De-Android

A simple APK, reverse engineer the logic, recreate the flag, and submit!

- Downloaded the BasicAndroidRE1.apk from `https://ctflearn.com/challenge/download/962`
- Googled on how to reverse engineer APK files, found references to use `apktool` to extract resources from an APK file, so installed apktools `choco install apktool`
- Attempted to decode tha APK file using `apktool -v decode BasicAndroidRE1.apk`
- Manually inspected the AndroidManifest.xml file in `BasicAndroidRE1` directory and found reference to `com.example.secondapp.MainActivity` which appears to be a Java class file
- Manually located the MainActivity `MainActivity.smali` file under `BasicAndroidRE1\smali\com\example\secondapp`, noticed that number of const_string references which appeared to be consistent with CTFlearn flag format, performed `grep const-string MainActivity.smali` and got

```
    const-string v1, "b74dec4f39d35b6a2e6c48e637c8aedb"
    const-string v2, "Success! CTFlearn{"
    const-string p1, "_is_not_secure!}"
```

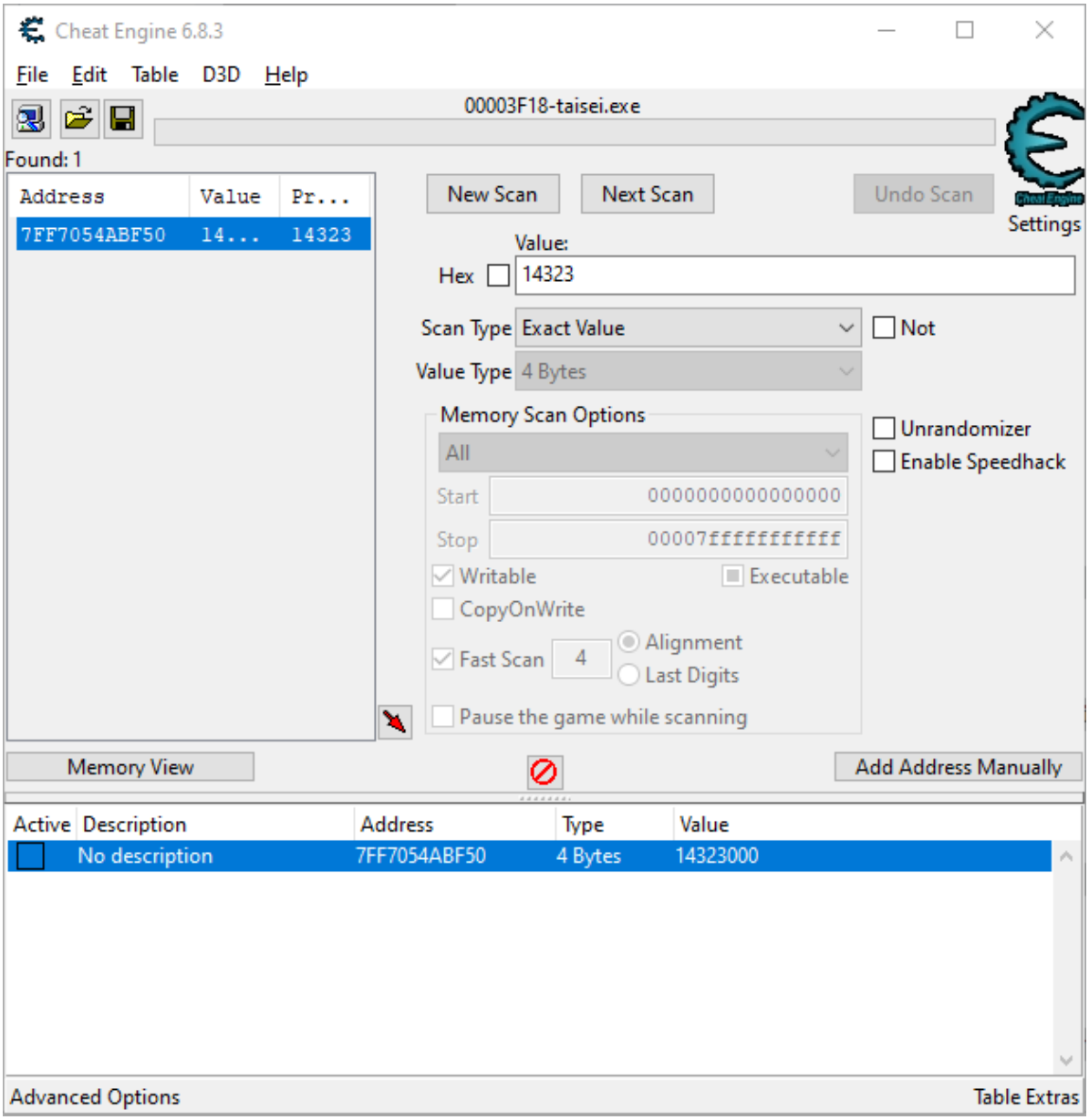
- Tried submitting CTFlearn{b74dec4f39d35b6a2e6c48e637c8aedb_is_not_secure!} but got nothing, then manually re-read the code and looks like the `b74dec4f39d35b6a2e6c48e637c8aedb` is an MD5 hash of a string, so checked on crackstation but found nothing
- Tried `https://md5.gromweb.com/?md5=b74dec4f39d35b6a2e6c48e637c8aedb` and found Sprint2019.
- Submitted: `CTFlearn{Spring2019_is_not_secure!}`

Taisei

Multiple ways to solve.

Each time your score is raised, special skills are added to your character.

Using special skill would let you invincible for a few seconds.



The most easy way is to use cheat engine and design a cheat.

javaisez3

Ah yes, another fun obfuscated java challenge. A bit of a rant, but Java tools are pretty terrible. On the c# side we have tools like ilspy and dnspy and cecil. In java land, however, many tools can't handle the slightest amount of obfuscation, and there are no debuggers that I know that work on the bytecode level, only on actual sources.

As for this jar specifically, I don't entirely know what it's deal is, but it seems that the class files in the jar (zip) are actually directories or something? Some tools won't detect the classes in the first place (jd-gui) and decompilers on javadecompilers.com will just straight up refuse to do anything.

I actually learned about two tools from a previous ctf that are great for these kinds of things.

For decompilation, recaf works great, and it was able to open the jar just fine and even decompile the code when switching to fernflower. Recaf has a bytecode editor, but it and java bytecode viewer both corrupt the jar with a `class not found ??????` error when run.

So for bytecode editing, the only tool I could get to successfully edit without corruption was cafebabe. It's actually pretty jank to work with since most functionality is behind right click menus without shortcuts. It also will occasionally corrupt the jar as well, but that's why we have backups.

Deobfuscating the code

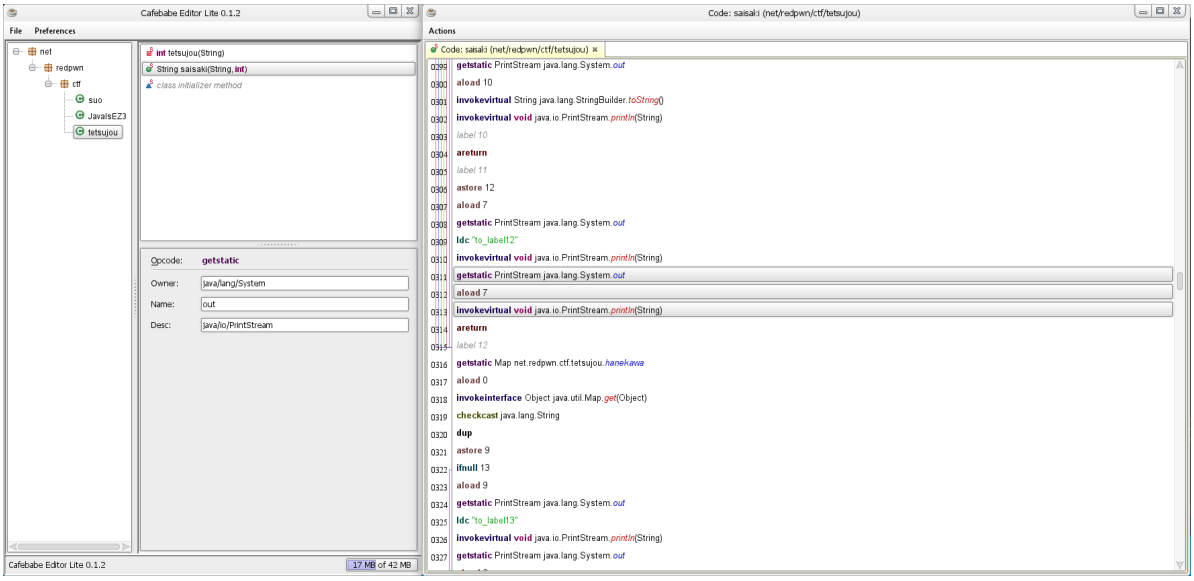
So even though we can decompile the code, it's still obfuscated. Here's the main method as an example:

```
public static void main(String[] var0) {
    redpwnCTF2021 var10000 = (redpwnCTF2021)null;
    if (var0.a<invokedynamic>(var0, tetsujou.saisaki("▣臭⋈′⋊臭◯ε⋈⋊幸⟫ᵔ⋈盏⟫ᵖ▣裸▣ᓫ⋊炮
ㄟᓄ", 322826692), -4280091229029863812L) == 0) {
        try {
            tetsujou.saisaki("ȝȝυ♁❧ƽƍƇ⟫ȝοᑕ✨ʸʂ৩♁❧Ꮤȝᑕ⁉ƽ",
649776694).a<invokedynamic>(tetsujou.saisaki("ȝȝυ♁❧ƽƍƇ⟫ȝοᑕ✨ʸʂ৩♁❧Ꮤȝᑕ⁉ƽ",
649776694), 8560971300846057061L).a<invokedynamic>(tetsujou.saisaki("ȝȝυ♁❧ƽƍƇ⟫
ȝοᑕ✨ʸʂ৩♁❧Ꮤȝᑕ⁉ƽ", 649776694).a<invokedynamic>(tetsujou.saisaki("ȝȝυ♁❧ƽƍƇ⟫
ȝοᑕ✨ʸʂ৩♁❧Ꮤȝᑕ⁉ƽ", 649776694), 8560971300846057061L), tetsujou.saisaki("鯈蒟鰋□鯚
蓐鯞□鮒蒐鰟□鰓蔰鯫□鮚蒟鰟□鰟", -784448576), 1235598990591485937L);
            null.a<invokedynamic>((Object)null, tetsujou.saisaki("듿ᜰ듀蓁듕□듏遾듩
ꙍ듀藒뒌ㅀ듅蓁듀ᵴ뒁蒼듄ᵨ듞蓁뒀□듟蘭듀ㅿ듕蘚듩ᜰ듀蓁듕□듄蘭듀ᜰ듏遾듩ꙍ듀藐뒂□뒌薩듃ь듟遾뒍□듒蘚뒌z
듞藸듘ㅅ듅蘑뒌ㅈ듩蘑듩w듄薛듀ω뒌瀙듩4듞藸듘ㅿ뒌薮듩ㅿ듀蘴듞□듈蘭듞ㅇ듏薊듟ꙍ뒥蘊뒄w듄蘭듟□듅藰뒌4듟
薊뒌ơ듄薛뒌ɪ듀薟듩□뒌薛듘ᴍ뒬", -1851298610), tetsujou.saisaki("礵德磯燗礹忱衼燋躋徼礧燲碌
頡祝熨躋徽碰燜磔徼礦", -558917396), -8331272066798825690L);
        } catch (Throwable var5) {}
    } else {
        if (var0[0].b<invokedynamic>(var0[0], tetsujou.saisaki("Ѫ륅ι霓ᖚ룤ы鬗ю륋Д
霖Ƶ뤅ᐱ霅", -1637188014), -47517957973123010731) != 48) {
            tetsujou.saisaki("□□□□□□□□□□□□□□□□□□□□", -2016726913).d<invokedynamic>
(tetsujou.saisaki("□□□□□□□□□□□□□□□□□□□□", -2016726913),
474225325441265L).b<invokedynamic>(tetsujou.saisaki("□□□□□□□□□□□□□□□□□□□□",
-2016726913).d<invokedynamic>(tetsujou.saisaki("□□□□□□□□□□□□□□□□□□□□", -2016726913),
474225325441265L), tetsujou.saisaki("□ʒ□□□k□□□ӟ□□□ə□□□K□□□ӟ□□□ґ□□□ӟ",
-846806080), tetsujou.saisaki("□□□퓀□□□뀿□□□뀏□□□뀓□□□", 801127825),
-1351703383126743055L);
            return;
        }

        String var6 = tetsujou.saisaki("徼碵征滑徘碑循滾徟磝徯滾從碑循滾徟磝徲溏怉泵应
澱", 1677756303);
```

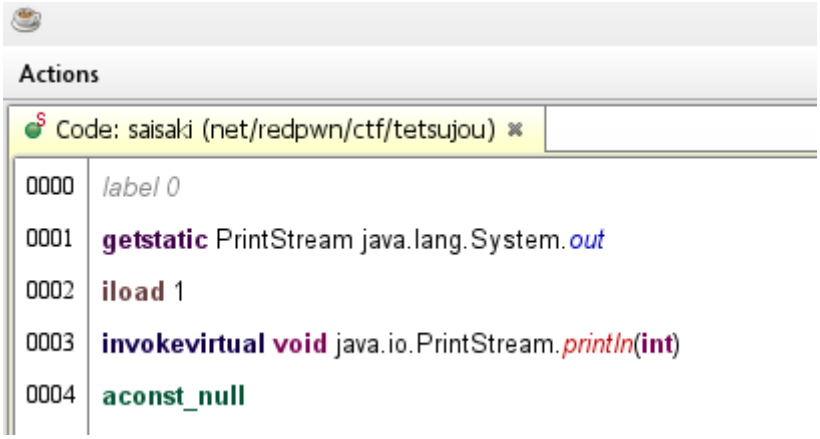
```
        char[] var1 = var6.b<invokedynamic>(var6, tetsujou.saisaki("ᄭᄆᆞ鑌ᄃᄆᆯ⏢錫ᄊᄆᆯ88鐰ᄭᄈᆞᄅ鐳", -1768915627), 3921157978488572744L);
        char[][] var2 = new char[][]{var1, null, null};
        int var3 = var0[0].b<invokedynamic>(var0[0], tetsujou.saisaki("ᓀᑕᓭ目ᒪ·ᑫ▫ᓀ▯ᓄᓂ·▯ᓀᑌ", -789459723), -4751795797312301073L) / 2;

        for(int var4 = 0; var4 < 2; ++var4) {
            int var10001 = var4 + 1;
            String var10002 = var0[0].b<invokedynamic>(var0[0], var4 * var3, (var4 + 1) * var3, tetsujou.saisaki("□□□쒲□□□쒽□□□쒧□□□쒧", -236704285), 2278661231839426149L);
            var2[var10001] = var10002.b<invokedynamic>(var10002, tetsujou.saisaki("狩淪犡胴狹涊状脒犰淳犄胴犥淢犹胈", 1785375413), 3921157978488572744L);
        }

        var2.a<invokedynamic>(var2, tetsujou.saisaki("ᄈ毛ᐃᑭᐃ毛ᐟᑫᐁ骰ᐁᓀᐃ廷ᐁh～毬～ᖶ本毯⏭ᛦ", 411433941), -4036077825718603401L);
        if (var2.a<invokedynamic>(var2, tetsujou.saisaki("붫ᄐ붸貰붫ᄐ붨賮붱ᄐ붨賽붸ᄐ붨廣붭ᄇ뷥뵏뷛ᄒ뭇厧", 1254712750), -43281413226819715O9L) & var2.a<invokedynamic>(var2, tetsujou.saisaki("앬◆앷▯앰◆앦▯앵◆씬▯앷◆씬▯앢◆앳▯앱◆았▯", -723903136), 8504114058794503371L) & var0[0].b<invokedynamic>(var0[0], tetsujou.saisaki("붉ᄇ뷥負뷙ᄋ뷱貯뷴ᄒ뷀養뷡ᄒ뷡販", 778462705), 634352354493306863L) == 1101317042) {
            tetsujou.saisaki("撺ᅦ摳ᄋ攦ᄂ擾ᄋ揕ᄀ掣/擄 -擊ᄅ", -1278287300).d<invokedynamic>(tetsujou.saisaki("撺ᅦ摳ᄋ攦ᄂ擾ᄋ揕ᄀ掣/擄 -擊ᄅ", -1278287300), 474225325441265L).b<invokedynamic>(tetsujou.saisaki("撺ᅦ摳ᄋ攦ᄂ擾ᄋ揕ᄀ掣/擄 -擊ᄅ", -1278287300).d<invokedynamic>(tetsujou.saisaki("撺ᅦ摳ᄋ攦ᄂ擾ᄋ揕ᄀ掣/擄 -擊ᄅ", -1278287300), 474225325441265L), tetsujou.saisaki("fm⋈radᵴ」MWY刀ℍtℨTHz₱radtc㎜³₱radᵴtcns₱nV」刀ℍ㎝㎜³tcPa₰MV˙MWひ", -921572424), tetsujou.saisaki("ᄑ붩ᄍ鏽ᄒ뷩ᄍ鎳ᄌ뷖ᄑ鏄ᄇᄇᄒ鏘ᄭ붩ᄒ", -266634598), -1351703383126743055L);
        } else {
            tetsujou.saisaki("썼◆썰▯쏰◆쎇▯썰◆쎍▯썵◆쎻▯", 1400642492).d<invokedynamic>(tetsujou.saisaki("썼◆썰▯쏰◆쎇▯썰◆쎍▯썵◆쎻▯", 1400642492), 474225325441265L).b<invokedynamic>(tetsujou.saisaki("썼◆썰▯쏰◆쎇▯썰◆쎍▯썵◆쎻▯", 1400642492).d<invokedynamic>(tetsujou.saisaki("썼◆썰▯쏰◆쎇▯썰◆쎍▯썵◆쎻▯", 1400642492), 474225325441265L), tetsujou.saisaki("▯□□□□□□□□□□㊍□□□□□□□□□□
```

```
getstatic PrintStream java.lang.System.out
[value here]
invokevirtual void java.io.PrintStream.println(String)
```

It's sort of a pain to do this manually, since even though you can select multiple instructions, only one can be right clicked on at a time, and therefore copied at a time. I wrote an autohotkey script to speed things up, but it was still tedious.



This prints out the constants used, but doesn't say which ones go where. Thankfully, we have a constant value that is passed in from earlier that we can use to match up the correct strings.




```
example output from this patch:
-2100823452
net.redpwn.ctf.JavaIsEZ3
857220391
java.lang.reflect.Array
...
so you can find strings that look like this:
tetsujou.saisaki("□□□ﾗ□□□ゐ□□□ﾁ□□□ヲ□□□ヲ□□□:", -2100823452)
and replace them with this:
"net.redpwn.ctf.JavaIsEZ3"
```

We can also do the same thing for `sui.teori` to print out the method and fields it loads as well.

This makes the main function slightly more readable.

```
public static void main(String[] var0) {
    redpwnCTF2021 var10000 = (redpwnCTF2021)null;
    if (var0.a<invokedynamic>(var0, "net.redpwn.ctf.JavaIsEZ3",
-4280091229029863812L) == 0) { //hachikuji (check array length)
        try {
            "javax.swing.UIManager".a<invokedynamic>("javax.swing.UIManager",
8560971300846057061L).a<invokedynamic>("javax.swing.UIManager".a<invokedynamic>
("javax.swing.UIManager", 8560971300846057061L), "javax.swing.UIManager",
1235598990591485937L);
            null.a<invokedynamic>((Object)null, "Silly-churl, billy-churl,
silly-billy hilichurl... Woooh!\n~A certain Wangsheng Funeral Parlor
director\n\n(This is not the flag, btw)", "javax.swing.JOptionPane",
-8331272066798825690L);
        } catch (Throwable var5) {}
    } else {
        if (var0[0].b<invokedynamic>(var0[0], "java.lang.String",
-4751795797312301073L) != 48) { //length
            "java.lang.System".d<invokedynamic>("java.lang.System",
474225325441265L).b<invokedynamic>("java.lang.System".d<invokedynamic>
("java.lang.System", 474225325441265L), "*fanfare* You've been pranked!",
"*fanfare* You've been pranked!", -1351703383126743055L);
            return;
        }

        String var6 = "WalnutGirlBestGirl_07/15";
        char[] var1 = var6.b<invokedynamic>(var6, "java.lang.String",
3921157978488572744L); //toCharArray
        char[][] var2 = new char[][]{var1, null, null};
        int var3 = var0[0].b<invokedynamic>(var0[0], "java.lang.String",
-4751795797312301073L) / 2; //length
```

```
        for(int var4 = 0; var4 < 2; ++var4) {
            int var10001 = var4 + 1;
            String var10002 = var0[0].b<invokedynamic>(var0[0], var4 * var3,
(var4 + 1) * var3, "java.lang.String", 2278661231839426149L); //substring
            var2[var10001] = var10002.b<invokedynamic>(var10002,
"java.lang.String", 3921157978488572744L); //toCharArray
        }

        var2.a<invokedynamic>(var2, "net.redpwn.ctf.JavaIsEZ3",
-40360778257186034O1L); //kanbaru
        if (var2.a<invokedynamic>(var2, "net.redpwn.ctf.JavaIsEZ3",
-43281413226819715O9L) & var2.a<invokedynamic>(var2, "net.redpwn.ctf.JavaIsEZ3",
85041140587945033711L) /*sengoku*/ & var0[0].b<invokedynamic>(var0[0],
"java.lang.String", 634352354493306863L) /*hashCode*/ == 1101317042) {
            //win (we can't see strings yet since this hasn't been executed)
            tetsujou.saisaki("擒╣摺ⵑ攫ヽ擄ᵎ揕┬掣/擄 -擊ᦓ",
-1278287300).d<invokedynamic>(tetsujou.saisaki("擒╣摺ⵑ攫ヽ擄ᵎ揕┬掣/擄 -擊ᦓ",
-1278287300), 474225325441265L).b<invokedynamic>(tetsujou.saisaki("擒╣摺ⵑ攫ヽ擄ᵎ揕┬
掣/擄 -擊ᦓ", -1278287300).d<invokedynamic>(tetsujou.saisaki("擒╣摺ⵑ攫ヽ擄ᵎ揕┬掣/擄 -擊
ᦓ", -1278287300), 474225325441265L), tetsujou.saisaki("fm⋈radᵴ」MWY㇇ℍ₸ℍz₱radt₸㎜³₱radᵴt₸ns₱nV
ˌ㇇ℍ㏎㎜³t₸Pa₹MV˙MWὈ", -921572424), tetsujou.saisaki("ᄑ뷡ᩚ鐳ᅟ붱ᩚ鎳ᄌ뷖ᄑ鏄ᄒ븓ᄒ鏘ᄼ뷡ᩚ", -266634598), -1351703383126743055L);
        } else {
            "java.lang.System".d<invokedynamic>("java.lang.System",
474225325441265L).b<invokedynamic>("java.lang.System".d<invokedynamic>
("java.lang.System", 474225325441265L), "*fanfare* You've been pranked!",
"java.io.PrintStream", -1351703383126743055L); //println
        }
    }
}
```

Cleaned up looks like this:

```
String walnut = "WalnutGirlBestGirl_07/15";
char[] walArr = walnut.toCharArray();
char[][] thrArr = new char[][]{walArr, null, null};
int inpStrLen = args[0].length() / 2;

for (int i = 0; i < 2; i++) {
    String subStr = args[0].substring(i * inpStrLen, (i + 1) * inpStrLen);
    thrArr[i+1] = subStr.toCharArray();
}

kanbaru(thrArr);
if (oshino(thrArr) && sengoku(thrArr) && args[0].hashCode() == 1101317042) {
    //win
} else {
    System.out.println("*fanfare* You've been pranked!");
}
```

Seems simple, thrArr contains a random constant string and two halves of the input. This array is passed into two functions and if they are both true, and the hash of the input matches, then we win.

Kanbaru does xoring on the input, here's what it looks like:

```
private static void kanbaru(char[][] inp) {
    //redpwnCTF2021 var10000 = (redpwnCTF2021)null;
    for (int i = 0; i < inp.hachikuji() - 1; i++) {
        char[] var2 = inp[i];
        char[] var3 = inp[i + 1];
        for (int j = 0; j < var2.hachikuji(); j++) {
            var3[j] ^= var2[j];
        }
    }
}
```

The second item in the inp array is xor'd with the first item, then the third item is xor'd with the second item. So we need to figure out the first half first, then we can xor the second with the first half to get the final flag.

oshino (checks first half of flag)

```
private static boolean oshino(char[][] inp) {
    //redpwnCTF2021 var10000 = (redpwnCTF2021)null;
    char[] inp1 = inp[1];
    String inp1Str = new String(inp1);
    if (inp1Str.hashCode() != 998474623) {
        return false;
    } else {
        int[] reg = new int[6];
        int j = 0;

        //load input in four byte chunks and xor with 0x07150715
        for (int i = 0; i < hachikuji(inp1); i += 4) {
            reg[j++] = (
                inp1[i] << 24 |
                inp1[i + 1] << 16 |
                inp1[i + 2] << 8 |
                inp1[i + 3]
            ) ^ 118818581;
        }

        int pos = 0;
        int[] stack = new int[15];
        int stackPos = 0;
        boolean retValue = true;

        while (true) {
            byte opcode = araragi[pos];
            byte var9;
            int var10;
            int var11;
            switch (opcode) {
            case 0: //pop into reg
                var9 = araragi[pos + 1];
                stackPos--;
                reg[var9] = stack[stackPos];
                pos += 2;
                break;
            case 1: //push from reg
                var9 = araragi[pos + 1];
```

```
                stack[stackPos++] = reg[var9];
                pos += 2;
                break;
            case 2: //return
                return retValue;
            case 3: //push int constant
                var11 = araragi[pos + 1] << 24 | araragi[pos + 2] << 16 |
                        araragi[pos + 3] << 8 | araragi[pos + 4];
                stack[stackPos++] = var11;
                pos += 5;
                break;
            case 4: //compare top two stack values
                stackPos--;
                var11 = stack[stackPos];
                stackPos--;
                var10 = stack[stackPos];
                retValue &= var10 == var11;
                pos++;
                break;
            case 5: //push short constant
                var11 = araragi[pos + 1] << 8 | araragi[pos + 2];
                stack[stackPos++] = var11;
                pos += 3;
                break;
            case 6: //push byte constant
                var9 = araragi[pos + 1];
                stack[stackPos++] = var9;
                pos += 2;
            }
        }
    }
}
```

This function and the second half checker are both tiny "vms" if you want to call them that. The code that oshino executes is something like this:

```
    pushInt 0x58480753  (3, 88, 72, 7, 83)
    pushInt 0x02460746  (3, 2, 70, 7, 70)
    pushInt 0x2B0A2E4C  (3, 43, 10, 46, 76)
    pushInt 0x2A007505  (3, 42, 0, 117, 5)
    pushInt 0x09057118  (3, 9, 5, 113, 24)
    pushInt 0x36180A1C  (3, 54, 24, 10, 28)
    pushReg 0           (1, 0)
    compareStack        (4)
    pushReg 1           (1, 1)
    compareStack        (4)
    pushReg 2           (1, 2)
    compareStack        (4)
    pushReg 3           (1, 3)
    compareStack        (4)
    pushReg 4           (1, 4)
    compareStack        (4)
    pushReg 5           (1, 5)
    compareStack        (4)
    return              (2)
```

It seems to do a simple compare with some ints, but we have two xors to worry about: the 0x07150715 constant in this function but also the walnut constant in the array.

```
str(xor(0x36180A1C, 0x07150715, int32("Waln"))) = "flag"
str(xor(0x09057118, 0x07150715, int32("utGi"))) = "{d1d"
str(xor(0x2A007505, 0x07150715, int32("rlBe"))) = "_yOu"
str(xor(0x2B0A2E4C, 0x07150715, int32("stGi"))) = "_knO"
str(xor(0x02460746, 0x07150715, int32("rl_O"))) = "w?_c"
str(xor(0x58480753, 0x07150715, int32("7/15"))) = "hr1s"
first half = flag{d1d_yOu_knOw?_chr1s
```

sengoku (checks second half of flag)

```
private static boolean sengoku(char[][] inp) {
    //redpwnCTF2021 var10000 = (redpwnCTF2021)null;
    char[] inp2 = inp[2];
    long[] reg = new long[15];

    int j = 0;
    for(int i = 0; i < inp2.hachikuji(); i += 8) {
        reg[j++] = (
            (long)inp2[i] << 56 | (long)inp2[i + 1] << 48 |
            (long)inp2[i + 2] << 40 | (long)inp2[i + 3] << 32 |
            (long)inp2[i + 4] << 24 | (long)inp2[i + 5] << 16 |
            (long)inp2[i + 6] << 8 | (long)inp2[i + 7]
        ) ^ 0x0302071503020715;
    }

    String inp2Str = new String(inp2);
    reg[j] = (long)inp2Str.hashCode();
    int pos = 0;
    long[] stack = new long[15];
    int stackPos = 0;

    while (true) {
        int opcode = hitagi[pos];
        int var8;
        int var9;
        long var10;
        switch (opcode) {
        case 0: //push long constant
            var10 = (long)hitagi[pos + 1] << 56 | (long)hitagi[pos + 2] << 48 |
                    (long)hitagi[pos + 3] << 40 | (long)hitagi[pos + 4] << 32 |
                    (long)hitagi[pos + 5] << 24 | (long)hitagi[pos + 6] << 16 |
                    (long)hitagi[pos + 7] << 8 | (long)hitagi[pos + 8];
            stack[stackPos++] = var10;
            pos += 9;
            break;
        case 1: //push int constant
            var10 = (long)hitagi[pos + 1] << 24 | (long)hitagi[pos + 2] << 16 |
                    (long)hitagi[pos + 3] << 8 | (long)hitagi[pos + 4];
            stack[stackPos++] = var10;
            pos += 5;
            break;
        case 2: //push short constant
            var10 = (long)hitagi[pos + 1] << 8 | (long)hitagi[pos + 2];
```

```
            stack[stackPos++] = var10;
            pos += 3;
            break;
        case 3: //push byte constant
            var10 = (long)hitagi[pos + 1];
            stack[stackPos++] = var10;
            pos += 2;
            break;
        case 4: //reg a equals reg b
            var8 = hitagi[pos + 1];
            var9 = hitagi[pos + 2];
            reg[0] = reg[var8] == reg[var9] ? 0L : 1L;
            pos += 3;
            break;
        case 5: //jump
            pos = hitagi[pos + 1];
            break;
        case 6: //jump if eqz
            if (reg[0] == 0L) {
                pos = hitagi[pos + 1];
            } else {
                pos += 2;
            }
            break;
        case 7: //jump if neqz
            if (reg[0] != 0L) {
                pos = hitagi[pos + 1];
            } else {
                pos += 2;
            }
            break;
        case 8: //xor reg a and reg b
            var8 = hitagi[pos + 1];
            var9 = hitagi[pos + 2];
            reg[var8] ^= reg[var9];
            pos += 3;
            break;
        case 9: //or reg a and reg b
            var8 = hitagi[pos + 1];
            var9 = hitagi[pos + 2];
            reg[var8] |= reg[var9];
            pos += 3;
        case 16: //and reg a and reg b
            var8 = hitagi[pos + 1];
            var9 = hitagi[pos + 2];
            reg[var8] &= reg[var9];
            pos += 3;
            break;
        case 17: //pop into reg
            var8 = hitagi[pos + 1];
            --stackPos;
            reg[var8] = stack[stackPos];
            pos += 2;
            break;
        case 18: //push from reg
            var8 = hitagi[pos + 1];
            stack[stackPos++] = reg[var8];
            pos += 2;
```

```
            break;
        case 19: //return
            return reg[0] == 0L;
        default:
            break;
        }
    }
}
```

Much of the same here, including the xor on the input. Just slightly different instructions.

```
    pushInt 0x66D63918           (1, 102, 214, 57, 24)
    pushLong 0x767058766B6E322E  (0, 118, 112, 88, 118, 107, 110, 50, 46)
    pushLong 0x7143146A706E1F21  (0, 113, 67, 20, 106, 112, 110, 31, 33)
    pushLong 0x6D667943394D396D  (0, 109, 102, 121, 67, 57, 77, 57, 109)
    popIntoReg 4                 (17, 4)
    popIntoReg 5                 (17, 5)
    popIntoReg 6                 (17, 6)
    popIntoReg 7                 (17, 7)
    jmp label2                   (5, 47)

label1:
    loadByte 1                   (3, 1)
    popIntoReg 0                 (17, 0)
    return                       (19)

label2:
    cmp 0, 4                     (4, 0, 4)
    jmpNeq label1                (7, 42)
    cmp 1, 5                     (4, 1, 5)
    jmpNeq label1                (7, 42)
    cmp 2, 6                     (4, 2, 6)
    jmpNeq label1                (7, 42)
    cmp 3, 7                     (4, 3, 7)
    jmpNeq label1                (7, 42)
    return                       (19)
```

Other than xoring with the first half of the flag, it's the same as oshino. Note that at this point the first half of the array has already been xor'd with walnut. There's also a hash of the string as input into this function, but we can pretty much ignore it since if the rest of the checks are correct, the hashcode will be too.

```
WalnutGirlBestGirl_07/15
str(xor(0x6D667943394D396D, 0x0302071503020715,
    int64("WalnutGi"), int64("flag{d1d"))) = "_is_4_Hu"
str(xor(0x7143146A706E1F21, 0x0302071503020715,
    int64("rlBestGi"), int64("_yOu_knO"))) = "_Tao_s1m"
str(xor(0x767058766B6E322E, 0x0302071503020715,
    int64("rl_07/15"), int64("w?_chr1s"))) = "p!_0715}"
second half = _is_4_Hu_Tao_s1mp!_0715}
```

Plugging it in

```
> java -jar javaisez3.jar flag{d1d_y0u_kn0w?_chr1s_is_4_Hu_Tao_s1mp!_0715}
Chute.  Now you know my secret
```

To be honest, I was kind of surprised this only got two solves but rp2sm got eight. Sorry java.