

首先是构造Frida在Java层的主动调用环境

```
function testsiua(plainText){
    Java.perform(function () {
        var datacollect =
Java.use('com.meituan.android.common.datacollection.DataCollectionJni');
        var ByteString = Java.use("com.android.okhttp.okio.ByteString");
        var Base64 = Java.use("java.util.Base64");

        // 参数1是上下文
        var currentApplication=
Java.use("android.app.ActivityThread").currentApplication();
        var context = currentApplication.getApplicationContext();
        // 参数2是明文字节数组
        // 使用String类将我们传入的字符串转成数组
        var JavaString = Java.use("java.lang.String");
        var bytesStr = JavaString.$new(plainText).getBytes();
        // 参数3是明文的长度
        var textLength = plainText.length;

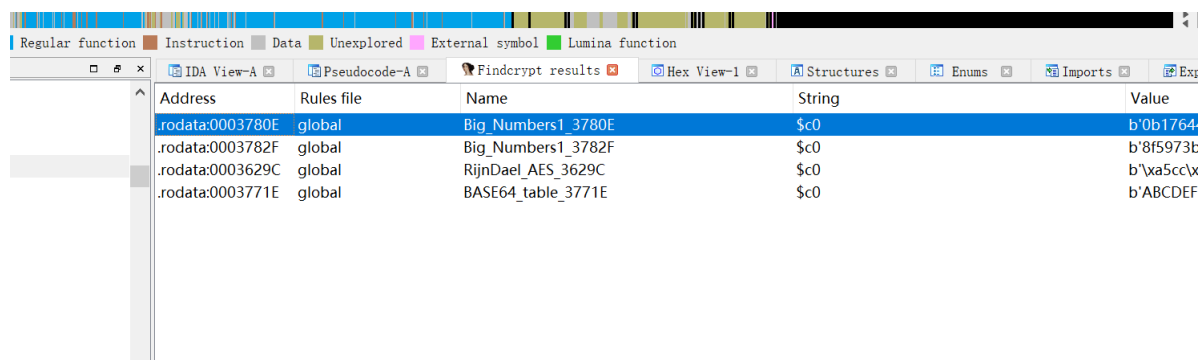
        // 主动调用
        var result = datacollect.packData(context, bytesStr, textLength)
        // 返回base64编码后的结果, 使用系统类转换成十六进制
        var base64result = ByteString.of(result).utf8()
        var base64before = Base64.getDecoder().decode(base64result);
        // console.log("base64 result:"+base64result);
        console.log("result:"+ByteString.of(base64before).hex());

    });
}
```

观察输入和输出

发现呈现出分组的情态, 且分组大小为32字节, 疑似AES。

进入SO, 使用Findcrypt



Address	Rules file	Name	String	Value
.rodata:0003780E	global	Big_Numbers1_3780E	\$c0	b'0b1764
.rodata:0003782F	global	Big_Numbers1_3782F	\$c0	b'8f5973t
.rodata:0003629C	global	Rijndael_AES_3629C	\$c0	b'\xa5ccx
.rodata:0003771E	global	BASE64_table_3771E	\$c0	b'ABCDEF

有一处AES

查找其交叉引用



```

        *((_BYTE *)v29 + i) = v6[i] ^ *((_BYTE *)v30 + i);
        sub_2F88(v29, v12, v28); // 出现位置1
        qmemcpy(v30, v12, 0x10u);
        a3 -= 16;
        v12 += 16;
        v6 += 16;
    }
    while ( a3 > 0xF );
    v7 += v23 + 16;
    v6 = v26;
    v9 = v27;
    a3 = v25 - v23;
}
if ( a3 )
{
    v14 = 0;
    do
    {
        *((_BYTE *)v29 + v14) = v6[v14] ^ *((_BYTE *)v30 + v14);
        ++v14;
    }
    while ( a3 != v14 );
    if ( a3 <= 0xF )
    {
        qmemcpy((char *)v29 + a3, (char *)v30 + a3, 16 - a3);
        sub_2F88(v29, v29, v28); // 出现位置2
        qmemcpy(v7, v29, 0x10u);
        v30[0] = v29[0];
        v30[1] = v29[1];
        v30[2] = v29[2];
        v30[3] = v29[3];
    }
}
}

```

位置1是明文长度大于一个分组的情况

位置2是明文长度只需要一个分组去加密的情况

显然位置2代码量更小，但逻辑是一样的，所以看2.

这个代码的理解依赖于对AES 实现源码的熟悉

```

void AES_CBC_encrypt_buffer(struct AES_ctx *ctx, uint8_t* buf, uint32_t length)
{
    uintptr_t i;
    uint8_t *Iv = ctx->Iv;
    for (i = 0; i < length; i += AES_BLOCKLEN)
    {
        XorWithIv(buf, Iv);
        Cipher((state_t*)buf, ctx->RoundKey);
        Iv = buf;
        buf += AES_BLOCKLEN;
    }
    /* store Iv in ctx for next call */
    memcpy(ctx->Iv, Iv, AES_BLOCKLEN);
}

```

对2F88进行静态分析或者动态调试

都可以发现这三个参数的意义



