

ItsOnFire

Challenge Description

The FLARE team is now enthusiastic about Google products and services but we suspect there is more to this Android game than meets the eye.

Files

Filename	Size	SHA256
ItsOnFire.apk	13,873,717 bytes	eb6e69930b39a97ea4fa8a3c50d54ab7f13ea1223fa9bd2bedca0bbd000b0d92

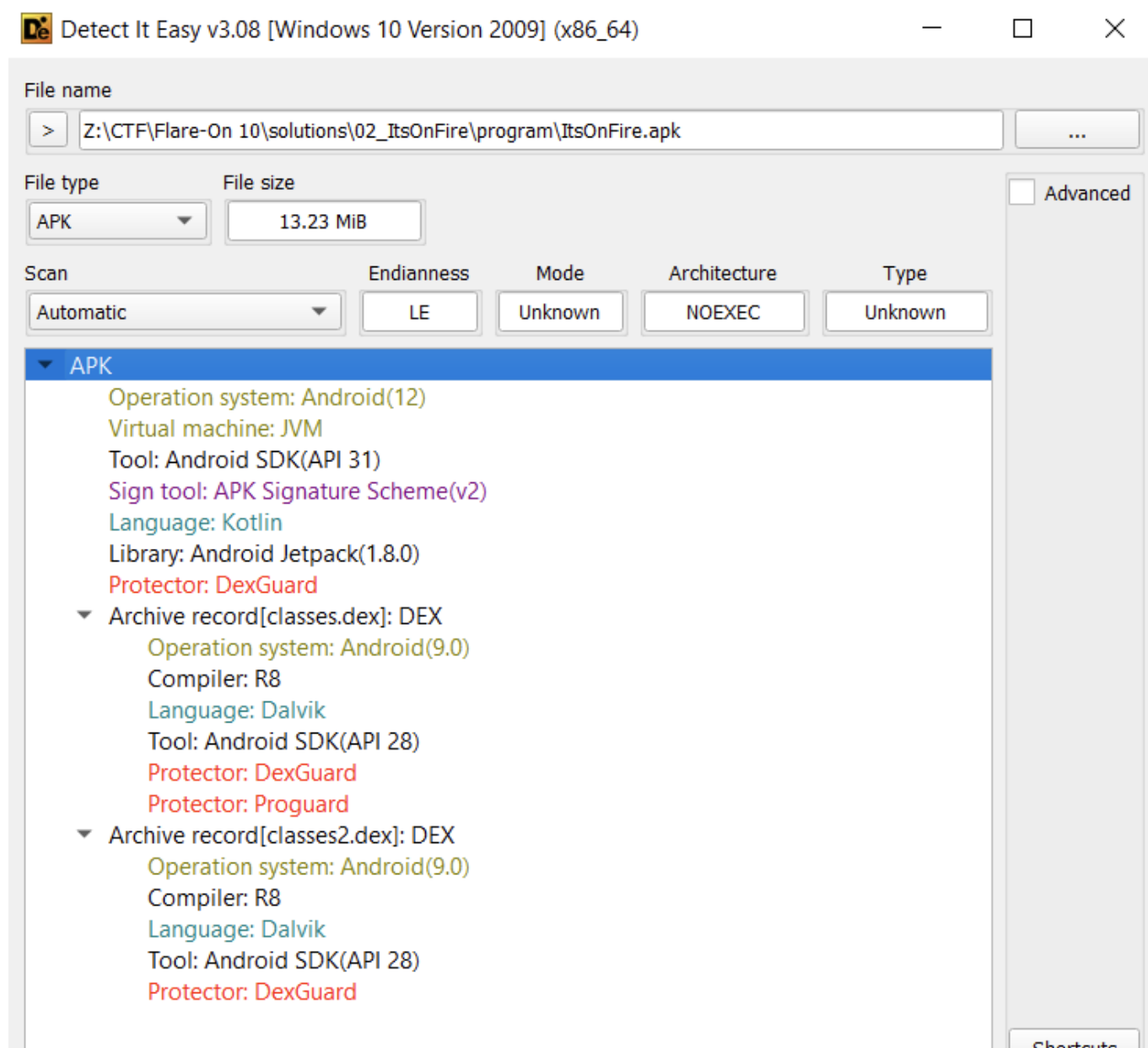
High-Level Summary

- TODO

Analysis

Basic Static

Detect It Easy



- Protector **DexGuard** & **ProGuard**

- **ProGuard** is an open-sourced, versatile optimizer for Kotlin, Java, and other JVM languages. ProGuard is primarily used to shrink apps. In fact, ProGuard makes apps up to 90% smaller and 20% faster, while providing some code obfuscation techniques for basic protection against reverse engineering.
- **DexGuard** provides all the functionality of ProGuard with the addition of numerous code hardening techniques to guard mobile applications against reverse engineering and tampering. DexGuard is primarily intended for Android mobile applications which need robust code protection and code optimization.
- DexGuard applies multiple obfuscation and encryption techniques to the app's code and SDKs, providing a layered protection profile. Beyond the code enhancements, it integrates RASP (runtime application self-protection) mechanisms, making it virtually impossible to gain access to the app's internal logic. A key advantage of DexGuard is that it leverages polymorphism; each app build has a different obfuscation configuration. This means that any knowledge or success a threat actor may gain is reset to zero with each app build and release.

We extract the APK file with 7z and (partially) decompile with **jadx**. It won't work 100% due to errors likely based on the APK protections.

Basic Dynamic

None yet, FOMO Invader game 🙄

Dirty Notes

- maybe interesting code
 - com.secure.itsonfire.PostByWeb
 - @Metadata d1 -> possibly encoded data -> looks like that is part of many classes
 - Seems to connect to an HTTP endpoint to url given to Constructor as param str
- from code: seems to be some kind of space invader game
- main logic in com/secure/itsonfire/MalwareInvadersView.java
 - there does not seem to be a winning condition?
- resources/res/raw contains a.png and b.png, both seem encrypted/encoded
- how did i get here? i just randomly opened decompiled java files
- TODO: describe a better lead to here, maybe from C2 string
- [f/b.java](#) seems to contain some crypto and raw resource reading code
 - i.e. part of function c():
 - e() reads the content of a raw resource (likely encrypted png)
 - a() is crc32
 - e() and d() are called from c()
 - c() is called from f() -> like called from the outside
 - b() wraps crypto, called from c()
- Looks like c() uses
 - e() to load resource bytes (possibly encrypted png)
 - d() to somehow retrieve data for SecretKeySpec
 - R.string.f2321d to get the crypto algo name?
 - R.string.f2337t to get IV value
 - then decrypts raw resource with b(string, e2, secretKeySpec, new IvParameterSpec(bytes2));
 - writes decrypted data to cachedir

```
byte[] b2 = b(string, e2, secretKeySpec, new IvParameterSpec(bytes2));
File file = new File(context.getCacheDir(),
context.getString(R.string.D));
FilesKt__FileReadWriteKt.writeBytes(file, b2);
```

- these functions may be used to decrypt the raw resources a.png / b.png which seem encrypted
- Grepping for f2321d and f2337t yield
 - `public static final int f2321d = 0x7f0e001e; in sources/com/secure/itsonfire/R.java`
 - `public static final int f2337t = 0x7f0e0049;`
- Grep for 0x7f0e001e yields `resources/res/values/public.xml: <public type="string" name="d" id="0x7f0e001e" />`
- and for 0x7f0e0049 yields `resources/res/values/public.xml: <public type="string" name="t" id="0x7f0e0049" />`
- d could refer to this string in `resources/res/values/strings.xml: <string name="d">AES/CBC/PKCS5Padding</string>`
- part of c() to retrieve the cypto key

```
String d2 = d(context);
Charset charset = Charsets.UTF_8;
byte[] bytes = d2.getBytes(charset);
Intrinsics.checkNotNullExpressionValue(bytes, "this as
java.lang.String).getBytes(charset)");
SecretKeySpec secretKeySpec = new SecretKeySpec(bytes,
context.getString(R.string.f2319b));
```

- f2319b -> 0x7f0e001c -> string name b -> "AES" (in strings.xml)
- so d(context) is the magic that should retrieve the key bytes

```
private final String d(Context context) {
    String string = context.getString(R.string.f2325h);
    Intrinsics.checkNotNullExpressionValue(string,
"context.getString(R.string.c2)");
    String string2 = context.getString(R.string.0);
    Intrinsics.checkNotNullExpressionValue(string2,
"context.getString(R.string.w1)");
    StringBuilder sb = new StringBuilder();
    sb.append(string.subSequence(4, 10));
    sb.append(string2.subSequence(2, 5));
    String sb2 = sb.toString();
    Intrinsics.checkNotNullExpressionValue(sb2,
"StringBuilder().apply(builderAction).toString()");
    byte[] bytes = sb2.getBytes(Charsets.UTF_8);
    Intrinsics.checkNotNullExpressionValue(bytes, "this as
java.lang.String).getBytes(charset)");
    long a2 = a(bytes);
    StringBuilder sb3 = new StringBuilder();
    sb3.append(a2);
    sb3.append(a2);
```

```
String sb4 = sb3.toString();
Intrinsics.checkNotNullExpressionValue(sb4,
"StringBuilder().apply(builderAction).toString()");
return StringsKt___StringsKt.slice(sb4, new IntRange(0, 15));
}
```

- R.string.f2325h -> 0x7f0e0023 -> string name h -> https://flare-on.com/evilc2server/report_token/report_token.php?token=
- also used in `out/sources/com/secure/itsonfire/MessageWorker.java`: `String str2 = getString(R.string.f2325h) + str;`
 - this could be C2 comms
- R.string.O = "wednesday"
- new string sb gets C2 subSequence(4,10) and O subSequence(2,5)

```
public CharSequence subSequence(int start, int end)
Parameters:
start - This is the index from where the subsequence starts, it is inclusive.
end - This is the index where the subsequence ends, it is exclusive.
```

- C2 subSequence(4,10) = "s://fl"

```
>>> c2 = "https://flare-on.com/evilc2server/report_token/report_token.php?token="
>>> c2[4:10]
b's://fl'
```

- O subSequence(2,5) = "dne"

```
>>> O = "wednesday"
>>> O[2:5]
'dne'

>>> c2 = "https://flare-on.com/evilc2server/report_token/report_token.php?token="
>>> O = "wednesday"
>>> c2[4:10] + O[2:5]
's://fldne'
```

- is then passed into `a()`, which calculates `crc32`
- `crc32` of "s://fldne" = 450830537

```
>>> from binascii import crc32
>>> crc32(bytes(c2[4:10] + O[2:5], "UTF-8"))
450830537
>>> hex(crc32(bytes(c2[4:10] + O[2:5], "UTF-8")))
'0x1adf20c9'
```

then the crc32 values is concatenated with itself to string sb4 sb4 should be "450830537450830537" then

from this sb4 value, d() returns `StringsKt___StringsKt.slice(sb4, new IntRange(0, 15));`

```
>>> double_crc32 = "450830537450830537"
>>> double_crc32[0:16]
'4508305374508305'
```

more interesting strings from that file

```
<string name="h">https://flare-on.com/evilc2server/report_token/report_token.php?
token=
```

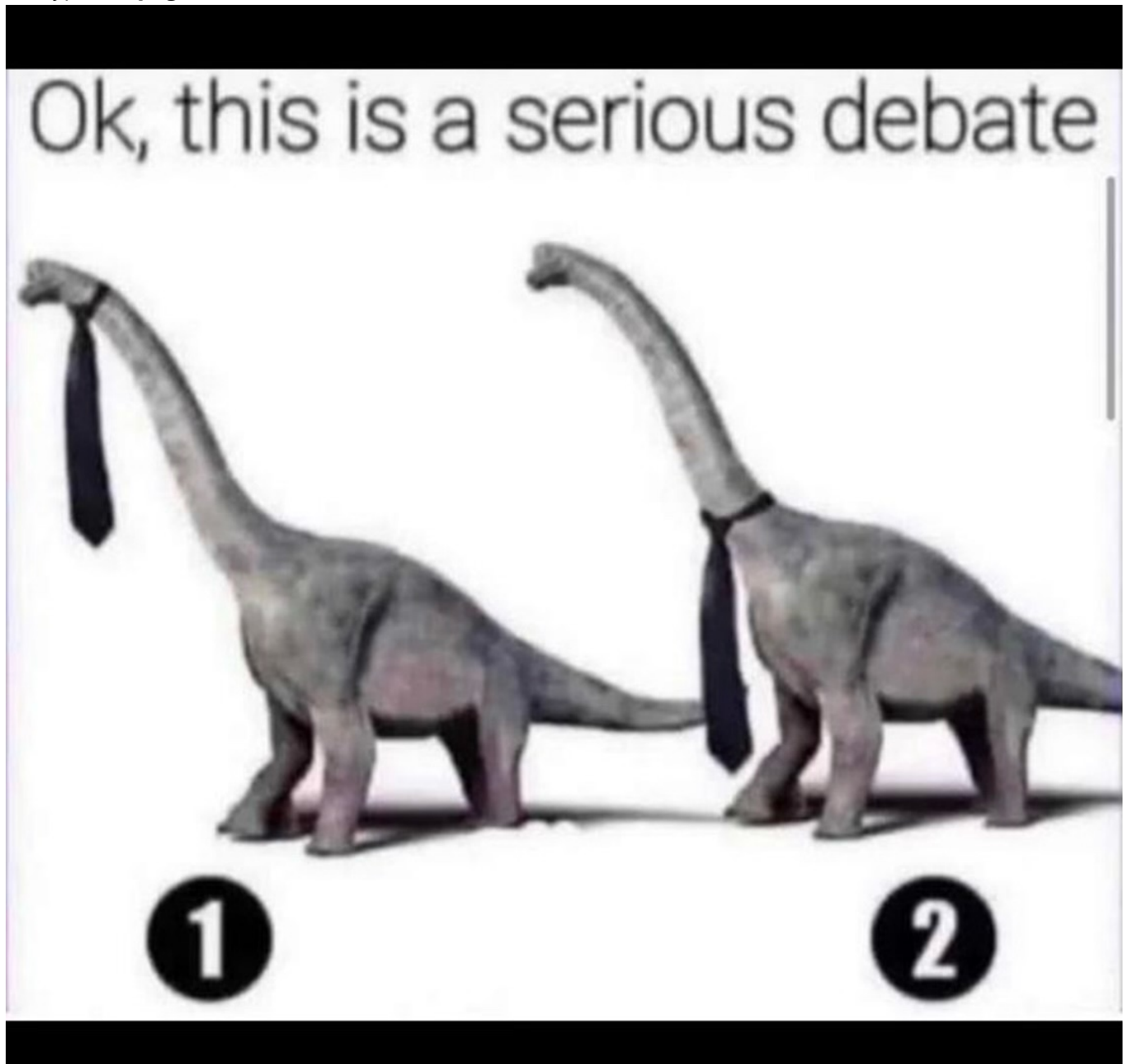
- would be awesome to really understand how this C2 URL is used and what for...
- summarizing crypto c():
 - crypto algo name: f2321d = 0x7f0e001e -> public string d -> AES/CBC/PKCS5Padding
 - iv value: f2337t = 0x7f0e0049 -> public string t -> abcdefghijklmnop
 - key value "4508305374508305"

Solve Script

[Decrypt a.png](#)

Decrypted **a.png**

Y0Ur3_0N_F1r3_K33P_601N6@flare-on.com

Decrypted **b.png**

Flag

Flag: Y0Ur3_0N_F1r3_K33P_601N6@flare-on.com