LSASS Memory Dumps: New Method for Dumping LSASS [Part 2] | Deep Instinct

By Asaf GilboaSecurity Researcher

Published: 2021-02-16 · Archived: 2026-04-05 16:01:50 UTC

In a previous article, we detailed the numerous ways to dump LSASS memory for credentials extraction, in this article we show a new way to dump LSASS without dropping any new tool on the endpoint.

MITRE Technique: T1003.001

Technical Overview

There is a very neat way to cause WerFault.exe (Windows Error Reporting process that handles process crashes) to create a memory dump of lsass.exe, in a directory of your choice. The major advantage of this technique is that it does not cause lsass.exe to crash, and since WerFault.exe is used to create file dumps all the time (not just lsass.exe), this method provides the added advantage of going undetected. WerFault.exe is a process known for dumping every crashing process, from an attacker standpoint this is appealing as their illicit credential extraction will appear benign because from a defender's viewpoint it's within the realm of normal activity.

This method relies on a mechanism introduced in Windows 7 called **Silent Process Exit**, which provides the ability to trigger specific actions for a monitored process in one of two scenarios; either the process terminates itself by calling ExitProcess(), or another process terminates it via the TerminateProcess() API.

There are multiple actions that can be configured to occur upon a silent process exit:

- Launch a monitor process
- Display a pop-up
- Create a dump file

Option #1 can be used as a persistence mechanism. For the purpose of this study, we describe how to use option #3 for dumping lsass.

To set-up a process for silent exit monitoring, a few registry settings must be set:

1. 1. The GlobalFlag for the process' Image File Execution Options must be set to include the flag

FLG_MONITOR_SILENT_PROCESS_EXIT (0x200)

2. SilentProcessExit must be set by either:

a. Global settings, under the key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit

b. Application-specific settings, under the key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit*ProcessName*

The SilentProcessExit settings are set by registry values, for that purpose the interesting ones are the following:

ReportingMode (REG_DWORD) – Bitwise OR of the following flags:

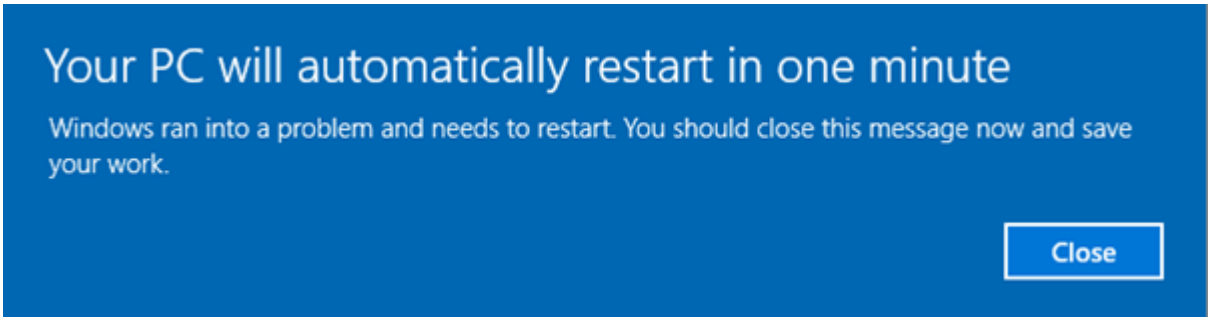
- LAUNCH_MONITORPROCESS (0x1) – Launch a monitor process
- LOCAL_DUMP (0x2) – Create a dump file for the process that caused the termination and the process that was terminated
- NOTIFICATION (0x4) – Display pop-up notification

LocalDumpFolder (REG_SZ) – The directory where the dump files will be created. Default location is %TEMP%\Silent Process Exit.

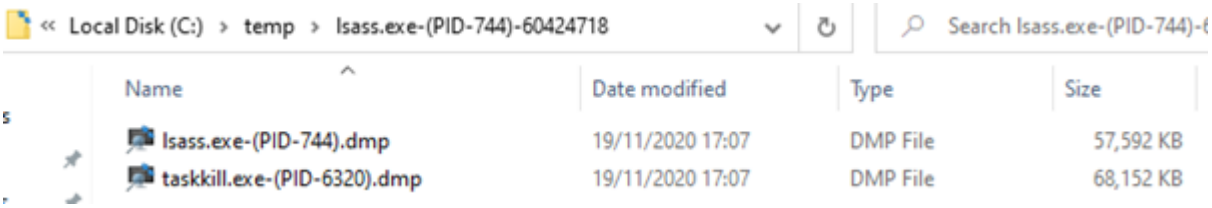
DumpType – Specifies the type of dump file (Micro, Mini, Heap or Custom) according to the MINIDUMP_TYPE enum. Full minidump is a value of MiniDumpWithFullMemory (0x2).

So, what would happen if the SilentProcessExit registry settings are set so that LSASS.exe will dump itself, and then either lsass.exe is killed or the computer is shut down?

To answer this, we use taskkill to terminate lsass. This brings up this message because Windows really doesn't like to have lsass.exe shut down:



A warning like this is problematic for an endpoint user to see during an attempt to gather credentials, but it does provide a new directory under C:\temp, which contains the full memory dump of lsass.



Nice!

A dump of taskkill.exe is also obtained which normally isn't accessible if the computer had been shut down, instead of terminating lsass.exe. This happened because the Silent Process Exit mechanism also causes the process that initiated the termination to be dumped as well.

The question we now need to ask ourselves is – how is the process dumped? Thanks to Hexacorn's blog, we know that when a process terminates it calls the RtlReportSilentProcessExit() API from ntdll.dll, which will communicate to the Windows Error Reporting service (WerSvc under WerSvcGroup) that the process is performing a silent exit. The WER service will then launch WerFault.exe which will do the dumping of the exiting process. The interesting thing to notice is that calling this API **does not cause the process to exit**. This prompted us to run this process on lsass.exe, to get the file dump, but without terminating lsass.

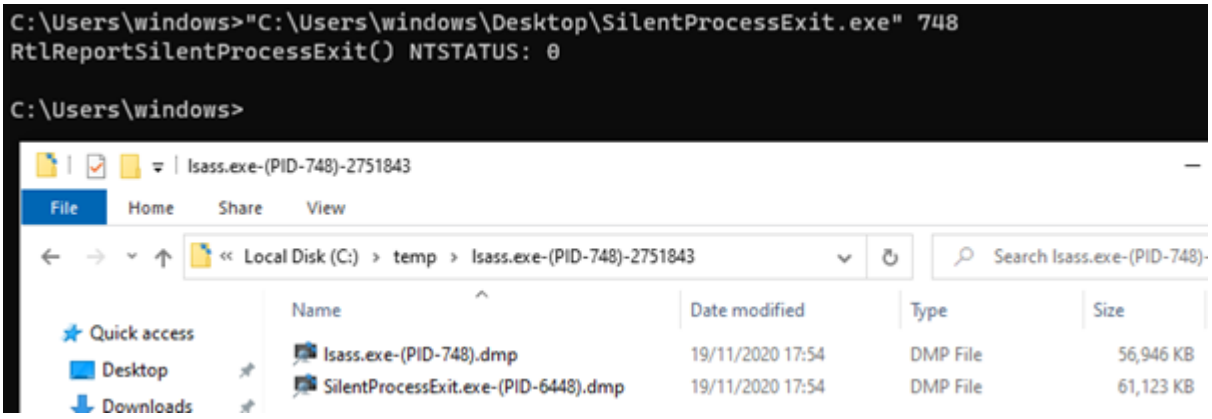
Here is the function definition of RtlReportSilentProcessExit():

```
<span style="color: #119fc2;">NTSTATUS</span><span style="color: #9d73c7;">(NTAPI</span>* <span styl
      <span style="color: #9d73c7;">_In_ </span><span style="color: #119fc2;">HANDLE</span>     Pro
      <span style="color: #9d73c7;">_In_</span><span style="color: #119fc2;">NTSTATUS</span>   Exit
      );
```

But what if we supply a ProcessHandle of LSASS.exe from OUTSIDE of LSASS?

Calling RtlReportSilentProcessExit this way would require a handle to lsass.exe with PROCESS_VM_READ permissions and also need the SeDebugPrivilege privilege, otherwise, the dump file will be created but without any content. In addition, an x64 process is required to open a handle to an x64 lsass process.

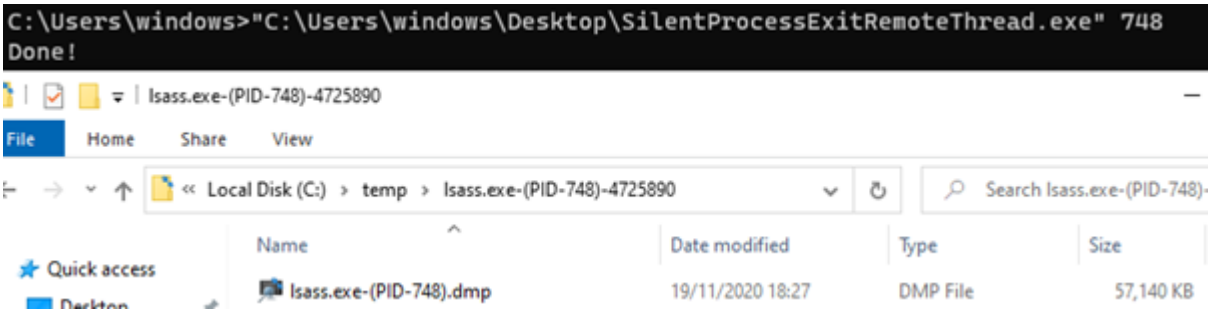
Using a program we have wrote that does just that, we can see here the dump file created:



That’s great!

Now, we can delete the unnecessary dump of our own process and send the lsass dump to our attacker server to have the credentials extracted.

But can we go even further and force lsass.exe to create a dump of itself? Using CreateRemoteThread on lsass.exe, we were able to cause it to run RtlReportSilentProcessExit:



Voila, Lsass.exe's own dump file!

From an EDR standpoint, it will appear as though lsass.exe requested a dump of itself from WER. Since WER is the mechanism in Windows which is responsible for creating dump files anyway, it is likely to be whitelisted as a process that creates a dump file of lsass.exe in order to reduce false-positives.

The code to perform both of these methods can be found in our GitHub repository.

Suggested Solutions

In the following section, we detail the measures that can be taken to detect dumping of the lsass.exe process.

Monitoring Registry

Set a rule of registry value creation of GlobalFlag:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\lsass.exe

GlobalFlag REG_DWORD 0x200

Note that GlobalFlag is a bitwise OR possibly numerous flags.

The following registry key should also be monitored for creation and for changes:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit\lsass.exe

Monitor Files

Set a rule for file creation for anything with the following pattern of file name:

"lsass*.dmp"

RunAsPPL

Windows enables the ability to launch the lsass.exe process as a Process Protected Light (PPL), which prevents any non-PPL process from using OpenProcess() to access lsass.exe. This neutralizes all methods described in this article (besides the full memory dump methods). The following registry value is required to be set:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa

RunAsPPL REG_DWORD 0x1

The downside of this method is that setting lsass.exe prevents any third-party DLLs from loading into it, including known and benign authentication packages. If your organization utilizes some smart-card solution, for example, this is not an option.

Due to the obscurity of this attack vector available AV and EDR solutions are not going to have these detection and mitigation configurations in place, rather they will need to be manually configured. In soon-to-be-delivered upcoming versions, Deep Instinct's customers can expect to have automatic protection from this technique within the credential dumping heuristic.

Summary

The numerous ways of dumping LSASS memory give attackers a range of options to stay undetected by antivirus products and EDRs. This new method that we have introduced to get a process dump of LSASS to disk, hasn't been utilized before while the use of WER has the added benefit of making the illicit memory extraction appear benign. This creates a ripe opportunity for hackers, with the possibility of many security environments having the file dump process whitelisted.
