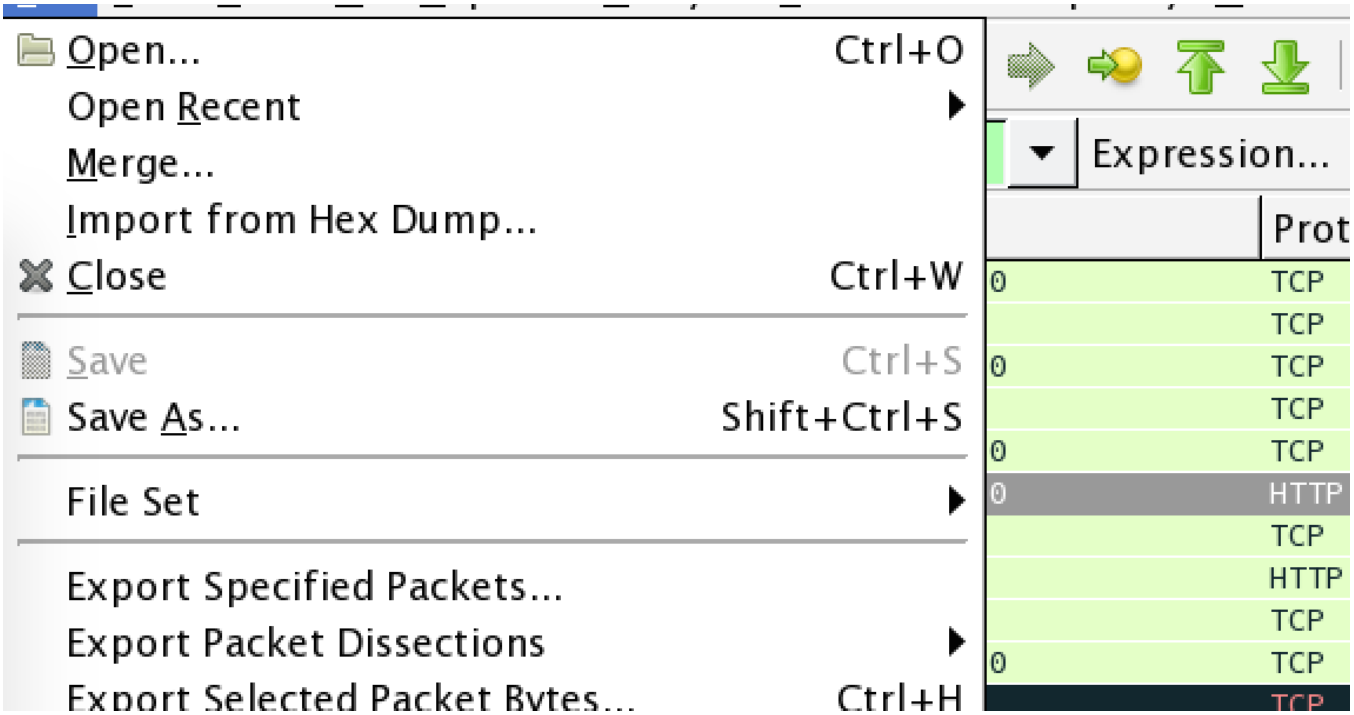
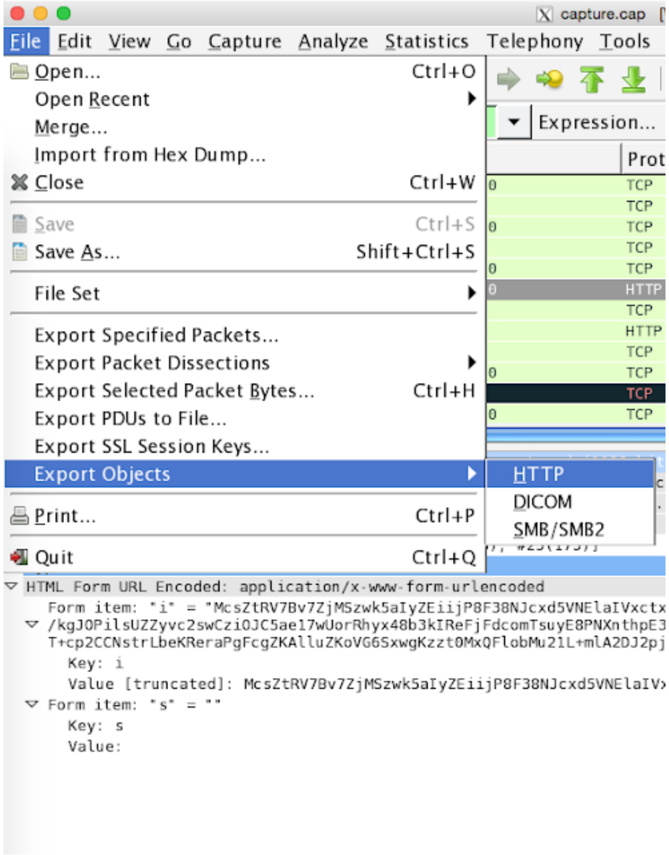
Reversing the C2C HTTP Emmental communication

blog.angelalonso.es/2015/10/reversing-c2c-http-emmental.html



In last post I explained how it was possible to decrypt the initial C&C communication from the data dumped from memory, with the support of a python script. In this post, I am going to follow the same approach, but using the information from the captured network traffic.
For that I will capture with Wireshark all the communication with the C&C while the malware is running. Then I can export all the 'objects' in the HTTP connection, which means the content of the HTTP request and response.



```
▽ HTML Form URL Encoded: application/x-www-form-urlencoded
    Form item: "i" = "McsZtRV7Bv7ZjMSzwk5aIyZEiijP8F38NJcxd5VNElaIVxctx
  ▽ /kgJOPilsUZZyvc2swCziOJC5ae17wUorRhyx48b3kIReFjFdcomTsuyE8PNXnthpE3
    T+cp2CCNstrLbeKReraPgFcgZKAlluZKoVG6SxwgKzzt0MxQFlobMu21L+mlA2DJ2pj
      Key: i
      Value [truncated]: McsZtRV7Bv7ZjMSzwk5aIyZEiijP8F38NJcxd5VNElaIV›
  ▽ Form item: "s" = ""
      Key: s
      Value:
```

Now, I have e in a folder all the files with the objects from the HTTP request:

```
$ ls main
main(1).php   main(11).php  main(13).php  main(15).php  main(3).php   main(5).php   main(7).php   main(9).php
main(10).php  main(12).php  main(14).php  main(2).php   main(4).php   main(6).php   main(8).php   main.php
```

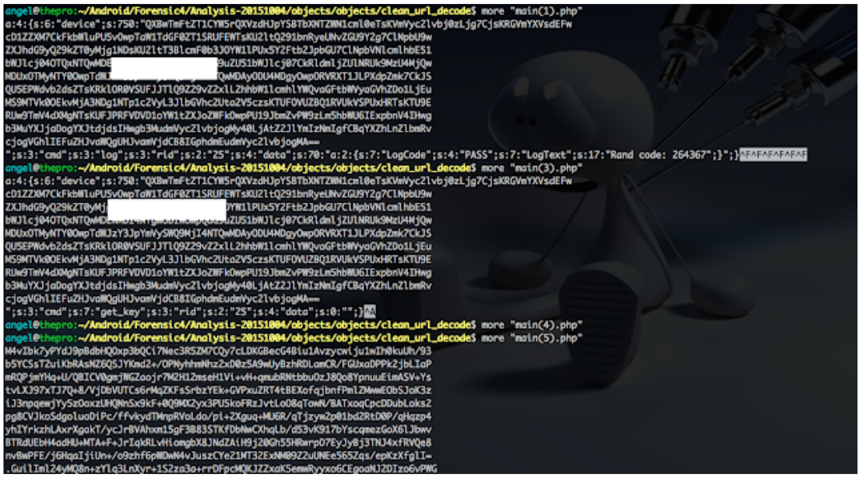
```
$ more main.php
i=McsZtRV7Bv7ZjMSzwk5aIyZEiijP8F38NJcxd5VNElaIVxctxxX9UWCGbUaOIYRxhMxTtA8nBYmT%0A%2FkgJOPilsUZZyvc2swCziOJC5ae17w
```

As the HTTP request is URL encoded, I need first to decode it, so I will adapt the python script created in this post to do it automatically. This is the script:

```
#!/usr/bin/python

from Crypto.Cipher import Blowfish
from Crypto import Random
from struct import pack
from binascii import hexlify, unhexlify
import sys
import urllib

file1 =  sys.argv[1]
file_out = sys.argv[2]

blfs_key =  open('/path/to/the/blfs.key','r')

url_encode = open(file1,'r')
url_encode_2 = url_encode.read()

url_decode = urllib.unquote(url_encode_2).decode('utf8')

file_ciphertext_base64 = url_decode
file_blfs_key = blfs_key.read()
ciphertext_raw = file_ciphertext_base64.decode("base64")

IV = "12345678"
_KEY = file_blfs_key
ciphertext = ciphertext_raw
KEY = hexlify(_KEY)[:50]
cipher = Blowfish.new(KEY, Blowfish.MODE_CBC, IV)
message = cipher.decrypt(ciphertext)
config_plain = open(file_out,'w')
config_plain.write(message)
```

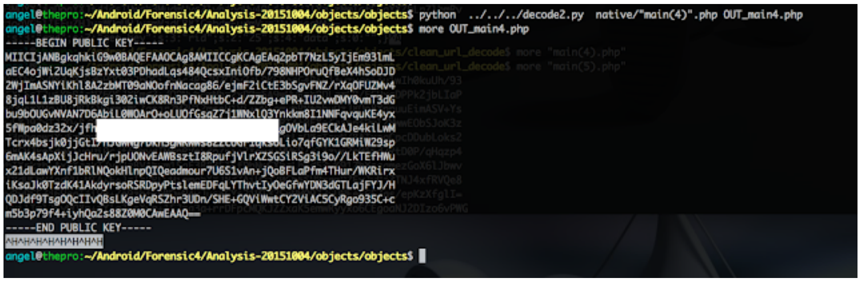
With this script it is easy to run a shell command with a loop 'for' to decrypt all the files in the directory. Bare in mind than the HTTP response are not URL encoded, so I will not need to perform that step on some of the files.

Now I should have decrypted all the information from each object. Looking at the first two HTTP POST requests I see this is the case, but for the third one, this is not the case and the data is still encrypted. What's going on here?



```
angel@thepro:~/Android/Forensic4/Analysis-20151004/objects/objects/clean_url_decode$ more "main(1).php"
a:4:{s:6:"device";s:750:"QXBwTmFtZT1CYW5rQXVzdHJpYSBTbXNTZWN1cml0eTsKVmVyc2lvbj0zLjg7CjsKRGVmYXVsdEFw
cD1ZZXM7CkFkbWluPU5vOwpTaW1TdGF0ZT1SRUFEWTsKU2ltQ291bnRyeUNvZGU9Y2g7ClNpbU9w
ZXJhdG9yQ29kZT0yMjg1NDsKU2ltT3BlcmF0b3JOYW1lPUx5Y2Ftb2JpbGU7ClNpbVNlcmlhbE51
bWJlcj04OTQxNTQwMDE[redacted]9uZUS1bWJlcj07CkRldmljZUlNRUk9MzU4MjQw
MDUxOTMyNTY0OwpTdWJ QwMDAyODU4NDgyOwpORVRXT1JLPXdpZmk7CkJS
QUSEPWdvb2dsZTsKRklORUVSUFJJTlQ9Z29vZ2xlL2hhbW1lcmhlYWQvaGFtbWVyaGVhZDo1LjEu
MS9MTVk0OEkvMjA3NDg1NTp1c2VyL3JlbGVhc2Uta2V5czsKTUFOVUZBQ1RVUkVSPUxHRTsKTU9E
RUw9TmV4dXMgNTsKUFJPRFVDVD1oYW1tZXJoZWFkOwpPU19JbmZvPW9zLm5hbWU6IExpbnV4IHwg
b3MuYXJjaDogYXJtdjdsIHwgb3MudmVyc2lvbjogMy40LjAtZ2JlYmIzNmIgfCBqYXZhLnZlbmRv
cjogVGhlIEFuZHJvaWQgUHJvamVjdCB8IGphdmEudmVyc2lvbjogMA==
";s:3:"cmd";s:3:"log";s:3:"rid";s:2:"25";s:4:"data";s:70:"a:2:{s:7:"LogCode";s:4:"PASS";s:7:"LogText";s:17:"Rand code: 264367";}";}^F^F^F^F^F^F^F
angel@thepro:~/Android/Forensic4/Analysis-20151004/objects/objects/clean_url_decode$ more "main(3).php"
a:4:{s:6:"device";s:750:"QXBwTmFtZT1CYW5rQXVzdHJpYSBTbXNTZWN1cml0eTsKVmVyc2lvbj0zLjg7CjsKRGVmYXVsdEFw
cD1ZZXM7CkFkbWluPU5vOwpTaW1TdGF0ZT1SRUFEWTsKU2ltQ291bnRyeUNvZGU9Y2g7ClNpbU9w
ZXJhdG9yQ29kZT0yMj[redacted]DYW1lPUx5Y2Ftb2JpbGU7ClNpbVNlcmlhbE51
bWJlcj04OTQxNTQwMD uZUS1bWJlcj07CkRldmljZUlNRUk9MzU4MjQw
MDUxOTMyNTY0OwpTdWJzY3JpYmVySWQ9MjI4NTQwMDAyODU4NDgyOwpORVRXT1JLPXdpZmk7CkJS
QUSEPWdvb2dsZTsKRklORUVSUFJJTlQ9Z29vZ2xlL2hhbW1lcmhlYWQvaGFtbWVyaGVhZDo1LjEu
MS9MTVk0OEkvMjA3NDg1NTp1c2VyL3JlbGVhc2Uta2V5czsKTUFOVUZBQ1RVUkVSPUxHRTsKTU9E
RUw9TmV4dXMgNTsKUFJPRFVDVD1oYW1tZXJoZWFkOwpPU19JbmZvPW9zLm5hbWU6IExpbnV4IHwg
b3MuYXJjaDogYXJtdjdsIHwgb3MudmVyc2lvbjogMy40LjAtZ2JlYmIzNmIgfCBqYXZhLnZlbmRv
cjogVGhlIEFuZHJvaWQgUHJvamVjdCB8IGphdmEudmVyc2lvbjogMA==
";s:3:"cmd";s:7:"get_key";s:3:"rid";s:2:"25";s:4:"data";s:0:"";}^A
angel@thepro:~/Android/Forensic4/Analysis-20151004/objects/objects/clean_url_decode$ more "main(4).php"
angel@thepro:~/Android/Forensic4/Analysis-20151004/objects/objects/clean_url_decode$ more "main(5).php"
M4vIbk7yPYdJ9pBdbHQ0xp3bQCi7Nec3RSZM7CQy7cLDKGBecG4Biu1AvzycwIju1wIh0kuUh/93
b5YCSsT2uiKbRAsNZ6QSJYKmd2+/OPNyhhmNhz2xD0z5A9wUyBzhRDLamCR/FGUxaDPPk2jbLIaP
mRQPjmYHq+U/QBICV0gmjWGZaojr7M2H12mseH1Vi+vH+qmubRNtbbu0zJ8Qo8YpnuuEimASV+Ys
tvLXJ97xTJ7Q+8/VjDbVUTCs6rMqZKFsSrbzYEk+GVPxuZRT4tBEXofqjbnfPmlZMwwEObSJoK3z
iJ3npqewjYy5zOoxzUHQNnSx9kF+0Q9MX2yx3PU5koFRzJvtLo08qTowN/BATxoqCpcDDubLoks2
pg8CVJkoSdgoluoDiPc/ffvkydTMnpRVoLdo/pi+2Xguq+MU6R/qTjzyw2p01bd2RtD0P/qHqzp4
yhIYrkzhLAxrXgakT/ycJrBVAhxm15gF3B83STKfDbNwCXhqLb/d53vK917bYscqmezGoX6lJbwv
BTRdUEbH4adHU+MTA+F+JrIqkRLvHiomgbX8JNdZAiH9j20Gh55HRwrpD7EyJyBj3TNJ4xfRVQe8
nvBwPFE/j6HqaIjiUn+/o9zhf6pWDwM4vJuszCYe21MT32ExNM09Z2uUNEe565Zqs/epKzXfglI=
.GuilIml24yMQ8n+zYlq3LnXyr+1S2za3a+rrDFpcMQKJZZxaK5emwRyyxo6CEgoaNJ2DIzo6vPWG
```

I am going to take a look to the HTTP response from the server, what information is being sent?



```
angel@thepro:~/Android/Forensic4/Analysis-20151004/objects/objects$ python  ../../../decode2.py  native/"main(4)".php OUT_main4.php
angel@thepro:~/Android/Forensic4/Analysis-20151004/objects/objects$ more OUT_main4.php
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAq2pbT7NzL5yIjEm93lmL
aEC4ojWi2UqKjsBzYxt03PDhadLqs484QcsxIniOfb/798NHPOruQfBeX4hSoDJD
2WjImASNYiKhl8A2zbMT09aNOofnNacag86/ejmF2iCtE3bSgvFNZ/rXqOFUZMv4
8jqL1L1zBU8jRkBkgi302iwCK8Rn3PfNxHtbC+d/ZZbg+ePR+IU2vwDMY0vmT3dG
bu9bOUGvNVAN7D6AbiL0WOArO+oLUOfGsqZ7j1WNxlQ3YnkkmBI1NNFqvquKE4yx
5fWpa0dz32x/jfh[redacted]gOVbLa9ECkAJe4kiLwM
Tcrx4bsjk0jjGtI Lio7qfGYK1GRMiW29sp
6mAK4sApXijJcHru/rjpUONvEAWBsztI8RpufjVlrXZSGSiRSg3i9o//LkTEfHWu
x21dLawYXnf1bRlNQokHlnpQIQeadmour7U6S1vAn+jQoBFLaPfm4THur/WKRirx
iKsaJk0TzdK41AkdyrsoRSRDpyPtslemEDFqLYThvtIyOeGfwYDN3dGTLajFYJ/H
QDJdf9TsgOQcIIvQBsLKgeVqRSZhr3UDn/SHE+GQViWwtCY2ViACSCyRgo935C+c
m5b3p79f4+iyhQa2s88Z0M0CAwEAAQ==
-----END PUBLIC KEY-----
^H^H^H^H^H^H^H^H^H
angel@thepro:~/Android/Forensic4/Analysis-20151004/objects/objects$ ▌
```

A Public Key!! really interesting stuff...

Actually, if I look further in the second HTTP request from the screenshot above I can see the following:

```
$ more "main(3).php"
a:4....
.....
cjogVGhlIEFuZHJvaWQgUHJvamVjdCB8IGphdmEudmVyc2lvbjogMA==

";s:3:"cmd";s:7:"get_key";s:3:"rid";s:2:"25";s:4:"data";s:0:"";}
```

This looks to me like the malware sends a request for a key and the server replies with the public key. So the only possibility is that the malware is using that key to encrypt the data so only the C&C can decrypt it with the private key.

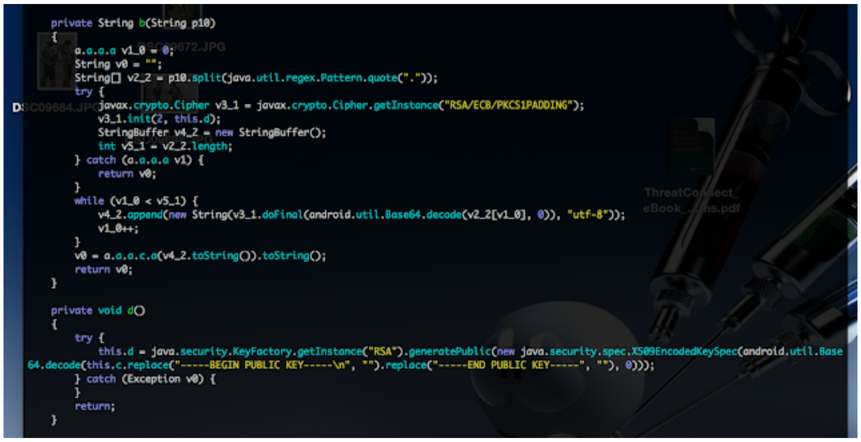
To confirm this is the case, I am going to check the source code of the malware with 'androguard' as I explained in previous post.

Looking at the code, I see there is a method with the string 'get_key' and I can see which other method is calling it:

```
In [10]: d.CLASS_Lorg_thoughtcrime_securesms_h_c.METHOD_c.pretty_show()
########## Method Information
Lorg/thoughtcrime/securesms/h/c;->c()V [access_flags=public]
########## Params
local registers: v0...v2
- return: void
####################
********************************************************************************
c-BB@0x0 :
0  (00000000) const-string        v0, 'get_key'
1  (00000004) const-string        v1, ''
2  (00000008) invoke-virtual      v2, v0, v1, Lorg/thoughtcrime/securesms/h/c;->a(Ljava/lang/String; Ljava/lang/String;)Ljava/lang/String;
3  (0000000e) move-result-object  v0
4  (00000010) iput-object         v0, v2, Lorg/thoughtcrime/securesms/h/c;->c Ljava/lang/String;
5  (00000014) invoke-virtual      v2, Lorg/thoughtcrime/securesms/h/c;->b()Ljava/lang/Boolean;
6  (0000001a) move-result-object  v0
7  (0000001c) invoke-virtual      v0, Ljava/lang/Boolean;->booleanValue()Z
```

```
8  (00000022) move-result         v0
9  (00000024) if-eqz              v0, 5 [ c-BB@0x28 c-BB@0x2e ]

c-BB@0x28 :
10 (00000028) invoke-direct       v2, Lorg/thoughtcrime/securesms/h/c;->d()V [ c-BB@0x2e ]

c-BB@0x2e :
11 (0000002e) return-void

********************************************************************************
########## XREF
F: Lorg/thoughtcrime/securesms/h/i; b (Landroid/content/Context;)V be
T: Lorg/thoughtcrime/securesms/h/c; b ()Ljava/lang/Boolean; 14
T: Lorg/thoughtcrime/securesms/h/c; d ()V 28
T: Lorg/thoughtcrime/securesms/h/c; a (Ljava/lang/String; Ljava/lang/String;)Ljava/lang/String; 8

####################
```

When decompiling the code I end up with some interesting Java methods:



```
    private String b(String p10)
    {
        a.a.a.a v1_0 = 0;
        String v0 = "";
        String[] v2_2 = p10.split(java.util.regex.Pattern.quote("."));
        try {
            javax.crypto.Cipher v3_1 = javax.crypto.Cipher.getInstance("RSA/ECB/PKCS1PADDING");
            v3_1.init(2, this.d);
            StringBuffer v4_2 = new StringBuffer();
            int v5_1 = v2_2.length;
        } catch (a.a.a.a v1) {
            return v0;
        }
        while (v1_0 < v5_1) {
            v4_2.append(new String(v3_1.doFinal(android.util.Base64.decode(v2_2[v1_0], 0)), "utf-8"));
            v1_0++;
        }
        v0 = a.a.a.c.a(v4_2.toString()).toString();
        return v0;
    }

    private void d()
    {
        try {
            this.d = java.security.KeyFactory.getInstance("RSA").generatePublic(new java.security.spec.X509EncodedKeySpec(android.util.Base
64.decode(this.c.replace("-----BEGIN PUBLIC KEY-----\n", "").replace("-----END PUBLIC KEY-----", ""), 0)));
        } catch (Exception v0) {
        }
        return;
    }
```

Looking tat the Java code I can see that the public key is used. But also, looking deeper into the code, I find another interesting method:

```
private String a(String p9)
  {
      String v1_0 = 0;
      String v0_0 = "";
      try {
          javax.crypto.Cipher v2_1 = javax.crypto.Cipher.getInstance("RSA/ECB/PKCS1PADDING");
          v2_1.init(1, this.d);
          String[] v3_2 = this.a(p9, 100);
          java.util.ArrayList v4_2 = new java.util.ArrayList();
          int v5 = v3_2.length;
      } catch (String v1) {
          return this.a.c(v0_0);
      }
      while (v1_0 < v5) {
          v4_2.add(android.util.Base64.encodeToString(v2_1.doFinal(v3_2[v1_0].getBytes()), 0));
          v1_0++;
      }
      v0_0 = android.text.TextUtils.join(".", v4_2);
      return this.a.c(v0_0);

  }
```

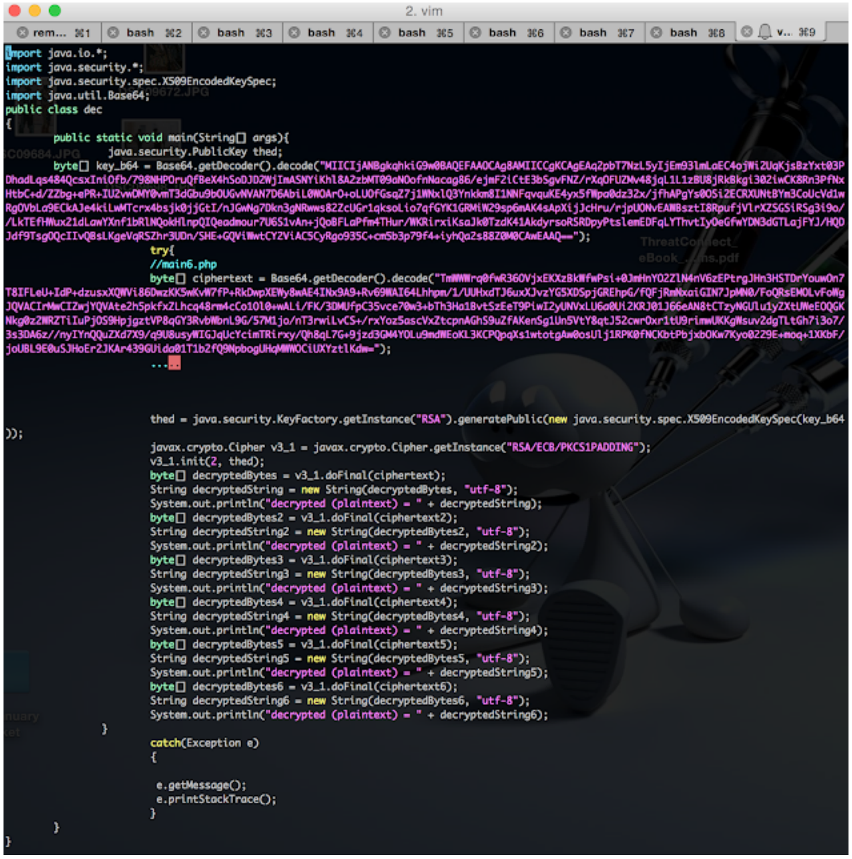
So basically, one method is for encryption and the other for decryption, and both of them are using the same public key. This is really interesting stuff.

So this is whats going on so far:

1. The compromised device sends the information encrypted with blowfish to the C&C

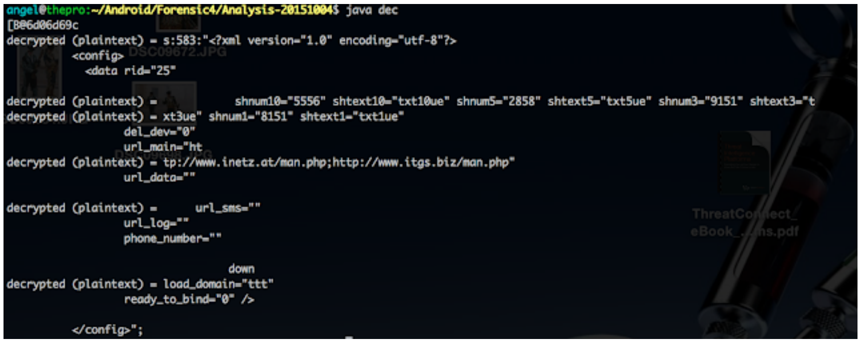
2. The C&C server replies with OK
3. The compromised device requests the public key
4. The C&C server replies with the public key
5. The compromised device encrypts the information with the public key and sends to the C&C
6. The C&C server can decrypt with it's private key
7. The C&C server sends data encrypted with the private key ->I need to verify this
8. The compromised device can decrypt with the public key > I need to verify this

To verify step 6 and 7, and as very quick PoC, I have created some Java code which takes the public key sent by the C&C and try to decrypt the successive messages sent by the C&C.



```
import java.io.*;
import java.security.*;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
public class dec
{
        public static void main(String[] args){
                java.security.PublicKey thed;
        byte[] key_b64 = Base64.getDecoder().decode("MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAq2pbT7NzL5yIjEm93lmLaEC4ojWi2UqKjsBzYxt03P
DhadLqs484QcsxIniOfb/798NHPOruQfBeX4hSoDJD2WjImASNYiKhl8A2zbMT09aNOofnNacag86/ejmF2iCtE3bSgvFNZ/rXqOFUZMv48jqL1L1zBU8jRkBkgi302iwCK8Rn3PfNx
HtbC+d/ZZbg+ePR+IU2vwDMY0vmT3dGbu9bOUGvNVAN7D6AbiL0WOArO+oLUOfGsqZ7j1WNxlQ3Ynkkm8I1NNFqvquKE4yx5fWpa0dz32x/jfhAPgYs0OSi2ECRXUNtBYm3CoUcVd1w
RgOVbLa9ECkAJe4kiLwMTcrx4bsjk0jjGtI/nJGwNg7Dkn3gNRwws82ZcUGr1qksoLio7qfGYK1GRMiW29sp6mAK4sApXijJcHru/rjpUONvEAMBsztI8RpufjVlrXZSGSiRSg3i9o/
/LkTEfHMux21dLawYXnf1bRlNQokHlnpQIQeadmour7U6S1vAn+jQoBFLaPfm4THur/WKRirxiKsaJk0TzdK41AkdyrsoRSRDpyPtslemEDFqLYThvtIyOeGfwYDN3dGTLajFYJ/HQD
Jdf9TsgOQcIIvQ8sLKgeVqRSZhr3UDn/SHE+GQViWwtCY2ViAC5CyRgo935C+cm5b3p79f4+iyhQa2s88Z0M0CAwEAAQ==");
                        try{
                        //main6.php
                        byte[] ciphertext = Base64.getDecoder().decode("TmWWWrq0fwR36OVjxEKXzBkWfwPsi+0JmHnYO2ZlN4nV6zEPtrgJHn3HSTDrYouwOn7
T8IFLeU+IdP+dzusxXQWVi86DwzKK5wKvW7fP+RkDwpXEWy8wAE4INx9A9+Rv69WAI64Lhhpm/1/UUHxdTJ6uxXJvzYG5XDSpjGREhpG/FQFjRmNxaiGIN7JpMN0/FoQRsEMOLvFoWg
JQVACIrMwCIZwjYQVAte2h5pkfxZLhcq48rm4cCo10l0+wALi/FK/3DMUfpC35vce70w3+bTh3Ha1BvtSzEeT9PiwI2yUNVxLU6a0Ui2KRJ01J66eAN8tCTzyNGUlu1yZXtUWeEOQGK
Nkg0z2WRZTiIuPjOS9HpjgztVP8qGY3RvbWbnL9G/57M1jo/nT3rwiLvCS+/rxYoz5ascVxZtcpnAGhS9uZfAKenSg1Un5VtY8qtJ52cwrOxr1tU9rimwUKKgWsuv2dgTLtGh7i3o7/
3s3DA6z//nyIYnQQuZXd7X9/q9U8usyWIGJqUcYcimTRirxy/Qh8qL7G+9jzd3GM4YOLu9mdWEoKL3KCPQpqXs1wtotgAw0osUlj1RPK0fNCKbtPbjxbOKw7Kyo0229E+moq+1XKbF/
joUBL9E0uSJHoEr2JKAr439GUida01T1b2fQ9NpbogUHqMWWOCiUXYztlKdw=");
                        ...
                        

                        thed = java.security.KeyFactory.getInstance("RSA").generatePublic(new java.security.spec.X509EncodedKeySpec(key_b64
));

                        javax.crypto.Cipher v3_1 = javax.crypto.Cipher.getInstance("RSA/ECB/PKCS1PADDING");
                        v3_1.init(2, thed);
                        byte[] decryptedBytes = v3_1.doFinal(ciphertext);
                        String decryptedString = new String(decryptedBytes, "utf-8");
                        System.out.println("decrypted (plaintext) = " + decryptedString);
                        byte[] decryptedBytes2 = v3_1.doFinal(ciphertext2);
                        String decryptedString2 = new String(decryptedBytes2, "utf-8");
                        System.out.println("decrypted (plaintext) = " + decryptedString2);
                        byte[] decryptedBytes3 = v3_1.doFinal(ciphertext3);
                        String decryptedString3 = new String(decryptedBytes3, "utf-8");
                        System.out.println("decrypted (plaintext) = " + decryptedString3);
                        byte[] decryptedBytes4 = v3_1.doFinal(ciphertext4);
                        String decryptedString4 = new String(decryptedBytes4, "utf-8");
                        System.out.println("decrypted (plaintext) = " + decryptedString4);
                        byte[] decryptedBytes5 = v3_1.doFinal(ciphertext5);
                        String decryptedString5 = new String(decryptedBytes5, "utf-8");
                        System.out.println("decrypted (plaintext) = " + decryptedString5);
                        byte[] decryptedBytes6 = v3_1.doFinal(ciphertext6);
                        String decryptedString6 = new String(decryptedBytes6, "utf-8");
                        System.out.println("decrypted (plaintext) = " + decryptedString6);
                }

                        catch(Exception e)
                        {

                         e.getMessage();
                         e.printStackTrace();
                        }
        }
}
```

Bingo! When I run the code I clearly see it works and my 'guess' was right:



```
angel@thepro:~/Android/Forensic4/Analysis-20151004$ java dec
[B@6d06d69c
decrypted (plaintext) = s:583:"<?xml version="1.0" encoding="utf-8"?>
            <config>
              <data rid="25"

decrypted (plaintext) =             shnum10="5556" shtext10="txt10ue" shnum5="2858" shtext5="txt5ue" shnum3="9151" shtext3="t
decrypted (plaintext) = xt3ue" shnum1="8151" shtext1="txt1ue"
                   del_dev="0"
                   url_main="ht
decrypted (plaintext) = tp://www.inetz.at/man.php;http://www.itgs.biz/man.php"
                   url_data=""

decrypted (plaintext) =      url_sms=""
                   url_log=""
                   phone_number=""

                                 down
decrypted (plaintext) = load_domain="ttt"
                   ready_to_bind="0" />

            </config>";
```

What is the information sent by the C&C? it looks like a new config.xml with new C&C URL..
Very interesting..

Looking to the code again, I see methods which performs the request for a new configuration file:

```
In [7]: d.CLASS_Lorg_thoughtcrime_securesms_xservices_b.source()
package org.thoughtcrime.securesms.xservices;
 class b extends android.os.AsyncTask {
    android.content.Context a;
    final synthetic org.thoughtcrime.securesms.xservices.XRepeat b;
    public b(org.thoughtcrime.securesms.xservices.XRepeat p1, android.content.Context p2)
    {
        this.b = p1;
        this.a = p2;
        return;
    }

    protected varargs String a(String[] p4)
    {
        org.thoughtcrime.securesms.h.i.a(this.a);
        org.thoughtcrime.securesms.h.i.c("CONF", "Check pull off urls", this.a);
        org.thoughtcrime.securesms.h.i.b(this.a);
        org.thoughtcrime.securesms.h.i.c(this.a);
        org.thoughtcrime.securesms.h.i.c("CONF", "Get config data from server", this.a);
        org.thoughtcrime.securesms.h.i.j(this.a);
        org.thoughtcrime.securesms.h.i.c("DATA", "Send data to server", this.a);
        return "OK";
    }

    protected void a(String p1)
    {
        super.onPostExecute(p1);
        return;
    }

    protected synthetic Object doInBackground(Object[] p2)
    {
        return this.a(((String[]) p2));
    }

    protected synthetic void onPostExecute(Object p1)
    {
        this.a(((String) p1));
        return;
    }

}
```

As the HTTP request to the C&C are encrypted with the Public key, I can't decrypt it. However, I could check in memory the information before is encrypted.

And this is what I found:

```
a:2:{s:7:"LogCode";s:4:"CONF";s:7:"LogText";s:27:"Get config data from server";}
```

Which matches the methods I checked previously :)