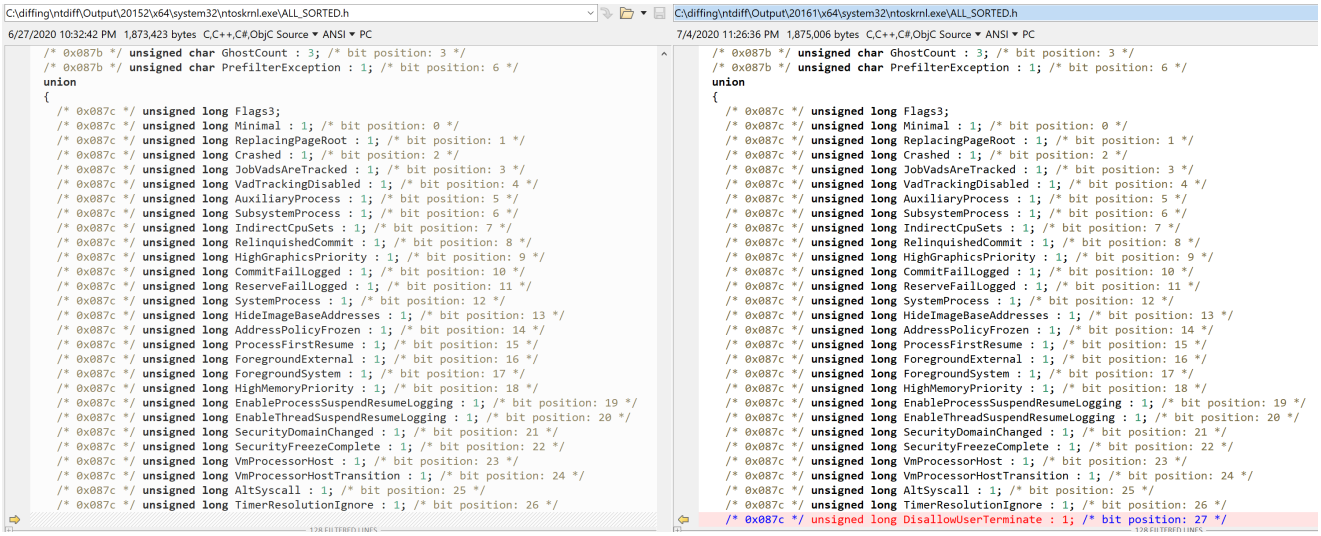
# Critical, Protected, DUT Processes in Windows 10

windows-internals.com/dut-processes-in-windows-10

By Yarden Shafir

We are all familiar with Microsoft's love for creating new and exciting ways to prevent certain processes from being terminated by the user. First were Critical processes in Windows XP 64-bit and Server 2003, which crashed the kernel if you killed them. Then, came Protected Process Light (PPL) in Windows 8.1, which prevented you from killing them at all. Perhaps it prevented too many other things too, because in a recent Windows 10 update, build 20161, we see yet another new addition to the `EPROCESS` flags ( `Flags3` , actually), called `DisallowUserTerminate` :




As this flag's name is pretty clear, its purpose doesn't need much explanation – any process that has this flag set cannot be terminated from user-mode. We can see that in `PspProcessOpen` :

```
if ( (*GrantedAccess & PROCESS_TERMINATE) != 0
  && AccessMode == UserMode
  && (TargetProcess->Flags3 & DisallowUserTerminate) != 0 )
{
  return STATUS_ACCESS_DENIED;
}
```

A user-mode caller can't open a handle to a process that has the `DisallowUserTerminate` flag set if the requested access mask contains `PROCESS_TERMINATE` .

So where is this flag set, and does this mean you can protect your processes from termination? The answer to the second question is simple – not really. For now, this flag can only be set by one path, and it's one specifically used for creating Hyper-V Memory Host ( `vmmem` ) processes.

Internally, this flag is set on process creation by `PspAllocateProcess`, based on the input parameter `CreateFlags` – flag 8 (let's call it `PSP_CREATE_PROCESS_FLAG_DISALLOW_TERMINATE`) is what sets `DisallowUserTerminate` as you can see below:

```
if ( (CreateFlags & 1) != 0 )
  process->Flags3 |= SystemProcess;
if ( (CreateFlags & 4) != 0 )
  process->Flags3 |= VmProcessorHost;
if ( (CreateFlags & 8) != 0 )
  process->Flags3 |= DisallowUserTerminate;
```

Unfortunately, this function only has 2 external callers, which always pass in 0 as `CreateFlags`, which obviously doesn't allow one to set any of these flags. The third, internal caller, is `PsCreateMinimalProcess`, which has a few internal uses in the system, such as the creation of Pico Processes used by WSL, and other special system processes such as "Memory Compression" and "Registry". Minimal processes are also created by `VmCreateMemoryProcesses`, which is one of the APIs that's exported through the VID Extension Host that myself, Gabrielle, and Alex described in our INFILTRATE 2020 talk.

Unlike the exported functions, the `PsCreateMimimalProcess` internal API receives the `CreateFlags` from its callers and forwards them to `PspAllocateProcess`, and `VmCreateMemoryProcesses` passes in `PSP_CREATE_PROCESS_FLAG_DISALLOW_TERMINATE` (0x8) unconditionally, as well as `PSP_CREATE_PROCESS_FLAG_VM_PROCESSOR_HOST` (0x4) if flag 0x20 (let's call it `VMP_CREATE_PROCESS_FLAG_VM_PROCESSOR_HOST`) was sent to it. You can see this logic below:

```
internalCreateFlagsTemp = ((CreateFlags & 1) << 13) | 0x4000;
if ( (CreateFlags & 2) == 0 )
  internalCreateFlagsTemp = (CreateFlags & 1) << 13;
internalCreateFlags = internalCreateFlagsTemp | 0x20000;
if ( (CreateFlags & 4) == 0 )
  internalCreateFlags = internalCreateFlagsTemp;
internalCreateFlagsTemp = internalCreateFlags | 0x400;
if ( (CreateFlags & 0x10) == 0 )
  internalCreateFlagsTemp = internalCreateFlags;
status = PsCreateMinimalProcess(
           ParentProcess,
           ProcessName,
           0i64,
           ParentProtection,
           (_TOKEN *)Token,
           internalCreateFlagsTemp,
           (CreateFlags & 0x20 | 0x40u) >> 3,// VmProcessorHost | DisallowUserTerminate
           0i64,
           Job,
           &ProcessHandle);
```

As mentioned, looking for callers for this function in IDA will not show any results, because this function, which is not exported, is shared with Vid.sys through an extension host and called by `VsmmNtSlatMemoryProcessCreate` when new `vmmem` processes are needed to manage memory in virtual machines managed by Hyper-V, and/or to contain the Virtual Processor (VP) scheduler threads when eXtended Scheduling (XS) is enabled as part of Windows Defender Application Guard (WDAG), Windows Containers, or Windows Sandbox.

Checking the value of `Flags3` in `vmmem` processes in the new build shows that `DisallowUserTerminate` is enabled for these processes, and:

```
1: kd> dx @$cursession.Processes.Where(p => p.KernelObject.DisallowUserTerminate).Select(p => p.Name)
@$cursession.Processes.Where(p => p.KernelObject.DisallowUserTerminate).Select(p => p.Name)
    [0x1a80]         : vmmem
    [0x169c]         : vmmem
```

Sadly, no other process can use this capability for now without manually editing the `EPROCESS` structure, which is extremely not recommended, as any code doing this is bound to break often and crash a lot of systems. So I'm sure 5 different AV companies are already adding code to it.

Read our other blog posts: