

[Sign up for a GitHub account](#)[Sign in](#)

Instantly share code, notes, and snippets.

[Create a gist now](#) **Lense** / [randomware.md](#)
Last active 3 months ago

SECCON 2016 quals: randomware

 [randomware.md](#)

randomware

SECCON 2016 quals

description

300 points

My PC suddenly got broken. Could you help me to recover it please?

NOTE: The disk can be virus-infected. DO NOT RUN any programs extracted from the disk outside of sandbox.

[disk.qcow2.zip](#)

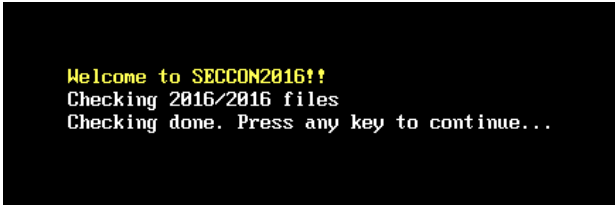
Challenge files is huge, please download it first. Password will release after 60min.

password: h9nn4c2955kik9qti9xphuxti

Solution

Boot

First thing I did was boot up the image. I used vmware with `qemu-img convert -f qcow2 -O vmdk disk.qcow2`. It's some custom bootloader that starts with



```
Welcome to SECCON2016!!
Checking 2016/2016 files
Checking done. Press any key to continue...
```

and then shows a rainbow:



```
Can you find the flag? :)
```

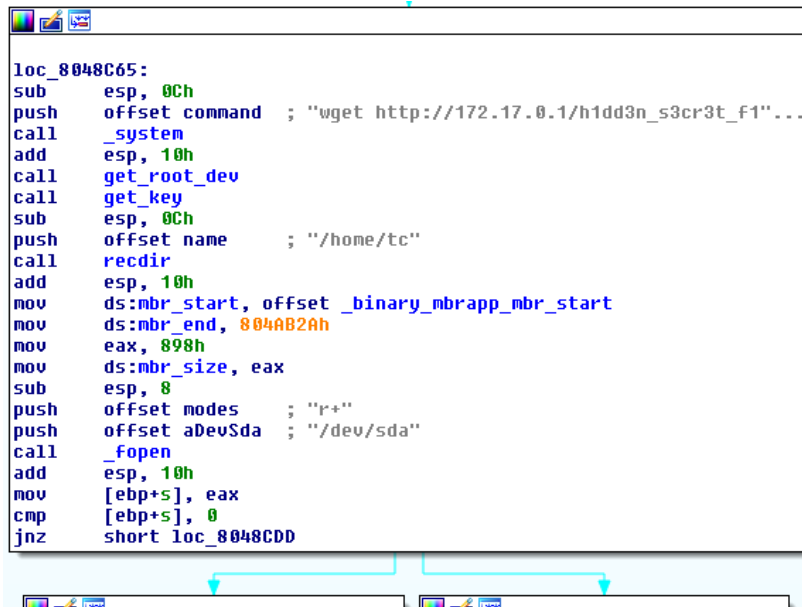
Extract

I converted qcow image to a raw disk image: `qemu-img convert -f qcow2 -O raw disk.qcow2 disk.raw`. Disk tools didn't work since the MBR was overwritten, so I ended up using binwalk to extract the files. First interesting thing was `23D70D`, which is an ELF. Looking at strings suggest that it's a Linux kernel, specifically: `Linux version 4.2.9-tinycore (tc@box) (gcc version 5.2.0 (GCC)) #1999 SMP Mon Jan 18 19:42:12 UTC 2016`. There also looks like a dump of the fs, split across multiple squashfs directories, but there wasn't anything all that interesting in there (except that firefox-esr is installed). I ended up doing `grep SECCON . -r` which has exactly one result: `512CE00`, a tar. Extracting that gives `opt/` and the very interesting `home/` directory.

Reverse

in `home/tc/` there is an ELF `getflag` and an encrypted `h1dd3n_s3cr3t_f14g.jpg`. I resisted the temptation to run `getflag` on my host and instead dropped it into IDA Pro. It's small and not stripped with only 5 non-library functions, and only 3 functions that really matter: `main`, `readdir`, and `encrypt`.

`main`



```

loc_8048C65:
sub     esp, 0Ch
push    offset command ; "wget http://172.17.0.1/h1dd3n_s3cr3t_f1"...
call    _system
add     esp, 10h
call    get_root_dev
call    get_key
sub     esp, 0Ch
push    offset name ; "/home/tc"
call    recdir
add     esp, 10h
mov     ds:mbr_start, offset _binary_mbrapp_mbr_start
mov     ds:mbr_end, 804AB2Ah
mov     eax, 898h
mov     ds:mbr_size, eax
sub     esp, 8
push    offset modes ; "r+"
push    offset aDevSda ; "/dev/sda"
call    _fopen
add     esp, 10h
mov     [ebp+5], eax
cmp     [ebp+5], 0
jnz     short loc_8048CDD

```

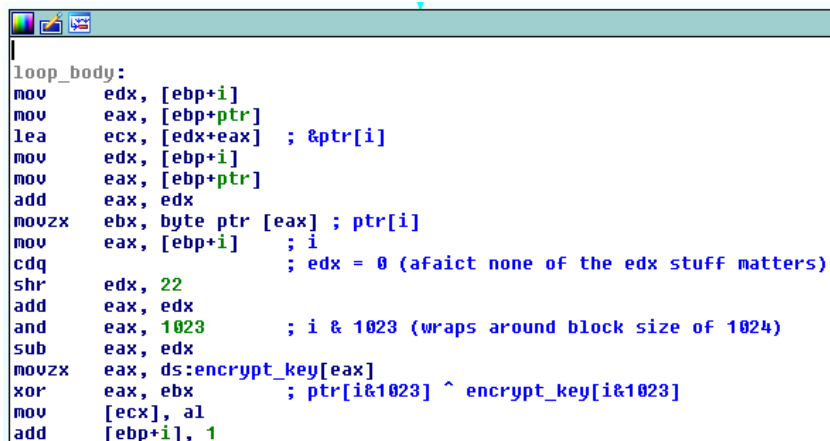
1. Downloads flag from a local server
2. Calls `get_root_dev` (just `stat("/home/", root_dev)` for some reason)
3. Calls `get_key` (sets `encrypt_key` to 1024 bytes from `/dev/urandom`)
4. Calls `recdir("/home/tc")`
5. Writes bootloader from memory to `/dev/sda`

`recdir`

Walks the filesystem from `/home/` and calls `encrypt` on files that have an extension in the whitelist which is pdf, xml, bin, txt, or various image and office types.

`encrypt`

Encrypts files with repeated xor key (length 1024). Encryption loop:



```

loop_body:
mov     edx, [ebp+i]
mov     eax, [ebp+ptr]
lea     ecx, [edx+eax] ; &ptr[i]
mov     edx, [ebp+i]
mov     eax, [ebp+ptr]
add     eax, edx
movzx   ebx, byte ptr [eax] ; ptr[i]
mov     eax, [ebp+i] ; i
cdq     ; edx = 0 (afaict none of the edx stuff matters)
shr     edx, 22
add     eax, edx
and     eax, 1023 ; i & 1023 (wraps around block size of 1024)
sub     eax, edx
movzx   eax, ds:encrypt_key[eax]
xor     eax, ebx ; ptr[i&1023] ^ encrypt_key[i&1023]
mov     [ecx], al
add     [ebp+i], 1

```

Decrypt

(Brief description of xor: $1^0=1$, $0^1=1$, $0^0=0$, $1^1=0$. $\text{ciphertext} = \text{key} \oplus \text{plaintext}$ but also $\text{key} = \text{plaintext} \oplus \text{ciphertext}$)

So, xor block ciphers are vulnerable to known-plaintext attacks, and we just have to find some plaintext. We know the header of a JPEG, so we can get the first few bytes of the key, but that's not enough.

mime-types

Fortunately, there are also hidden files in the home directory. Searching for whitelisted file extensions gives some mime-type XML files, and `blocklist.xml` and `revocations.txt` in a Firefox profile. The mime-type files were all < 1024 bytes, with the longest, `.local/share/mime/packages/user-extension-xhtml.xml`, being 254 bytes. I happen to have the same file on my computer with a matching size, so by xoring the two files I got the first 254 bytes of the key (since $\text{ciphertext} \oplus \text{plaintext} = \text{key}$). Around 1/4 of the key still isn't enough to view the image, though, so I moved on to the Firefox files.

```
<?xml version="1.0" encoding="UTF-8"?>
<mime-info xmlns="http://www.freedesktop.org/standards/shared-mime-info">
  <mime-type type="application/x-extension-xhtml">
    <comment>xhtml document</comment>
    <glob pattern="*.xhtml"/>
  </mime-type>
</mime-info>
```

blocklist.xml

After some research I discovered that `blocklist.xml` is updated once a day. The one in the challenge was last modified Nov 28, and mine was last modified Dec 10 with a significantly different size. From `about:config` I found `https://blocklist.addons.mozilla.org/blocklist/3/%APP_ID%/%APP_VERSION%/%PRODUCT%/%BUILD_ID%/%BUILD_TARGET%/%LOCALE%/%CHANNEL%/%OS_VERSION%/%DISTRIBUTION%/%DISTRIBUTION_VERSION%/%PING_COUNT%/%TOTAL_PING_COUNT%/%DAYS_SINCE_LAST_PING%/,` but I don't know valid arguments, and it looks like there's no way to request historical versions of the file. I spent a fair amount of time on that before giving up and going the more effort way: since XML has a standard format, you can guess a few characters after what's decrypted, use that to get a few more bytes of the key, and repeat for any index multiple of 1024 in the file.

Example

We have a plaintext mime-type file and a ciphertext mime-type file which we xor to get the first 254 bytes of the key. xoring the partial key with the start of the encrypted `blocklist.xml` gives:

```
<?xml version="1.0"?>
<blocklist xmlns="http://www.mozilla.org/2006/addons-blocklist" lastupdate="1456184414000">
  <emItems>
    <emItem blockID="i58" id="webmaster@buzzzzvideos.info">
      <versionRange minVersion="0" maxVersion=
```

We assume that the value of `maxVersion` is a string (to match `minVersion` and because it's that way in the newest version of the file), so we now have 255 plaintext bytes which we can xor with the ciphertext and get 255 bytes of the key. Next, we shift to index 1024 in the ciphertext (since the xor cipher has a block size of 1024) and xor 255 bytes with the key to get more plaintext:

```
      <versionRange minVersion="0" maxVersion="*" severity="1">
    </versionRange>
    <prefs>
  </prefs>
</emItem>
<emItem blockID="i105" id="{95ff02bc-ffc6-45f0-a5c8-619b8226a9de}">
```

(there's a newline and a bunch of spaces at the end there) Sadly, we don't know what comes next this time, so we move on to index 2048.

We repeat until we get the key to length 1024.

Note: if you're doing this in vim, you'll want these options: `:set binary` and `:set noendofline`. They will keep vim from messing with tabs and spaces and from appending a hidden newline to the end of the file.

Code

I scripted as much as possible, but it still requires manual intervention on each iteration.

```
#!/usr/bin/python2

def xor(xs, ys):
    return "".join([chr(ord(x) ^ ord(y)) for x, y in zip(xs, ys)])

def fread(path):
    with open(path) as f:
        return f.read()

def fwrite(path, data):
    with open(path, "w") as f:
        return f.write(data)

gooduserxml = fread("user-extension-xhtml.xml")
baduserxml = fread("home/tc/.local/share/mime/packages/user-extension-xhtml.xml")
badflag = fread("home/tc/h1dd3n_s3cr3t_f14g.jpg")
badblocklist = fread("home/tc/.mozilla/firefox/wir5mrb.default/blocklist.xml")
# If working key is there, use it, else calculate first 254 bytes
try:
    key = fread("key")
except IOError:
    key = xor(gooduserxml, baduserxml)

i = 0
while len(key) != 1024:
    fwrite("partialblocklist.xml", xor(badblocklist[i:i+len(key)], key))
    # Append to partialblocklist.xml, and then press enter
    raw_input("cur key len: " + str(len(key)))

    partialblocklist = fread("partialblocklist.xml")
    key = xor(partialblocklist, badblocklist[i:i+len(partialblocklist)])[:1024]
    fwrite("key", key)
    i = i + 1024 if i < len(badblocklist) - 2048 else 0

fwrite("flag.jpg", xor(badflag, key))
```

Flag

Picture of some meat with text SECCON{This is Virtual FAT too}

Lense commented on 11 Dec 2016

Owner

turns out there was squashfs-root-57/usr/local/firefox-ESR/browser/blocklist.xml

Sign up for free

to join this conversation on GitHub. Already have an account? [Sign in to comment](#)