Beyond good ol' Run key, Part 119

hexacorn.com/blog/2019/10/11/beyond-good-ol-run-key-part-119

October 11, 2019 *in Anti-Forensics, Autostart (Persistence)*

Pretty much everyone knows about the AEDebug key.

Turns out this key has a twin brother, called *AeDebugProtected*.

When the *WerpGetDebugger* function is called, it calls *NtQueryInformationProcess* to retrieve a basic information about the process. If the process's basic info data at position 28 (32-bit!) returns a dword value that if masked with 1 is non-zero then *AeDebugProtected* key is used...

Okay, this is confusing... Let's step back.

The traditional way of calling *NtQueryInformationProcess* with *ProcessBasicInformation* class is typically delivered using a 'classic' definition of _PROCESS_BASIC_INFORMATION structure being:

```
typedef struct _PROCESS_BASIC_INFORMATION {
    PVOID Reserved1;
    PPEB PebBaseAddress;
    PVOID Reserved2[2];
    ULONG_PTR UniqueProcessId;
    PVOID Reserved3;
} PROCESS_BASIC_INFORMATION;
```

Of course, available source codes online can give us a more precise definition of this structure e.g. Process Hacker source code defines it as:

```
typedef struct _PROCESS_BASIC_INFORMATION
{
    NTSTATUS ExitStatus;
    PPEB PebBaseAddress;
    ULONG_PTR AffinityMask;
    KPRIORITY BasePriority;
    HANDLE UniqueProcessId;
    HANDLE InheritedFromUniqueProcessId;
} PROCESS_BASIC_INFORMATION, *PPROCESS_BASIC_INFORMATION;
```

In both cases the size of a structure is 24 bytes (32-bit system!). When I looked at the code of *WerpGetDebugger* I was surprised to see that some calls to *NtQueryInformationProcess* / *ProcessBasicInformation* rely on a structure that is 32 bytes long!

Okay, so now we know there is some extra information provided by *NtQueryInformationProcess* to *WerpGetDebugger* function and that data determines which debug key is being used. Since this *ntdll* function is simply passed to kernel, I went to look at the code of *NtQueryInformationProcess* inside *ntoskrnl.exe*. I quickly discovered that the function does indeed expect a structure that is either 24 or 32 bytes long. Cool.

Continuing my analysis I noticed that the the field at the position 28 is filled in with a result of a call to a function called *PsIsProtectedProcess*. Protected processes [DOC warning] is a technology described in the past, so not a biggie. And the fact this is what is being checked by the function is of course something we should have expected, given the name used by the Registry Key I mentioned earlier, however... at least we can confirm this with our code analysis...

```
switch ( ProcessInformationClass )
{
  case 0:
    if ( ProcessInformationLength == 32 )
    {
      tmpProcessInformation = ProcessInformation;
      *ProcessInformation = 32;
      ms_exc.registration.TryLevel = -2;
      v6 = ProcessInformation + 4;
      tmpProcessInformationLen = 32;
    }
    else
    {
      if ( ProcessInformationLength != 24 )
        goto LABEL_328;
      tmpProcessInformation = 0;
      tmpProcessInformationLen = 24;
    }
```

And here we are with a few conclusive bits:

```
*(tmpProcessInformation + 28) = 0;
if ( PsIsProtectedProcess(v19) )
  *(tmpProcessInformation-2 + 28) = 1;
```

- *NtQueryInformationProcess* / *ProcessBasicInformation* may use 2 different structure versions!
- The field at offset 28 (32-bit!) tells us if the process is protected or not. Depending on this, different *AeDebug* Registry key will be used to launch debugger when the app crashes.

The longer structure can be prototyped as this:

```
typedef struct _PROCESS_BASIC_INFORMATION_EXT
{
NTSTATUS ExitStatus;
PPEB PebBaseAddress;
ULONG_PTR AffinityMask;
KPRIORITY BasePriority;
HANDLE UniqueProcessId;
HANDLE InheritedFromUniqueProcessId;
ULONG unknown;
ULONG IsProtectedProcess;
}
```

At this is how we arrived at *Beyond good ol' Run key, Part 119*.

Okay, not quite yet.

The value under this Protected Registry key that the *WerpGetDebugger* function will use is not *Debugger*, but *ProtectedDebugger*. Yup, we are talking:

> HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebugProtected\ProtectedDebugger=<exe>

Comments are closed.