

DECEMBER 4-7, 2017 EXCEL / LONDON, UK



Tal Liberman
Eugene Kogan





About Us



- @Tal Liberman
 - Security Research Team Leader @ enSilo
 - Reverse Engineering, Research, Low Level Expert
 - Author on **BreakingMalware**
- Eugene Kogan
 - Principal Development Lead @ enSilo
 - Former Tech Lead @ Imperva
 - Kernel Expert



Overview

- Brief history of evasion techniques
- AV scanners
- Transacted NTFS (TxF)
- Evolution of Windows process loader
- Doppelgänging execution flow (+ live demo)
- "Mitigation in Redstone" The Story of a BSOD



Brief History of Evasion Techniques

- Advanced Code Injections Overview
 - GhostWriting
 - AtomBombing
 - PowerLoader + PowerLoaderEx
 - PROPagate
 - ...
- Reflective Loading
- Process Hollowing



GhostWriting



A paradox: Writing to another process without opening it nor actually writing to it

- Injection method from over 10 years ago
- Has never received much attention
- Inject arbitrary code into explorer.exe without:
 - OpenProcess
 - WriteProcessMemory
 - CreateRemoteThread



GhostWriting



A paradox: Writing to another process without opening it nor actually writing to it

- Find 2 patterns in NTDLL
 - Move pattern
 - mov [REG1], REG2 ; mov [eax], ebx
 - ret
 - Jmp pattern
 - jmp 0x0 ;(eb fe)
- Write-What-Where(What, Where)
 - SetThreadContext(...):
 - EIP=Move pattern
 - ESP=NewStack
 - REG1=Where
 - REG2=What

Original post by c0de90e7: http://blog.txipinet.com/2007/04/05/69-a-paradox-writing-to-another-process-without-openning-it-nor-actually-writing-to-it/



GhostWriting



A paradox: Writing to another process without opening it nor actually writing to it

- Using write-what-where:
 - Write shellcode to stack
 - Write VirtualProtect parameters to stack
- Using SetThreadContext:
 - Call VirtualProtect
 - Call shellcode



AtomBombing

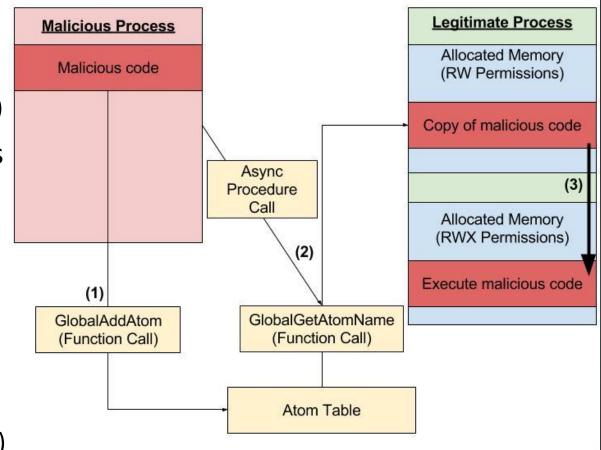


- Injection technique we published in October 2016
- Exploits the global atom table and APCs
- Used in the wild by Dridex



AtomBombing – Write-What-Where

- GlobalAddAtom
- NtQueueApcThread(..., GlobalGetAtomNameW, ...)
- Copy code to RW memory in target process
- Copy ROP chain to target process
- ROP chain
 - ZwAllocateVirtualMemory(..., RWX, ...);
 - memcpy(RWX, RW, ...);
 - Shellcode()
- Initiate ROP chain
 - NtQueueApcThread(..., NtSetContextThread, ...)





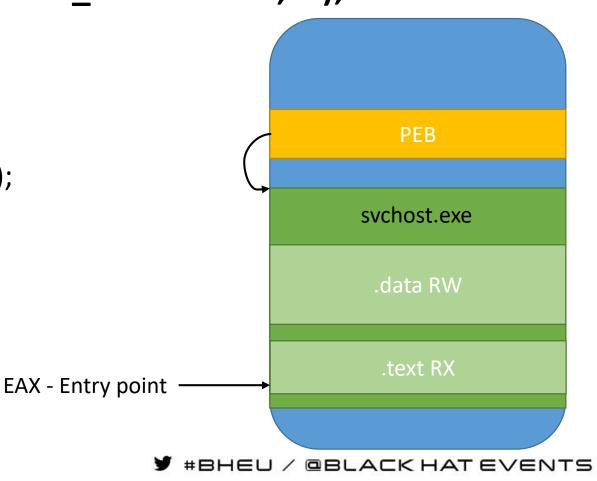


- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);





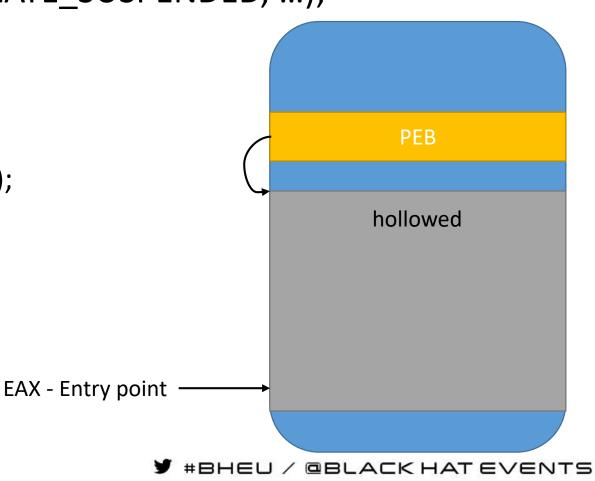
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







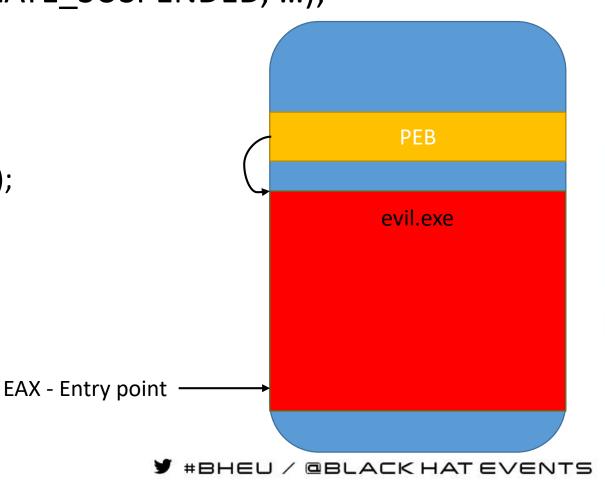
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







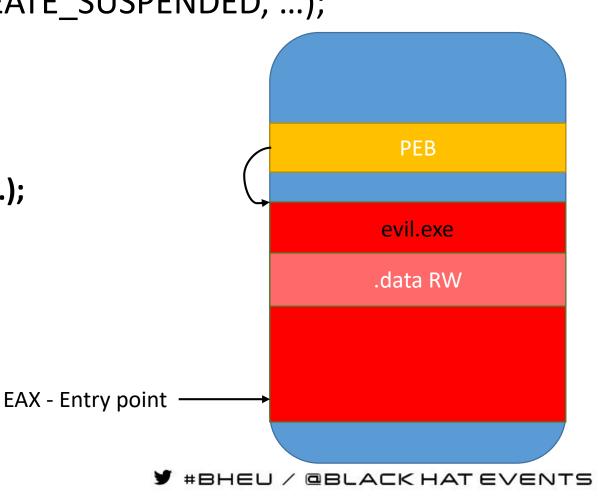
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







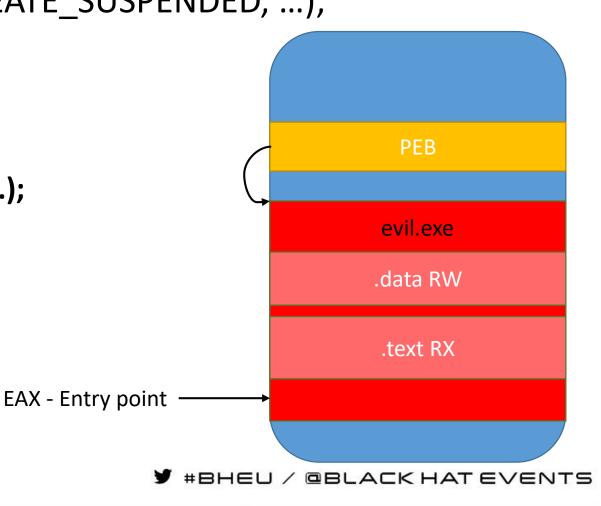
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







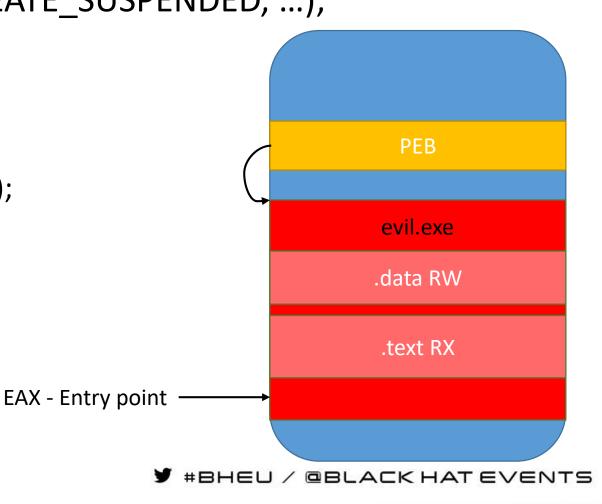
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







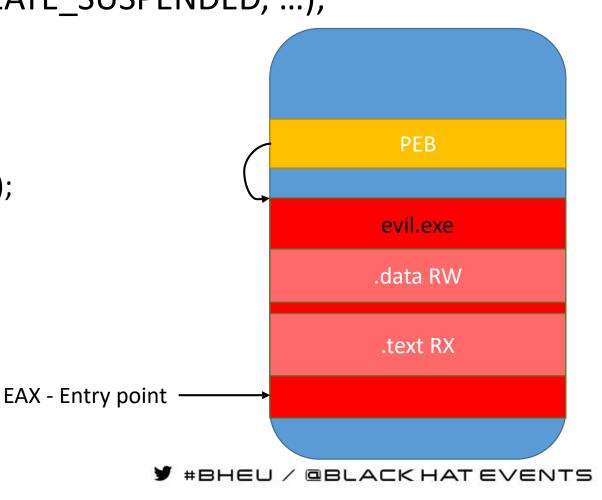
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







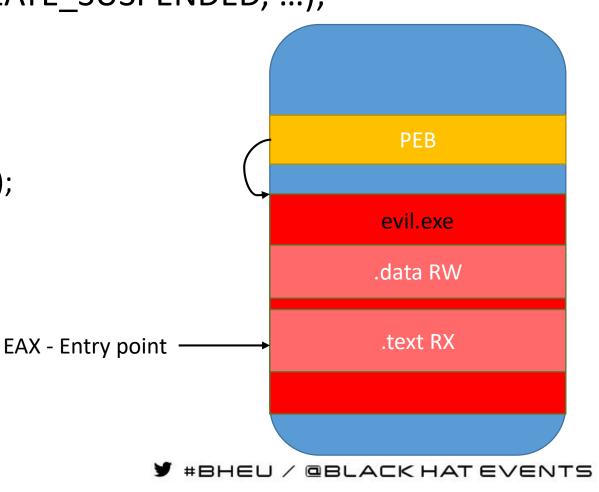
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







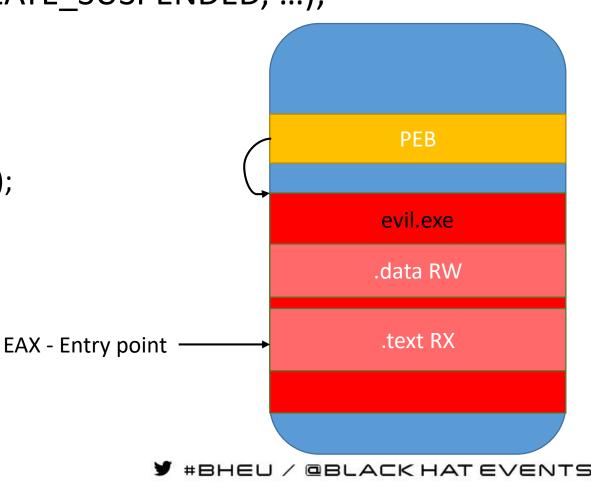
- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);







- CreateProcess("svchost.exe", ..., CREATE_SUSPENDED, ...);
- NtUnmapViewOfSection(...);
- VirtualAllocEx(...);
- For each section:
 - WriteProcessMemory(..., EVIL_EXE, ...);
- Relocate Image*
- Set base address in PEB*
- SetThreadContext(...);
- ResumeThread(...);





Process Hollowing - Issues



- The most trivial implementations create an image that is entirely RWX
 - Easy to detect in numerous ways
- Unmap and VirtualAllocEx/NtAllocateVirtualMemory() with correct protection
 - Unmapping of main module is highly suspicious
 - ETHREAD.Win32StartAddress → VadType != VadImageMap
- Overwrite original executable without unmapping
 - _MMPFN.u4.PrototypePte == 0 (0 means private/not shared, should be 1 shared)
 - If not paged cause page in
 - In forensics PTE.u.Soft.PageFileHigh != 0
- Unmap and remap as non image
 - Vad.Flags.VadType != VadImageMap
- Unmap and remap as image
 - ETHREAD.Win32StartAddress != Image.AddressOfEntryPoint
 - EPROCESS.ImageFilePointer != VAD(ETHREAD.Win32StartAddress).Subsection.ControlArea.FilePointer *
 - On Win < 10 EPROCESS.SectionObject



Quick Recap



- Process hollowing not so great anymore
- Rest of techniques
 - Missing file mapping
 - Suspicious
- We need something new
- Wouldn't it be cool if we could create a fileless mapped file?
- But AVs scan files
 - We need to understand how scanners work

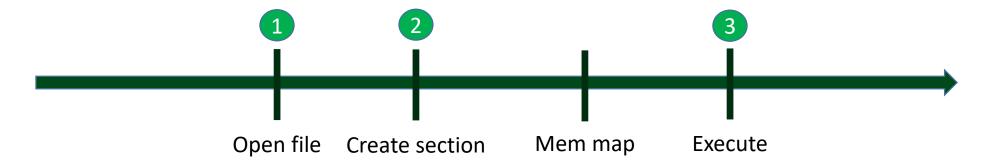






AV Scanners – Scan on execute

File execution timeline



- Where to intercept?
 - 1. Minifilter File open/create
 - 2. Minifilter IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
 - 3. Process create notify routine (executables only)



AV Scanners – Challenges



- How to open the file for scanning?
 - From User mode / Kernel
 - By File name/ FileId / using existing file object
- Rescan on each change is not practical
- Scan file before the execution
 - File content be altered before execution begins

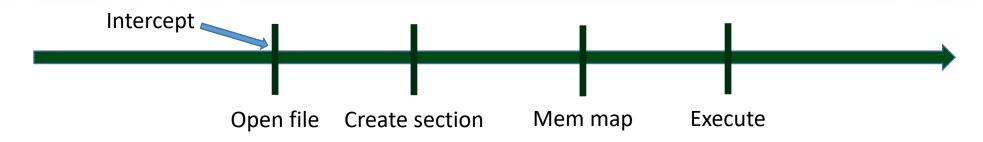












Block during file open (partial stack)

AV Blocks here

```
FLTMGR!FltpPerformPostCallbacks+0x2a5
nt!ObOpenObjectByNameEx+0x1dd
nt!IoCreateFileEx+0x115
nt!NtCreateUserProcess+0x431
------ Kernel mode -----
ntdll!NtCreateUserProcess+0x14
```





Scan intercepted file while blocked (partial stack)

nt!ObpLookupObjectName+0x8b2

nt!ObOpenObjectByNameEx+0x1dd

FLTMGR!FltCreateFile+0x8d

AV minifilter code here

FLTMGR!FltpDispatch+0xe9

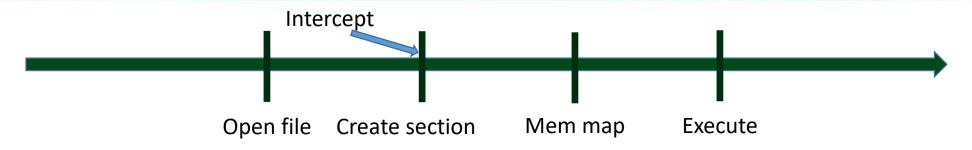
nt!IopXxxControlFile+0xd9c

nt!NtDeviceIoControlFile+0x56

nt!KiSystemServiceCopyEnd+0x13







Block during ACQUIRE_FOR_SECTION_SYNC...

AV Blocks here

FLTMGR!FltpPerformPreCallbacks+0x2ea
nt!FsRtlAcquireToCreateMappedSection+0x4e
nt!FsRtlCreateSectionForDataScan+0xa6
FLTMGR!FltCreateSectionForDataScan+0xec
WdFilter!MpCreateSection+0x138



AV Scanners - ACQUIRE_FOR_SECTION_SYNC

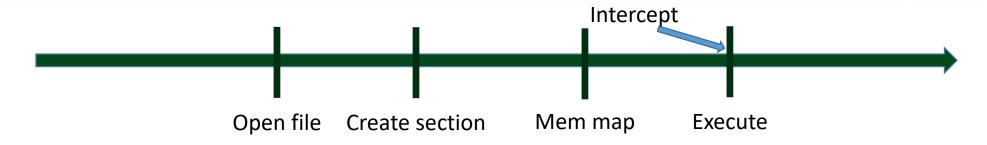
- Flags are misleading
 - SEC IMAGE unavailable
 - Possible to pass PAGE_READONLY

Data Or Executable?

```
typedef union _FLT_PARAMETERS {
    ... ;
struct {
    FS_FILTER_SECTION_SYNC_TYPE SyncType;
    ULONG POINTER_ALIGNMENT PageProtection;
} AcquireForSectionSynchronization;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
PAGE_READONLY
```







Block during process creation partial stack

AV Blocks here

```
nt!PspCallProcessNotifyRoutines+0x1cf
nt!PspInsertThread+0x5ea
nt!NtCreateUserProcess+0x8be
------ Kernel mode -----
ntdll!NtCreateUserProcess+0x14
KERNEL32!CreateProcessWStub+0x53
```



AV Scanners – Process Notification

- PsSetCreateProcessNotifyRoutine<u>Ex</u> available Windows Vista SP1+
 - Can be achieved in other ways SSDT (XP remember?)
- Available only for main executable
 - Not useful for DLL loading
 - Blind to process hollowing



AV Scanners – Summary



- It is not an easy job to create an AV
- Performance vs coverage tradeoff
 - How often files are opened and sections are mapped
- Variety of operating systems and file systems
 - From XP to Win 10
 - Different CPUs 32 bit and 64 bit
 - FAT, NTFS, Network
- Not complicated enough?



NTFS Transactions



Transactional

NTFS



NTFS Transactions - Facts



- A.K.A. TxF
- Introduced in Windows Vista
- Implemented in NTFS driver (Kernel)
 - For local disks
- Microsoft proposed use cases: Files update or DTC
- Simplifies handling of a rollback after multiple file changes
 - For example during installation process



NTFS Transactions - Facts



- Taken from Storage Developer conference 2009:
 - TxF accounts for ~30% of NTFS driver size on AMD64
 - MSDN lists 19 new Win32 *Transacted() APIs
 - 22 file I/O APIs whose behavior is affected by TxF
- Deprecated on arrival
- Still used today (almost 11 years later)



NTFS Transactions – API examples

- Application explicitly uses transactions
- CreateTransaction()
- CommitTransaction(), RollbackTransaction()
- CreateFileTransacted(), DeleteFileTransacted(),
 RemoveDirectoryTransacted(), MoveFileTransacted()
- Most functions that work with handles should work with transactions



NTFS Transactions – Usage Example

- hTransaction = CreateTransaction(NULL, NULL, 0, 0, 0, 0, NULL);
- hFile = CreateFileTransacted(FILE_NAME, hTransaction);
- WriteFile(hFile);
- CloseHandle(hFile);
- CommitTransaction(hTransaction);
- CloseHandle(hTransaction);



Quick Recap



What we have so far?





Quick Recap



- Naturally, transactions make life hard for AV vendors
- We want to create a process from transacted file
- However process creation does not support transacted files directly
- We need dive into process creation on Windows to find a way to do it



Windows Process Loader Evolution

- Comparing kernel32!CreateProcessW between XP and 10 gives the impression that MS completely changed how processes are created
- A deeper examination shows that Microsoft simply moved most of the code from kernel32 to ntoskrnl (and somehow the function in kernel32 became longer)
- Logically the steps remain mostly the same, at least for our purposes



Process Loader Evolution – XP



```
💶 🚄 🖼
                         loc 7C81D0C9:
                                                  ; OpenOptions
                                 FILE SHARE READ or FILE SHARE DELETE; ShareAccess
  7C81D0CD 8D 85 DC F8 FF+lea
                                                 ; IoStatusBlock
  7C81D0D4 8D 85 30 F8 FF+lea
                                 eax, [ebp+ObjectAttributes]
                                                 ; ObjectAttributes
                                 FILE EXECUTE or SYNCHRONIZE; DesiredAccess
                                 eax, [ebp+FileHandle]
  7C81D0E0 8D 85 88 F9 FF+lea
                                                 ; FileHandle
                                 [ebp+Status], eax
  7C81D0E9 89 85 AC F8 FF+mov
                                 eax, ebx
  7C81D0F1 0F 8D 37 B4 FF+jge
                                  loc 7C81852E
                                       Hidden blocks
💶 🚄 🚾
                                               ; FileHandle
                       loc 7C81853A:
                               [ebp+FileHandle]
7C81853A FF B5 88 F9 FF+push
                                               ; AllocationAttributes
                                               ; SectionPageProtection
                       push
                                               ; MaximumSize
                                               ; ObjectAttributes
                               SECTION_ALL_ACCESS ; DesiredAccess
                                               ; SectionHandle
                               ds: imp NtCreateSection@28 ; NtCreateSection(x,x,x,x,x,x,x)
                               edi, eax
                               [ebp+Status], edi
                               edi, ebx
                               short loc 7C81858C
```



Kernel

Process Loader Evolution – XP



CreateProcessW

- CreateProcessInternalW
 - NtOpenFile Open image file
 - NtCreateSection Create section from opened image file
 - NtCreateProcessEx Create process from section
 - PspCreateProcess Actually create the process
 - ObCreateObject Create the EPROCESS object
 - Add process to list of processes
 - BasePushProcessParameters Copy process parameters
 - RtlCreateProcessParameters Create process parameters
 - NtAllocateVirtualMemory Allocate memory for process parameters
 - NtWriteVirtualMemory Copy process parameters to allocated memory
 - NtWriteVirtualMemory Write address to PEB.ProcessParameters
 - RtlDestroyProcessParameters Destroy process parameters
 - BaseCreateStack Create Stack for process
 - NtCreateThread Create main thread
 - NtResumeThread Resume main thread



Process Loader Evolution – 10



```
eax ; DriverContext
                       xor
                               eax, eax
                               eax ; Options
                               eax ; InternalParameters
                               eax ; CreateFileType
                               eax ; EaLength
                               eax ; EaBuffer
                               FILE SYNCHRONOUS IO NONALERT or FILE NON DIRECTORY FILE; CreateOptions
                               edi ; Disposition
                               FILE SHARE READ or FILE SHARE DELETE; ShareAccess
                               esi ; FileAttributes = FILE ATTRIBUTE NORMAL
                       push
                               eax : AllocationSize
                               eax ; IoStatusBlock
                               eax ; ObjectAttributes
                       push
                               FILE EXECUTE or SYNCHRONIZE; DesiredAccess
                               eax ; FileHandle
                               IoCreateFileEx@60 ; IoCreateFileEx(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x)
                                             Hidden blocks
 mov
                          push
                                 [ebp+R v CreateProcessContext.R image_file_handle_64] ; R_p_FileHandle
                         push
                                 [ebp+R_v_CreateProcessContext.R_new_process_token_object_5C]; int
                         push
                                 edx, [ebp+ObjectAttributes]; R_p_ObjectAttributes
                                 ecx, [ebp+R_v_CreateProcessContext.R_special_image_section_handle_6C] ; SectionHandle
                          lea
                          call
                                 MmCreateSpecialImageSection@24 ; MmCreateSpecialImageSection(x,x,x,x,x,x)
                         mov
                                 ebx, eax
                          push
                                 esi ; HandleInformation
                          test
                                 ebx, ebx
                                 loc 81B1C4CA
```



ackhat Process Loader Evolution -10



- CreateProcessW
 - CreateProcessInternalW
 - BasepCreateProcessParameters Create process parameters
 - RtlCreateProcessParametersEx Create process parameters
 - NtCreateUserProcess Create process from file
 - PspBuildCreateProcessContext Build create process context
 - IoCreateFileEx Open image file
 - MmCreateSpecialImageSection Create section from image file
 - PspCaptureProcessParams Copy process parameters from user mode
 - PspAllocateProcess Create process from section
 - ObCreateObject Create EPROCESS object
 - MmCreatePeb Create PEB for process
 - PspSetupUserProcessAddressSpace Allocate and copy process
 - KeStackAttachProcess Attach to process memory
 - ZwAllocateVirtualMemory Allocate memory for process parameters
 - PspCopyAndFixupParameters Copy process parameters to process
 - Memcpy
 - Set PEB.ProcessParameters
 - KiUnstackDetachProcess Detach from process memory
 - PspAllocateThread Create thread
 - PspInsetProcess Insert process to list of processes
 - PspInsertThread Insert thread to list of threads
 - PspDeleteCreateProcessContext Delete process create context
 - RtlDestroyProcessParameters Delete process parameters
 - NtResumeThread Start main thread







Windows Process Loader Evolution

- NtCreateUserProcess used instead of NtCreateProcessEx
- NtCreateProcessEx receives a handle to a section
- NtCreateUserProcess receives a file path
- NtCreateProcessEx still available used in creation of minimal processes (nt!PsCreateMinimalProcess)
- All the supporting user-mode code is not available post XP
 - We need to implement it ourselves



Doppelgänging - Motivation



- Load and execute arbitrary code
- In context of legitimate process
- None of the suspicious process hollowing API calls
 - NtUnmapViewOfSection
 - VirtualProtectEx
 - SetThreadContext
- AV will not scan at all / AV will scan "clean" files only
- Will not be discovered by advanced forensics tools



Doppelgänging - Overview



- We break Doppelgänging into 4 steps:
 - Transact Overwrite legitimate executable with a malicious one
 - Load Load malicious executable
 - Rollback Rollback to original executable
 - Animate Bring the Doppelgänger to life





- Create a transaction
 - hTransaction = CreateTransaction(...);
- Open a "clean" file transacted
 - hTransactedFile = CreateFileTransacted("svchost.exe",
 GENERIC_WRITE | GENERIC_READ, ..., hTransaction, ...)
- Overwrite the file with malicious code
 - WriteFile(hTransactedFile, MALICIOUS_EXE_BUFFER, ...);





- Create a transaction
 - hTransaction = CreateTransaction(...);
- Open a "clean" file transacted
 - hTransactedFile = CreateFileTransacted("svchost.exe",
 GENERIC_WRITE | GENERIC_READ, ..., hTransaction, ...)
- Overwrite the file with malicious code
 - WriteFile(hTransactedFile, MALICIOUS_EXE_BUFFER, ...);





- Create a transaction
 - hTransaction = CreateTransaction(...);
- Open a "clean" file transacted
 - hTransactedFile = CreateFileTransacted("svchost.exe",
 GENERIC_WRITE | GENERIC_READ, ..., hTransaction, ...)
- Overwrite the file with malicious code
 - WriteFile(hTransactedFile, MALICIOUS_EXE_BUFFER, ...);

File svchost.exe





- Create a transaction
 - hTransaction = CreateTransaction(...);
- Open a "clean" file transacted
 - hTransactedFile = CreateFileTransacted("svchost.exe", GENERIC WRITE | GENERIC READ, ..., hTransaction, ...)
- Overwrite the file with malicious code
 - WriteFile(hTransactedFile, MALICIOUS_EXE_BUFFER, ...);





Doppelgänging - Load



- Create a section from the transacted file
 - NtCreateSection(&hSection, ..., PAGE_READONLY, SEC_IMAGE, hTransactedFile);
- The created section will point to our malicious executable

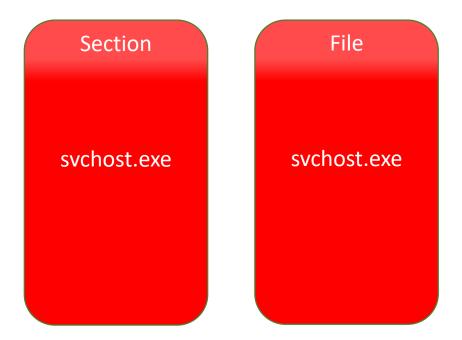




Doppelgänging - Load



- Create a section from the transacted file
 - NtCreateSection(&hSection, ..., PAGE_READONLY, SEC_IMAGE, hTransactedFile);
- The created section will point to our malicious executable

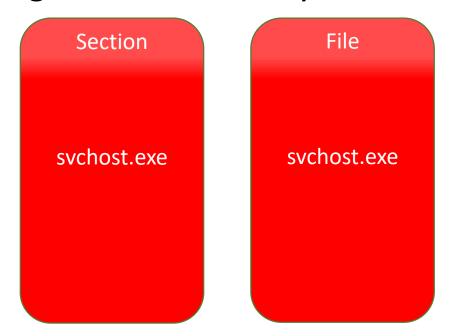




Doppelgänging - Rollback



- Rollback the transaction
 - RollbackTransaction(hTransaction);
- Effectively removes our changes from the file system

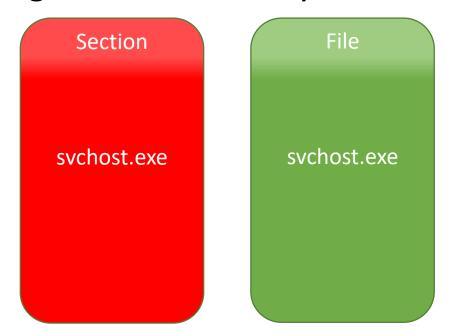




Doppelgänging - Rollback



- Rollback the transaction
 - RollbackTransaction(hTransaction);
- Effectively removes our changes from the file system

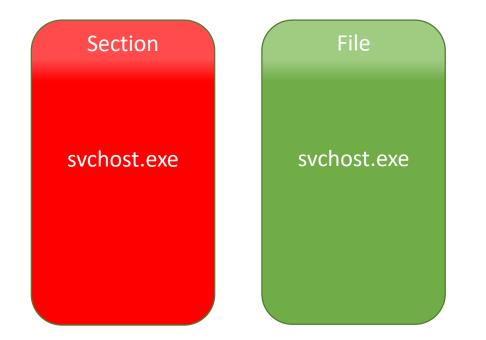




Doppelgänging - Animate



- Create process and thread objects
 - NtCreateProcessEx(&hProcess, ..., hSection, ...);
 - NtCreateThreadEx(&hThread, ..., hProcess, MALICIOUS_EXE_ENTRYPOINT, ...);

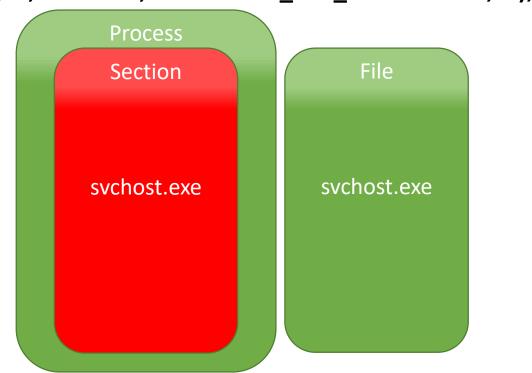




Doppelgänging - Animate



- Create process and thread objects
 - NtCreateProcessEx(&hProcess, ..., hSection, ...);
 - NtCreateThreadEx(&hThread, ..., hProcess, MALICIOUS_EXE_ENTRYPOINT, ...);





Doppelgänging - Animate



- Create process and thread objects
 - NtCreateProcessEx(&hProcess, ..., hSection, ...);
 - NtCreateThreadEx(&hThread, ..., hProcess, MALICIOUS_EXE_ENTRYPOINT, ...);
- Create process parameters
 - RtlCreateProcessParametersEx(&ProcessParams, ...);
- Copy parameters to the newly created process's address space
 - VirtualAllocEx(hProcess, &RemoteProcessParams, ..., PAGE_READWRITE);
 - WriteProcessMemory(hProcess, RemoteProcessParams, ProcessParams, ...);
 - WriteProcessMemory(hProcess, RemotePeb.ProcessParameters, &RemoteProcessParams, ...);
- Start execution of the doppelgänged process
 - NtResumeThread(hThread, ...);



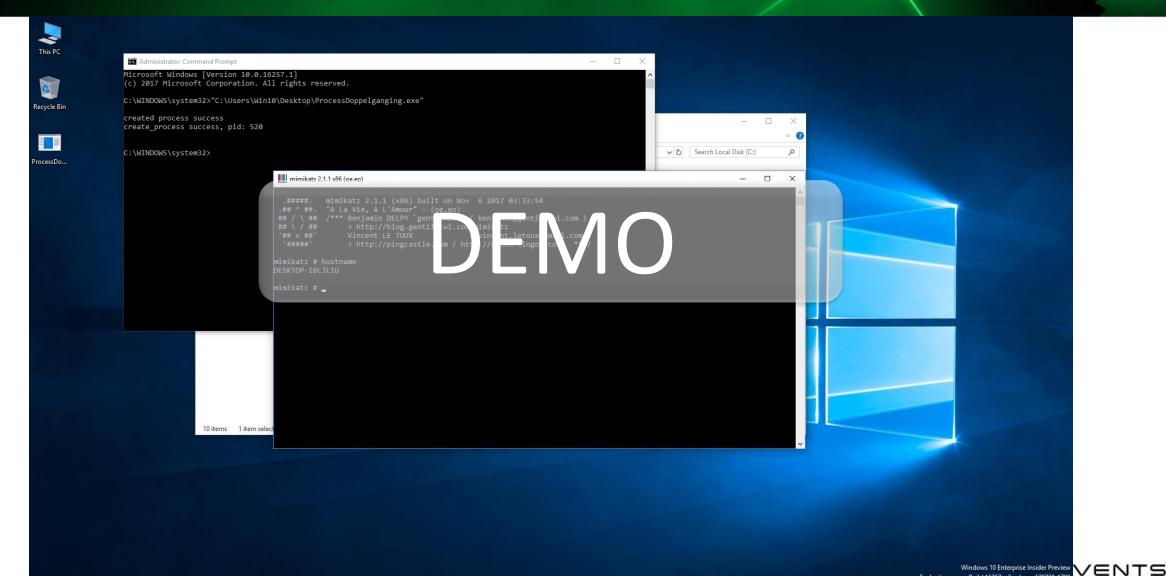
Type here to search

Doppelgänging in Action



Evaluation copy. Build 16257.rs3_release.170728-1700

g^Q ^ **(5)** 12:14 AM





"Mitigation in Redstone" The story of a BSOD



- Everything worked well on Windows 7
- First run on Windows 10 BSOD
- Reported by James Forshaw*
- Null pointer dereference

*https://bugs.chromium.org/p/project-zero/issues/detail?id=852



"Mitigation in Redstone" The story of a BSOD



- How to get over it?
 - PsCreateMinimalProcess
- MS was nice enough to fix it for this talk;)



Affected Products



Product	Tested OS	Result
Windows Defender	Windows 10	Bypass
AVG Internet Security	Windows 10	Bypass
Bitdefender	Windows 10	Bypass
ESET NOD 32	Windows 10	Bypass
Qihoo 360	Windows 10	Bypass
Symantec Endpoint Protection	Windows 7 SP1	Bypass
McAfee VSE 8.8 Patch 6	Windows 7 SP1	Bypass
Kaspersky Endpoint Security 10	Windows 7 SP1	Bypass
Kaspersky Antivirus 18	Windows 7 SP1	Bypass
Symantec Endpoint Protection 14	Windows 7 SP1	Bypass
Panda	Windows 8.1	Bypass
Avast	Windows 8.1	Bypass



Detection / Prevention



Realtime

- Scan using file object available in create process notification routine (Vista+)
 - On error, block
 - What to do about DLLs?
- Scan all sections, even data sections performance issue to consider

Forensics

- WriteAccess == TRUE for the FILE_OBJECT associated with process
- EPROCESS. ImageFilePointer is NULL (Win 10)



Summary



- Process will look legitimate
- Uses Windows loader (no need for a custom one)
- Mapped correctly to an image file on disk, just like any legit process
- No "unmapped code" which is usually detected by modern solutions
- Can also be leveraged to load DLLs
- Fileless
- Even advanced forensics tools such as Volatility will not detect it
- Works on all Windows versions since Vista
- Bypasses all tested security products



Special Thanks



- Omri Misgav Security Researcher @ enSilo
- <u>@UdiYavo</u> CTO @ enSilo
- This research wouldn't be possible without you



DECEMBER 4-7, 2017 EXCEL / LONDON, UK

Questions?

Thank you
Tal Liberman
Eugene Kogan
http://breakingmalware.com