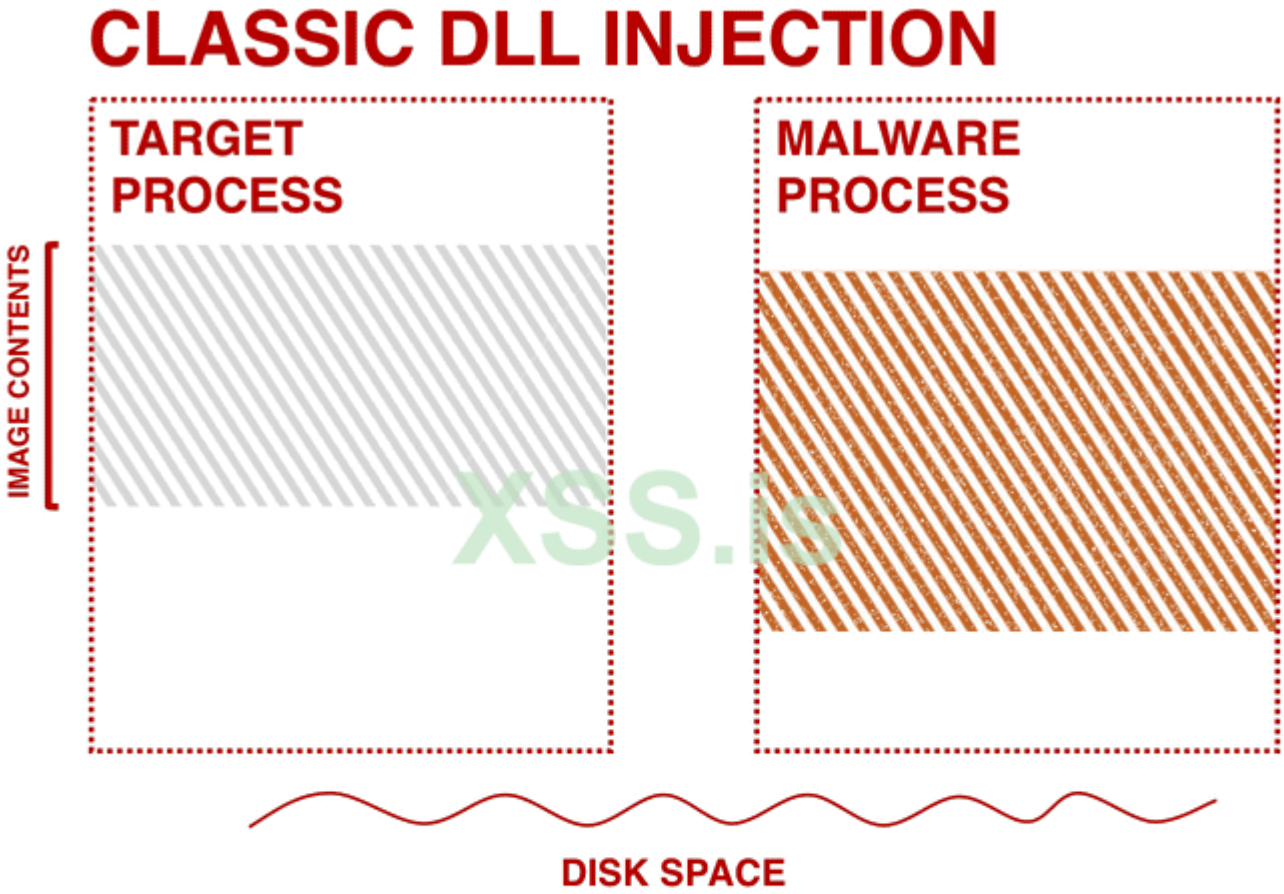
Статья Десять методов инжекции процесса: технический обзор распространенных и актуальных методов инжекции процесса

Инжекция процесса - это широко распространенный метод уклонения от защиты, который часто используется в вредоносных программах и безфайловых объектах и влечет за собой выполнение пользовательского кода в адресном пространстве другого процесса. Процесс инжекции улучшает скрытность, а некоторые методы также достигают стойкости. Хотя существует множество методов инжекции процессов, в этом блоге я представляю десять методов, которые можно увидеть в дикой среде, которые запускают вредоносный код от имени другого процесса. Кроме того, я предоставляю скриншоты для многих из этих методов, чтобы облегчить реверс инжиниринг и анализ вредоносных программ, помогая обнаружению и защите от этих распространенных методов.

1. Классическая инжекция DLL через функции CreateRemoteThread и LoadLibrary

Этот метод является одним из наиболее распространенных методов, используемых для внедрения вредоносных программ в другой процесс. Вредоносная программа записывает путь к своей вредоносной динамически подключаемой библиотеке (DLL) в виртуальном адресном пространстве другого процесса и обеспечивает его загрузку удаленным процессом, создавая удаленный поток в целевом процессе.

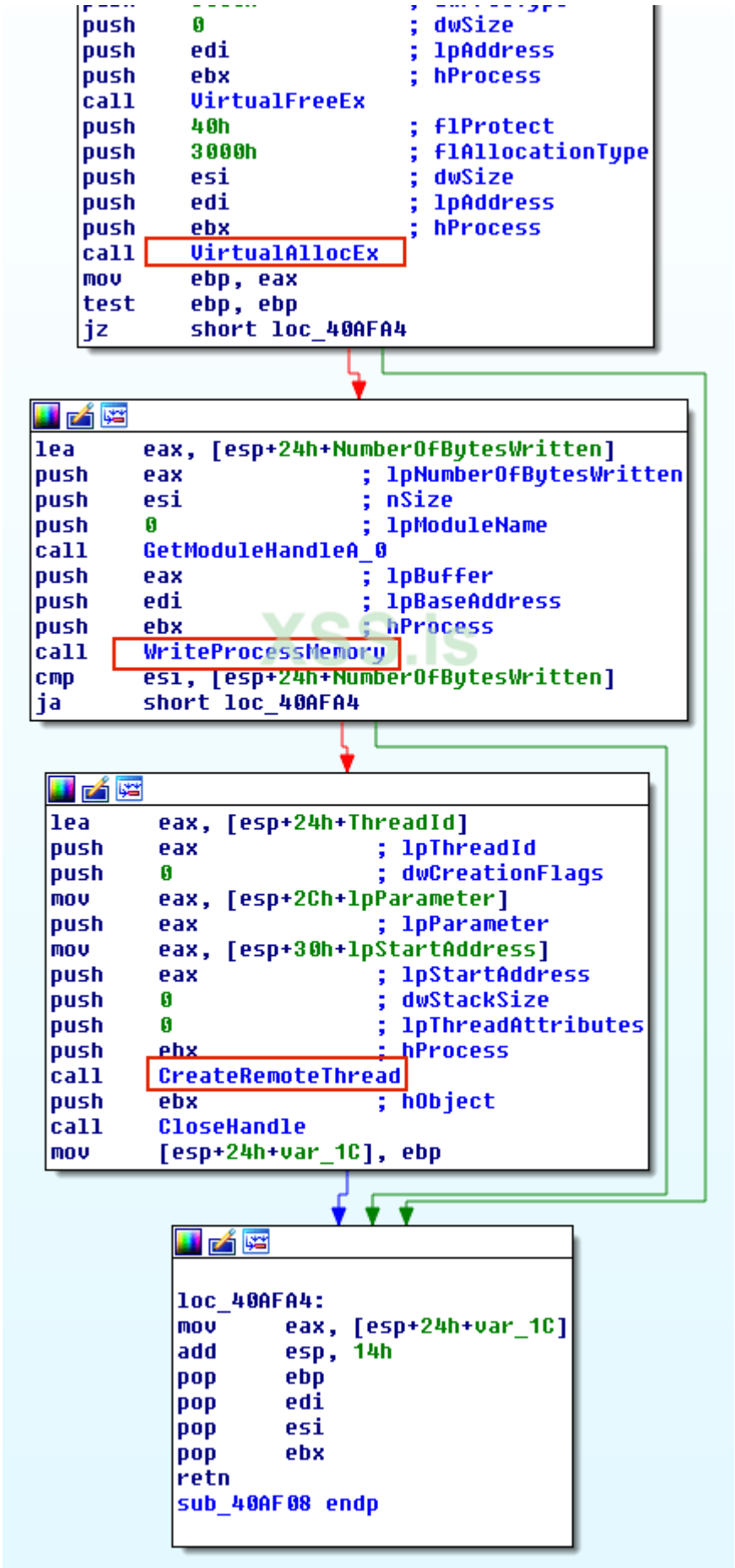


ENDGAME.

Сначала вредоносная программа должна быть нацелена на процесс для внедрения (например, svchost.exe). Обычно это делается путем поиска в процессах и вызова трех интерфейсов прикладных программ (API): CreateToolhelp32Snapshot, Process32First и Process32Next. CreateToolhelp32Snapshot - это API, используемый для перечисления состояний кучи или модуля указанного процесса или всех процессов, и он возвращает моментальный снимок. Process32First извлекает информацию о первом процессе в моментальном снимке, а затем Process32Next используется в цикле для их итерации. После нахождения целевого процесса вредоносная программа получает дескриптор целевого процесса, вызывая OpenProcess.

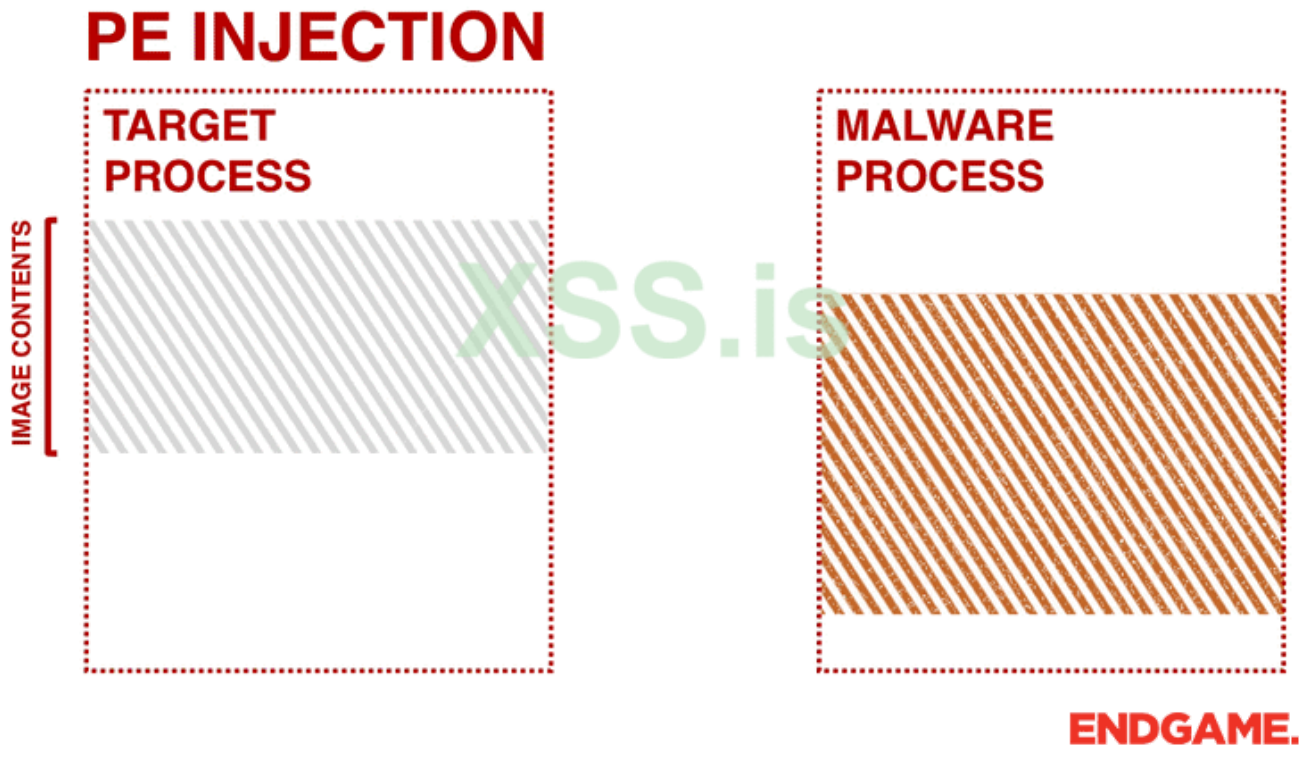
Как показано на рисунке 1, вредоносная программа вызывает VirtualAllocEx, чтобы выделить место для записи пути к своей DLL. Затем вредоносная программа вызывает WriteProcessMemory, чтобы записать путь в выделенную память. Наконец, для выполнения кода в другом процессе вредоносная программа вызывает такие API-интерфейсы, как CreateRemoteThread, NtCreateThreadEx или RtlCreateUserThread. Последние две функции недокументированы. Однако общая идея состоит в том, чтобы передать адрес LoadLibrary одному из этих API, чтобы удаленный процесс выполнял DLL от имени вредоносной программы.

CreateRemoteThread отслеживается и помечается многими продуктами безопасности. Кроме того, требуется вредоносная DLL на диске, которая может быть обнаружена. Учитывая, что злоумышленники чаще всего инжектируют код для обхода защиты, искушенные злоумышленники, вероятно, не будут использовать этот подход. На приведенном ниже снимке экрана показано вредоносное ПО с именем Rebhip, выполняющее эту технику.


push 0 ; dwSize
push edi ; lpAddress
push ebx ; hProcess
call VirtualFreeEx
push 40h ; flProtect
push 3000h ; flAllocationType
push esi ; dwSize
push edi ; lpAddress
push ebx ; hProcess
call VirtualAllocEx
mov ebp, eax
test ebp, ebp
jz short loc_40AFA4
lea eax, [esp+24h+NumberOfBytesWritten]
push eax ; lpNumberOfBytesWritten
push esi ; nSize
push 0 ; lpModuleName
call GetModuleHandleA_0
push eax ; lpBuffer
push edi ; lpBaseAddress
push ebx ; hProcess
call WriteProcessMemory
cmp esi, [esp+24h+NumberOfBytesWritten]
ja short loc_40AFA4
lea eax, [esp+24h+ThreadId]
push eax ; lpThreadId
push 0 ; dwCreationFlags
mov eax, [esp+2Ch+lpParameter]
push eax ; lpParameter
mov eax, [esp+30h+lpStartAddress]
push eax ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
push ebx ; hProcess
call CreateRemoteThread
push ebx ; hObject
call CloseHandle
mov [esp+24h+var_1C], ebp
loc_40AFA4:
mov eax, [esp+24h+var_1C]
add esp, 14h
pop ebp
pop edi
pop esi
pop ebx
retn
sub_40AF08 endp

2. Инжекция PORTABLE EXECUTABLE (PE INJECTION)

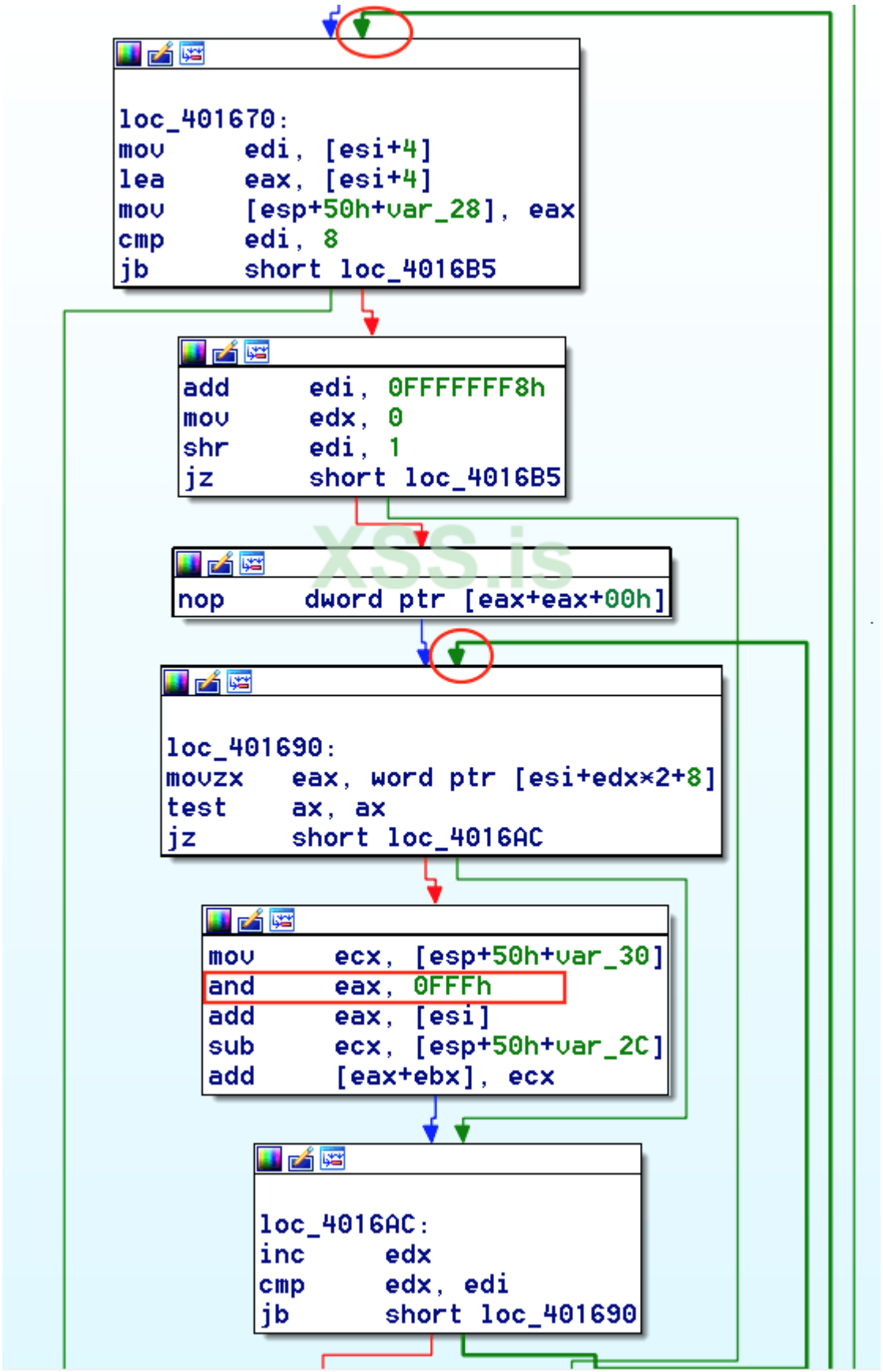
Вместо передачи адреса LoadLibrary вредоносная программа может скопировать свой вредоносный код в существующий открытый процесс и заставить его выполниться (либо с помощью небольшого шеллкода, либо с помощью вызова CreateRemoteThread). Одно из преимуществ PE инжекции по сравнению с техникой LoadLibrary состоит в том, что вредоносное ПО не должно сбрасывать вредоносную DLL на диске. Подобно первому способу, вредоносная программа выделяет память в хост-процессе (например, с помощью вызова VirtualAllocEx), и вместо записи "пути DLL" она записывает свой вредоносный код, вызывая WriteProcessMemory. Однако препятствием при таком подходе является изменение базового адреса скопированного образа. Когда вредоносная программа внедряет свой PE в другой процесс, она имеет новый базовый адрес, который непредсказуем, что требует от него динамического пересчета фиксированных адресов своего PE. Чтобы преодолеть это, вредоносная программа должна найти адрес своей таблицы перемещения в хост-процессе и разрешить абсолютные адреса скопированного образа, просматривая свои дескрипторы перемещения.



Этот метод аналогичен другим методам, таким как рефлективное инжектирование DLL и модуль памяти, поскольку они не сбрасывают файлы на диск. Тем не менее, подходы к использованию модулей памяти и отражающей DLL еще более незаметны. Они не зависят от каких-либо дополнительных API-интерфейсов Windows (например, CreateRemoteThread или LoadLibrary), поскольку они загружаются и выполняются в

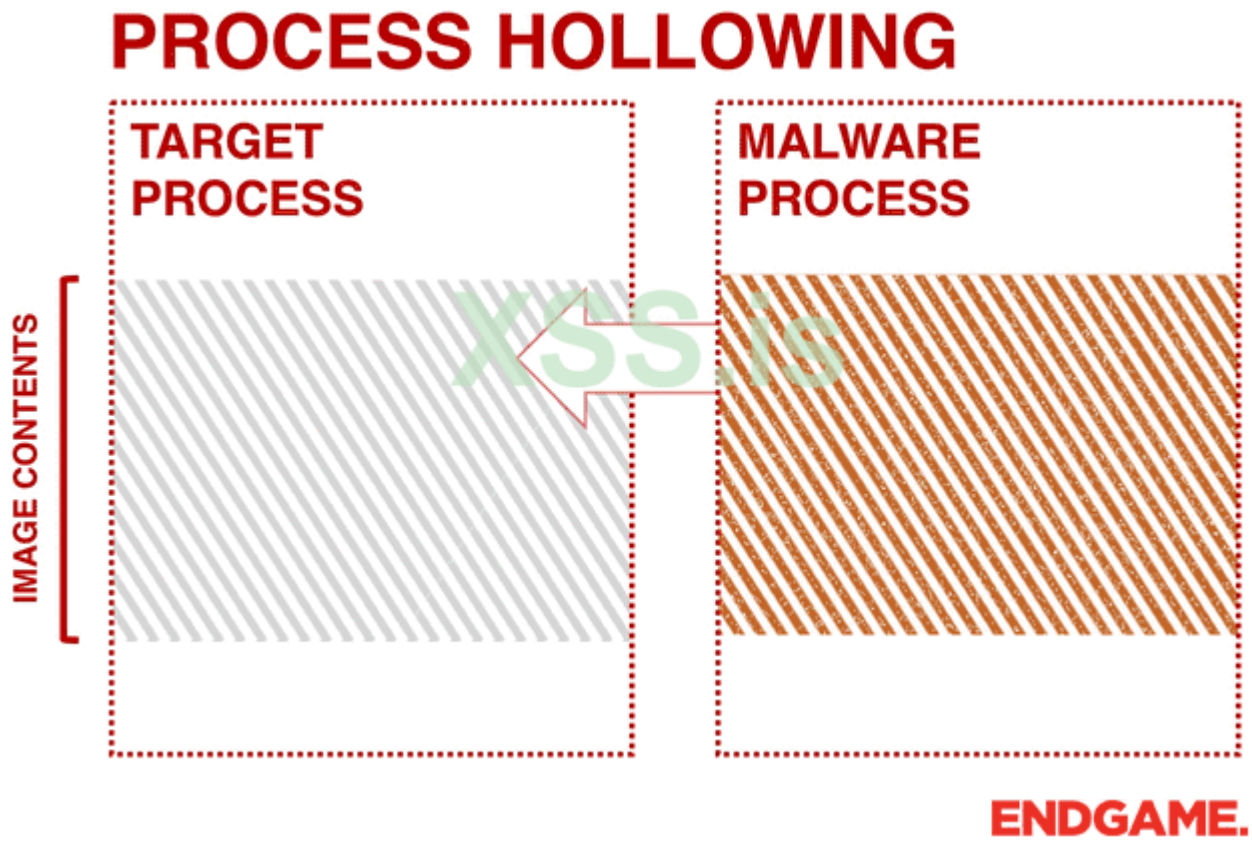
памяти. Отраженная DLL-инжекция работает путем создания DLL, которая отображает себя в памяти при выполнении, вместо того, чтобы полагаться на загрузчик Windows. Модуль памяти аналогичен рефлективной инжекции DLL, за исключением того, что инжектор или лоадер отвечает за отображение целевой DLL в память вместо отображения самой библиотеки DLL. В предыдущем сообщении в блоге эти два подхода к памяти широко обсуждались.

При анализе PE-инъекции очень часто можно увидеть циклы (обычно два цикла for, один вложен в другой) перед вызовом CreateRemoteThread. Этот метод довольно популярен среди криптографов (программ, которые шифруют и запутывают вредоносные программы). На рисунке 2 образец использует преимущества этого метода. В коде есть два вложенных цикла для настройки таблицы перемещения, которые можно увидеть перед вызовами WriteProcessMemory и CreateRemoteThread. Инструкция "and 0x0fff" также является еще одним хорошим индикатором, показывающим, что первые 12 бит используются для получения смещения в виртуальном адресе содержащего блока перемещения. Теперь, когда вредоносная программа пересчитала все необходимые адреса, все, что ей нужно сделать, - это передать свой начальный адрес в CreateRemoteThread и запустить его.



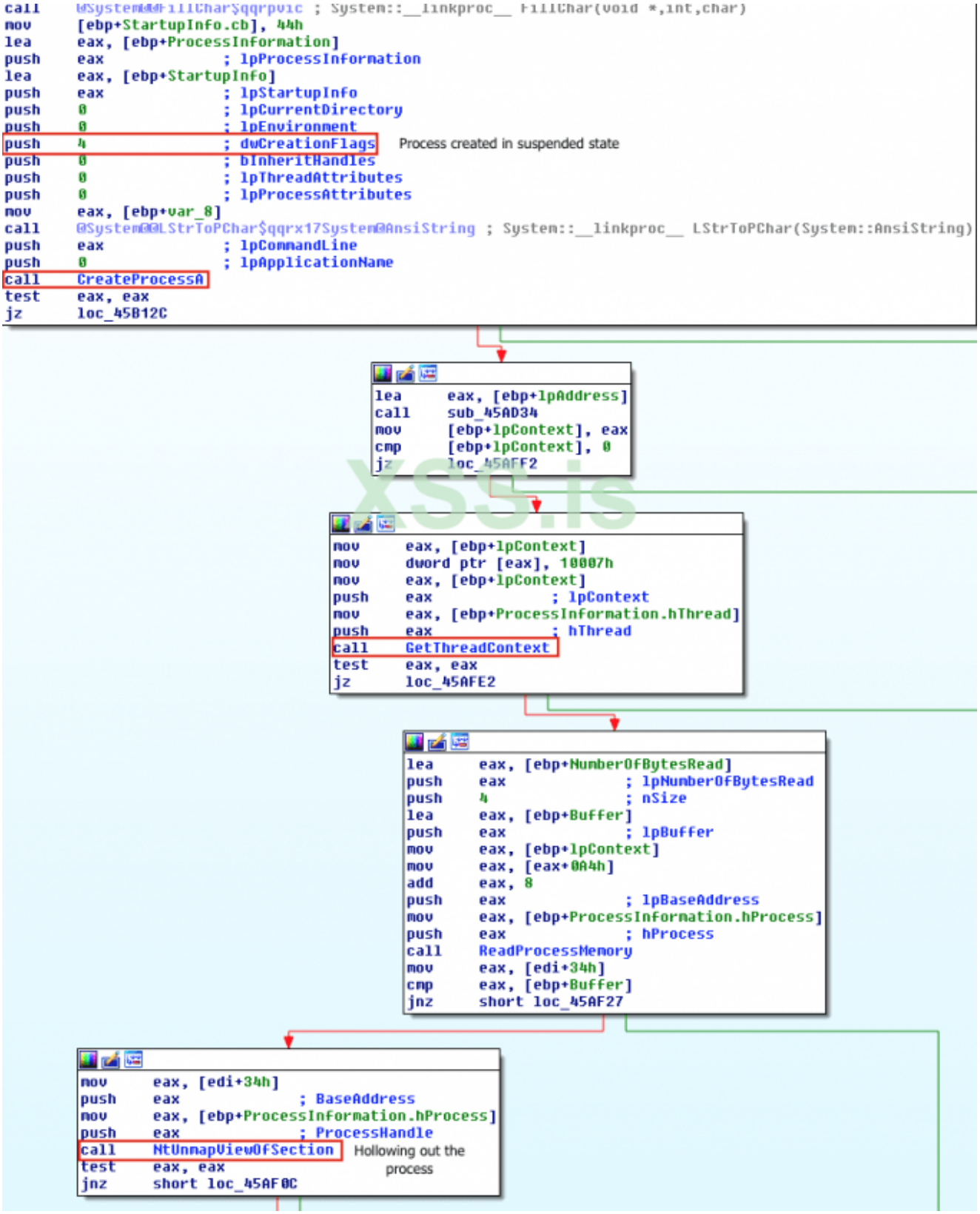
3. Вытеснение процесса (А.К.А замена процесса и запуск PE)

Вместо внедрения кода в хост-программу (например, внедрение DLL) вредоносное ПО может выполнять методику, известную как process hollowing. Process hollowing происходит, когда вредоносная программа извлекает (вытесняет) законный код из памяти целевого процесса и перезаписывает пространство памяти целевого процесса (например, svchost.exe) вредоносным исполняемым файлом.



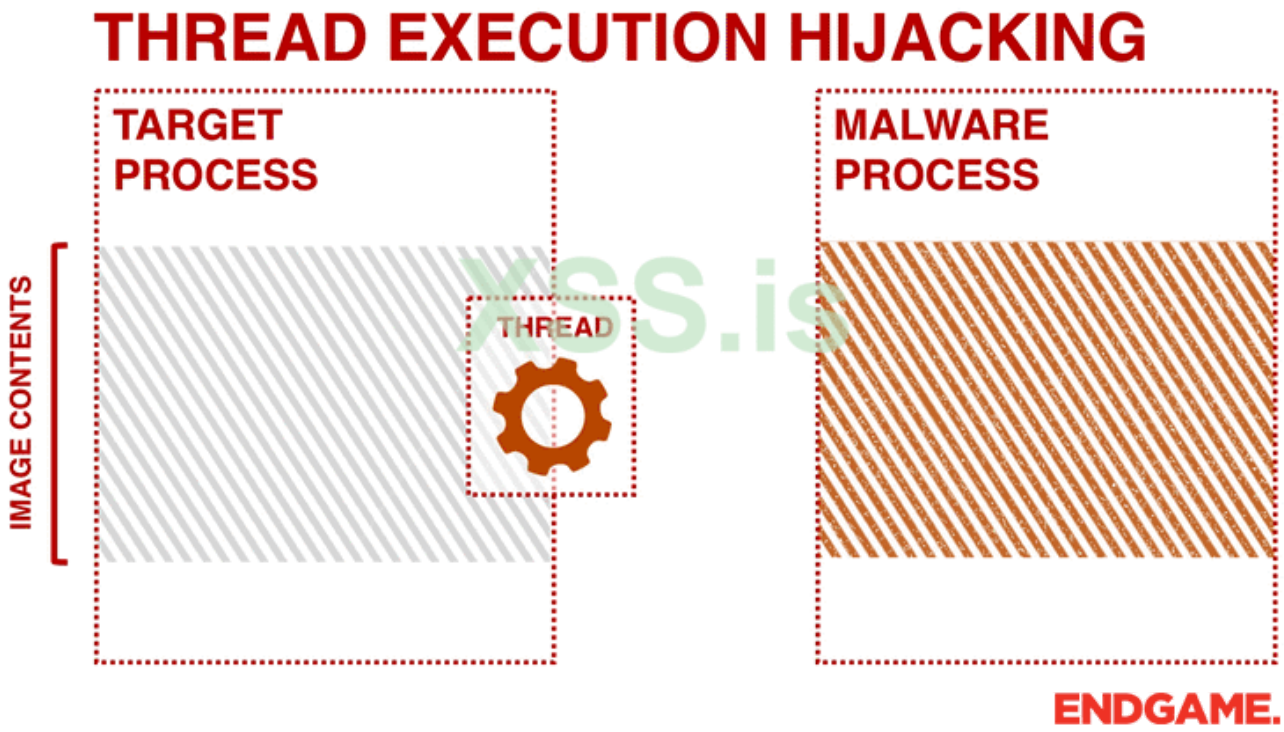
Сначала вредоносная программа создает новый процесс для размещения вредоносного кода в режиме ожидания. Как показано на рисунке 3, это делается путем вызова CreateProcess и установки флага создания процесса в CREATE_SUSPENDED (0x00000004). Основной поток нового процесса создается в приостановленном состоянии и не запускается до тех пор, пока не будет вызвана функция ResumeThread. Затем вредоносная программа должна выгрузить содержимое легитимного файла с его вредоносной полезной нагрузкой. Это делается путем отмены отображения памяти целевого процесса путем вызова либо ZwUnmapViewOfSection, либо NtUnmapViewOfSection. Эти два API в основном освобождают всю память, указанную секцией. Теперь, когда память не отображена, загрузчик выполняет VirtualAllocEx, чтобы выделить новую память для вредоносной программы, и использует WriteProcessMemory для записи каждого из разделов вредоносной программы в целевое пространство процесса. Вредоносная программа вызывает SetThreadContext,

чтобы указать точку входа на новый раздел кода, который она написала. В конце концов, вредоносная программа возобновляет приостановленный поток, вызывая ResumeThread, чтобы вывести процесс из приостановленного состояния.



4. Угон потока (A.K.A SUSPEND, INJECT, AND RESUME (SIR))

Эта методика имеет некоторые сходства с методикой вытеснения, описанной ранее. При перехвате выполнения потока вредоносное ПО нацеливается на существующий поток процесса и избегает любых "шумных процессов" или операций по созданию потоков. Следовательно, во время анализа вы, вероятно, увидите вызовы CreateToolhelp32Snapshot и Thread32First, за которыми следует OpenThread.



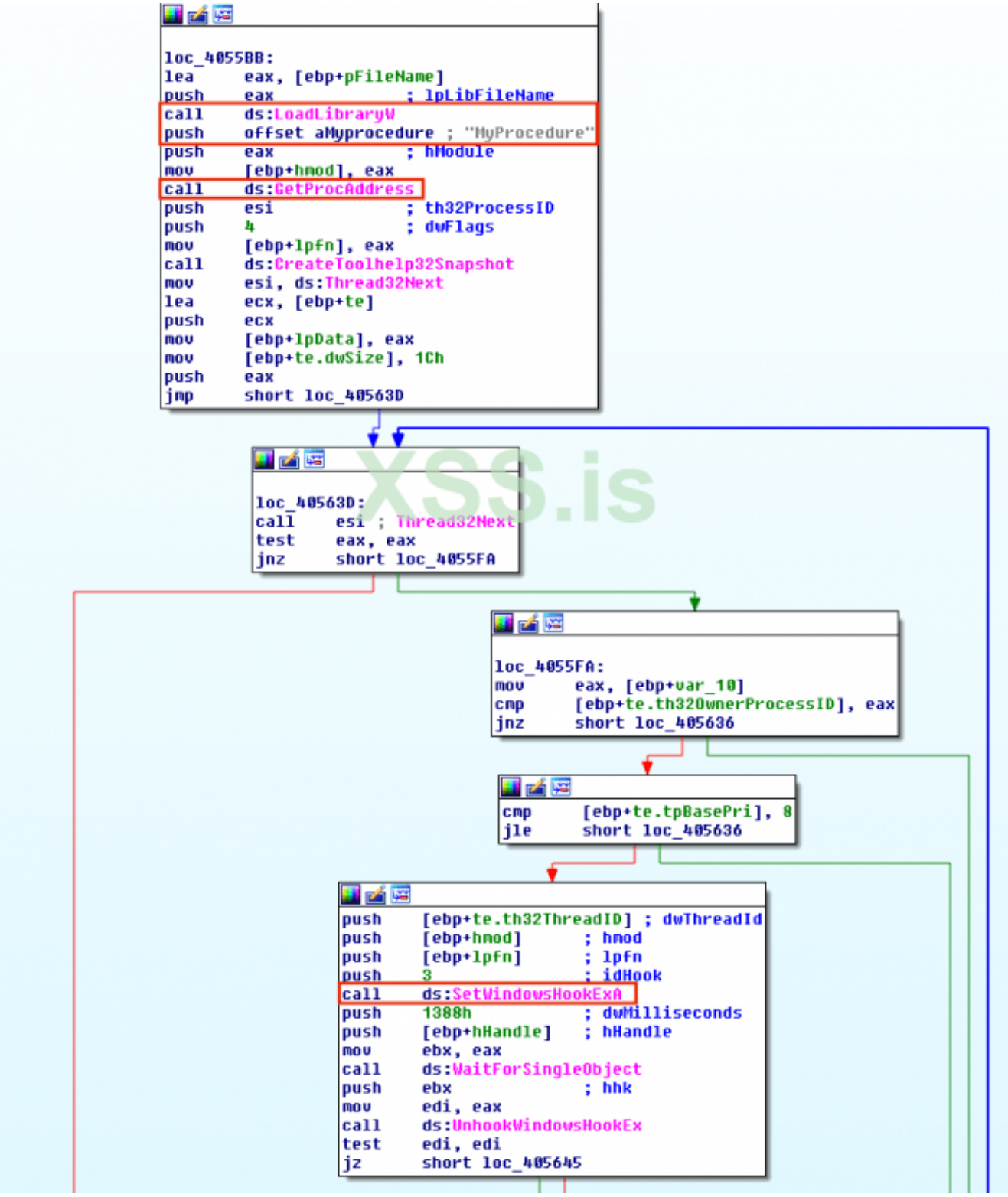
Получив дескриптор целевого потока, вредоносная программа переводит поток в приостановленный режим, вызывая SuspendThread, чтобы выполнить его инжекцию. Вредоносная программа вызывает VirtualAllocEx и WriteProcessMemory для выделения памяти и выполнения внедрения кода. Код может содержать шеллкод, путь к вредоносной DLL и адрес LoadLibrary.

На рисунке 4 показан общий троян, использующий эту технику. Чтобы перехватить выполнение потока, вредоносная программа изменяет регистр EIP (регистр, который содержит адрес следующей инструкции) целевого потока, вызывая SetThreadContext. После этого вредоносная программа возобновляет поток для выполнения шелл-кода, записанного в хост-процесс. С точки зрения злоумышленника, подход SIR может быть проблематичным, поскольку приостановка и возобновление потока в середине системного вызова может привести к сбою системы. Чтобы избежать этого, более сложные вредоносные программы будут возобновляться и повторяться позже, если регистр EIP находится в диапазоне NTDLL.dll.

5. Хук инжекция через функцию SETWINDOWSHOOKEX

Хукинг - это метод, используемый для перехвата вызовов функций. Вредоносные программы могут использовать функцию перехвата, чтобы загружать их вредоносную DLL при возникновении события, инициируемого в определенном потоке. Обычно это делается путем вызова SetWindowsHookEx для установки подпрограммы перехвата в

цепочку перехвата. Функция SetWindowsHookEx принимает четыре аргумента. Первый аргумент - это тип события. События отражают диапазон типов хуков и варьируются от нажатия клавиш на клавиатуре (WH_KEYBOARD) до вводов мыши (WH_MOUSE), CBT и так далее. Второй аргумент - это указатель на функцию, которую вредоносная программа хочет вызвать при выполнении события. Третий аргумент - это модуль, который содержит функцию. Таким образом, очень часто можно увидеть вызовы LoadLibrary и GetProcAddress перед вызовом SetWindowsHookEx. Последний аргумент этой функции - поток, с которым должна быть связана подключаемая процедура. Если это значение установлено равным нулю, все потоки выполняют действие при запуске события. Тем не менее, вредоносные программы обычно нацелены на один поток для меньшего шума, поэтому также можно увидеть вызовы CreateToolhelp32Snapshot и Thread32Next перед SetWindowsHookEx, чтобы найти и нацелиться на один поток. Как только DLL внедряется, вредоносная программа выполняет свой вредоносный код от имени процесса, который его threadId был передан в функцию SetWindowsHookEx. На рисунке 5 Locky Ransomware реализует эту технику.



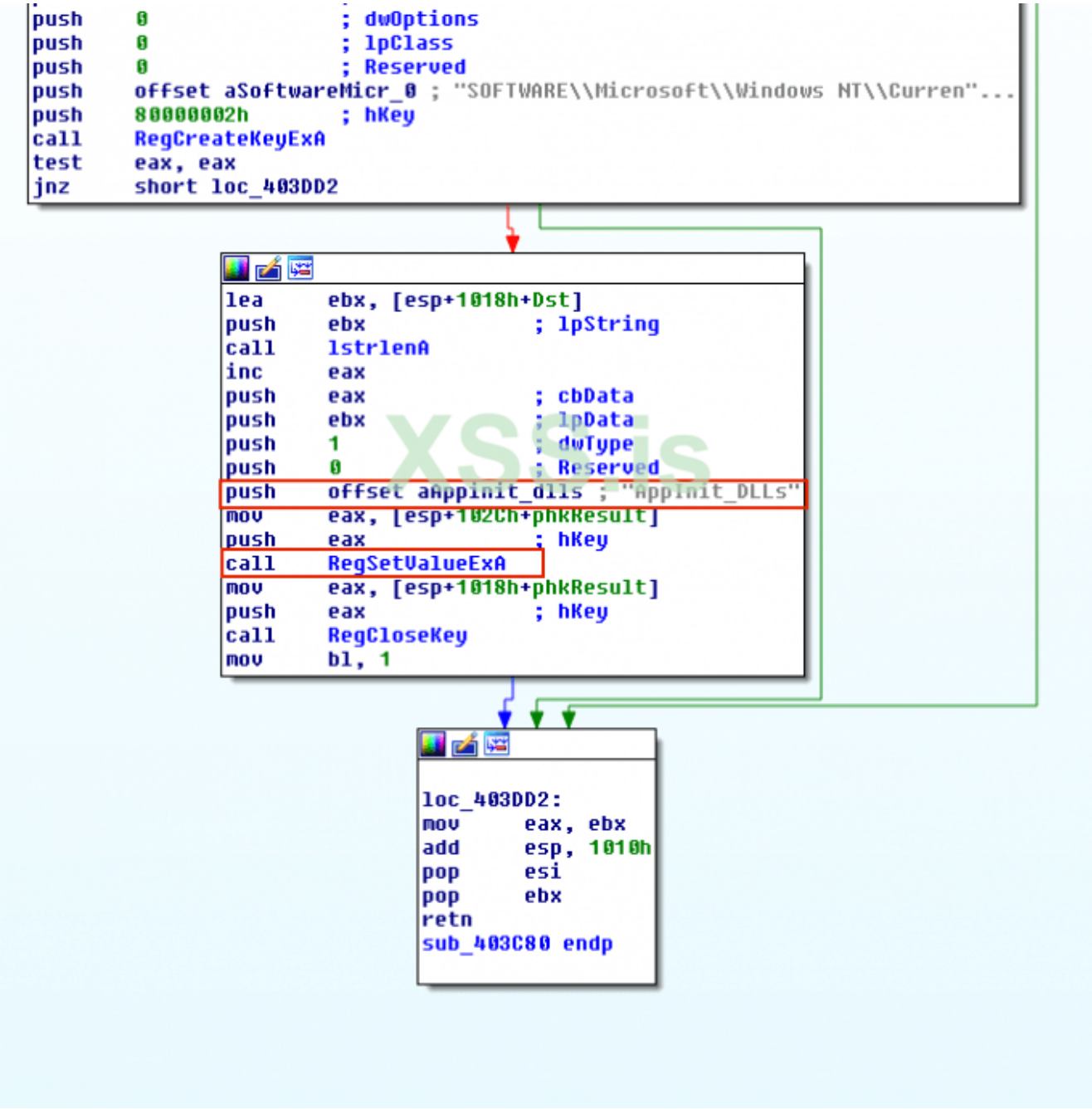
6. Инжекция и постоянство через модификацию реестра(например, APPINIT_DLLS, APPCERTDLLS, IFEO)

Appinit_DLL, AppCertDlls и IFEO (параметры выполнения файловых образов) - это все ключи реестра, которые вредоносные программы используют как для внедрения, так и для сохранения. Записи расположены в следующих местах:

HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\Appinit_Dlls
HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\Appinit_Dlls
HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls
HKLM\Software\Microsoft\Windows NT\currentversion\image file execution options

AppInit_DLLs

Вредоносное ПО может вставить местоположение своей вредоносной библиотеки в раздел реестра Appinit_Dlls, чтобы другой процесс мог загрузить свою библиотеку. Каждая библиотека в этом разделе реестра загружается в каждый процесс, который загружает User32.dll. User32.dll - очень распространенная библиотека, используемая для хранения графических элементов, таких как диалоговые окна. Таким образом, когда вредоносная программа модифицирует этот подраздел, большинство процессов загрузит вредоносную библиотеку. Рисунок 6 демонстрирует трояна Ginwui, использующего этот подход для инъекций и постоянства. Он просто открывает раздел реестра Appinit_Dlls, вызывая RegCreateKeyEx, и изменяет его значения, вызывая RegSetValueEx.

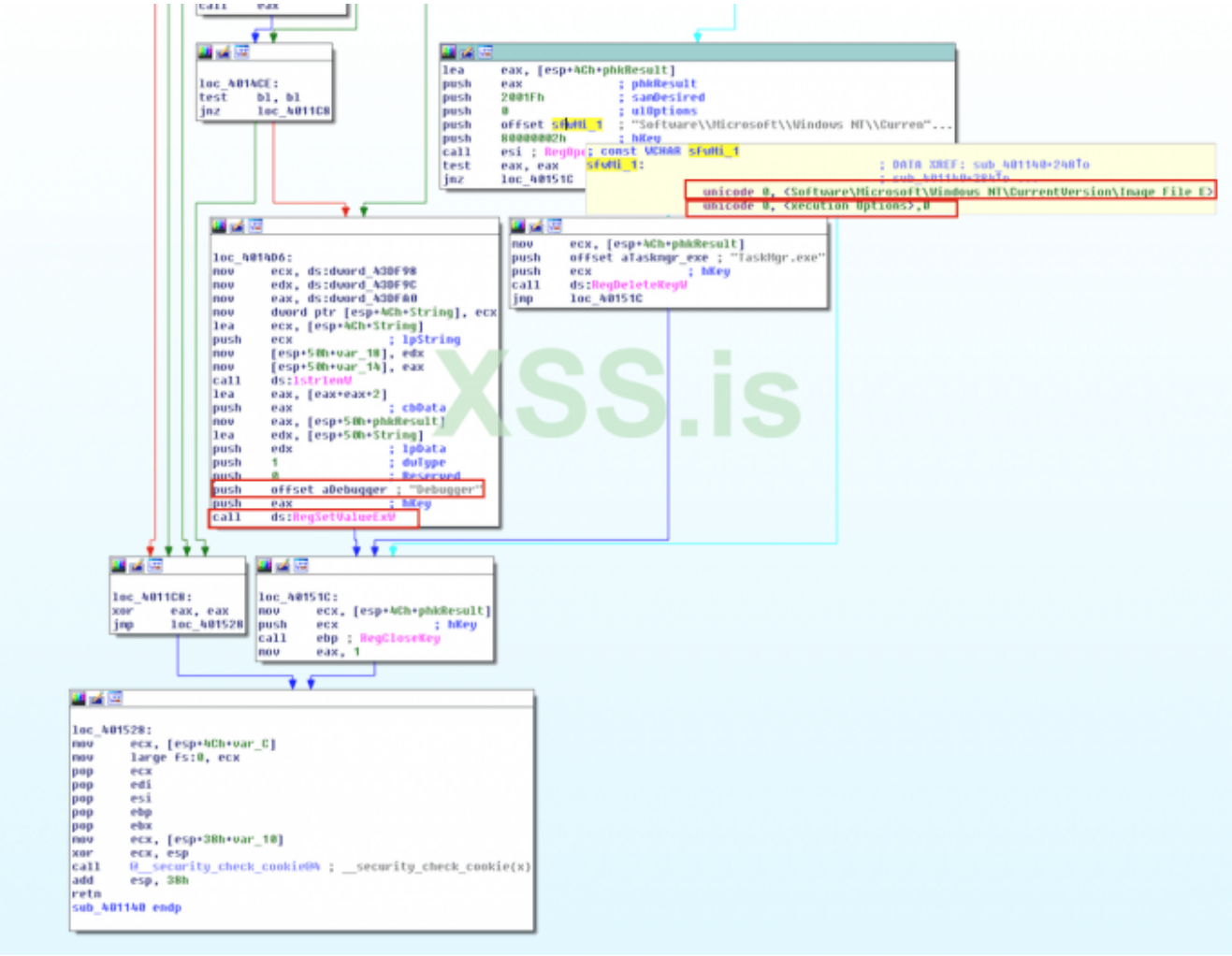


AppCertDlls

Этот подход очень похож на подход AppInit_DLLs, за исключением того, что библиотеки DLL в этом разделе реестра загружаются в каждый процесс, который вызывает функции Win32 API CreateProcess, CreateProcessAsUser, CreateProcessWithLogonW, CreateProcessWithTokenW и WinExec.

Image File Execution Options (IFEO)

IFEO обычно используется для отладочных целей. Разработчики могут установить "Значение отладчика" в этом разделе реестра, чтобы присоединить программу к другому исполняемому файлу для отладки. Поэтому всякий раз, когда исполняемый файл запускается, программа, к которой он прикреплен, запускается. Чтобы использовать эту функцию, вы можете просто указать путь к отладчику и присоединить его к исполняемому файлу, который вы хотите проанализировать. Вредоносное ПО может изменить этот раздел реестра, чтобы внедрить себя в целевой исполняемый файл. На рисунке 7 троян Diztakun реализует эту технику, изменяя значение отладчика диспетчера задач.

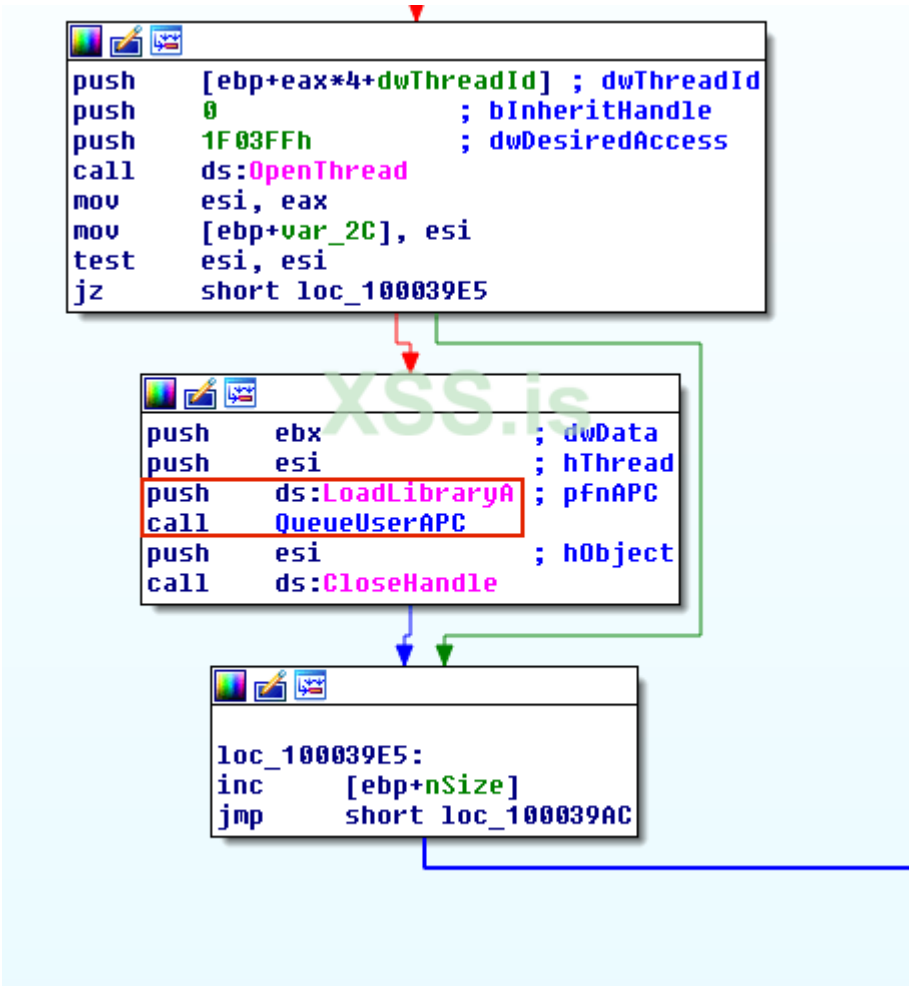


7. Инжекция APC и ATOMBOMBING

Вредоносные программы могут использовать преимущества асинхронных вызовов процедур (APC), чтобы заставить другой поток выполнить свой пользовательский код, подключив его к очереди APC целевого потока. Каждый поток имеет очередь APC, которые ожидают выполнения, когда целевой поток переходит в изменяемое состояние. Поток переходит в состояние оповещения, если вызывает функции SleepEx, SignalObjectAndWait, MsgWaitForMultipleObjectsEx, WaitForMultipleObjectsEx или

WaitForSingleObjectEx. Вредоносная программа обычно ищет любой поток, который находится в изменяемом состоянии, а затем вызывает OpenThread и QueueUserAPC, чтобы поставить APC в очередь. QueueUserAPC принимает три аргумента: 1) дескриптор целевого потока; 2) указатель на функцию, которую вредоносная программа хочет запустить; 3) и параметр, который передается указателю функции. На рисунке 8 вредоносная программа Amanahe сначала вызывает OpenThread для получения дескриптора другого потока, а затем вызывает QueueUserAPC с LoadLibraryA в качестве указателя на функцию, чтобы внедрить свою вредоносную DLL в другой поток.

AtomBombing - это метод, который был впервые введен в enSilo (http://blog.ensilo.com/atombombing-a-code-injection-that-bypasses-current-security-solutions), а затем использован в Dridex V4. Как мы подробно обсуждали в предыдущем посте, методика также основана на инъекции APC. Однако он использует таблицы атомов для записи в память другого процесса.

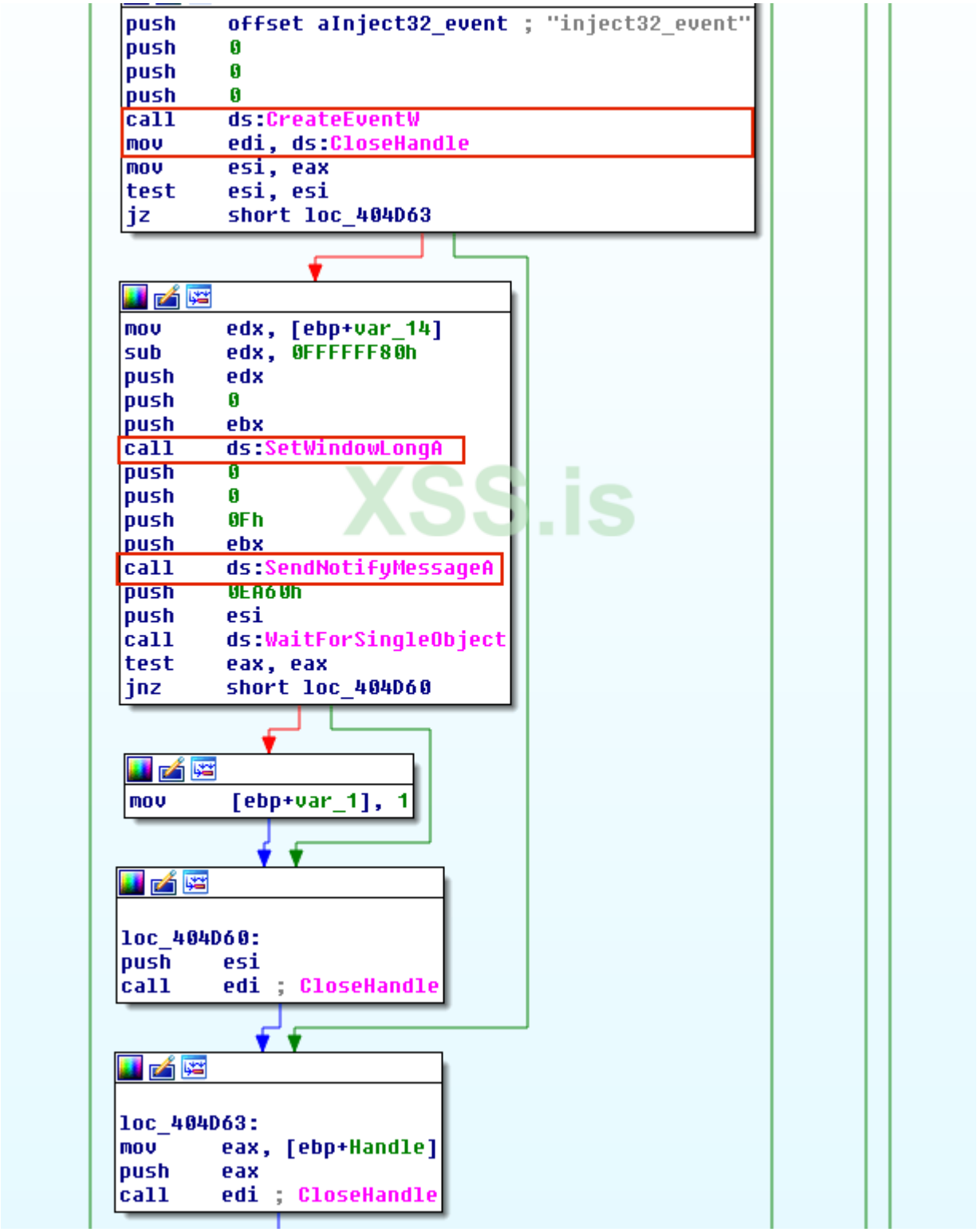


8. EXTRA WINDOW MEMORY INJECTION (EWMI) через функцию SETWINDOWLONG

EWMI опирается на инжекцию в дополнительную память окна трея Explorer'a и несколько раз использовался среди семейств вредоносных программ, таких как Gapz и PowerLoader. При регистрации класса окна приложение может указать количество дополнительных байтов памяти, называемое дополнительной памятью окна (EWM). Однако в EWM не так много места. Чтобы обойти это ограничение, вредоносная программа записывает код в общий раздел explorer.exe и использует SetWindowLong и SendNotifyMessage, чтобы указатель функции указывал на шелл-код, а затем выполнял его.

Вредоносная программа имеет два варианта записи в секцию с общим доступом. Она может либо создать общую секцию и сопоставить её с собой и другим процессом (например, explorer.exe), либо просто открыть общую секцию, которая уже существует. У первого есть издержки на выделение пространства кучи и вызов NTMapViewOfSection в дополнение к нескольким другим вызовам API, поэтому последний подход используется чаще. После того, как вредоносная программа записывает свой шелл-код в общую секцию, она использует GetWindowLong и SetWindowLong для доступа и изменения дополнительной памяти окна "Shell_TrayWnd". GetWindowLong - это API, используемый для извлечения 32-разрядного значения с указанным смещением в дополнительную память окна объекта класса окна, а SetWindowLong используется для изменения значений с указанным смещением. Делая это, вредоносная программа может просто изменить смещение указателя функции в классе окна и указать его на шелл-код, записанный в раздел общего доступа.

Как и большинство других методов, упомянутых выше, вредоносная программа должна запускать код, который она записала. В ранее обсуждавшихся методах вредоносные программы достигали этого путем вызова таких API-интерфейсов, как CreateRemoteThread, QueueUserAPC или SetThreadContext. При таком подходе вредоносная программа вместо этого запускает внедренный код, вызывая SendNotifyMessage. После выполнения SendNotifyMessage Shell_TrayWnd получает и передает управление по адресу, на который указывает значение, ранее установленное SetWindowLong. На рисунке 9 вредоносная программа с именем PowerLoader использует эту технику.

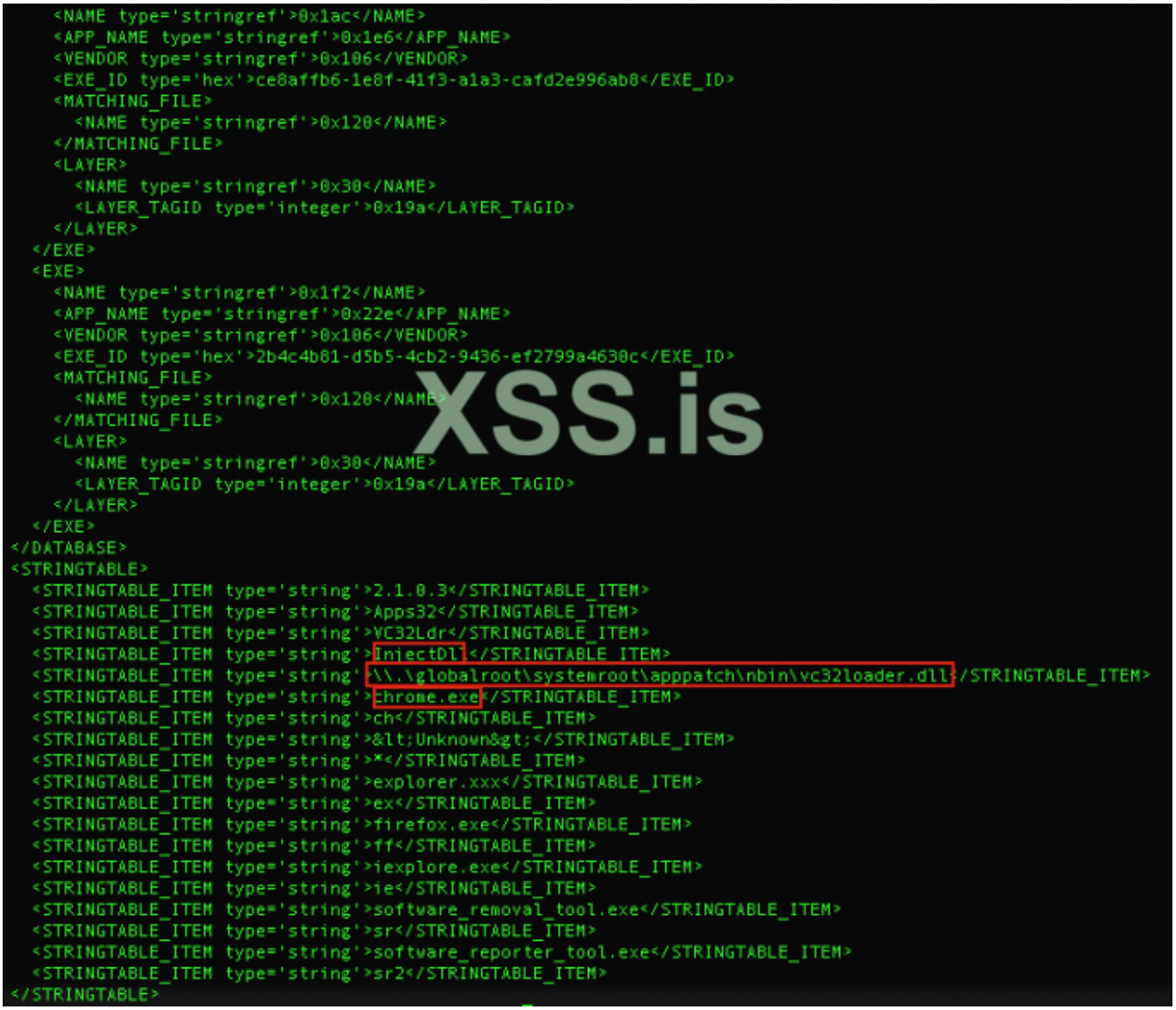


9. Инжекция с использованием SHIMS

Microsoft предоставляет Shims разработчикам в основном для обратной совместимости. Оболочки позволяют разработчикам вносить исправления в свои программы без необходимости переписывать код. Используя shims, разработчики могут рассказать операционной системе, как обращаться со своим приложением.

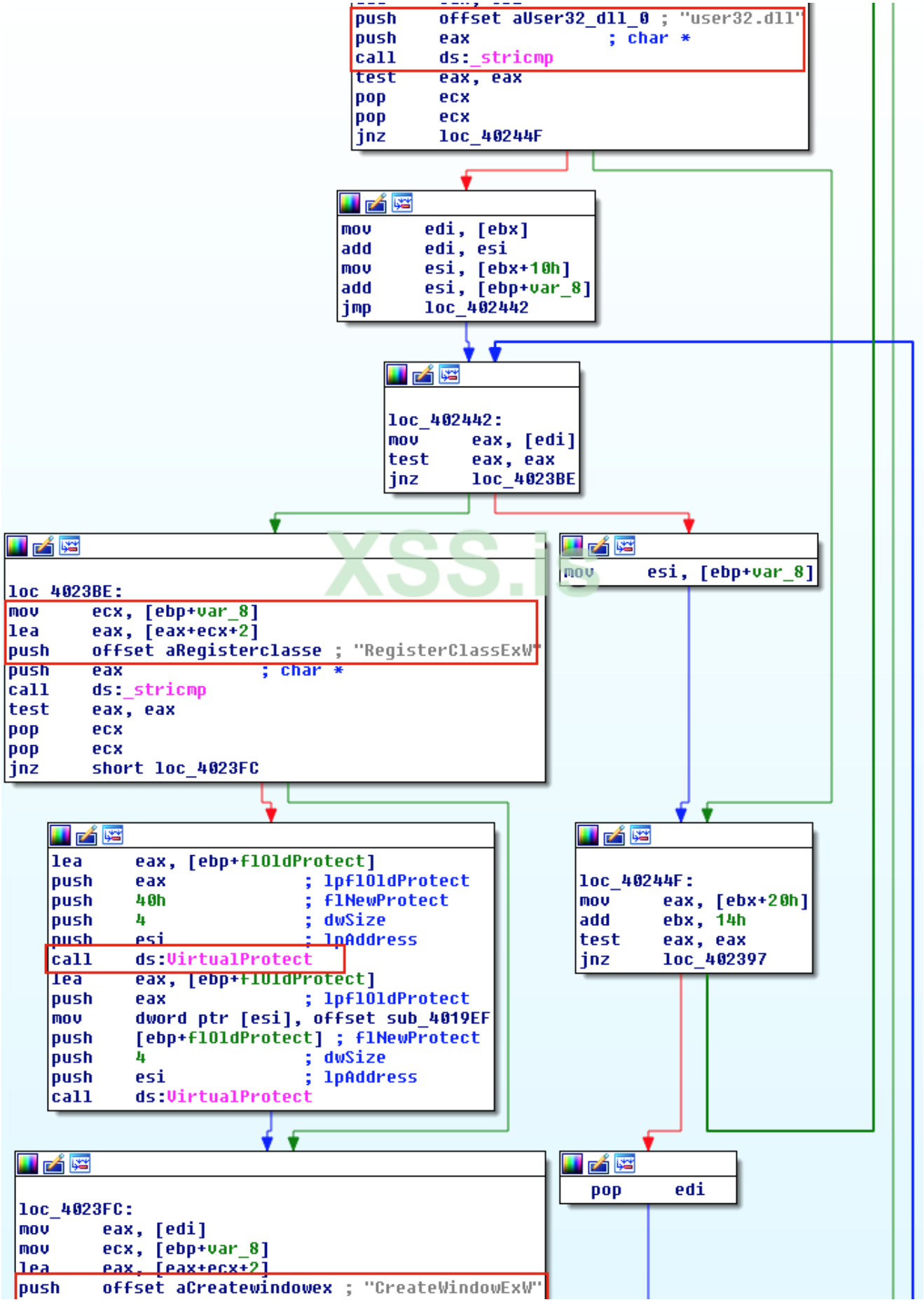
Shims по сути являются способом подключения к API и нацеливания на конкретные исполняемые файлы. Windows запускает движок Shim, когда загружает двоичный файл для проверки баз данных, чтобы применить соответствующие исправления.

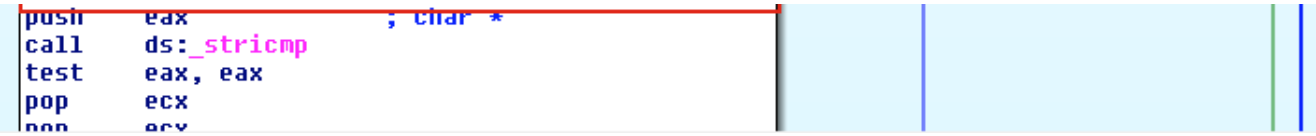
Есть много исправлений, которые можно применить, но избранные вредоносные программы - те, которые в некоторой степени связаны с безопасностью (например, DisableNX, DisableSEH, InjectDLL и так далее). Для установки базы данных shimm вредоносные программы могут использовать различные подходы. Например, один из распространенных подходов - просто запустить sdbinst.exe и указать его на вредоносный файл sdb. На рисунке 10 рекламное ПО "Search Protect by Conduit использует shim для постоянства и инжекции. Оно выполняет InjectDLL в Google Chrome для загрузки vc32loader.dll. Существует несколько инструментов для анализа файлов SDB, но для анализа SDB, перечисленных ниже, я использовал python-SDB.



10. IAT хукинг и INLINE хукинг (A.K.A руткиты пользовательского режима)

IAT-перехват и встроенный перехват обычно известны как руткиты пользовательского пространства. IAT-перехват - это метод, который вредоносная программа использует для изменения таблицы адресов импорта. Когда легитимное приложение вызывает API, расположенный в DLL, замещенная функция выполняется вместо исходной. Напротив, при встроенном перехвате вредоносное ПО изменяет саму функцию API. На рисунке 11 вредоносная программа FinFisher выполняет перехват IAT, изменяя то, куда указывает функция CreateWindowEx.


push offset aUser32_dll_0 ; "user32.dll"
push eax ; char *
call ds:_stricmp
test eax, eax
pop ecx
pop ecx
jnz loc_40244F
mov edi, [ebx]
add edi, esi
mov esi, [ebx+10h]
add esi, [ebp+var_8]
jmp loc_402442
loc_402442:
mov eax, [edi]
test eax, eax
jnz loc_4023BE
loc_4023BE:
mov ecx, [ebp+var_8]
lea eax, [eax+ecx+2]
push offset aRegisterclasse ; "RegisterClassExW"
push eax ; char *
call ds:_stricmp
test eax, eax
pop ecx
pop ecx
jnz short loc_4023FC
mov esi, [ebp+var_8]
lea eax, [ebp+flOldProtect]
push eax ; lpflOldProtect
push 40h ; flNewProtect
push 4 ; dwSize
push esi ; lpAddress
call ds:VirtualProtect
lea eax, [ebp+flOldProtect]
push eax ; lpflOldProtect
mov dword ptr [esi], offset sub_4019EF
push [ebp+flOldProtect] ; flNewProtect
push 4 ; dwSize
push esi ; lpAddress
call ds:VirtualProtect
loc_40244F:
mov eax, [ebx+20h]
add ebx, 14h
test eax, eax
jnz loc_402397
loc_4023FC:
mov eax, [edi]
mov ecx, [ebp+var_8]
lea eax, [eax+ecx+2]
push offset aCreatewindowex ; "CreateWindowExW"
pop edi



Вывод

В этой статье я рассмотрел десять различных методов, которые вредоносная программа использует, чтобы скрыть свою активность в другом процессе. В целом, вредоносное ПО либо напрямую внедряет свой шелл-код в другой процесс, либо вынуждает другой процесс загрузить свою вредоносную библиотеку. В Таблице 1 я классифицировал различные методики и предоставил образцы использованные в качестве справочного материала для наблюдения за каждым методом инъекции, описанным в этом посте. Цифры, приведенные в этом посте, помогут исследователю распознать различные методы борьбы с вредоносным ПО.

	Shellcode Injection	Forcing A DLL To Be Loaded	Sha256
1. DLL Injection		X	07b8f25e7b536f5b6f686c12d04edc37e11347c8acd5c53f98a174723078c365
2. PE Injection	X		ce8d7590182db2e51372a4a04d6a0927a65b2640739f9ec01cfd6c143b1110da
3. Process Hollowing	X		eae72d803bf67df22526f50fc7ab84d838efb2865c27aef1a61592b1c520d144
4. Thread Execution Hijacking	X		787cbc8a6d1bc58ea169e51e1ad029a637f22560660cc129ab8a099a745bd50e
5. Hook Injection		X	5d6ddb8458ee5ab99f3e7d9a21490ff4e5bc9808e18b9e20b6dc2c5b27927ba1
6. Registry Modification		X	9f10ec2786a10971eddc919a5e87a927c652e1655ddbbae72d376856d30fa27c
7. APC Injection		X	f74399cc0be275376dad23151e3d0c2e2a1c966e6db6a695a05ec1a30551c0ad
8. Shell Tray Window Injection	X		5e56a3c4d4c304ee6278df0b32afb62bd0dd01e2a9894ad007f4cc5f873ab5cf
9. Shim Injection		X	6d5048baf2c3bba85adc9ac5ffd96b21c9a27d76003c4aa657157978d7437a20
10. IAT and Inline Hooking	X	X	f827c92fbe832db3f09f47fe0dcaafd89b40c7064ab90833a1f418f2d1e75e8e

Злоумышленники и исследователи регулярно открывают новые методы для достижения инъекций и обеспечения скрытности. В этом посте подробно описаны десять распространенных методов, но есть и другие, такие как угон COM. Защитники никогда не будут "готовы" в своей миссии по обнаружению и предотвращению скрытого внедрения процесса, потому что противники никогда не прекратят вводить новшества.

Источник: https://www.elastic.co/blog/ten-pro...-technical-survey-common-and-trending-process
Автор перевода: yashechka
Переведено специально для портала XSS.is (c)