method(smali_file, new_method)
                added_methods += self.get_declared_method_number_in_text(new_method.getvalue())
        else:
            break
```

总结

Obfuscapk中涉及的混淆技术包括加密，重命名，打乱控制流等绝大部分java层常见的混淆技术，组合在一起使用还是能有比较好的效果的，也能够在此基础之上二次开发