



ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно  
&& видно?



Напишите в чат, если есть проблемы!

Ставьте  если все хорошо

# Статический анализ кода

Антистатика, Антидебаг

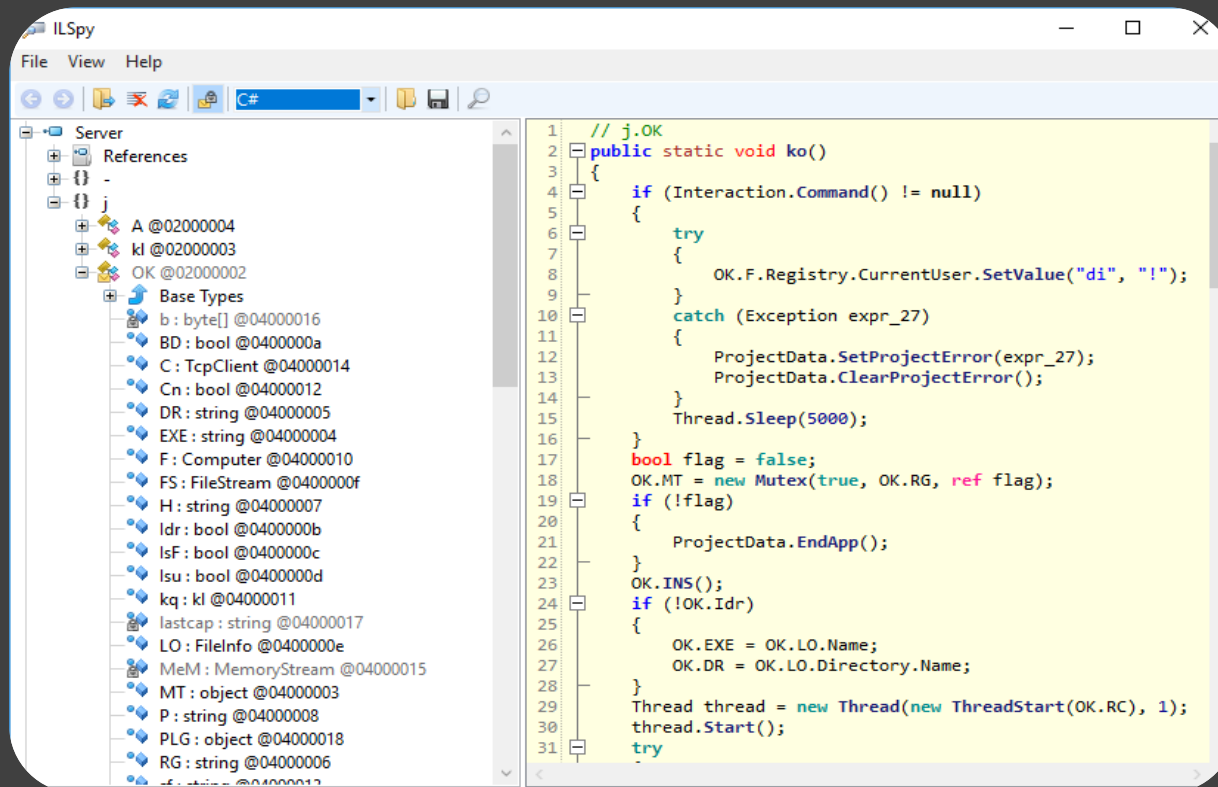


1

# Статический анализ кода

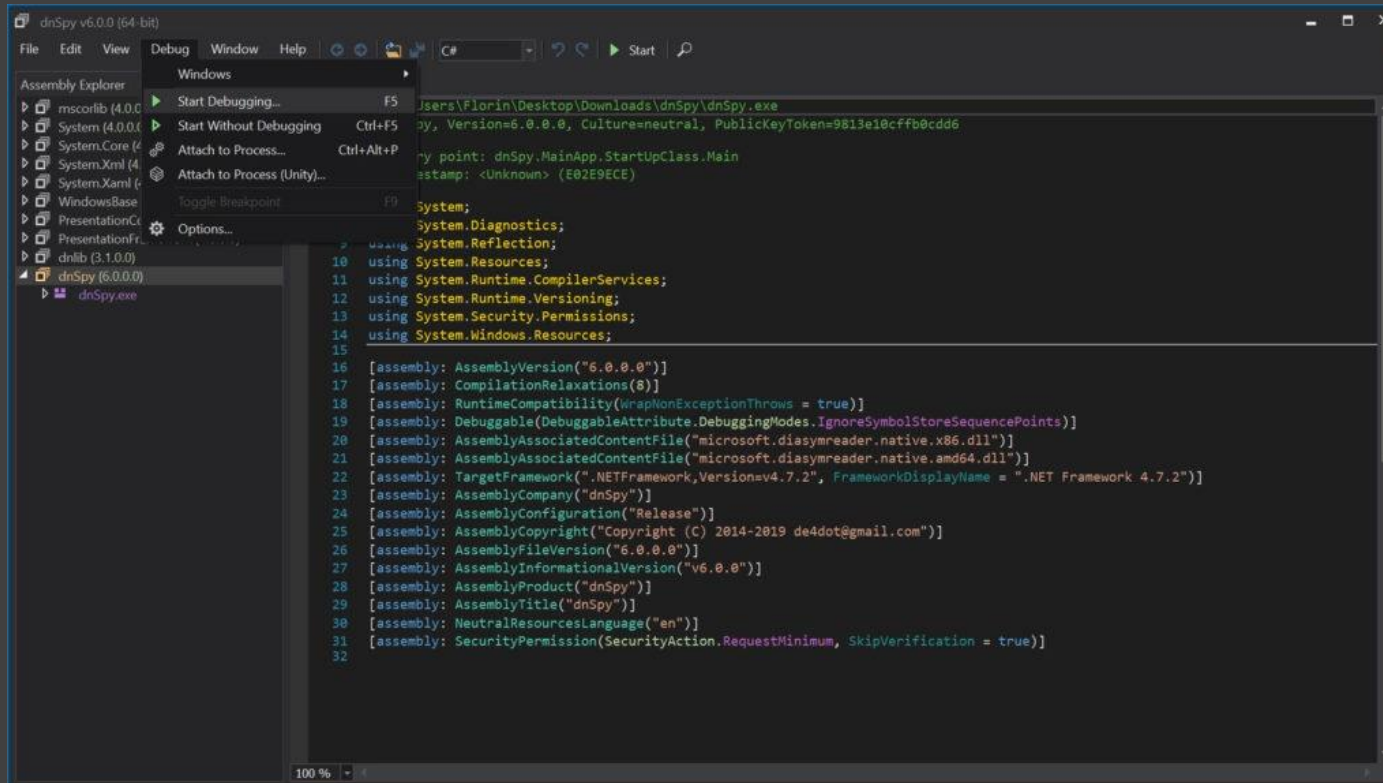


# Декомпилятор .NET приложений



```
1 // j.OK
2 public static void ko()
3 {
4     if (Interaction.Command() != null)
5     {
6         try
7         {
8             OK.F.Registry.CurrentUser.SetValue("di", "!");
9         }
10        catch (Exception expr_27)
11        {
12            ProjectData.SetProjectError(expr_27);
13            ProjectData.ClearProjectError();
14        }
15        Thread.Sleep(5000);
16    }
17    bool flag = false;
18    OK.MT = new Mutex(true, OK.RG, ref flag);
19    if (!flag)
20    {
21        ProjectData.EndApp();
22    }
23    OK.INS();
24    if (!OK.Idr)
25    {
26        OK.EXE = OK.LO.Name;
27        OK.DR = OK.LO.Directory.Name;
28    }
29    Thread thread = new Thread(new ThreadStart(OK.RC), 1);
30    thread.Start();
31    try
```

## Декомпилятор .NET приложений



```
File Edit View Debug Window Help
Windows
Start Debugging... F5
Start Without Debugging Ctrl+F5
Attach to Process... Ctrl+Alt+P
Attach to Process (Unity)...
Toggle Breakpoint F9
Options...
Assembly Explorer
mscorlib (4.0.0)
System (4.0.0)
System.Core (4.0.0)
System.Xml (4.0.0)
System.Xml.Linq (4.0.0)
WindowsBase (4.0.0)
PresentationCore (4.0.0)
PresentationFramework (4.0.0)
dnlb (3.1.0.0)
dnSpy (6.0.0.0)
  dnSpy.exe
J:\Users\Florin\Desktop\Downloads\dnSpy\dnSpy.exe
System.Reflection.Emit, Version=6.0.0.0, Culture=neutral, PublicKeyToken=9813e10cfff0cdd6
System.Diagnostics, Version=6.0.0.0, Culture=neutral, PublicKeyToken=9813e10cfff0cdd6
MainApp: dnSpy.MainApp.StartupClass.Main
Assembly: dnSpy, Version=6.0.0.0, Culture=neutral, PublicKeyToken=9813e10cfff0cdd6
AssemblyName: dnSpy
Assembly: dnSpy, Version=6.0.0.0, Culture=neutral, PublicKeyToken=9813e10cfff0cdd6
System;
System.Diagnostics;
System.Reflection;
System.Resources;
using System.Resources;
using System.Runtime.CompilerServices;
using System.Runtime.Versioning;
using System.Security.Permissions;
using System.Windows.Resources;
[assembly: AssemblyVersion("6.0.0.0")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: AssemblyAssociatedContentFile("microsoft.diasymreader.native.x86.dll")]
[assembly: AssemblyAssociatedContentFile("microsoft.diasymreader.native.amd64.dll")]
[assembly: TargetFramework(".NETFramework,Version=v4.7.2", FrameworkDisplayName = ".NET Framework 4.7.2")]
[assembly: AssemblyCompany("dnSpy")]
[assembly: AssemblyConfiguration("Release")]
[assembly: AssemblyCopyright("Copyright (C) 2014-2019 de4dot@gmail.com")]
[assembly: AssemblyFileVersion("6.0.0.0")]
[assembly: AssemblyInformationalVersion("v6.0.0")]
[assembly: AssemblyProduct("dnSpy")]
[assembly: AssemblyTitle("dnSpy")]
[assembly: NeutralResourcesLanguage("en")]
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerification = true)]
100 %
```

## Декомпилятор java приложений

The screenshot shows the Java Decompiler application window. The title bar reads "Java Decompiler". The menu bar includes "File", "Edit", "Navigate", and "Help". The toolbar contains icons for file operations. The main window is divided into two panes. The left pane shows a project tree for "petstore-ejb3.0.jar" with folders for "META-INF", "petstore", "ejb", "dao", "model", and "util". The "model" folder is expanded, showing several class files, with "EntityCategory.class" selected. The right pane displays the decompiled Java code for "EntityCategory.class". The code includes imports for "javax.persistence.PersistenceContext" and "javax.persistence.Table". It defines an "Entity" with a table name "CATEGORY". The class "EntityCategory" has a constructor that initializes "id" to 0 and "details" to a new "HashMap". It also has a "getId()" method that returns "this.id".

```
import javax.persistence.PersistenceContext;
import javax.persistence.Table;

@Entity
@Table(name="CATEGORY")
public class EntityCategory
{
    @PersistenceContext
    private EntityManager em;
    private int id;
    private Map<String, EntityCategoryDetail> details;

    public EntityCategory()
    {
        this.id = 0;
        this.details = new HashMap();
    }

    @Id
    @GeneratedValue
    @Column(name="ID")
    public int getId()
    {
        return this.id;
    }
}
```

[github.com/java-decompiler](https://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  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  loc_401ADC:                ; CODE XREF: .text:00401AD0↑j
.text:00401ADC      jmp     short loc_401AD3
.text:00401ADC      ; -----
.text:00401ADE      dw     0A280h
.text:00401AE0      ; -----
```

## Признаки «антистатике»

```
mov     byte ptr [esi], 53h
push   esi
push   0
push   0
push   0
call   dword ptr [ebx+6Ch]
test   eax, eax
jz     loc_401B81
mov    [ebp-20Ch], eax
call   $+5
jnz   short near ptr loc_401ACD+2
jz    short near ptr loc_401ACD+2
```

Используются  
смещения  
относительно меток

```
add     bh, [ebx+5Eh]
jmp     short loc_401ADC
-----
db     80h
-----
sub     esi, 1AC9h ; C0
jmp     short loc_401AE0
-----
db     0A2h
```

Встречаются байты,  
не входящие в  
команды

```
cmc
adc     al, 1Ch
-----
add     esi, 2A7Eh ;
jmp     short loc_401AFE
-----
xchg   eax, ecx
cmc
adc     al, 1Ch
```

Местами код  
выглядит резко  
нелогичным

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
.0040201F: 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
```

Код работает!

```
.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
```

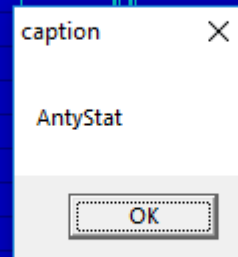
```
.00404000: FF156A00
.00404006: 6A00
.00404008: FF156A00
.0040400E: 0000
.00404010: 0000
```

CommandLineA

tProcess

x],al

x],al



## 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
.0040201B: 0000          add     [ecx], 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
```



*Любые байты, желательно,  
начинающиеся с опкодов команд  
переходов*

```
L1:
```

```
    invoke MessageBox,HWND_DESKTOP,body,cap,MB_OK
```

```
    invoke Start2
```

```
.end s
```

```
section
```

```
Start2
```

E801020304	call	004432208 --X
FF156A006809	call	d,[00968006A]
104000	adc	[eax][0],al
6800104000	push	000401000 ;'AntyStat' --12
6A00	push	0
FF1598304000	call	MessageBoxA

```
    invoke ExitProcess,0
```

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         jmps    .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   mov     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

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
```

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

```
def nop_obfuscation(offsets, data):  
    patched = bytearray(data)  
    prev = 0  
  
    for offset in offsets:  
        patched[offset] = 0xeb          #jumps 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)  
]
```

## 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)
```

- 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 не прямых (неявных) переходов

Все инструкции разделяются на:

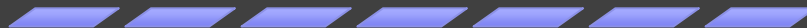
Последовательные (add, mov, lea, push и т.д.)



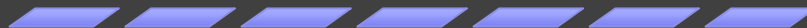
Условные (jnz, jz и т.д.)



Безусловные (jmp)



Вызовы функция (call)



Инструкции возврата (ret)



3

# Антидинамика

## WinAPI

Функция	Описание способа
IsDebuggerPresent	Проверка флага PEB в текущем процессе
CheckRemoteDebuggerPresent	Проверка флага PEB в любом процессе
NtSetInformationThread / ZwSetInformationThread	Вызываются, чтобы скрыть нужный поток от отладчика
NtQueryObject	Получить хендль объекта DEBUG
OutputDebugString	При неудачном 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 AtlThunkListPtr : Ptr32 _SLIST_HEADER
+0x024 IFEKey : 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 AtlThunkListPtr32 : 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 PEВ 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
    
```

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);

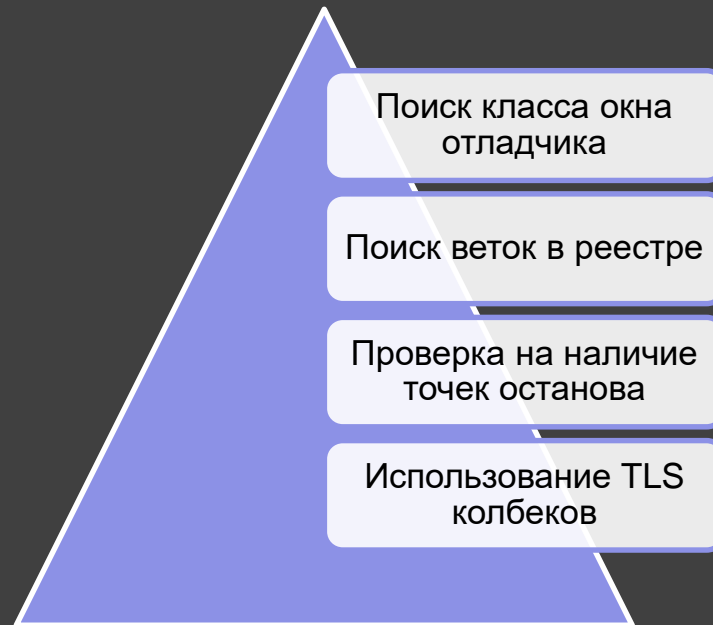
API-функция NtQueryInformationThread (ThreadInformationClass = ThreadTimes (0x01);

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](#)

O T U S

Вопросы???





Пакулов Артур

[A.Pakulov.Otus@Gmail.com](mailto:A.Pakulov.Otus@Gmail.com)

Спасибо  
за внимание!

