Convex one-one convex

DEX file obfuscation and encryption

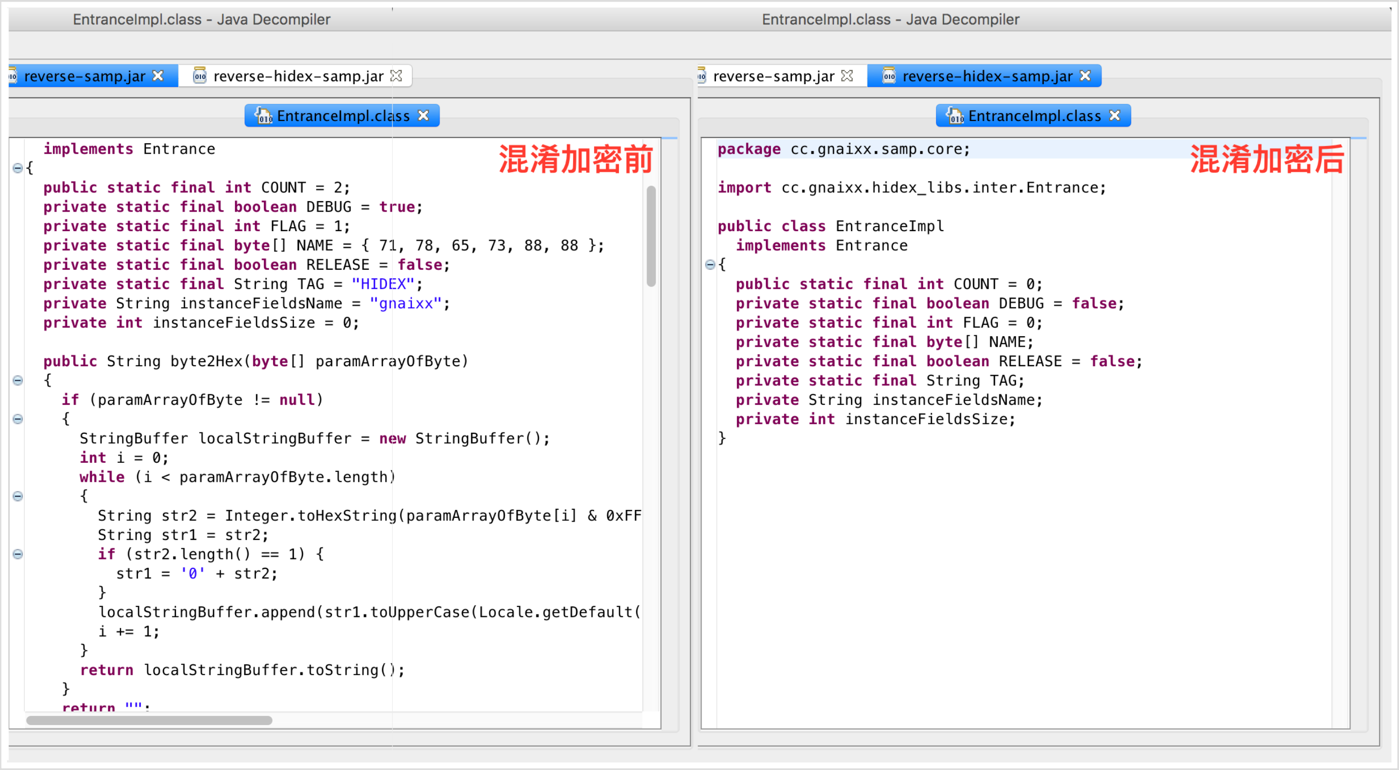
2017-02-06 | android | 1402

Some apps now consider loading some modules in the form of hot loads for security (such as encryption algorithms) or user experience (hot patch fixing bugs). Therefore, it is necessary to encrypt this part of dex. If dex is a repaired encryption algorithm, you don' t want to be decompiled by someone. Of course, you can also use a cryptographic algorithm to encrypt dex directly, and you can decrypt it before loading. However, the best encryption is to make it difficult for you to tell whether you encrypted it or not. In the reverse process, we get a dex that can be directly decompiled into Java source code. We probably think that this dex file can be analyzed without encryption.

0x00 Foreword

Obfuscated encryption is mainly used to hide the key code in dex files, ranging from light to heavy: the hiding of static variables, the repeated definition of functions, the hiding of functions, and the hiding of entire classes. The obfuscated dex file can still be decompiled into Java source code by tools such as dex2jar jade, but the key code inside is not visible.

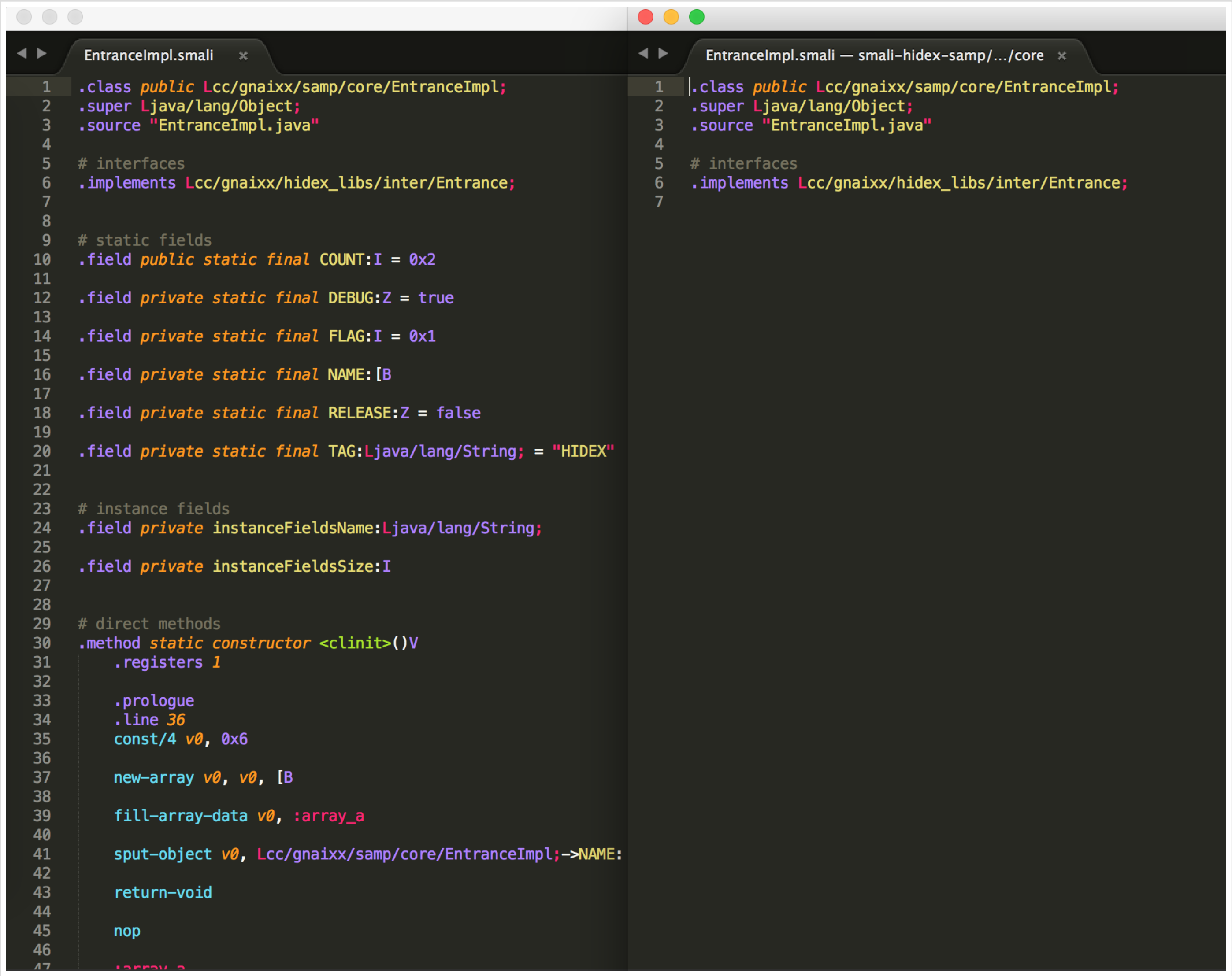
Java renderings:



```java
  implements Entrance
{
  public static final int COUNT = 2;
  private static final boolean DEBUG = true;
  private static final int FLAG = 1;
  private static final byte[] NAME = { 71, 78, 65, 73, 88, 88 };
  private static final boolean RELEASE = false;
  private static final String TAG = "HIDEX";
  private String instanceFieldsName = "gnaixx";
  private int instanceFieldsSize = 0;
  
  public String byte2Hex(byte[] paramArrayOfByte)
  {
    if (paramArrayOfByte != null)
    {
      StringBuffer localStringBuffer = new StringBuffer();
      int i = 0;
      while (i < paramArrayOfByte.length)
      {
        String str2 = Integer.toHexString(paramArrayOfByte[i] & 0xFF
        String str1 = str2;
        if (str2.length() == 1) {
          str1 = '0' + str2;
        }
        localStringBuffer.append(str1.toUpperCase(Locale.getDefault(
        i += 1;
      }
      return localStringBuffer.toString();
    }
    return "";
```

```java
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;

public class EntranceImpl
  implements Entrance
{
  public static final int COUNT = 0;
  private static final boolean DEBUG = false;
  private static final int FLAG = 0;
  private static final byte[] NAME;
  private static final boolean RELEASE = false;
  private static final String TAG;
  private String instanceFieldsName;
  private int instanceFieldsSize;
}
```

Smali renderings:



```
.class public Lcc/gnaixx/samp/core/EntranceImpl;
.super Ljava/lang/Object;
.source "EntranceImpl.java"

# interfaces
.implements Lcc/gnaixx/hidex_libs/inter/Entrance;


# static fields
.field public static final COUNT:I = 0x2

.field private static final DEBUG:Z = true

.field private static final FLAG:I = 0x1

.field private static final NAME:[B

.field private static final RELEASE:Z = false

.field private static final TAG:Ljava/lang/String; = "HIDEX"


# instance fields
.field private instanceFieldsName:Ljava/lang/String;

.field private instanceFieldsSize:I


# direct methods
.method static constructor <clinit>()V
    .registers 1

    .prologue
    .line 36
    const/4 v0, 0x6

    new-array v0, v0, [B

    fill-array-data v0, :array_a

    sput-object v0, Lcc/gnaixx/samp/core/EntranceImpl;->NAME:

    return-void

    nop

    :array_a
```

```
.class public Lcc/gnaixx/samp/core/EntranceImpl;
.super Ljava/lang/Object;
.source "EntranceImpl.java"

# interfaces
.implements Lcc/gnaixx/hidex_libs/inter/Entrance;

```

Source address and instructions for use on **hidex-hack** on github

0x01 dex format analysis

The dex file format is described in more detail in the previous article. The specifics can be seen in the **dex file format analysis** . Here is a brief introduction to the layout of the entire dex file.

1.header (dex header)

header dex overview of the entire document distribution, including: **Magic** , **Checksum** , **Signature** , **FILE_SIZE** , **header_size** , **endian_tag** , **Link** , **Map** , **string_ids** , **type_ids** , **proto_ids** , **field_ids** , **method_ids** , **class_defs** , **Data** .

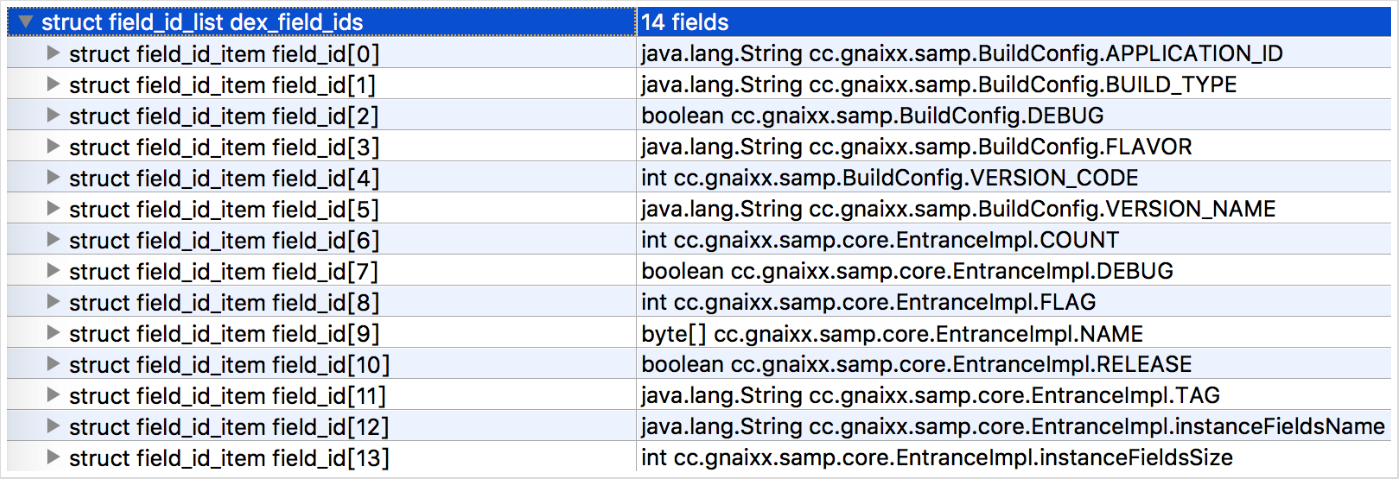
- **Checksum** and **signature** are checksums that need to be modified after modification

- **String_ids** , **type_ids** , **proto_ids** , **field_ids** , **method_ids** stores different types of values as type array sections (which I **pick** up)
- **Class_defs** stored class definition is also the focus of our changes
- **Data** is a data store, including all data

2. Type Array Section

Type Array section contains **string_ids** , **type_ids** , **proto_ids** , **field_ids** , **method_ids** . Respectively: string, type, function signature, attributes, functions. Each section stores an array of data of the corresponding type. You can use 010Editor to analyze binary data.

Property example:



▼ struct field_id_list dex_field_ids	14 fields
▶ struct field_id_item field_id[0]	java.lang.String cc.gnaixx.samp.BuildConfig.APPLICATION_ID
▶ struct field_id_item field_id[1]	java.lang.String cc.gnaixx.samp.BuildConfig.BUILD_TYPE
▶ struct field_id_item field_id[2]	boolean cc.gnaixx.samp.BuildConfig.DEBUG
▶ struct field_id_item field_id[3]	java.lang.String cc.gnaixx.samp.BuildConfig.FLAVOR
▶ struct field_id_item field_id[4]	int cc.gnaixx.samp.BuildConfig.VERSION_CODE
▶ struct field_id_item field_id[5]	java.lang.String cc.gnaixx.samp.BuildConfig.VERSION_NAME
▶ struct field_id_item field_id[6]	int cc.gnaixx.samp.core.EntranceImpl.COUNT
▶ struct field_id_item field_id[7]	boolean cc.gnaixx.samp.core.EntranceImpl.DEBUG
▶ struct field_id_item field_id[8]	int cc.gnaixx.samp.core.EntranceImpl.FLAG
▶ struct field_id_item field_id[9]	byte[] cc.gnaixx.samp.core.EntranceImpl.NAME
▶ struct field_id_item field_id[10]	boolean cc.gnaixx.samp.core.EntranceImpl.RELEASE
▶ struct field_id_item field_id[11]	java.lang.String cc.gnaixx.samp.core.EntranceImpl.TAG
▶ struct field_id_item field_id[12]	java.lang.String cc.gnaixx.samp.core.EntranceImpl.instanceFieldsName
▶ struct field_id_item field_id[13]	int cc.gnaixx.samp.core.EntranceImpl.instanceFieldsSize

3. The definition

of the **class definition** class is the focus of the modification, which saves the structure of all classes and is the most complex part of the structure of the entire dex file. Including: static attribute variables, member number variables, virtual functions, direct functions, static functions and other data.

0x02 implementation function

By analyzing the dex file format, there are four types of obfuscated encryption that can now be implemented:

1. Static variables hidden
2. Duplicate function definition
3. Function hiding
4. Class definition hide

The four obfuscated implementations are implemented by modifying the fields in the **class_def** structure. The structure of **class_def** can be seen in the json format (only fields are listed here):

```
{
 "class_def" : {
     "class_idx" : 01
     "static_values_off" : 000 ,
     "class_data_off" : 001 ,
     "class_data" : {
          "direct_methods_size" : 001 ,
          "virtual_methods_size" : 002 ,
          "virtual_methods" :[
               {
                    "code_off" : 003
               },
               {
                    "code_off" : 004
               }
          ]
     }
 }
}
```

Field meaning:

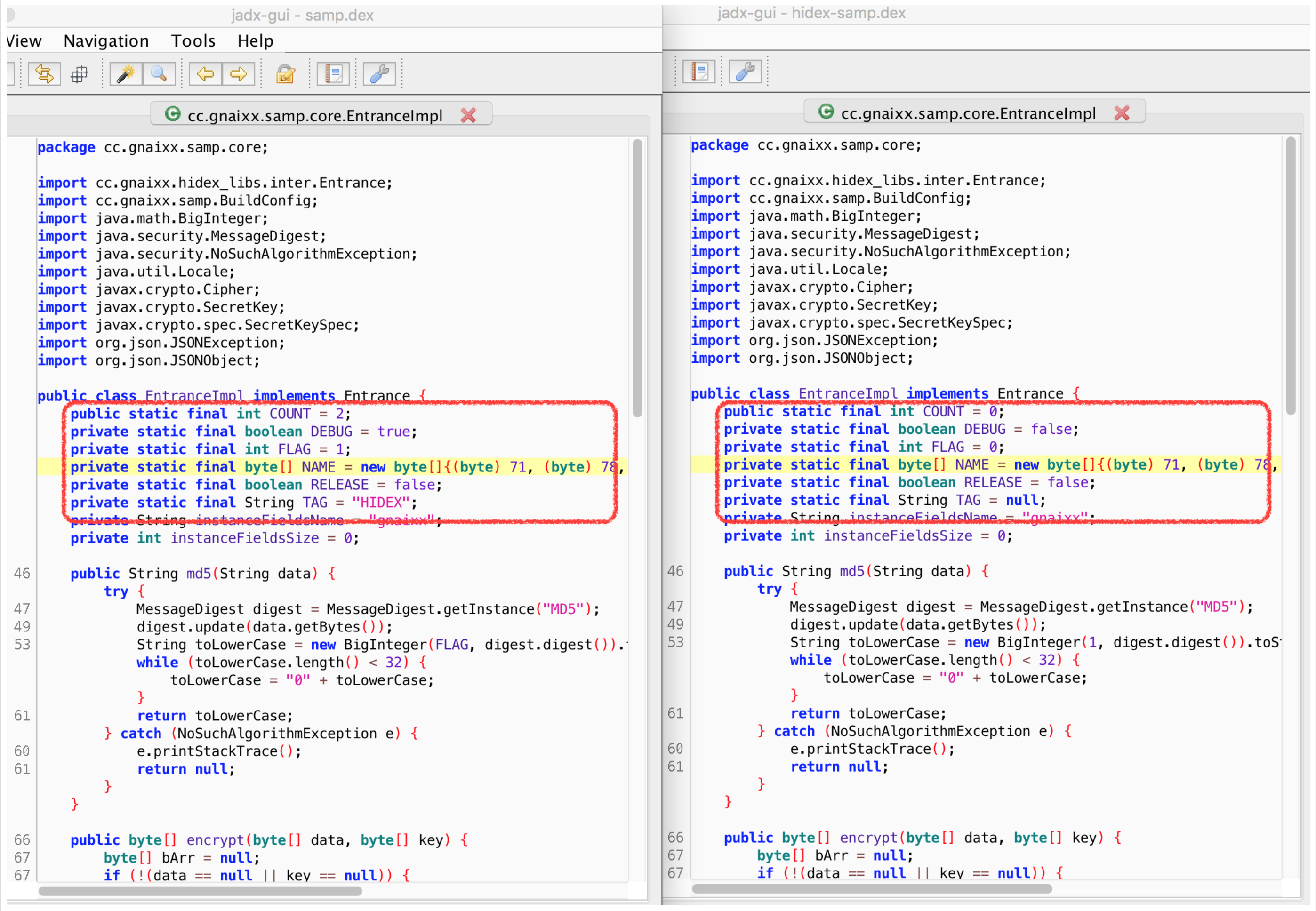
- Class_idx: the name of the class name, an index of type_ids
- Class_def: class definition structure
- Static_values_off: static variable value offset
- Class_data_off: class definition offset
- Class_data: class definition structure
- Direct_methods_size: number of direct functions
- Virtual_methods_size: number of virtual functions
- Virtual_methods: virtual function structures
- Code_off: function code offset

It is easy to get the implementation of the four functions through the above field descriptions, the following one.

1 static variables hidden

Static_vaules_off holds the offset of the value of the static variable in each class, pointing to a list in the data area, in the format of **?encode_array_item** , or 0 if there is no such **entry** . So to achieve the static variable assignment hidden only need to modify the **static_values_off** value to 0.

To achieve results:


jadx-gui - samp.dex
View Navigation Tools Help
cc.gnaixx.samp.core.EntranceImpl
jadx-gui - hidex-samp.dex
cc.gnaixx.samp.core.EntranceImpl

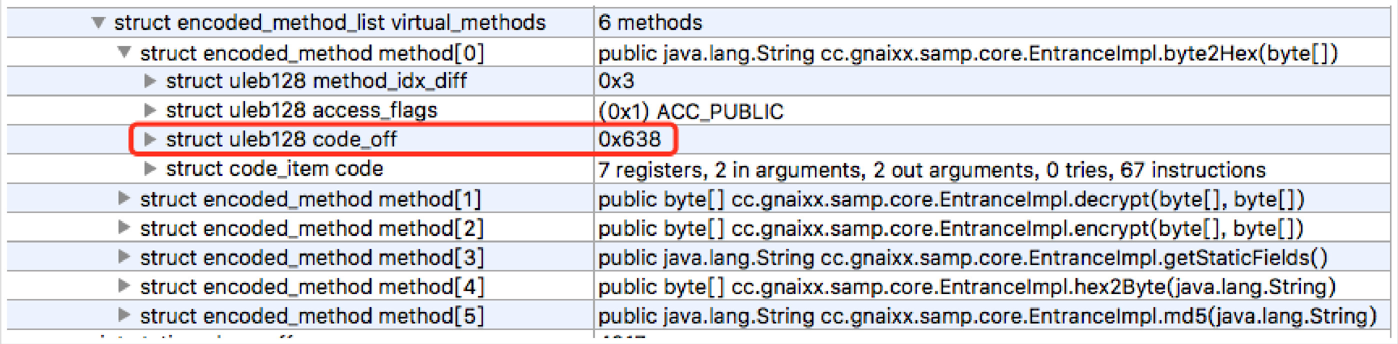
```java
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;
import cc.gnaixx.samp.BuildConfig;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Locale;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.json.JSONException;
import org.json.JSONObject;

public class EntranceImpl implements Entrance {
    public static final int COUNT = 2;
    private static final boolean DEBUG = true;
    private static final int FLAG = 1;
    private static final byte[] NAME = new byte[]{(byte) 71, (byte) 78,
    private static final boolean RELEASE = false;
    private static final String TAG = "HIDEX";
    private String instanceFieldsName = "gnaixx";
    private int instanceFieldsSize = 0;

    public String md5(String data) {
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            digest.update(data.getBytes());
            String toLowerCase = new BigInteger(FLAG, digest.digest()).
            while (toLowerCase.length() < 32) {
                toLowerCase = "0" + toLowerCase;
            }
            return toLowerCase;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }

    public byte[] encrypt(byte[] data, byte[] key) {
        byte[] bArr = null;
        if (!(data == null || key == null)) {
```

```java
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;
import cc.gnaixx.samp.BuildConfig;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Locale;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.json.JSONException;
import org.json.JSONObject;

public class EntranceImpl implements Entrance {
    public static final int COUNT = 0;
    private static final boolean DEBUG = false;
    private static final int FLAG = 0;
    private static final byte[] NAME = new byte[]{(byte) 71, (byte) 78,
    private static final boolean RELEASE = false;
    private static final String TAG = null;
    private String instanceFieldsName = "gnaixx";
    private int instanceFieldsSize = 0;

    public String md5(String data) {
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            digest.update(data.getBytes());
            String toLowerCase = new BigInteger(1, digest.digest()).toS
            while (toLowerCase.length() < 32) {
                toLowerCase = "0" + toLowerCase;
            }
            return toLowerCase;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }

    public byte[] encrypt(byte[] data, byte[] key) {
        byte[] bArr = null;
        if (!(data == null || key == null)) {
```

The static array data here is not hidden because I don't know how to do it. ☹

2 function definition

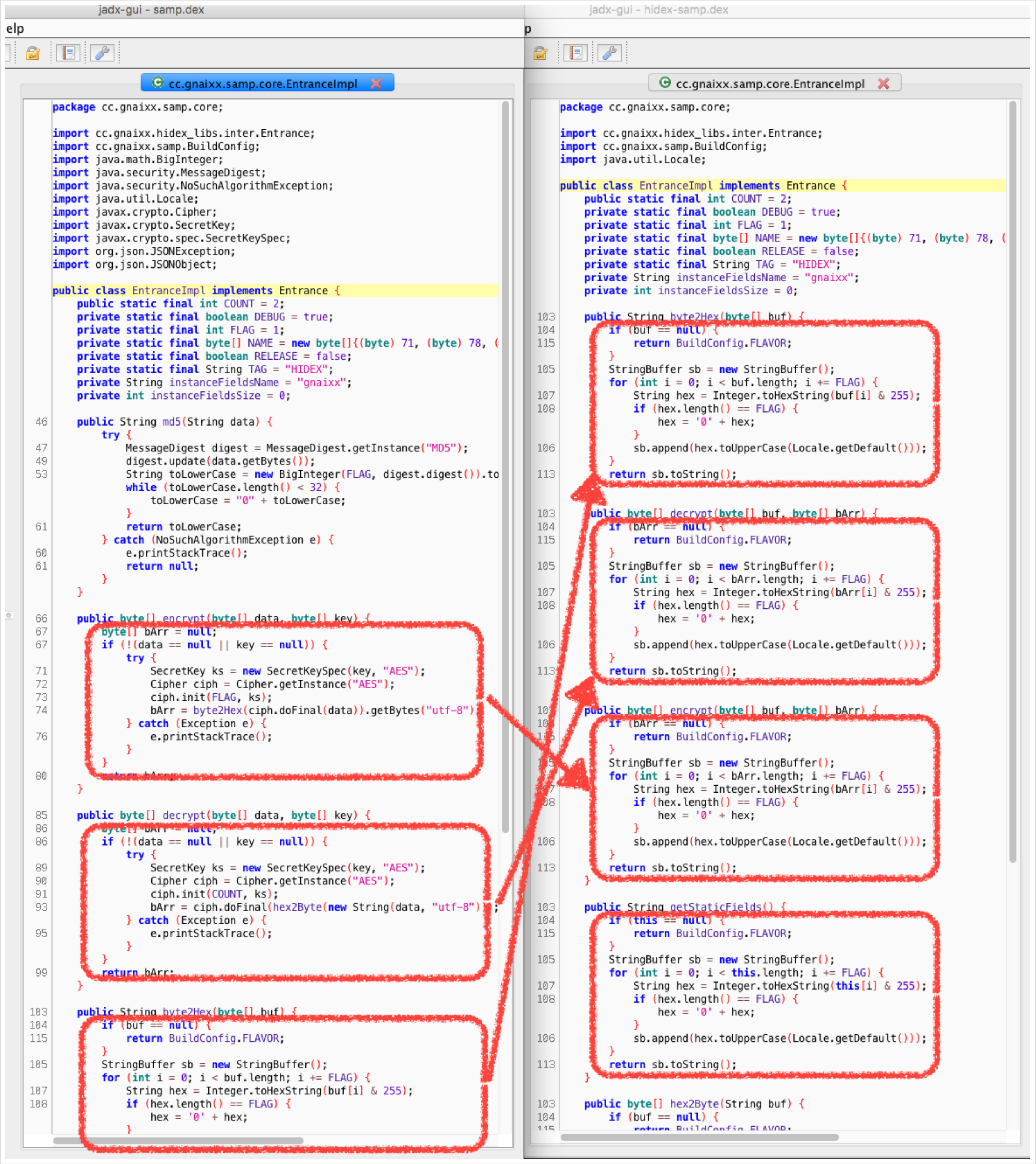
Class_def -> class_data -> virtual_methods -> code_ff represents the code offset address of a function in a class. It should be mentioned a concept: all functions are achieved virtual functions in Java, C ++, and it is not the same, all the changes here are **virtual_methods** in **code_off** .



▼ struct encoded_method_list virtual_methods	6 methods
▼ struct encoded_method method[0]	public java.lang.String cc.gnaixx.samp.core.EntranceImpl.byte2Hex(byte[])
▶ struct uleb128 method_idx_diff	0x3
▶ struct uleb128 access_flags	(0x1) ACC_PUBLIC
▶ struct uleb128 code_off	0x638
▶ struct code_item code	7 registers, 2 in arguments, 2 out arguments, 0 tries, 67 instructions
▶ struct encoded_method method[1]	public byte[] cc.gnaixx.samp.core.EntranceImpl.decrypt(byte[], byte[])
▶ struct encoded_method method[2]	public byte[] cc.gnaixx.samp.core.EntranceImpl.encrypt(byte[], byte[])
▶ struct encoded_method method[3]	public java.lang.String cc.gnaixx.samp.core.EntranceImpl.getStaticFields()
▶ struct encoded_method method[4]	public byte[] cc.gnaixx.samp.core.EntranceImpl.hex2Byte(java.lang.String)
▶ struct encoded_method method[5]	public java.lang.String cc.gnaixx.samp.core.EntranceImpl.md5(java.lang.String)

Implementation: Read the code offset of the first function and change the next function offset to the first value.

To achieve results:


jadx-gui - samp.dex
elp
cc.gnaixx.samp.core.EntranceImpl
jadx-gui - hidex-samp.dex
p
cc.gnaixx.samp.core.EntranceImpl

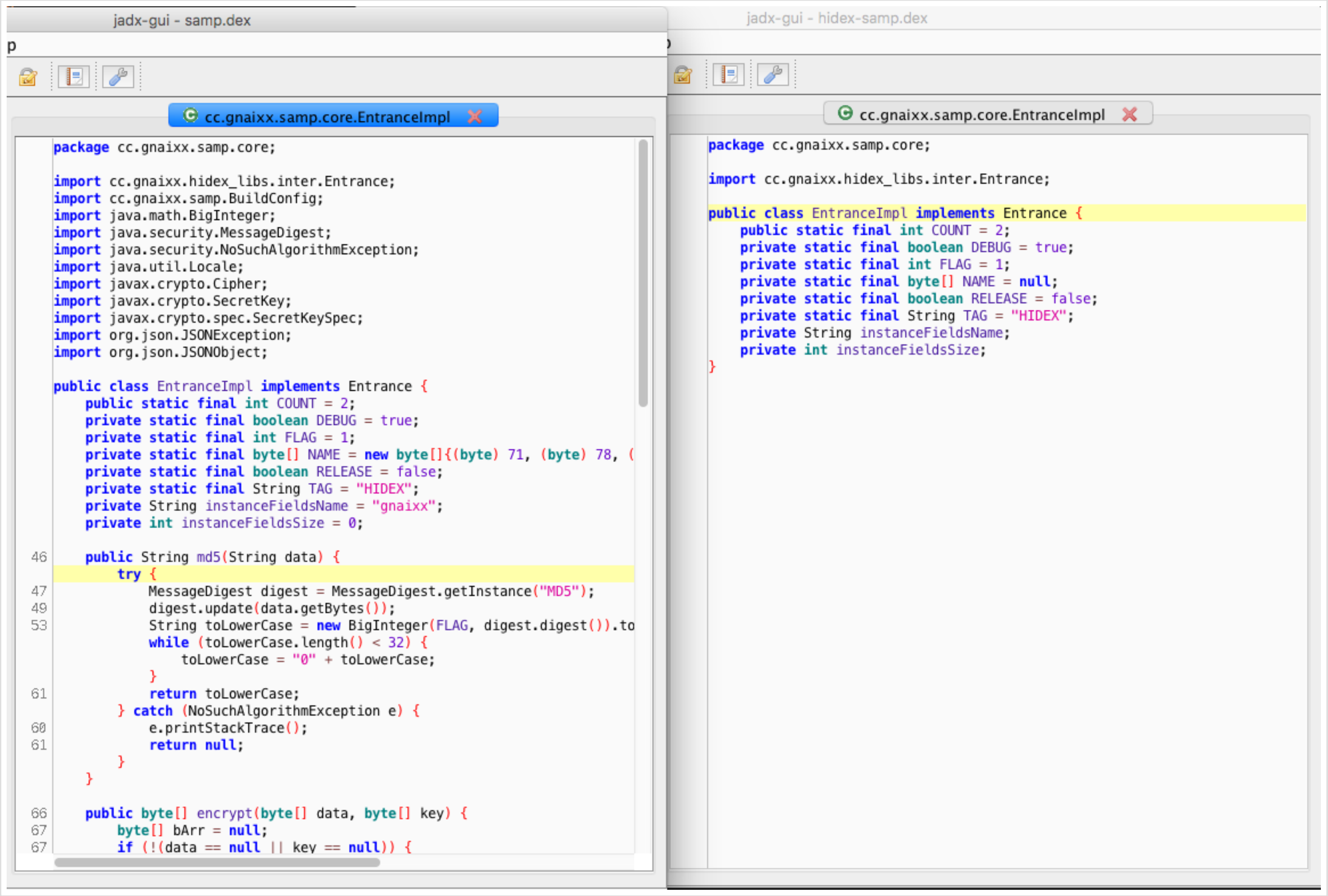
```
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;
import cc.gnaixx.samp.BuildConfig;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Locale;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.json.JSONException;
import org.json.JSONObject;

public class EntranceImpl implements Entrance {
    public static final int COUNT = 2;
    private static final boolean DEBUG = true;
    private static final int FLAG = 1;
    private static final byte[] NAME = new byte[]{(byte) 71, (byte) 78, (
    private static final boolean RELEASE = false;
    private static final String TAG = "HIDEX";
    private String instanceFieldsName = "gnaixx";
    private int instanceFieldsSize = 0;

    public String md5(String data) {
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            digest.update(data.getBytes());
            String toLowerCase = new BigInteger(FLAG, digest.digest()).to
            while (toLowerCase.length() < 32) {
                toLowerCase = "0" + toLowerCase;
            }
            return toLowerCase;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }

    public byte[] encrypt(byte[] data, byte[] key) {
        byte[] bArr = null;
        if (!(data == null || key == null)) {
            try {
                SecretKey ks = new SecretKeySpec(key, "AES");
                Cipher ciph = Cipher.getInstance("AES");
                ciph.init(FLAG, ks);
                bArr = byte2Hex(ciph.doFinal(data)).getBytes("utf-8");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return bArr;
    }

    public byte[] decrypt(byte[] data, byte[] key) {
        byte[] bArr = null;
        if (!(data == null || key == null)) {
            try {
                SecretKey ks = new SecretKeySpec(key, "AES");
                Cipher ciph = Cipher.getInstance("AES");
                ciph.init(COUNT, ks);
                bArr = ciph.doFinal(hex2Byte(new String(data, "utf-8")));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return bArr;
    }

    public String byte2Hex(byte[] buf) {
        if (buf == null) {
            return BuildConfig.FLAVOR;
        }
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < buf.length; i += FLAG) {
            String hex = Integer.toHexString(buf[i] & 255);
            if (hex.length() == FLAG) {
                hex = '0' + hex;
            }
```

```
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;
import cc.gnaixx.samp.BuildConfig;
import java.util.Locale;

public class EntranceImpl implements Entrance {
    public static final int COUNT = 2;
    private static final boolean DEBUG = true;
    private static final int FLAG = 1;
    private static final byte[] NAME = new byte[]{(byte) 71, (byte) 78, (
    private static final boolean RELEASE = false;
    private static final String TAG = "HIDEX";
    private String instanceFieldsName = "gnaixx";
    private int instanceFieldsSize = 0;

    public String byte2Hex(byte[] buf) {
        if (buf == null) {
            return BuildConfig.FLAVOR;
        }
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < buf.length; i += FLAG) {
            String hex = Integer.toHexString(buf[i] & 255);
            if (hex.length() == FLAG) {
                hex = '0' + hex;
            }
            sb.append(hex.toUpperCase(Locale.getDefault()));
        }
        return sb.toString();

    public byte[] decrypt(byte[] buf, byte[] bArr) {
        if (bArr == null) {
            return BuildConfig.FLAVOR;
        }
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < bArr.length; i += FLAG) {
            String hex = Integer.toHexString(bArr[i] & 255);
            if (hex.length() == FLAG) {
                hex = '0' + hex;
            }
            sb.append(hex.toUpperCase(Locale.getDefault()));
        }
        return sb.toString();

    public byte[] encrypt(byte[] buf, byte[] bArr) {
        if (bArr == null) {
            return BuildConfig.FLAVOR;
        }
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < bArr.length; i += FLAG) {
            String hex = Integer.toHexString(bArr[i] & 255);
            if (hex.length() == FLAG) {
                hex = '0' + hex;
            }
            sb.append(hex.toUpperCase(Locale.getDefault()));
        }
        return sb.toString();
    }

    public String getStaticFields() {
        if (this == null) {
            return BuildConfig.FLAVOR;
        }
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < this.length; i += FLAG) {
            String hex = Integer.toHexString(this[i] & 255);
            if (hex.length() == FLAG) {
                hex = '0' + hex;
            }
            sb.append(hex.toUpperCase(Locale.getDefault()));
        }
        return sb.toString();
    }

    public byte[] hex2Byte(String buf) {
        if (buf == null) {
            return BuildConfig.FLAVOR;
```

3 function hiding

Class_def -> class_data -> virtual_methods_size and **class_def -> class_data -> direct_methods_size** records the number of functions in the class definition, or 0 if no function is defined. So as long as the value is changed to 0, the function definition will be hidden.

To achieve results:


jadx-gui - samp.dex
p
cc.gnaixx.samp.core.EntranceImpl
jadx-gui - hidex-samp.dex
cc.gnaixx.samp.core.EntranceImpl

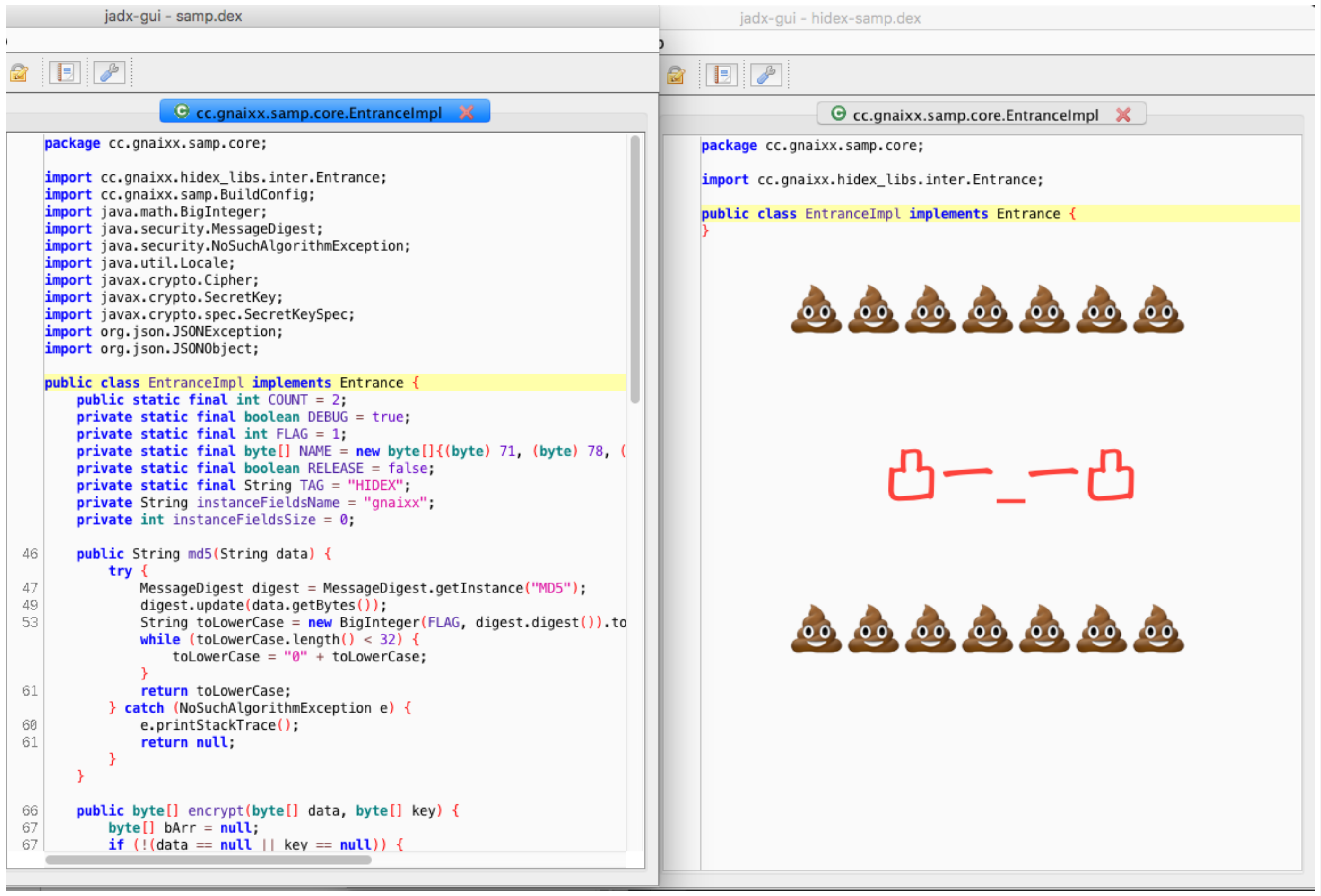
```
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;
import cc.gnaixx.samp.BuildConfig;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Locale;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.json.JSONException;
import org.json.JSONObject;

public class EntranceImpl implements Entrance {
    public static final int COUNT = 2;
    private static final boolean DEBUG = true;
    private static final int FLAG = 1;
    private static final byte[] NAME = new byte[]{(byte) 71, (byte) 78, (
    private static final boolean RELEASE = false;
    private static final String TAG = "HIDEX";
    private String instanceFieldsName = "gnaixx";
    private int instanceFieldsSize = 0;

    public String md5(String data) {
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            digest.update(data.getBytes());
            String toLowerCase = new BigInteger(FLAG, digest.digest()).to
            while (toLowerCase.length() < 32) {
                toLowerCase = "0" + toLowerCase;
            }
            return toLowerCase;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }

    public byte[] encrypt(byte[] data, byte[] key) {
        byte[] bArr = null;
        if (!(data == null || key == null)) {
```

```
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;

public class EntranceImpl implements Entrance {
    public static final int COUNT = 2;
    private static final boolean DEBUG = true;
    private static final int FLAG = 1;
    private static final byte[] NAME = null;
    private static final boolean RELEASE = false;
    private static final String TAG = "HIDEX";
    private String instanceFieldsName;
    private int instanceFieldsSize;
}
```

4 class definition hide

Class_def -> class_data_off holds the class-defined offset address, which is the **class_def -> class_data** address. If this value is 0 then all implementations will be hidden. After hiding, everything hidden in the class definition is hidden including member variables, member functions, static variables, and static functions.

To achieve results:


jadx-gui - samp.dex
cc.gnaixx.samp.core.EntranceImpl
jadx-gui - hidex-samp.dex
cc.gnaixx.samp.core.EntranceImpl
💩💩💩💩💩💩💩
凸—_—凸
💩💩💩💩💩💩💩

```java
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;
import cc.gnaixx.samp.BuildConfig;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Locale;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.json.JSONException;
import org.json.JSONObject;

public class EntranceImpl implements Entrance {
    public static final int COUNT = 2;
    private static final boolean DEBUG = true;
    private static final int FLAG = 1;
    private static final byte[] NAME = new byte[]{(byte) 71, (byte) 78, (
    private static final boolean RELEASE = false;
    private static final String TAG = "HIDEX";
    private String instanceFieldsName = "gnaixx";
    private int instanceFieldsSize = 0;

    public String md5(String data) {
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            digest.update(data.getBytes());
            String toLowerCase = new BigInteger(FLAG, digest.digest()).to
            while (toLowerCase.length() < 32) {
                toLowerCase = "0" + toLowerCase;
            }
            return toLowerCase;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }

    public byte[] encrypt(byte[] data, byte[] key) {
        byte[] bArr = null;
        if (!(data == null || key == null)) {
```

```java
package cc.gnaixx.samp.core;

import cc.gnaixx.hidex_libs.inter.Entrance;

public class EntranceImpl implements Entrance {
}
```

0x03 data read

The above chapter mainly introduces the principle of function realization. Next, we will introduce the specific implementation. To achieve modify **class_def** in the field, we must first resolve the entire file structure dex, but of course you can only field we need. The dex structure I defined in the tool is as follows, because the **class_def** structure is more complex and therefore has a package definition:

```
→ tree -L 2
.
├── DexFile.java
├── FieldIds.java
├── Header.java
├── MapList.java
├── MethodIds.java
├── ProtoIds.java
├── StringIds.java
├── TypeIds.java
└── cladef
    ├── ClassData.java
    ├── ClassDefs.java
    ├── Code.java
    ├── EncodedField.java
    ├── EncodedMethod.java
    ├── EncodedValue.java
    └── StaticValues.java
```

Perhaps you might doubt, we need only modify the function achieved when **class_def** why do you need to read **string_ids** these segments. This is because, as mentioned above **class_def -> class_idx** save actually **type_ids** the serial number, and **type_ids** saved is **string_ids** serial number.

For flexible configuration, we only need to configure the class name to be hidden when running the tool. For example, we need to hide the implementation of a certain class. hack_me_size: cc.gnaixx.samp.core.EntranceImpl The specific implementation of the configuration file is described in the next section.

DexFile.java defines the entire dex file structure. The implementation is relatively simple. Only one **read (byte[] dexBuff)** function reads the entire dex file format.

DexFile.java:

```
Public class DexFile {
  Public static final int HEADER_LEN = 0x70 ;

  Public Header header;
  Public StringIds stringIds;
  Public TypeIds typeIds;
  Public ProtoIds protoIds;
  Public FieldIds fieldIds;
  Public MethodIds methodIds;
  Public ClassDefs classDefs;
  Public MapList mapList;

  //reader dex
  Public void read ( byte [] dexBuff) {
    //read header
    Byte [] headerbs = subdex(dexBuff, 0 , HEADER_LEN);
    Header = new Header(headerbs);

    //read string_ids
    stringIds = new StringIds(dexBuff, header.stringIdsOff, header.stringIdsSize);

    //read type_ids
    typeIds = new TypeIds(dexBuff, header.typeIdsOff, header.typeIdsSize);
    //read proto_ids
    protoIds = new ProtoIds(dexBuff, header.protoIdsOff, header.protoIdsSize);

    //read field_ids
    fieldIds = new FieldIds(dexBuff, header.fieldIdsOff, header.fieldIdsSize);

    //read method_ids
methodIds       = new MethodIds(dexBuff, header.methodIdsOff, header.methodIdsSize);

    //read class_defs
    classDefs = new ClassDefs(dexBuff, header.classDefsOff, header.classDefsSize);

    //read map_list
    mapList = new MapList(dexBuff, header.mapOff);
  }
}
```

The first step is to read the **header** because it stores the offset addresses and number of other sections.

Header.java:

```
Public class Header {

  Public byte [] magic = new byte [MAGIC_LEN];
  Public int     checksum;
  Public byte [] signature = new byte [SIGNATURE_LEN];
  Public int     fileSize;
  Public int     headerSize;
  Public int     endianTag;
  Public int     linkSize;
  Public int     linkOff;
  Public int     mapOff;
  Public int     stringIdsSize;
  Public int     stringIdsOff;
  Public int     typeIdsSize;
  Public int     typeIdsOff;
  Public int     protoIdsSize;
  Public int     protoIdsOff;
  Public int     fieldIdsSize;
  Public int     fieldIdsOff;
  Public int     methodIdsSize;
  Public int     methodIdsOff;
  Public int     classDefsSize;
  Public int     classDefsOff;
  Public int     dataSize;

  Public int     dataOff
```

```
    Public int    dataOff;

    Public Header ( byte [] headerBuff) {
        Reader reader = new Reader(headerBuff, 0 );
        This .magic = reader.subdex(MAGIC_LEN);
        This .checksum = reader.readUint();
        This .signature = reader.subdex(SIGNATURE_LEN);
        //......
    }

    Public void write ( byte [] dexBuff) {
        Writer writer = new Writer(dexBuff, 0 );
        Writer.replace(magic, MAGIC_LEN);
        writer.writeUint(checksum);
        Writer.replace(signature, SIGNATURE_LEN);
        //.....
    }
}
```

It is relatively simple to know the offset address and the number of each section to read, such as the read of the **string_ids** section.

StringIds.java:

```
Public class StringIds {

    Class StringId {
        Int dataOff;          // string offset location
        Uleb128 utf16Size;    //string length
        Byte data[];          // string data

        Public StringId ( int dataOff, Uleb128 uleb128, byte [] data) {
            This .dataOff = dataOff;
            This .utf16Size = uleb128;
            This .data = data;
        }
    }

    StringId stringIds[];

    Public StringIds ( byte [] dexBuff, int off, int size) {
        This .stringIds = new StringId[size];

        Reader reader = new Reader(dexBuff, off);
        For ( int i = 0 ; i < size; i++) {
            Int dataOff = reader.readUint();
            Uleb128 utf16Size = getUleb128(dexBuff, dataOff);
            Byte [] data = subdex(dexBuff, dataOff + 1 , utf16Size.getVal());
            StringId stringId = new StringId(dataOff, utf16Size, data);
            stringIds[i] = stringId;
        }
    }

    Public String getData ( int id) {
        //return "(" + id + ")" + new String(stringIds[id].data);
        Return  new String(stringIds[id].data);
    }
}
```

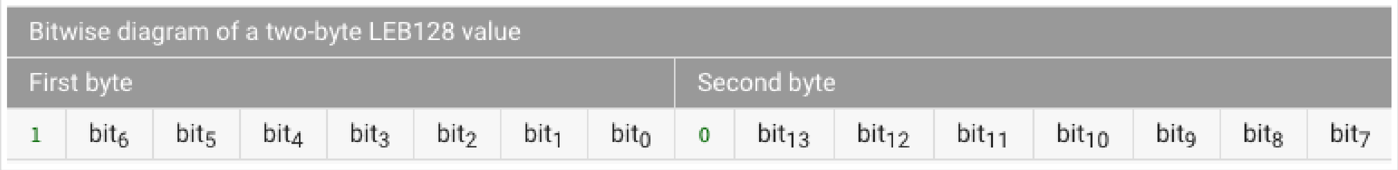
Reading area and other sections of **string_ids** similar, but **class_def** section zone structure is more complex, it may be reading too much trouble. But in fact, we do not have many values to use, just pay attention to those fields just fine.

ClassDefs.java:

```
Public  class  ClassDefs {

    Public  class  ClassDef {
        Public  int        classIdx;       //class type, corresponding to type_ids
        Public  int        accessFlags;     //access type, enum
        Public  int        superclassIdx;   //supperclass type, corresponding to type_ids
        Public  int        interfacesOff;   //Interface offset, corresponding to type_list
        Public  int        sourceFileIdx;   // source file name, corresponding string_ids
        Public  int        annotationsOff; //class annotations, location in data area, corresponding an
        Public HackPoint classDataOff;    / / class specific data used in the data area, the format is c
        Public HackPoint staticValueOff; / / Located in the data area, the format is encoded_array_i

        Public StaticValues staticValues;   // if classDataOff is not 0
        Public ClassData classData;      // staticValueOff is not 0 exists

        Public  ClassDef ( int classIdx, int accessFlags,
                     Int superclassIdx, int interfacesOff,
                     Int sourceFileidx, int annotationsOff,
                     HackPoint classDataOff, HackPoint staticValueOff) {
            This .classIdx = classIdx;
            This .accessFlags = accessFlags;
            This .superclassIdx = superclassIdx;
            This .interfacesOff = interfacesOff;
            This .sourceFileIdx = sourceFileidx;
            This .annotationsOff = annotationsOff;
            This .classDataOff = classDataOff;
            This .staticValueOff = staticValueOff;
        }

        Public  void  setClassData (ClassData classData) {
            This .classData = classData;
        }

        Public  void  setStaticValue (StaticValues staticValues) {
            This .staticValues = staticValues;
        }
    }

    Int      offset; //offset position
    Int      size;    //size

    Public ClassDef classDefs[];

    Public  ClassDefs ( byte [] dexBuff, int off, int size)  {
        This .offset = off;
        This .size = size;

        Reader reader = new Reader(dexBuff, off);
        classDefs = new ClassDef[size];
        For ( int i = 0 ; i < size; i++) {
            Int classIdx = reader.readUint();
            Int accessFlags = reader.readUint();
            Int superclassIdx = reader.readUint();
            Int interfacesOff = reader.readUint();
            Int sourcFileIdx = reader.readUint();
            Int annotationOff = reader.readUint();

            HackPoint classDataOff = new HackPoint(HackPoint.UINT, reader.getOff(), reader.readU
            HackPoint staticValueOff = new HackPoint(HackPoint.UINT, reader.getOff(), reader.read

            ClassDef classDef = new ClassDef(
                    classIdx, accessFlags,
                    superclassIdx, interfacesOff,
                    sourcFileIdx, annotationOff,
                    classDataOff, staticValueOff);

            If (staticValueOff.value != 0 ){
                Reader reader1 = new Reader(dexBuff, staticValueOff.value);
                Uleb128 staticSize = reader1.readUleb128();
                StaticValues staticValues = new StaticValues(staticSize);
                classDef.setStaticValue(staticValues);
            }

            If (classDataOff.value != 0 ){
                classDef.setClassData( new ClassData(dexBuff, classDataOff.value));
            }
            classDefs[i] = classDef;
        }
    }

    Public  void  write ( byte [] dexBuff) {
        Writer writer = new Writer(dexBuff, offset);
        For ( int i= 0 ; i<size; i++){
            ClassDef classDef = classDefs[i];
            writer.writeUint(classDef.classIdx);
            writer.writeUint(classDef.accessFlags);
            writer.writeUint(classDef.superclassIdx);
            writer.writeUint(classDef.interfacesOff);
            writer.writeUint(classDef.sourceFileIdx);
            writer.writeUint(classDef.annotationsOff);

            writer.writeUint(classDef.classDataOff.value);
            If (classDef.classDataOff.value != 0 ){
                classDef.classData.write(dexBuff, classDef.classDataOff.value);
            }

            writer.writeUint(classDef.staticValueOff.value);
            If (classDef.staticValueOff.value != 0 ){
                // Do not do it temporarily
            }
        }
    }
}
```

Here we need to introduce one of the dex-specific data types **LEB128** officially introduced as follows:

> LEB128 ("Little-Endian Base 128") is a variable-length encoding for signed or unsigned integer quantities. The format was borrowed from the DWARF3 specification. In a .dex file, LEB128 is only ever used to encode 32-bit quantity. .
>
> Each remaining bit of encoded sequence consists of one to five bytes, which together represents a single 32-bit value. Each remaining has its most significant bit set except for the final byte in the sequence, which has its most significant bit clear. Of each byte are payload, with the least significant seven bits of the quantity in the first byte, the next seven in the second byte and so on. In the case of a signed LEB128 (sleb128), the most significant payload bit of the final Byte in the sequence is sign-extended to produce the final value. In the unsigned case (uleb128), any bits not represents represented are interpret as 0.



Bitwise diagram of a two-byte LEB128 value															
First byte								Second byte							
1	bit$_6$	bit$_5$	bit$_4$	bit$_3$	bit$_2$	bit$_1$	bit$_0$	0	bit$_{13}$	bit$_{12}$	bit$_{11}$	bit$_{10}$	bit$_9$	bit$_8$	bit$_7$

That is, LEB128 is an indefinite-length encoding based on 1 Byte. If the highest bit of the first Byte is 1, it means that the next Byte is needed to describe until the highest bit of the last Byte is 0. The remaining Bits of each Byte are used to represent the data.

The code uses ULeb128.java (unsigned) to indicate this structure. By analyzing the Android source code Leb128.h, we can see that LEB128 represents an indefinite long format, but only 4 bytes are used in Android, so only need to use int That' s it.

ULeb128.java:

```
Public class Uleb128 {
  Byte [] realVal; // stored byte data
  Int val; // Integer data represented

  Public Uleb128 ( byte [] realVal, int val) {
    This .realVal = realVal;
    This .val = val;
  }

  Public int getSize () {
    Return this .realVal.length;
  }

  Public int getVal () {
    Return this .val;
  }

  Public byte [] getRealVal(){
    Return this .realVal;
  }
}
```

Bytes to ULEB128:

```
//Reader.java
Public Uleb128 readUleb128 () {
    Int value = 0 ;
    Int count = 0 ;
    Int count = 0 ;
    Byte realVal[] = new  byte [ 4 ];
    Boolean flag = false ;
    Do {
      Flag = false ;
      Byte seg = buffer[offset];
      If ((seg & 0x80 ) == 0x80 ) { // high 8 bits are 1
        Flag = true ;
      }
      Seg = ( byte ) (seg & 0x7F );
      Value += seg << ( 7 * count);
      realVal[count] = buffer[offset];
      Count++;
      Offset++;
    } while (flag);
    Return  new Uleb128(BufferUtil.subdex(realVal, 0 , count), value);
  }
```

Integer to ULEB128:

```
//Trans.java
Public  static Uleb128 intToUleb128 ( int val)  {
    Byte [] realVal = new  byte [] { 0x00 , 0x00 , 0x00 , 0x00 }; //int has a maximum length of 4
    Int bk = val;
    Int len = 0 ;
    For ( int i = 0 ; i < realVal.length; i++) {
      Len = i + 1 ; // The minimum length is 1
      realVal[i] = ( byte ) (val & 0x7F ); //Get the low 7-bit value
      If (val > ( 0x7F )) {
        realVal[i] |= 0x80 ; //the high bit is 1 plus
      }
      Val = val >> 7 ;
      If (val <= 0 ) break ;
    }
    Uleb128 uleb128 = new Uleb128(BufferUtil.subdex(realVal, 0 , len), bk);
    Return uleb128;
  }
```

0x04 HackPoint format

HackPoint represents the modified data structure. All the fields to be modified are represented by the **HackPoint** type in the **code** . **The HackPoint** type has three fields: type, offset, and value, all of which are of type int: type, offset address, and original value. There are three types of **uint (unsigned int), ushort (unsigned short 2byte), and uleb128** . All three data are sufficient with int storage.

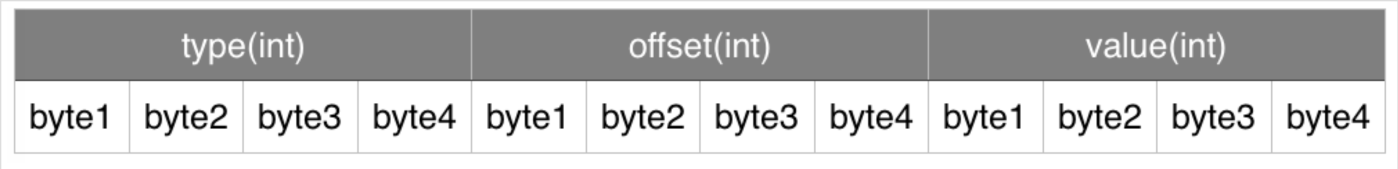
HackPoint.java:

```
Public class HackPoint implements Cloneable {

  Public static final int UINT = 0x01 ;
  Public static final int USHORT = 0x02 ;
  Public static final int ULEB128 = 0x03 ;

  Public int type;      //data type
  Public int offset;    // offset address
  Public int value;     //original value

  Public HackPoint ( int type, int offset, int val) {
    This .type = type;
    This .offset = offset;
    This .value = val;
  }

  @Override
  Public HackPoint clone () {
    HackPoint hp = null ;
    Try {
      Hp = (HackPoint) super .clone();
    } catch (CloneNotSupportedException e) {
      e.printStackTrace();
    }
    Return hp;
  }
}
```

After the modification, all **HackPoint** data will be written at the end of the dex file. The end of the dex file is the **map_list** section. The data format is:

```
Struct map_list{
    Ushort type;
    Ushort unused;
    Uint size;
    Uint offset;
};
```

It is exactly 12 byte, so the format of the **HackPoint** write dex file is:



type(int)				offset(int)				value(int)			
byte1	byte2	byte3	byte4	byte1	byte2	byte3	byte4	byte1	byte2	byte3	byte4

0x05 configuration file

The definition of the configuration file is relatively simple to look at the example to know:

```
################################################### ##
# hack_class: Hide class definition
# hack_sf_val: Hide static variables
# hack_me_size: Hide methods
# hack_me_def: repeat function definition (whichever comes first)
################################################### ###


# Hide static variable values
Hack_sf_val: cc.gnaixx.samp.core.EntranceImpl

# Repeat function definition (whichever comes first)
Hack_me_def: cc.gnaixx.samp.core.EntranceImpl

#Hide function implementation
Hack_me_size: cc.gnaixx.samp.core.EntranceImpl

# Hide the entire class implementation
Hack_class: cc.gnaixx.samp.core.EntranceImpl cc.gnaixx.samp.BuildConfig
```

When multiple classes need to implement the same function, they only need to be separated by spaces.

Profile reading code:

```
Public  static Map<String, List<String>> readConfig(String path) {
  Try {
    Map<String, List<String>> config = new HashMap<>();
    FileReader fr = new FileReader(path);
    BufferedReader br = new BufferedReader(fr);
    String line;
    While ((line = br.readLine()) != null ) {

      If (!line.startsWith( "#" ) && !line.equals( "" )) {
        String conf[] = line.split( ":" );
        If (conf.length != 2 ) {
          Log( "warning" , "error config at :" + line);
          System.exit( 0 );
        }

        String key = conf[ 0 ];
        String values[] = conf[ 1 ].split( " " );
```

```
                List<String> valueList = new ArrayList<>();
                For ( int i = 0 ; i < values.length; i++) {
                    If (values[i] != null && !values[i].equals( "" )) {
                        valueList.add(values[i]);
                    }
                }
                Config.put(key, valueList);
            }
        }
        Fr.close();
        Br.close();
        Return config;
    } Catch (Exception e) {
        e.printStackTrace();
    }
    Return  null ;
}
```

0x06 dex confusion hidden

The dex file obfuscation and hiding mainly includes three steps:

1. Modify the **HackPoint** and save it to the end of the dex file
2. Fix Header

1. Modify HackPoint

Traverse the **class_def_item** by the configuration class in the obtained configuration file :

```
//Find the class location of the configuration file
Private  void  seekHP (ClassDefs.ClassDef[] classDefItem, List<String> conf, String type, SeekC
    If (conf == null ) {
        Return ;
    }
    For ( int i = 0 ; i < conf.size(); i++) {
        String classname = conf.get(i);
        Boolean isDef = false ;
        For ( int j = 0 ; j < classDefItem.length; j++) {
            String className = dexFile.typeIds.getString(dexFile, classDefItem[j].classIdx); //Find c
            className = pathToPackages(className); //Get the class name
            If (className.equals(classname)) {
                callBack.doHack(classDefItem[j], this .hackPoints); //specific operation
                Log(type, conf.get(i));
                isDef = true ;
            }
        }
        If (isDef == false ) {
            Log( "warning" , "con' t find class:" + classname);
        }
    }
}

// Specific callback handling
Interface SeekCallBack {
    Void doHack (ClassDefs.ClassDef classDefItem, List<HackPoint> hackPoints) ;
}
```

Hide the static variable value:

```
// Hide static variable initialization
Private void hackSfVal (ClassDefs.ClassDef[] classDefItem, List<String> conf) {
    seekHP(classDefItem, conf, Constants.HACK_SF_VAL, new SeekCallBack() {
        @Override
        Public void doHack (ClassDefs.ClassDef classDefItem, List<HackPoint> hackPoints) {
            HackPoint point = classDefItem.staticValueOff.clone();  //Get static variable data offset
            hackPoints.add(point);                        //Add modify point
            classDefItem.staticValueOff.value = 0 ;          // change the offset of the static variable to 0 (hide as
        }
    });
}
```

The function is repeatedly defined:

```
// repeat function definition
Private void hackMeDef (ClassDefs.ClassDef[] classDefItem, List<String> conf) {
    seekHP(classDefItem, conf, Constants.HACK_ME_DEF, new SeekCallBack() {
        @Override
        Public void doHack (ClassDefs.ClassDef classDefItem, List<HackPoint> hackPoints) {
            // The first one is the default

            Int virtualMeSize = classDefItem.classData.virtualMethodsSize.value;
            Int virtualMeCodeOff = 0 ;
            For ( int i = 0 ; i < virtualMeSize; i++) {
                If (i == 0 ) {
                    virtualMeCodeOff = classDefItem.classData.virtualMethods[i].codeOff.value;
                } Else {
                    HackPoint point = classDefItem.classData.virtualMethods[i].codeOff.clone();
                    hackPoints.add(point);
                    classDefItem.classData.virtualMethods[i].codeOff.value = virtualMeCodeOff;
                }
            }
        }
    });
}
```

Function hiding:

```
// Hide function definition
Private  void  hackMeSize (ClassDefs.ClassDef[] classDefItem, List<String> conf) {
    seekHP(classDefItem, conf, Constants.HACK_ME_SIZE, new SeekCallBack() {
        @Override
        Public  void  doHack (ClassDefs.ClassDef classDefItem, List<HackPoint> hackPoints)  {
            HackPoint directPoint = classDefItem.classData.directMethodsSize.clone(); //Also need to change
            HackPoint virtualPoint = classDefItem.classData.virtualMethodsSize.clone();
            hackPoints.add(directPoint);
            hackPoints.add(virtualPoint);
            classDefItem.classData.directMethodsSize.value = 0 ;
            classDefItem.classData.virtualMethodsSize.value = 0 ;
        }
    });
}
```

Hidden class:

```
// Hide static variable initialization
Private  void  hackSfVal (ClassDefs.ClassDef[] classDefItem, List<String> conf)  {
    seekHP(classDefItem, conf, Constants.HACK_SF_VAL, new SeekCallBack() {
        @Override
        Public  void  doHack (ClassDefs.ClassDef classDefItem, List<HackPoint> hackPoints)  {
            HackPoint point = classDefItem.staticValueOff.clone();   //Get static variable data offset
            hackPoints.add(point);                    //Add modify point
            classDefItem.staticValueOff.value = 0 ;          // change the offset of the static variable to 0 (hide as
        }
    });
}
```

Add HackPoint data to dex file:

```
 //Keep modified information
Private  void  appendHP ()  {
    Byte [] pointsBuff = new  byte []{};
    For ( int i = 0 ; i < hackPoints.size(); i++) {
```

```
    Byte [] pointBuff = hackpToBin(hackPoints.get(i));
    pointsBuff = BufferUtil.append(pointsBuff, pointBuff, pointBuff.length);
  }
  dexBuff = BufferUtil.append(dexBuff, pointsBuff, pointsBuff.length);
}

//hackPoint to binary
Public  static  byte [] hackpToBin(HackPoint point) {
  ByteBuffer bb = ByteBuffer.allocate( 4 * 3 );
  Bb.put(intToBin_Lit(point.type));
  Bb.put(intToBin_Lit(point.offset));
  Bb.put(intToBin_Lit(point.value));
  Return bb.array();
}

// little endian binary
Public  static  byte [] intToBin_Lit( int integer){
  Byte [] bin = new  byte []{
      ( byte ) ((integer >> 0 ) & 0xFF ),
      ( byte ) ((integer >> 8 ) & 0xFF ),
      ( byte ) ((integer >> 16 ) & 0xFF ),
      ( byte ) ((integer >> 24 ) & 0xFF )
  };
  Return bin;
}
```

Dex files are saved as little-endian data

2. Repair Header

There are three data repaired in **Header** :

1. File length
2. Checksum
3. Signature

Modify the code:

```
//Modify the header
Private  void  hackHeader () {
  //Modify the file length
  Header header = dexFile.header;
  header.fileSize = this .dexBuff.length;
  Header.write(dexBuff); //Need to modify the file length before calculating signature checksur
```

```
    // Repair signature check
    Log( "old_signature" , binToHex(dexFile.header.signature));
    Byte [] signature = signature (dexBuff, SIGNATURE_LEN + SIGNATURE_OFF);
    Header.signature = signature;
    Log( " new_signature " , binToHex(signature));
    Header.write(dexBuff); // need to write sinature before calculating the checksum, convex
    // Fix the checksum check
    Log( "old_checksum" , intToHex(dexFile.header.checksum));
    Int checksum = checksum_Lit(dexBuff, CHECKSUM_LEN + CHECKSUM_OFF);
    Header.checksum = checksum;
    Log( "new_checksum" , intToHex(checksum));
    Header.write(dexBuff);
}

//Calculate signature
Public  static  byte [] signature( byte [] data, int off) {
    Int len = data.length - off;
    Byte [] signature = SHA1(data, off, len);
    Return signature;
}
//sha1 algorithm
Public  static  byte [] SHA1( byte [] decript, int off, int len) {
    Try {
        MessageDigest digest = MessageDigest.getInstance( "SHA-1" );
        Digest.update(decript, off, len);
        Byte messageDigest[] = digest.digest();
        Return messageDigest;
    } Catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    Return  null ;
}

//Calculate the checksum value
Public  static  int  checksum_Lit ( byte [] data, int off)  {
    Byte [] bin = checksum_bin(data, off);
    Int value = 0 ;
    For ( int i = 0 ; i < UINT_LEN; i++) {
        Int seg = bin[i];
        If (seg < 0 ) {
            Seg = 256 + seg;
        }
        Value += seg << ( 8 * i);
    }
    Return value;
}
//Calculate checksum
Public  static  byte [] checksum_bin( byte [] data, int off) {
    Int len = data.length - off;
    Adler32 adler32 = new Adler32();
    Adler32.reset();
    Adler32.update(data, off, len);
    Long checksum = adler32.getValue();
    Byte [] checksumbs = new  byte []{
        ( byte ) checksum,
        ( byte ) (checksum >> 8 ),
        ( byte ) (checksum >> 16 ),
        ( byte ) (checksum >> 24 )};
    Return checksumbs;
}
```

This part of code address: **HidexHandle.java**

0x07 dex restore

Relative to the encryption and decryption process is much simpler, as long as one by one according to **HackPoint** data repair just fine. Here is a brief explanation of the repair procedure:

1. Reading Header **map_list** offset and the number, as **HackPoint** data stored in **map_list** after
2. Read **HackPoint** data and repair dex file
3. Fix **file_size, checksum, signature in** Header

Java implementation

Repair key source code:

```
// Fix dex file
Public  byte [] redex() {
  Int mapOff = getUint(dexBuff, MAP_OFF_OFF); //Get map_off
  Int mapSize = getUint(dexBuff, mapOff); //Get map_size
  Int hackInfoStart = mapOff + UINT_LEN + (mapSize * MAP_ITEM_LEN); //Get the hackinfo st
  Int hackInfoLen = dexBuff.length - hackInfoStart; // Get hackinfo length
  hackInfoBuff = subdex(dexBuff, hackInfoStart, hackInfoLen); //Get hack data
  Int dexLen = dexBuff.length - hackInfoLen;
  dexBuff = subdex(dexBuff, 0 , dexLen); // Truncate the original dex length
  HackPoint[] hackPoints = Trans.binToHackP(hackInfoBuff);   //fix hack
  For ( int i = 0 ; i < hackPoints.length; i++) {
    Log( "hackPoint" , JSON.toJSONString(hackPoints[i]));
    Recovery(hackPoints[i]);
  }
  Byte [] fileSize = intToBin_Lit (dexLen); // fix file length
  Replace(dexBuff, fileSize, FILE_SIZE_OFF, UINT_LEN);
```

```
        Byte [] signature = signature (dexBuff, SIGNATURE_LEN + SIGNATURE_OFF); // repair signat
        Replace(dexBuff, signature, SIGNATURE_OFF, SIGNATURE_LEN);
        Byte [] checksum = checksum_bin(dexBuff, CHECKSUM_LEN + CHECKSUM_OFF); //fix checks
        Replace(dexBuff, checksum, CHECKSUM_OFF, CHECKSUM_LEN);
        Log( "fileSize" , dexLen);
        Log( "signature" , binToHex(signature));
        log("checksum", binToHex_Lit(checksum));
        return this.dexBuff;
    }
    //□原原始□
    private void recovery(HackPoint hackPoint) {
        Writer writer = new Writer(this.dexBuff, hackPoint.offset);
        if (hackPoint.type == HackPoint.USHORT) {
            writer.writeUshort(hackPoint.value);
        }
        else if (hackPoint.type == HackPoint.UINT) {
            writer.writeUint(hackPoint.value);
        }
        else if (hackPoint.type == HackPoint.ULEB128) {
            Uleb128 uleb128 = Trans.intToUleb128(hackPoint.value);
            writer.writeUleb128(uleb128);
        }
    }
```

C++ □□

工具本身就是□了□□安全加固，□□用 java □□意□就小了□多，所以工具包里面的□□我是用 NDK □□的。

修□□□源□□

```
//解密dex
void recode(char* source, uint sourceLen, char* target, uint* targetLen){
    uint mapOff = readUint(source, MAP_OFF_OFF); //□取map_off
    uint mapSize = readUint(source, mapOff); //□取map_size
    LOGD("mapInfo: {map_off:%d, map_size:%d}", mapOff, mapSize);

    uint hackInfoOff = mapOff + UINT_LEN + (mapSize * MAP_ITEM_LEN); //定位hackInfo位置
    uint hackInfoLen = sourceLen - hackInfoOff; //hackInfo□度
    char* hackInfo = (char *) calloc(hackInfoLen, sizeof(char));
    memcpy(hackInfo, source + hackInfoOff, hackInfoLen); //□制hackInfo
    LOGD("hackInfo: {hackInfo_off:%d, hackInfo_len}", hackInfoOff, hackInfoLen);

    uint hackPointSize = hackInfoLen / sizeof(HackPoint); //□取hackPoint□□体
    HackPoint* hackPoints = (HackPoint *) calloc(hackPointSize, sizeof(HackPoint));
    initHP(hackPoints, hackInfo, hackPointSize); //将hockInfo □化□□□体

    *targetLen = hackInfoOff;
    memcpy(target, source, *targetLen); //□□原始□度

    //□□数□
    for(int i=0; i<hackPointSize; i++){
        recoverHP(target, hackPoints[i]);
    }
    LOGD("Recover HackPoint success");

    //修□hearder
    recoverHeader(target, *targetLen);

    Free(hackInfo);
    Free(hackPoints);
}
```

Full source address: **hidex.cpp**

0x09 summary

The overall functionality is still relatively simple, and the code implemented is not very complicated, but these need to be based on the understanding of the dex file format.
Another disadvantage of this tool is the loading problem of dex. DexClassLoad loading dex in Android only supports file path loading, unlike ClassLoad in Java which can support binary stream loading, so there is an encrypted dex cache when loading dex, which is very dangerous. So the point of the next study is that the custom DexClassLoad implementation does not load. (Many security consolidation vendors have long since realized that).
Although the function is not powerful, there are many shortcomings, but it also spent a lot of time on their own research, a little understanding of the dex file format, it is worth it.

# android # dex # hidex

❮ DEX file format analysis

DEX file loading process analysis ❯

5条□□ **□一_一□** 1 **登□**

♡ **推□** 1 **分享** **□分最高**

加入□□...

通□以下方式登□

或注□一□ DISQUS □号 ?

姓名

Jack Yang • 8□月前

□好 能留□□系方式□ 可以加我QQ：2604470550 看□的代□好多□看□□□有□是用的□□□件打□的so文件

回□ • 分享 ›

□一_一□ 管理□ ➦ Jack Yang • 8□月前

□果是分□代□ so 文件可以用 ida 打□□□果是分□格式so dex 都可以用 010 Editor 打□

回□ • 分享 ›

Jack Yang ➦ □一_一□ • 8□月前

大□□真的□急，求指□ 加一下QQ□

回□ • 分享 ›

Jack Yang ➦ □一_一□ • 8□月前

□的□□□件，我□□不能用□□在MAC上□行□□提示我□有安装dx，mac上好像□有dx□□有□照□的方法修改完dex后，可以正常打包，但打包后的apk不能正常安装？ 能加一下QQ□□□□也行：912356442

回□ • 分享 ›

□一_一□ 管理□ • 9□月前

TEST

回□ • 分享 ›

在 □一_一□ 上□有

DEX文件加□流程分□

1条□□ • 9□月前

□一_一□ — TEST

NDK□□-ReleaseStringUTFChars□用的□

2条□□ • 9□月前

□一_一□ — 是的，□□候□□不足