

Forum:[Hack Forums](#)

Thread: [Critical Process Protection](#)

MrPoBot - October 6, 2019, 9:33 am

Hey HF, as part of my campaign to become more widely known on this forum i have decided to start sharing some of my more interesting code snippets with yall, hope you enjoy!

The code below protects the process using 2 Native API calls to the windows kernel, once the void has been called the only safe way to terminate the process would be for the thread itself, or a program of equal integrity to call a de-escalation from within the calling process.

```
Code using Microsoft.Win32;
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Threading;
```

```
namespace Client.Helper
{
    public static class ProcessCritical
    {
        public static void SystemEvents_SessionEnding(object sender, SessionEndingEventArgs e)
        {
            if(true) //replace true with whatever condition must be met
                Exit();
        }
        public static void Set()
        {
            try
            {
                SystemEvents.SessionEnding += new
                SessionEndingEventHandler(SystemEvents_SessionEnding);
                Process.EnterDebugMode(); //entering debug mode is essential for this specific call to
                work
                RtlSetProcessIsCritical(1, 0, 0); //add the critical flag for the current process
            }
            catch { }
        }
        public static void Exit()
        {
            try
            {
                RtlSetProcessIsCritical(0, 0, 0); //remove the critical flag from the current process
            }
            catch
            {
                while (true)
                {

```

```

        Thread.Sleep(-1); //Permanet Thread.Sleep to prevent BSOD if the process is unable
to de-escalate
        //you could add a custom error handeler here or just continue to the process but doing
so isnt recommended
    }
}
}

#region "Native Methods"
[DllImport("ntdll.dll", SetLastError = true)]
private static extern void RtlSetProcessIsCritical(UInt32 v1, UInt32 v2, UInt32 v3);
#endregion
}
}
}

```

MavenGradle - October 6, 2019, 1:19 pm

Code try

```

{
    SystemEvents.SessionEnding += new
SessionEndingEventHandler(SystemEvents_SessionEnding);
    Process.EnterDebugMode(); //entering debug mode is essential for this specific call to
work
    RtlSetProcessIsCritical(1, 0, 0); //add the critical flag for the current process
}
catch { }

```

What kind of error handling is this? It serves absolutely no purpose and you should instead opt to perform proper error handling.

Code while (true)

```

{
    Thread.Sleep(-1); //Permanet Thread.Sleep to prevent BSOD if the process is unable
to de-escalate
    //you could add a custom error handeler here or just continue to the process but doing
so isnt recommended
    }I'm going to kill myself. A while (true) to make it impossible to correctly close the process
and a Thread.Sleep which can be avoided if you coded this properly. Two of the worst bad practices
in the same code snippet. Well done.

```

Code public static void SystemEvents_SessionEnding(object sender, SessionEndingEventArgs e)

```

{
    if(true) //replace true with whatever condition must be met
        Exit();
    }This makes absolutely no sense. A programmer is supposed to recompile his code every time
he wants to exit or not?

```

You'll be famous alright but not for the right reasons.

MrPoBot - October 8, 2019, 12:39 am

(Oct 6, 2019 01:19 PM)MavenGradle Wrote: [\[CITE\]](#) Code try

```

{
    SystemEvents.SessionEnding += new
SessionEndingEventHandler(SystemEvents_SessionEnding);
    Process.EnterDebugMode(); //entering debug mode is essential for this specific call to

```

work

```
RtlSetProcessIsCritical(1, 0, 0); //add the critical flag for the current process
}
```

catch { }

What kind of error handling is this? It serves absolutely no purpose and you should instead opt to perform proper error handling.

```
Code while (true)
{
    Thread.Sleep(-1); //Permanet Thread.Sleep to prevent BSOD if the process is unable
to de-escalate
    //you could add a custom error handeler here or just continue to the process but doing
so isnt recommended
    }I'm going to kill myself. A while (true) to make it impossible to correctly close the process
and a Thread.Sleep which can be avoided if you coded this properly. Two of the worst bad practices
in the same code snippet. Well done.
```

```
Code public static void SystemEvents_SessionEnding(object sender, SessionEndingEventArgs e)
{
    if(true) //replace true with whatever condition must be met
        Exit();
    }This makes absolutely no sense. A programmer is supposed to recompile his code every time
he wants to exit or not?
```

You'll be famous alright but not for the right reasons.
Ouch, the idea behind this is to let others modify this to suit their own needs. As for the error handling the alternative is a BSOD if a exception is called, i chose the lesser of 2 evils and explicitly state that you should probably do this a different way such as wait and retry or give the user a prompt to save & close everything before calling the exception. However, i wont shy away from criticism and while harsh the code has flaws which need fixing perhaps propose changes rather then calling me shit?

Vyxn - October 8, 2019, 4:08 am
(Oct 6, 2019 01:19 PM)MavenGradle Wrote: [\[CITE\]](#) ...
Are you Odus?

Mike024WK_ - October 8, 2019, 7:30 am
Another way that can be used on other processes aswell

```
WINAPI ->
[DllImport("ntdll.dll", SetLastError = true)]
private static extern int NtSetInformationProcess(IntPtr hProcess, int processInformationClass, ref
int processInformation, int processInformationLength);
```

```
CODE ->
int isCritical = 1;
int BreakOnTermination = 0x1D;
Process.EnterDebugMode();
NtSetInformationProcess(Process.GetCurrentProcess().Handle, BreakOnTermination, ref isCritical,
sizeof(int));
```

NOT MY CODE, Found on the WWW.

(Oct 8, 2019 12:39 AM)MrPoBot Wrote: [\[CITE\]](#) Ouch, the idea behind this is to let others modify this to suit their own needs. As for the error handling the alternative is a BSOD if a exception is called, i chose the lesser of 2 evils and explicitly state that you should probably do this a different way such as wait and retry or give the user a prompt to save & close everything before calling the exception. However, i wont shy away from criticism and while harsh the code has flaws which need fixing perhaps propose changes rather than calling me Clay Davis?

What I mean by your error handling is that you implement no logging at all. There's no way for a user to know what went wrong, or hell, even other developers (since this is an undocumented unmanaged function in ntdll). Don't get me wrong, I'm not trying to demotivate you or stop you from posting, this is simply done to point out some logically flawed statements. I was hoping that by pointing out those flaws, you might be able to figure it out yourself. When you're using undocumented code, you should better realize what you're doing or what you're trying to accomplish instead of complaining about random BSOD's. See this for an explanation why:

<https://codingvision.net/tips-and-tricks/c-make-a-critical-process-bsod-if-killed>

Quote:Whenever a process like this finishes its execution (or it's terminated) Windows will respond with an authentic Blue Screen of Death. Makes sense, right? This isn't a bug. This is specifically built into Windows because guess what, critical processes are supposed to be CRITICAL to the functioning of the operating system. That's why this function is undocumented, to prevent people from abusing this and because it serves no purpose to userland applications (most likely, or the devs didn't have enough time to do so).

Ideally what you'd do in a situation like this is listen to the return value of the function or if it doesn't return a status (normally it should return an [NTSTATUS](#)), [check the process critical flag](#) (or check with [ZwQueryProcessInformation](#) function with the ProcessBreakOnTermination [0x1D] parameter for windows versions older than 8.1) after attempting to remove it and if it fails perhaps even check why it fails since you tell it to set the last error:

```
Code [DllImport("ntdll.dll", SetLastError = true)]
```

You can check said error status using the following code:

```
Code Marshal.GetLastWin32ErrorAfter
```

which you should take the appropriate action. For example, retry it. If the flag can't be removed from your process, you're bound to run into BSOD's because that's simply normal windows behavior. One might even argue that this is ideal for skid malware that automatically runs when the computer boots up.

But there you go, the description above should be more than enough to you or anyone else looking to fix your code. I give zero fucks about your code, because like I said, it serves no purpose in userland applications unless you're a malware developer which I'm not, nor do I want to help malware developers. The analysis that I wrote above is simply there to once again point out the flawed logic.

(Oct 8, 2019 04:08 AM)Vyxn Wrote: [\[CITE\]](#) Are you Odus?

No, I'm MavenGradle. You can see the username of a person by looking to the left of the post (or just above the post if you're using a smaller screen or mobile device).

(Oct 8, 2019 07:30 AM)Mike024WK_ Wrote: [\[CITE\]](#) Another way that can be used on other processes aswell

WINAPI ->

```
[DllImport("ntdll.dll", SetLastError = true)]
```

```
private static extern int NtSetInformationProcess(IntPtr hProcess, int processInformationClass, ref int processInformation, int processInformationLength);
```

CODE ->

```
int isCritical = 1;  
int BreakOnTermination = 0x1D;  
Process.EnterDebugMode();  
NtSetInformationProcess(Process.GetCurrentProcess().Handle, BreakOnTermination, ref isCritical,  
sizeof(int));
```

NOT MY CODE, Found on the WWW.

See description in this comment (above) to see why this code snippet is also not properly set up. Yes, this one can be used to specify a different process by using the pointer to the process handle, but you're still not doing it right.