Forensics / Académie de l'investigation - Dans les nuages

by bak, dans le cadre du France CyberSecurity Challenge.

Problème

Énoncé

> Le poste en cours d'analyse est connecté à un serveur web à l'adresse 10.42.42.132. Le serveur web est protégé par une authentification.
>
> Retrouvez le nom d'utilisateur et le mot de passe de cette connexion.
>
> Format du flag : FCSC{utilisateur:mot_de_passe}. Ce flag est sensible à la casse.
>
> Le fichier de dump à analyser est identique au challenge C'est la rentrée.

Une série de challenges partage le même vidage mémoire, celui-ci en est le dernier.

Interprétation

Cet énoncé nous indique une connexion web à l'adresse 10.42.42.132. Dans un premier temps, l'idée sera de d'identifier le processus en charge de la connexion afin de pouvoir obtenir les identifiants qu'il aura probablement conservé en mémoire.

Obtenir un profil volatility qui fonctionne

Afin de pouvoir analyser ce vidage mémoire à l'aide de l'outil `volatility`, il faut d'abord créer un profil correspondant à la machine d'où provient le vidage mémoire. La méthode permettant d'obtenir ce profil ne sera pas détaillée ici car beaucoup d'autres l'auront probablement déjà fait et je n'apporterai rien de plus.

Partons du principe que nous avons un profil volatility fonctionnel pour une machine *Debian 9.2.1 (Linux version 5.4.0-4-amd64)*.

Tracer la connexion vers 10.42.42.132

L'énoncé faisant mention d'une connexion vers l'adresse 10.42.42.132, un moyen facile d'identifier le processus à l'origine de celle-ci est d'utiliser le module `linux_netstat` de volatility:

```
$ python ~/src/volatility/vol.py -f dmp.mem --profile=LinuxDebian_5_4_0-4-
amd64x64 linux_netstat | grep "10\.42\.42\.132"
Volatility Foundation Volatility Framework 2.6.1
TCP      10.42.42.131    :60750 10.42.42.132    :   80 CLOSE_WAIT
chromium/119187
```

Le processus `chromium` ayant pour PID 119187 montre une connexion en attente de fermeture. Cela signifie que la connexion est plutôt récente, et que des informations intéressantes sont probablement encore stockées dans la mémoire du processus.

Autre information importante, la connexion a été effectuée sur le port 80, en utilisant donc le protocole HTTP, et non HTTPS.

Analyse de la mémoire du processus chromium/119187

Volatility permet d'extraire la mémoire d'un processus à l'aide du module `linux_dump_map` :

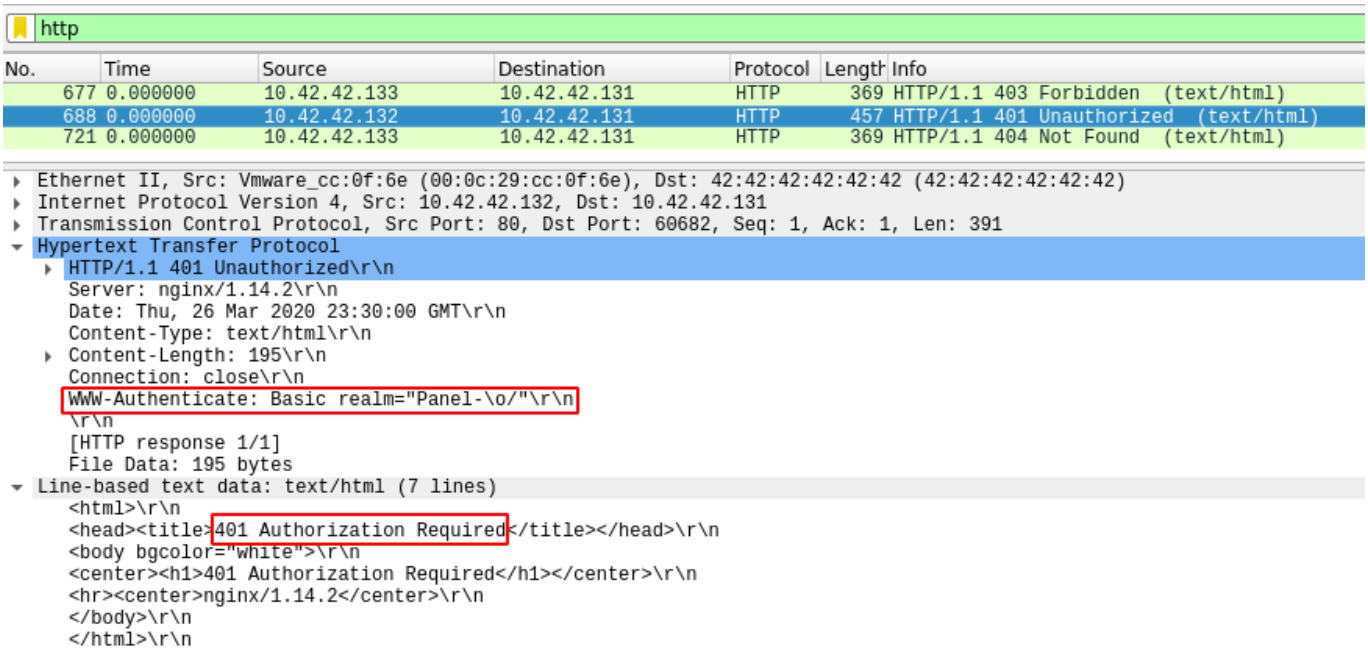
```
$ mkdir chromium_119187-memory
$ python vol.py -f dmp.mem --profile=LinuxDebian_5_4_0-4-amd64x64
linux_dump_map -p 119187 --dump-dir chromium_119187-memory/
Volatility Foundation Volatility Framework 2.6.1
Task       VM Start           VM End                       Length Path
---------- ------------------ ------------------ ------------------ ----
    119187 0x00005649c7d25000 0x00005649ca5f2000          0x28cd000
chromium_119187-memory/task.119187.0x5649c7d25000.vma
    119187 0x00005649ca5f2000 0x00005649d127a000          0x6c88000
chromium_119187-memory/task.119187.0x5649ca5f2000.vma
    119187 0x00005649d127a000 0x00005649d184b000           0x5d1000
chromium_119187-memory/task.119187.0x5649d127a000.vma
    119187 0x00005649d184b000 0x00005649d18ca000            0x7f000
chromium_119187-memory/task.119187.0x5649d184b000.vma
[...]
    119187 0x00007ffdb87f3000 0x00007ffdb87f6000             0x3000
chromium_119187-memory/task.119187.0x7ffdb87f3000.vma
    119187 0x00007ffdb87f6000 0x00007ffdb87f7000             0x1000
chromium_119187-memory/task.119187.0x7ffdb87f6000.vma
```

Afin d'effectuer une première analyse de la mémoire de ce processus, les outils `strings` et `grep` ont été utilisés. Néanmoins, il était difficile de trier telle quantité d'informations et plusieurs hypothèses restaient invérifiées. Par exemple, la méthode d'authentification utilisée par le serveur web, ou encore, la présence des identifiants dans la mémoire de ce processus.

Dans le but de restreindre le champ de recherche, une recherche sur la chaîne "10.42.42.132" a éte effectuée parmi tous les fichiers *.vma* :

```
$ grep -Ri "10.42.42.132"
Fichier binaire task.119187.0x7fb1cc000000.vma correspondant
```

Concernant le mode d'authentification employé, plusieurs marqueurs laissent penser que c'est l'*HTTP Basic* qui est en oeuvre ici. À l'aide de l'outil `Bulk extractor`, une capture de paquets réseaux a pu être extraite. Dans celle-ci, 3 requêtes HTTP peuvent être observées à destination de l'adresse 10.42.42.131. De plus, le serveur demande une authentification HTTP Basic, qui est fournie par le client mais semble incorrecte :



Reproduire l'infrastructure

Afin d'éclaircir toutes les hypothèses émises et d'avoir une meilleur compréhension de l'organisation de la mémoire du processus chromium, il a été décidé de reproduire l'infrastructure du challenge sur la machine virtuelle ayant servi à créer le profil volatility.

Pour simuler le serveur hébergé à l'adresse 10.42.42.132, le script python suivant a été utilisé :

```
import BaseHTTPServer
from SimpleHTTPServer import SimpleHTTPRequestHandler
import sys
import base64

key = ""

class AuthHandler(SimpleHTTPRequestHandler):
    ''' Main class to present webpages and authentication. '''
    def do_HEAD(self):
        print "send header"
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def do_AUTHHEAD(self):
        print "send header"
        self.send_response(401)
        self.send_header('WWW-Authenticate', 'Basic realm=\"Test\"')
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def do_GET(self):
        global key
        ''' Present frontpage with user authentication. '''
        if self.headers.getheader('Authorization') == None:
```

```
            self.do_AUTHHEAD()
            self.wfile.write('no auth header received')
            pass
        elif self.headers.getheader('Authorization') == 'Basic '+key:
            SimpleHTTPRequestHandler.do_GET(self)
            pass
        else:
            self.do_AUTHHEAD()
            self.wfile.write(self.headers.getheader('Authorization'))
            self.wfile.write('not authenticated')
            pass

def test(HandlerClass = AuthHandler,
         ServerClass = BaseHTTPServer.HTTPServer):
    BaseHTTPServer.test(HandlerClass, ServerClass)


if __name__ == '__main__':
    if len(sys.argv)<3:
        print "usage SimpleAuthServer.py [port] [username:password]"
        sys.exit()
    key = base64.b64encode(sys.argv[2])
    test()
```

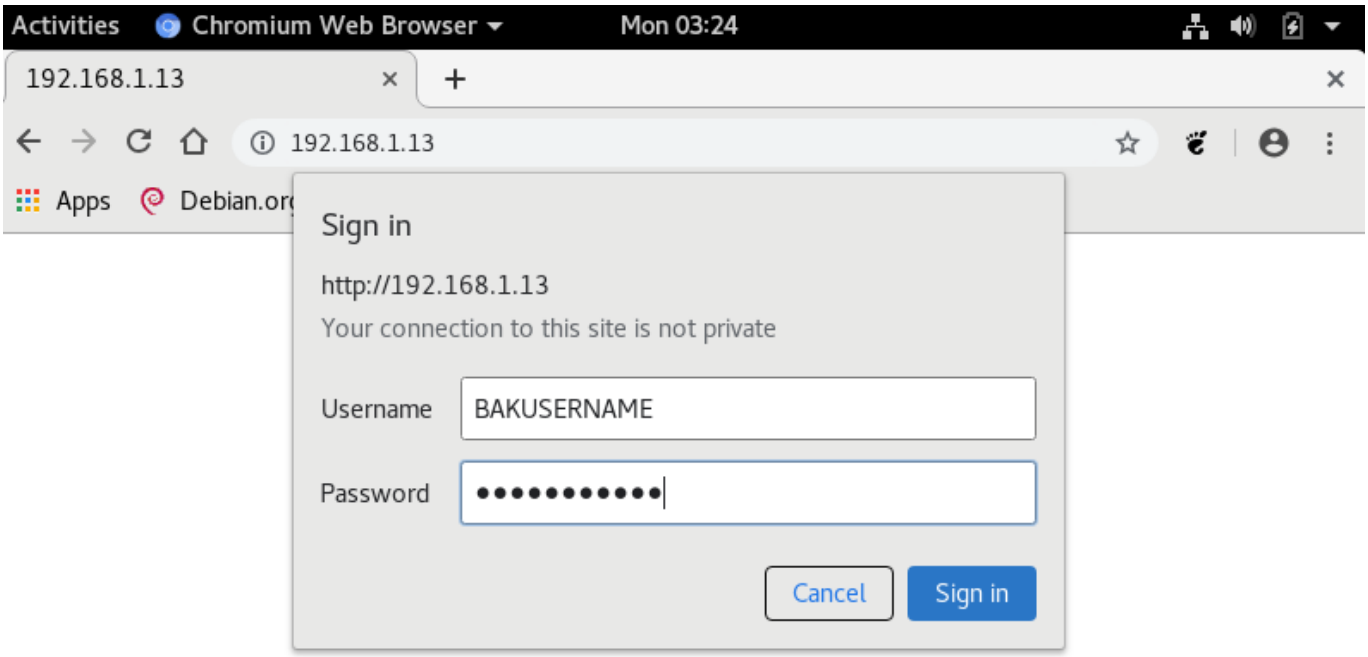
Celui-ci émule simplement un serveur HTTP demandant une authentification HTTP Basic.

`chromium` a donc été installé sur la machine virtuelle, puis, après avoir consulté plusieurs sites aléatoires, une connexion vers le serveur HTTP contrôlé a été établie.

Côté serveur :

```
$ sudo python http_basic_auth_server.py 80 BAKUSERNAME:BAKPASSWORD
Serving HTTP on 0.0.0.0 port 80 ...
```

Côté client :



Le couple d'identifiants **BAKUSERNAME:BAKPASSWORD** a été choisi afin d'être facilement reconnaissable en mémoire.

Un vidage de la mémoire de la machine virtuelle a ensuite été effectué, en utilisant le même outil que les concepteurs du challenge :

```
root@debian:/home/user/LiME/src# insmod lime-5.4.0-4-amd64.ko
"path=/tmp/mydmp.mem format=lime"
```

Analyse de la mémoire de la machine virtuelle

Une fois le vidage mémoire terminé (c'est un peu long, surtout quand cela nécessite de redimmensionner le disque attribué à la machine virtuelle), la mémoire du processus chromium peut être extraite de la même manière que précédemment.

En analysant la mémoire de ce processus à l'aide de `strings` et `grep`, plusieurs choses ont pu être observées :

```
$ cd mychromium-dump/
$ strings -an5 -es *.vma | grep BAKUSERNAME
$ strings -an5 -el *.vma | grep BAKUSERNAME
BAKUSERNAME
BAKUSERNAME
BAKUSERNAME
BAKUSERNAME
BAKUSERNAME
BAKUSERNAME
BAKUSERNAME
BAKUSERNAME
```

```
BAKUSERNAME
$ strings -an5 -el *.vma | grep -A1 BAKUSERNAME
BAKUSERNAME
e to undefined property {0}
--
BAKUSERNAME
BAKPASSWORD
--
BAKUSERNAME
#",64&3/"
--
BAKUSERNAME
sion 10
--
BAKUSERNAME
google.com
--
BAKUSERNAME
Times New Roman
--
BAKUSERNAME
&Copy
--
BAKUSERNAME
BAKUSERNA
--
BAKUSERNAME
BAKPASSWORD
```

- Les identifiants sont bien présents en mémoire.
- Ils sont encodés en *16-bit little endian* (paramètre **-el** de la commande `strings`).
- Le mot de passe est juxtaposé au nom d'utilisateur, dans la sortie de la commande `strings`.

L'inspiration

Fort de ces nouvelles informations, la mémoire du processus chromium du challenge a pu être étudiée d'un oeil nouveau :

```
$ cd chromium_119187-dump/
$ strings -an5 -el *.vma > strings.txt
$ less strings.txt
[...]
9Z9Z9
YZ9:9
62626262626262
6262626262
6262626262626262626262
i%^%^
U4343u43543
!$).056;>ACENQVZZ^ceiow{{{~
Admin3Kz7
```

```
5sdtYh68

 !&(/01=CEIORS_bcfghmvwxy}
-78;?ABDFGLMNPTUVW[]jkz{|
+234569HJKQXYZq~
#%'*@
Colibre
colibre
notosansmono
Belarusian
0123456789
01234567890
++55OQXXnn
[...]
```

Au milieu de toutes ces chaînes de caractères, **Admin3Kz7** semble particulièrement intéressant. Selon les observations effectuées précedemment, la chaîne suivante serait le mot de passe...

Bingo ! Flag : `FCSC{Admin3Kz7:5sdtYh68}`.