NahamCon CTF 2021: Solutions

Arthur Verschaeve (hi@arthurverschaeve.be)

December 2021

1 Warmups

1.1 Veebee

Judging by the extension, I assumed the given file contained the encoded version of VBScript. John Hammond, the author of this challenge, has published a tool to decode these[1]. I used this tool, and the resulting output file contained the following:

```
’ VeeBee goes buzz buzz
’
’
MsgBox("Sorry, not that easy!")
MsgBox("Okay, actually, you’re right. It is that easy.")
MsgBox("flag{f805593d933f5433f2a04f082f400d8c}")
```

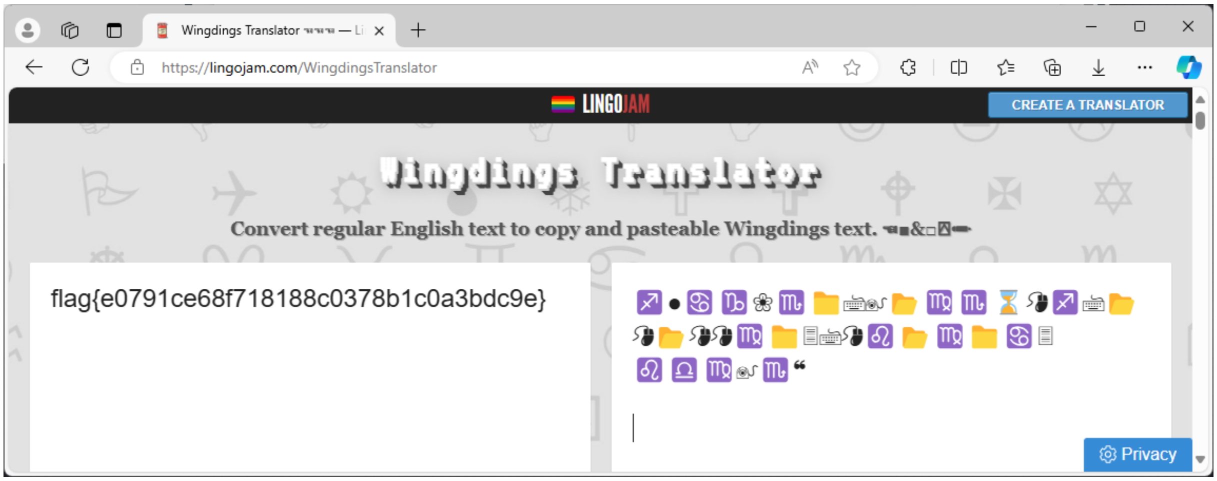
No need to actually execute this file, the flag is right there. The flag is `flag{f805593d933f5433f2a04f082f400d8c}`.

1.2 Read The Rules

The source code of the CTF rules page contains a HTML comment with the flag:
`flag{90bc54705794a62015369fd8e86e557b}`.

1.3 Chicken Wings

The symbols in the given text file are called Wingdings. There are various translators online:



Wingdings Translator

Convert regular English text to copy and pasteable Wingdings text.

flag{e0791ce68f718188c0378b1c0a3bdc9e}

The flag is `flag{e0791ce68f718188c0378b1c0a3bdc9e}`.

1.4 Car keys

The given string already looks like the flag, because of the {}-characters. That made me suspect it's some kind of simple cipher, and the other given word, `QWERTY`, is probably the key. I tried some different cipher tools online,

[1] https://github.com/JohnHammond/vbe-decoder

focusing on the ones with a key. The one that worked is called the *Keyed Ceasar Cipher*[2]. The result was flag{6f980c0101c8aa361977cac06508a3de}

1.5 Esab64

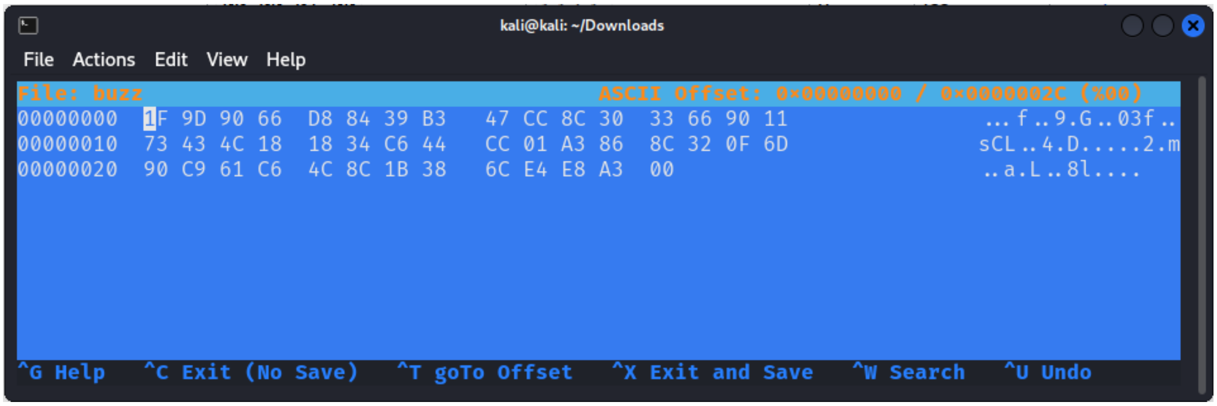
The challenge name is a hint on itself: reversing and base64 decoding are key to this challenge. Applying both to the given text, I'm not quite there yet, but then reversing again, I get the flag:

```
>>> import base64
>>> given_text = "mxWYntnZiVjMxEjY0kDOhZWZ4cjYxIGZwQmY2ATMxEzN1FjN113X"
>>> given_text_reverse = given_text[::-1]
>>> given_text_reverse_decoded = base64.b64decode(given_text_reverse)
>>> given_text_reverse_decoded
b'_}e61e711106bd0db1b78efa894b1125bf{galf'
>>> given_text_reverse_decoded[::-1]
b'flag{fb5211b498afe87b1bd0db601117e16e}_'
```

I don't know what that last underscore is doing there, but the flag is flag{fb5211b498afe87b1bd0db601117e16e}.

1.6 Buzz

Opening the given file in a hex editor, I see the first bytes are 1F 9D. This is the magic number of .z files.



```
File: buzz                                                ASCII Offset: 0x00000000 / 0x0000002C (%00)
00000000   1F 9D 90 66   D8 84 39 B3   47 CC 8C 30   33 66 90 11                  ...f..9.G..03f..
00000010   73 43 4C 18   18 34 C6 44   CC 01 A3 86   8C 32 0F 6D                  sCL..4.D.....2.m
00000020   90 C9 61 C6   4C 8C 1B 38   6C E4 E8 A3   00                           ..a.L..8l....
```

After adding the right extension, I opened the file up in the default archive utility and found only one file inside, which contained the flag. The flag is flag{b3a33db7ba04c4c9052ea06d9ff17869}.

2 Mobile

2.1 Andra

I used apktool to decompile the given APK file, and then found the flag inside of an XML file:

```
$ apktool d andra.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.7.0-dirty on andra.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/kali/.local/share/apktool/framework
   ↪ /1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
```

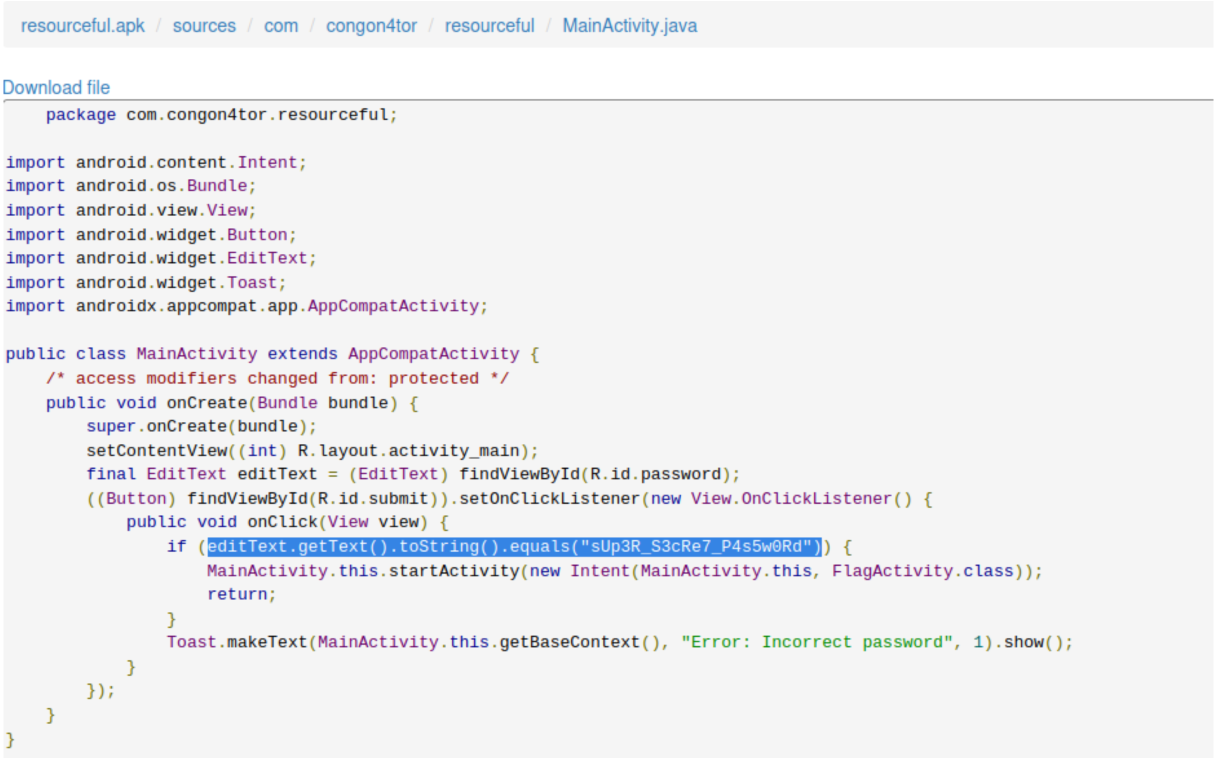
[2] https://www.boxentriq.com/code-breaking/keyed-caesar-cipher

```
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

$ cd andra

$ grep -r "flag{" .
./res/layout-v17/activity_flag.xml:        <EditText android:textSize="16.0dip"
    ↪  android:textStyle="bold" android:textColor="@color/white" android:
    ↪ gravity="center_horizontal" android:layout_width="fill_parent" android:
    ↪ layout_height="wrap_content" android:layout_marginLeft="10.0dip" android:
    ↪ layout_marginTop="40.0dip" android:layout_marginRight="10.0dip" android:
    ↪ text="flag{d9f72316dbe7ceab0db10bed1a738482}" android:textAlignment="
    ↪ center" />
./res/layout/activity_flag.xml:        <EditText android:textSize="16.0dip"
    ↪ android:textStyle="bold" android:textColor="@color/white" android:gravity
    ↪ ="center_horizontal" android:layout_width="fill_parent" android:
    ↪ layout_height="wrap_content" android:layout_marginLeft="10.0dip" android:
    ↪ layout_marginTop="40.0dip" android:layout_marginRight="10.0dip" android:
    ↪ text="flag{d9f72316dbe7ceab0db10bed1a738482}" />
```

The flag is visible in the result of this last `grep` call: `flag{d9f72316dbe7ceab0db10bed1a738482}`.

2.2 Resourceful

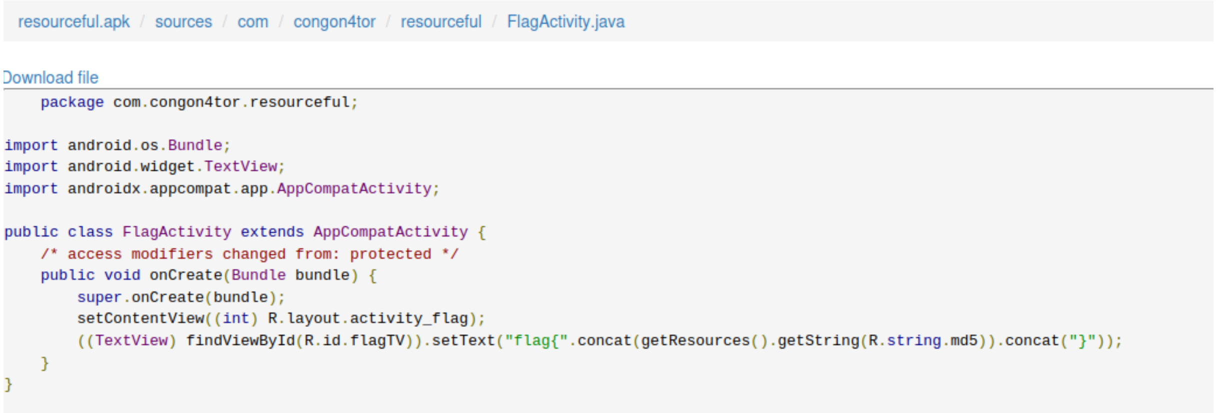
I used an online tool[3] to decompile the given APK, and had a look at the source code. The `MainActivity.java` file contains the password we'd need to access the app.


resourceful.apk / sources / com / congon4tor / resourceful / MainActivity.java
Download file

```java
    package com.congon4tor.resourceful;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_main);
        final EditText editText = (EditText) findViewById(R.id.password);
        ((Button) findViewById(R.id.submit)).setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                if (editText.getText().toString().equals("sUp3R_S3cRe7_P4s5w0Rd")) {
                    MainActivity.this.startActivity(new Intent(MainActivity.this, FlagActivity.class));
                    return;
                }
                Toast.makeText(MainActivity.this.getBaseContext(), "Error: Incorrect password", 1).show();
            }
        });
    }
}
```

I suppose I could install the app on a VM, use this password and I would maybe get the flag. However, the `startActivity` call below starts some `FlagActivity`. Inside, it looks like we are printing the flag:

[3] https://apktool.org

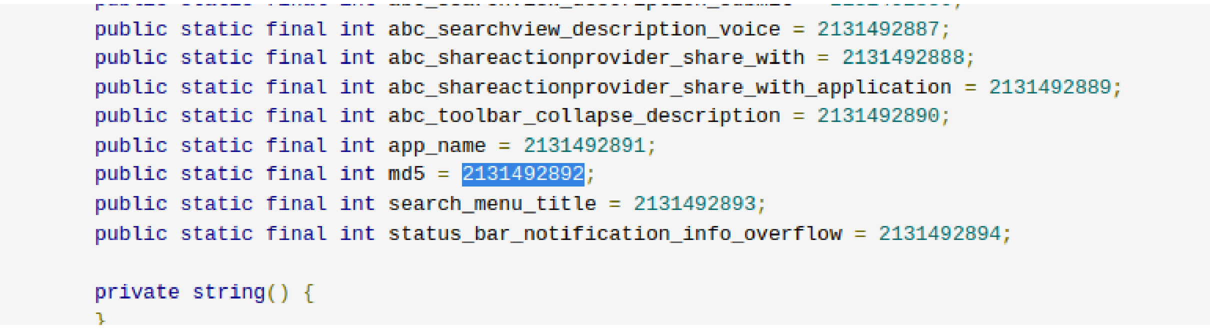


resourceful.apk / sources / com / congon4tor / resourceful / FlagActivity.java

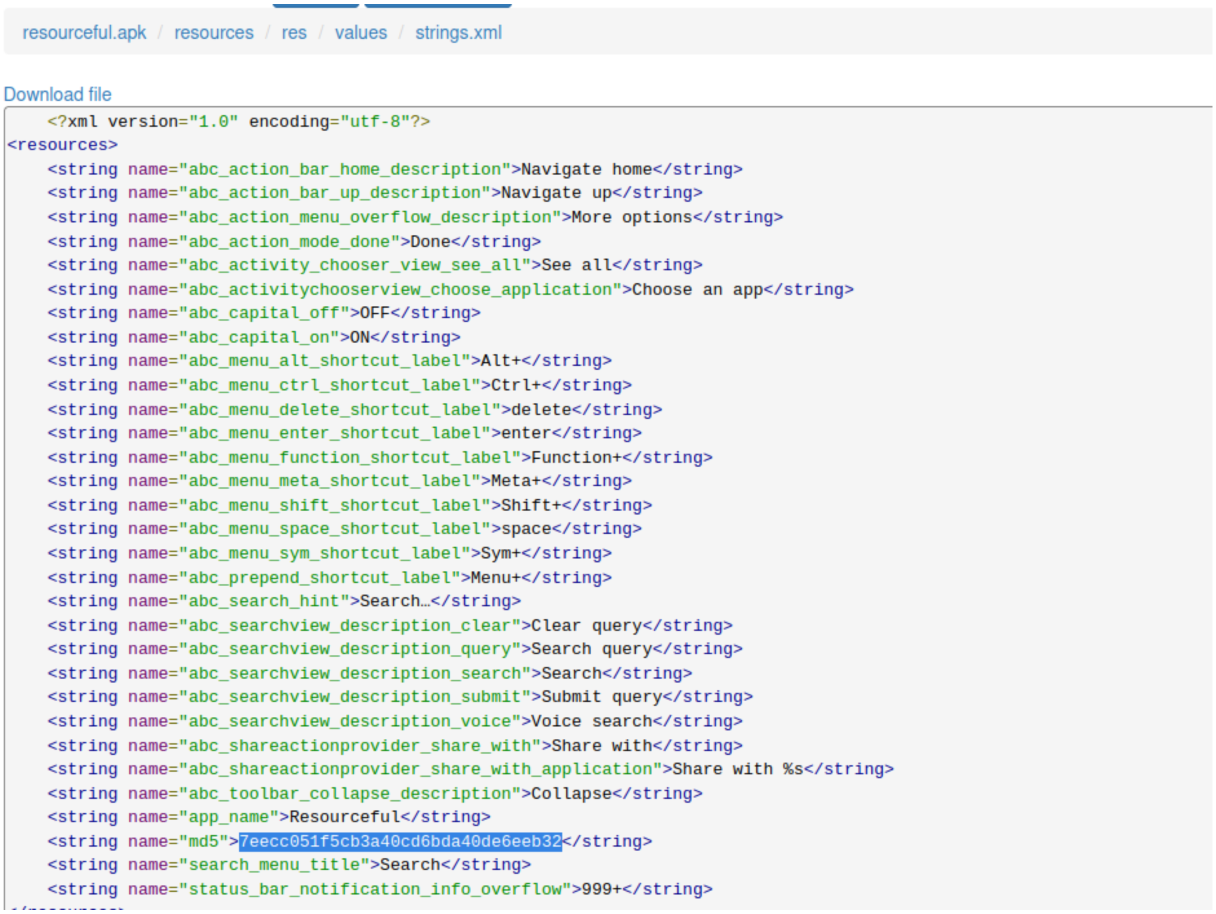
Download file

```
    package com.congon4tor.resourceful;

import android.os.Bundle;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class FlagActivity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_flag);
        ((TextView) findViewById(R.id.flagTV)).setText("flag{".concat(getResources().getString(R.string.md5)).concat("}"));
    }
}
```

The flag isn't in the code, however, it is stored in some resource called `md5` within `R`. I started looking for this value in the `resources` folder, and found two results:



```
        public static final int abc_searchview_description_voice = 2131492887;
        public static final int abc_shareactionprovider_share_with = 2131492888;
        public static final int abc_shareactionprovider_share_with_application = 2131492889;
        public static final int abc_toolbar_collapse_description = 2131492890;
        public static final int app_name = 2131492891;
        public static final int md5 = 2131492892;
        public static final int search_menu_title = 2131492893;
        public static final int status_bar_notification_info_overflow = 2131492894;

        private string() {
        }
```



resourceful.apk / resources / res / values / strings.xml

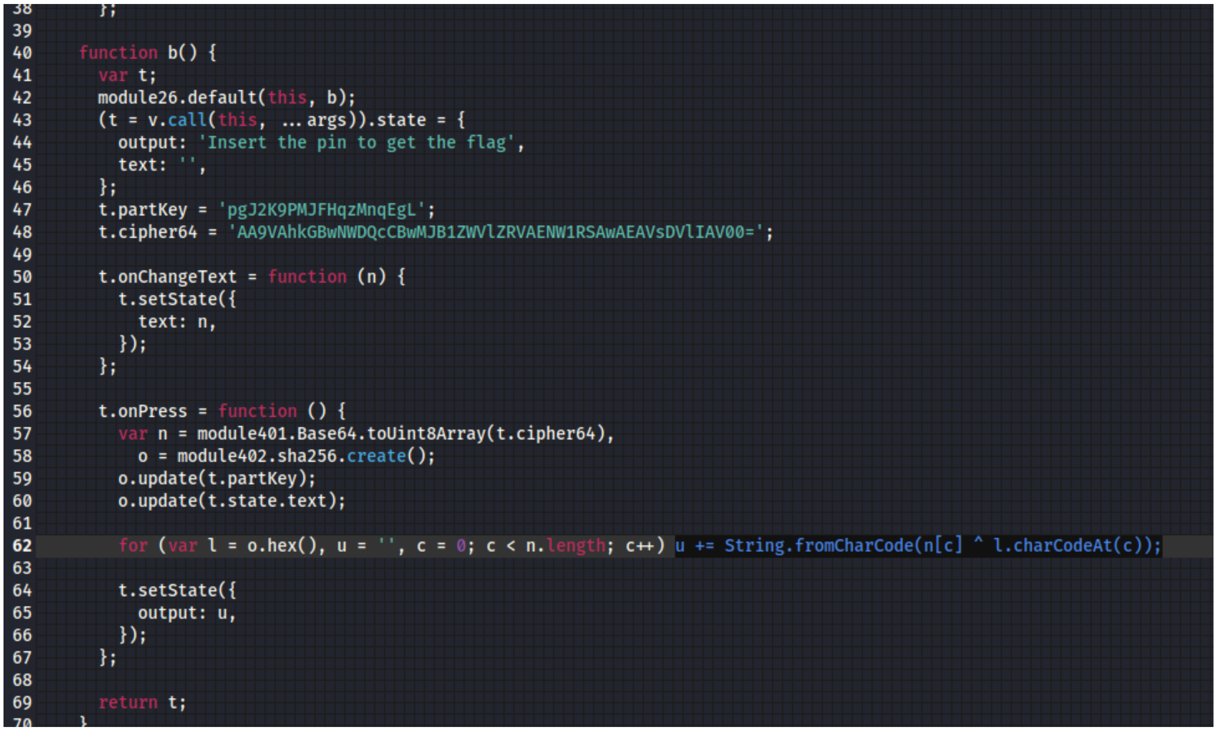
Download file

```
    <?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="abc_action_bar_home_description">Navigate home</string>
    <string name="abc_action_bar_up_description">Navigate up</string>
    <string name="abc_action_menu_overflow_description">More options</string>
    <string name="abc_action_mode_done">Done</string>
    <string name="abc_activity_chooser_view_see_all">See all</string>
    <string name="abc_activitychooserview_choose_application">Choose an app</string>
    <string name="abc_capital_off">OFF</string>
    <string name="abc_capital_on">ON</string>
    <string name="abc_menu_alt_shortcut_label">Alt+</string>
    <string name="abc_menu_ctrl_shortcut_label">Ctrl+</string>
    <string name="abc_menu_delete_shortcut_label">delete</string>
    <string name="abc_menu_enter_shortcut_label">enter</string>
    <string name="abc_menu_function_shortcut_label">Function+</string>
    <string name="abc_menu_meta_shortcut_label">Meta+</string>
    <string name="abc_menu_shift_shortcut_label">Shift+</string>
    <string name="abc_menu_space_shortcut_label">space</string>
    <string name="abc_menu_sym_shortcut_label">Sym+</string>
    <string name="abc_prepend_shortcut_label">Menu+</string>
    <string name="abc_search_hint">Search…</string>
    <string name="abc_searchview_description_clear">Clear query</string>
    <string name="abc_searchview_description_query">Search query</string>
    <string name="abc_searchview_description_search">Search</string>
    <string name="abc_searchview_description_submit">Submit query</string>
    <string name="abc_searchview_description_voice">Voice search</string>
    <string name="abc_shareactionprovider_share_with">Share with</string>
    <string name="abc_shareactionprovider_share_with_application">Share with %s</string>
    <string name="abc_toolbar_collapse_description">Collapse</string>
    <string name="app_name">Resourceful</string>
    <string name="md5">7eecc051f5cb3a40cd6bda40de6eeb32</string>
    <string name="search_menu_title">Search</string>
    <string name="status_bar_notification_info_overflow">999+</string>
```

The last one is the flag: `flag{7eecc051f5cb3a40cd6bda40de6eeb32}`.

2.3 Microscopium

Again, `apktool` was used to decompile the APK. Inside of the resulting files, I found something that looked a lot like compiled/minified JavaScript. That file is called `index.android.bundle`, which is a typical name for a *React Native* bundle. I used the *React Native Decompiler*[4], which threw out JavaScript that was a lot more readable, in separate module files. The `400.js` file contains the code that seems relevant: at least it handles the password/pin and decrypts something.



```
    };

  function b() {
    var t;
    module26.default(this, b);
    (t = v.call(this, ...args)).state = {
      output: 'Insert the pin to get the flag',
      text: '',
    };
    t.partKey = 'pgJ2K9PMJFHqzMnqEgL';
    t.cipher64 = 'AA9VAhkGBwNWDQcCBwMJB1ZWVlZRVAENW1RSAwAEAVsDVlIAV00=';

    t.onChangeText = function (n) {
      t.setState({
        text: n,
      });
    };

    t.onPress = function () {
      var n = module401.Base64.toUint8Array(t.cipher64),
        o = module402.sha256.create();
      o.update(t.partKey);
      o.update(t.state.text);

      for (var l = o.hex(), u = '', c = 0; c < n.length; c++) u += String.fromCharCode(n[c] ^ l.charCodeAt(c));

      t.setState({
        output: u,
      });
    };

    return t;
  }
```

I recreated the relevant steps in this program in a separate JavaScript file, trying out different pin codes (assuming a 4-digit one first) using a loop. I couldn't get it to work with the actual module files, but from the context it was clear that those were `js-sha256` and `js-base64`, which I just pulled of NPM.

```
var t = {};

var sha256 = require('js-sha256');
var base64 = require('js-base64');

t.partKey = 'pgJ2K9PMJFHqzMnqEgL';
t.cipher64 = 'AA9VAhkGBwNWDQcCBwMJB1ZWVlZRVAENW1RSAwAEAVsDVlIAV00=';

var n = base64.toUint8Array(t.cipher64);
var o = sha256.create();

// Assuming a 4-digit code
for (var i = 0; i <= 9999; i++) {
        o = sha256.create();
        o.update(t.partKey);
        o.update(i.toString());

        for (var l = o.hex(), u = '', c = 0; c < n.length; c++) {
                u += String.fromCharCode(n[c] ^ l.charCodeAt(c));
        }
```

[4] https://github.com/richardfuca/react-native-decompiler

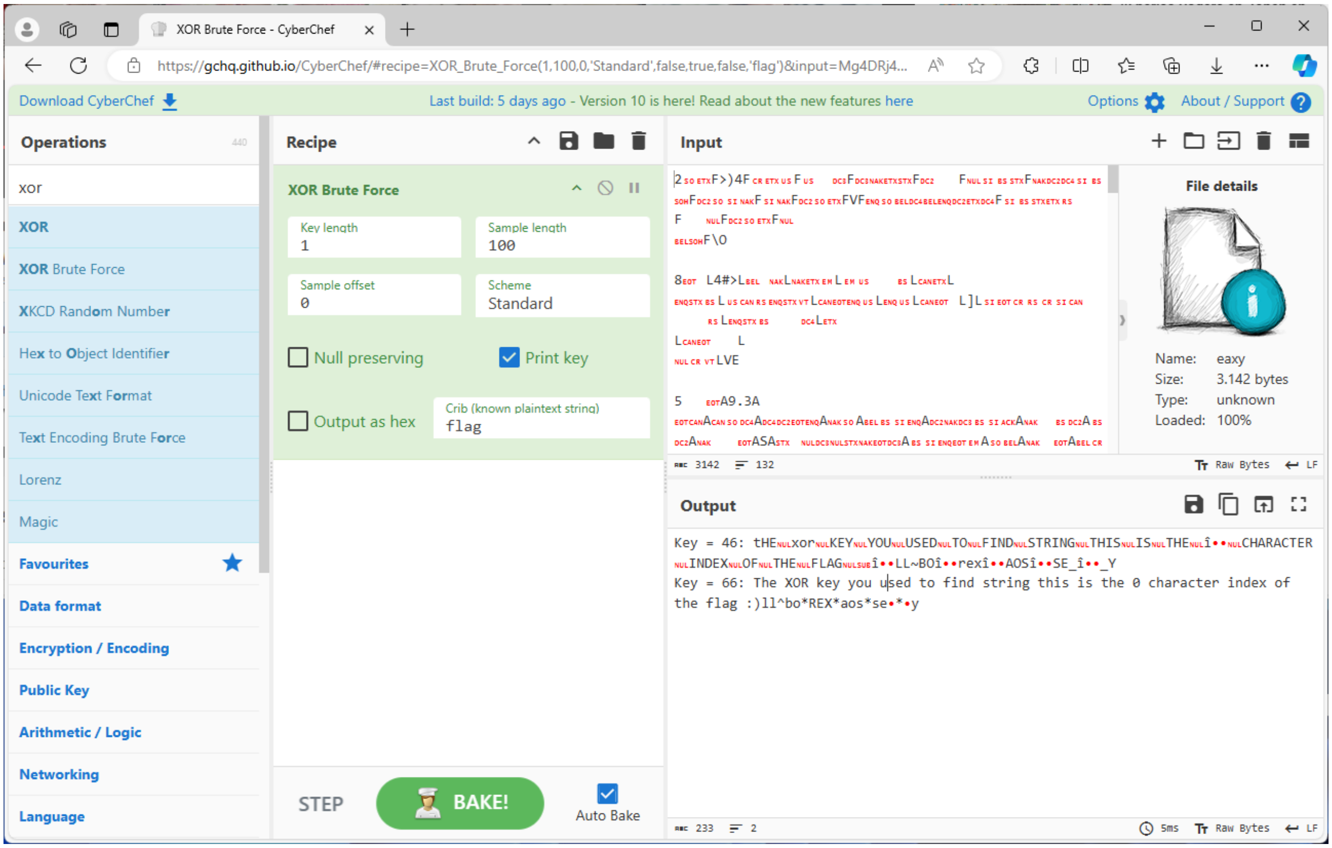
```
        if (u.startsWith("flag{")) {
                console.log("Pin:  " + i);
                console.log("Flag:  " + u);
        }
}
```

Executing this, I got the flag: `flag{06754e57e02b0c505149cd1055ba5e0b}`

3 Cryptography

3.1 Eaxy

The name of this challenge immediately made me think of a XOR cipher. The *XOR brute force* functionality in CyberChef, with *"flag"* as known text, to the rescue:



XOR Brute Force

Key length
1

Sample length
100

Sample offset
0

Scheme
Standard

□ Null preserving ☑ Print key

□ Output as hex

Crib (known plaintext string)
flag

Output

```
Key = 46: tHENULxorNULKEYNULYOUNULUSEDNULTONULFINDNULSTRINGNULTHISNULISNULTHENULî••NULCHARACTER
NULINDEXNULOFNULTHENULFLAGNULSUBî••LL~BOî••rexî••AOSî••SE_î••_Y
Key = 66: The XOR key you used to find string this is the 0 character index of
the flag :)ll^bo*REX*aos*se•*•y
```

Assuming I can find every character of the string in a similar way, I wrote the following Python script to do exactly that.

```
flag = [' ']*38

for key in list(range(256)):
    eaxy_file = open('eaxy', 'rb')
    b = bytearray([c ^ key for c in eaxy_file.read()])

    if b'The XOR key you used' in b:
        for i in b.split(b'this is the ')[1:]:
            flag[int(i[0:2])] = chr(key)
    eaxy_file.close()

print(''.join(flag))
```

This got me the result `flag{16edfce5c12443b61828af6cab90dc79}`.