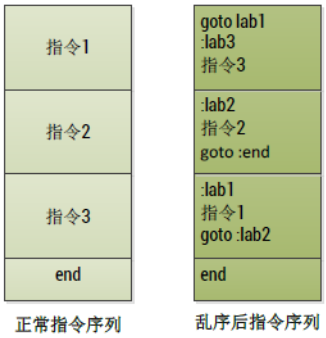
导读

为了增加Dex文件反编译后的阅读难度，我们可以对dex文件的指令打乱顺序，并加上goto语句保持原有的运行逻辑。但是，在实现过程中还是发现了不少略坑的地方。

0、原理介绍

一图胜千言



1、如何实现

引用自http://bbs.pediy.com/showthread.php?t=183116

以两个整数相加为例，Java 代码如下所示：

代码：

```
1. public void test(){

2. int a = 1;

3. int b = 2;

4. int c = a + b;

5. System.out.println(c);

6. }
```

反编译后的 smali 代码如下所示

代码：

```
1. .method public test()V

2. .locals 4

3.

4. .prologue

5. .line 24
6. const/4 v0, 0x1

7.

8. .line 25

9. .local v0, a:I

10. const/4 v1, 0x2
11.

12. .line 26

13. .local v1, b:I

14. add-int v2, v0, v1

15.
16. .line 27

17. .local v2, c:I

18. sget-object v3, Ljava/lang/System;->out:Ljava/io/PrintStream;

19.
```

```
20. invoke-virtual {v3, v2}, Ljava/io/PrintStream;->println(I)V
21.

22. .line 28

23. return-void
```

我们可以根据上述提到的代码乱序原理，将 test 这个函数乱序成如下代码所示(删除了.line)：
代码：

```
1. .method public test()V

2. .locals 4

3.

4. .local v2, c:I

5. goto :lab1

6. :lab3

7. sget-object v3, Ljava/lang/System;->out:Ljava/io/PrintStream;

8. invoke-virtual {v3, v2}, Ljava/io/PrintStream;->println(I)V

9. goto :end

10.

11. .local v1, b:I

12. :lab2

13. add-int v2, v0, v1

14. goto :lab3

15.

16. .local v0, a:I

17. :lab1

18. const/4 v0, 0x1

19. const/4 v1, 0x2

20. goto :lab2

21.

22. :end

23. return-void

24. .end method
```

最后使用 apktool 重新打包发布。进行代码乱序可以在一定程度上增加逆向分析的难度，例
如可以使用 dex2jar+jd-GUI 工具来分析上述乱序前后的代码。

2、实际应用
看起来挺简单的对吧。
但...
我写了个Java脚本把一个程序反编译后对smali码按如上办法随机打乱并回编，然后，bug bug bug...
而且，很多bug的报错是完全莫名奇妙，找不到其根源或与代码打乱有啥关系。

之后，我对出错的smali逐个挑出来，并分析其中的指令。
总结出存在以下规律。

规律1 smali文件含以下指令前缀实现乱序极可能出bug：

```
.catch
if
.array-data
.packed-switch
.sparse-switch
```

有些情况含以上指令乱序后不出错，但不能完全排除出错误的可能。

规律2 分割时机
smali码中如含有行号，以行号作为分割。
smali码中如不含行号，以"　　move"指令前缀作为分割。

3、乱序效果
以开源中国Android客户端为例，乱序后dex2jar
乱序前

```
private View.OnClickListener O = new View.OnClickListener()
{
  public void onClick(View paramAnonymousView)
  {
    TweetDetail.a(TweetDetail.this, TweetDetail.h(TweetDetail.this), TweetDetail.D(TweetDetail.this), true);
    TweetDetail.a(TweetDetail.this, TweetDetail.h(TweetDetail.this), TweetDetail.i(TweetDetail.this), 0, TweetDetail.p(TweetDetail.this), 2)
  }
};
```

乱序后

```
private View.OnClickListener O = new View.OnClickListener()
{
  public void onClick(View paramAnonymousView)
  {
    TweetDetail localTweetDetail1 = TweetDetail.this;
    int i = TweetDetail.h(TweetDetail.this);
    break label83;
    TweetDetail localTweetDetail2;
    int j;
    int k;
    while (true)
    {
      TweetDetail.a(localTweetDetail2, j, k, 0, localHandler2, 2);
      return;
      Handler localHandler2 = TweetDetail.p(TweetDetail.this);
    }
    while (true)
    {
      TweetDetail.a(localTweetDetail1, i, localHandler1, true);
      localTweetDetail2 = TweetDetail.this;
      j = TweetDetail.h(TweetDetail.this);
      k = TweetDetail.i(TweetDetail.this);
      break;
      label83: Handler localHandler1 = TweetDetail.D(TweetDetail.this);
    }
  }
};
```

大大地增加了逆向后查看代码逻辑的难度。

乱序后通过Androguard查看相似度对比：
开源中国 乱序前后apk对比相似度为 82%
唯品会 乱序前后apk对比相似度为 70%

大家有什么更好的想法欢迎一起讨论。