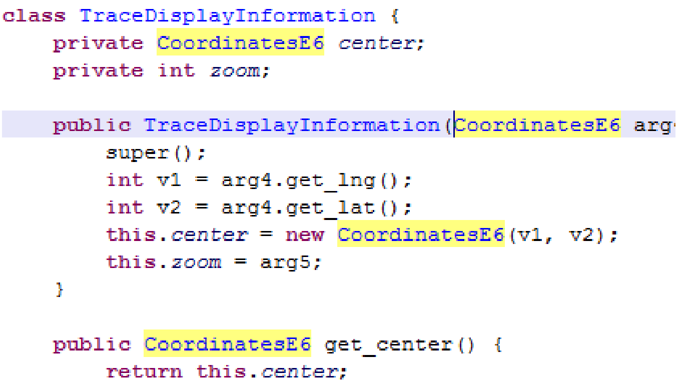
JEB
The Interactive Android Decompiler

JEB is the most powerful Android app decompiler, designed for security professionals. Cut your reverse engineering time and leverage the API for your automation needs.



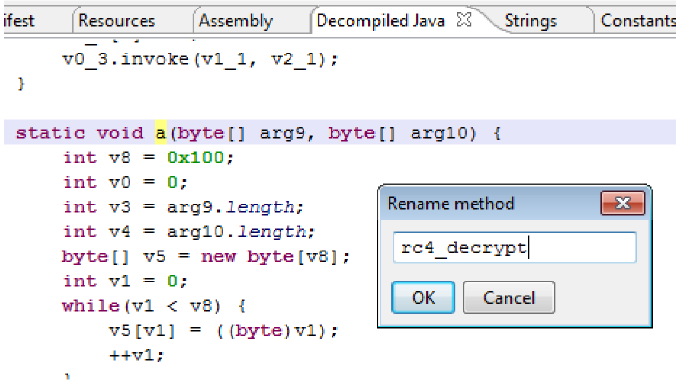
```
class TraceDisplayInformation {
    private CoordinatesE6 center;
    private int zoom;

    public TraceDisplayInformation(CoordinatesE6 arg
        super();
        int v1 = arg4.get_lng();
        int v2 = arg4.get_lat();
        this.center = new CoordinatesE6(v1, v2);
        this.zoom = arg5;
    }

    public CoordinatesE6 get_center() {
        return this.center;
```

POWERFUL

JEB's unique feature is its ability to decompile true Dalvik bytecode to Java source code. No more unreliable dex-to-jar conversion: Our in-house, full-fledged Dalvik decompiler produces clean, structured, and semantically correct code.

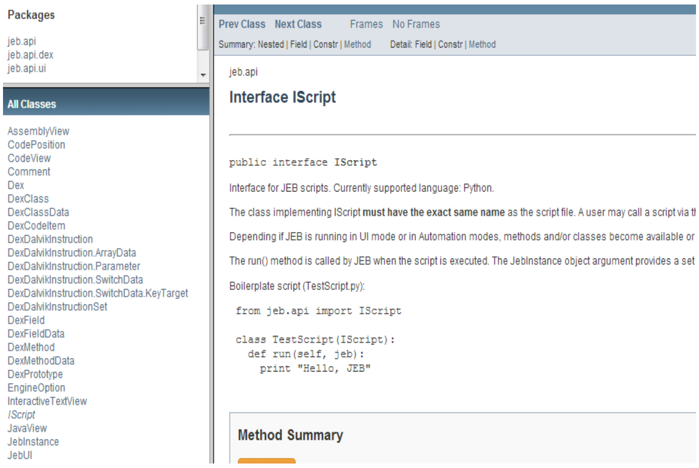
FLEXIBLE

Analysts need flexible tools, especially when they deal with obfuscated or protected pieces of code. JEB's user interface allows you to examine cross-references, rename methods, fields, classes, navigate between code and data, take notes, add comments, etc.



```
        v0_3.invoke(v1_1, v2_1);
    }

    static void a(byte[] arg9, byte[] arg10) {
        int v8 = 0x100;
        int v0 = 0;
        int v3 = arg9.length;
        int v4 = arg10.length;
        byte[] v5 = new byte[v8];
        int v1 = 0;
        while(v1 < v8) {
            v5[v1] = ((byte)v1);
            ++v1;
        }
```

rc4_decrypt


Prev Class
Next Class
Frames
No Frames
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method

Packages

jeb.api
jeb.api.dex
jeb.api.ui

All Classes

AssemblyView
CodePosition
CodeView
Comment
Dex
DexClass
DexClassData
DexCodeItem
DexDalvikInstruction
DexDalvikInstruction.ArrayData
DexDalvikInstruction.Parameter
DexDalvikInstruction.SwitchData
DexDalvikInstruction.SwitchData.KeyTarget
DexDalvikInstructionSet
DexField
DexFieldData
DexMethod
DexMethodData
DexPrototype
EngineOption
InteractiveTextView
IScript
JavaView
JebInstance
JebUI

jeb.api

Interface IScript

```
public interface IScript
```

Interface for JEB scripts. Currently supported language: Python.

The class implementing IScript **must have the exact same name** as the script file. A user may call a script via t

Depending if JEB is running in UI mode or in Automation modes, methods and/or classes become available or

The run() method is called by JEB when the script is executed. The JebInstance object argument provides a set

Boilerplate script (TestScript.py):

```
 from jeb.api import IScript

 class TestScript(IScript):
   def run(self, jeb):
     print "Hello, JEB"
```

Method Summary

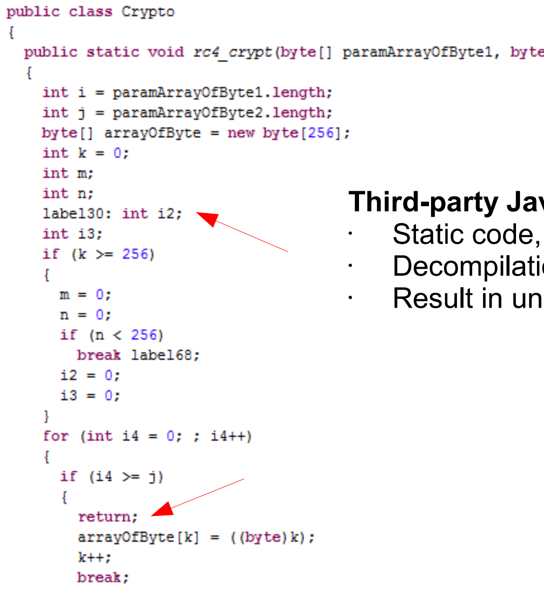
EXTENSIBLE

Leverage the application programming interface (API) to extend JEB with scripts and plugins. Example: Access the AST of decompiled Java code to remove obfuscation layers; Use non-interactive JEB to automate back-end processing.

Languages offered through the API: Python, Java.

JEB
The Interactive Android Decompiler

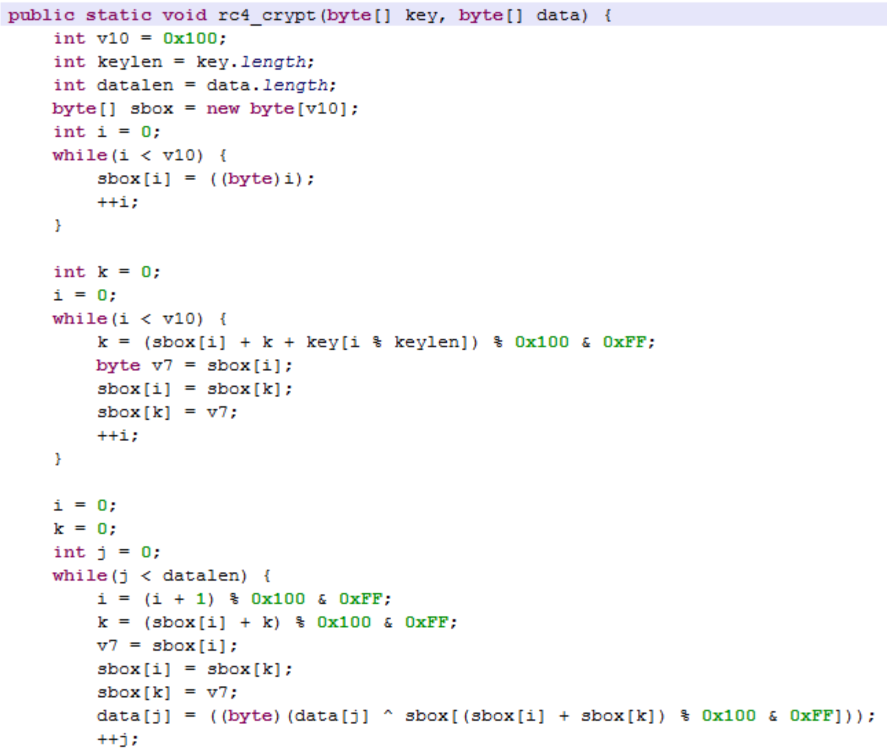
Two major advantages of JEB over existing tools are its **interactivity** and **industrial-grade decompiler output**. They allow reverse engineers to analyze and gradually understand complex pieces of code.



```
public class Crypto
{
  public static void rc4_crypt(byte[] paramArrayOfByte1, byte[] paramArray
  {
    int i = paramArrayOfByte1.length;
    int j = paramArrayOfByte2.length;
    byte[] arrayOfByte = new byte[256];
    int k = 0;
    int m;
    int n;
    label130: int i2;
    int i3;
    if (k >= 256)
    {
      m = 0;
      n = 0;
      if (n < 256)
        break label68;
      i2 = 0;
      i3 = 0;
    }
    for (int i4 = 0; ; i4++)
    {
      if (i4 >= j)
      {
        return;
        arrayOfByte[k] = ((byte)k);
        k++;
        break;
```

Third-party Java decompiler output (left)

- Static code, no interactivity
- Decompilation errors (arrows)
- Result in unreadability and poor usability



```
public static void rc4_crypt(byte[] key, byte[] data) {
    int v10 = 0x100;
    int keylen = key.length;
    int datalen = data.length;
    byte[] sbox = new byte[v10];
    int i = 0;
    while(i < v10) {
        sbox[i] = ((byte)i);
        ++i;
    }

    int k = 0;
    i = 0;
    while(i < v10) {
        k = (sbox[i] + k + key[i % keylen]) % 0x100 & 0xFF;
        byte v7 = sbox[i];
        sbox[i] = sbox[k];
        sbox[k] = v7;
        ++i;
    }

    i = 0;
    k = 0;
    int j = 0;
    while(j < datalen) {
        i = (i + 1) % 0x100 & 0xFF;
        k = (sbox[i] + k) % 0x100 & 0xFF;
        v7 = sbox[i];
        sbox[i] = sbox[k];
        sbox[k] = v7;
        data[j] = ((byte)(data[j] ^ sbox[(sbox[i] + sbox[k]) % 0x100 & 0xFF]));
        ++j;
```

JEB's output (right), after the code was analyzed by an engineer.

The method code is neatly structured and readable.

Find more examples on our website.

Ask for a demo version and find out more about pricing details on our website.