Practical Malware Analysis & Triage
Malware Analysis Report

Malware.stage0.exe

Nov 2022 | Zandmann | v1.0

Table of Contents

Executive Summary

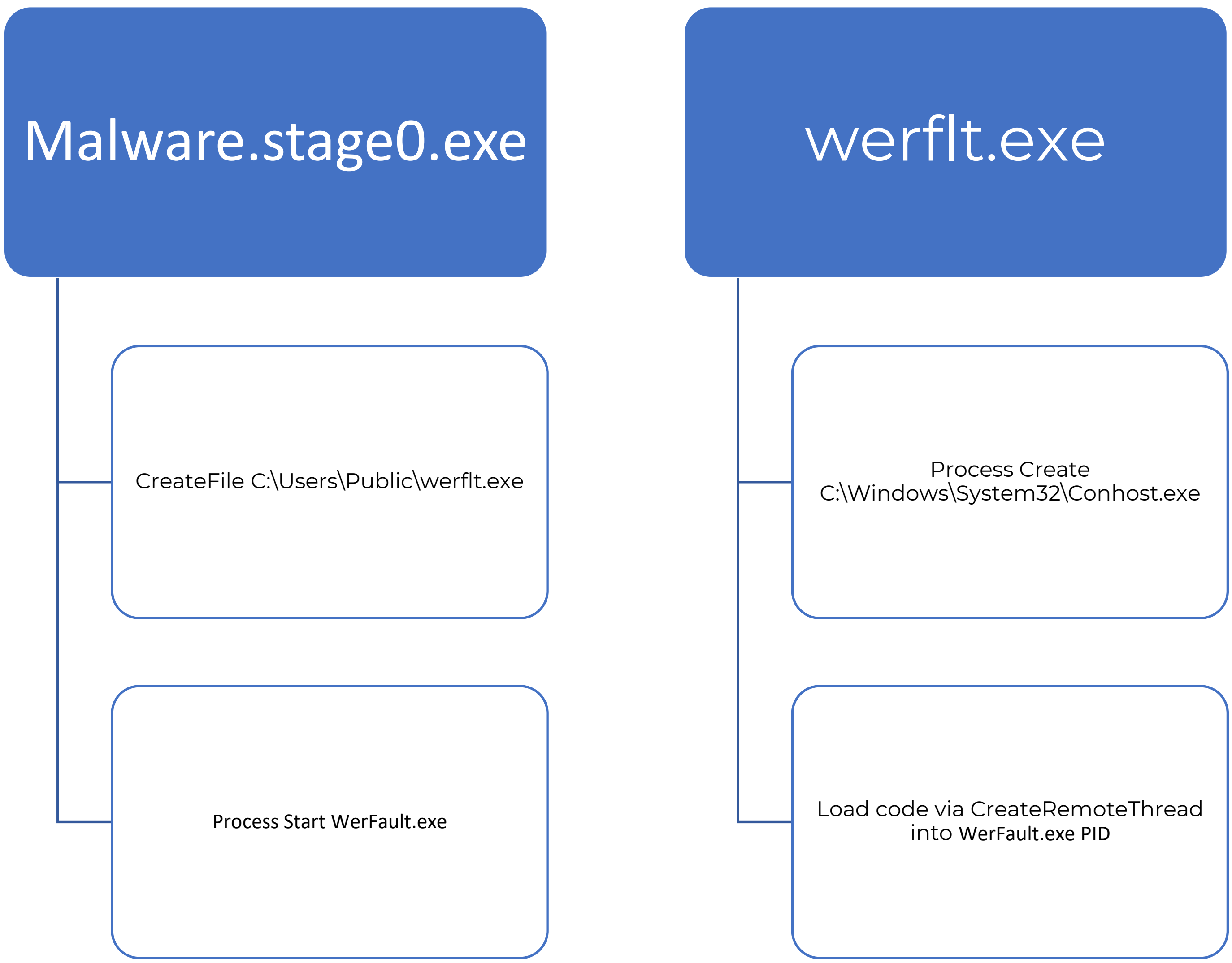
SHA256 hash	FCA62097B364B2F0338C5E4C5BAC86134CEDFFA4F8DDF27EE9901734128952E3

Malware.stage0.exe is a 32-bit dropper binary first identified on May 14th 2021. It is targeting Windows OS and it is using process injection in order to evade detection and run it's reverse shell code inside legitimate Werfault.exe process.

YARA signature rules are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

Malware.stage0.exe consists of two parts: a packed stage 1 dropper and a stage 2 command execution program. Stage 1 creates a stage 2 executable C:\Users\Public\werflt.exe and starts WerFault.exe process, allowing stage 2 binary to inject it's code into WerFault.exe process.
WerFault.exe is then attempting to connect to localhost on port 8443. If succeeds, reverse shell is spawned.



Malware.stage0.exe
Nov 2022
v1.0

Malware Composition

DemoWare consists of the following components:

File Name	SHA256 Hash
Malware.stage0.exe	fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3
werflt.exe	0516009622b951c6c08fd8d81a856eaab70c02e6bc58d066bbdfafe8c6edabea

Malware.stage0.exe

The initial executable that creates a file C:\Users\Public\werflt.exe and start WerFault.exe process.

Werflt.exe

Created executable file containing the second stage payload.

Basic Static Analysis

{Screenshots and description about basic static artifacts and methods}

Stage 1

FCA62097B364B2F0338C5E4C5BAC86134CEDFFA4F8DDF27EE9901734128952E3

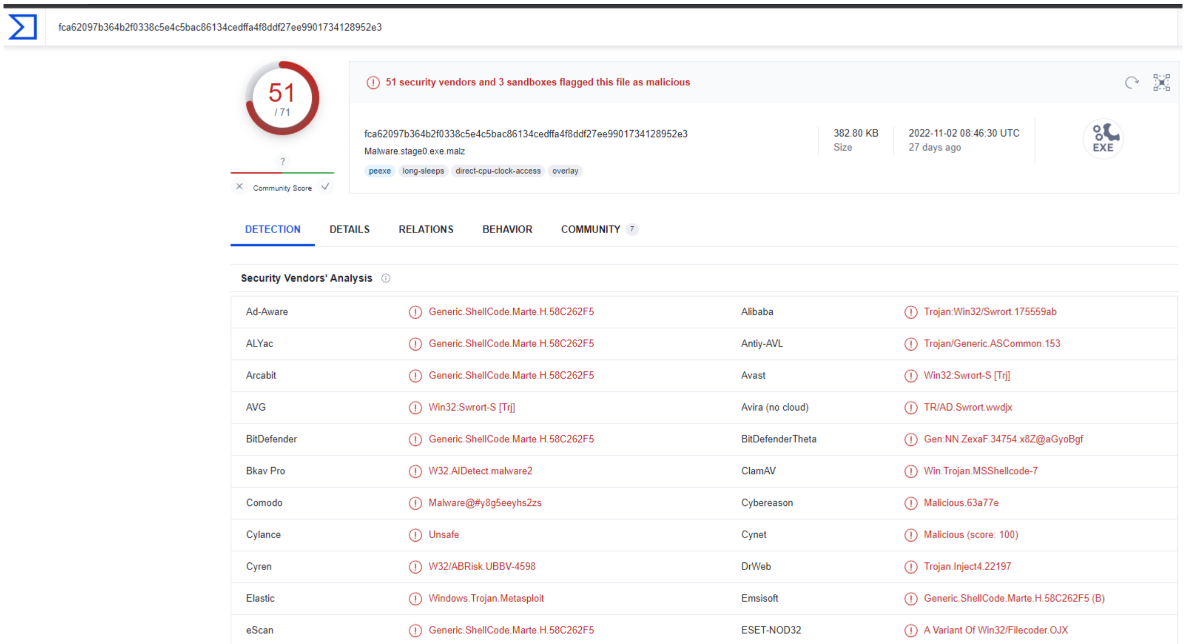


Figure 1 - Virus Total result for dropper file

STRINGS

```
@C:\Users\Public\werflt.exe
@C:\Windows\SysWOW64\WerFault.exe
C:\\Users\\Administrator\\source\\repos\\CRTInjectorConsole\\Release\\CRTInjectorConsole
.pdb
```

We may assume, the binary is written in .nim

```
C:\Users\[redacted]\Desktop
λ cat floss.txt | grep .nim
fatal.nim
io.nim
fatal.nim
@iterators.nim(222, 11) `len(a) == L` the length of the string changed while iterating over it
streams.nim
strutils.nim
oserr.nim
@iterators.nim(222, 11) `len(a) == L` the length of the string changed while iterating over it
@osproc.nim(770, 14) `p.errStream == nil or
@osproc.nim(769, 14) `p.outStream == nil or
@osproc.nim(703, 14) `args.len == 0`
stdlib_io.nim.c
stdlib_times.nim.c
stdlib_os.nim.c
@mstage0.nim.c
stdlib_assertions.nim.c
_nimAddInt
_nimSubInt
stdlib_widestrs.nim.c
_nimToCStringConv
_nimZeroMem
_nimGC_setStackBottom
@nimGCvisit@8
@nimIntToStr@4
@nimRegisterThreadLocalMarker@4
@nimInt64ToStr@8
```

Figure 2 - floss output for dropper file

With some help of pestudio we may spot, virtualized section and embedded files

section > virtualized	.bss	2
overlay > file-ratio	15.10%	2
overlay > entropy	4.665	2
overlay > size	59187 bytes	2
file > embedded	signature: executable, location: .rdata, offset: 0x0000BE28, size: 9060	2
file > embedded	signature: unknown, location: overlay, offset: 0x00051400, size: 59187	2
overlay > signature > name	unknown	2

Figure 3 - indicators (pestudio)

IMPORTS

Imports might give us a tip of binary capabilities.

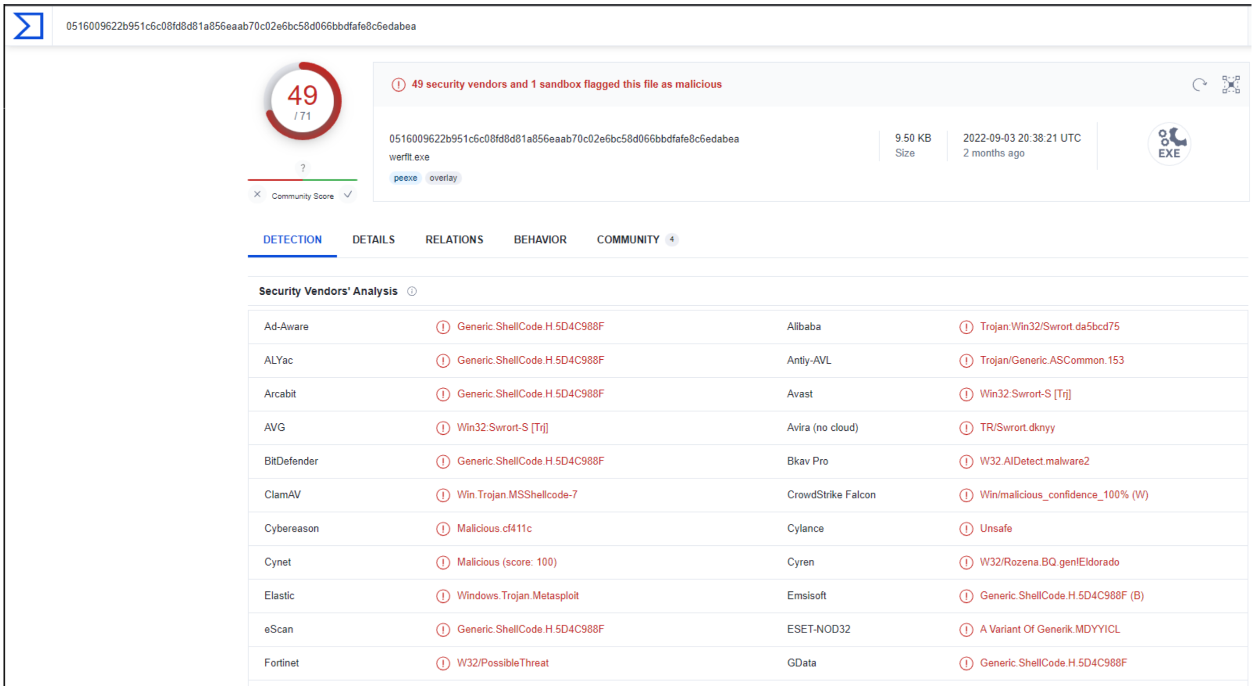
imports (71)	flag (4)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (8)	type (1)	ordinal (0)	library (3)
VirtualProtect	x	0x0001B44E	0x0001B44E	1469 (0x05BD)	memory	implicit	-	KERNEL32.dll
GetCurrentProcessId	x	0x0001B2E4	0x0001B2E4	544 (0x0220)	execution	implicit	-	KERNEL32.dll
GetCurrentThreadId	x	0x0001B2FA	0x0001B2FA	548 (0x0224)	execution	implicit	-	KERNEL32.dll
TerminateProcess	x	0x0001B3F2	0x0001B3F2	1401 (0x0579)	execution	implicit	-	KERNEL32.dll

Figure 4 - imports data for dropper (pestudio)

VirtualProtect is often used by malware to modify memory protection (often to allow write or execution). Therefore, this might indicate mentioned Process Injection technique.

Stage2

0516009622b951c6c08fd8d81a856eaab70c02e6bc58d066bbdfafe8c6edabea


0516009622b951c6c08fd8d81a856eaab70c02e6bc58d066bbdfafe8c6edabea

49 / 71

Community Score

49 security vendors and 1 sandbox flagged this file as malicious

0516009622b951c6c08fd8d81a856eaab70c02e6bc58d066bbdfafe8c6edabea
werfit.exe
peexe overlay

9.50 KB Size

2022-09-03 20:38:21 UTC
2 months ago

EXE

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 4

Security Vendors' Analysis

Ad-Aware	Generic.ShellCode.H.5D4C988F	Alibaba	Trojan:Win32/Swrort.da5bcd75
ALYac	Generic.ShellCode.H.5D4C988F	Antiy-AVL	Trojan/Generic.ASCommon.153
Arcabit	Generic.ShellCode.H.5D4C988F	Avast	Win32:Swrort-S [Trj]
AVG	Win32:Swrort-S [Trj]	Avira (no cloud)	TR/Swrort.dknyy
BitDefender	Generic.ShellCode.H.5D4C988F	Bkav Pro	W32.AIDetect.malware2
ClamAV	Win.Trojan.MSShellcode-7	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cybereason	Malicious.cf411c	Cylance	Unsafe
Cynet	Malicious (score: 100)	Cyren	W32/Rozena.BQ.gen!Eldorado
Elastic	Windows.Trojan.Metasploit	Emsisoft	Generic.ShellCode.H.5D4C988F (B)
eScan	Generic.ShellCode.H.5D4C988F	ESET-NOD32	A Variant Of Generik.MDYYICL
Fortinet	W32/PossibleThreat	GData	Generic.ShellCode.H.5D4C988F

Figure 5 - Virus Total result for stage 2

STRINGS:

```
!This program cannot be run in DOS mode.
C:\Users\Administrator\source\repos\CRTInjectorConsole\Release\CRTInjectorConsole.pdb
WriteProcessMemory
OpenProcess
CloseHandle
VirtualAllocEx
CreateRemoteThread
GetModuleHandleW
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level='asInvoker' uiAccess='false' />
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

IMPORTS

At this point we might be pretty sure, this binary is performing process injection

imports (46)	flag (6)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (7)	type (1)	ordinal (0)	library (8)
WriteProcessMemory	x	0x00002878	0x00002878	1567 (0x061F)	memory	implicit	-	KERNEL32.dll
OpenProcess	x	0x0000288E	0x0000288E	1039 (0x040F)	execution	implicit	-	KERNEL32.dll
CreateRemoteThread	x	0x000028BC	0x000028BC	234 (0x00EA)	execution	implicit	-	KERNEL32.dll
GetCurrentThreadId	x	0x00002C88	0x00002C88	543 (0x021F)	execution	implicit	-	KERNEL32.dll
GetCurrentProcessId	x	0x00002C72	0x00002C72	539 (0x021B)	execution	implicit	-	KERNEL32.dll
TerminateProcess	x	0x00002C28	0x00002C28	1424 (0x0590)	execution	implicit	-	KERNEL32.dll

Figure 6 - imports data for dropper

Basic Dynamic Analysis

{Screenshots and description about basic dynamic artifacts and methods}

After execution of Malware.stage0.exe, CreateFile werflt.exe operation is performed

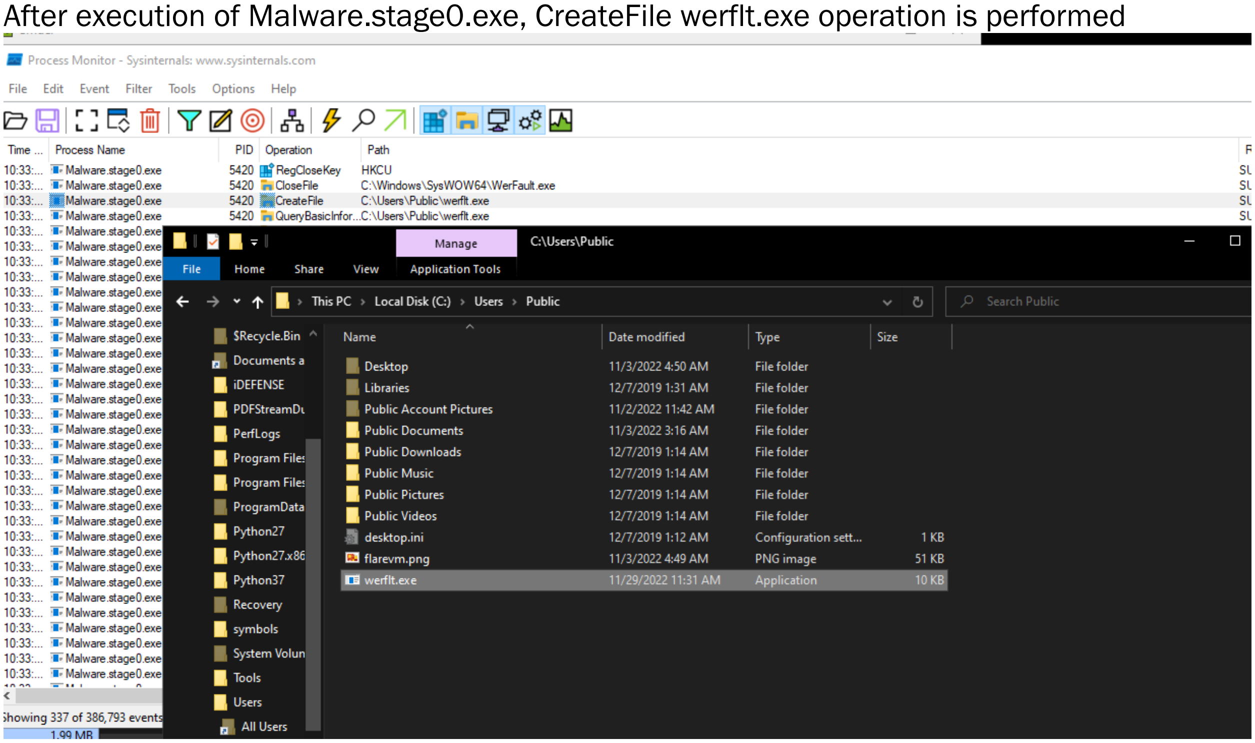


Figure 7 - FileCreate operation for werflt.exe

legitimate WerFault.exe run and werflt.exe execute with legitimate process (WerFault.exe) PID as argument

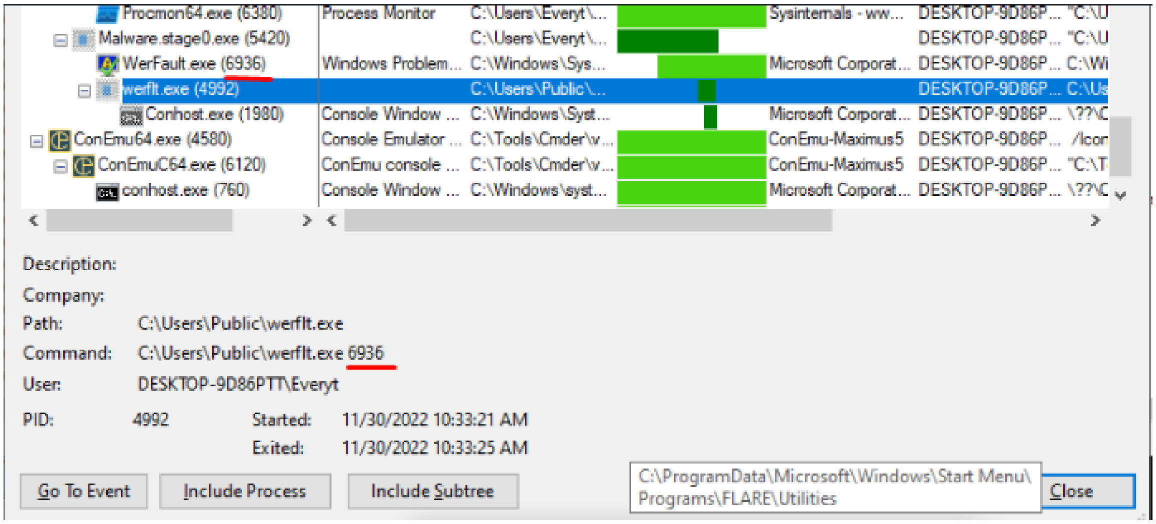


Figure 8 - Process Tree after dropper execution

Cmdline was also spotted for a brief moment

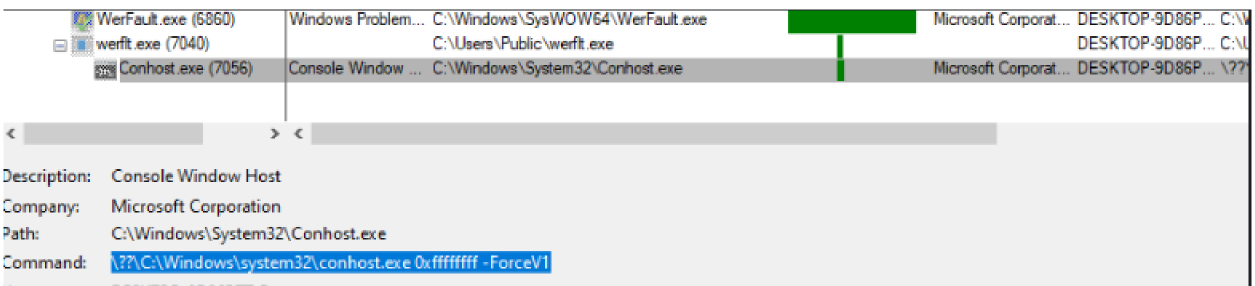


Figure 9 - Process Tree after dropper execution

WerFault.exe tries to connect to 127.0.0.1 on port 8443

Figure 10 - output from tcpview

After setting up ncat listener we receive a reverse shell and TCP connection is established

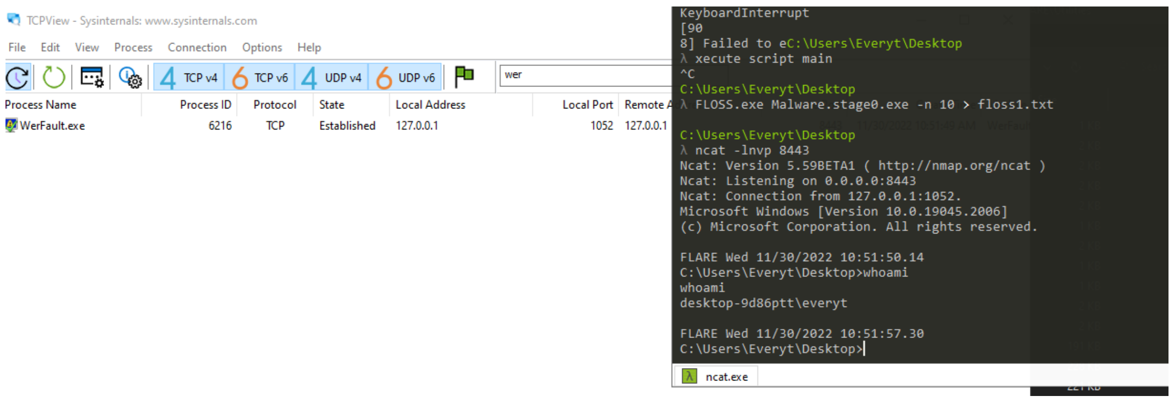


Figure 11 - TCP connection for reverse shell

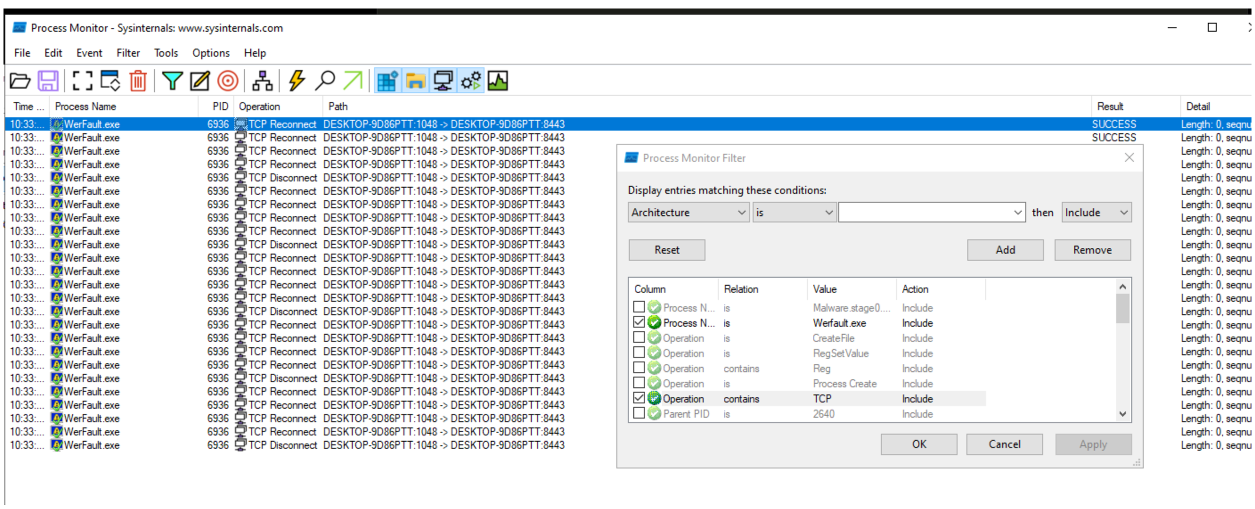


Figure 12 - TCP connections (process monitor)

Advanced Static Analysis

{Screenshots and description about findings during advanced static analysis}

In assembly we may observe a typical pattern for CreateRemoteThread with process injection

```
[0x00401000]
  ;-- section..text:
 159: int main (int32_t arg_ch);
 ; var LPCVOID lpBuffer @ ebp-0x14c
 ; var int32_t var_4h @ ebp-0x4
 ; arg int32_t arg_ch @ ebp+0xc
 push    ebp                          ; [00] -r-x section size 4096 named .text
 mov     ebp, esp
 sub     esp, 0x14c
 mov     eax, dword [0x403004]
 xor     eax, ebp
 mov     dword [var_4h], eax
 mov     eax, dword [arg_ch]
 mov     ecx, 0x51                    ; 'Q' ; 81
 push    esi
 push    edi
 mov     esi, 0x402110
 lea     edi, [lpBuffer]
 push    dword [eax + 4]              ; const char *str
 rep     movsd dword es:[edi], dword ptr [esi]
 movsb   byte es:[edi], byte ptr [esi]
 call    dword [atoi]                 ; 0x40205c ; int atoi(const char *str)
 add     esp, 4
 push    eax
 push    0                            ; BOOL bInheritHandle
 push    0x1fffff                     ; DWORD dwDesiredAccess
 call    dword [OpenProcess]          ; 0x402004 ; HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bI...
 push    0x40                         ; '@' ; 64
 push    0x3000
 push    0x145                        ; 325
 mov     edi, eax
 push    0                            ; LPVOID lpAddress
 push    edi                          ; HANDLE hProcess
 call    dword [VirtualAllocEx]       ; 0x40200c ; LPVOID VirtualAllocEx(HANDLE hProcess, LPVOID lpA...
 push    0                            ; SIZE_T *lpNumberOfBytesWritten
 mov     esi, eax
 lea     eax, [lpBuffer]
 push    0x145                        ; 325 ; SIZE_T nSize
 push    eax                          ; LPCVOID lpBuffer
 push    esi                          ; LPVOID lpBaseAddress
 push    edi                          ; HANDLE hProcess
 call    dword [WriteProcessMemory]   ; 0x402000 ; BOOL WriteProcessMemory(HANDLE hProcess, LPVOID l...
```

Figure 13 - CreateRemoteThread code snippet (cutter)

```
push    0
push    0
push    0
push    esi
push    0
push    0                        ; LPSECURITY_ATTRIBUTES lpThreadAttributes
push    edi                      ; HANDLE hProcess
call    dword [CreateRemoteThread] ; 0x402010 ; HANDLE CreateRemoteThread(HANDLE hProcess, LPSECU...
push    edi                      ; HANDLE hObject
call    dword [CloseHandle]      ; 0x402008 ; BOOL CloseHandle(HANDLE hObject)
mov     ecx, dword [var_4h]
xor     eax, eax
pop     edi
xor     ecx, ebp
pop     esi
call    fcn.0040109f
mov     esp, ebp
pop     ebp
ret
```

Figure 14 - CreateRemoteThread code snippet (cutter)

API calls:

OpenProcess

```
add     esp, 4
push    eax
push    0                   ; BOOL bInheritHandle
push    0x1fffff            ; DWORD dwDesiredAccess
call    dword [OpenProcess] ; 0x402004 ; HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bI...
push    0x40    call   dword [OpenProcess]    ; 0x402004 ; HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId)
push    0x3000
```

Figure 15 - OpenProcess API call code snippet

uses 3 parameters, with the most interesting one being dwProcessId, which is used in order to get access to WerFault.exe process

and desiredAccess

PROCESS_ALL_ACCESS (0x1fffff)	All possible access rights for a process object.

Figure 16 - reference do MS documentation

dwProcessId was stored in eax after arg_ch was moved into it before this function call

```
mov dword [var_4h], eax
mov eax, dword [arg_ch]
mov ecx, 0x51
```

Figure 17 - OpenProcess API call code snippet

VirtualAllocEx

Next, eax (process handle at this point) was moved into edi

```
push    0x145
mov     edi, eax
push    0
```

Figure 18 - VirtualAllocEx API call code snippet

And edi is used in next function in order to allocate memory inside of that process

```
mov     edi, eax
push    0                       ; LPVOID lpAddress
push    edi                     ; HANDLE hProcess
call    dword [VirtualAllocEx]  ; 0x40200c ; LPVOID VirtualAllocEx(HANDLE hProcess, LPVOID lpA...
push    0                       ; SIZE_T *lpNumberOfBytesWritten
mov     esi, eax    call   dword [VirtualAllocEx]   ; 0x40200c ; LPVOID VirtualAllocEx(HANDLE hProcess, LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect)
lea     eax, [lpBuffer]
```

Figure 19 - VirtualAllocEx API call code snippet

WriteProcessMemory

The same handle is used in this API call in order to write to allocated section of its memory with bytes in previously declared variable

```
159: int main (int32_t arg_ch);
; var LPCVOID lpBuffer @ ebp-0x14c
; var int32_t var_4h @ ebp-0x4
; arg int32_t arg_ch @ ebp+0xc
```

Figure 20 - WriteProcessMemory API call code snippet

```
lea     eax, [lpBuffer]
push    0x145                       ; 325 ; SIZE_T nSize
push    eax                         ; LPCVOID lpBuffer
push    esi                         ; LPVOID lpBaseAddress
push    edi                         ; HANDLE hProcess
call    dword [WriteProcessMemory]  ; 0x402000 ; BOOL WriteProcessMemory(HANDLE hProcess, LPVOID l...
```

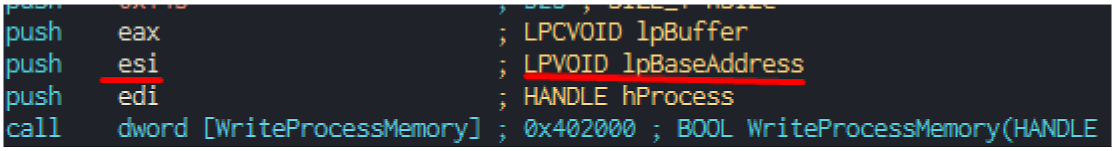
Figure 21 - WriteProcessMemory API call code snippet

```
mov     esi, eax
lea     eax, [lpBuffer]
push    0x145                       ; 325 ; SIZE_T nSize
push    eax                         ; LPCVOID lpBuffer
push    esi                         ; LPVOID lpBaseAddress
push    edi                         ; HANDLE hProcess
call    dword [WriteProcessMemory]  ; 0x402000 ; BOOL WriteProcessMemory(HANDLE hProcess, LPVOID l...
push    0
push    0    call   dword [WriteProcessMemory] ; 0x402000 ; BOOL WriteProcessMemory(HANDLE hProcess, LPVOID lpBaseAddress, LPCVOID lpBuffer, SIZE_T nSize, SIZE_T *lpNumberOfBytesWritten)
```

Figure 22 - WriteProcessMemory API call code snippet

CreateRemoteThread

Two parameters are used in this API call esi – Start address which is the base address of the data written during VirtualAlloc call

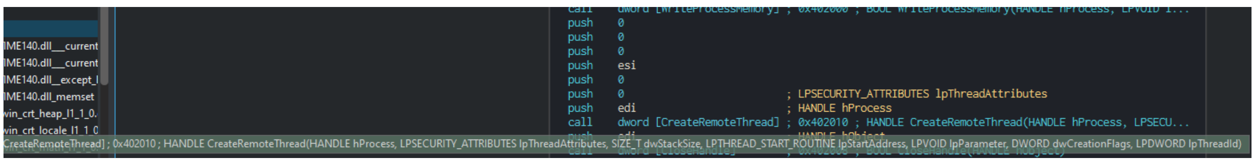


```
push    eax                       ; LPCVOID lpBuffer
push    esi                       ; LPVOID lpBaseAddress
push    edi                       ; HANDLE hProcess
call    dword [WriteProcessMemory] ; 0x402000 ; BOOL WriteProcessMemory(HANDLE
```

Figure 23 - CreateRemoteThread API call code snippet

```
LPTHREAD_START_ROUTINE lpStartAddress
```

edi – process handle



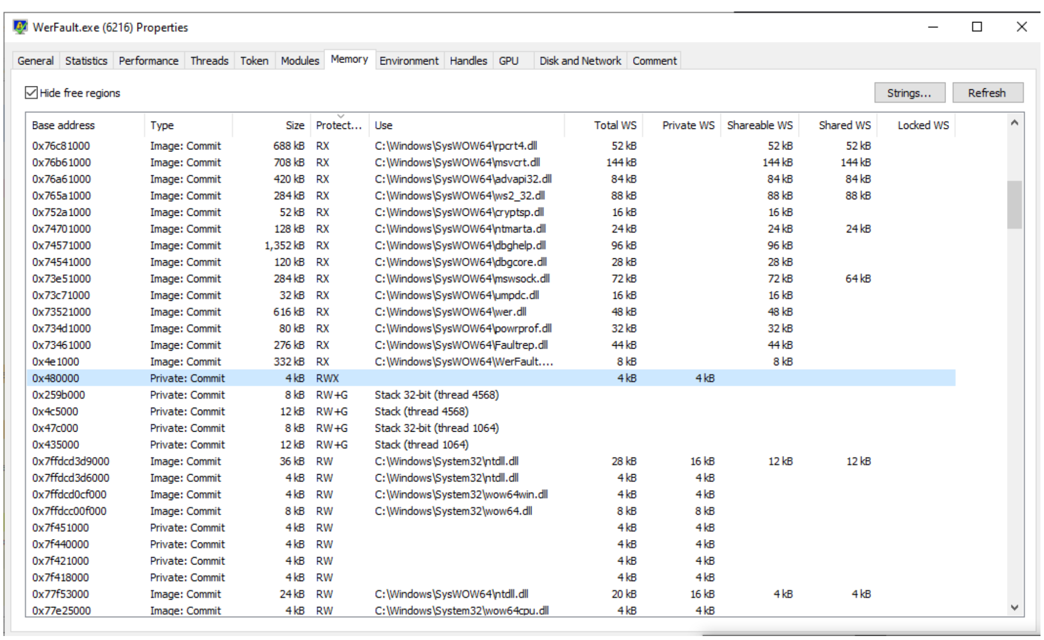
```
push    0
push    0
push    0
push    esi
push    0
push    0                           ; LPSECURITY_ATTRIBUTES lpThreadAttributes
push    edi                         ; HANDLE hProcess
call    dword [CreateRemoteThread]  ; 0x402010 ; HANDLE CreateRemoteThread(HANDLE hProcess, LPSECU...
```

```
CreateRemoteThread] ; 0x402010 ; HANDLE CreateRemoteThread(HANDLE hProcess, LPSECURITY_ATTRIBUTES lpThreadAttributes, SIZE_T dwStackSize, LPTHREAD_START_ROUTINE lpStartAddress, LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId)
```

Figure 24 - CreateRemoteThread API call code snippet

With the above actions, a shellcode was injected into WerFault.exe process.

After having a closer look at WerFault.exe in Process Hacker we may observe an extensive amount of permissions (RWX) for a particular section



WerFault.exe (6216) Properties

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | GPU | Disk and Network | Comment

Hide free regions — Strings... — Refresh

Base address	Type	Size	Protect...	Use	Total WS	Private WS	Shareable WS	Shared WS	Locked WS
0x76c81000	Image: Commit	688 kB	RX	C:\Windows\SysWOW64\rpcrt4.dll	52 kB		52 kB	52 kB	
0x76b61000	Image: Commit	708 kB	RX	C:\Windows\SysWOW64\msvcrt.dll	144 kB		144 kB	144 kB	
0x76a61000	Image: Commit	420 kB	RX	C:\Windows\SysWOW64\advapi32.dll	84 kB		84 kB	84 kB	
0x765a1000	Image: Commit	284 kB	RX	C:\Windows\SysWOW64\ws2_32.dll	88 kB		88 kB	88 kB	
0x752a1000	Image: Commit	52 kB	RX	C:\Windows\SysWOW64\cryptsp.dll	16 kB		16 kB		
0x74701000	Image: Commit	128 kB	RX	C:\Windows\SysWOW64\ntmarta.dll	24 kB		24 kB	24 kB	
0x74571000	Image: Commit	1,352 kB	RX	C:\Windows\SysWOW64\dbghelp.dll	96 kB		96 kB		
0x74541000	Image: Commit	120 kB	RX	C:\Windows\SysWOW64\dbgcore.dll	28 kB		28 kB		
0x73e51000	Image: Commit	284 kB	RX	C:\Windows\SysWOW64\mswsock.dll	72 kB		72 kB	64 kB	
0x73c71000	Image: Commit	32 kB	RX	C:\Windows\SysWOW64\umpdc.dll	16 kB		16 kB		
0x73521000	Image: Commit	616 kB	RX	C:\Windows\SysWOW64\wer.dll	48 kB		48 kB		
0x734d1000	Image: Commit	80 kB	RX	C:\Windows\SysWOW64\powrprof.dll	32 kB		32 kB		
0x73461000	Image: Commit	276 kB	RX	C:\Windows\SysWOW64\Faultrep.dll	44 kB		44 kB		
0x4e1000	Image: Commit	332 kB	RX	C:\Windows\SysWOW64\WerFault....	8 kB		8 kB		
0x480000	Private: Commit	4 kB	RWX		4 kB	4 kB			
0x259b000	Private: Commit	8 kB	RW+G	Stack 32-bit (thread 4568)					
0x4c5000	Private: Commit	12 kB	RW+G	Stack (thread 4568)					
0x47c000	Private: Commit	8 kB	RW+G	Stack 32-bit (thread 1064)					
0x435000	Private: Commit	12 kB	RW+G	Stack (thread 1064)					
0x7ffdcd3d9000	Image: Commit	36 kB	RW	C:\Windows\System32\ntdll.dll	28 kB	16 kB	12 kB	12 kB	
0x7ffdcd3d6000	Image: Commit	4 kB	RW	C:\Windows\System32\ntdll.dll	4 kB	4 kB			
0x7ffdcd0cf000	Image: Commit	4 kB	RW	C:\Windows\System32\wow64win.dll	4 kB	4 kB			
0x7ffdcc00f000	Image: Commit	8 kB	RW	C:\Windows\System32\wow64.dll	8 kB	8 kB			
0x7f451000	Private: Commit	4 kB	RW		4 kB	4 kB			
0x7f440000	Private: Commit	4 kB	RW		4 kB	4 kB			
0x7f421000	Private: Commit	4 kB	RW		4 kB	4 kB			
0x7f418000	Private: Commit	4 kB	RW		4 kB	4 kB			
0x77f53000	Image: Commit	24 kB	RW	C:\Windows\SysWOW64\ntdll.dll	20 kB	16 kB	4 kB	4 kB	
0x77e25000	Image: Commit	4 kB	RW	C:\Windows\System32\wow64cpu.dll	4 kB	4 kB			

Figure 25 - Process Hacker

With injected shellcode in it

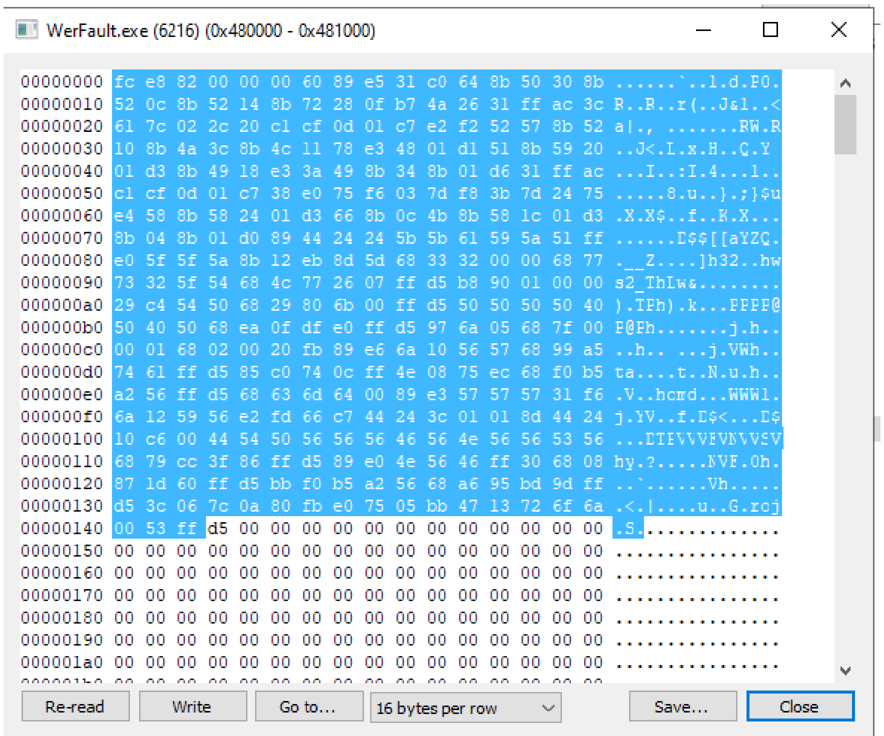


Figure 26 - Process Hacker

Advanced Dynamic Analysis

{Screenshots and description about advanced dynamic artifacts and methods}

API calls present in stage1 file

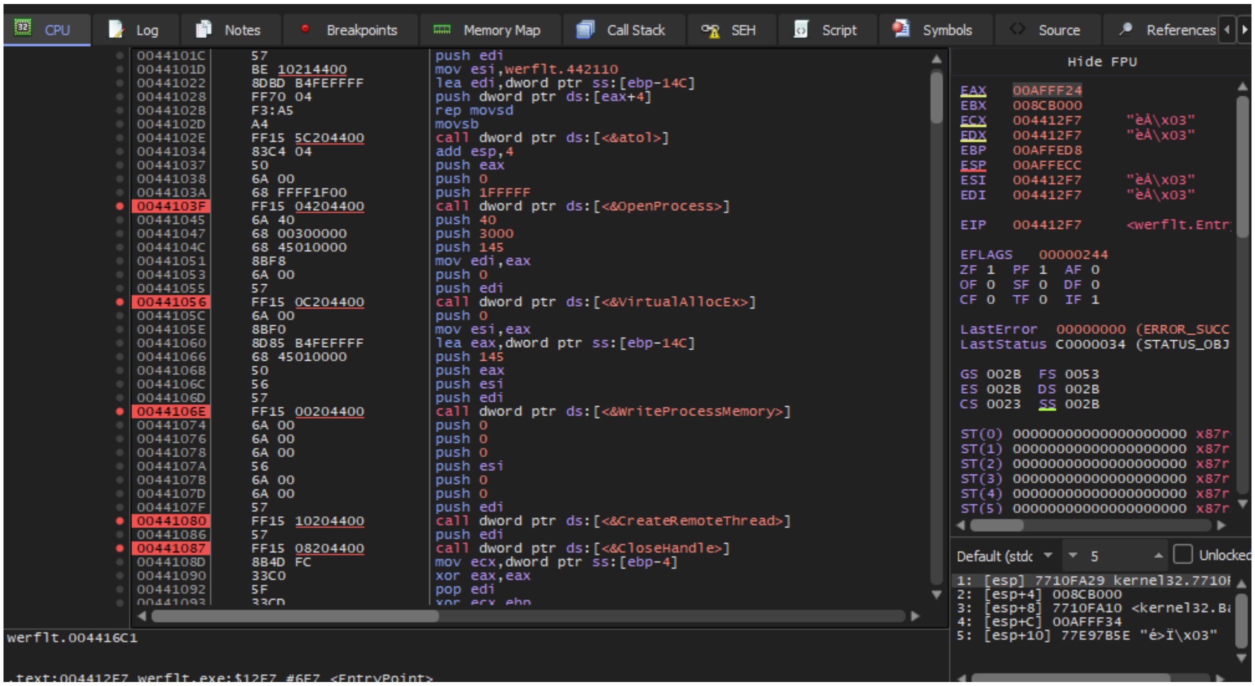


Figure 27 - main API calls (x32dbg)

Indicators of Compromise

The full list of IOCs can be found in the Appendices.

Network Indicators

{Description of network indicators}

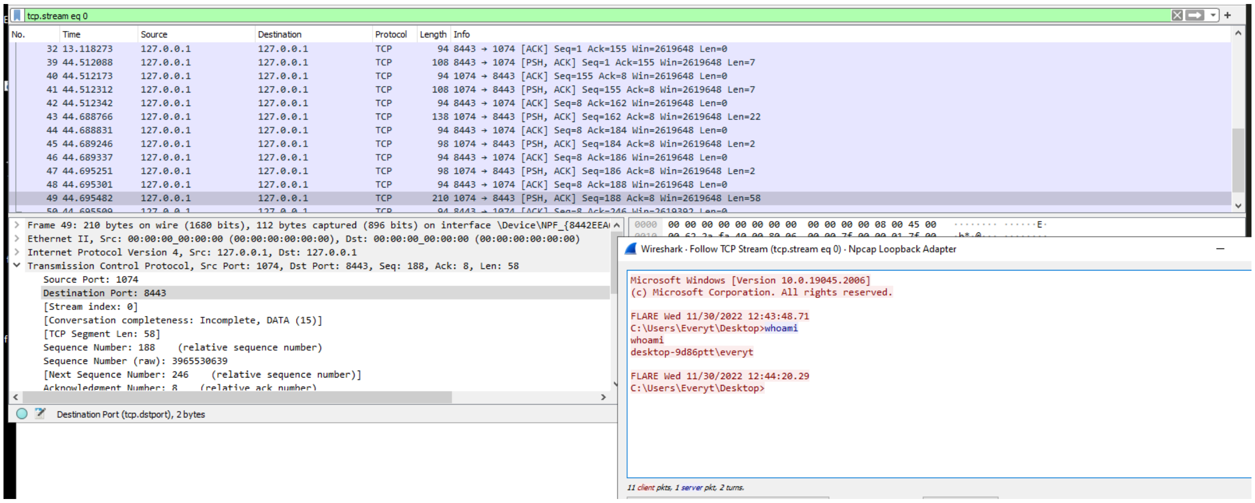


Figure 28 - WireShark Packet Capture of reverse shell connection

Host-based Indicators

{Description of host-based indicators}

Strings:
@C:\Users\Public\werflt.exe
@C:\Windows\SysWOW64\WerFault.exe

Registry (RegSetValue):
HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-108361916-3091764824-3706894550-1001\\Device\HarddiskVolume2\Users\Public\werflt.exe

Filename:
Malware.stage0.exe
werflt.exe
C:\Users\Public\werflt.exe

sha256 hash:
fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3
0516009622b951c6c08fd8d81a856eaab70c02e6bc58d066bbdfafe8c6edabea

Rules & Signatures

A full set of YARA rules is included in Appendix A.

{Information on specific signatures, i.e. strings, URLs, etc}

Appendices

A. Yara Rules

Full Yara repository located at: https://github.com/Zandmann/YARA

```
rule Yara_Malware {

    meta:
        last_updated = "2022-11-30"
        author = "Zandmann"
        description = "Yara for Malware.stage0.exe"

    strings:
        // Fill out identifying strings and other criteria
        $string1 = "C:\\Users\\Public\\werflt.exe" ascii nocase
        $string2 = "C:\\Windows\\SysWOW64\\WerFault.exe" ascii nocase
        $string3 = "CRTInjectorConsole.pdb" ascii nocase
        $PE_magic_byte = "MZ"
        $sus_hex_string = { FF 15 10 20 40 }

    condition:
        // Fill out the conditions that must be met to identify the binary
        $PE_magic_byte at 0 and
        ($string1 and $string2) or

        ($sus_hex_string and $string3)
}
```

B. Callback IPs

IPs	Port
127.0.0.1	8443

C. Decompiled Code Snippets

```
mov     esi, 0x402110
lea     edi, [lpBuffer]
push    dword [eax + 4]            ; const char *str
rep     movsd dword es:[edi], dword ptr [esi]
movsb   byte es:[edi], byte ptr [esi]
call    dword [atoi]               ; 0x40205c ; int atoi(const char *str)
add     esp, 4
push    eax
push    0                          ; BOOL bInheritHandle
push    0x1fffff                   ; DWORD dwDesiredAccess
call    dword [OpenProcess]        ; 0x402004 ; HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bI...
push    0x40                       ; '@' ; 64
push    0x3000
push    0x145                      ; 325
mov     edi, eax
push    0                          ; LPVOID lpAddress
push    edi                        ; HANDLE hProcess
call    dword [VirtualAllocEx]     ; 0x40200c ; LPVOID VirtualAllocEx(HANDLE hProcess, LPVOID lpA...
push    0                          ; SIZE_T *lpNumberOfBytesWritten
mov     esi, eax
lea     eax, [lpBuffer]
push    0x145                      ; 325 ; SIZE_T nSize
push    eax                        ; LPCVOID lpBuffer
push    esi                        ; LPVOID lpBaseAddress
push    edi                        ; HANDLE hProcess
call    dword [WriteProcessMemory] ; 0x402000 ; BOOL WriteProcessMemory(HANDLE hProcess, LPVOID l...
push    0
push    0
push    0
push    esi
push    0
push    0                          ; LPSECURITY_ATTRIBUTES lpThreadAttributes
push    edi                        ; HANDLE hProcess
call    dword [CreateRemoteThread] ; 0x402010 ; HANDLE CreateRemoteThread(HANDLE hProcess, LPSECU...
push    edi                        ; HANDLE hObject
call    dword [CloseHandle]        ; 0x402008 ; BOOL CloseHandle(HANDLE hObject)
mov     ecx, dword [var_4h]
xor     eax, eax
pop     edi
xor     ecx, ebp
```

Figure 29 - Process Injection Routine in Cutter

D. MITRE

T1055.003
T1129

```
C:\Users\Public>capa werflt.exe
loading : 100%|                                                            | 485/485 [00:00<00:00, 610.13     rules/s]
matching: 100%|                                                               | 60/60 [00:02<00:00, 23.91 functions/s]
+------------------------+------------------------------------------------------------------------------------+
| md5                    | 0da707ecf411cf8859a221879cc60ea4                                                   |
| sha1                   | b52f520eae2f03ce043602a2361ebf4af64e3f47                                           |
| sha256                 | 0516009622b951c6c08fd8d81a856eaab70c02e6bc58d066bbdfafe8c6edabea                   |
| path                   | werflt.exe                                                                         |
+------------------------+------------------------------------------------------------------------------------+

+------------------------+------------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                   |
|------------------------+------------------------------------------------------------------------------------|
| DEFENSE EVASION        | Process Injection::Thread Execution Hijacking [T1055.003]                          |
| EXECUTION              | Shared Modules [T1129]                                                             |
+------------------------+------------------------------------------------------------------------------------+

+-----------------------------+-------------------------------------------------------------------------------+
| MBC Objective               | MBC Behavior                                                                  |
|-----------------------------+-------------------------------------------------------------------------------|
| MEMORY                      | Allocate Memory [C0007]                                                       |
| PROCESS                     | Create Thread [C0038]                                                         |
|                             | Terminate Process [C0018]                                                     |
+-----------------------------+-------------------------------------------------------------------------------+

+------------------------------------------------------+------------------------------------------------------+
| CAPABILITY                                           | NAMESPACE                                            |
|------------------------------------------------------+------------------------------------------------------|
| contains PDB path                                    | executable/pe/pdb                                    |
| contain a resource (.rsrc) section                   | executable/pe/section/rsrc                           |
| inject thread                                        | host-interaction/process/inject                      |
| terminate process                                    | host-interaction/process/terminate                   |
| terminate process via fastfail (2 matches)           | host-interaction/process/terminate                   |
| parse PE header (2 matches)                          | load-code/pe                                         |
| spawn thread to RWX shellcode                        | load-code/shellcode                                  |
+------------------------------------------------------+------------------------------------------------------+
```

Figure 30 - MITRE mapping (CAPA)