Decrypting strings with a JEB script

cryptax.medium.com/decrypting-strings-with-a-jeb-script-1af522fa4979

@cryptax December 17, 2020

@cryptax

Dec 17, 2020

.

4 min read

In this article, we unpacked a malicious version of the "*Tous Anti Covid*" application. We know the main entry point of the payload DEX is `dad.calm.invest.qusalkrlkyy`.

```
<activity android:name="dad.calm.invest.qusalkrlkyy">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.app.role.SMS"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

the main entry point is located in the payload DEX (decrypted and dynamically loaded by the malware)

Encrypted strings

Using the decrypted payload DEX we unpacked in the previous article, we quickly notice in that class there are several encrypted strings. In this article, we'll see **how to create a JEB script to decrypt the strings**.

```
protected void onCreate(Bundle arg7) {
    super.onCreate(arg7);
    Point v7 = new Point();
    this.getWindowManager().getDefaultDisplay().getSize(v7);
    this.a.e(this, this.c.ao, v7.x);
    this.a.e(this, this.c.ap, v7.y);
    try {
        if(this.a.j(this, this.c.R).contains(this.a("YzVlODc2NTE="))) {
            this.a.a(this.d, this.a("ZWJlOTcwNDE0MjczMzZlZTdhZmE4NzdhYWMxY2YzYTA4MGZlNjM4MTkxZDA3Ng=="))
        }
    }
    catch(Exception unused_ex) {
        this.a.a(this.d, this.a("ZWJlOTcwNDE0MjczMzZlZTdhZmE4NzdhYWMxY2YzYTA4MGZlNjM4MTkxZDM3Ng=="));
        e v0 = this.a;
```

The strings are Base64 encoded + encryption by a()

`a()` is a method which first decodes Base64 and then decrypts data using a custom algorithm. This algorithm is initialized with a hard coded key. Below, I show a “de-obfuscated” version of `a()`:

```
private String decrypt(String encrypted_string) { try {    return new String(new
DecryptionAlgo(this.c.key.getBytes())
.decrypt(e.createByteArray(                                new String(
Base64.decode(encrypted_string, 0),                                    “UTF-8”))); }...
```

Implementing the decryption algo in Python

I want to decrypt encrypted strings, ok? To do so, I need to understand the decryption algorithm, and automate that in a JEB script. JEB scripts are written in **Python**, so I’ll have to **port Java code to Python**.

The first thing `a()` does is Base64 decoding. This is easily done in Python with the `base64` package and `b64decode()`.

Then, the code converts the base64 string to a byte array and decrypts the byte array using `b()`.

```
public final byte[] b(byte[] arg7) {
    byte[] v0 = new byte[arg7.length];
    int v1;
    for(v1 = 0; v1 < arg7.length; ++v1) {
        this.b = (this.b + 1) % 0x100;
        int[] v3 = this.a;
        int v4 = this.b;
        this.c = (this.c + v3[v4]) % 0x100;
        DecryptionAlgo.a(v4, this.c, v3);
        v0[v1] = (byte)(this.a[(this.a[this.b] + this.a[this.c]) % 0x100] ^ arg7[v1]);
    }

    return v0;
}
```

this is the (obfuscated) decryption algo
This decryption algorithm can quite easily be ported to Python:

1. No need for explicit memory allocations in Python such as `new byte`. We simply need to say `v0` is an array `[]`.
2. The for loop is transformed to `for i in range(…)`.
3. In Python, you cannot “assign a value to an array” with `v0[v1]=...`. Rather, we can “append” a value to an array.
4. The algorithm calls another method `a()`. If you look at its code, it simply swaps the values of indices `v4` and `this.c` in array `v3`. That’s easy to implement in Python too.

```
  def decrypt(self, buf):
    result = []
    for i in range(0, len(buf)):
      self.b = (self.b + 1) % 0x100
      v3 = self.transformed_key
      v4 = self.b
      self.c = (self.c + v3[v4]) % 0x100
      swap_values(v4, self.c, v3)
      result.append(self.transformed_key[(self.transformed_key[self.b] + self.transformed
_key[self.c]) % 0x100] ^ buf[i]);
    return ''.join([chr(c) for c in result])
```

The same algorithm, ported to Python. Here, decrypt is part of a Python class.
The other part we need to take care of is the algorithm's key. We see the code instantiates a decryption object with a hard-coded key (its value is `"dcpmeyucapxy"`). The object constructor prepares the key with a custom algorithm (see below).

```
  private static int[] c(byte[] arg5) {
      int[] v1 = new int[0x100];
      int v2 = 0;
      int v3;
      for(v3 = 0; v3 < 0x100; ++v3) {
          v1[v3] = v3;
      }

      int v3_1 = 0;
      while(v2 < 0x100) {
          v3_1 = (v3_1 + v1[v2] + arg5[v2 % arg5.length] + 0x100) % 0x100;
          DecryptionAlgo.a(v2, v3_1, v1);
          ++v2;
      }

      return v1;
  }
```

This method is called by the decryption object constructor (dad.calm.invest.c.c). It needs to be ported to Python.
The port of the method is not a problem, as modulo operator exist both in Java and Python.

Wrapping the algorithm into a JEB script

Once all elements of the decryption algorithm are implemented (we can test it in a standalone script), we need to wrap this up in a JEB script. JEB experts could do wonders, automatically recognize the strings, decrypt them and replace by the decrypted value (see scripts such as this one). However, I am not an expert, and I can't spend hours on writing a script either. So, we'll do something simpler: the JEB user is expected to select the string to decrypt, the script will automatically decrypt the string and add a comment next to it with the decrypted version. My script will not handle error cases, feel free to enhance 😄.

```
class tous_decrypt(IScript):
    def run(self, ctx):
        f = ctx.getFocusedFragment()
        if not f:
            print("[-] Select a text (no focused fragment)")
            return

        sel = f.getSelectedText() or f.getActiveItemAsText()
        if not sel:
            print("[-] Select a text (no selected text)")
            return

        b64encoded_string = sel.strip(' \'\"')
        decrypted_string = self.do_decrypt(b64encoded_string)
        print("Decrypting: {} --> {}".format(b64encoded_string, decrypted_string))

        a = f.getActiveAddress()
        if a and isinstance(f.getUnit(), IInteractiveUnit):
            comment0 = f.getUnit().getComment(a)
            comment = decrypted_string + '\n' + comment0 if comment0 else decrypted_string
            f.getUnit().setComment(a, comment)
```

Creating a JEB script. The main class must derive from IScript (defined in package com.pnfsoftware.jeb.client.api that we must import) , and the class must implement a method named run()
The JEB script (1) gets the selected string (`getSelectedText` or `getActiveItemAsText`), (2) then we send that string to the decryption algorithm we implemented. The result is the decrypted string. Finally, (3) we add that string as a comment (`setComment`). I largely inspired my code from this one to do that.

Source code script to decrypt strings (GitHub).

Running the script

Finally, put the Python script somewhere JEB can access it. Then, select a string to decrypt. Then, File > Scripts > Run Script. Select the script. And it should work :) The print commands go to JEB's logger console, the decrypted string is added as comment. For next times, you can simply use F2 to run the same script.

Very handy! Have a look at the video to see it in action.

I use F2 key to run the JEB script
— cryptax