

# BÀI TẬP CTF

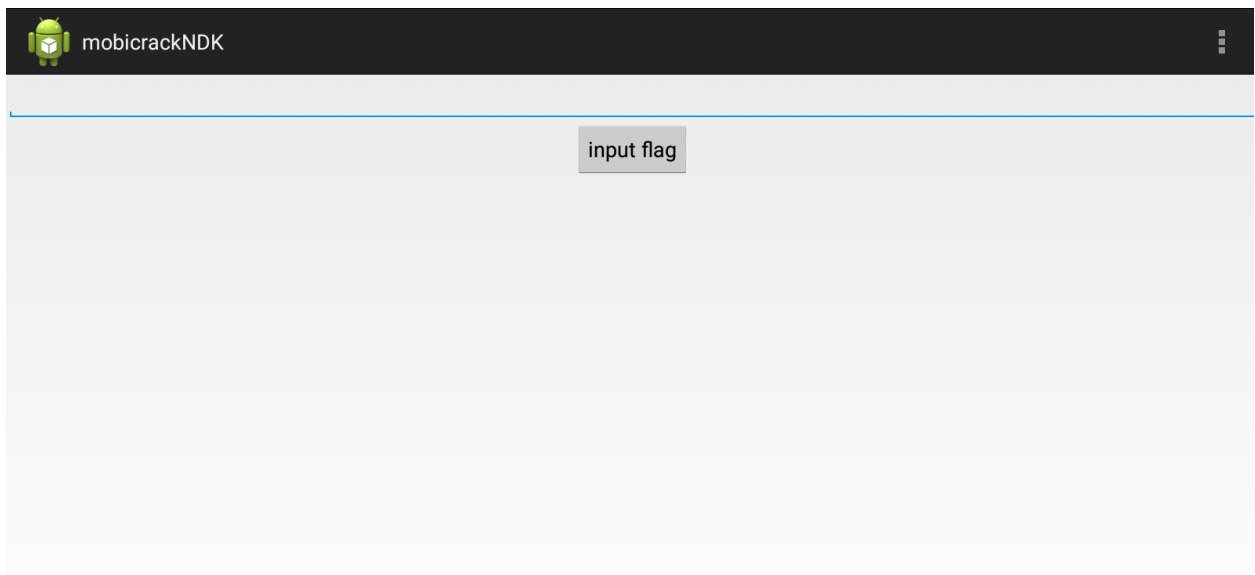
## Bảo mật web và ứng dụng – NT213.M21.ANTN

Giảng viên hướng dẫn: *Đỗ Hoàng Hiển*

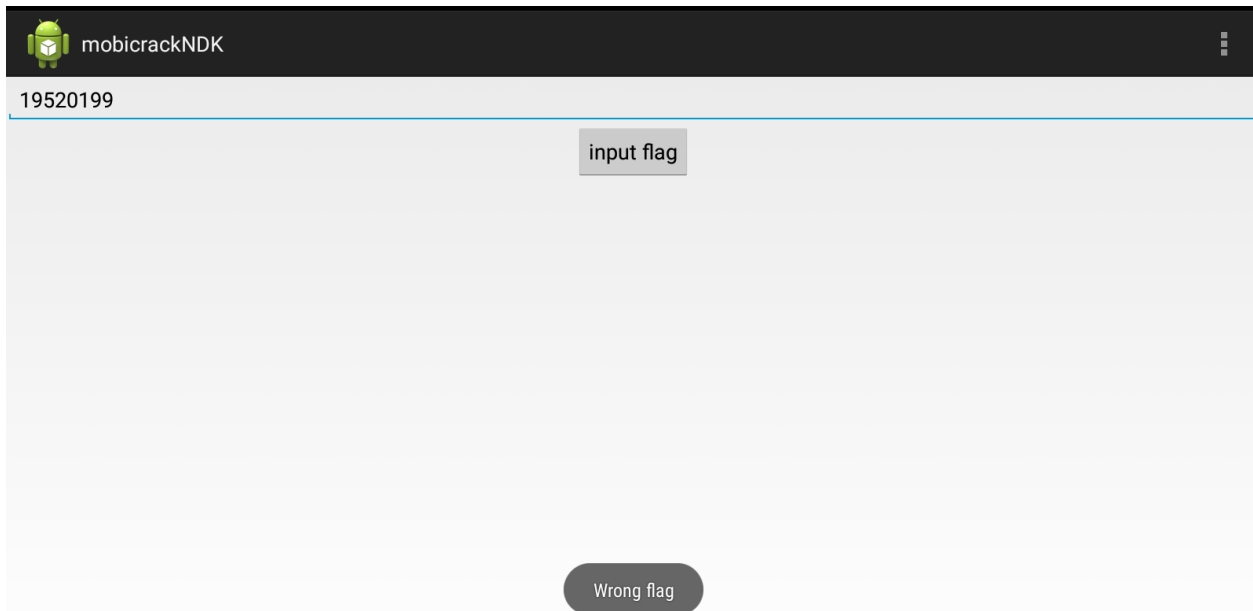
Sinh viên thực hiện: *19520199 – Lê Tôn Nhân*

### Challenge 6: mobicrackNDK

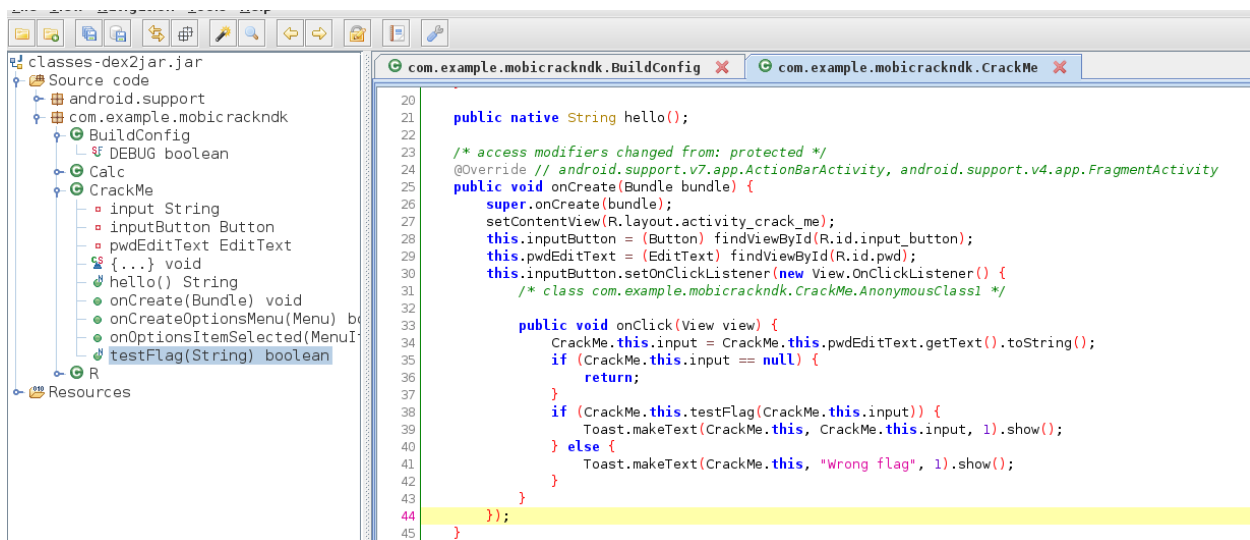
- Cài đặt ứng dụng, ở đây em sử dụng LDPlayer để chạy ứng dụng
- Ta thấy ứng dụng đơn giản gồm một input nhập vào và nút input flag để kiểm tra chuỗi đầu vào



- Thử nhập với chuỗi **19520199** và nhấn **input flag** thì xuất hiện thông báo **Wrong flag**



- Giải nén file **mobicrackNDK.apk**, sau đó sử dụng **dex2jars** để chuyển file **class.dex** thành file **jar**
- Sử dụng **jadx-gui** để dịch ngược ứng dụng và xem các logic cơ bản của nó
- Ta dễ dàng để nhận biết được hoạt động chính của chương trình nằm trong CrackMe. Ta tiến hành phân tích đoạn code trong CrackMe



- Ta thấy ở hàm onclick sẽ kiểm tra chuỗi đầu vào của chúng ta (**pwdEditText**). Nếu chuỗi nhập vào không đáp ứng yêu cầu của hàm **testFlag** thì xuất ra **Wrong flag**

```
public void onClick(View view) {
    CrackMe.this.input = CrackMe.this.pwdEditText.getText().toString();
    if (CrackMe.this.input == null) {
        return;
    }
    if (CrackMe.this.testFlag(CrackMe.this.input)) {
        Toast.makeText(CrackMe.this, CrackMe.this.input, 1).show();
    } else {
        Toast.makeText(CrackMe.this, "Wrong flag", 1).show();
    }
}
```

- Tiến hành kiểm tra hàm **testFlag**. Hàm này được load lên từ thư viện do đó ta sử dụng IDA để kiểm tra

```
static {
    System.loadLibrary("mobicrackNDK");
}
```

```
public native boolean testFlag(String str);
```

- Kiểm tra trong IDA thì ta không thấy hàm nào có tên **testFlag** cả.
- Tiến hành phân tích hàm **JNI\_Onload**

```
1 jint JNI_OnLoad(JavaVM *vm, void *reserved)
2 {
3     int v3; // r5
4     char *v4; // r7
5     int v5; // r1
6     FILE *stream; // [sp+4h] [bp-24h]
7     int v8; // [sp+Ch] [bp-1Ch] BYREF
8
9     v8 = 0;
10    printf("JNI_OnLoad");
11    if ( (*vm)->GetEnv(vm, (void **)&v8, 65540) )
12        goto LABEL_6;
13    v3 = v8;
14    v4 = classPathName[0];
15    stream = (FILE *)((_char *)&_SF + 168);
16    fprintf((FILE *)((_char *)&_SF + 168), "RegisterNatives start for '%s'", classPathName[0]);
17    v5 = (*(int (__fastcall **)(int, char *)))(*(int (__fastcall **)(int, char *)))(v3 + 24)(v3, v4);
18    if ( !v5 )
19    {
20        fprintf(stream, "Native registration unable to find class '%s'", v4);
21    LABEL_6:
22        fputs("GetEnv failed", (FILE *)((_char *)&_SF + 168));
23        return -1;
24    }
```

```

25 if ( (*int (__fastcall **)(int, int, char **, int))(*(_DWORD *)v3 + 860))(v3, v5, off_400C, 2) < 0 )
26 {
27     fprintf(stream, "RegisterNatives failed for '%s'", v4);
28     goto LABEL_6;
29 }
30 return 65540;
31 }

```

- Ta thấy ở hàm này chương trình thực hiện việc đăng ký động một class với hàm **off\_400C**
- Click vào **off\_400C** để kiểm tra

```

400C off_400C          DCD aTestflag          ; DATA XREF: JNI_OnLoad+60↑o
400C                  ; JNI_OnLoad+68↑o ...
400C                  ; "testFlag"
4010                  DCD aLjavaLangStrin_0    ; "(Ljava/lang/String;)Z"
4014                  DCD abcdefghijklmn+1

```

- Ta biết được hàm này chính là hàm **TestFlag** của ta và tên hàm tương ứng của nó là **abcdefghijklmn**
- Tiến hành phân tích hàm **abcdefghijklmn**
- Đầu vào của chúng ta được lưu trong biến **v13**
- Ta thấy điều kiện đầu tiên của đầu vào là phải có độ dài bằng 16.

```

1 bool __fastcall abcdefghijklmn(int a1, int a2, int a3)
2 {
3     _BOOL4 v5; // r4
4     size_t i; // r6
5     int v7; // r2
6     int v8; // r4
7     int v9; // r0
8     const char *v10; // r5
9     int v12; // [sp+4h] [bp-C4h]
10    const char *v13; // [sp+8h] [bp-C0h]
11    char s2[12]; // [sp+14h] [bp-B4h] BYREF
12    char v15[12]; // [sp+20h] [bp-A8h] BYREF
13    char s[128]; // [sp+2Ch] [bp-9Ch] BYREF
14
15    if ( !jniEnv )
16        jniEnv = a1;
17    memset(s, 0, sizeof(s));
18    v13 = (const char *)(*int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)jniEnv + 676)(jniEnv, a3, 0);
19    v5 = 0;
20    if ( strlen(v13) == 16 )
21    {
22        for ( i = 0; i != 8; ++i )
23            s2[i] = v13[i] - i;
24        v5 = 0;
25        s2[8] = 0;
26        if ( !strcmp(seed[0], s2) )

```

- Nếu thỏa ta đến với điều kiện tiếp theo sẽ so sánh 8 ký tự đầu vào với biến **seed**. Với **seed** có giá trị là **QfIMn`fH**

4004 seed	DCD aQflmnFh	; DATA XREF: LOAD:00000204↑o
4004		; abcdefghijklmn+64↑o ...
4004		; "QflMn`fH"

- Tiếp theo ta thấy phương thức tính **calcKey** được gọi và kết quả được lưu biến **v10**

```

26 if ( !strcmp(seed[0], s2) )
27 {
28     v12 = (*(int (__fastcall **)(int, const char *)))(_DWORD *)jniEnv + 24)(jniEnv, "com/example/mobicrackndk/Calc");
29     if ( !v12 )
30     {
31         _android_log_print(4, "log", "class,failed");
32         goto LABEL_11;
33     }
34     v7 = (*(int (__fastcall **)(int, int, const char *, const char *)))(_DWORD *)jniEnv + 452)((
35         jniEnv,
36         v12,
37         "calcKey",
38         ">()V");
39     if ( !v7 )
40     {
41         _android_log_print(4, "log", "method,failed");
42 LABEL_11:
43         exit(1);
44     }
45     JNIEnv::CallStaticVoidMethod(jniEnv, v12, v7);
46     v8 = (*(int (__fastcall **)(int, int, const char *, const char *)))(_DWORD *)a1 + 576)((
47         a1,
48         v12,
49         "key",
50         "Ljava/lang/String;");
51
52     if ( !v8 )
53         _android_log_print(4, "log", "fid,failed");
54     v9 = (*(int (__fastcall **)(int, int, int)))(_DWORD *)a1 + 580)(a1, v12, v8);
55     v10 = (const char *)(*(int (__fastcall **)(int, int, _DWORD)))(_DWORD *)jniEnv + 676)(jniEnv, v9, 0);
56     while ( i < strlen(v10) + 8 )
57     {
58         v15[i - 8] = v13[i] - i;
59         ++i;
60     }
61     v15[8] = 0;
62     v5 = strcmp(v10, v15) == 0;
63 }
64 return v5;
65 }

```

- Kết quả hàm calc sẽ gán key bằng chuỗi **c7^WVHZ**, ta có thể xem hàm này ở **calcactivity**
- Do đó **v10** sẽ là **c7^WVHZ**,

```

1 package com.example.mobicrackndk;
2
3 public class Calc {
4     public static String key;
5
6     public static void calcKey() {
7         key = new StringBuffer("c7^WVHZ").reverse().toString();
8     }
9 }

```


- Cuối cùng ta thấy nó so sánh 8 ký tự này với 8 ký tự cuối cùng của chuỗi ta nhập vào theo thứ tự đảo ngược nếu thỏa thì trả về true
- Do đó chuỗi nhập vào của ta sẽ là seed + key\_reverse = QfLMn`fH + reverse(c7^WVHZ,)
- Tiến hành tạo đoạn script để tính toán flag

```
1 seed = 'QfLMn`fH'
2 key = 'c7^WVHZ,'[::-1]
3 cyphertext = seed + key
4 plaintext = []
5 for i in range(16):
6     plaintext.append(chr(ord(cyphertext[i]) + i))
7 print "".join(c for c in plaintext)
```

- Flag lúc này ta tìm được là **QgnPrelO4cRackEr**

```
(kali㉿kali)-[~/.../Bai Tap 10/APKs/APKs/mobicrack]
$ python mobicrack.py
QgnPrelO4cRackEr
```

- Tuy nhiên thử check thì đây chưa phải là flag của ta


mobicrackNDK

QgnPrelO4cRackEr

input flag

Wrong flag

- Kiểm tra kỹ lại thì ta thấy chuỗi **seed** được sử dụng trong hàm **\_init\_my**

```

1 size_t _init_my()
2 {
3     size_t i; // r7
4     char *v1; // r4
5     size_t result; // r0
6
7     for ( i = 0; ; ++i )
8     {
9         v1 = seed[0];
10        result = strlen(seed[0]);
11        if ( i >= result )
12            break;
13        t[i] = v1[i] - 3;
14    }
15    seed[0] = t;
16    byte_4038 = 0;
17    return result;
18 }

```

- Ta thấy ở hàm này, chuỗi **seed** của chúng ta sẽ giảm đi 3 mỗi ký tự do đó kịch bản của chúng ta sẽ là

```

1 seed = 'QfLMn`fH'
2 seed = "".join(chr(ord(c) - 3) for c in seed)
3 key = 'c7^VVHZ,'[::-1]
4 cyphertext = seed + key
5 plaintext = []
6 for i in range(16):
7     plaintext.append(chr(ord(cyphertext[i]) + i))
8 print "".join(c for c in plaintext)

```

- Thực thi để thu flag

```

(kali@kali)-[~/Bai Tap 10/APKs/APKs/mobicrack]
$ python mobicrack.py
NdkMobiL4cRackEr

```

- Kiểm tra lại chuỗi này với ứng dụng thì ta thấy chuỗi ta xuất hiện trên thông báo



mobicrackNDK

NdkMobiL4cRackEr

input flag

NdkMobiL4cRackEr

**FLAG: NdkMobiL4cRackEr**