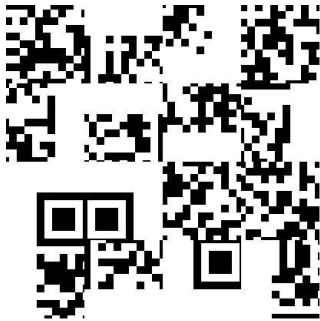


(ES)

Hemos recibido un mensaje anónimo: “La bomba hará explosión en 30 segundos”. Solamente nos han dado unas coordenadas, y hemos descubierto que pertenecen a una base secreta de agentes químicos. Te has conseguido infiltrar y lo que parece ser una imagen que da acceso a la bomba en cuestión. El problema es que esta ha sido modificada para asegurarse de que nadie la desactiva. ¿Podrás conseguirlo?

Mucha suerte.





Y en la Web aparece esta imagen:



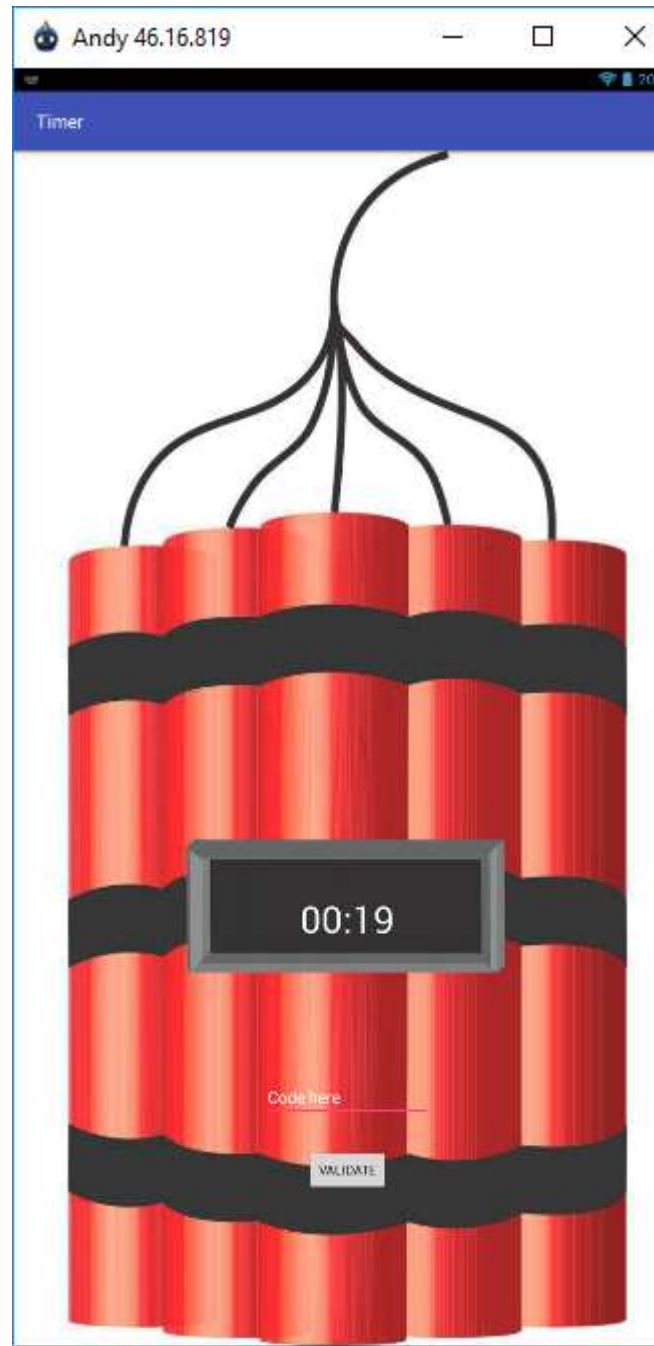
- Lo primer componemos el puzle del QR, colocamos cada pieza en su sitio con un editor de imágenes. Para ello primero cortamos los bordes de la imagen que sobran (derecha y abajo) y ponemos las líneas Guía a 102 px cada una (para un total de 408). Vamos copiando y pegando en otra imagen, y colocando en su posición:



- y el QR nos llega a la URL:
  - o <http://www.mediafire.com/file/lj5uzz3ogppq5n0/timer.apk>
- Descargamos el apk, que es una aplicación Android. Lo convertimos a .jar y lo descompactamos con el WinRar:

	kotlin	15/01/2018 13:14	Carpeta de archivos	
	META-INF	15/01/2018 13:14	Carpeta de archivos	
	res	15/01/2018 13:14	Carpeta de archivos	
	AndroidManifest.xml	15/01/2018 13:14	XML Document	2 KB
	classes.dex	15/01/2018 13:14	Archivo DEX	3.893 KB
	resources.arsc	15/01/2018 13:14	Archivo ARSC	221 KB

- Lo ejecutamos dentro de un emulador (Andy v.46), y vemos que nos pide un texto, y si tardamos más de los 30 segundos da error:



- Decompilamos el apk y el dex con distintas herramientas, y parte del código está ofuscado, por lo que no podemos saber exactamente lo que hace. Con la unión de las siguientes tres herramientas saco la máxima información, cada una da lo suyo:
  - o Decompilamos con dex2jar.jar y visualizamos con JD-GUI, da un resultado bastante bueno, decompila algunas clases pero otras clases (como la MainActivity) da error internal.
  - o El JAXD-gui da mejor resultado, decompila todas las clases, da error de Decode en algunos métodos.
  - o Bajo también la herramienta apktool, que se usa apktool d timer.apk, que lo decompila y genera unos ficheros .smali que son como una ayuda para el reserving.
- Vemos que hay distintas funciones en la Clase MainActivity que no hacen nada, y el botón de la aplicación invoca a la función showCode().
- Copiamos y pegamos el código de esta función, más la clase SecurityKt a una aplicación java, y vamos eliminando las referencias que no valen.

- Del fichero MainActivityiy.smali sacamos por ejemplo el código de la función A, que luego se usa en la función ShowCode:

```

# The value of this static final field might be set in the static constructor
.field private static final A:Ljava/lang/String; = "AxQLEVZ1CD09ZB8nHxt1AQ8IFX4YcR85ZAB4GwcCHDYcb0dsSw"
.field public static final Companion:Lcom/uad/timer/MainActivity$Companion;
# The value of this static final field might be set in the static constructor
.field private static final SECRET_KEY:Ljava/lang/String; = "deck_riuk_rick_rock_rick_ruck_tirk_rack_rick_ring_truck_reck"

```

- Adaptamos algunas funciones como la de decode64 y encode64, que ya vienen en la Java 1.8. Y algunos de los métodos los voy toqueteando hasta que compile bien. Entonces genero un mail y lo lanzo:
- Al ejecutar la función nos devuelve la flag:
  - o **UAM{B0mb\_D1S4BL3D\_YOU\_R0CK}**
- El código final queda:
  - o Clase principal (CalculaFlag):

```

private void showCode() {
    try {
        String key = SecurityKt.base64Decode("VUFNXzM=");
        String base64Decode = SecurityKt.base64Decode(" " + "AxQLEVZ1CD09ZB8nHxt1AQ8IFX4YcR85ZAB4GwcCHDYcb0dsSw" + " ");
        byte[] bytes = key.getBytes("UTF8");
        String tempo = SecurityKt.encodeDecode(base64Decode, bytes);
        String flag = SecurityKt.base64Decode(tempo);
        Trazas.getLogger().error ("El flag es: " + flag);
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

Y la clase SecurityKt (ver los java):