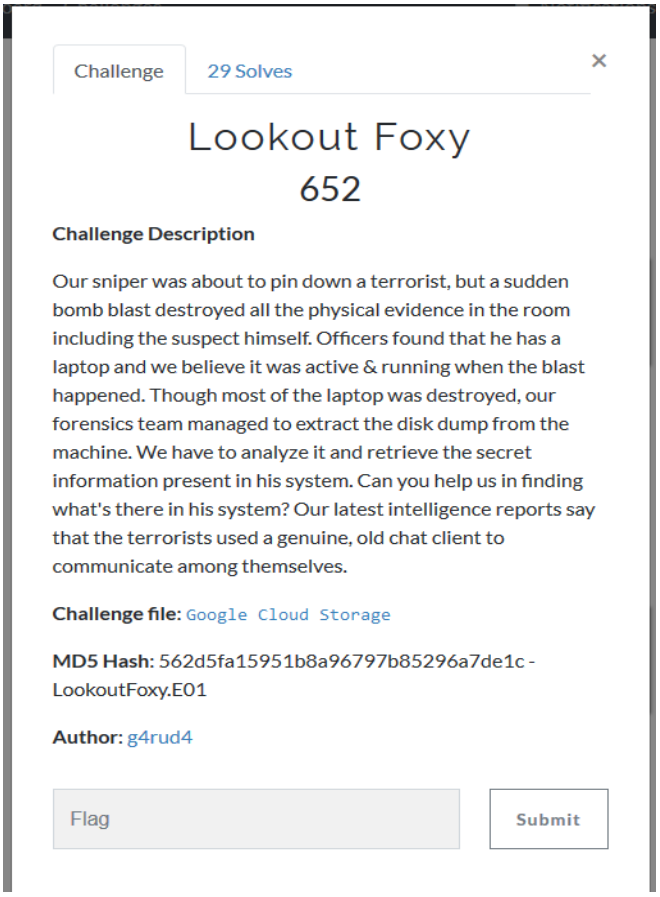
InCTF

Category - Forensics

Challenge - Lookout Foxy

File – LookoutFoxy.EO1 (1.4ghz)

One of four really good Forensic challenges, this one was solved as a Team effort by **Tri{Hacking}**, with all contributing and learning as we progressed through.
.


Challenge 29 Solves

Lookout Foxy
652

Challenge Description

Our sniper was about to pin down a terrorist, but a sudden bomb blast destroyed all the physical evidence in the room including the suspect himself. Officers found that he has a laptop and we believe it was active & running when the blast happened. Though most of the laptop was destroyed, our forensics team managed to extract the disk dump from the machine. We have to analyze it and retrieve the secret information present in his system. Can you help us in finding what's there in his system? Our latest intelligence reports say that the terrorists used a genuine, old chat client to communicate among themselves.

Challenge file: Google Cloud Storage

MD5 Hash: 562d5fa15951b8a96797b85296a7de1c - LookoutFoxy.E01

Author: g4rud4

Flag Submit

A quick **file LookoutFoxy** confirms that it is a EWF/Expert Witness/EnCase image, there are a few ways we could open this, you can mount it directly in linux or go the way we did and use Autopsy, if you don't have Autopsy or need a refresh on how to load the E01 then the official user manual is pretty good.

Once loaded in, we started to have a root around the file system, the clue seems to indicate an old `chat client` may have been used to communicate with each other. At this point **bantahacka** found some interesting files under ***Temporary Internet Files/Content.IE5*** where the user ***CRIMSON*** had a file secret.gpg

r / r	detail[2]	2020-03-18 06:10:01 (EDT)	2020-03-18 06:10:01 (EDT)	2020-03-18 06:10:01 (EDT)	2020-03-18 06:10:01 (EDT)	1378	0
r / r	prev[1]	2020-07-27 10:02:50 (EDT)	2020-07-27 10:02:50 (EDT)	2020-07-27 10:02:50 (EDT)	2020-07-27 10:02:50 (EDT)	53	0
r / r	protocol-core.0b1417ef97fa[1].css	2020-03-18 01:04:13 (EDT)	2020-03-18 01:04:13 (EDT)	2020-03-18 01:04:13 (EDT)	2020-03-18 01:04:13 (EDT)	55856	0
r / r	secret.gpg	2020-07-27 10:09:14 (EDT)	2020-07-27 10:09:14 (EDT)	2020-07-27 10:09:14 (EDT)	2020-07-27 10:09:14 (EDT)	2901	0

A quick check on gpg (GNU Privacy Guard) shows they are encrypted files using RFC4880. Running **File** on it gives us **secret.gpg: PGP RSA encrypted session key - keyid: D2F6493F C12CF3A1 RSA (Encrypt or Sign) 3072b**

This (and a google search) shows we need to use the owners key to decrypt it. Another `keyword` search on Autopsy finds this

	Internet Files/Content.IE5/U0PDHEH3/secret.gpg	10:09:14
r / r	C:/Documents and Settings/crimson/Local Settings/Temporary Internet Files/Content.IE5/U0PDHEH3/secret.gpg:Zone.Identifier	2020-07 10:09:14
r / r	C:/Program Files/GPG/secret.key	2020-03 07:52:54

Armed we both of these, we go about decrypting the file:

Firstly load the key

```
clueless@kali:~/Documents/inctf$ gpg --import secret.key
gpg: key 35E453B7B6FB578A: public key "Danial Banjamin <danial.benjamin008@gmail.com>" imported
gpg: key 35E453B7B6FB578A: secret key imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg:       secret keys read: 1
gpg:   secret keys imported: 1
```

Secondly, decrypt the file

```
clueless@kali:~/Documents/inctf$ gpg --output decrypt.txt --decrypt secret.gpg
gpg: encrypted with 3072-bit RSA key, ID 21EE2AF2B818EB2B, created 2020-03-21
      "Danial Banjamin <danial.benjamin008@gmail.com>"
clueless@kali:~/Documents/inctf$
```

Opening up the decrypted file, we have a fair amount of text some in English and some in Latin – half way down we find what we are looking for, part one of the Flag.

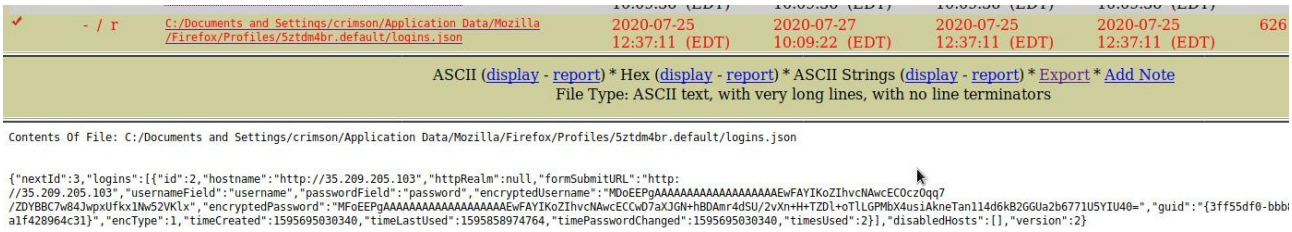
```
Here is the first part:

Iaculis at erat pellentesque adipiscing commodo elit at. Pulvinar elementum integer enim neque. Diam solli
 feugiat sed lectus vestibulum mattis. Felis eget nunc lobortis mattis aliquam faucibus purus. Suscipit ad
teger quis auctor. Etiam tempor orci eu lobortis elementum. Suspendisse sed nisi lacus sed viverra. Sceler
 pellentesque habitant morbi tristique. Sit amet consectetur adipiscing elit pellentesque habitant morbi t

Pharetra et ultrices neque ornare aenean. Faucibus a pellentesque sit amet porttitor eget dolor morbi non.
erisque eleifend donec pretium vulputate sapien. Elit duis tristique sollicitudin nibh sit amet commodo. P
. Et ultrices neque ornare aenean euismod. Viverra justo nec ultrices dui sapien eget. Sed cras ornare arc
neque convallis a cras semper auctor. Id nibh tortor id aliquet lectus proin. Elit sed vulputate mi sit am

Important string: inctf{!_h0p3_y0u_L1k3d_s0lv1ng_7h3_F1rs7_p4r7_

Aenean euismod elementum nisi quis eleifend. Vitae elementum curabitur vitae nunc. Tincidunt ornare massa
ementum tempus egestas sed sed risus. Habitasse platea dictumst vestibulum rhoncus est pellentesque elit u
 ipsum consequat nisl vel pretium lectus quam id. Euismod elementum nisi quis eleifend quam adipiscing. Vo
nare arcu. Augue eget arcu dictum varius duis at. Leo vel fringilla est ullamcorper eget nulla. Enim nulla
ltrices tincidunt arcu non sodales neque. Arcu dui vivamus arcu felis bibendum ut. In nulla posuere sollic
rci. Eget aliquet nibh praesent tristique magna. Id interdum velit laoreet id donec ultrices. Nisl pretium
viverra suspendisse. Aliquam ut porttitor leo a. Euismod nisi porta lorem mollis aliquam ut. Aliquam sem e
enenatis.
```

After translating the `latin` it does not throw up any more clues (or none that triggered with us). It does give us a **gmail** address, and some more routing round the file system **bantahaka** comes up trumps again with finding some email files under Outlook Express – this however turns out to be another route to Part 1 of the Flag (details below).

Back to digging around in **Autopsy** and yep you guessed it **bantahaka** has found some more interesting stuff. Within the Firefox History there is some interesting searches regarding how passwords are stored in a browser and if saved passwords can be decrypted, a quick search on google (result here) gives us:

"The passwords stored in **logins.json** are encrypted, but the encryption key is stored in key4.db (previously in key3.db) and without a master password you merely need to place the two files in Firefox profile to see the passwords in the Password Manger"

Another quick search in **Autopsy** for **logins.json** find the file, and we can see:



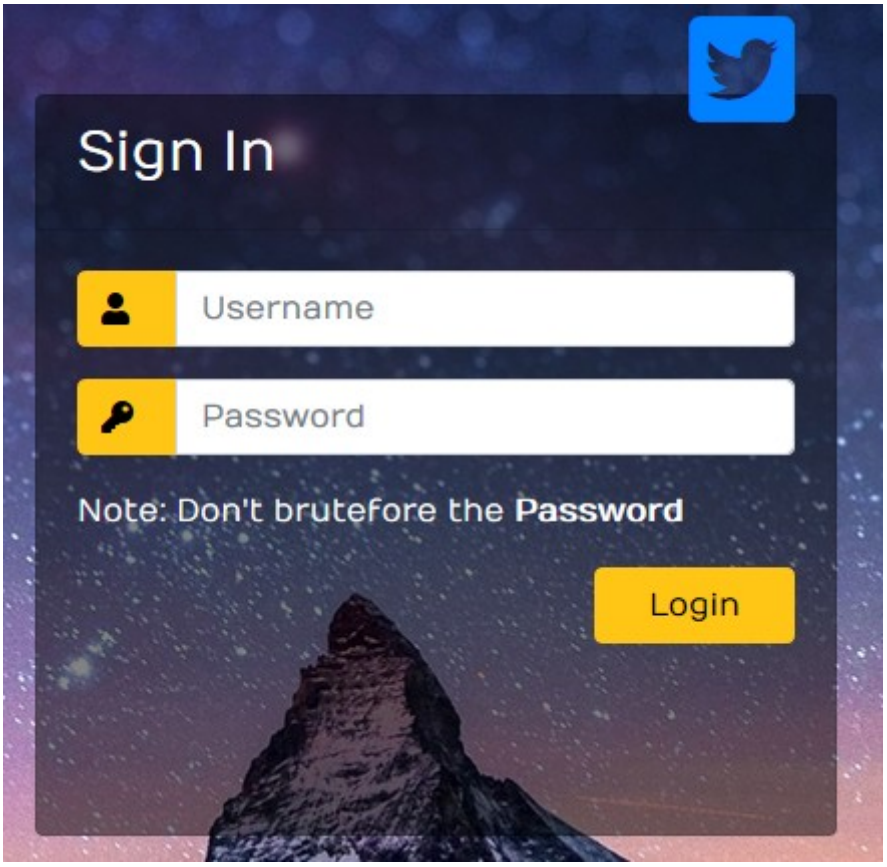
At this point another team member **Stealthsploit** chucks us the tool **firepwd**, this again pointed to Firefox >= 32 (needing logins.json and key3.db (we already had logins.json) – another search in **Autopsy** finds the file. Armed with both of these we can quickly crack the password.

By default, firepwd.py processes key3b.db and logins.json in the current directory, so we put them all in one place and run:

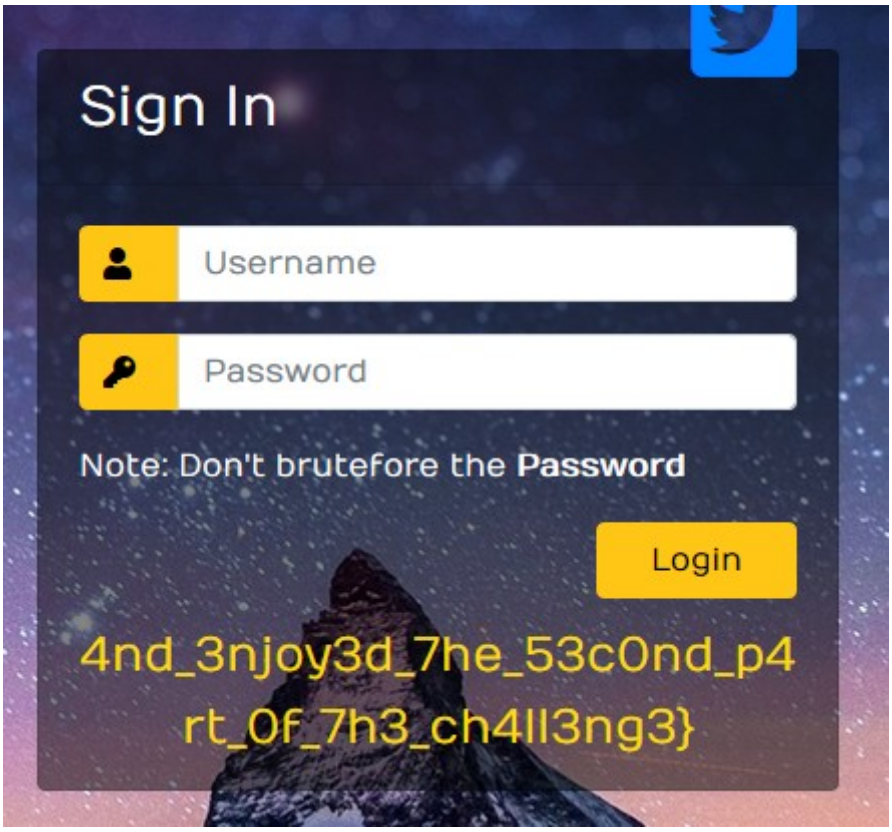
```
clueless@kali:~/Documents/inctf/firepwd$ python3 firepwd.py
 SEQUENCE {
   SEQUENCE {
     OBJECTIDENTIFIER 1.2.840.113549.1.12.5.1.3 pbeWithSha1AndTripleDES-CBC
     SEQUENCE {
       OCTETSTRING b'4507526ee8d32883fd34e3f0113ee54a16c21bb7'
       INTEGER b'01'
     }
   }
   OCTETSTRING b'019ed1d283e95703c15e3b8e6084f4f63e93b9b11ebee2c1b2f7c94750fc91fe6c266a3e50d8a173b63950e1f3069f2cf6ccee31e9ba483820a5ec27fc
f57112a689f99505cdbb35bafccf60c8f96508de0670319a6c7ca10a8db3fecc8d984'
 }
decrypting privKeyData
 SEQUENCE {
   INTEGER b'00'
   SEQUENCE {
     OBJECTIDENTIFIER 1.2.840.113549.1.1.1 pkcs-1
     NULL 0
   }
   OCTETSTRING b'3043020100021100f800000000000000000000000000000102010002190097983791895 87cf73261e9853e234534b9ba2f254ab06894020100020100020
100020100020115'
 }
decoding b'3043020100021100f800000000000000000000000000000102010002190097983791895 87cf73261e9853e234534b9ba2f254ab06894020100020100020100020
100020115'
 SEQUENCE {
   INTEGER b'00'
   INTEGER b'00f800000000000000000000000000001'
   INTEGER b'00'
   INTEGER b'009798379189587cf73261e9853e234534b9ba2f254ab06894'
   INTEGER b'00'
   INTEGER b'00'
   INTEGER b'00'
   INTEGER b'00'
   INTEGER b'15'
 }
decrypting login/password pairs
http://35.209.205.103:b'Danial_Banjamin',b'2!6BQ&e626g#YNWxsQWV9^knO8#85*E%6Zaxr@At42'
```

This has now given us a **website**, user name and password.

The website presents us with a login box:



Inputting the username and password we got earlier, give us:



Giving us the whole flag

inctf{!_h0p3_y0u_L1k3d_s0lv1ng_7h3_F1rs7_p4r7_4nd_3njoy3d_7he_53c0nd_p4rt_0f_7h3_ch4ll3ng3}

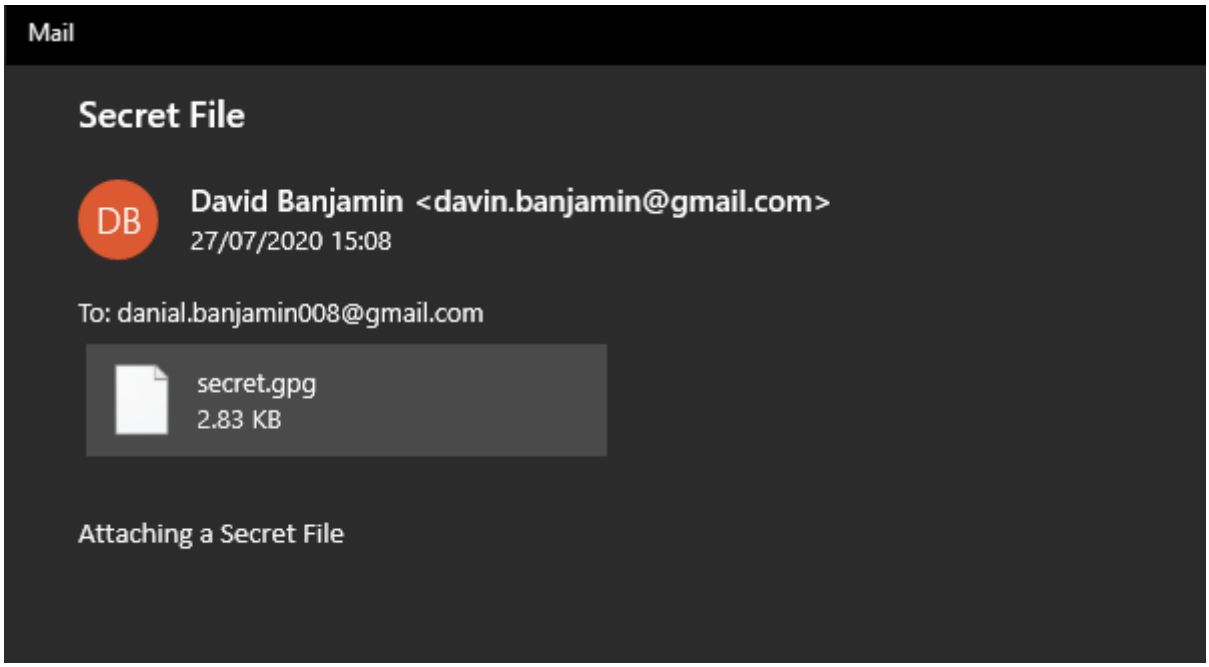
We managed to finish this we about 15 mins to spare.

Another partial route to Part 1 (and probably more in keeping with the clue about communication).

We found some .dbx files under user **CRIMSON** for outlook express, using **undbx** we restore the following .dbs files – inbox, deleted, sent, folders, offline and draft

```
C:\Windows\System32\cmd.exe
UnDBX v0.21 (Nov 19 2013)
Extracting 0 messages from Deleteditems.dbx to C:\Users\USER\Desktop/Deleteditems:
0 messages saved, 0 skipped, 0 errors, 0 files moved
Extracting 0 messages from Drafts.dbx to C:\Users\USER\Desktop/Drafts:
0 messages saved, 0 skipped, 0 errors, 0 files moved
DBX file Folders.dbx does not contain messages
Extracting 1 messages from Inbox.dbx to C:\Users\USER\Desktop/Inbox:
0 messages saved, 1 skipped, 0 errors, 0 files moved
DBX file offline.dbx does not contain messages
Extracting 0 messages from Outbox.dbx to C:\Users\USER\Desktop/Outbox:
0 messages saved, 0 skipped, 0 errors, 0 files moved
DBX file Pop3uidl.dbx does not contain messages
Extracting 0 messages from SentItems.dbx to C:\Users\USER\Desktop/SentItems:
0 messages saved, 0 skipped, 0 errors, 0 files moved
Extracted 5 out of 8 DBX files
Press any key to continue . . .
```

As you can see, there is 1 email in the Inbox – opening that up gives us the secret.gpg (which we found earlier)



You can then follow the same route above to get the secret.key and decrypt the file.

Thanks to **InCTF** and **g4rud4** for a really interesting challenge.