Антиотладочные техники

Игорь Черватюк

Александр Трифанов

Андрей Басарыгин

Москва, 2018

Понятие отладчика

Отладчик – программа, предназначенная для поиска ошибок в других программах.

Отладчики можно поделить на два типа – уровня исходного кода (source-level debugger) и низкого уровня (low-level debugger).

Можно поделить на два других типа – отладчики уровня приложений и отладчики уровня ядра.

Основные функции – возможность выполнять код пошагово, устанавливать точки останова (breakpoints), менять значения переменных «на лету».

Windows Debugging API

В операционной системе семейства Windows для нативного процесса отладки существует специальный набор API-функций (https://msdn.microsoft.com/en-us/library/ms809754.aspx).

Для того, чтобы начать отладку – отладчику нужно запустить отлаживаемый процесс специальным образом или присоединиться к существующему процессу.

После того, как это происходит – отладчик «привязывается к процессу» и устанавливается зависимость: parent process и child process. Если отладчик завершает работу, то и отлаживаемый процесс тоже завершается.

Windows Debugging API: CreateProcess

```
BOOL CreateProcess(
   LPCTSTR lpszImageName,        /* address of image file name */
   LPCTSTR lpszCommandLine,      /* address of the command line */
   LPSECURITY_ATTRIBUTES lpsaProcess,  /* optional process attrs */
   LPSECURITY_ATTRIBUTES lpsaThread,   /* optional thread attrs */
   BOOL fInheritHandles,         /* new process inherits handles? */
   DWORD fdwCreate,                          /* creation flags */
   LPVOID lpvEnvironment,    /* address of optional environment */
   LPTSTR lpszCurDir,        /* address of new current directory */
   LPSTARTUPINFO lpsi,                /* address of STARTUPINFO */
   LPPROCESS_INFORMATION lppi);  /* address of PROCESSINFORMATION */
```

Параметр fdwCreate должен быть установлен в значение DEBUG_PROCESS или DEBUG_ONLY_THIS_PROCESS.

Windows Debugging API: DebugActiveProcess

```
BOOL WINAPI DebugActiveProcess(
  _In_ DWORD dwProcessId
);
```

В качестве параметра для функции используется идентификатор процесса.

Windows Debugging API: Цикл отладки

После того, как отладчик присоединился к какому-либо процессу – отладчик запускает цикл в котором он «слушает» систему на предмет возникновения событий отладки – Debug Event.

Основные такие события перечислены ниже:

- CREATE_PROCESS_DEBUG_EVENT \ EXIT_PROCESS_DEBUG_EVENT
- CREATE_THREAD_DEBUG_EVENT \ EXIT_THREAD_DEBUG_EVENT
- LOAD_DLL_DEBUG_EVENT \ UNLOAD_DLL_DEBUG_EVENT
- EXCEPTION_DEBUG_EVENT
- OUTPUT_DEBUG_STRING_DEBUG_EVENT

Windows Debugging API: Обработка событий

Основные функции для обработки событий:

WaitForDebugEvent()

ContinueDebugEvent()

Функции для изменения контекста:

GetThreadContext() \ Wow64GetThreadContext()

SetThreadContext() \ Wow64SetThreadContext()

Все остальные можно посмотреть здесь:

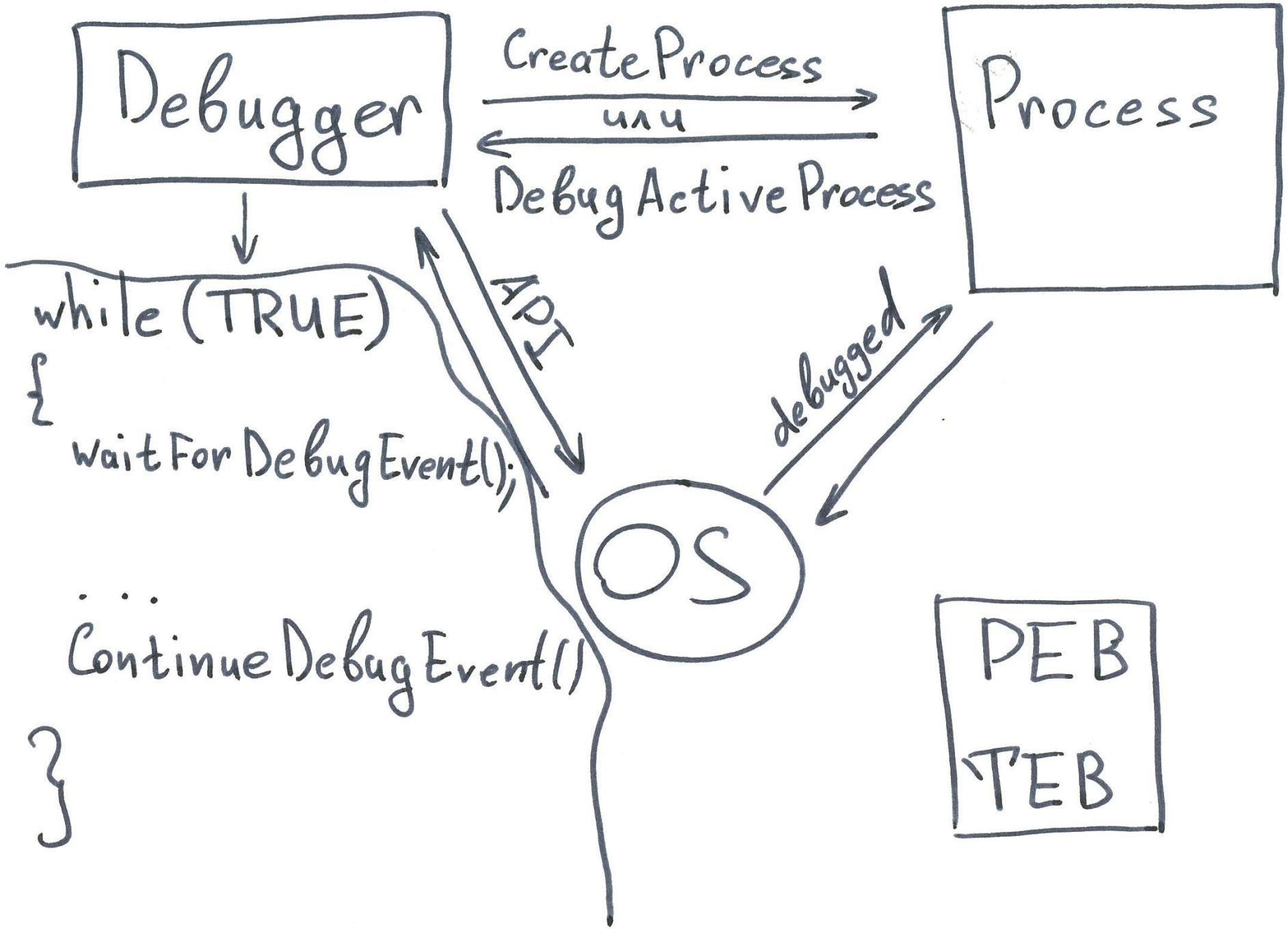
https://msdn.microsoft.com/en-us/library/windows/desktop/ms679303(v=vs.85).aspx

```
while (TRUE)
    {
    /* Wait for 1/10 second for a debug event. */
    if (WaitForDebugEvent (&de, (DWORD)100))
        {
        switch (de.dwDebugEventCode)
            {
            case EXCEPTION_DEBUG_EVENT:
                ProcessExceptionEvent (&de);
                break;

            case CREATE_PROCESS_DEBUG_EVENT:
                ProcessCreateProcessEvent (&de);
                break;

                .
                .
                .

            case default:
                ProcessUnknownDebugEvent (&de);
                break;
            }
        ContinueDebugEvent (de.dwProcessId,
                            de.dwThreadId,
                            DBG_CONTINUE);
        }
    else
        /* Perform periodic debugger responsibilities. */
    }
```


Debugger
CreateProcess
или
DebugActiveProcess
Process
while (TRUE)
{
WaitForDebugEvent();
...
ContinueDebugEvent()
}
API
debugged
OS
PEB
TEB

PEB и TEB

Process Environment Block

Структура процесса, содержащая информацию об окружении, параметрах запуска, базовому адресу. Более подробно:

https://www.aldeid.com/wiki/PEB-Process-Environment-Block

Thread Environment Block

Структура, содержащая информацию о потоке в текущем процессе. Каждый поток имеет свой отдельный TEB. Более подробно:

https://www.aldeid.com/wiki/TEB-Thread-Environment-Block

Почему ссылка не на MSDN?

Anti-Debugging

Обнаружение отладчика в Windows средах

Проверка на наличие точек останова

Обнаружение программных точек останова.

Программа заранее знает сколько у нее байт 0xCC на определенном участке памяти.

```
bool CheckForCCBreakpoint(void* pMemory,  size_t SizeToCheck)
{
    unsigned char *pTmp = (unsigned char*)pMemory;
    for (size_t i = 0; i < SizeToCheck; i++)
    {
        if(pTmp[i] == 0xCC)
            return true;
    }

    return false;
}
```

Проверка на наличие точек останова

Обнаружение аппаратных точек останова.

```
int CheckHardwareBreakpoints()
{
    unsigned int NumBps = 0;

    CONTEXT ctx;
    ZeroMemory(&ctx, sizeof(CONTEXT));

    ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS;

    HANDLE hThread = GetCurrentThread();

    if(GetThreadContext(hThread, &ctx) == 0)
        return -1;

    if(ctx.Dr0 != 0)
        ++NumBps;
    if(ctx.Dr1 != 0)
        ++NumBps;
    if(ctx.Dr2 != 0)
        ++NumBps;
    if(ctx.Dr3 != 0)
        ++NumBps;

    return NumBps;
}
```

Проверка на наличие точек останова

Обнаружение Guard pages.

- Без кода. Ссылки на примеры дальше в презентации.
- Создается страница памяти PAGE_GUARD и происходит обращение к ней.
- У отладчика есть права на обращение, у обычного процесса – нет.
- Если есть STATUS_GUARD_PAGE_VIOLATION – значит нет отладчика.

Вызовы API

IsDebuggerPresent

CheckRemoteDebuggerPresent

FindWindow

OutputDebugString

Вызовы API: IsDebuggerPresent

```
int main()
{
    if (IsDebuggerPresent() == 0)
    {
        puts("No Debugger");
    } else {
        puts("Debugged");
    }

    return 0;
}
```

```
bool fs_chk(VOID)
{
    char IsDbgPresent = 0;

    __asm {
        mov eax, fs:[30h]
            mov al, [eax + 2h]
            mov IsDbgPresent, al
    }
    if (IsDbgPresent)
    {
        return TRUE;
    }
    return FALSE;
}
```

Вызовы API: CheckRemoteDebuggerPresent

Same stuff, different place.

Вызывает NtQueryInformationProcess и проверяет, что DebugPort не равен 0.

Вызовы API: FindWindow

```
bool wdw_class()
{
    HWND hOlly = FindWindow(_T("OLLYDBG"), NULL);
    HWND hIDA = FindWindow(_T("QWidget"), NULL);
    HWND hWdbg = FindWindow(_T("WinDbgFrameClass"), NULL);
    HWND hImm = FindWindow(_T("ID"), NULL);

    if ((hOlly) || (hIDA) || (hWdbg) || (hImm))
    {
        return TRUE;
    }

    return FALSE;
}
```

Вызовы API: OutputDebugString

Функция более не поддерживается. Пишет сообщение в отладчик. Если отладчика нет – выдает ошибку.

```
DWORD errorValue = 1234;
SetLastError(errorValue);
OutputDebugString("somestring");
if(GetLastError() == errorValue)
{
    [...SNIP...]
}
else
{
    [...SNIP...]
}
```

Старые версии OllyDbg некорректно обрабатывают данную функцию.

```
OutputDebugString("%s%s%s%s%s%s%s%s%s%s");
```

Тайминг

GetTickCount

GetLocalTime

GetSystemTime

timeGetTime

NtQueryPerformanceCounter

rdtsc (Read Time Stamp Counter)

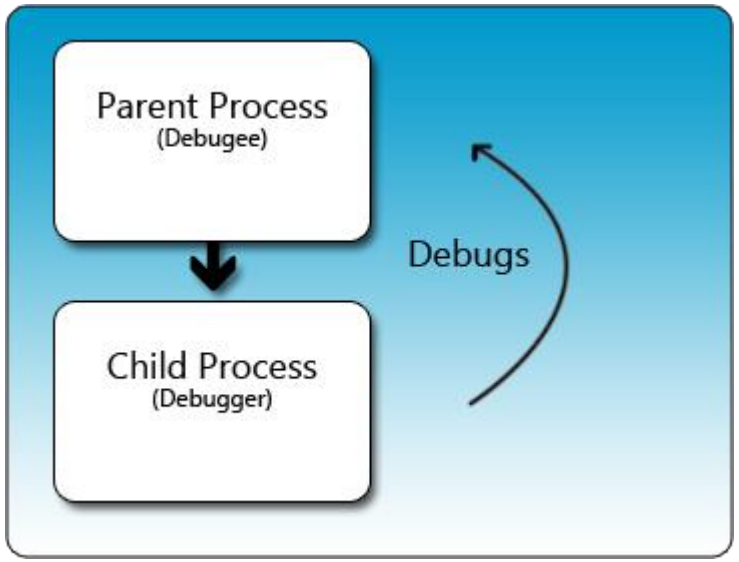
```
DWORD Counter = GetTickCount();
if (Counter < 0xFFFFF)
{
    exit(1);
}
```

```
unsigned int time1 = 0;
unsigned int time2 = 0;
__asm
{
    RDTSC
    MOV time1,EAX
    RDTSC
    MOV time2, EAX

}
if ((time2 - time1) > 100)
{
    exit(1);
}
```

Self-debugging

Программа сформирована таким образом, что она является отладчиком для самой себя.




```
void DebugSelf()
{
HANDLE hProcess = NULL;
DEBUG_EVENT de;
PROCESS_INFORMATION pi;
STARTUPINFO si;
ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
ZeroMemory(&si, sizeof(STARTUPINFO));
ZeroMemory(&de, sizeof(DEBUG_EVENT));

GetStartupInfo(&si);

CreateProcess(NULL, GetCommandLine(), NULL, NULL, FALSE,
        DEBUG_PROCESS, NULL, NULL, &si, &pi);

ContinueDebugEvent(pi.dwProcessId, pi.dwThreadId, DBG_CONTINUE);

WaitForDebugEvent(&de, INFINITE);
}
```

Флаги

Trap flag

IsDebugged

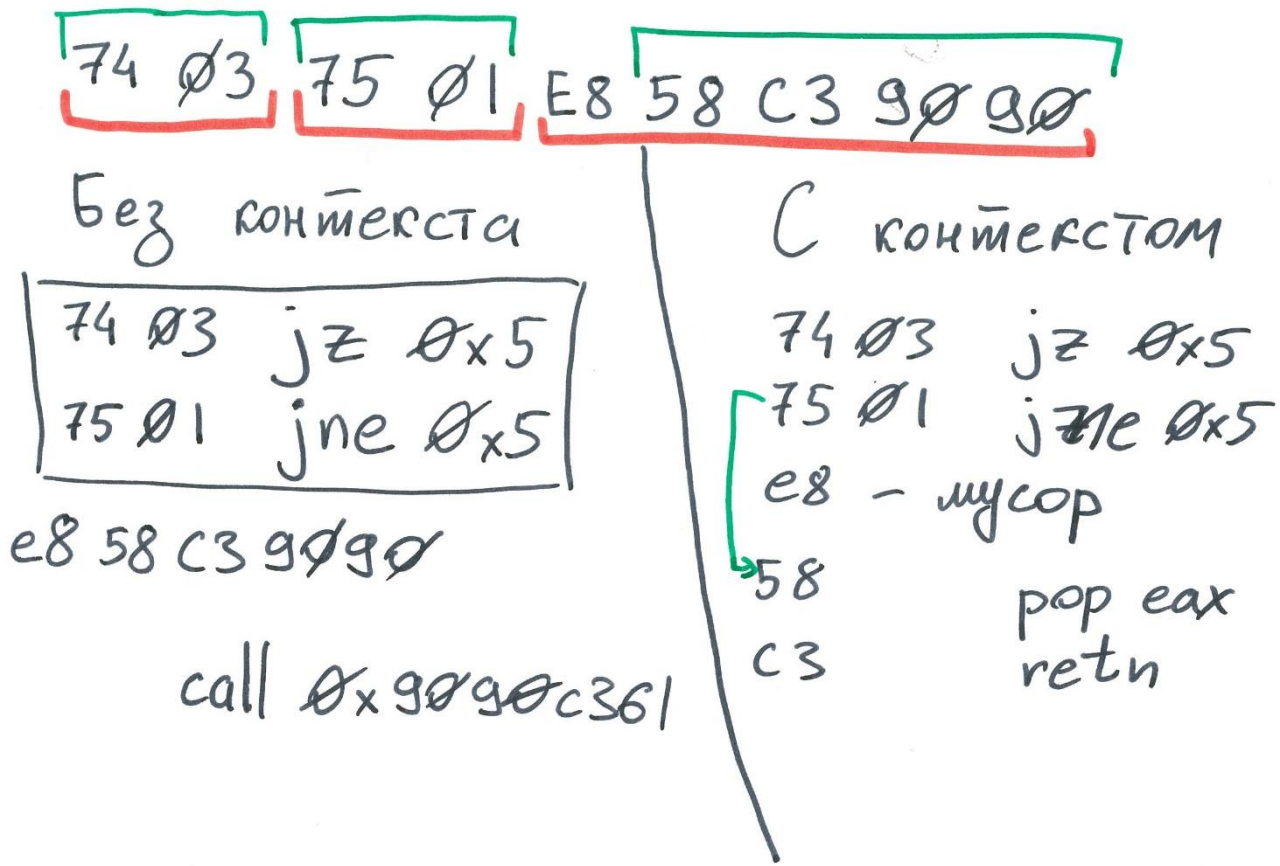
NtGlobalFlag

Heap flags

Anti-static (analysis)

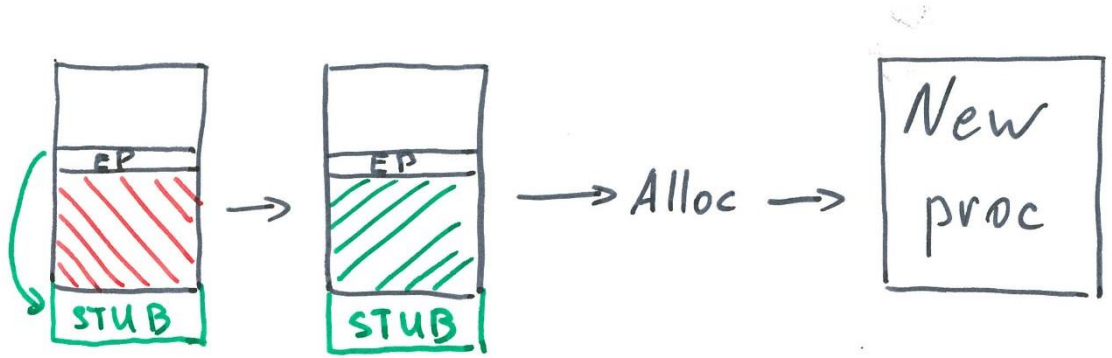
Затруднение анализа инструкций

Затруднение статического анализа




“nope”

Упаковщики и загрузчики



Packers —// — Loaders

А еще есть самописные виртуальные машины. Можно посмотреть например здесь: https://vxlab.info/inside-vm/

Или если вы фанат 8-\16-битных игр, то можете оценить, что любой эмулятор NES или Sega – по сути – та же виртуальная машина.

Anti-VM

Затруднение анализа с использованием виртуальных машин

Специфика окружения

Артефакты файловой системы

Специфические сервисы

Специфическое оборудование или его свойства

Инструкции (sidt, cpuid)

Проверка наличия соединения с Интернетом

Проверка доступности файлов

Запланированная практика

- Ручная упаковка и распаковка UPX
- Демонстрация возможности создания crypto-stub
- Решение crackme – Squirtle
- На дом: FlareOn 01/07 challenge
- На дом: Обнаружение VirtualBox

Полезные ссылки

https://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide#DebugObjectHandle

http://antukh.com/blog/2015/01/19/malware-techniques-cheat-sheet/

https://www.aldeid.com/wiki/Main_Page

https://github.com/xoreaxeaxeax/movfuscator/blob/master/slides/domas_2015_the_movfuscator.pdf

http://handlers.sans.edu/tliston/ThwartingVMDetection_Liston_Skoudis.pdf