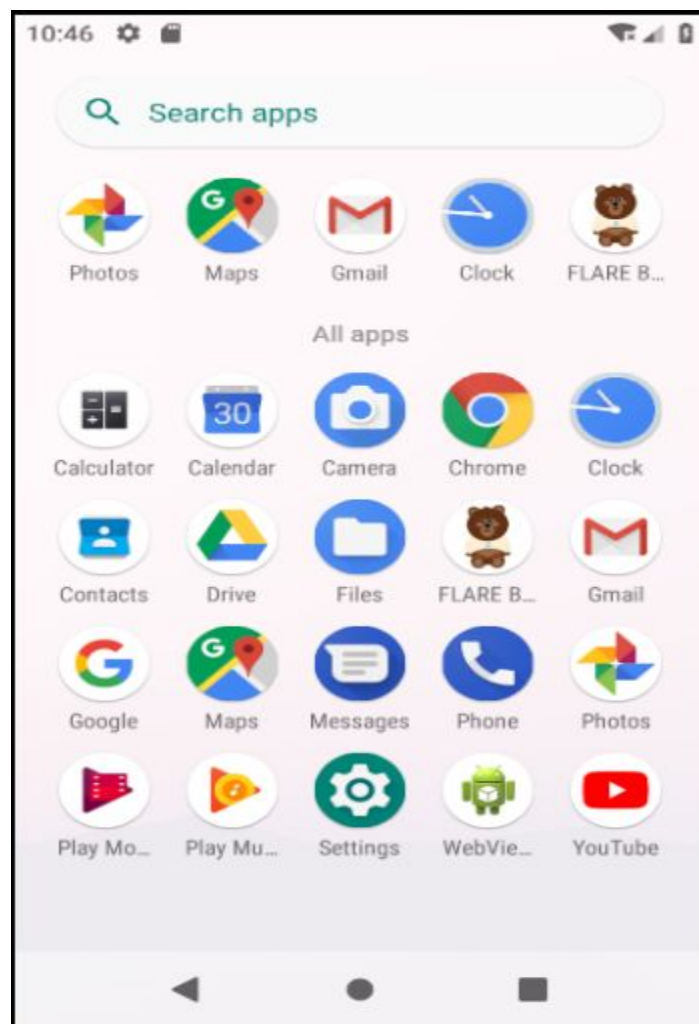


FlareBear (FlareOn CTF)

Description:

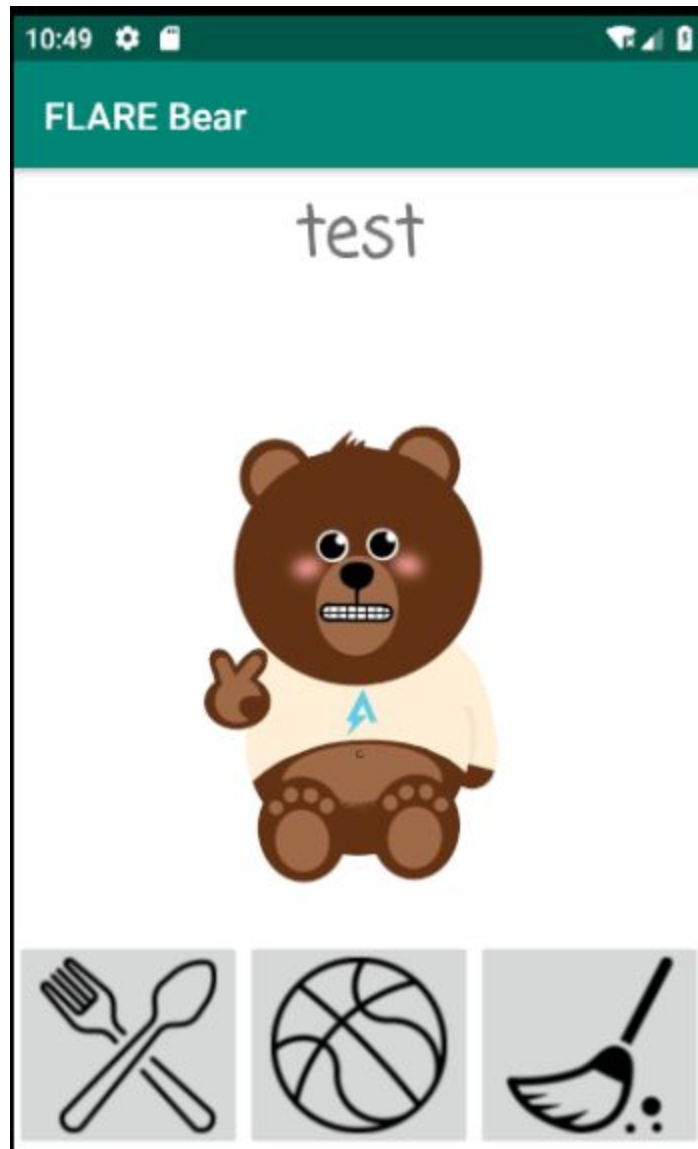
We at Flare have created our own Tamagotchi pet, the flarebear. He is very fussy. Keep him alive and happy and he will give you the flag.

We have given with apk file which is android application package file. I quickly installed it in Galaxy Nexus Device.

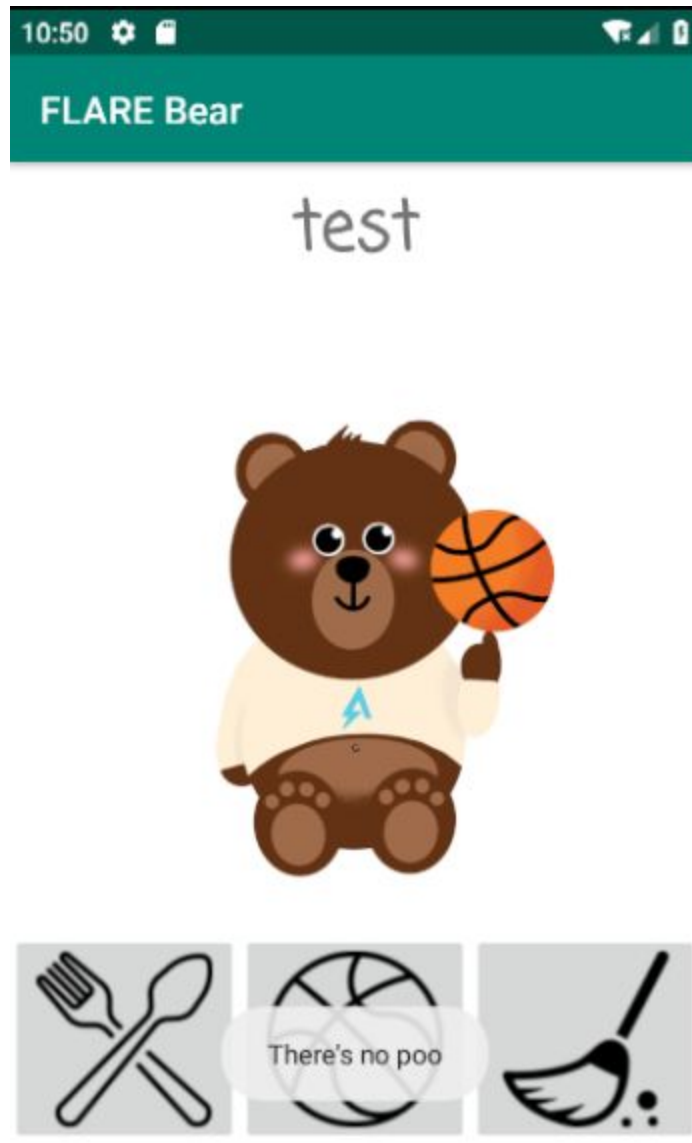




I've created New Flare Bear and it shows a fun bear like below.










We can see it has got 3 activities to play with. I just randomly clicked on Feed, Play and Clean icons and it showed up **There's no poo** message.



So i've started reversing the app to understand the logic behind this message.

RE

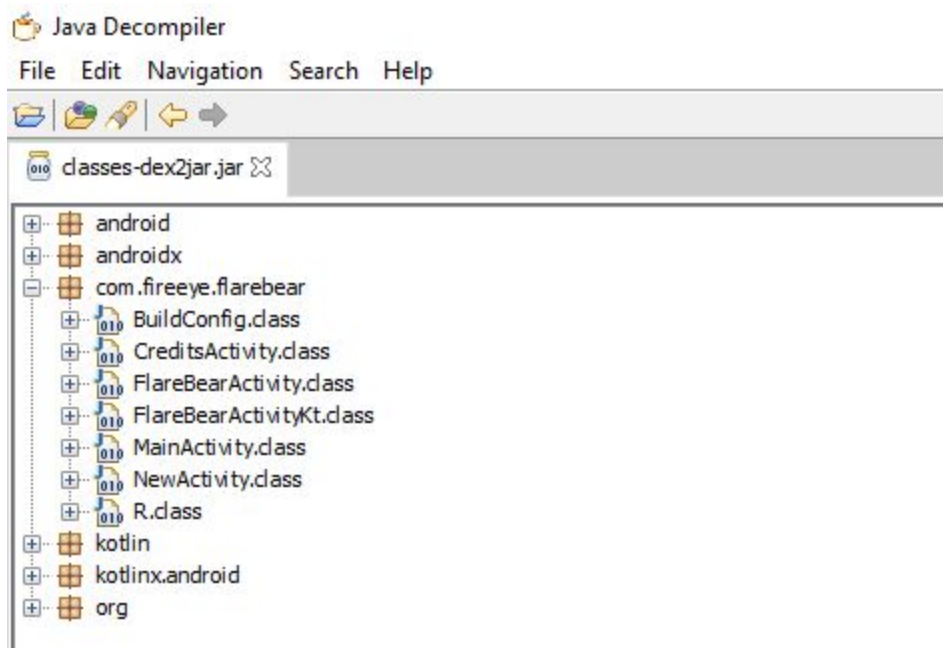
I've renamed the apk to zip and extracted. We get **classes.dex** file which is a Dalvik Executable file. It does contain compiled code of this application.

 kotlin	30-09-2019 10:28	File folder	
 META-INF	30-09-2019 10:28	File folder	
 res	30-09-2019 10:28	File folder	
 AndroidManifest.xml	22-05-2019 16:55	XML Document	3 KB
 classes.dex	22-05-2019 16:55	DEX File	3,334 KB
 classes-dex2jar.jar	30-09-2019 10:29	Executable Jar File	2,933 KB
 resources.arsc	22-05-2019 16:55	ARSC File	249 KB

We can use **dex2jar** tool to convert this compiled executable to java class files. I always go with frameworks to have everything segregated at one place. So i used **appie** for this challenge which has all required tools at one place.

```
$ d2j-dex2jar classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar
Detail Error Information in File .\classes-error.zip
Please report this file to one of following link if possible (any one).
https://sourceforge.net/p/dex2jar/tickets/
https://bitbucket.org/pxb1988/dex2jar/issues
https://github.com/pxb1988/dex2jar/issues [no attachment support, not preferred]
dex2jar@googlegroups.com
```

We have jar file which is Java Archive file contains java class files. We can use **jdgui** tool to view the content of files.



We could see the main package of the application **com.fireeye.flarebear**. There is an interesting class file named as **FlareBearActivity.class** i believe it holds the information which we are looking for.

It does have the strings we are looking for in **@Metadata** section.

```
@Metadata(bv={1, 0, 3}, "Lcom/fireeye/flarebear/FlareBearActivity;",  
"Landroid/support/v7/app/CompatActivity;", "()V", "handler",  
"Landroid/os/Handler;", "getHandler", "()Landroid/os/Handler;", "setHandler",  
"(Landroid/os/Handler;)V", "activityUi", "", "drawable",  
"Landroid/graphics/drawable/Drawable;", "drawable2", "addPooUi",  
"Landroid/widget/ImageView;", "addPoos", "changeClean", "clean", "",  
"changeFlareBearImage", "changeHappy", "happy", "changeImageAndTag",  
"changeMass", "mass", "view", "Landroid/view/View;", "cleanUi", "dance",  
"danceWithFlag", "decrypt", "", "password", "", "data", "feed", "feedUi", "getPassword",  
"getStat", "activity", "", "getState", "", "key", "defValue", "incrementPooCount",  
"isEcstatic", "", "isHappy", "onCreate", "savedInstanceState", "Landroid/os/Bundle;",  
"play", "playUi", "removePoo", "restorePoo", "saveActivity", "activityType",  
"setFlareBearName", "setMood", "setState", "value", "rotN", "n", "Companion",  
"app_release"}, k=1, mv={1, 1, 15})
```

danceWithFlag looks obvious so let's dig into that.

```
public final void setMood()  
{  
    if (isHappy())  
    {  
  
        ((ImageView)_$_findCachedViewById(R.id.flareBearImageView)).setTag("happy");  
        if (isEcstatic()) {  
            danceWithFlag();  
        }  
    }  
    else  
    {  
        ((ImageView)_$_findCachedViewById(R.id.flareBearImageView)).setTag("sad");  
    }  
}
```

So we have to meet certain conditions in order to trigger **danceWithFlag()** function.

1. isHappy()
2. isEcstatic()

isHappy() ?

It contains below conditions

```
public final boolean isHappy()
{
    int j = getStat('f');
    int i = getStat('p');
    double d = j / i;
    boolean bool;
    if ((d >= 2.0D) && (d <= 2.5D)) {
        bool = true;
    } else {
        bool = false;
    }
    return bool;
}
```

It fetches value of **j** from **getStat()** function by passing **f** as input argument. Similarly **i** by passing **p** as argument. Then calculates **d = j/i** and result of d contains floating value.

It must satisfy (d>=2.0D) && (d<=2.5D)

Let's look at what is **getStat()** function

```
public final int getStat(char paramChar)
{
    Object localObject =
PreferenceManager.getDefaultSharedPreferences((Context)this).getString("activity",
""");
    Intrinsic.checkExpressionValueIsNotNull(localObject, "act");
    localObject = (CharSequence)localObject;
    int j = 0;
    int k;
    for (int i = j; j < ((CharSequence)localObject).length(); i = k)
    {
        int m;
```

```

    if (((CharSequence) localObject).charAt(j) == paramChar) {
        m = 1;
    } else {
        m = 0;
    }
    k = i;
    if (m != 0) {
        k = i + 1;
    }
    j++;
}
return i;
}

```

Looks a little complex. But what i figured out is that it is fetching **f** and **p** value from **SharedPreferences**.

I clicked on Feed, Play and Clean icons in mobile app. We can take shell access using utility called **adb** (Android Debug Bridge)

I checked **/data/data/com.fireeye.flarebear** folder and it does contain **shared_prefs** folder.

```

generic_x86_64:/ # cd /data/data/com.fireeye.flarebear
cd /data/data/com.fireeye.flarebear
generic_x86_64:/data/data/com.fireeye.flarebear # ls
ls
cache code_cache shared_prefs
generic_x86_64:/data/data/com.fireeye.flarebear #

```

It has **com.fireeye.flarebear_preferences.xml** file with below content.


```
generic_x86_64:/data/data/com.fireeye.flarebear/shared_prefs # cat *.xml
cat *.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <float name="poo" value="0.34" />
  <string name="activity">fpc</string>
  <int name="mass" value="8" />
  <int name="happy" value="5" />
  <string name="name">nice</string>
  <int name="clean" value="4" />
</map>
generic_x86_64:/data/data/com.fireeye.flarebear/shared_prefs #
```

Cool so it is taking **f** and **p** values from here. But what are **mass**, **happy** and **clean** ?

I believe when we click **feed** it does assign some values to **mass**, **happy** and **clean** params. Also same for **play** and **clean**

```
public final void feed(@NotNull View paramView)
{
    Intrinsic.checkParameterIsNotNull(paramView, "view");
    saveActivity("f");
    changeMass(10);
    changeHappy(2);
    changeClean(-1);
    incrementPooCount();
    feedUi();
}

public final void play(@NotNull View paramView)
{
    Intrinsic.checkParameterIsNotNull(paramView, "view");
    saveActivity("p");
    changeMass(-2);
    changeHappy(4);
    changeClean(-1);
    playUi();
}
```

```

public final void clean(@NotNull View paramView)
{
    Intrinsics.checkNotNull(paramView, "view");
    saveActivity("c");
    removePoo();
    cleanUi();
    changeMass(0);
    changeHappy(-1);
    changeClean(6);
    setMood();
}

```

So we can list down some math stuff here.

F : M = 10, H=2, C=-1 Poo=.34

P : M=-2, H=4, C=-1

C : M=0, H=-1, C=6 Poo=0

F (Feed), P (Play), C (Clean), M (Mass), H (Happy), C (Clean)

Instead fiddling with first check **isHappy()** i went ahead to bypass next check called **isEcstatic()**

```

public final boolean isEcstatic()
{
    boolean bool2 = false;
    int i = getState("mass", 0);
    int j = getState("happy", 0);
    int k = getState("clean", 0);
    boolean bool1 = bool2;
    if (i == 72)
    {
        bool1 = bool2;
        if (j == 30)
        {
            bool1 = bool2;
            if (k == 0) {
                bool1 = true;
            }
        }
    }
    return bool1;
}

```

This code looks much simpler than before. We have to create a pattern to match M=72, H=30 and C=0 if it matches then we can get flag from app.

To make it true i've identified below pattern

$F*8+P*4+C*2$ then values become $M = 72$ and $H = 30$ and $C = 0$ which makes the above condition true.

I've clicked Feed 8 times and Play 4 times then Clean 2 times which showed up the flag.



That was such a cool challenge. Out of curiosity I went ahead identifying where this flag being placed as we have source with us.

```



public final void danceWithFlag()
{
    Object localObject1 = getResources().openRawResource(2131427328);
    Intrinsic.checkExpressionValueIsNotNull(localObject1, "ecstaticEnc");
    Object localObject2 = ByteStreamsKt.readBytes((InputStream)localObject1);
    localObject1 = getResources().openRawResource(2131427329);
    Intrinsic.checkExpressionValueIsNotNull(localObject1, "ecstaticEnc2");
    byte[] arrayOfByte = ByteStreamsKt.readBytes((InputStream)localObject1);
    localObject1 = getPassword();
    try
    {
        localObject2 = decrypt((String)localObject1, (byte[])localObject2);
        localObject1 = decrypt((String)localObject1, arrayOfByte);
        localObject2 = BitmapFactory.decodeByteArray((byte[])localObject2, 0, localObject2.length);
        localObject2 = new BitmapDrawable(getResources(), (Bitmap)localObject2);
        localObject1 = BitmapFactory.decodeByteArray((byte[])localObject1, 0, localObject1.length);
        localObject1 = new BitmapDrawable(getResources(), (Bitmap)localObject1);
        dance((Drawable)localObject2, (Drawable)localObject1);
        return;
    }
    catch (Exception localException)
    {
        for (;;) {}
    }
}

```

You see two files ? **ecstaticEnc**, **ecstaticEnc2**

But where are they ?

I checked resource directory inside the apk and it has a folder called raw.

This PC > Downloads > Flarebear > 3 - Flarebear > flarebear - Copy > res > raw				
Name	Date modified	Type	Size	
 ecstatic	22-05-2019 16:55	File	433 KB	
 ecstatic2	22-05-2019 16:55	File	454 KB	

Cool we have those two encrypted files with us. I believe one file will provide one image view with left side dance move of bear. And similarly second one.

We can also see how the password getting constructed

```

public final String getPassword()
{

```

```
int i = getStat('f');
int j = getStat('p');
int k = getStat('c');
String str2 = "";
String str1;
switch (i % 9)
{
default:
    str1 = "";
    break;
case 8:
    str1 = "*";
    break;
case 7:
    str1 = "&";
    break;
case 6:
    str1 = "@";
    break;
case 5:
    str1 = "#";
    break;
case 4:
    str1 = "!";
    break;
case 3:
    str1 = "+";
    break;
case 2:
    str1 = "$";
    break;
case 1:
    str1 = "-";
    break;
case 0:
    str1 = "_";
}
switch (k % 7)
{
default:
    str2 = "";
    break;
case 6:
    str2 = "@";
```

```

        break;
    case 4:
        str2 = "&";
        break;
    case 3:
        str2 = "#";
        break;
    case 2:
        str2 = "+";
        break;
    case 1:
        str2 = "_";
        break;
    case 0:
        str2 = "$";
    }
    String str5 = StringsKt.repeat((CharSequence)"flare", i / k);
    String str3 = StringsKt.repeat((CharSequence)rotN("bear", i * j), j * 2);
    String str4 = StringsKt.repeat((CharSequence)"yeah", k);
    StringBuilder localStringBuilder = new StringBuilder();
    localStringBuilder.append(str5);
    localStringBuilder.append(str1);
    localStringBuilder.append(str3);
    localStringBuilder.append(str2);
    localStringBuilder.append(str4);
    return localStringBuilder.toString();
}

```

Then it calls decrypt() function

```

public final byte[] decrypt(@NotNull String paramString, @NotNull byte[] paramArrayOfByte)
{
    Intrinsic.checkParameterIsNotNull(paramString, "password");
    Intrinsic.checkParameterIsNotNull(paramArrayOfByte, "data");
    Object localObject1 = Charset.forName("UTF-8");
    Intrinsic.checkExpressionValueIsNotNull(localObject1, "Charset.forName(charsetName)");
    localObject1 = "pawsitive_vibes!".getBytes((Charset)localObject1);
    Intrinsic.checkExpressionValueIsNotNull(localObject1, "(this as java.lang.String).getBytes(charset)");
    localObject1 = new IvParameterSpec((byte[])localObject1);
    paramString = paramString.toCharArray();
    Intrinsic.checkExpressionValueIsNotNull(paramString, "(this as java.lang.String).toCharArray()");
    Object localObject2 = Charset.forName("UTF-8");
    Intrinsic.checkExpressionValueIsNotNull(localObject2, "Charset.forName(charsetName)");
    localObject2 = "NaClNaClNaCl".getBytes((Charset)localObject2);
    Intrinsic.checkExpressionValueIsNotNull(localObject2, "(this as java.lang.String).getBytes(charset)");
    paramString = new PBEKeySpec(paramString, (byte[])localObject2, 1234, 256);
    paramString = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateSecret((KeySpec)paramString);
    Intrinsic.checkExpressionValueIsNotNull(paramString, "secretKeyFactory.generateSecret(pbKeySpec)");
    localObject2 = new SecretKeySpec(paramString.getEncoded(), "AES");
    paramString = Cipher.getInstance("AES/CBC/PKCS7Padding");
    paramString.init(2, (Key)localObject2, (AlgorithmParameterSpec)localObject1);
    paramString = paramString.doFinal(paramArrayOfByte);
    Intrinsic.checkExpressionValueIsNotNull(paramString, "cipher.doFinal(data)");
    return paramString;
}

```

We could see SALT='NaClNaClNaCl' and IV will be 'pawsitive_vibes!' we can also aware that they used **AES/CBC/PKCS7Padding** to encrypt this flag. Using the above information we can get the flag by decrypting it.

Before doing that we should know what is the password. So we must know what are i,j and k values must be.

i = getStat('f')

```

public final int getStat(char paramChar) {
    String str =
    PreferenceManager.getDefaultSharedPreferences((Context)this).getString("activity",
    "");
    Intrinsic.checkExpressionValueIsNotNull(str, "act");
    CharSequence charSequence = (CharSequence)str;
    byte b1 = 0;
    byte b2;
    for (b2 = b1; b1 < charSequence.length(); b2 = b) {
        boolean bool;
        if (charSequence.charAt(b1) == paramChar) {
            bool = true;
        } else {
            bool = false;
        }
        byte b = b2;
    }
}

```

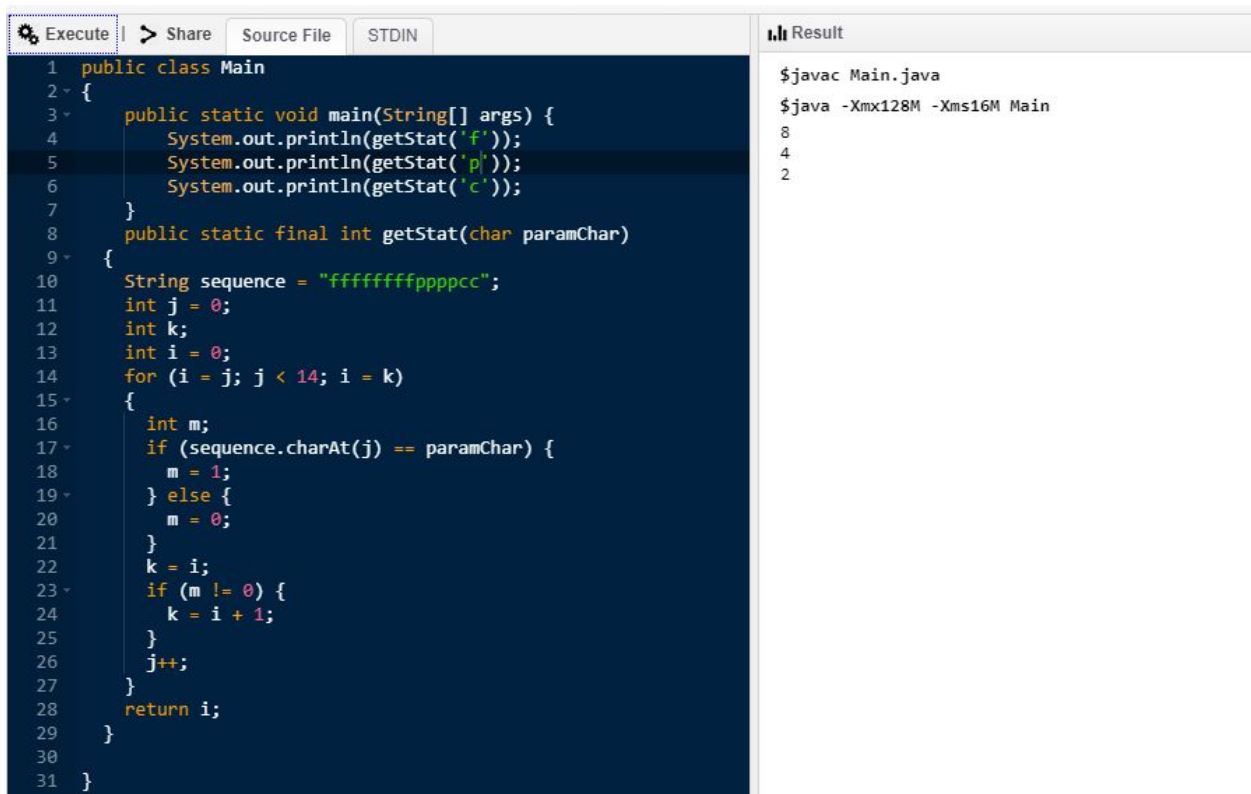


```

    if (bool)
        b = b2 + true;
    b1++;
}
return b2;
}

```

I've modified this code according to the pattern in shared_prefs and it gave output as **8,4,2**



The screenshot shows an IDE with a code editor on the left and a result pane on the right. The code editor contains the following Java code:

```

1 public class Main
2 {
3     public static void main(String[] args) {
4         System.out.println(getStat('f'));
5         System.out.println(getStat('p'));
6         System.out.println(getStat('c'));
7     }
8     public static final int getStat(char paramChar)
9     {
10        String sequence = "ffffffffppppcc";
11        int j = 0;
12        int k;
13        int i = 0;
14        for (i = j; j < 14; i = k)
15        {
16            int m;
17            if (sequence.charAt(j) == paramChar) {
18                m = 1;
19            } else {
20                m = 0;
21            }
22            k = i;
23            if (m != 0) {
24                k = i + 1;
25            }
26            j++;
27        }
28        return i;
29    }
30 }
31 }

```

The result pane shows the following output:

```

$javac Main.java
$java -Xmx128M -Xms16M Main
8
4
2

```

Now we are aware of i,j and k values. Now the code became

```

public final String getPassword()
{
    int i = 8;
    int j = 4;
    int k = 2;
    String str2 = "";
    String str1;

```



```
switch (i % 9)
{
default:
    str1 = "";
    break;
case 8:
    str1 = "*";
    break;
case 7:
    str1 = "&";
    break;
case 6:
    str1 = "@";
    break;
case 5:
    str1 = "#";
    break;
case 4:
    str1 = "!";
    break;
case 3:
    str1 = "+";
    break;
case 2:
    str1 = "$";
    break;
case 1:
    str1 = "-";
    break;
case 0:
    str1 = "_";
}
switch (k % 7)
{
default:
    str2 = "";
    break;
case 6:
    str2 = "@";
    break;
case 4:
    str2 = "&";
    break;
case 3:
```

```

        str2 = "#";
        break;
    case 2:
        str2 = "+";
        break;
    case 1:
        str2 = "_";
        break;
    case 0:
        str2 = "$";
    }
    String str5 = StringsKt.repeat((CharSequence)"flare", i / k);
    String str3 = StringsKt.repeat((CharSequence)rotN("bear", i * j), j * 2);
    String str4 = StringsKt.repeat((CharSequence)"yeah", k);
    StringBuilder localStringBuilder = new StringBuilder();
    localStringBuilder.append(str5);
    localStringBuilder.append(str1);
    localStringBuilder.append(str3);
    localStringBuilder.append(str2);
    localStringBuilder.append(str4);
    return localStringBuilder.toString();
}

```

You could see password is being constructed using several string place holders. As we know i,j and k values we can easily figure out str1 and str2 values.

Str1 = “*”

Str2 = “+”

What is repeat function ? Let’s dig into that.

String str5 = StringsKt.repeat((CharSequence)"flare", i / k);

The repeat function code is as below.

```

public static final String repeat(CharSequence paramCharSequence, int paramInt)
{
    int k = 0;
    int m = 1;
    int j;
    if (paramInt >= 0) {

```

```

    j = 1;
} else {
    j = 0;
}
if (j != 0)
{
    String str = "";
    Object localObject = str;
    if (paramInt != 0) {
        if (paramInt != 1)
        {
            j = paramCharSequence.length();
            localObject = str;
            if (j != 0) {
                if (j != 1)
                {
                    localObject = new StringBuilder(paramCharSequence.length() * paramInt);
                    if (1 <= paramInt) {
                        for (j = m;; j++)
                        {
                            ((StringBuilder)localObject).append(paramCharSequence);
                            if (j == paramInt) {
                                break;
                            }
                        }
                    }
                }
                localObject = ((StringBuilder)localObject).toString();
            }
        }
        else
        {
            int i = paramCharSequence.charAt(0);
            paramCharSequence = new char[paramInt];
            j = paramCharSequence.length;
            for (paramInt = k; paramInt < j; paramInt++) {
                paramCharSequence[paramInt] = i;
            }
            localObject = new String(paramCharSequence);
        }
    }
}
else
{
    localObject = paramCharSequence.toString();
}
}

```

```

    }
    return (String)localObject;
}
}

```

It takes two arguments. I've tried compiling the above code and it produced below output.

```

1 public class Main
2 {
3     public static void main(String[] args) {
4         Object localObject="";
5         int k = 0;
6         int m = 1;
7         int j;
8         CharSequence paramCharSequence = "flare";
9         int paramInt=4;
10        if (paramInt >= 0) {
11            j = 1;
12        } else {
13            j = 0;
14        }
15        if (j != 0)
16        {
17            String str = "";
18            localObject = str;
19            if (paramInt != 0) {
20                if (paramInt != 1)
21                {
22                    j = paramCharSequence.length();
23                    localObject = str;
24                    if (j != 0) {
25                        if (j != 1)
26                        {
27                            localObject = new StringBuilder(paramCharSequence.length() * paramInt);
28                            if (1 <= paramInt) {
29                                for (j = m;; j++)
30                                {
31                                    ((StringBuilder)localObject).append(paramCharSequence);
32                                    if (j == paramInt) {
33                                        break;
34                                    }
35                                }
36                            }
37                            localObject = ((StringBuilder)localObject).toString();
38                        }
39                    }
40                }
41            }
42        }
43        else
44        {
45            localObject = paramCharSequence.toString();
46        }
47        System.out.println((String)localObject);
48    }
49 }
50 }
51 }
52 }

```

Result

```

$javac Main.java
$java -Xmx128M -Xms16M Main
flareflareflareflare

```

Now **str5 = 'flareflareflare'**

Also there's `rotN()` function. Which does capitalization of given string.


```

        SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        KeySpec keySpec = new
PBEKeySpec(password.toCharArray(),salt,1234,256);
        SecretKey tmp = factory.generateSecret(keySpec);
        SecretKey secret = new SecretKeySpec(tmp.getEncoded(), "AES");
        // file decryption
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secret, new IvParameterSpec(iv));
        FileInputStream fis = new FileInputStream("ecstatic");
        FileOutputStream fos = new FileOutputStream("ecstatic_decrypted");
        byte[] in = new byte[64];
        int read;
        while ((read = fis.read(in)) != -1) {
            byte[] output = cipher.update(in, 0, read);
            if (output != null)
                fos.write(output);
        }
        byte[] output = cipher.doFinal();
        if (output != null)
            fos.write(output);
        fis.close();
        fos.flush();
        fos.close();
        System.out.println("File Decrypted.");
    }
}

```

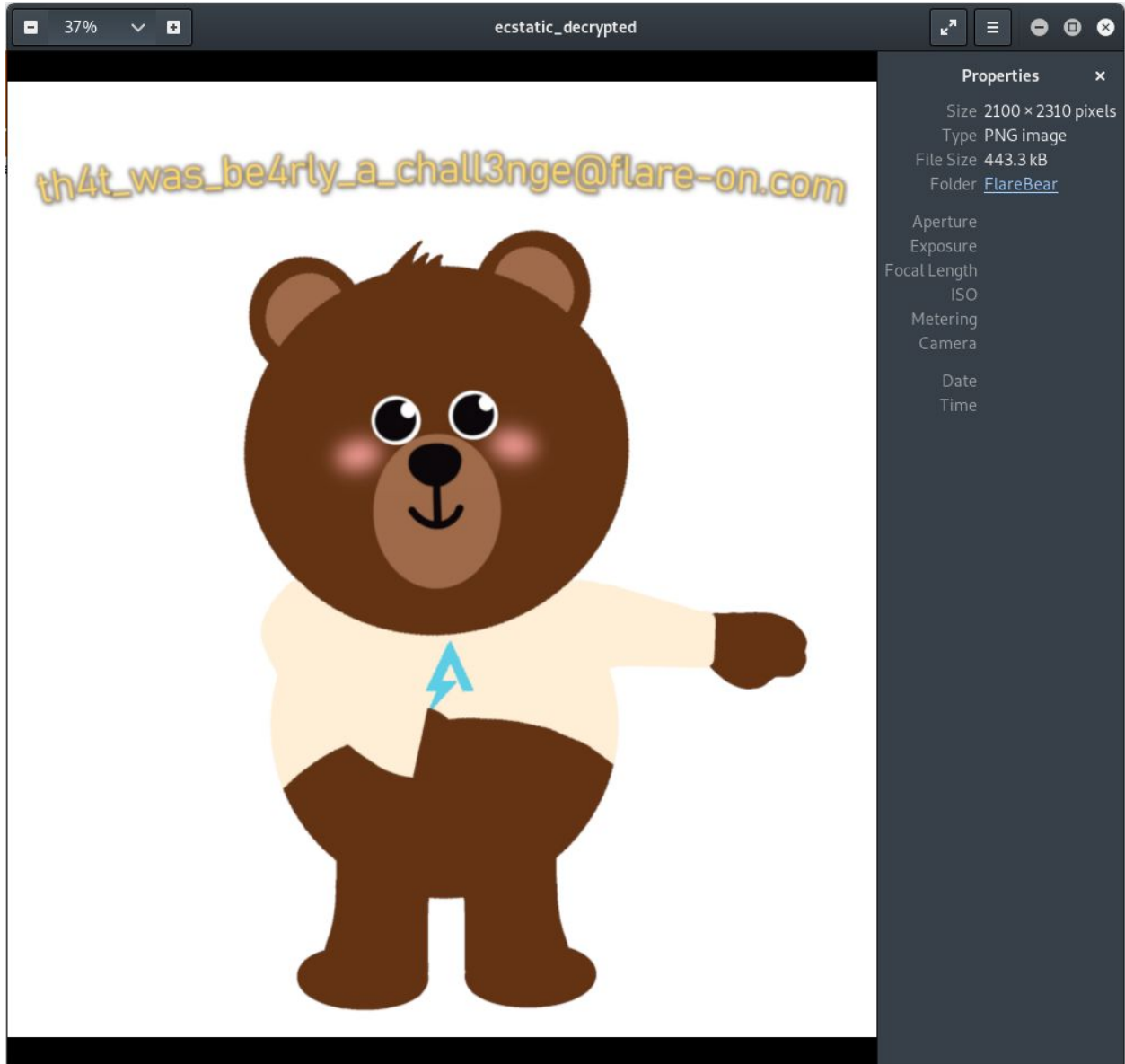
You could see author wrote decrypt() function using **PKCS7Padding** but that doesn't make much difference with **PKCS5Padding** so i just proceeded with above padding.

```

root@MrR3boot: ~/FlareBear$ javac AESFileDecryption.java
root@MrR3boot: ~/FlareBear$ java AESFileDecryption
File Decrypted.
root@MrR3boot: ~/FlareBear$ nano AESFileDecryption.java
root@MrR3boot: ~/FlareBear$ javac AESFileDecryption.java
root@MrR3boot: ~/FlareBear$ java AESFileDecryption
Exception in thread "main" javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
    at java.base/com.sun.crypto.provider.CipherCore.unpad(CipherCore.java:975)
    at java.base/com.sun.crypto.provider.CipherCore.fillOutputBuffer(CipherCore.java:1056)
    at java.base/com.sun.crypto.provider.CipherCore.doFinal(CipherCore.java:853)
    at java.base/com.sun.crypto.provider.AESCipher.engineDoFinal(AESCipher.java:446)
    at java.base/javax.crypto.Cipher.doFinal(Cipher.java:2085)
    at AESFileDecryption.main(AESFileDecryption.java:35)

```

For **ecstatic** I haven't faced any issues but for **ecstatic2** it didn't decrypted properly. Then switching back to java 1.7 sdk resolved the issue.



th4t_was_be4rly_a_chall3nge@flare-on.com



Properties

Size 2100 × 2310 pixels

Type PNG image

File Size 443.3 kB

Folder [FlareBear](#)

Aperture

Exposure

Focal Length

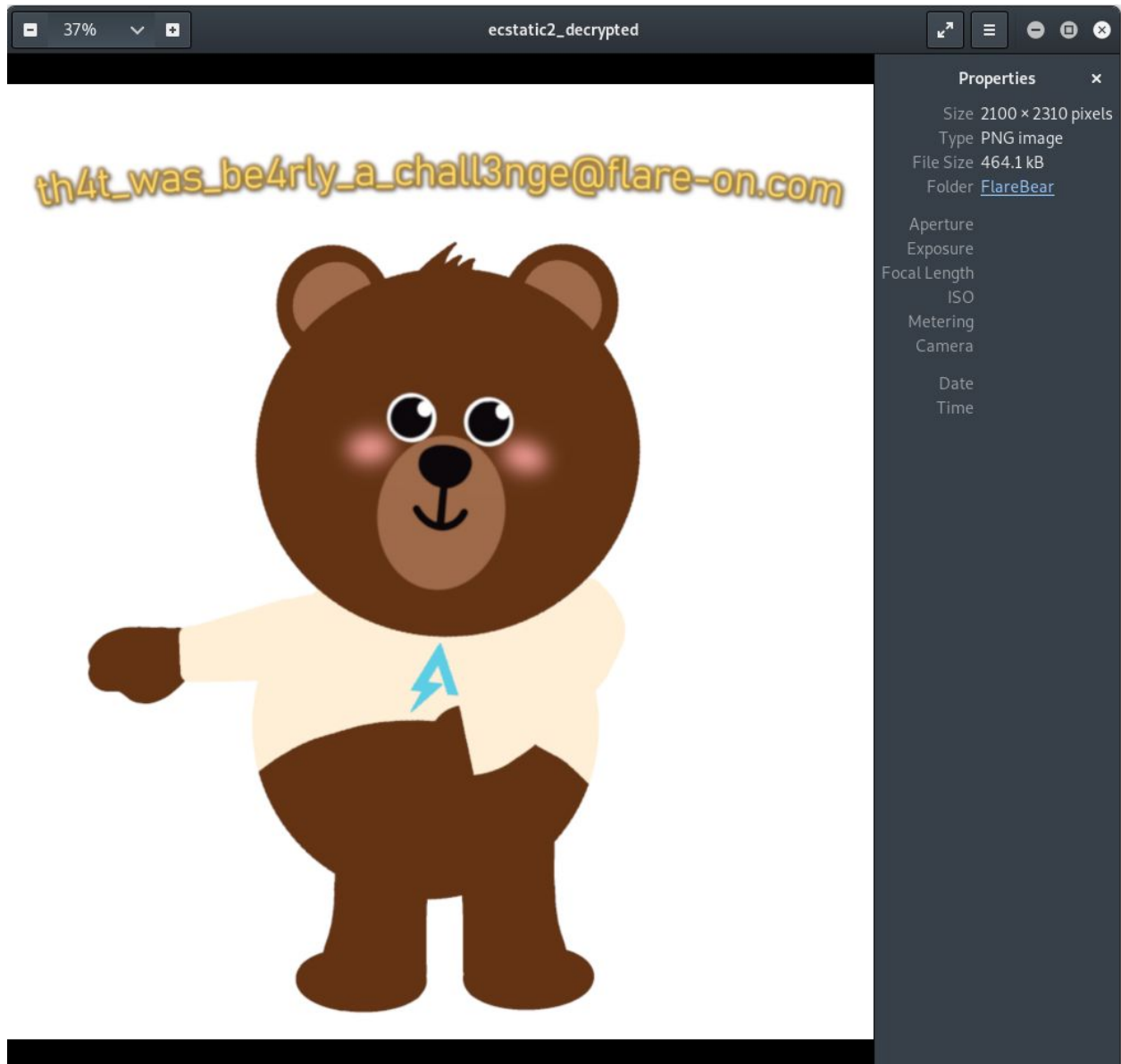
ISO

Metering

Camera

Date

Time



References:

1. <https://gist.github.com/scotttam/874426/e5a0e1317995e9388083eb455c5bb160ec2e1afd>
2. <https://stackoverflow.com/questions/19856324/exception-in-thread-main-java-security-invalidkeyexception-illegal-key-size-o>
3. <https://www.attachmate.com/documentation/rweb/rweb-installguide/data/b1gdutii.htm>
4. <https://facingissuesonit.com/2017/10/30/exception-java-security-nosuchalgorithm-exception-cannot-find-any-provider-supporting-aesecbpkcs7padding/>