Practical Malware Analysis & Triage Malware Analysis Report

Process Injection Malware

Nov 2022 | NoobUltraProMax | v1.0

Table of Contents

Executive Summary

SHA256 hash	fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3
MD5 hash	6d8895c63a77ebe5e49b656bdefdb822

MalwareStage0.exe is a ProcessInjector-dropper malware sample first identified on May 14th, 2021. It is a Nim-compiled dropper that runs on the x86 Windows operating system. It consists of one unpacked payloads that is executed following a successful spearphishing attempt. Symptoms of infection includes a new process in system werflt.exe which could be confused with a real microsoft software named WerFault.exe.

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: an unpacking stage where werfault.exe is executed. Werfault.exe is the Windows Error Reporting process of Windows 10. Second stage creates a file in C drive with slightly different name werflt.exe, this file is a Remote Process Injector. This malware is executed with parameter as PID of the original error checking program, now the shellcode is executed with the privileges of the Werfault.exe .After execution of the second stage it open a port on localhost (hxxps://localhost:8443)

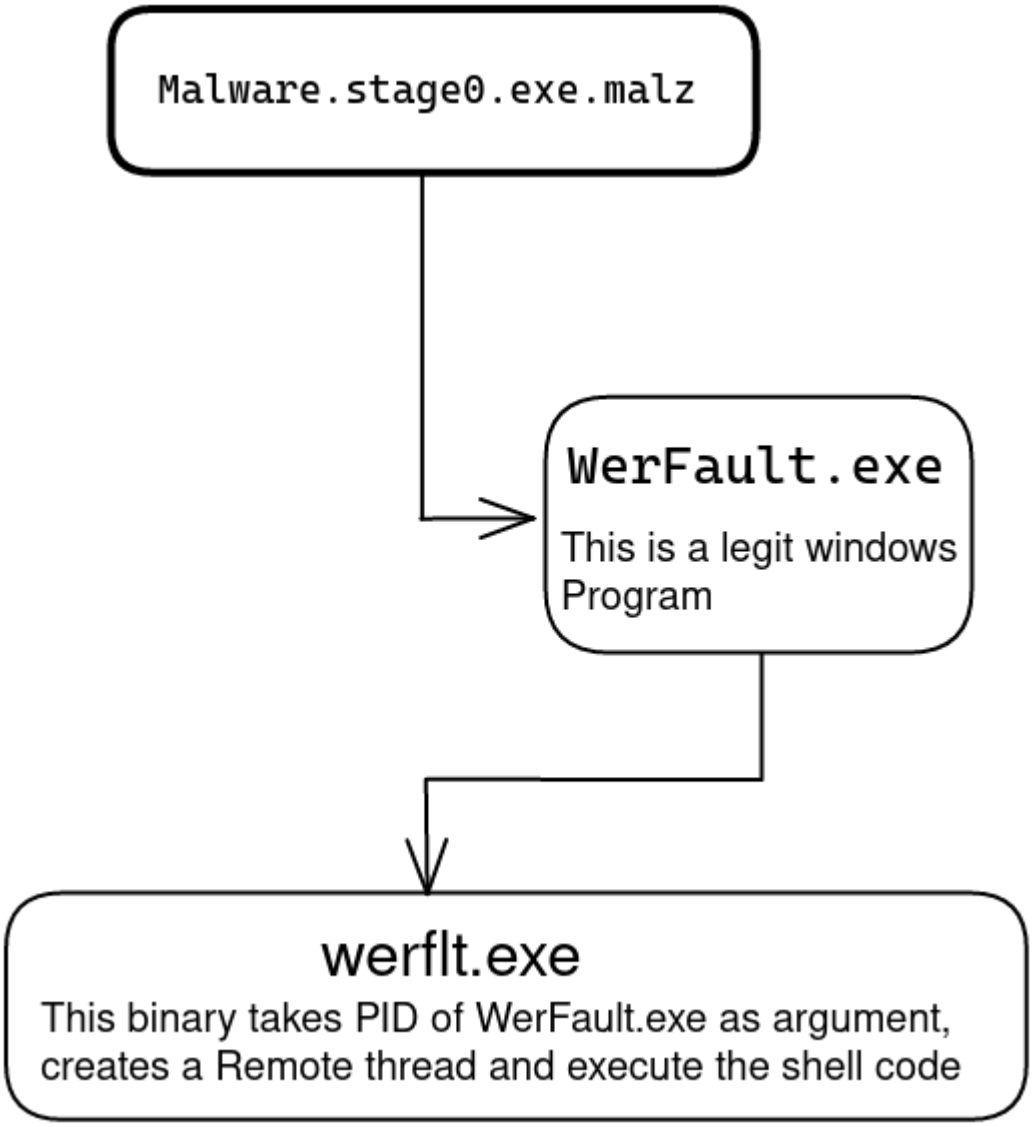

Illustration 1: Flow Diagram of Malware

Malware Composition

Malware consists of the following components:

File Name	SHA256 Hash
Malware.Stage0.exe	fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3
werflt.exe	0516009622b951c6c08fd8d81a856eaab70c02e6bc58d066bbdfafe8c6edabea

malware.exe

The initial executable that runs after a successful spearphish. It acts as a trojan horse carrying a process injector malware, which when executed unpacks the payload, werflt.exe.

werflt.exe:

This binary is unpacked when Malware.exe is detonated. This binary is a Remote Thread Process Injector Malware.

Basic Static Analysis

- **Virustotal Information :**
  - Uploading hash/Binary
  - Its a known malware, 51 security vendors and 3 sandboxes flagged this file as malicious
  - https://www.virustotal.com/gui/file/fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3

- **Strings/Floss output Analysis :**
  - “`$ floss -n 9 Malware.exe.malz`”
  - “`$ cat floss.out | grep nim | wc -l`”
  - which gives output of “37”
  - We can conclude that the binary is written in “nim” Programming Language.
  - Some other interesting strings -

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

- **PEView/PE-bear :**
  - Compression Analysis
    - Section Headers (.text)
      - Marginal difference between raw size and virtual size
      - It’s an un-packed binary
  - Compiled Time/Date
    - Image_file_headers : “Thu 7.10.2021 17:43:04 UTC”
  - Suspicious Imports
    - kernel32.dll
    - mscvrt.dll
    - user32.dll

- **PEStudio**
    - Indicators
        - Suspicious Strings : 18
        - Suspicious Imports : 4
    - Flagged Imports/API calls
        - Enumeration
            - GetCurrentProcessId
            - GetCurrentThreadId
        - Injection
            - VirtualProtect
        - Helper
            - TerminateProcess
    - Flagged Strings
        - Apart from Import Strings
        - “CreateProcess”
        - “OpenProcess”
        - “Suspend Thread”
        - “WriteProcessMemory”

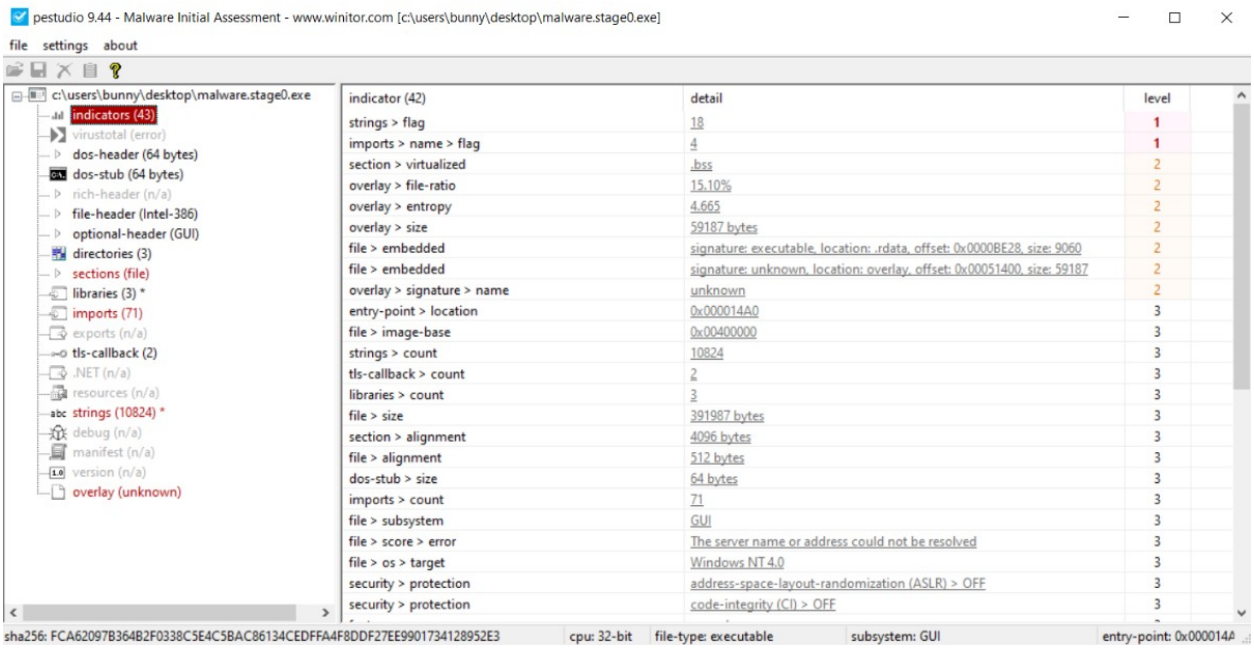

Illustration 2: PEStudio - Indicators

Basic Dynamic Analysis

- **Host Based Indicators :**
  - When Malware is detonated a new file is created C:\Users\Public\ werflt.exe

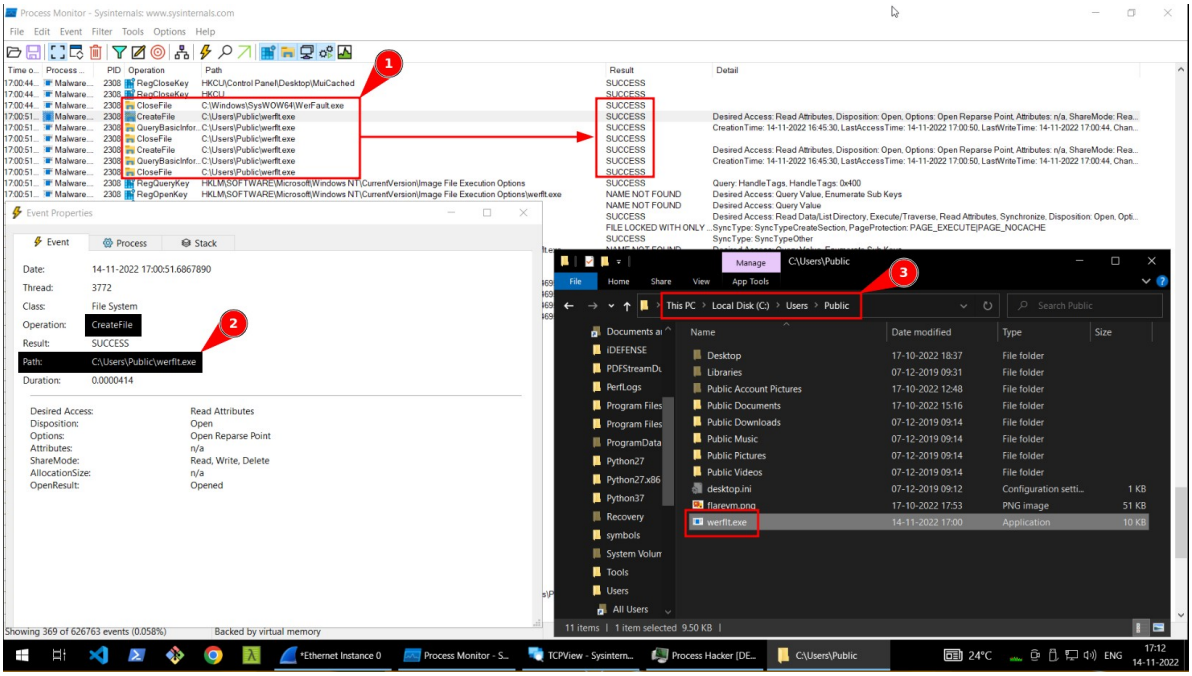

Illustration 3: ProcMon catches a new file creation- C:\Users\Public\werflt.exe

  - Process Tree in Procmon shows werflt.exe
    - werflt.exe is executed with parameter PID of WerFault.exe

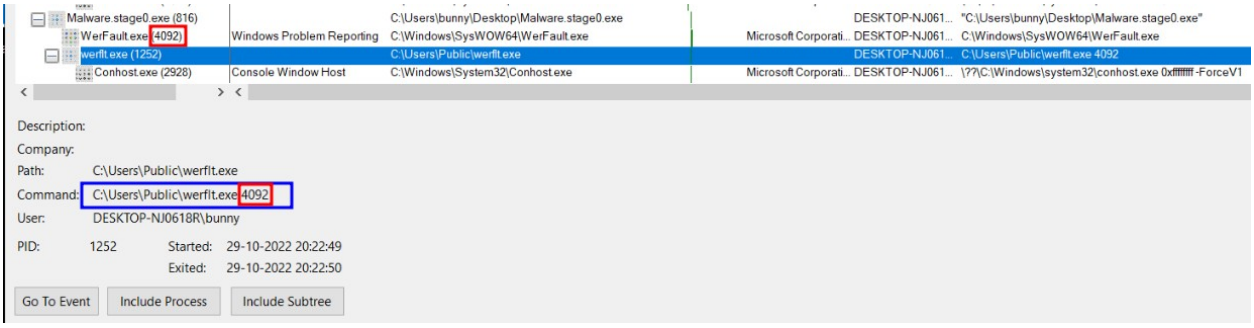

Illustration 4: ProcessTree - Malware runs werfault.exe, which runs werflt.exe

- **Network Based Indicators**
  - TcpView catches a process “WerFault.exe” which opens port 8443 on localhost.

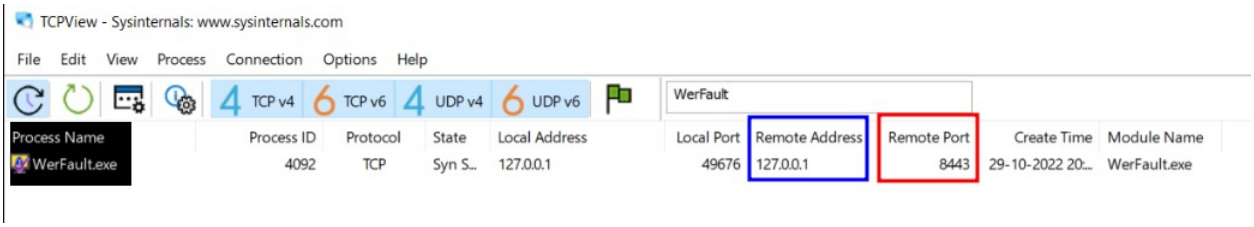

Illustration 5: TCPView - Process WerFault opens port on localhost

- We can try and connect to this port
  - Starting a listener on machine `ncat.exe -lvnp 8443`

```
C:\Users\bunny
λ ncat.exe -lvnp 8443
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:8443
Ncat: Connection from 127.0.0.1:49675.
Microsoft Windows [Version 10.0.19043.2130]
(c) Microsoft Corporation. All rights reserved.

FLARE 16-11-2022 16:02:25.04
C:\Users\bunny\Desktop>whoami
whoami
desktop-nj0618r\bunny
```

Illustration 6: netcat set up to listen on port 8443

Aadvanced Static Analysis

- Cutter – Disassembled code if werflt.exe

```
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...
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
```

Illustration 7: Decompiled Binary in ASM

- Analysis of De-compiled x86 Binary :
  - Main Function takes in “arg_ch” parameter
  - variable is defined in main “lpBuffer”
  - “arg_ch is moved to “eax”
    - This eax will be used in future API call
  - 1st API call : OpenProcess
    - It takes 3 parameters
      - Desired Access Level (0x1fffff)
      - Boolean InheritHandle (0)
      - ProcessId (eax)
  - “eax” is moved to “edi”
  - 2nd API call : VirtualAlloc
    - Allocates memory in the opened process
    - It takes 5 parameters
      - hProcess (edi)
      - lpAddress (0)
      - Size (0x145)
      - AllocationType (0x3000)
      - Protect (0x40)
  - 3rd API call : WriteProcessMemory
    - Writes shellcode in the allocated memory
    - Takes 5 parameters
      - hProcess (edi)
      - lpBaseAddress (esi)
      - lpBuffer (eax) (IPBuffer)
      - Size_T nSize (0x145)
      - lpNumberOfBytesWritten (0)
  - 4th API call : CreateRemoteThread
    - 7 parameters
      - First is hProcess (edi)
      - Fourth is lpStartAddress (esi)
      - Every other parameter is 0
  - 5th API call : Closehandle
    - 1 parameter
      - Handle Hobject (edi)
  - Fourth call creates Remote thread on local machine.

Advanced Dynamic Analysis

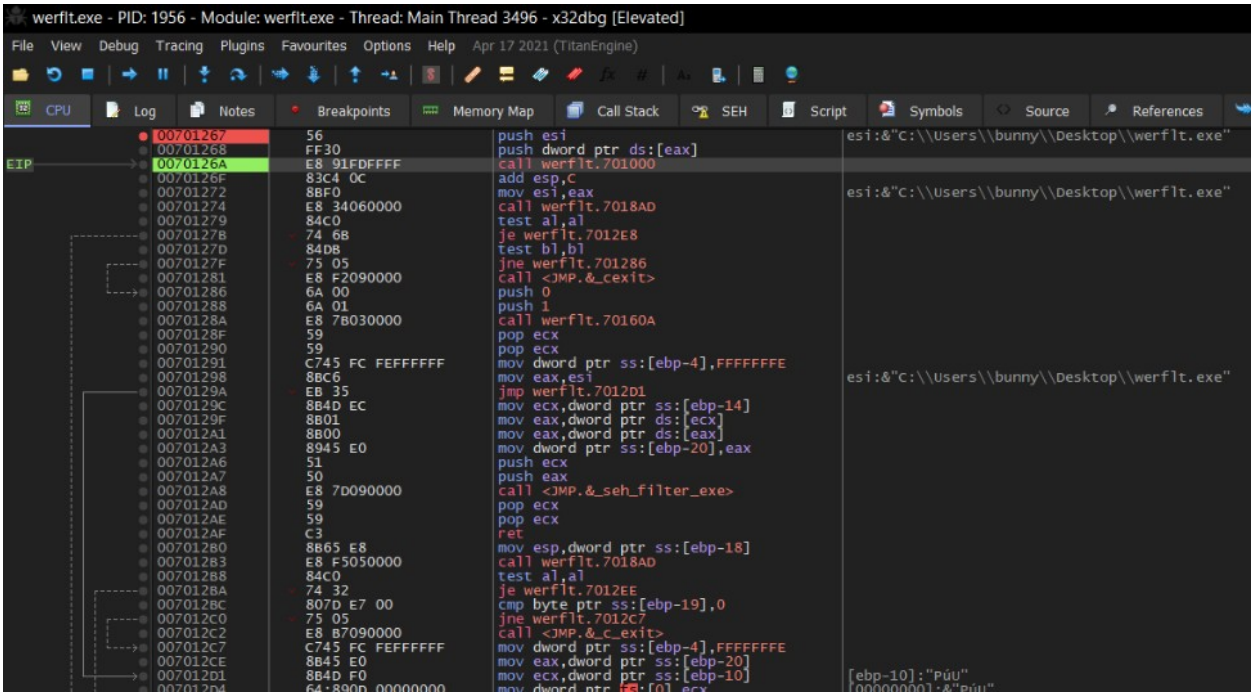

Illustration 8: werflt.exe debugging

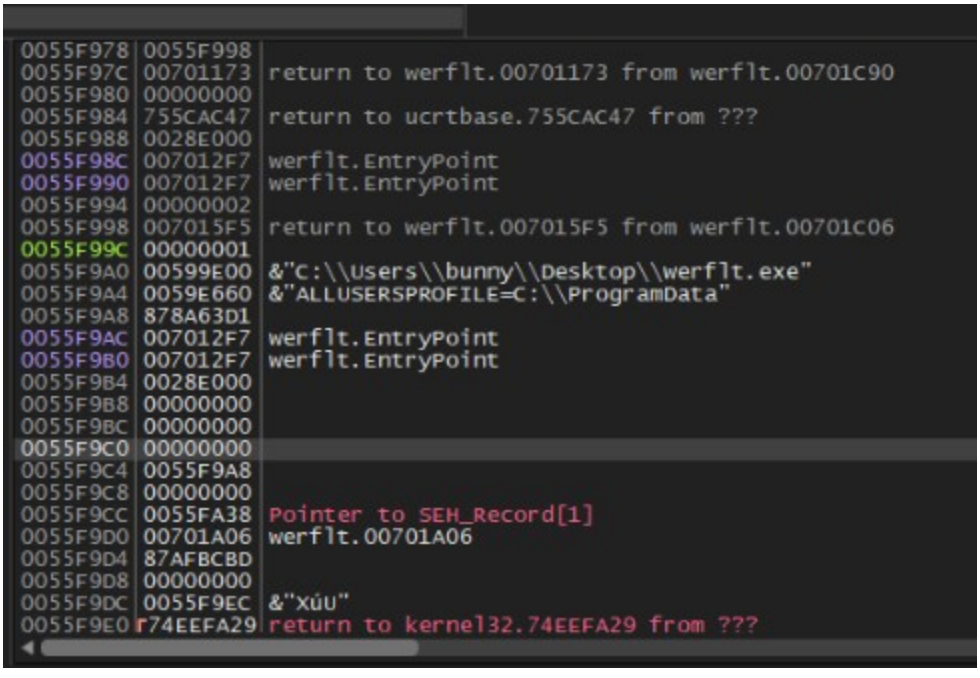

Illustration 9: werflt.exe Stack

Indicators of Compromise

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

Network Indicators

Werflt.exe injects shell code in WerFault.exe and executes it with its PID.
The Process persist when a shell is connected to a remote machine.

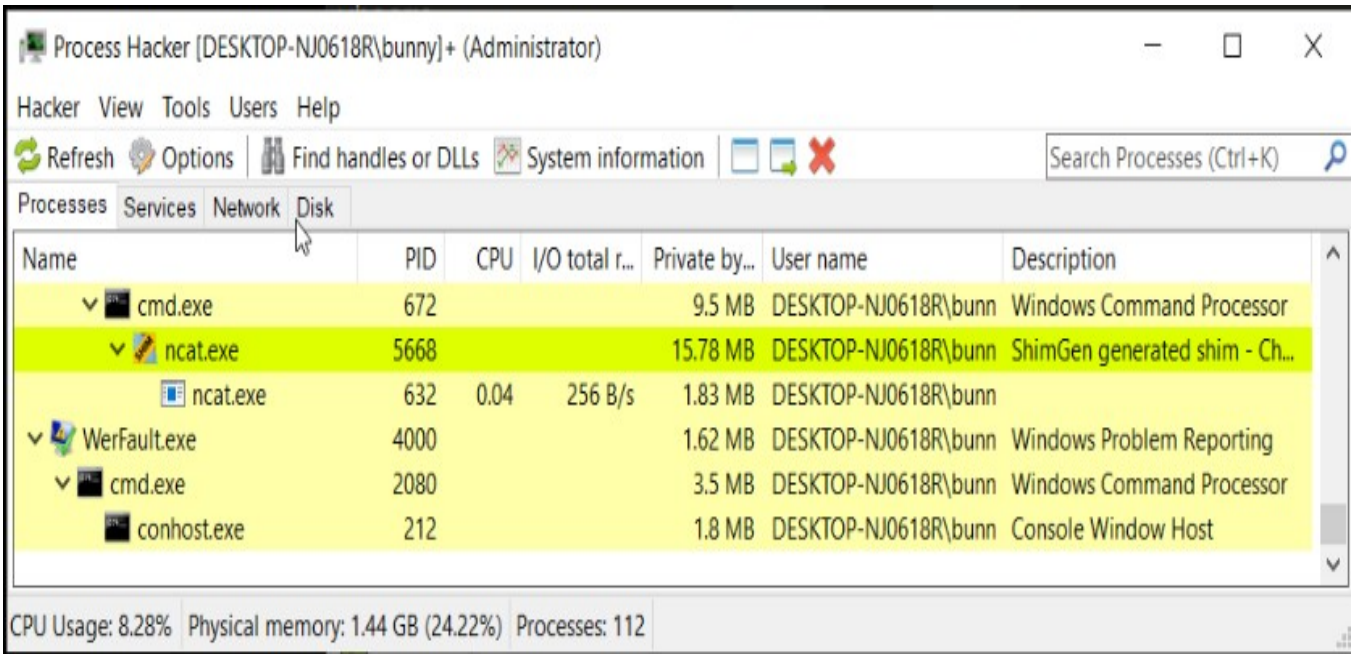

Illustration 10: werflt.exe runs shellcode as WerFault.exe

Host-based Indicators

We can open this process to see memory protections, There's one with Full Read-Write-Execute (RWX) permission, which is very suspicious for a process like WerFault which is a error logging process.

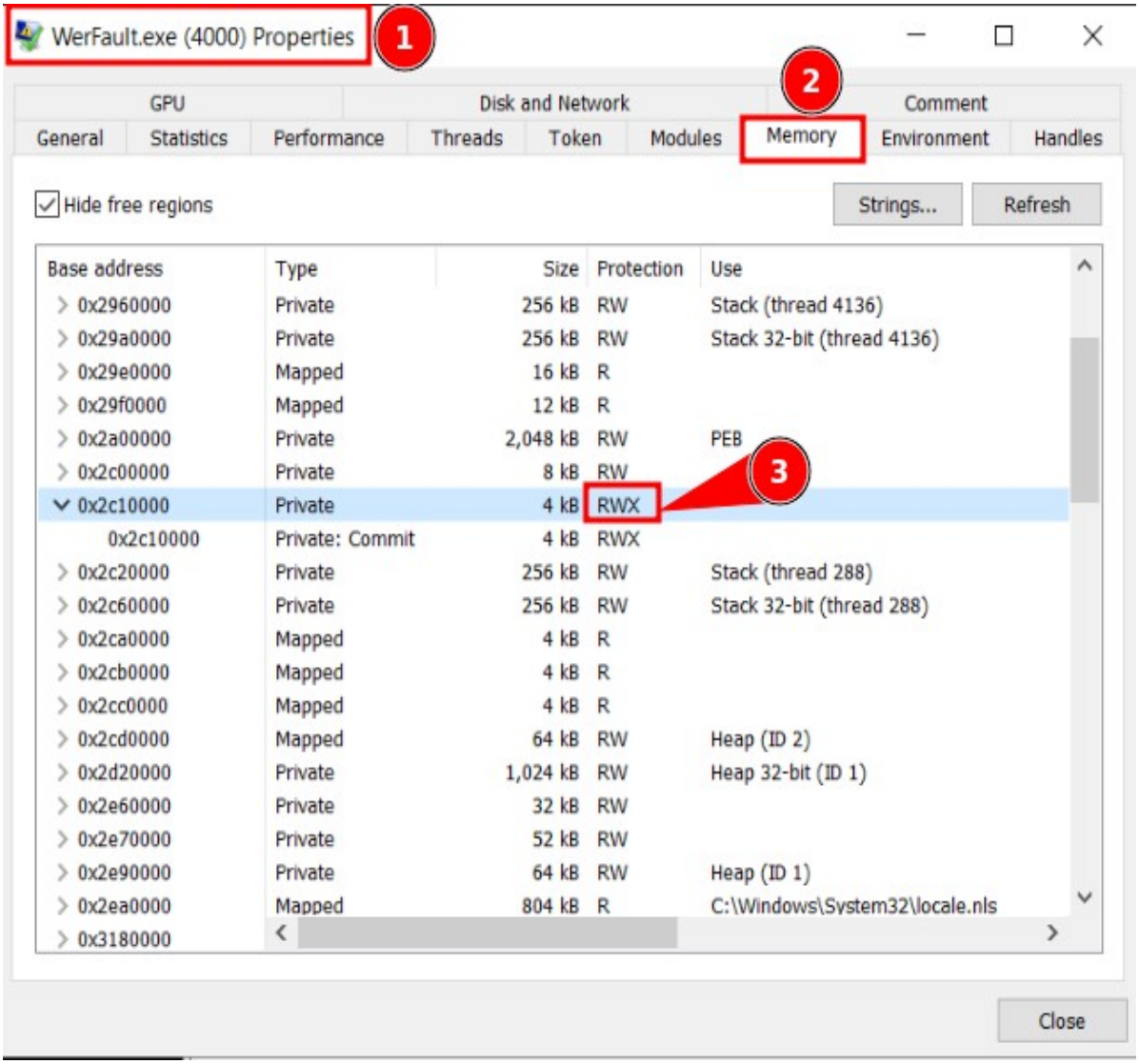

Illustration 11: Werfault.exe has R-W-X

Rules & Signatures

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

- String1 is set as Nim
- String2 is set as werflt.exe
- Suspicious bit is added as string
- Magic Number is MZ so 32-bit binary
- Callback URL is localhost on port 8443

Appendices

A. Yara Rules

```
rule stage0_malware {

    meta:
        last_updated = "2022-11-16"
        author = "NoobUltraProMax"
        description = "Yara signature of Malware.stage0.exe"

    strings:
        $string1 = "nim"
        $string2 = "C:\Users\Public\werflt.exe" ascii
        $hex_string = { 43 52 54 49 6E 6A 65 63 74 6F 72 43 6F 6E 73 6F }
        $PE_magic_byte = "MZ"

    condition:
        $PE_magic_byte at 0 and
        ($string1 and $string2) or

        $sus_hex_string
}
```

B. Callback URLs

Domain	Port
Hxxp://localhost	8443

C. Hex ShellCode Snippets

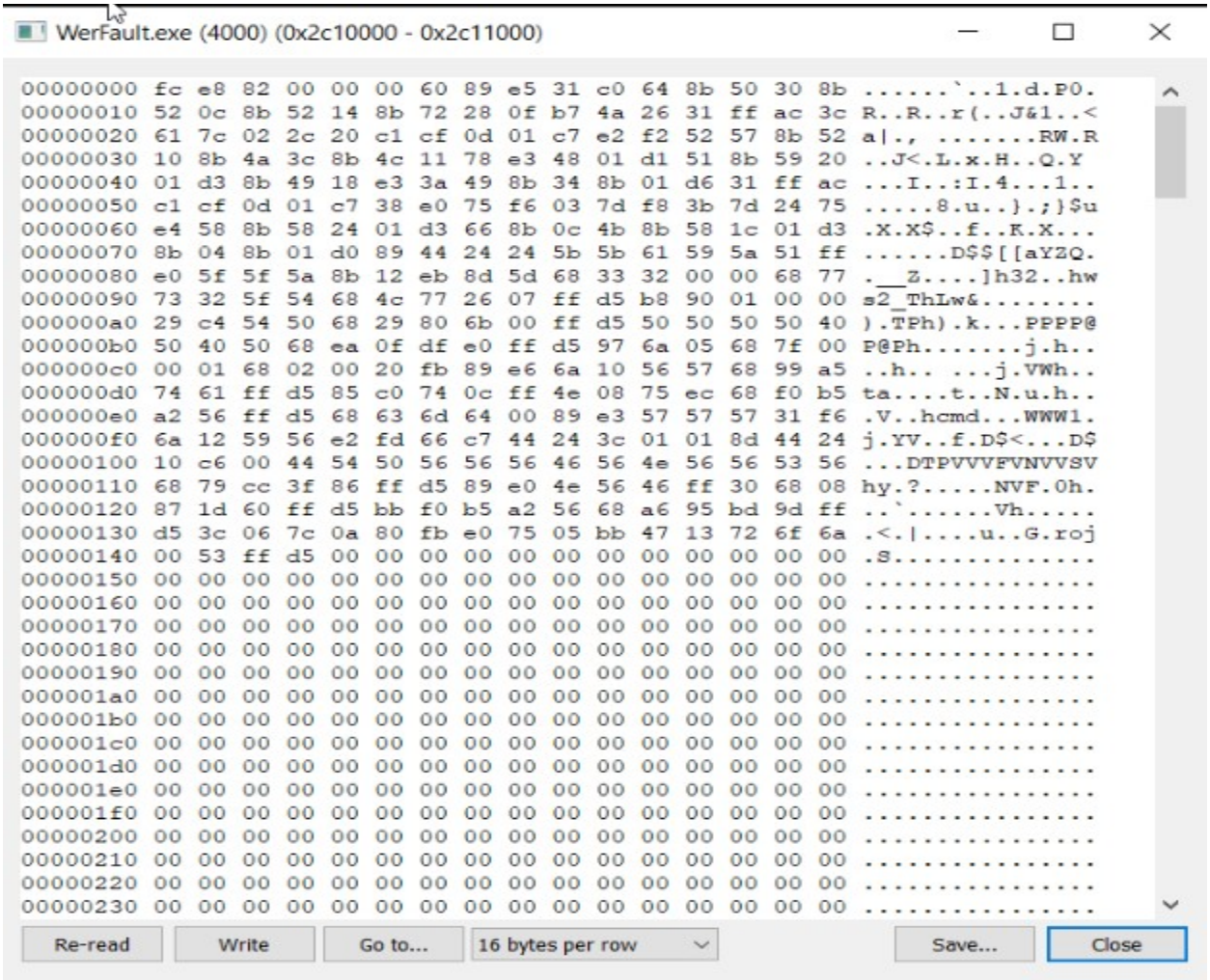

Illustration 12: Process Injection Routine in Cutter