Deception in Depth - LSASS Injection

blog.spookysec.net/DnD-LSASS-Injection

29 Aug 2021

Welcome back to the second post in the Deception in Depth series! Today we're going to be exploring LSASS injection, this is a fairly interesting topic and is actually suprisingly simple. On the surface LSASS injection might seem like a complex topic that requires a deep understanding of Windows, the Windows memory structure and knowledge of the LSASS process. It sounds like you might need to write directly into the LSASS process (which very well might be a way…), however, it's not!

Introducing HoneyCred!

A Security researcher by the name Stephen Hosom (0xHosom on Twitter) figured out an incredibly simple way to write into the LSASS process in a maner that doesn't cause it to crash using *one* Windows API call. As it turns out, this is the CreateProcessWithLogonW API. He made this handly little tool called HoneyCred (written in Go) to do it for us!

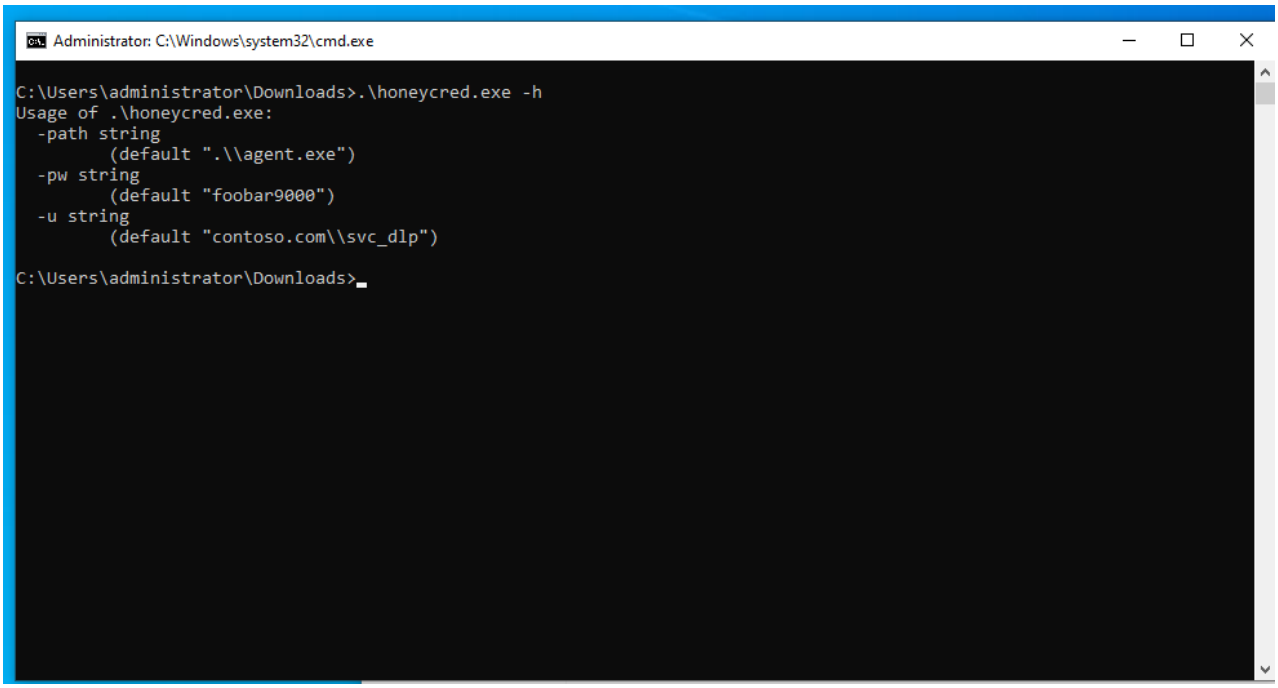
Labbin' it up

We're going to be using the same lab setup as last time. Here's a quick refresher:

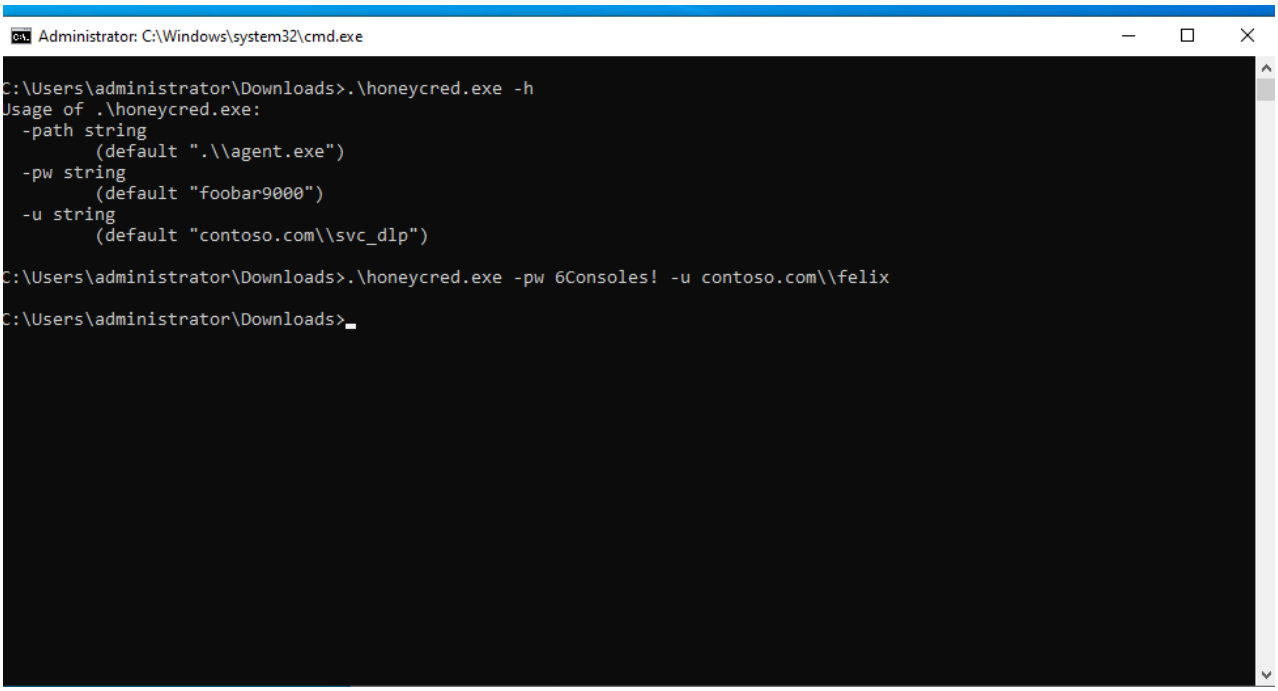
Domain: CONTOSO / contoso.com **Devices:** 1x DC (DC / dc.contoso.com) and 2x Workstations (WKS01 / wks01.contoso.com and WKS02 / wks02.contoso.com)

Let's hop into our lab environment and take a look at it in action! You can download HoneyCred directly from the releases tab on Github, the latest release is (unfortunately) the only release… However, I'd like to do something special later on in this post :D

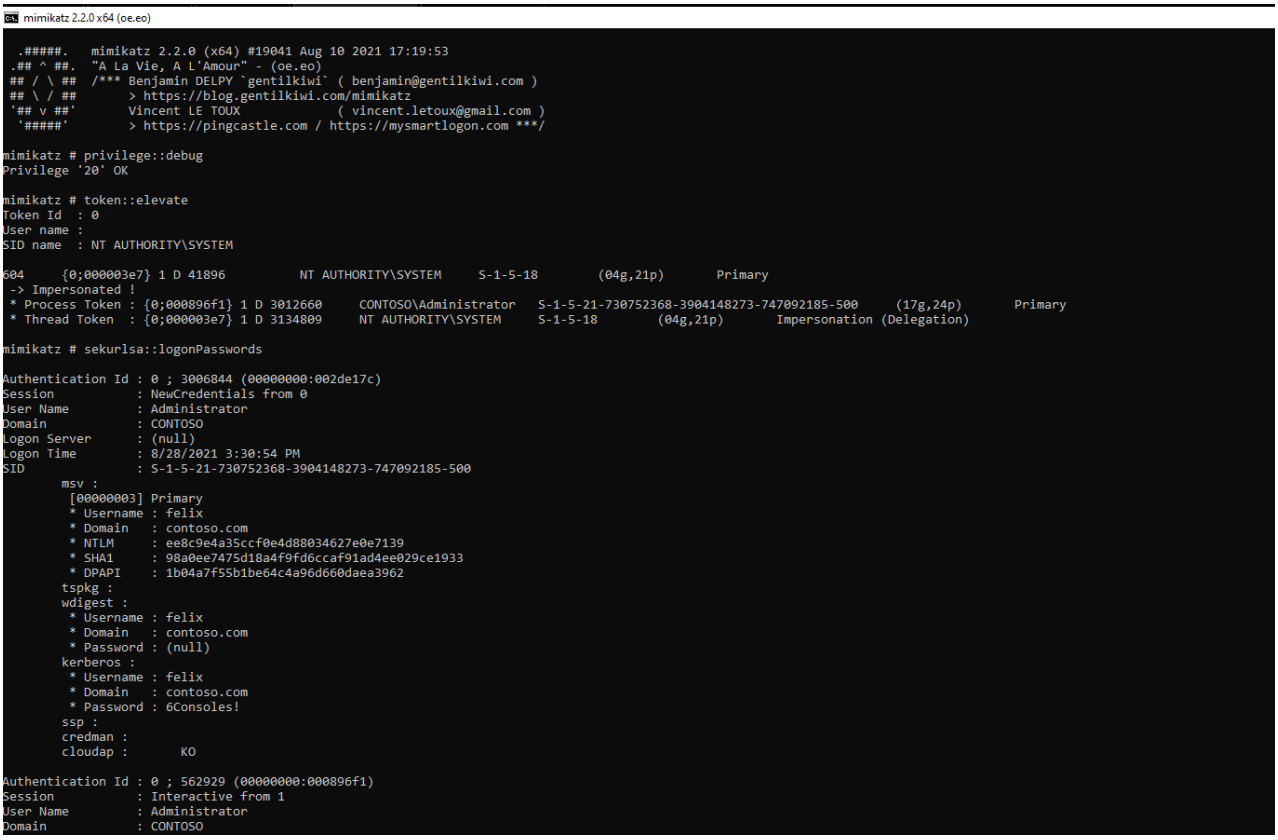
Help Menu - It's pretty straight forward, we've got 2 mandatory arguments (unless you want the credentials CONTOSO\svc_dlp:foobar9000 to be injected into memory) - the Username string (-u) and the Password string (-pw). The last option (-path) is used to specify an agent which injects into LSASS routinely every 3600 seconds



Let's try it out! We're going to attempt to inject the credentials "CONTOSO\Felix:6Consoles!" and then attempt to dump credentials with Mimikatz, Secretsdump, and CrackMapExec and see if our HoneyCreds show up.



It appears to have worked without any issues or errors. Let's start with Mimikatz.



Success! 1/1 it appears to work great so far! Lets move onto Secretsdump

Secretsdump appears to have failed. Lets move onto CrackMapExec

CrackMapExec also seems to have failed, unfortunately. I did some further testing to see if this happened to be an issue with injecting invalid credentials vs injecting valid credentials, and it seemed that it makes zero difference when dumping with Secretsdump

and CrackMapExec.

The one thing I might note is that dumping credentials with LSASSy might work due to the fact that it can dump the LSASS process in multiple ways (such as procdump, rundll32, etc). I can't verify at this time unfortunately.

Rebuilding Honeycred in C++

So as soon as I saw this project (HoneyCred), I immediately became fascinated with this project. I knew that I wanted to build something like that, so I did! I fired up Visual Studio and was off to the races. The (CreateProcessWithLogonW) [https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createprocesswithlogonw] wiki page is super detailed and is super helpful when referencing the Windows API. Right at the top of the page, we have all the arguments we need to supply to use the API (See code block below). All of the arguments fortunately have a detailed description that outlines what each thing is (if it's not self explanatory) which is super nice.

```
BOOL CreateProcessWithLogonW(
  LPCWSTR               lpUsername,
  LPCWSTR               lpDomain,
  LPCWSTR               lpPassword,
  DWORD                 dwLogonFlags,
  LPCWSTR               lpApplicationName,
  LPWSTR                lpCommandLine,
  DWORD                 dwCreationFlags,
  LPVOID                lpEnvironment,
  LPCWSTR               lpCurrentDirectory,
  LPSTARTUPINFOW        lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation
);
```

Argument Breakdown

Let's start by breaking it down argument by argument. We're going to rapidfire the first 3, since they're the easiest.

Data Type	Argument
LPCWSTR	lpUsername
LPCWSTR	lpDomain
LPCWSTR	lpPassword

All three of these are LPCWSTR, which are essentially just fancy strings. We can declare these with the following:

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
        LPCWSTR userName = L"user"; // The username that will be injected into
LSASS
    LPCWSTR userDomain = L"domain.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"pass"; // The User Password that will be injected
into LSASS
}
```

Now we're going to move onto a few of the more difficult ones, dwLogonFlags.

Data Type	Argument
DWORD64	dwLogonFlags

This seems incredibly daunting at first – what in the hell is a DWORD64? It's simply an 8-byte value* 0x00000000-0xFFFFFFFF. Microsoft gives us options here, so we're not completely left in the dark. We have two options here:

LOGON_WITH_PROFILE - 0x00000001 **LOGON_NETCREDENTIALS_ONLY** - 0x00000002

*Note: DWORDs can be different length depending on Operating System architecture

The Microsoft wiki states that LOGON_WITH_PROFILE actually validates credentials over the network, however (oddly enough), LOGON_NETCREDENTIALS_ONLY does **not**. This will be a prime candidate for us to use. Let's update the code:

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
        LPCWSTR userName = L"user"; // The username that will be injected into
LSASS
    LPCWSTR userDomain = L"domain.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"pass"; // The User Password that will be injected
into LSASS

        CreateProcessWithLogonW(userName, userDomain, userPassword, 0x00000002, ,
, , , , ,);
}
```

Next up is another tricky one – lpApplicationName. This one seems to be tricky and the name is a bit misleading. Microsoft actually wants the full path to the program. Not an arbitrary name. Fortunately that's also easy enough to supply.

Data Type	Argument
LPCWSTR	lpApplicationName

Let's update the code!

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
        LPCWSTR userName = L"user"; // The username that will be injected into
LSASS
    LPCWSTR userDomain = L"domain.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"pass"; // The User Password that will be injected
into LSASS
    LPCWSTR applicationName = L"C:\\Users\\administrator\\Desktop\\agent.exe";

        CreateProcessWithLogonW(userName, userDomain, userPassword, 0x00000002,
applicationName, , , , , ,);
}
```

Note: the Agent defined here is simply a 60-minute sleep timer in a while loop to keep a running proccess for the credentials to stay injected into LSASS

Next up is another fairly tricky one – lpCommandLine. This one, I couldn't actually find a ton of info on. I *believe* this is arguments for the command line parameters that needs to be ran.

Data Type	Argument
LPWSTR	lpCommandLine

We're going to assign this a NULL value which is a valid argument according to the Microsoft wiki. Let's update the code again:

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
        LPCWSTR userName = L"user"; // The username that will be injected into
LSASS
    LPCWSTR userDomain = L"domain.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"pass"; // The User Password that will be injected
into LSASS
    LPCWSTR applicationName = L"C:\\Users\\administrator\\Desktop\\agent.exe";

        CreateProcessWithLogonW(userName, userDomain, userPassword, 0x00000002,
applicationName, NULL , , , , ,);
}
```

Next up, we've got another DWORD - dwCreationFlags. This one actually requires us to look at the Microsoft Wiki for (Process Creation Flags)[https://docs.microsoft.com/en-us/windows/win32/procthread/process-creation-flags]. .

Data Type	Argument
DWORD64	dwCreationFlags

Most of these flags seem as if they're for specialized purposes like Debugging, both 0x01000000 and 0x04000000 seem like the normal flags for creating a process. 0x04000000 seems (to me) like the most normal setting for a process, so this is what we're going to run with! Let's add it to the code:

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
        LPCWSTR userName = L"user"; // The username that will be injected into
LSASS
    LPCWSTR userDomain = L"domain.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"pass"; // The User Password that will be injected
into LSASS
    LPCWSTR applicationName = L"C:\\Users\\administrator\\Desktop\\agent.exe";

        CreateProcessWithLogonW(userName, userDomain, userPassword, 0x00000002,
applicationName, NULL, 0x04000000, , , ,);
}
```

Next up is another quick one – lpEnvironment. This flag is used if we have a “Pointer to an Environment Block for a new process”. I believe this means if we already have a process allocated that we would like to use. If we do not have one (which we do not), we can set this to a null value.

Data Type	Argument
LPVOID	dwCreationFlags

Let’s add it to the code:

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
        LPCWSTR userName = L"user"; // The username that will be injected into
LSASS
    LPCWSTR userDomain = L"domain.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"pass"; // The User Password that will be injected
into LSASS
    LPCWSTR applicationName = L"C:\\Users\\administrator\\Desktop\\agent.exe";

        CreateProcessWithLogonW(userName, userDomain, userPassword, 0x00000002,
applicationName, NULL, 0x04000000, NULL, , ,);
}
```

We’ve got a really simple one up next, lpCurrentDirectory - This one is self explanatory, it just wants current directory for the process.

Data Type	Argument
LPCWSTR	lpCurrentDirectory

Let’s add it to the code:

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
        LPCWSTR userName = L"user"; // The username that will be injected into
LSASS
    LPCWSTR userDomain = L"domain.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"pass"; // The User Password that will be injected
into LSASS
    LPCWSTR applicationName = L"C:\\Users\\administrator\\Desktop\\agent.exe";
        LPCWSTR currentDirectory = L"C:\\";

        CreateProcessWithLogonW(userName, userDomain, userPassword, 0x00000002,
applicationName, NULL, 0x04000000, NULL, currentDirectory, ,);
}
```

We're going to bundle the last two together as a Two-In-One – lpStatupInfo and lpProcessInformation.

Data Type	Argument
LPSTARTUPINFOW	lpStartupInfo
LPPROCESS_INFORMATION	lpProcessInformation

This one is a bit bigger, we need to create a struct that contains the startupinfo for the new process and have a out-variable for the created process information.

You know? that wasn't actually that hard to explain. The code behind it is actually really simple.

```
#include <iostream>
#include <windows.h>
#include <processthreadsapi.h>

using namespace std;
int main()
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    si.dwFlags = 0x00000001;
    si.wShowWindow = 0;
    LPCWSTR userName = L"felix"; // The username that will be injected into LSASS
    LPCWSTR userDomain = L"contoso.com"; // The Logon Domain that will be injected
into LSASS
    LPCWSTR userPassword = L"xQc2021!!"; // The User Password that will be
injected into LSASS
    LPCWSTR applicationName = L"C:\\Users\\administrator\\Desktop\\agent.exe";
    LPCWSTR currentDirectory = L"C:\\";

    CreateProcessWithLogonW(userName, userDomain, userPassword, 0x00000002,
applicationName, NULL, 0x04000000, NULL, currentDirectory, &si, &pi);
}
```

We also had to add two news flags – they're basically used to hide the newly created process.

The Agent

The Agent is a fairly simple, but we're going to cover it quickly – Here's the code behind it:

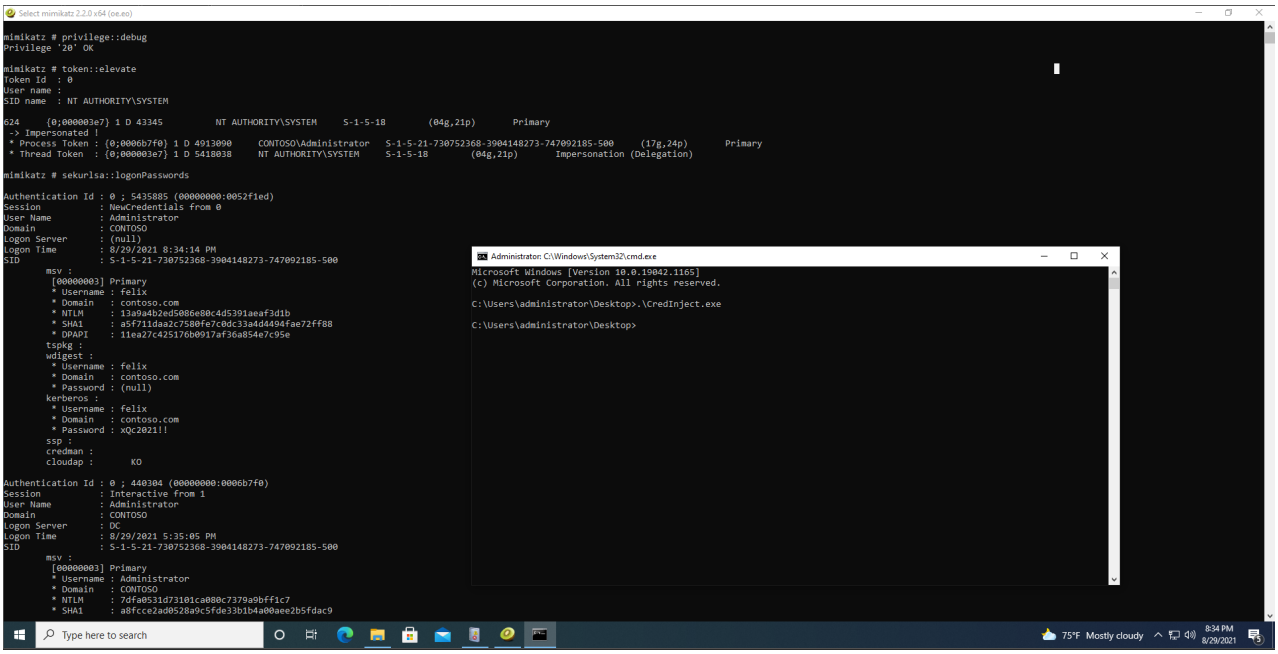
```
#include <stdio.h>
#include <Windows.h>
#include <ctime>

using namespace std;
int main() {
        while(true) {
                sleep(3600);
                return 0;
        }
}
```

Here, we have a sleep statement in a while loop that runs forever that sleeps for 1 hour. So now we're ready to compile and test!

Testing

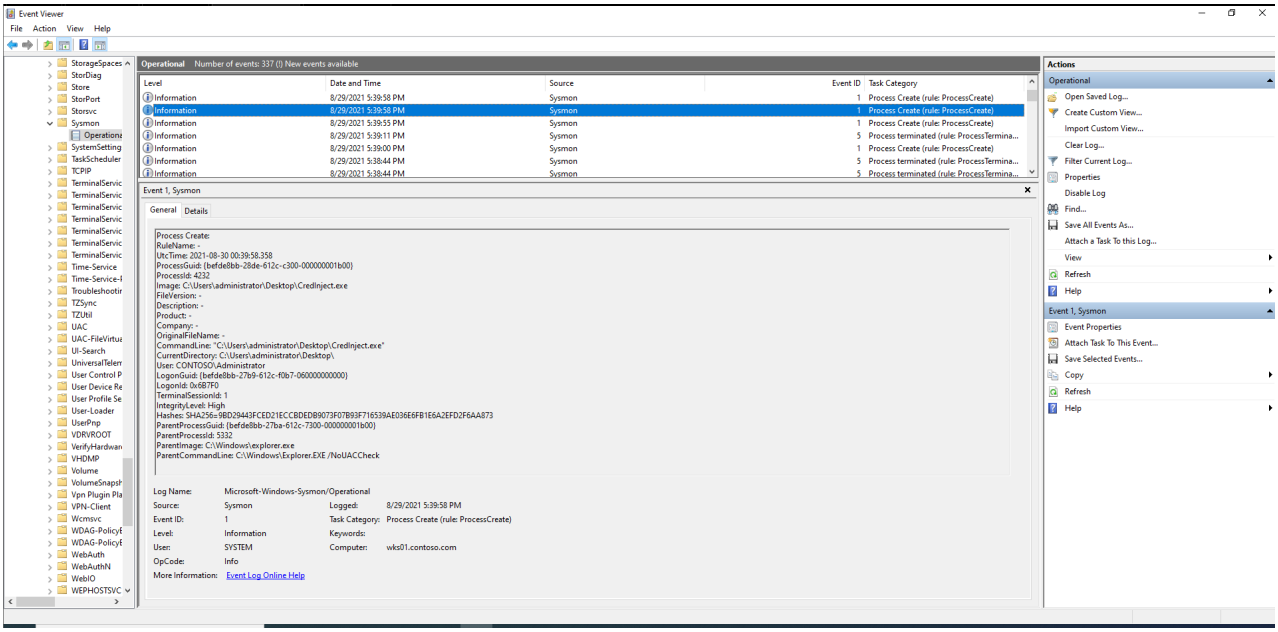
Now we're going to run the binary and check Mimikatz…



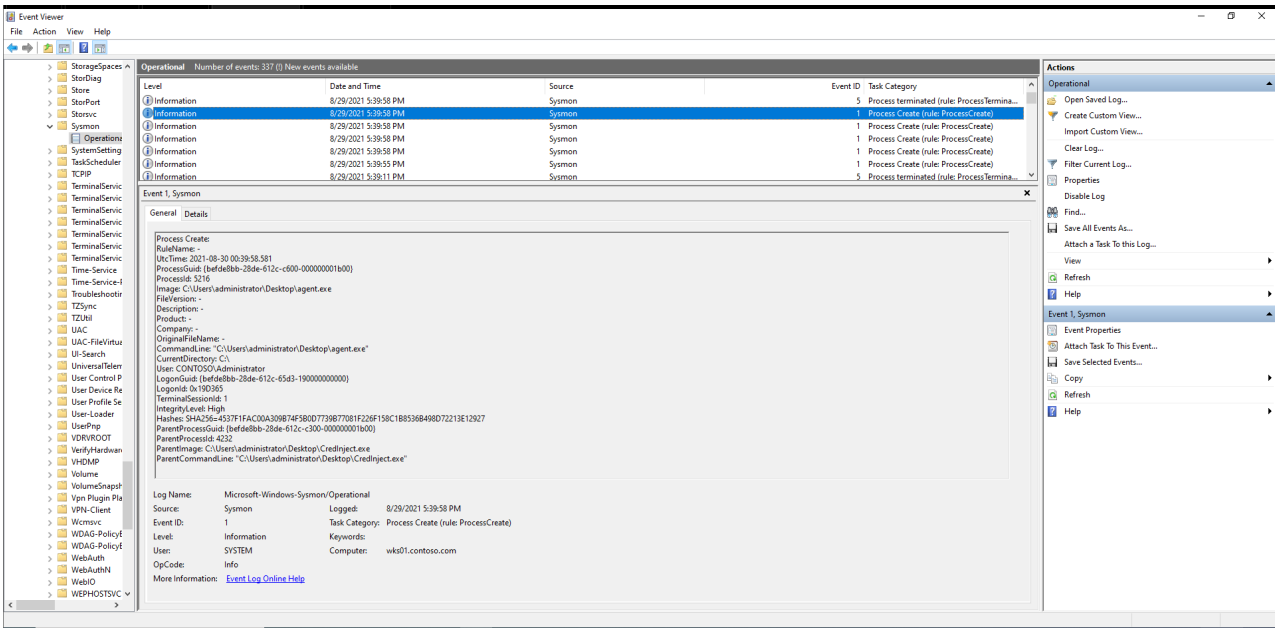
and ta-da! It worked! We've successfully been able to inject credentials into LSASS. Lets see what we can dig up on the Forensics front.

Sysmon Events Generated

Type 1 - Process Create This alert is generated when CredInject is ran



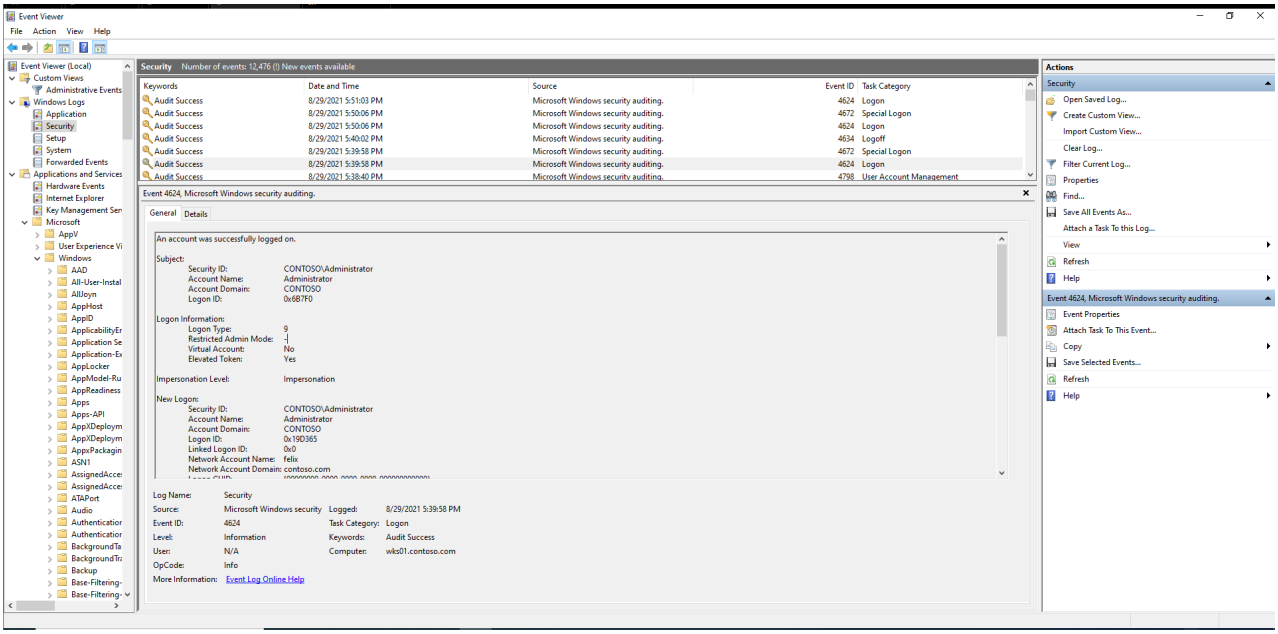
Type 1 - Proccess Create This alert is generated when the Agent is ran.



Interestingly enough, the User is flagged as the user who ran CredInject – not the injected credentials…

Built in Events

Interestingly enough a “Logon”, “Special Logon” and “Logoff” is also generated at the same time CredInject was ran.



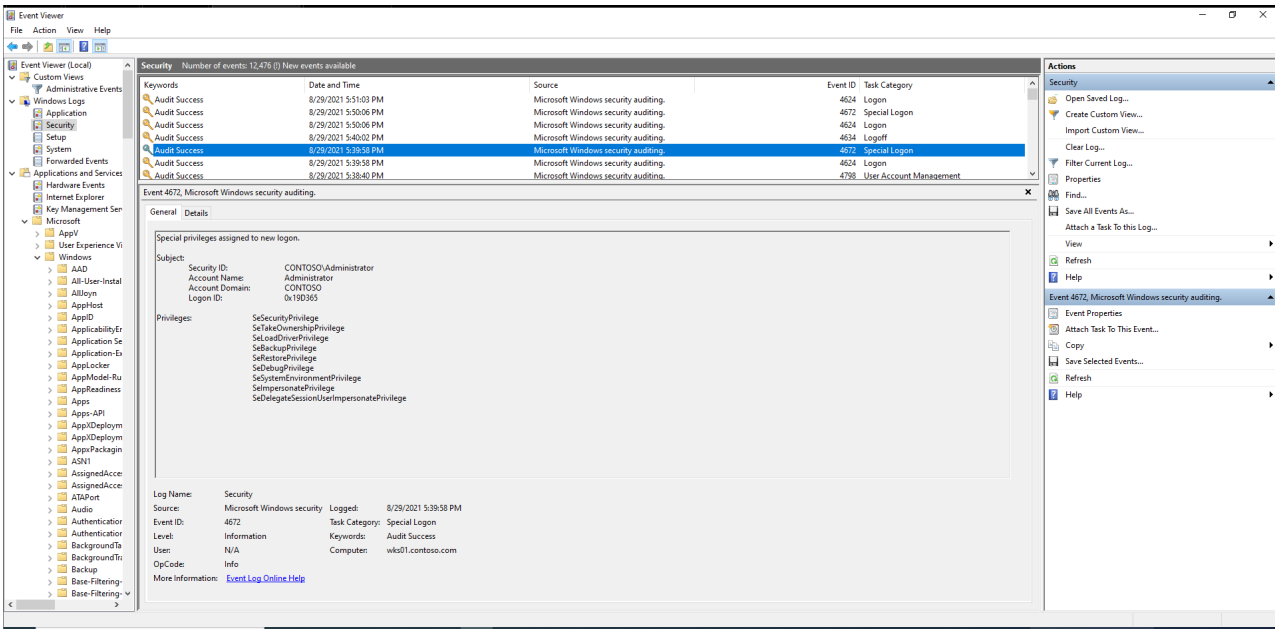
The interesting thing here is that the Logon type is a Type 9, which is generated from `runas /network` which actually makes a fair bit of sense – in our code, recall the dwLogonFlags was set to 0x00000002 or LOGON_NETCREDENTIALSONLY which (recall from the Microsoft Wiki) does not attempt to validate credentials.

The logon option. This parameter can be 0 (zero) or one of the following values.

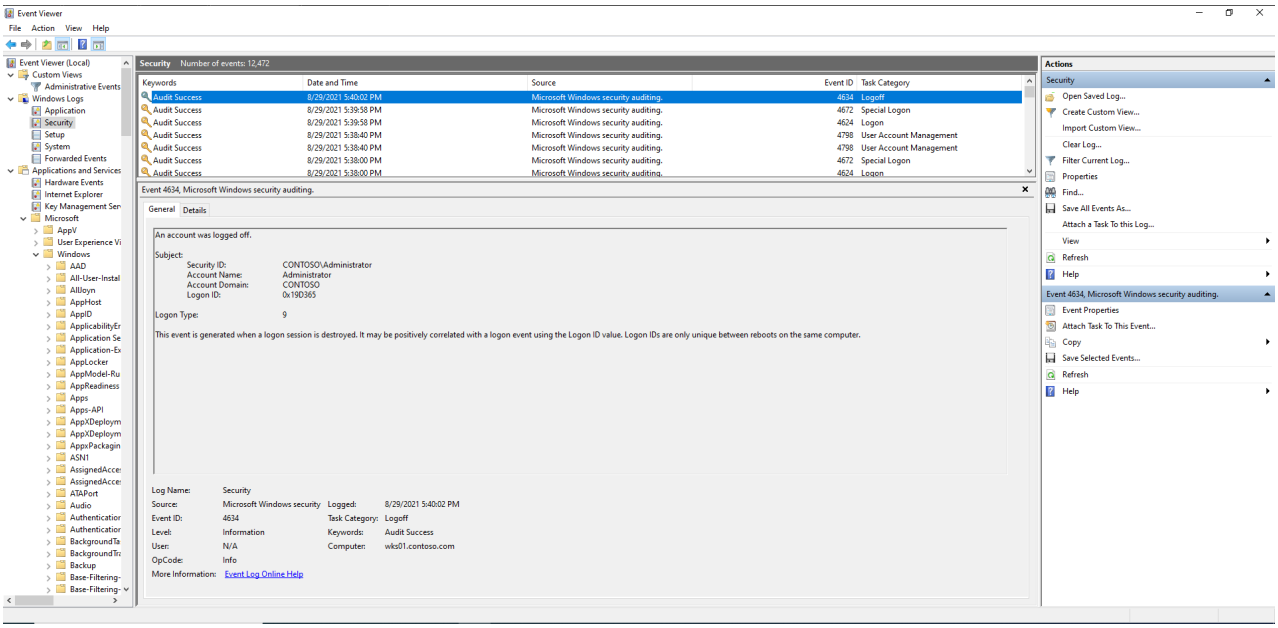
Value	Meaning
LOGON_WITH_PROFILE 0x00000001	Log on, then load the user profile in the **HKEY_USERS** registry key. The function returns after the profile is loaded. Loading the profile can be time-consuming, so it is best to use this value only if you must access the information in the **HKEY_CURRENT_USER** registry key. **Windows Server 2003:** The profile is unloaded after the new process is terminated, whether or not it has created child processes. **Windows XP:** The profile is unloaded after the new process and all child processes it has created are terminated.
LOGON_NETCREDENTIALS_ONLY 0x00000002	Log on, but use the specified credentials on the network only. The new process uses the same token as the caller, but the system creates a new logon session within LSA, and the process uses the specified credentials as the default credentials. This value can be used to create a process that uses a different set of credentials locally than it does remotely. This is useful in inter-domain scenarios where there is no trust relationship. The system does not validate the specified credentials. Therefore, the process can start, but it may not have access to network resources.

If you were interested in creating a YARA rule to flag on this, the Network Account Name and the Account Name do not match as well as a GUID of {00000000-0000-0000-0000-000000000000}. These are fairly unique (in my opinion).

And details of the Special Logon event:



And details of the logoff event:



In total, 3 events were generated, a 4624, 4672, and 4634.

It seems like we've reached a natural conclusion in this article. I wish that it would be possible to retrieve the injected credentials in LSASS – but that seems as though it's a limitation with the CreateProcessWithLogonW API endpoint, or potentially the LOGON_NETCREDENTIALSONLY flag.

If you liked this post, let me know! Your feedback is greatly appriciated!

Comments