ExaTrack

Hey Uroburos! What's up ?

ExaTrack - Stéfan Le Berre (stefan.le-berre [at] exatrack.com)

Uroburos est un malware/APT, détecté en 2014, qui avait fait couler beaucoup d'encre dans le monde de la sécurité informatique. Il s'est distingué des autres par son driver 64b (rootkit) pour Windows, avec un bypass de PatchGuard. De plus le driver n'est pas signé, le malware utilisait alors une vulnérabilité dans un driver tiers pour passer en noyau. Pour plus de détails sur ce qui a déjà été fait sur ce malware nous vous invitons à lire la publication de "Andrzej Dereszowski" et "Matthieu Kaczmarek" en référence [1].

Il y a quelques mois maintenant nous avons identifié un code Uroburos/Turla datant de 2017. Après vérification le driver s'est révélé être une évolution de celui utilisé en 2014. Par doute nous avons creusé un peu ce nouveau driver et nous sommes rendu compte qu'il avait de sérieuses différences avec l'original, tout en gardant une même base. Nous présenterons donc ici une analyse de ce rootkit 64 bit.

L'analyse portera sur l'identification du rootkit depuis un dump mémoire (pour se placer dans le cadre d'une recherche de compromission), puis nous étudierons une partie de son nouveau protocole de communication. L'objectif est de pouvoir identifier la présence de ce rootkit à distance et sans avoir à s'authentifier sur une machine. Il faut noter que le rootkit ne cible que les serveurs.

Le code que nous analyserons est le suivant :
https://www.virustotal.com/en/file/f28f406c2fcd5139d8838b52da703fc6ffb8e5c00261d86aec90c28a20cfaa5b/analysis

Nous allons donc nous placer dans le cas d'une recherche de compromission sur un serveur. Pour ce faire nous utiliserons l'outil DumpIt de Comae (https://www.comae.io/) et analyseront le crashdump généré.

Identification de la compromission en noyau

Le driver se cache assez bien dans l'espace noyau, il n'est pas présent dans la liste des modules chargés et l'intégrité des autres modules est correcte.

Pour assister l'analyse nous utiliserons un outil interne à ExaTrack, il a pour but de valider l'intégrité du noyau et de mettre en avant les potentielles anomalies présentes.

Windows possède un système de Callback, lors de certains événements sur le système, comme la création de processus, des fonctions arbitraires peuvent être appelées. Parmi les points de contrôles regardés il y a donc les Callback kernel Windows.

Dans notre cas, en les regardant nous identifions une anomalie :

```
>>> ccb
# Check CallBacks
 [*] Checking \Callback\TcpConnectionCallbackTemp : 0xfffffa8002f38360
 [*] Checking \Callback\TcpTimerStarvationCallbackTemp : 0xfffffa8004dfd640
 [*] Checking \Callback\LicensingData : 0xfffffa80024bc2f0
[...]
 [*] PspLoadImageNotifyRoutine
 [*] PspCreateProcessNotifyRoutine
  Callback fffffa8004bc2874 -> SUSPICIOUS ***Unknown*** 48895c2408574881ec30010000488bfa
```

Une callback de `PspCreateProcessNotifyRoutine` est appelée lors de la création d'un processus. Elle est particulièrement intéressante pour modifier en amont les données et donc le comportement d'un nouveau processus. L'outil a identifié une entrée qu'il considère suspecte car pointant vers une adresse mémoire n'étant pas affectée à un driver.

Une deuxième anomalie est présente dans les callbacks, elle est moins visible car elle n'impacte pas en profondeur le fonctionnement du système, mais le modifie légèrement.

```
 [...]
 [*] IopNotifyShutdownQueueHead
  Name : Null
  Driver Object : fffffa80032753e0
    Driver : \Driver\Null
    Address: fffff88001890000
    Driver : Null.SYS
  Name : 000000a6
  Driver Object : fffffa8003d2adb0
    Driver : \Driver\usbhub
    Address: fffff88000da6000
    Driver : usbhub.sys
[...]
>>> cirp \Driver\Null
    Driver : \Driver\Null
    Address: fffff88001890000
    Driver : Null.SYS
    DriverUnload : fffff88001895100  c:\windows\system32\drivers\null.sys
        IRP_MJ_CREATE                     fffff88001895008  Null.SYS
        IRP_MJ_CREATE_NAMED_PIPE          fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_CLOSE                      fffff88001895008  Null.SYS
        IRP_MJ_READ                       fffff88001895008  Null.SYS
        IRP_MJ_WRITE                      fffff88001895008  Null.SYS
        IRP_MJ_QUERY_INFORMATION          fffff88001895008  Null.SYS
        IRP_MJ_SET_INFORMATION            fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_QUERY_EA                   fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_SET_EA                     fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_FLUSH_BUFFERS              fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_QUERY_VOLUME_INFORMATION   fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_SET_VOLUME_INFORMATION     fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_DIRECTORY_CONTROL          fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_FILE_SYSTEM_CONTROL        fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_DEVICE_CONTROL             fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_INTERNAL_DEVICE_CONTROL    fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_SHUTDOWN                   fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_LOCK_CONTROL               fffff88001895008  Null.SYS
```

```
        IRP_MJ_CLEANUP                      fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_CREATE_MAILSLOT              fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_QUERY_SECURITY               fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_SET_SECURITY                 fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_POWER                        fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_SYSTEM_CONTROL               fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_DEVICE_CHANGE                fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_QUERY_QUOTA                  fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_SET_QUOTA                    fffff80002abb1d4  ntoskrnl.exe
        IRP_MJ_PNP                          fffff80002abb1d4  ntoskrnl.exe
```

Le driver "\Driver\Null" est référencé dans la liste des modules appelés lors d'une extinction du système. Ce driver n'est pas censé être dans cette liste. Hors sa table IRP ne semble pas avoir été modifiée et pointe même vers ntoskrnl (pour l'interface IRP_MJ_SHUTDOWN). Nous n'avons pas identifié de réelle utilité à cette callback.

Continuons la recherche d'anomalies, Windows possède de nombreux systèmes de filtres dans ses gestionnaires d'IO (Input Output). Parmi ces IO il y a le réseau, il se découpe en plusieurs parties, celle qui nous intéressera ici est NetIO.

NetIo propose lui aussi un système de callbacks, elles permettent d'agir sur les données réseaux échangées. Ces callbacks sont nommées "Callout" et sont donc des callbacks réseaux, les structures ne sont pas documentées et ne figurent pas dans les symbols de Windows. C'est donc un bon endroit où se positionner pour un malware. Dans le dump on trouve 5 callbacks qui pointent vers du code n'appartenant, encore une fois, à aucun driver.

```
>>> cnetio
  [*] NetIo Callouts (callbacks) : fffffa8004965000 (4790)
  Callback fffffa8004bd9580 -> SUSPICIOUS ***Unknown*** 488bc448895808488950105556574154
  Callback fffffa8004bca6b0 -> SUSPICIOUS ***Unknown*** 33c0c3cc40534883ec20488b89500100
  Callback fffffa8004bd9af8 -> SUSPICIOUS ***Unknown*** 4883ec286683f91474066683f916750f
  Callback fffffa8004bd9ca0 -> SUSPICIOUS ***Unknown*** 48895c24084889742410574883ec4048
  Callback fffffa8004bd9de0 -> SUSPICIOUS ***Unknown*** 4c8bdc49895b0849896b104989731857
```

L'une de ces fonctions sera plus longuement étudiée dans la suite du document.

Enfin nous allons rechercher des drivers qui seraient chargés mais qui chercheraient à se cacher de Windows.

```
>>> pe
[...]
  [OK] fffff88001899000 : \SystemRoot\System32\Drivers\Beep.SYS
  [OK] fffff88000da6000 : \SystemRoot\system32\DRIVERS\usbhub.sys
  [NO] fffffa8004bb8000 (Header overwritten)
  [OK] fffff88006a00000 : \SystemRoot\system32\DRIVERS\E1G6032E.sys
  [OK] fffff880017d2000 : \SystemRoot\System32\Drivers\Npfs.SYS
[...]
>>> dq fffffa8004bb8000 100
 FFFFFA8004BB8000 0000000300000000  0000FFFF00000004  ....?...?...¦¦..
 FFFFFA8004BB8010 00000000000000B8  0000000000000040  ........@.......
 FFFFFA8004BB8020 0000000000000000  0000000000000000  ................
```

```
 FFFFFA8004BB8030 0000000000000000  000000D800000000  ................
 FFFFFA8004BB8040 CD09B4000EBA1F0E  685421CD4C01B821  ??.?....!.?L.!Th
 FFFFFA8004BB8050 72676F7270207369  6F6E6E6163206D61  is program canno
 FFFFFA8004BB8060 6E75722065622074  20534F44206E6920  t be run in DOS
 FFFFFA8004BB8070 0A0D0D2E65646F6D  0000000000000024  mode....$.......
 FFFFFA8004BB8080 095520395A3B417D  0955203909552039  }A;Z9 U.9 U.9 U.
 FFFFFA8004BB8090 095520A609542039  0955203C092E28A6  9 T.. U..(..< U.
 FFFFFA8004BB80A0 0955203B0928E61E  09552051093 8E61E  ?.(.; U.?.8.Q U.
 FFFFFA8004BB80B0 09552038092FE61E  09552038092DE61E  ?./.8 U.?.-.8 U.
 FFFFFA8004BB80C0 0955203968636952  0000000000000000  Rich9 U.........
 FFFFFA8004BB80D0 0000000000000000  0006866400000000  ............d.?.
 FFFFFA8004BB80E0 000000005900F3CF  202200F000000000  ...Y.........."
 FFFFFA8004BB80F0 00042E000008020B  000000000001BC00  ??....?...?.....
>>> list fffffa8004bb8000 fffffa8004bbb000
    FFFFFA8004BB8000 rwx-
    FFFFFA8004BB9000 rwx-
    FFFFFA8004BBA000 rwx-
```

Là encore une anomalie importante est observable. Un driver est présent en mémoire et a écrasé ses entêtes MZ et PE, probablement pour ne pas être trouvé en cas de recherches sur de la mémoire brute. Son adressage correspond aux callbacks suspectes identifiées précédemment et il est mappé avec les droits RWX.

Tous ces éléments confortent l'idée qu'un malware opère depuis le noyau. Nous allons maintenant analyser une partie de son code (principalement la communication réseau) pour avoir une idée de son fonctionnement.

Analyse du driver

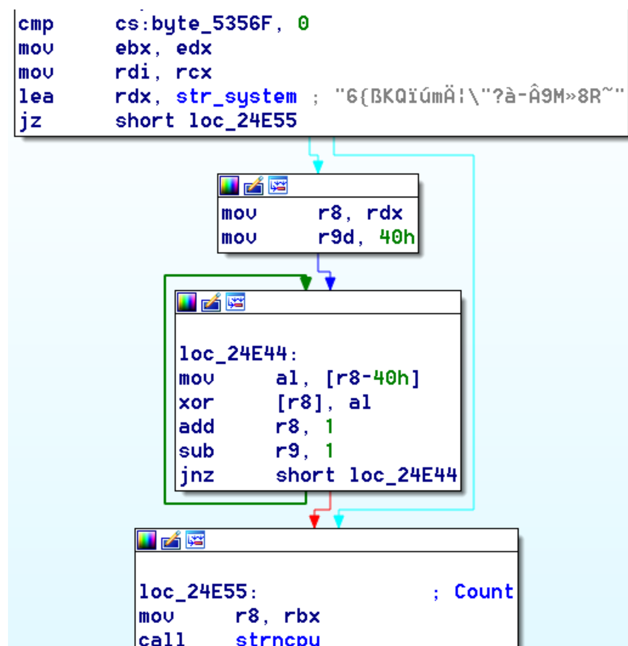
Point d'entrée

Durant son initialisation le driver cible rapidement le Device "Null". Il récupère un pointeur vers l'objet, puis va l'enregistrer dans la liste des callbacks de "shutdown" évoquée précédemment. Il enregistre également la callback lors des créations de processus.

```
 [...]
      if ( (unsigned int)get_top_deviceObjet(L"\\Device\\Null", &device_obj_null)
        && (result = get_top_deviceObjet(L"\\Device\\Beep", &device_obj_null),
(_DWORD)result) )
      {
        __asm { xchg    rbx, qword ptr cs:isNullDeviceFailed }
      }
      else
      {
        v5 = IoRegisterShutdownNotification(device_obj_null);
        if ( v5 || (drvobj_null = device_obj_null->DriverObject, (v5 =
sub_4E21C(byte_1188D)) != 0) )
[...]
            PsSetCreateProcessNotifyRoutine(cbCreateProcess, 0i64);
[...]
```

Chiffrement des chaînes de caractères

Pour éviter une indentification trop rapide par les anti-virus les chaînes de caractères discriminantes à Uroburos sont chiffrées. Chaque bloc de données chiffrées fait 0x40 octets, une opération XOR est effectuée avec les 0x40 octets précédents.



La fonction de déchiffrement peut se représenter comme suit (un oneliner est plus simple pour IDA). Il est ainsi possible déchiffrer toutes les chaînes.

```
Python>def decrypt(addr, clen): return ''.join(chr(b) for b in
[struct.unpack('B'*clen,idaapi.get_many_bytes(addr,64))[a] ^
struct.unpack('B'*clen,idaapi.get_many_bytes(addr-clen,clen))[a] for a in xrange(clen)])

Python>[decrypt( 0x53530 + (i*0x80) , 0x40).replace("\x00",'') for i in xrange(38)]
['system', 'isapi_http', 'isapi_log', 'isapi_dg', 'isapi_openssl', 'shell.{F21EDC09-85D3-
4eb9-915F-1AFA2FF28153}', 'Hd1', 'Hd2', 'RawDisk1', 'RawDisk2', 'wininet_activate',
'dmtev', 'Ultra3', 'Ultra3', 'services_control', 'fixdata.dat', '$NtUninstallQ817473$',
'fdisk.sys', 'fdisk_mon.exe', '400', '16', '{AAAA1111-2222-BBBB-CCCC-DDDD3333EEEE}',
'~WA434.tmp', '~WA4276.tmp', '.', '~WA356.tmp', 'rasmon.dll', 'rasman.dll', 'user',
'internat', 'NTUSER.DAT', 'ntuser.dat.LOG1', '.', 'mscrt.dll', 'msvcrt.dll', '0', '1',
'.']
```

Une règle YARA de la fonction cryptographique est fournie en annexe.

Interception réseau

Comme vu plus haut, des callbacks réseaux sont installées. Elles seront enregistrées par la fonction "FwpsCalloutRegister0" (qui permet d'ajouter un filtre réseau) et donne la possibilité au driver de faire suivre ou non les données.

```
 v20 = addCalloutAddress(
         &stru_14930,
         &a2,
```

```
          DeviceObject,
          (__int64)intercept_packet,
          (__int64)&ret_null,
          (__int64)a6,
          (__int64)&v47,
          (__int64)&v34,
          &a9,
          &a10);
```

La fonction "intercept_packet" (à l'adresse `fffffa8004bd9580` dans le dump mémoire) va analyser les données transitant par les connexions réseaux. Chose intéressante, elle ne regardera pas les données passant par le port 139, elle ne regardera que les données reçues, et seulement si l'hôte est le serveur.

```
  if ( v9 || LOWORD(a1->layerId) == 20 && a1->pIP_infos->src_port == 139 )
    return;
  if ( LOWORD(a1->layerId) == 22 && a1->pIP_infos->src_port == 139 )
    return;
[...]
    fwpsCopyStreamDataToBuffer0(v8, datas_tcp_buffer, *(_QWORD *)(v8 + 48), &v31);
[...]
      buffer_type_2 = find_and_decode_datas(datas_tcp_buffer, v24, *((_DWORD *)v11 +
0x1FF) == 0, &a4a);
```

La fonction " find_and_decode_datas" est chargée de tester les différents protocoles de communication acceptés. Dans notre cas nous étudierons une communication HTTP. L'objectif étant de voir s’il est possible, à distance, de déterminer si un serveur est compromis par Uroburos.

Le malware valide que le message HTTP est bien conforme à un message standard. Puis il cherchera à trouver un message lui étant destiné dans l'un des arguments de l'entête HTTP.

```
  if ( space_offset_1 != 3i64
    || ((v18 = *(_WORD *)Buf < (unsigned __int16)str_GET, *(_WORD *)Buf !=
(_WORD)str_GET)
     || (v19 = Buf[2], v18 = (unsigned __int8)v19 < BYTE2(str_GET), v19 !=
BYTE2(str_GET)) ? (v20 = -v18 - (v18 - 1)) : (v20 = 0),
        v20) )
  {
    if ( space_offset_1 != 4 || *(_DWORD *)Buf != str_POST )
      return 0i64;
  }
[...]
    if ( *(_DWORD *)start_word_2 != *(_DWORD *)"http://"
[...]
  if ( v33 != *(_DWORD *)"HTTP/" || (v35 = v32[4], v34 = v35 < aHttp_0[4], v35 !=
aHttp_0[4]) )
[...]
    || !(unsigned int)check_and_decode_buffer(&Buf[nextline], v14, response_tag,
out_decoded_b64, v7) )
[...]
```

La fonction "check_and_decode_buffer" va identifier le premier ": " de la ligne et interpréter la suite, soit la valeur de l'argument HTTP.

```
      v15 = memchr(v10, ':', (unsigned int)(v14 - 1));
      if ( !v15 || *((_BYTE *)v15 + 1) != ' ' )
        return (unsigned int)v5;
```

S'en suit un contrôle du message au moyen de plusieurs checksums.

```
    v10 = get_checksum(datas, 12);
    result = (unsigned __int64)reverse_littleendian(v10) == *(_DWORD *)v4;
```

La fonction de checksum est assez particulière, elle utilise un "threefish256" modifié. Le hash de sortie est tronqué à 4 octets.

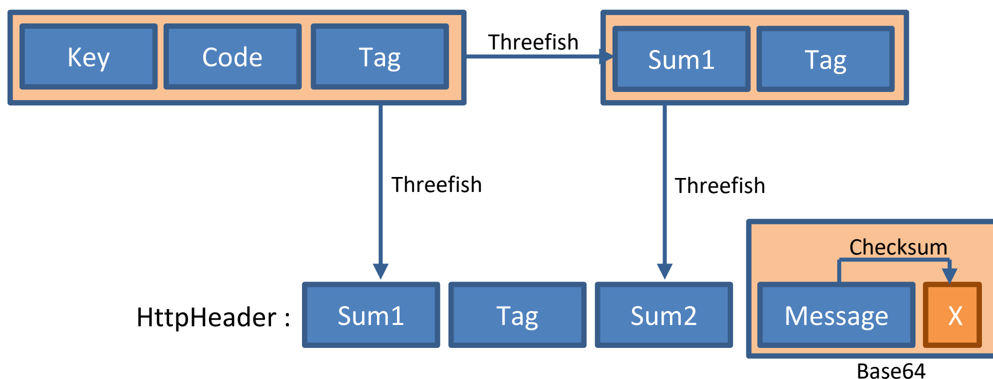
```
    crypto_it(v3, v6, (v5 - 1) >> 5, 32);
[...]
    memcpy((void *)(v3 + *(_QWORD *)(v3 + 8) + 64), v6, v5);
    *(_QWORD *)(v3 + 8) += v5;
```

La fonction de hashage est utilisée trois fois pour obtenir un hash final, la première fois avec un buffer statique qui va permettre d'initialiser les données, la seconde avec les données à hasher et la dernière avec le hash de ces données. Ce hash final sera tronqué sur 4 octets et utilisée comme checksum.

Une clé est initialisée, il y en a probablement une par cible. Elle permet d'avoir un secret partagé entre le malware et l'opérateur. Elle servira dans le calcul des hashs mais ne sera pas envoyée dans la requête.

Le message est contenu à la fin de la ligne, il fait 7 octets + 1 octet de contrôle d'intégrité. Celui-ci n'est qu'une addition des 7 octets précédents. Ce bloc de 8 octets est ensuite encodé en base64.

Les données de la requête HTTP seront ajustées comme suit :



Il est intéressant de noter que l'élément "Code" n'est pas présent dans la requête finale. Quatre valeurs peuvent être utilisées et elles sont bruteforcées par le malware lors de la vérification.

Quand un serveur est contacté avec une requête de ce type (sur un port déjà ouvert) c'est le rootkit qui va nous répondre (les données ne seront pas transférées au userland).

Si le message correspond au format attendu le driver va envoyer une réponse, elle aussi formatée. Elle possède une taille variable de padding avec des octets aléatoires.

```
    if ( reply_datas[6] & 2 )
    {
      v8 = 8 * (rand() % -32);
      v4 = v8;
      v9 = &v21[-v8];
      if ( v8 )
      {
        v10 = v8;
        do
        {
          *v9++ = rand();
          --v10;
        }
        while ( v10 );
      }
    }
  }
  *(_BYTE *)(v7 + 0xBE0);
  sprintf(Dest, "HTTP/1.1 200 OK\r\nContent-Length: %u\r\nConnection: %s\r\n\r\n",
(unsigned int)(v4 + 8));
```

Seuls les 8 premiers octets répondent à un format spécifique, toutes les autres données sont aléatoires. La validité de ce block se fait par un checksum additionnel entre les 7 premiers octets, avec le résultat stocké dans l'octet 8. Ce cheksum est similaire à celui évoqué précédemment.

Nous sommes donc en mesure de développer un PoC pour vérifier à distance si un serveur est compromis par cette version du malware :

```
> request_builder.py 192.168.48.133 8080
datas :
 0000000000000000 E8 F6 E8 4E 72 61 03 EA  C8 B3 DD 8D 25 D0 26 12  ...Nra♥.....%.&↕
 0000000000000010 B7 F9 50 E5 8C D2 01 62  A0 37 2F FB AD C8 91 DA  ..P...☺b.7/.....
 0000000000000020 44 A5 53 C7 1D 76 0E 4D  AC AF F7 18 F4 12 57 A2  D.S.↔v♫M...↑.↕W.
 0000000000000030 A0 75 3B 0F 50 C5 6C 55  31 4B A1 9F D0 2E F4 F4  .u;☼P.lU1K......
 0000000000000040 30 39 93 13 1A DF B8 A2  B4 7C DB 88 55 DE 26 98  09.‼→....|..U.&.
 0000000000000050 98 04 29 6F AF 25 CF 9F  FA F5 90 0D D8 23 E9 97  .♦)o.%.......#..
  [*] checksum OK - Host is compromised
```

Ce PoC est disponible sur notre site [2].

Différences et similarités entre Uroburos 2014 et 2017

Plusieurs choses sont étonnantes depuis la version 2014 du malware. Cette liste n'est vraiment pas exhaustive mais permet de se rendre compte de l'état du projet aujourd'hui.

Les noms des fichiers sont restés les mêmes et le nom du service aussi, ce qui le rend facilement détectable par n'importe quel IOC.

Le driver est toujours chargé avec un exploit VirtualBox. Une exploitation de vulnérabilité noyau est donc faite à chaque redémarrage du système.

Le bypass PatchGuard a été supprimé, ce qui limite forcément les altérations du noyau par la suite.

```
Driver 2014:
  if ( v2 )
    installService(v3);
  v4 = PG_bypass();
  if ( v4 )
    goto LABEL_16;
  ObjectAttributes.Length = 48;
  ObjectAttributes.RootDirectory = 0i64;

Driver 2017:
    if ( v2 )
      installService(cp_DriverObject);
    ObjectAttributes.RootDirectory = 0i64;
    ObjectAttributes.SecurityDescriptor = 0i64;
```

Le driver `\Driver\Null` est toujours utilisé et le device `\Device\FWPMCALLOUT` y est toujours attaché.

```
>>> drv_stack \Driver\Null
- Stack of device name : \Device\FWPMCALLOUT
  Driver Object : fffffa80032753e0
    Driver : \Driver\Null
    Address: fffff88001890000
    Driver : Null.SYS

- Stack of device name : \Device\Null
  Driver Object : fffffa80032753e0
    Driver : \Driver\Null
    Address: fffff88001890000
    Driver : Null.SYS
```

Le système de chiffrement des chaînes de caractères est resté le même.

L'enregistrement dans les `IopNotifyShutdownQueueHead` du driver est une bonne idée en soit mais nous n'avons pas constaté son utilisation. Une utilisation possible de cette callback serait, qu'une fois le malware est injecté en noyau, qu'il supprime ses clés registres et ne les réécrivent qu'au moment de l'extinction.

Le système de checksum a évolué, le malware utilise Threefish et le format son message a évolué depuis la version de 2014. L'objectif étant surement d'échapper aux signatures réseaux.

Des fonctionnalités ont été supprimées et certaines parties du code nous semblent moins avancées qu'en 2014. En revanche la correction d'une faute d'anglais `221 Service closing transmittion channel` pour donner `221 Closing transmission channel` nous permet de dater le driver comme une version plus récente (voir référence [1]).

Globalement le rootkit possède encore de sérieux atouts, mais aussi plusieurs négligences, comme les noms des fichiers et des clés registres, laissent à penser qu'il ne sera utilisé que sur des serveurs isolés. Malgré une apparente perte de vitesse les malwares en noyau (64b) sont encore bien présents et ne sont pas prêt de disparaitre car leur présence est plus difficile à identifier qu'en userland.

Références

[1] Publication de "Andrzej Dereszowski" et "Matthieu Kaczmarek", l'une des plus abouties sur le sujet :
http://artemonsecurity.com/uroburos.pdf

[2] PoC de requête http :
http://www.exatrack.com/public/uroburos_poc.py

[3] SNAKE CAMPAIGN & CYBER ESPIONAGE TOOLKIT:
http://artemonsecurity.com/snake_whitepaper.pdf

Annexes

Signatures YARA :

```
rule Sig
{
  strings:
    $strings_crypt = { 4d 8b c1 41 ba 40 00 00 00 41 ?? ?? ?? 41 ?? ?? 49 83 c0 01 49 83
ea 01 75 ??}
    $hash_part1 = { 49 c1 c3 0e 4e ?? ?? ?? 4c 33 dd 4c 03 c7 4c 03 c1 48 c1 c0 10 49 33
c0 4d 03 c3 48 03 e8 48 c1 c8 0c 48 33 c5 49 c1 cb 07 4d 33 d8 4c 03 c0 49 03 eb 49 c1 c3
17 4c 33 dd 48 c1 c8 18 49 33 c0 4d 03 c3 48 03 e8 49 c1 cb 1b 4d 33 d8 4c 03 df 4c 03 d9
48 c1 c0 05 48 33 c5 4a ?? ?? ?? ??}
  condition:
    1 of them
}
```