ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Статический анализ кода

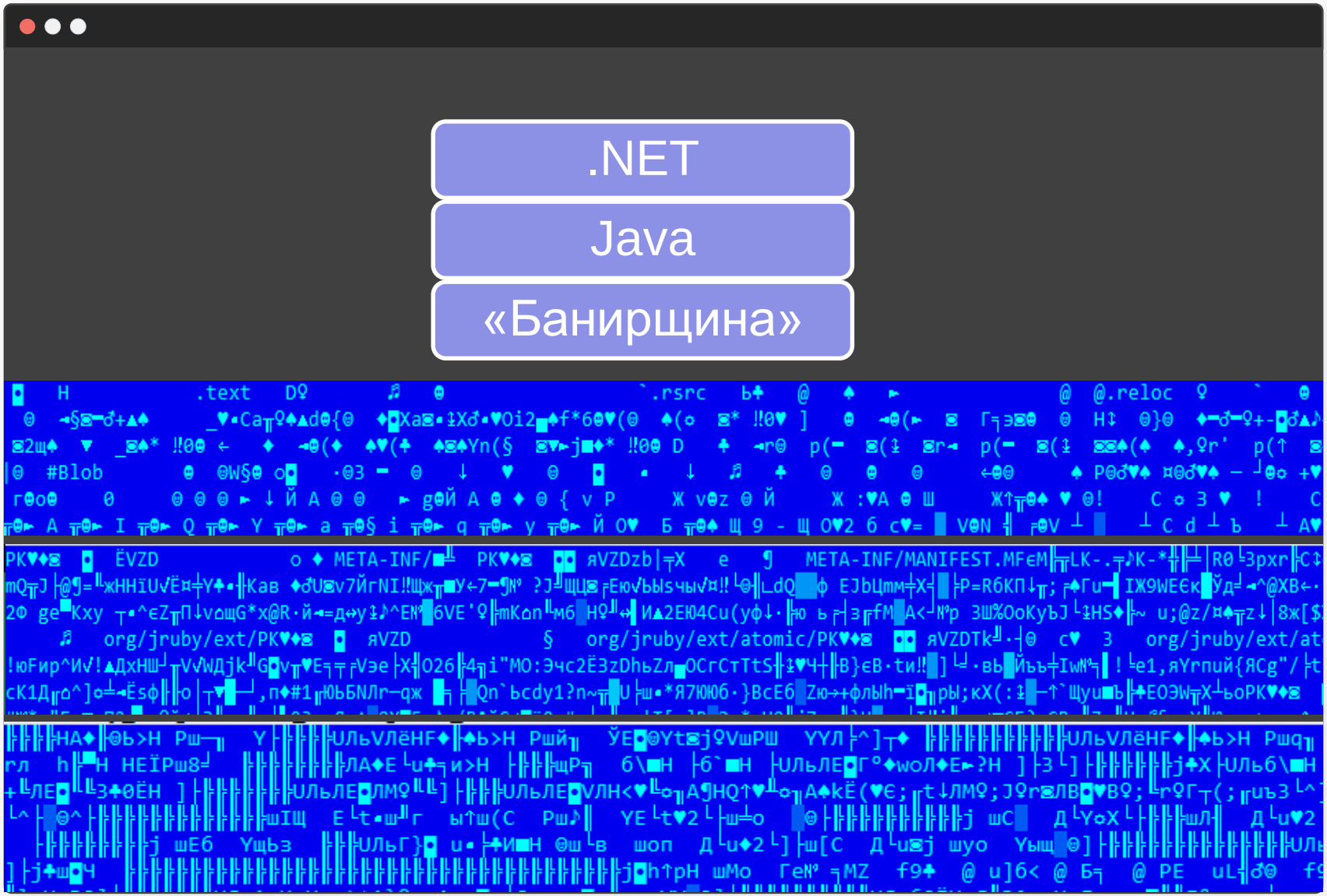
Антистатика, Антидебаг

Пакулов Артур

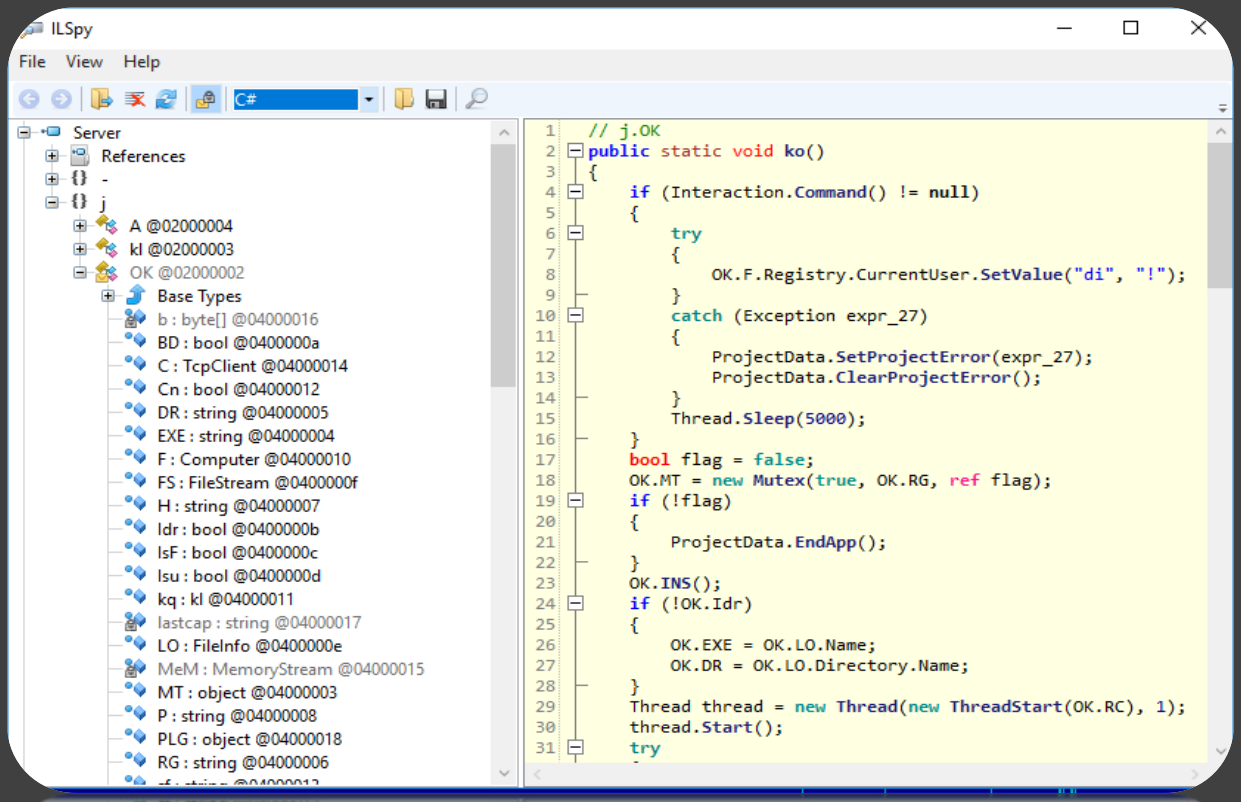
1

Статический анализ кода

Статический анализ

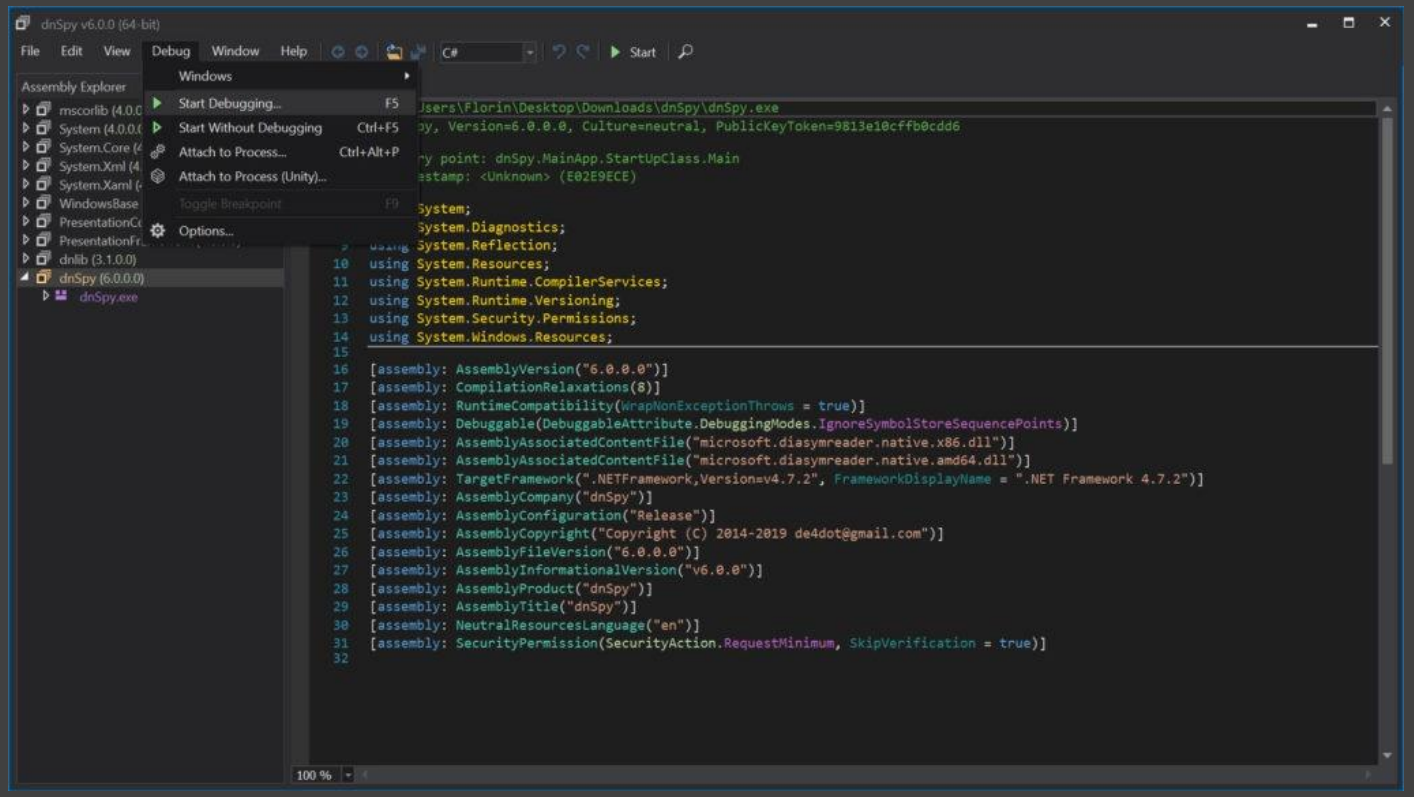


Декомпилятор .NET приложений



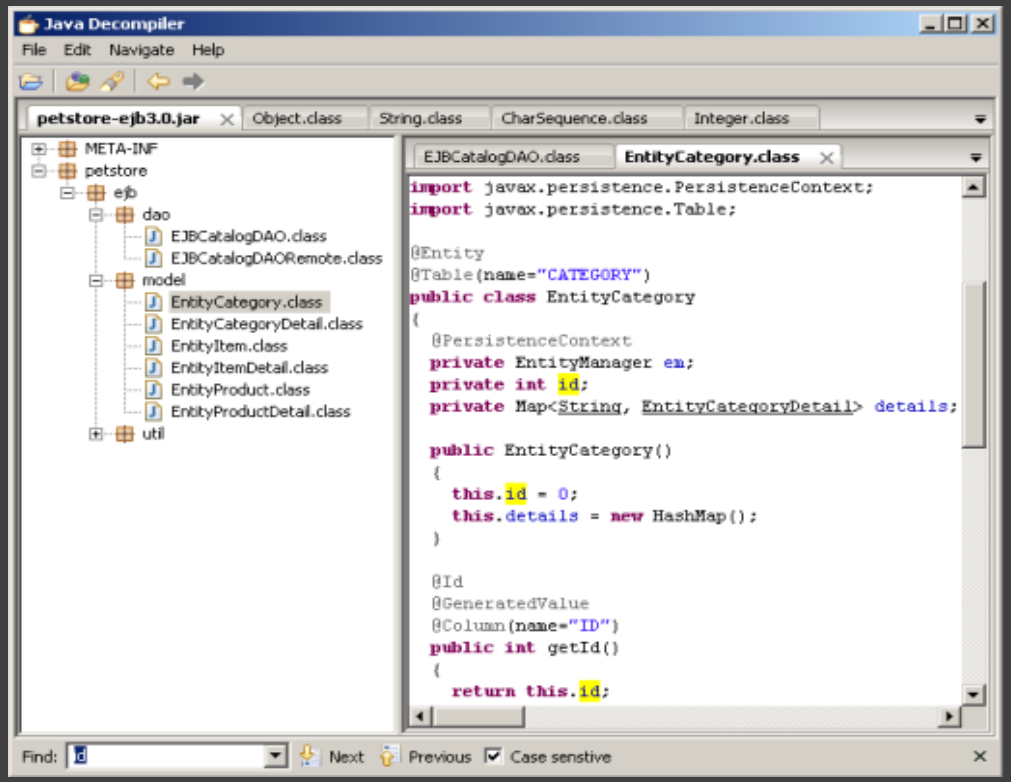
ILSpy/releases

Декомпилятор .NET приложений



dnSpy/releases

Декомпилятор java приложений



github.com/java-decompiler

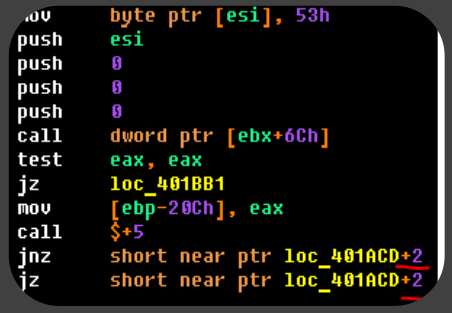
2

Антистатика

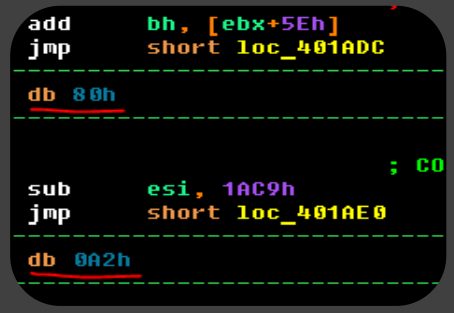
Smoke downloader

```
.text:00401AB6                  test    eax, eax
.text:00401AB8                  jz      loc_401BB1
.text:00401ABE                  mov     [ebp-20Ch], eax
.text:00401AC4                  call    $+5
.text:00401AC9                  jnz     short near ptr loc_401ACD+2
.text:00401ACB                  jz      short near ptr loc_401ACD+2
.text:00401ACD
.text:00401ACD loc_401ACD:                              ; CODE XREF: .text:00401AC9↑j
.text:00401ACD                                          ; .text:00401ACB↑j
.text:00401ACD                  add     bh, [ebx+5Eh]
.text:00401AD0                  jmp     short loc_401ADC
.text:00401AD0 ; ---------------------------------------------------------------------------
.text:00401AD2                  db 80h
.text:00401AD3 ; ---------------------------------------------------------------------------
.text:00401AD3
.text:00401AD3 loc_401AD3:                              ; CODE XREF: .text:loc_401ADC↓j
.text:00401AD3                  sub     esi, 1AC9h
.text:00401AD9                  jmp     short loc_401AE0
.text:00401AD9 ; ---------------------------------------------------------------------------
.text:00401ADB                  db 0A2h
.text:00401ADC ; ---------------------------------------------------------------------------
.text:00401ADC
.text:00401ADC loc_401ADC:                              ; CODE XREF: .text:00401AD0↑j
.text:00401ADC                  jmp     short loc_401AD3
.text:00401ADC ; ---------------------------------------------------------------------------
.text:00401ADE                  dw 0A280h
.text:00401AE0 ; ---------------------------------------------------------------------------
```

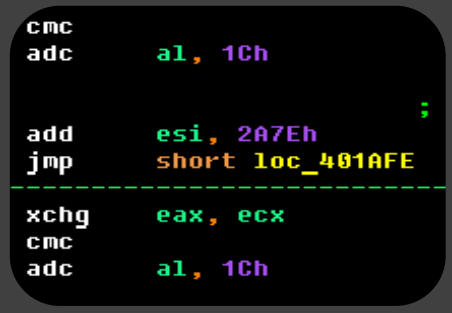
Признаки «антистатики»



Используются смещения относительно меток



Встречаются байты, не входящие в команды



Местами код выглядит резко нелогичным

Будет ли работать такой код?

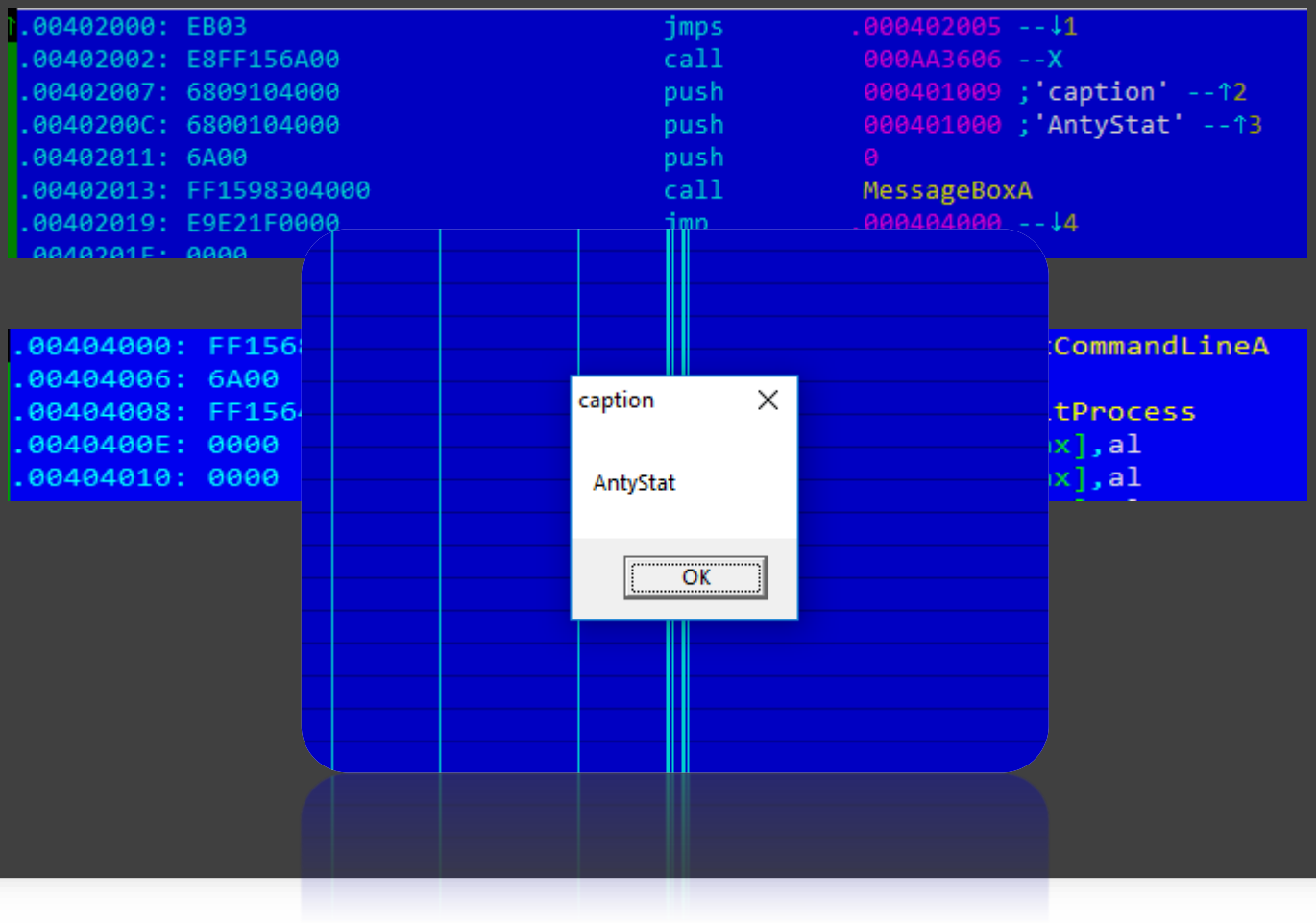
Обратите внимание на команды перехода

```
.00402000: EB03                    jmps        .000402005 --↓1
.00402002: E8FF156A00              call         000AA3606 --X
.00402007: 6809104000              push         000401009 ;'caption' --↑2
.0040200C: 6800104000              push         000401000 ;'AntyStat' --↑3
.00402011: 6A00                    push         0
.00402013: FF1598304000            call         MessageBoxA
.00402019: E9E21F0000              jmp         .000404000 --↓4
.0040201E: 0000                    add          [eax],al
```

```
.00404000: FF1568304000            call        GetCommandLineA
.00404006: 6A00                    push        0
.00404008: FF1564304000            call        ExitProcess
.0040400E: 0000                    add         [eax],al
.00404010: 0000                    add         [eax],al
```

Будет ли работать такой код?

Код работает!



Скомпилируем код

```
format PE GUI
include 'win32ax.inc'
section 'data' readable writeable
        body db "AntyStat",0
        cap db "caption"
.code
start:
        invoke  MessageBox,HWND_DESKTOP, body, cap, MB_OK
        jmp Start2
.end start

section 'code2' readable writeable executable
Start2:
        invoke GetCommandLine
        invoke  ExitProcess,0
```

Выглядит он так

```
.00402000: 6A00                  push        0
.00402002: 6809104000            push        000401009 ;'caption' --↑1
.00402007: 6800104000            push        000401000 ;'AntyStat' --↑2
.0040200C: 6A00                  push        0
.0040200E: FF1598304000          call        MessageBoxA
.00402014: E9E71F0000            jmp         .000404000 --↓3
.00402019: 0000                  add         [eax],al
```

Скомпилируем код

```
format PE GUI
include 'win32ax.inc'
section 'data' readable writeable
        body db "AntyStat",0
        cap db "caption"
.code
start:
        jmp L1
        db 0xE8,01,02,03,04
        db 0xff, 0x15
L1:
        invoke  MessageBox,HWND_DESKTOP,body,cap,MB_OK
.end s
sectio
Start2
        invoke  ExitProcess,0
```

Любые байты, желательно, начинающиеся с опкодов команд переходов

```
E801020304              call        004432208 --X
FF156A006809            call        d,[00968006A]
104000                  adc         [eax][0],al
6800104000              push        000401000 ;'AntyStat' --↑2
6A00                    push        0
FF1598304000            call        MessageBoxA
```

Инструкции перехода с одинаковыми операндами

Вернёмся к «Smoke»

```
7504                jnz         .000401ACF --↓1
7402                jz          .000401ACF --↓1
027B5E              add          bh,[ebx][05E]
EB0A                jmps        .000401ADC --↓2
8081EEC91A0000      add          b,[ecx][0001AC9EE],0
EB05                jmps        .000401AE0 --↓3
A2EBF580A2          mov          [0A280F5EB],al
EB05               3jmps        .000401AE7 --↓4
8888F5141C81        mov          [eax][-07EE3EB0B],cl
C67E                #UD
2A00                sub          al,[eax]
00EB                add          bl,ch
0F91F5              setno        ch
141C                adc          al,01C
B9372C0000         6mov          ecx,000002C37 ;'  ,7'
EB07                jmps        .000401B01 --↓5
91                  xchg         ecx,eax
F5                  cmc
141C                adc          al,01C
EBF3                jmps        .000401AF3 --↑6
80EB0F              sub          bl,00F
7DF4                jge         .000401AF9 --↑7
141C                adc          al,01C
BA60220000         Amov          edx,000002260 ;'  "`'
```

[Jz,Jnz]/[Jnz.Jz] друг за другом ~ jmp

Скрипт «отчистки»

1. Находим нужные команды и запоминаем из смещения
2. Патчим

```
def find_jz_jnz(patterns, data):
    result = []
    value = None
    pattern_num = 0

    for p in patterns:
        N = len(data)
        cur_offset = 0
        while 1==1:
            m = p.search(data[cur_offset:N])
            if m:
                data_fnd = m.group(0)
                rva_1 = m.group(1)
                rva_2 = m.group(2)

                cur_offset = cur_offset + m.start()
                if abs(ord(rva_2)-ord(rva_1)) == 2:
                    Debug("offset: 0x%08x, sigs: %s, rva1: %s, rva2: %s\n" % (cur_offset, data_fnd.encode('hex'), rva_1.
                    encode('hex'), rva_2.encode('hex')))
                    result.append(cur_offset)
                cur_offset = cur_offset + (m.end() - m.start())
            else:
                break
    return result
```

Скрипт «отчистки»

1. Находим нужные команды и запоминаем из смещения
2. Патчим

```
def nop_obfuscation(offsets, data):
    patched = bytearray(data)
    prev = 0

    for offset in offsets:
        patched[offset] = 0xeb      #jmps xx
    return patched
```

Поиск по регулярному выражению

```
patterns = [
    re.compile(b""
               "\x74([\x00-\xFF])"
               "\x75([\x00-\xFF])", re.DOTALL),

    re.compile(b""
               "\x75([\x00-\xFF])"
               "\x74([\x00-\xFF])", re.DOTALL)
]
```

Скрипт «отчистки»

OTUS

Главная функция

```
if __name__ == '__main__':
    buf = ReadFile(r'D:\Reverse\Bots\Smoke\exe\unpack.rxr_')
    jz_jnz_offsets = find_jz_jnz(patterns, buf)
    patched_buf = nop_obfuscation(jz_jnz_offsets, buf)
    sf('patche.bin', patched_buf, True)
```

Linear sweep

- Семантический анализ: где заканчивается первая инструкция, там сразу начинается вторая
- Все что передано анализатору - код
- Анализ производится линейно начиная с первого байта
- Инструкции декодируются последовательно до конца переданного набора байт

Плюсы:

Покрывается вся секция кода

Минусы:

Не предпринимается попыток «понять» граф потока управления

Невозможно отличить код от данных

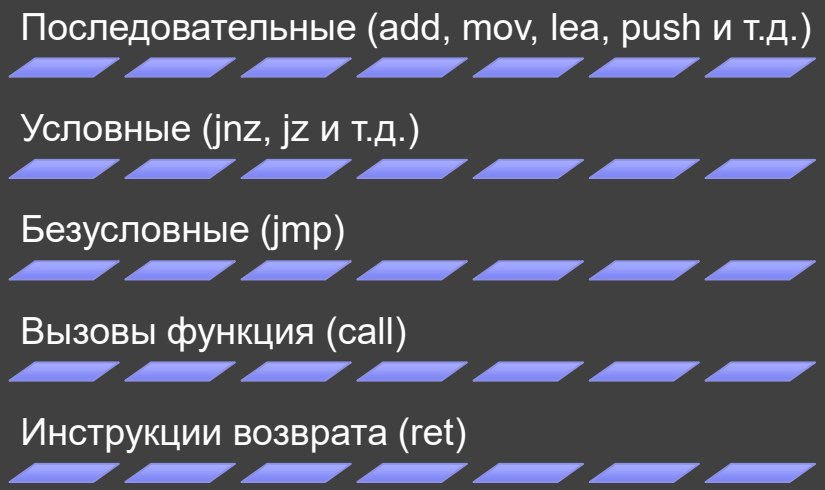
8B 45 D4 EB 04 8B 03 05 08 30 40 00 E8 72 71 F3 FF C3

Address	Hex dump	Command
004C9E7D	8B45 D4	MOV EAX, [EBP-2C]
004C9E80	EB 04	JMP SHORT 004C9E86
004C9E82	8B03	MOV EAX, [EBX]
004C9E84	0508 304000	ADD EAX, 00403008
004C9E89	E8 7271F3FF	CALL 00401000
004C9E8E	C3	RETN

8B 45 D4 EB 04 8B 03 05 08 30 40 00 E8 72 71 F3 FF C3

Address	Hex dump	Command
004C9E7D	8B45 D4	MOV EAX, [EBP-2C]
004C9E80	EB 04	JMP SHORT 004C9E86
004C9E82	8B030508	DD 0805038B // mycop
004C9E86	3040 00	XOR [EAX], AL
004C9E89	E8 7271F3FF	CALL 00401000
004C9E8E	C3	RETN

Основан на концепции графа потока выполнения
Плюсы:
Отделение кода от данных
Минусы:
Сложность автоматического анализа непрямых (неявных) переходов
Все инструкции разделяются на:



3

Антидинамика

WinAPI

Функция	Описание способа
IsDebuggerPresent	Проверка флага PEB в текущем процессе
CheckRemoteDebugger Present	Проверка флага PEB в любом процессе
NtSetInformationThread / ZwSetInformationThread	Вызываются, чтобы скрыть нужный поток от отладчика
NtQueryObject	Получить хендль объекта DEBUG
OutputDebugString	При неудачнов выполнении, GetLastError выставит код ошибки
CsrGetProcessID	Вернёт хендл, если у процесса есть SeDebugPrivilege (отладчики включают такие привилегии)
CloseHandle / NtClose	Когда процесс под отладкой, вызов данной функции с невалидным дескриптором приведёт к исключению

Ручная проверка

```
0:001> dt _PEB
ntdll!_PEB
   +0x000 InheritedAddressSpace : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged    : UChar
   +0x003 BitField         : UChar
   +0x003 ImageUsesLargePages : Pos 0, 1 Bit
   +0x003 IsProtectedProcess : Pos 1, 1 Bit
   +0x003 IsImageDynamicallyRelocated : Pos 2, 1 Bit
   +0x003 SkipPatchingUser32Forwarders : Pos 3, 1 Bit
   +0x003 IsPackagedProcess : Pos 4, 1 Bit
   +0x003 IsAppContainer   : Pos 5, 1 Bit
   +0x003 IsProtectedProcessLight : Pos 6, 1 Bit
   +0x003 IsLongPathAwareProcess : Pos 7, 1 Bit
   +0x004 Mutant           : Ptr32 Void
   +0x008 ImageBaseAddress : Ptr32 Void
   +0x00c Ldr              : Ptr32 _PEB_LDR_DATA
   +0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData    : Ptr32 Void
   +0x018 ProcessHeap      : Ptr32 Void
   +0x01c FastPebLock      : Ptr32 _RTL_CRITICAL_SECTION
   +0x020 AtlThunkSListPtr : Ptr32 _SLIST_HEADER
   +0x024 IFEOKey          : Ptr32 Void
   +0x028 CrossProcessFlags : Uint4B
   +0x028 ProcessInJob     : Pos 0, 1 Bit
   +0x028 ProcessInitializing : Pos 1, 1 Bit
   +0x028 ProcessUsingVEH  : Pos 2, 1 Bit
   +0x028 ProcessUsingVCH  : Pos 3, 1 Bit
   +0x028 ProcessUsingFTH  : Pos 4, 1 Bit
   +0x028 ProcessPreviouslyThrottled : Pos 5, 1 Bit
   +0x028 ProcessCurrentlyThrottled : Pos 6, 1 Bit
   +0x028 ReservedBits0    : Pos 7, 25 Bits
   +0x02c KernelCallbackTable : Ptr32 Void
   +0x02c UserSharedInfoPtr : Ptr32 Void
   +0x030 SystemReserved   : Uint4B
   +0x034 AtlThunkSListPtr32 : Ptr32 _SLIST_HEADER
   +0x038 ApiSetMap        : Ptr32 Void
   +0x03c TlsExpansionCounter : Uint4B
   +0x040 TlsBitmap        : Ptr32 Void
   +0x044 TlsBitmapBits    : [2] Uint4B
   +0x04c ReadOnlySharedMemoryBase : Ptr32 Void
   +0x050 SharedData       : Ptr32 Void
   +0x054 ReadOnlyStaticServerData : Ptr32 Ptr32 Void
   +0x058 AnsiCodePageData : Ptr32 Void
   +0x05c OemCodePageData  : Ptr32 Void
   +0x060 UnicodeCaseTableData : Ptr32 Void
   +0x064 NumberOfProcessors : Uint4B
   +0x068 NtGlobalFlag     : Uint4B
   +0x070 CriticalSectionTimeout : _LARGE_INTEGER
   +0x078 HeapSegmentReserve : Uint4B
   +0x07c HeapSegmentCommit : Uint4B
   +0x080 HeapDeCommitTotalFreeThreshold : Uint4B
   +0x084 HeapDeCommitFreeBlockThreshold : Uint4B
```

Ручная проверка

Функция	Описание способа
IsDebugged Flag	Проверка флага PEB в текущем процессе. Если флаг установлен, то процесс под отладкой
NtGlobalFlag	Если флаг == 0x68, то процесс под отладкой
Heap Flag	ядро определяет, создана ли куча в отладчике

```
mov eax, dword ptr fs:[30h]
mov ebx, byte ptr [eax+2]
test ebx, ebx
jz NoDebuggerDetected
```

```
push dword ptr fs:[30h]
pop edx
cmp byte ptr [edx+2], 1
je DebuggerDetected
```

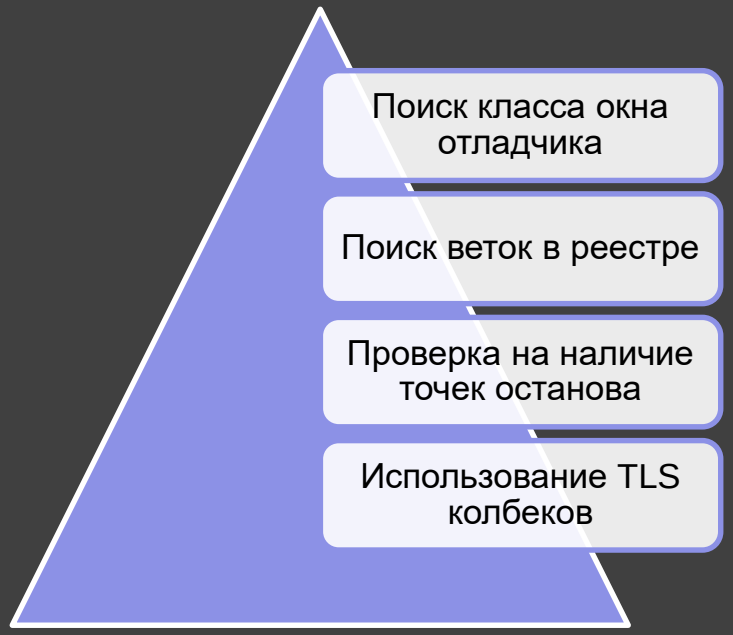
Замеры времени

- RDTSC
- API-функция GetTickCount;
- API-функция timeGetTime (из winmm.dll);
- API-функция QueryPerformanceCounter;
- API-функция GetSystemTimeAsFileTime;
- API-функция GetProcessTimes;
- API-функция KiGetTickCount (или вызов прерывания int 0x2A);
- API-функция NtQueryInformationProcess (ProcessInformationClass = ProcessTimes (0x04);
- API-функция NtQueryInformationThread (ThreadInformationClass = ThreadTimes (0x01);
- поля структуры KUSER_SHARED_DATA.

Пример

```
a = GetTickCount();
MaliciousActivityFunction();
b = GetTickCount();
delta = b-a;
if ((delta) > 0x1A)
{
	// Отладчик обнаружен
}
else
{
	// Отладчик не найден
}
```

Поиск артефактов



anti-reverse-engineering-protection-techniques

OTUS

Вопросы???

Пакулов Артур

A.Pakulov.Otus@Gmail.com

Спасибо
за внимание!