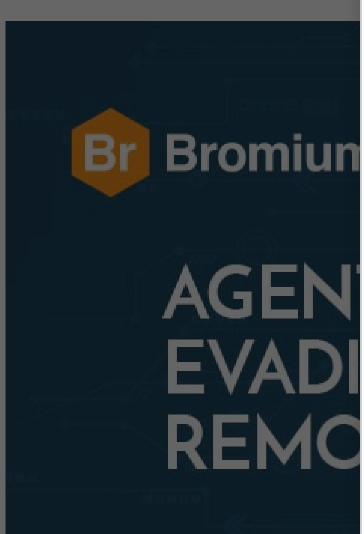
Agent Tesla: Evading EDR by Removing API Hooks

by Ratnesh Pandey on Augu



Written by Toby Gray and Ratne

Endpoint detection and respon
malicious activity that is genera
analysed to detect anomalous a
application program interface (
We recently came across a phis
Trojan. While analysing the fore
memory tampering events in the address space of ntdll.dll, the dynamic-link library (DLL) that exports the Windows Native API. The payload was isolated by Bromium Secure Platform and captured the malware.

The Agent Tesla downloader arrived as a .xls file which drops and executes the primary payload. In this blog post we cover the unhooking of APIs by the dropper to evade detection by tools such as EDR that rely on hooking. In a subsequent blog post, we provide an in-depth analysis of the campaign.

System Calls

A system call is a function in the kernel of an operating system that services requests from users and provides a barrier so that underlying high-privilege resources cannot be directly accessed by the user. On Windows systems, the ntdll.dll library contains user mode system calls. Information about these system calls are stored in an array of function pointers and the System Service Descriptor Table (SSDT).

```
//
// System Service Table Desc
//
typedef struct _KSERVICE_TAB
{
    PULONG_PTR Base;
    PULONG Count;
    ULONG Limit;
#if defined(_IA64_)
    LONG TableBaseGpOffset;
#endif
    PUCHAR Number;
} KSERVICE_TABLE_DESCRIPTOR,
```

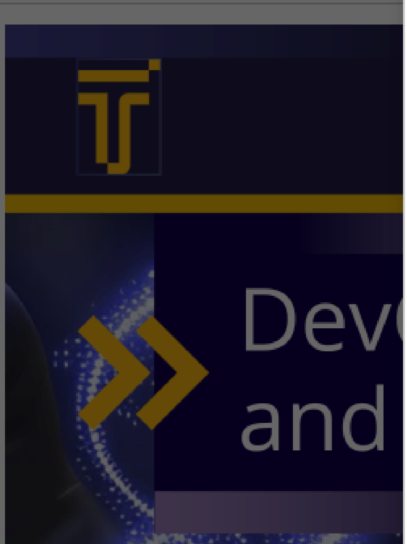


Figure 1 – The SERVICE_TABLE_

calls.

The "Base" points to the function pointer array and the system call number is an index into this array. These functions are used to request the kernel to perform some action, such as allocating virtual memory in the case of NtAllocateVirtualMemory. For the rest of this discussion we'll focus on NtProtectVirtualMemory, which is an undocumented system call that's used to change the permissions of memory.

32-bit code on 64-bit Windows

When a 32-bit program is run on a 64-bit Windows machine, it runs under a system known as Windows on Windows 64 (or WoW64 for short). Because the kernel is running in 64-bit mode, system calls from 32-bit programs all go via a wrapper function, Wow64SystemServiceCall, which is at a known location in memory. This means that ntdll.dll, which contains many system call functions, has a very repetitive structure:

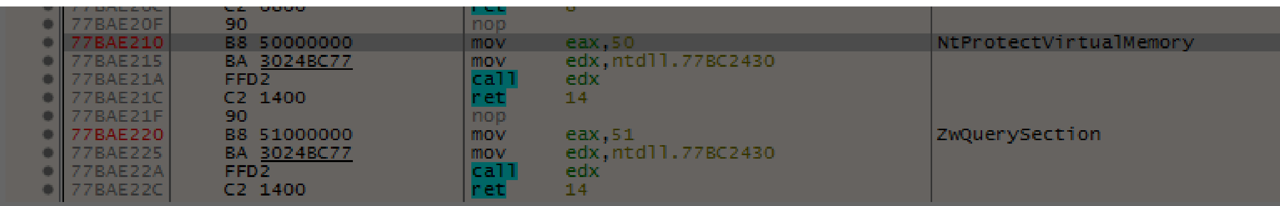


Figure 2 – Disassembly of NtProtectVirtualMemory.

The four lines of the disassembly code of NtProtectVirtualMemory are broken down as:

- Load the system call number 0x50 into the eax register
- Put the location of Wow64Transition (0x77BC2430 in the above screenshot) into the edx register
- Call the function at edx
- Return from this function

The next system call function, ZwQuerySection, is immediately after this one and follows the same structure, the only difference being loading 0x51 as the system call number rather than 0x50.

Hooking APIs

Security products use API hooking to intercept and record system API calls from software. One way of accomplishing this is to r

When hooked, the first instruct

code generated by the hooking

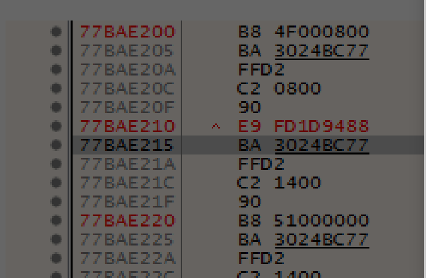


Figure 3 – Original functic

In figure 3, the first instruction

it redirects execution of the cod

have generated code at that adc

- Performs the action that hool
  - Recording the API call
  - Modifying the API call
  - Blocking the API to sto
- Performs the replaced instruc
- Jumps execution back to the

As the original function then continues execution as usual, neither the code calling the API nor the system kernel are aware that the function call has been intercepted.

Malware Unhooking API Hooks

Email Header

- From: Alhaji Nasiru <sales@gossipnewspro.info >
- To: <–<Redacted>–.com>
- Subject: New Purchase Order for August
- Date: Sun, 28 Jul 2019 16:41:52 -0700
- Attachment: Signeded-revised-PI.xls

Downloader

- Filename: Signeded-revised-PI.xls
- Size: 82 KB (83968 bytes)
- MD5: C081E4AA1FBEC4857E88E4FBF91FE90E
- SHA-1: 1F6527CBD8BC83132A89C4F66A897A576259C4A1
- SHA-256: 42BD54E60C86AE02BCD9BCD02FA82C9D77D831F3EED77DD924E2E6976B9A5808

Dropper

- Filename: v4bc6f.exe [Win32.Trojan.Injector]
- File size: 936 KB (958464 bytes)
- MD5: 97BD950CA1FBD49A632A876A05E7ACEF
- SHA-1: 6FD6E4B676BD363B817F54F067684A14BA31E053
- SHA-256: 851AC0EF0956156EFCDDDB15288A6DF82009940D58F851D006732675F3B9AD1D

Modern malware typically relies on polymorphism and obfuscation techniques to evade static detection by signature-based detection technologies such as anti-virus. EDR tools work differently by monitoring system activity and flagging suspicious events if there is a deviation from normal application behaviour or if there is a match against known malicious patterns. Most of these events are generated by hooking APIs. Some security solutions also use API hooks to block malicious processes if a suspicious event is triggered.

When analysing this malicious sample we noticed some unique code that was modifying the memory-mapped ntdll.dll before launching its payload, Agent Tesla (bin.exe). The malware allocates the shellcode and then performs the following actions:

- Call NtProtectVirtualMemory in ntdll.dll's address space to change its memory permissions of the region to PAGE_EXECUTE_READWRITE
- Removes the API hooks from
- Call NtProtectVirtualMemory PAGE_EXECUTE_READ
- After removing the hooks, it e

Figure 4

The malicious code loads the a scanning through the memory 0x004A0A50 is incrementing the ntdll.dll to examine.

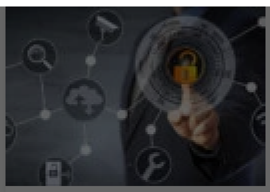
The first check at 0x004A0A51 is described previously, so skip over that comparison and jump to 0x004A0A6B. The check at 0x004A0A6B is for the value of Wow64Transition and if it's found then the instructions starting at 0x004A0A6F are performed. In sequence these are:

- Write the value in eax (which is a counter for the system call number, so 0x50 in the case of NtProtectVirtualMemory) out to 5 bytes before the location of the value of Wow64Transition.
- Write the byte 0xB8 out to 6 bytes before the location of the value of Wow64Transition.
- Increment the value in eax, moving the system call number onto the next value

The result of writing out the 5 bytes (4 for eax and one for 0xB8) is to replace any hooking instruction (such as jmp 0x004F0012 in the previous example) with the original instruction that was there (which is mov eax, 50 in the previous example).

The end result is that the malicious code can now call system APIs, safe in the knowledge that its requests won't be monitored or blocked by any hooks.

This unhooking isn't an issue for Bromium Secure Platform as the malicious activity will still all be contained inside a micro-virtual machine (uVM) where hardware-backed isolation is used for protection.



UnitedHealth Hires Longtime Cybersecurity Executive as CISO

Small Businesses Boosting Cybersecurity as Threats Grow: ITRC

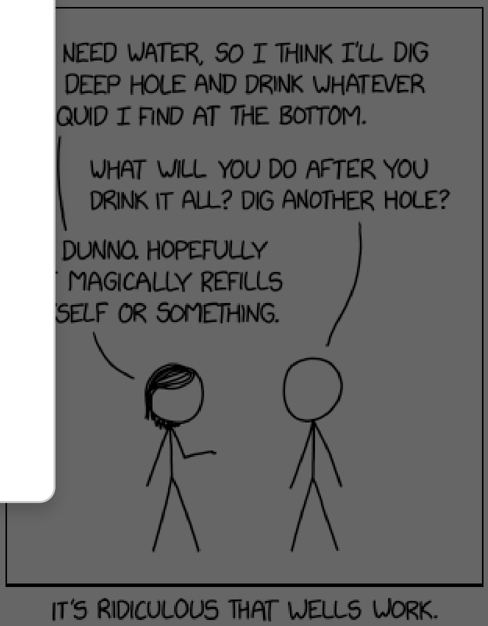
Top Stories »

GreyNoise: AI's Central Role in Detecting Security Flaws in IoT Devices

Microsoft's Controversial Recall Feature Release Delayed Again

CISA Strategic Plan Targets Global Cooperation on Cybersecurity

urity Humor »



Randall Munroe's XKCD 'Wells'

Download Free eBook

Agent Tesla Payload

- Filename: bin.exe [ByteCode-MSIL.Spyware.Ielib]
- File size: 331.5 KB (339456 bytes)
- MD5: 640CA1048F2AED048CB209234FA080B9
- SHA-1: 58790A758B31E80648DB288BA86F49F7DC05D89B
- SHA-256: 53997AF9CF992BF7A97E54F79A1474A1C0023133D7B97B861A278BAA238C9421

The post Agent Tesla: Evading EDR by Removing API Hooks appeared first on Bromium.

Recent Articles By Author

- Ransomware Goes Fileless, Uses Malicious Documents and PowerShell to Encrypt Files
- Reawakening of Emotet: An Analysis of its JavaScript Downloader
- Dridex's Bag of Tricks: An Analysis of its Masquerading and Code Injection Techniques

More from Ratnesh Pandey

Agent Tesla, api, EDR, hooking, ntdll.

← Movie Tickets Service Expose

The