MOSCOW FORENSICS DAY ‘25

МАКСИМ СУХАНОВ

«РОЛЬ ОБРАТНОЙ РАЗРАБОТКИ
В ПОНИМАНИИ АРТЕФАКТОВ:
КОГДА ДОКУМЕНТАЦИЯ — ВРАГ,
А ДЕКОМПИЛЯТОР — ДРУГ»

MFD

Роль обратной разработки в понимании артефактов: когда документация – враг, а декомпилятор – друг

Максим Суханов
Руководитель направления обнаружения киберугроз и реагирования на инциденты CICADA8

ОБО МНЕ

- Ранее работал в BI.ZONE, Group-IB
- 15 лет в сферах Digital Forensics, Incident Response, Malware Analysis, Threat Intelligence
- Занимался обратной разработкой форматов NTFS, FAT12/16/32, exFAT, реестра Windows, теневых копий
- Создаю утилиты и библиотеки:
  - yarp (парсер файлов реестра Windows),
  - dfir_ntfs (парсер томов NTFS, FAT12/16/32 exFAT, а также теневых копий),
  - winmem_decompress (распаковщик сжатых страниц из дампов памяти),
  - и многое другое…
- Ищу и нахожу уязвимости в парсерах файловых систем, загрузчиках и других местах

МАКСИМ СУХАНОВ

Руководитель направления обнаружения киберугроз и реагирования на инциденты CICADA8

m.suhanov@cicada8.ru

Проблема

- Есть наблюдаемые артефакты, следы, аномалии…
  - Как, в принципе, обосновать их интерпретацию?
- Два подхода:
  - Сослаться на кого-то: статью, книгу, документацию, иные чужие результаты.
  - Найти ответ и обосновать самому: провести экспертный эксперимент и т. п.

Проблема в одной цитате

«Я доверяю документации»

02

«Обман» со стороны вендоров

CIDADA8

Временных меток в NTFS нет!

- Это нам сообщает официальная документация [1][2]:

STANDARD_INFORMATION structure

01/06/2021

[This structure is valid only for version 3 of NTFS volumes; it may be altered in future versions.]

Contains the standard information attribute. This attribute is present in every base file record and must be resident.

Syntax

C++ Copy

```
typedef struct _STANDARD_INFORMATION {
  UCHAR Reserved[0x30];
  ULONG OwnerId;
  ULONG SecurityId;
} STANDARD_INFORMATION, *PSTANDARD_INFORMATION;
```

Members

Reserved

Reserved.

OwnerId

The identifier of the file owner, from the security index.

SecurityId

The security identifier for the file.

FILE_NAME structure

01/07/2021

[This structure is valid only for version 3 of NTFS volumes; it may be altered in future versions.]

Represents a file name attribute. A file has one file name attribute for every directory it is entered into.

Syntax

C++ Copy

```
typedef struct _FILE_NAME {
  FILE_REFERENCE ParentDirectory;
  UCHAR          Reserved[0x38];
  UCHAR          FileNameLength;
  UCHAR          Flags;
  WCHAR          FileName[1];
} FILE_NAME, *PFILE_NAME;
```

Members

ParentDirectory

A file reference to the directory that indexes to this name. See MFT_SEGMENT_REFERENCE.

Reserved

Reserved.

FileNameLength

The length of the file name, in Unicode characters.

Flags

The file name flags.

FILE_NAME_NTFS (0x01)

FILE_NAME_DOS (0x02)

FileName

The first character of the file name.

Ответ

- Видимо, вендор защищает служебные структуры NTFS от «кривых ручек».
- Поэтому «ненужные» поля «зарезервированы».

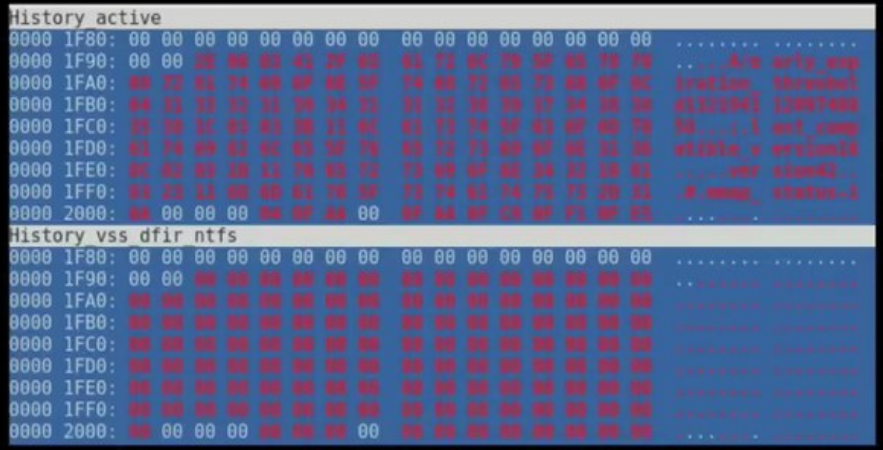
03

Забыли, что все уже не так

CIDADA8

Пропали пользовательские данные из теневой копии!

- А такого быть не может [3]!



Can I exclude files from a shadow copy to save space?

VSS is designed to create shadow copies of entire volumes. Temporary files, such as paging files, are automatically omitted from shadow copies to save space.

To exclude specific files from shadow copies, use the following registry key: `FilesNotToSnapshot`.

> ⓘ Note
>
> The `FilesNotToSnapshot` registry key is intended to be used only by applications. Users who attempt to use it encounter limitations such as:
>
> - It can't delete files from a shadow copy that was created on a Windows Server by using the Previous Versions feature.
> - It can't delete files from shadow copies for shared folders.
> - It can delete files from a shadow copy that was created by using the Diskshadow utility, but it can't delete files from a shadow copy that was created by using the VssAdmin utility.
> - Files are deleted from a shadow copy on a best-effort basis. This means that they aren't guaranteed to be deleted.

For more information, see Excluding Files from Shadow Copies.

Ответ

- Начиная с Windows 8, по умолчанию пользовательские файлы исключаются из теневой копии.
  - ✓ Но в некоторых случаях все равно попадают!
- Это поведение официально не было задокументировано.
- Известны, как минимум, два ответа вендора клиентам, где это поведение описано…
  - Из них один случай в контексте атаки шифровальщика! [4]

*7 We have also received a response from Microsoft indicating that it is highly likely this function is the cause of the file corruption.

*8 Although the reasons for this specification change have not been made public, we speculate that it is related to performance issues from saving all data into a snapshot, utilization efficiency issues in the snapshot area, excessive expansion of user data, and the fact that “file history” is now recommended for backing up user data.

Детали

- Сначала теневая копия создается, затем из нее исключаются кластеры с пользовательскими данными.
- За процесс исключения отвечает «*SrTasks.exe*».
- Пользовательские файлы находятся по встроенным правилам (в частности, по расширениям) путем итерации по записям в $MFT.

Задокументировали другую реализацию файловой системы...

- Не от Windows NT,
  а от Windows 95 [5]!

Yes, the (in)famous fatgen103.pdf document, retired from Microsoft's own site, but still available at
https://web.archive.org/web/20210723100623/http://download.microsoft.com/

It sternly says at page 14:

"It is really quite simple how this works. The FAT type– one of FAT12, FAT16, or FAT32–is determined by the count of clusters on the volume and nothing else."

But see below...:

[...]
> *But as Alain pointed Microsoft FAT implementation (fastfat.sys) for*
> *FAT32 detection does not use number of clusters, but rather explicit*
> *marking of FAT32. Same behavior has Linux kernel implementation*
> *(msdos.ko and vfat.ko) and same in dosfstools project. Hence this*
> *non-standard FAT32 can be still created by mkfs.fat.*

Actually, what I meant in my previous mail was that *everything else* that is referred to in the FAT32 extended boot block (root directory location, etc.) would depend on this explicit marking (fatLen/bigFatLen), but not the number of FAT bits themselves.

However, your remark got me thinking, and so I tried it out on a Windows 10 VM.

... and lo and behold: the mere presence of a FAT32 extended boot block (as signaled by fat_len being 0) was enough to make it use 32 fat bits, even if the number of clusters was too low for FAT32.

Sorry for having been to easily misled by an assertive, but nonfactual, Microsoft statement.

Given this finding, I'll shortly make a new mtools release that will consider presence of FAT32 extended boot block as well for picking appropriate number of FAT bits.

Regards,

Alain

04

Кейс: Prefetch

Что хранится в этих файлах?

Op-EXPLORER.EXE-[...]-[...].pf

Гипотезы и версии

1. *Уж не троян ли?*
2. *[…] когда сама программа запрашивает отслеживание самой себя*
3. *boot prefetch*
4. *[…] процесс, […] живущий исключительно в оперативке*

- Правильный ответ есть в блоге про ASP.NET [6], но там артефакт (файл «*Op-*.pf*») упоминается вскользь…

Правильный ответ из блога

The features simple uses the operation recorder API. See OperationStart and OperationEnd APIs on msdn. This APIs require an integer used as an Id of the operation. We use a hash code of the app domain app id, which is a string.

```
string domainId = HttpRuntime.AppDomainAppId;
int operationId = domainId.GetHashCode();
```

The OperationStart call is made the first time the app domain is created. The OperationEnd API is called at the end of the request in which we called OperationStart. A .pf file should be created on the prefetch directory:

```
C:>dir %windir%Prefetch
Volume in drive C has no label.
Volume Serial Number is CC5B-113C

Directory of C:WindowsPrefetch

02/16/2012  04:17 PM    <DIR>          .
02/16/2012  04:17 PM    <DIR>          ..
02/16/2012  04:17 PM           164,832 Op-W3WP.EXE-90E54C33-765F360E.pf
               1 File(s)        164,832 bytes
               2 Dir(s)  30,203,154,432 bytes free
```

Декомпилируем ядро: PfSnOperationProcess

```
59          do
60          {
61            if ( *v13 == 92 )
62              break;
63            ++v10;
64            --v13;
65          }
66          while ( (unsigned __int64)v13 >= v11 );
67          if ( v10 && v11 + 2i64 * (unsigned int)(v12 - v10) )
68          {
69            RtlStringCbPrintfW(pszDest, 60ui64, L"Op-%.17s-%08X");
70            v14 = *(_DWORD *)(a1 + 4);
71            v15 = v14 & 1;
72            v2 = (*(_BYTE *)(a1 + 1) & 1) == 0;
73            v20 = *(_DWORD *)(a1 + 8);
74            if ( v2 )
75            {
76              if ( (v14 & 2) != 0 || v17 == 2 )
77              {
78                v15 |= 2u;
79              }
80              else if ( (v14 & 4) != 0 )
81              {
82                v15 |= 4u;
83              }
84              PfSnBeginScenario(Process, pszDest, 1u, v15, 0i64, 0i64);
85            }
86            else
87            {
88              PfSnEndProcessTrace(Process, (unsigned int)(v15 + 8), pszDest);
89            }
90            v9 = 0;
91          }
92          else
93          {
94LABEL_27:
95            v9 = STATUS_INTERNAL_ERROR;
96          }
97        }
98        goto LABEL_17;
```

CIDADA8

Декомпилируем ядро: PfSnSetPrefetcherInformation

```
__int64 __fastcall PfSnSetPrefetcherInformation(__int64 a1, __int128 *a2, int a3, KPROCESSOR_MODE a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( a3 == 32 )
  {
    v12 = *(_PREFETCHER_INFORMATION *)a2;
    if ( *(_QWORD *)&v12.Version != 0x6B75684300000001i64 )
      return (unsigned int)-1073741811;
    if ( v12.PrefetcherInformationClass <= 8u && (v7 = 0x128, _bittest(&v7, v12.PrefetcherInformationClass))
      || SeSinglePrivilegeCheck(SeProfileSingleProcessPrivilege, a4) )
    {
      if ( v12.PrefetcherInformationClass != 3 )
      {
        switch ( v12.PrefetcherInformationClass )
        {
          case 5:
            if ( v12.PrefetcherInformationLength != 12 )
              return (unsigned int)-1073741811;
            if ( a4 && ((__int64)v12.PrefetcherInformation & 3) != 0 )
              ExRaiseDatatypeMisalignment();
            *(_QWORD *)&v12.Version = *(_QWORD *)v12.PrefetcherInformation;
            v12.PrefetcherInformationClass = *((_DWORD *)v12.PrefetcherInformation + 2);
            v8 = PfSnOperationProcess((__int64)&v12);
            break;
          case 6:
            if ( v12.PrefetcherInformationLength == 72 )
            {
              if ( a4 && ((__int64)v12.PrefetcherInformation & 3) != 0 )
                ExRaiseDatatypeMisalignment();
              v16[0] = *(_OWORD *)v12.PrefetcherInformation;
              v16[1] = *((_OWORD *)v12.PrefetcherInformation + 1);
              v16[2] = *((_OWORD *)v12.PrefetcherInformation + 2);
              v16[3] = *((_OWORD *)v12.PrefetcherInformation + 3);
              v17 = *((_QWORD *)v12.PrefetcherInformation + 8);
              if ( LODWORD(v16[0]) == 2 )
              {
                PfSnPrefetchCacheEntryUpdate(v16);
                return 0;
              }
              return (unsigned int)-1073741811;
            }
            return (unsigned int)-1073741306;
          case 8:
```

PrefetcherInformationClass = 5?

- Есть заголовочный файл из Process Hacker [7].
- Происхождение названий неизвестно (скорее всего, из старых заголовочных файлов от компании Microsoft).
- **PrefetcherBootControl – неактуальное название** (коммит датирован 2011 годом)…

```
55 typedef enum _PREFETCHER_INFORMATION_CLASS
56 {
57     PrefetcherRetrieveTrace = 1, // q: CHAR[]
58     PrefetcherSystemParameters, // q: PF_SYSTEM_PREFETCH_PARAMETERS
59     PrefetcherBootPhase, // s: PF_BOOT_PHASE_ID
60     PrefetcherRetrieveBootLoaderTrace, // q: CHAR[]
61     PrefetcherBootControl // s: PF_BOOT_CONTROL
62 } PREFETCHER_INFORMATION_CLASS;
```

PrefetcherInformationClass = 5

- Запись трассы инициируется через вызов функции OperationStart() (из advapi32.dll).
  - Никакого отношения к загрузке!
- Запись трассы завершается через вызов функции OperationEnd().
- Не более одной трассы на процесс в один момент времени.
- Если в трассе почти нет I/O, то она не сохраняется!
  - Вроде бы: надо не менее 32 запросов I/O.

Ссылки на материалы

1. https://learn.microsoft.com/en-us/windows/win32/devnotes/standard-information
2. https://learn.microsoft.com/en-us/windows/win32/devnotes/file-name
3. https://learn.microsoft.com/en-us/windows-server/storage/file-server/volume-shadow-copy-service#can-i-exclude-files-from-a-shadow-copy-to-save-space
4. https://www.iij.ad.jp/en/dev/iir/pdf/iir_vol37_focused1_EN.pdf
5. https://lists.gnu.org/archive/html/info-mtools/2022-09/msg00003.html
6. https://devblogs.microsoft.com/dotnet/asp-net-performance-prefetch-and-multi-core-jitting/
7. https://processhacker.sourceforge.io/doc/ntpfapi_8h_source.html#l00061

CICADA8

Спасибо!
Вопросы?

МАКСИМ СУХАНОВ

Руководитель направления обнаружения киберугроз
и реагирования на инциденты CICADA8

m.suhanov@cicada8.ru