Скидка 20% на рекламу для читателей
Реклама помогает нам развивать проекты
Скрыть рекламу

Хакер

Антиотладка. Теория и практика защиты приложений от дебага

Andrey Fedotov aka Hacker, Vladislav Ustyugov aka Blackfox, Alexey Malanov aka Tailless

Содержание статьи

1. IsDebuggerPresent() и структура PEB
2. Process Environment Block
3. IsDebugPtr
4. Flags и ForceFlags
5. CheckRemoteDebuggerPresent() и NtQueryInformationProcess
6. Trace in MSExceptionPolicy
7. Debug Object
8. Process Debug Port
9. Проверка родительского процесса
10. TLS Callbacks
11. Отладочные регистры
12. NtSetInformationThread
13. Timing checks
14. SwDebugPrivilege
15. SetHandleInformation
16. Заключение

В материале рассматриваются отладка приближает многие программисты: один остает в ней, другой, наоборот, пытается ее предотвратить. Мы расскажем, как отладчик определяет или автоматически изменяет состояние разработанного кода. Мы в подробностях рассмотрим разные методы борьбы с дебагом — от простых до довольно изощренных.

Поскольку сейчас популярны не только архитектуры x86, но и x64-ОС, можно спокоя старые способы обнаружения отладчика. Другие будут нацелены только на x64-версии, потому что многие современные приложения в архитектуре x86 В этой статье с примерами кодов мы будем использовать ОС Windows 7 x64 и отладчик x64dbg, обновленный вариант отладчика x32dbg, который будет работать и на x64, и на x86.

IsDebuggerPresent() и структура PEB

Начнем с самого простого и элементарного: отладчик в функции IsDebuggerPresent() будет бы запрашивать. Эта функция использует для определения своего статуса в очень простой и очень быстрый способ — она просто смотрит в поле BeingDebugged структуры PEB.

Что представляет собой Windows? IsDebuggerPresent() Эта функция обращается к структуре PEB.

Process Environment Block

Блок окружения процесса (PEB) автоматически загружается операционной системой, инициализирует переменные среды, аргументы запуска, а также ряд других параметров, которые понадобятся самому. Он содержит много полей: например, отладчик может узнать информацию о текущем процессе, например о загруженных модулях. Получить структуру PEB можно, обратившись к полю по смещению 0x60 (0x30) для x64 (x32) ОС.

Соответственно, если запустить в отладке функцию IsDebuggerPresent(), мы увидим что-то вроде:

```
mov     eax,dword ptr ds:[30h]
movzx   eax,byte ptr [eax+2]
ret
```

А на x64 есть будет такое:

```
mov     rax,qword ptr gs:[60h]
movzx   eax,byte ptr [rax+2]
ret
```

Что значит byte ptr [eax+2]? Что этому смещению соответствует поле BeingDebugged в структуре PEB, которое и анализирует сама в данном случае. Как вещи можно использовать PEB для определения отладки?

NtGlobalFlag

Все флаги отладки системы выставляет флаг NT_GLOBAL_FLAG_DEBUGGED, FLG_HEAP_ENABLE_TAIL_CHECK, FLG_HEAP_ENABLE_FREE_CHECK и поле NtGlobalFlag. Когда отладчик подключается к приложению, ставится флаг NtGlobalFlag, который определяет, какие отладочные возможности включены. Для x32 ОС — по смещению 0x68 (0xBC). Следующая NtGlobalFlag в PEB для x64 составляет 0x2, для x64 — 0x70. Чтобы посмотреть флаги среды разработки, можно использовать функцию GetNtGlobalFlag() или, в любом случае, вы можете использовать дизассемблер:

```
kernel32!GetNtGlobalFlag:
mov     eax,dword ptr ds:[7FFE0068]
ret
```

```
mov     eax,dword ptr [7FFE0068]
ret
```

```
mov     eax,dword ptr gs:[60]
mov     eax,dword ptr [eax+0BC]
ret
```

```
mov     rax,qword ptr gs:[60]
mov     eax,dword ptr [rax+70]
ret
```

Flags и ForceFlags

PEB также содержит указатель на структуру _heap, в которой есть поле Flags и ForceFlags. Когда процессом создающим и приложением, поле Flags и ForceFlags принимают значение отладки. Получить адрес при отладке на данных flags рулем, поле Flags для дебага flags, ForceFlags для дебага ForceFlags.

```
kernel32!GetProcessHeap:
mov     eax,dword ptr ds:[7FFE0030]
ret
mov     eax,dword ptr [eax+18]
ret
```

```
mov     eax,dword ptr [7FFE0030]
ret
mov     eax,dword ptr [eax+40]
ret
```

```
mov     rax,qword ptr gs:[60]
mov     rax,qword ptr [rax+30]
ret
mov     rax,qword ptr [rax+58]
ret
```

```
mov     rax,qword ptr gs:[60]
mov     rax,qword ptr [rax+30]
ret
mov     rax,qword ptr [rax+70]
ret
```

Если поле заголовка в куче отлично нулевым, то произошло решение с тем, что идет отладка. Код для CheckRemoteDebuggerPresent:

```
BOOL CheckRemoteDebuggerPresent(
  [in]  HANDLE hProcess,
  [out] PBOOL pDebuggerPresent
);
if (CheckRemoteDebuggerPresent(hProcess, &bDebuggerPresent)
{
  printf("Debugger present!");
}
else
{
  printf("Error %d", GetLastError());
}
```

Код передает параметры hProcessHandle и адреса в функции NtQueryInformationProcess:

```
NtQueryInformationProcess(
  [in]      HANDLE           ProcessHandle,
  [in]      PROCESSINFOCLASS ProcessInformationClass,
  [out]     PVOID            ProcessInformation,
  [in]      ULONG            ProcessInformationLength,
  [out, opt] PULONG          ReturnLength
);
```

Структура ERROR_PORT, или Port handle, содержит много информации о процессе, указателе на начальную структуру данных, а так же на PEB. Достаточно вызвать NtQueryInformationProcess() с параметрами ProcessBasicInformation, для x32 ОС — по смещению 0x08 (0x10), а для начальных структуры, кроме PEB. Все процессы имеют эту структуру.

Если поле заголовка в куче отлично нулевым, то произошло решение с тем, что идет отладка. Код для CheckRemoteDebuggerPresent:

```
BOOL CheckRemoteDebuggerPresent(
  [in]  HANDLE hProcess,
  [out] PBOOL pDebuggerPresent
);
if (CheckRemoteDebuggerPresent(hProcess, &bDebuggerPresent)
{
  printf("Debugger present!");
}
else
{
  printf("Error %d", GetLastError());
}
```

Код передает параметры hProcessHandle и адреса в функции NtQueryInformationProcess:

```
NtQueryInformationProcess(
  [in]      HANDLE           ProcessHandle,
  [in]      PROCESSINFOCLASS ProcessInformationClass,
  [out]     PVOID            ProcessInformation,
  [in]      ULONG            ProcessInformationLength,
  [out, opt] PULONG          ReturnLength
);
```

Структура ERROR_PORT, или Port handle, содержит много информации о процессе, указателе на начальную структуру данных, а так же на PEB. Достаточно вызвать NtQueryInformationProcess() с параметрами ProcessBasicInformation, для x32 ОС — по смещению 0x08 (0x10), а для начальных структуры, кроме PEB. Все процессы имеют эту структуру.

Trace in MSExceptionPolicy

Отладчик может использовать для определения своего статуса и трассировки выполняемых функций. Функция RtlpGetExceptionFunction() вызывается в ее коде исключение — которая называется RtlpGetExceptionFunction(). Для x32 ОС, что при трассировке отладчик в коде ошибка RtlpGetExceptionFunction() вызывается.

```
RtlpGetExceptionFunction:
mov     edi,edi
push    ebp
mov     ebp,esp
push    esi
mov     esi,dword ptr ds:[7FFE0030]
test    dword ptr [esi+40],4
jne     ntdll!RtlpGetExceptionFunction+0x1a
```

```
RtlpGetExceptionFunction:
mov     rax,qword ptr gs:[60]
test    byte ptr [rax+70],4
jne     ntdll!RtlpGetExceptionFunction+0x1a
```

Debug Object

При отладке приложения создается DebugObjects, объект отладки. Если отладчик подключен к процессу, в поле DebugObjectHandle будет лежать значение. Функция NtQueryInformationProcess() вызывается для того, чтобы определить наличие такого объекта. Для x32 ОС — по смещению 0x2C (0x4C), а для x64 — 0x78. Если DebugObjectHandle содержит значение, то в памяти создан объект отладки.

```
NtQueryInformationProcess(
  [in]      HANDLE           ProcessHandle,
  [in]      PROCESSINFOCLASS ProcessInformationClass,
  [out]     PVOID            ProcessInformation,
  [in]      ULONG            ProcessInformationLength,
  [out, opt] PULONG          ReturnLength
);
if (Status == 0x00000000 && DebugObj != NULL && (DebugObj & 0x7fffffff) == 0)
{
  printf("Debugger present!");
}
else
{
  printf("Error %d", GetLastError());
}
```

ProcessDebugFlags

Следующая проверка отладки, которая нам позволит функция NtQueryInformationProcess, — это поле ProcessDebugFlags, имеющее номер 0x1F. Передаем значение ProcessDebugFlags для системного вызова NtQueryInformationProcess() — в ответ мы получаем значение, которое отражает в структуре EPROCESS. Если поле равно нулю, это значит, что в данном процессе присутствует отладчик. Код вызова NtQueryInformationProcess() аналогичен, значение только номер ProcessDebugFlags вместо ProcessDebugObjectHandle:

```
NtQueryInformationProcess(
  [in]      HANDLE           ProcessHandle,
  [in]      PROCESSINFOCLASS ProcessInformationClass,
  [out]     PVOID            ProcessInformation,
  [in]      ULONG            ProcessInformationLength,
  [out, opt] PULONG          ReturnLength
);
if (Status == 0x00000000 && ProcessDebugFlags == 0) and (and 0x7fffffff) == 0)
{
  printf("Debugger present!");
}
else
{
  printf("Error %d", GetLastError());
}
```

Проверка родительского процесса

Если поле заголовка в куче отлично нулевым и так, что мы можем проверить, или можно было запустить приложение, которое как защищает. Используется или отладчиком. Этот способ проверки отладчика, который запускает приложение, основывается на том, что в системе есть только один процесс explorer.exe, который запускает все остальные процессы. Если пользователь решил запустить свою программу отладчиком, то проверка в очереди ему покажет, что родитель — не explorer.exe. Если родитель explorer.exe, то проверка в очереди ему покажет explorer.exe.

Проверить родительский процесс мы будем при помощи уже известной нам функции NtQueryInformationProcess и структуры PROCESS_BASIC_INFORMATION. Для x32 ОС — по смещению 0x14 (0x28), а структура стоит или запущенный процесс в системе можно как показать:

```
NtQueryInformationProcess(ProcessHandle, ProcessBasicInformation, &pbi, sizeof(pbi)
```

Итак, в InheritedFromUniqueProcessId содержится ID процесса, который запускает наш. Его можно использовать как отдел, чтобы определить, запущен ли наш файл, название процесса и сравнить с именем отладчика или процесса, не нарушая или не это.

TLS Callbacks

Этот антиотладочный метод интересен прилагается к тем, что мы встречаем в предыдущем разделе. Это TLS Callbacks, которые выполняются перед тем, как начинается выполнение основного кода. Если вы, например, хотите отладчик, у вас в коде есть TLS Callbacks. Отладчик не сможет отслежить, что происходит в коде. Чтобы отладчик смог отслежить, что происходит в коде, нужно отключить TLS Callbacks. Чтобы TLS Callbacks, которые стоят перед тем, как начинается выполнение основного кода, нужно отключить, что проверяет. Чтобы отключить этот метод обнаружения, необходимо заменить инструкцию ссылки на TLS Callbacks.

```
.rdata:0000000140A2A000 ; IMAGE_TLS_DIRECTORY64 _tls_used
.rdata:0000000140A2A000 _tls_used       dq 0C0A00000C0A0A000h
.rdata:0000000140A2A000                 ; Data1
.rdata:0000000140A2A008                 dq 0C0A00000C0A0A008h
.rdata:0000000140A2A008                 ; Data2
.rdata:0000000140A2A010                 dq 0C0A00000C0A0A010h
.rdata:0000000140A2A010                 ; Data3
.rdata:0000000140A2A018                 dq 0C0A00000C0A0A018h
.rdata:0000000140A2A018                 ; Address of index
.rdata:0000000140A2A020                 dq 0C0A00000C0A0A020h
.rdata:0000000140A2A020                 ; Address of callbacks
.rdata:0000000140A2A028                 dd 0A0A0000h
.rdata:0000000140A2A028                 ; Size of zero fill
.rdata:0000000140A2A028                 dd 0A0A0000h
.rdata:0000000140A2A028                 ; Characteristics
```

Отладочные регистры

Если в отладочных регистрах есть какие-то данные, то это еще один признак. Это дает в тех, что отладочные регистры — адресно-контрольные регистры и используются для доступа к данным можно только в режиме ядра. Но мы попробуем получить значение этого регистра. Для x32 ОС — по смещению 0x04 (0x08) и для x64 — 0x08 (0x10). Если отладочных нет, то значение равно нулю. Если отладочные есть, то значение отличается от нуля.

```
kernel32!GetThreadContext:
mov     edi,edi
push    ebp
mov     ebp,esp
push    esi
mov     esi,dword ptr ds:[7FFE0030]
test    dword ptr [esi+40],4
jne     ntdll!RtlpGetExceptionFunction+0x1a
```

```
kernel32!GetThreadContext:
mov     rax,qword ptr gs:[60]
test    byte ptr [rax+70],4
jne     ntdll!RtlpGetExceptionFunction+0x1a
```

NtSetInformationThread

Еще один отладочный метод отслеживания основан на передаче флага в NtSetInformationThread() совершается в структуре _ETHREAD на смещение 0x74 в функции ntdll!NtSetInformationThread. Вот как выглядит прологом функции:

```
NTSYSAPI NTSTATUS
NtSetInformationThread(
  [in]  HANDLE          ThreadHandle,
  [in]  THREADINFOCLASS ThreadInformationClass,
  [in]  PVOID           ThreadInformation,
  [in]  ULONG           ThreadInformationLength
);
```

Если нужно отрезает наш поток от отладчика, перехватывает отправляет ему отладочные сообщения, или же, удобнее, вообще, ставит поток на паузу. Особенность NtSetInformationThread() заключается в том, что он разворачивает и работает благодаря системе возвращения ОС. Вот код, который реализует отслеживание главного потока прологом от отладчика:

```
NTSTATUS Stat = NtSetInformationThread(ThreadHandle, NULL, NULL, 11);
```

NtCreateThreadEx

Подобно предыдущей работает и функция NtCreateThreadEx. Она появилась в Windows Vista/server 2008. Все того же можно использовать в качестве способа контролировать для отладчика. Для x32 ОС — по смещению 0x0C (0x18) и для x64 — 0x10 (0x20). Последние параметры NtCreateThreadEx, flags, pAttributeList и BOOL CreateSuspended является будет лежащие для работоспособ. Прологом функции:

```
NTSTATUS
NtCreateThreadEx(
  [out]   PHANDLE ThreadHandle,
  [in]    ACCESS_MASK DesiredAccess,
  [in]    POBJECT_ATTRIBUTES ObjectAttributes,
  [in]    HANDLE ProcessHandle,
  [in]    PTHREAD_START_ROUTINE lpStartAddress,
  [in]    PVOID lpParameter,
  [in]    ULONG CreateThreadFlags,
  [in]    SIZE_T ZeroBits,
  [in]    SIZE_T StackSize,
  [in]    SIZE_T MaximumStackSize,
  [in]    PPS_ATTRIBUTE_LIST AttributeList
);
```

Код отключения отладчика:

```
DWORDS ulFlag = 1;

NTSTATUS status = NtSetInformationThread(
  GetCurrentThread(),
  (THREADINFOCLASS)0x11,
  (PVOID)&ulFlag,
  (ULONG)sizeof(ulFlag));
if (status == STATUS_SUCCESS)
{
  printf("Success");
}
else
{
  printf("Error %d", GetLastError());
}
```

SetHandleInformation

Функция SetHandleInformation() применяется для установки свойств дескриптора объектов, на которых указывает hObject. Прологом функции выглядит следующим образом:

```
BOOL SetHandleInformation(
  [in]  HANDLE hObject,
  [in]  DWORD dwMask,
  [in]  DWORD dwFlags
);
```

Чтобы объектом разлочить — например, что может быть задержек, отбрасывание дает выделение. Мы можем этим использовать: создадим выделен с флагом:

```
HANDLE hMutex = CreateMutex(NULL, FALSE, _T("MyMutexObj");
SetHandleInformation(hMutex, HANDLE_FLAG_PROTECT_FROM_CLOSE, HANDLE_FLAG_PROTECT_FROM
_tprintf(_T("Mutex created\n");
CloseHandle(hMutex);
_tprintf(_T("Mutex closed\n");
```

```
_tprintf(_T("Mutex created\n");
HANDLE hMutex = CreateMutex(NULL, FALSE, _T("MyMutexObj");
SetHandleInformation(hMutex, HANDLE_FLAG_PROTECT_FROM_CLOSE, HANDLE_FLAG_PROTECT_FROM
_tprintf(_T("Mutex created\n");
CloseHandle(hMutex);
_tprintf(_T("Mutex closed\n");
```

Заключение

Мы рассмотрели несколько способов защиты приложения от отладчика. И сказали показать, почему методы отладки и разработки, как они работают на низком уровне. Чтобы лучше разбирается в том, что происходит в системе, мы рекомендуем, как работает ОС, как приложения взаимодействуют с ядром ОС, как ядро ОС взаимодействует с аппаратным обеспечением. Мы рассмотрели несколько способов определения отладчика, а также способы, которые помогают ему обнаружить, рекомендуем и автоматическое снятие расставлено и значения.

Rate this article

Mr. Xerof
Хакер

Реклама Скрыть рекламу Пожаловаться на рекламу
Скрыть все баннеры

Другие статьи по теме

Антиотладка для Linux: Теория и практика защиты приложений от дебага
Linux

Теория и практика: Как пишут ботов для игры World of Tanks
World of Tanks

Обзор ПО для взломщиков: Отладка и исследование приложений для Windows
Обзоры

Дебаг: теория, практика: Как отладить свой код, не сходя с ума
Разработка

Теория и практика: Как пишут ботов для игры World of Tanks
World of Tanks

Обзор ПО для взломщиков: Отладка и исследование приложений для Windows
Обзоры

Подписывайтесь на нас в Telegram!
Самые свежие статьи и лучшие посты
Подписаться

Все комментарии 0
Вы можете сделать: ЛОГИН ПАРОЛЬ Выход
Чтобы подписаться на комментарии, нужно выбрать одну из двух вещей:
Если у вас уже есть логин, то достаточно ввести его и пароль, а потом нажать на кнопку «Вход».
Если же вы еще не регистрировались на сайте, то вам нужно ввести дополнительно еще и свой e-mail, и после нажатия на кнопку «Регистрация» вам будет выслано письмо с паролем. Вы сможете сразу же после этого зайти на сайт под своим логином и паролем, а также изменить и логин, и пароль в своем профиле.
Оставлять комментарии с анонимных прокси-серверов, а также с прокси-серверов, замеченных в спаме, мы не можем.
Оставлять комментарии с IP-адреса, который был заблокирован по чьей-то жалобе, мы также не можем.
Комментарий, который будет выглядеть как спам, может не пройти фильтр и не появиться сразу же после отправки.
Комментарий, который будет выглядеть как спам, может быть отредактирован модератором.
Если вы оставляете ссылку, то в комментарии должно быть минимум 20 слов, иначе он может не пройти фильтр