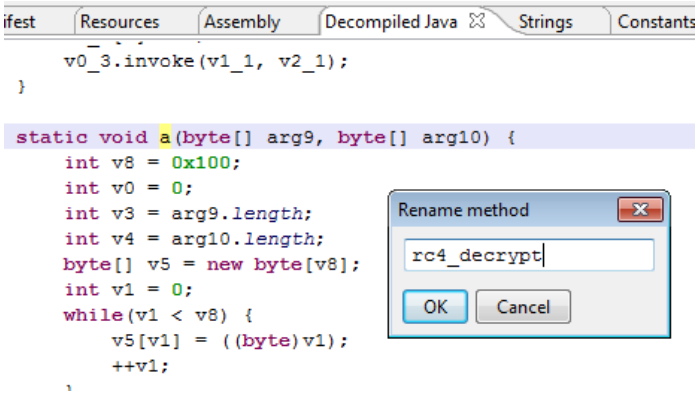
JEB
Le Décompilateur Interactif Android

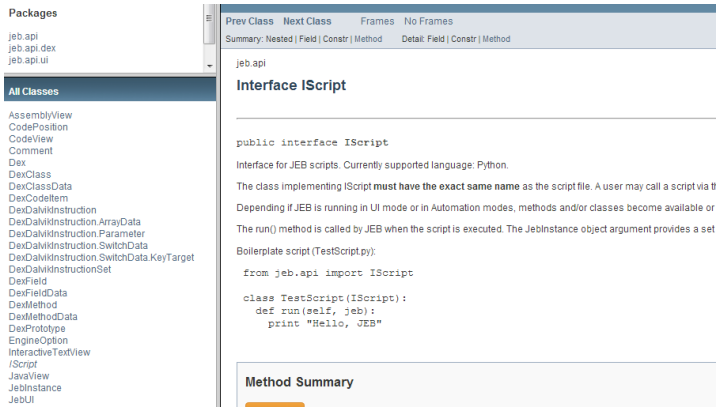
JEB est un décompilateur d'applications Android conçu pour les professionnels de la sécurité informatique, rétro-ingénieurs et auditeurs d'applications.

```
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;
```

Puissant.
Une spécificité unique de JEB est sa capacité à décompiler directement du bytecode Dalvik vers du code source Java. Il n'y a pas besoin d'outils de conversion dex>jar. Notre décompilateur propriétaire prend en considération toutes les subtilités des fichiers DEX.

Flexible.
Les rétro-ingénieurs ont besoin d'outils flexibles, en particulier quand ils analysent des programmes obfusqués ou protégés. L'interface utilisateur de JEB permet d'examiner les réferences croisées, de renommer classes, méthodes et champs, de naviguer à travers le code et les données, de prendre des notes et commentaires d'analyses.





Extensible.
Exploitez l'interface de programmation de JEB (API – Application Programming Interface) pour étendre les fonctionnalités de JEB avec des scripts et plugins. Accédez a l'AST du code Java decompilé pour enlever des couches d'obfuscation. Utilisez JEB en mode non-interactif pour vos besoins d'automatisation.

L'API est disponible pour les langages Python et Java.

JEB
Le Décompilateur Interactif Android

Deux avantages majeurs de JEB par rapport aux outils existants sont son interactivité et sa flexibilité, ainsi que la qualité du code Java produit par le décompilateur. Ces avantages permettent aux rétro-ingénieurs d'analyser et comprendre du code complexe.

```
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;
    label30: 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;
```

Décompilateur Java tiers (à gauche)
- Code statique, aucune interactivité
- Erreurs de décompilation (flèches)
- Résultat: faible lisibilité, utilisation limitée, erreurs d'interprétation

JEB (à droite), après analyse de code par un rétro-ingénieur.

Le code de la méthode est lisible, clairement structuré, et correct.

Retrouvez d'autres exemples sur notre site web.

```
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;
```

Tous les détails, version démo, coûts et licenses d'utilisation, sont sur notre site web.