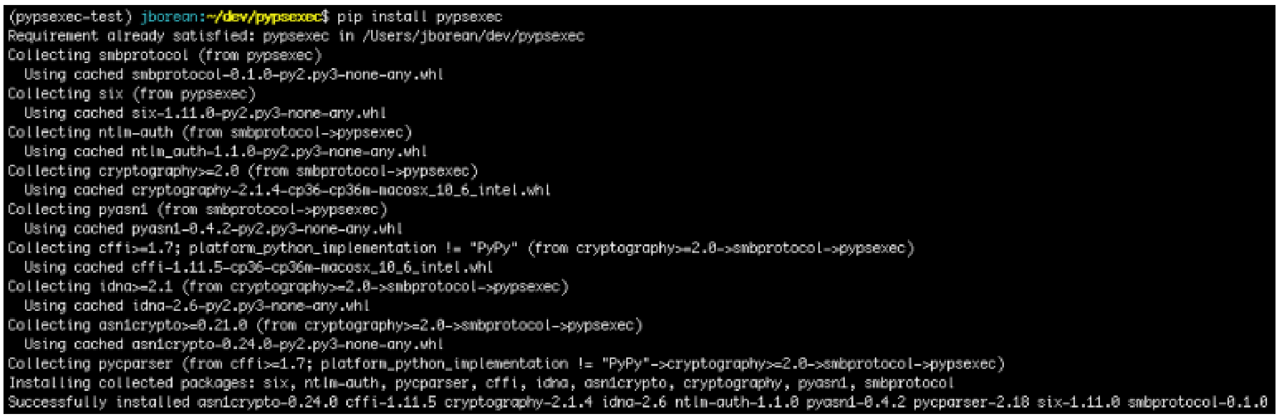
Introducing PsExec for Python

bloggingforlogging.com/2018/03/12/introducing-psexec-for-python

March 12, 2018



```
(pypsexec-test) jborean:~/dev/pypsexec$ pip install pypsexec
Requirement already satisfied: pypsexec in /Users/jborean/dev/pypsexec
Collecting smbprotocol (from pypsexec)
  Using cached smbprotocol-0.1.0-py2.py3-none-any.whl
Collecting six (from pypsexec)
  Using cached six-1.11.0-py2.py3-none-any.whl
Collecting ntlm-auth (from smbprotocol->pypsexec)
  Using cached ntlm_auth-1.1.0-py2.py3-none-any.whl
Collecting cryptography>=2.0 (from smbprotocol->pypsexec)
  Using cached cryptography-2.1.4-cp36-cp36m-macosx_10_6_intel.whl
Collecting pyasn1 (from smbprotocol->pypsexec)
  Using cached pyasn1-0.4.2-py2.py3-none-any.whl
Collecting cffi>=1.7; platform_python_implementation != "PyPy" (from cryptography>=2.0->smbprotocol->pypsexec)
  Using cached cffi-1.11.5-cp36-cp36m-macosx_10_6_intel.whl
Collecting idna>=2.1 (from cryptography>=2.0->smbprotocol->pypsexec)
  Using cached idna-2.6-py2.py3-none-any.whl
Collecting asn1crypto>=0.21.0 (from cryptography>=2.0->smbprotocol->pypsexec)
  Using cached asn1crypto-0.24.0-py2.py3-none-any.whl
Collecting pycparser (from cffi>=1.7; platform_python_implementation != "PyPy"->cryptography>=2.0->smbprotocol->pypsexec)
Installing collected packages: six, ntlm-auth, pycparser, cffi, idna, asn1crypto, cryptography, pyasn1, smbprotocol
Successfully installed asn1crypto-0.24.0 cffi-1.11.5 cryptography-2.1.4 idna-2.6 ntlm-auth-1.1.0 pyasn1-0.4.2 pycparser-2.18 six-1.11.0 smbprotocol-0.1.0
```

Over the past few months I've been trying to find a way that gives people more options around running commands on a Windows host remotely. Currently you have a few options available to you that enable this;

- Configure WinRM
- Bake in commands to the startup process, like a Windows answer file or AWS user data
- Use a tool like PsExec from another Windows host

The last point is where I am going to focus this blog post on, in particular I will talk about a new Python package called pypsexec.

Why PsExec

Before I go into the what, I need to explain why I am trying to run commands like PsExec and not just use something like WinRM. Ultimately PsExec has a few advantages over these protocols/tools like;

- No custom service or agent is required on the host
- It is only reliant on Server Message Block (SMB) which is setup and enabled on all Windows hosts
- Due to the minimal dependencies, it is really simple to allow PsExec to connect remotely compared to WinRM
- It is not platform dependent, can run on local hosts or hosts in AWS or some other cloud provider
- Allows you to easily escape WinRM hell or run as the SYSTEM account (more info around WinRM limitation can be found on this blog post)

In saying that, the PsExec model does have a few disadvantages such as;

- On Windows versions older than Server 2012 or Windows 8, there is no data encryption available
- Can only authenticated with an account that is a member of the local Administrators group
- May require some relaxing of Windows's UAC settings if not in a domain environment
- The overhead required to run a command means it is slow to start compared to WinRM

Ultimately I wanted to have an open source PsExec alternative that I can use in situations where WinRM is not available. This means I can

- Use it to run bootstrapping scripts or adhoc commands on Windows host without requiring WinRM to be setup
- Use this library to setup WinRM on newly installed hosts without the requirement of another Windows host
- Reduce the time it takes to copy files, WinRM is really slow with file transfers while SMB is designed for this process
- Satisfy my general curiosity around how PsExec works and get a better understanding of the SMB protocol

What is it

While PsExec is the most common name or term given to this process, it is actually a set of processes that is uses builtin protocols in Windows to work. The most common one is called PsExec and written by Mark Russinovich as part of the Sysinternals package. I'll go into more details on how the protocol works further down but ultimately it leverages SMB and RPC to start a service on a Windows host and use that to execute the desired process.

While PsExec is probably the most popular tool that works with this model, there are a few other tools out there which offer similar capabilities. These tools are;

There are some others out there but these are the only ones I know that work today. Unfortunately none of these tools really fit what I am looking for, the closest is Impacket but it has a few issues from my perspective. Ultimately I needed a way to run commands using the PsExec model that fit the following requirements;

- Not reliant on Windows, this rules out PsExec, PAExec, and RemCom as they use Win32 APIs to talk to the remote Windows host
- Works on the current supported versions of Windows, Impacket is ruled out as it uses RemCom which in turn doesn't support 64-bit architectures
- Can easily integrate into Ansible, Impacket is close but doesn't support Python 3 and would make this step difficult

In the end I decided that I needed to write some code (turned out to be a lot) in Python to fit my requirements and ultimately that ended up with 2 Python libraries, smbprotocol and pypsexec.

Host requirements

One of the reasons I looked into using the PsExec model is because it didn't require a lot of steps to setup on a Windows host. These are the only things that need to be done on the Windows host for this library to work;

- Enable incoming traffic through port 445 – `netsh advfirewall firewall set rule name="File and Printer Sharing (SMB-In)" dir=in new enable=Yes`
- The `ADMIN$` share is enabled – this is enabled by default
- Use a account that is a member of the local Administrators group
- The connection user to have a full elevated (administrative) token on a remote logon

The first 3 requirements are quite simple to set up and can either be done using sysprep images or through things like an Windows answer file during the setup. The last requirement is probably the biggest stumbling block when it comes to using this tool. To understand this restriction you first need to understand logon tokens and how they work from Windows Vista and onwards. A logon token contains the rights and groups of the account during the initial logon and since Windows Vista, the token only contains the rights of a limited user account regardless if they are an administrator. You can still run processes with the full administrative rights that is granted to the user but it must go through UAC to elevate the token (Right click -> Run as Administrator).

When running things remotely, there is no GUI to right click and say `Run as Administrator` or any UAC prompt when the process asked for admin rights so it will fail. We need admin rights to be able to open SCMR and manage the service that will run our process. Ultimately for this to work we need the Windows host to not filter a remote logon token and this can be done through multiple ways;

- In a domain environment, use any domain account that is a member of the local Administrators group
- Any local Administrator will work if LocalAccountTokenFilterPolicy is set to `1` which disables the filtering
- Use the builtin Administrator account (SID S-1-5-21-*-500), this account is typically disabled on desktop variants and this only works if AdminApprovalMode is not Enabled – this is not Enabled by default
- For local accounts, any local Administrator will work if EnableLUA is not Enabled – this is Enabled by default
- Disable UAC entirely (please don't do this)

As you can see, a domain environment makes this simple as Windows will automatically use the full elevated token on a remote logon which satisfies our requirements. If using a local account, either the initial builtin Administrator account (without AdminApprovalMode) being enabled or another local admin account with `LocalAccountTokenFilterPolicy` being set to 1 will work. Disabling the `EnableLUA` option will also work but it also affects local processes and runs them under the full token by default, effectively bypassing UAC in those scenarios.

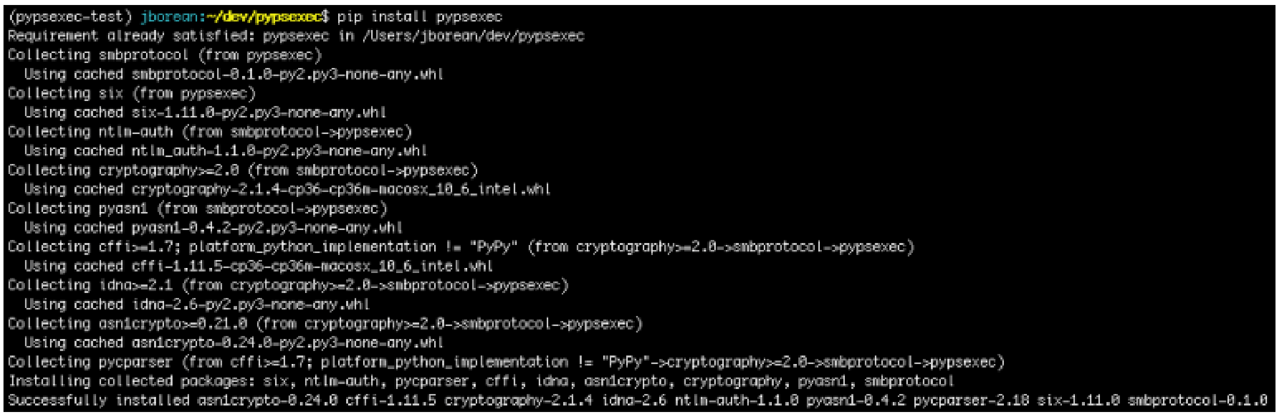
To disable the `LocalAccountTokenFilterPolicy`, you can run the following PowerShell script;

```
$reg_path = "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"
$reg_prop_name = "LocalAccountTokenFilterPolicy"

$reg_key = Get-Item -Path $reg_path
$reg_prop = $reg_key.GetValue($reg_prop_name)
if ($null -ne $reg_prop) {
    Remove-ItemProperty -Path $reg_path -Name $reg_prop_name
}

New-ItemProperty -Path $reg_path -Name $reg_prop_name -Value 1 -PropertyType DWord
```

I will have to warn you, this can have some security implications for your Windows host so make sure you are aware of the risks and don't follow the instructions blindly.

Using pypsexec

Now onto the fun part, getting it to run a command. The first step is to have a working Python install, you can run this on Python 2.6, 2.7, 3.4 and newer. To install the pypsexec library, simply run `pip install pypsexec` and it will be installed for you.



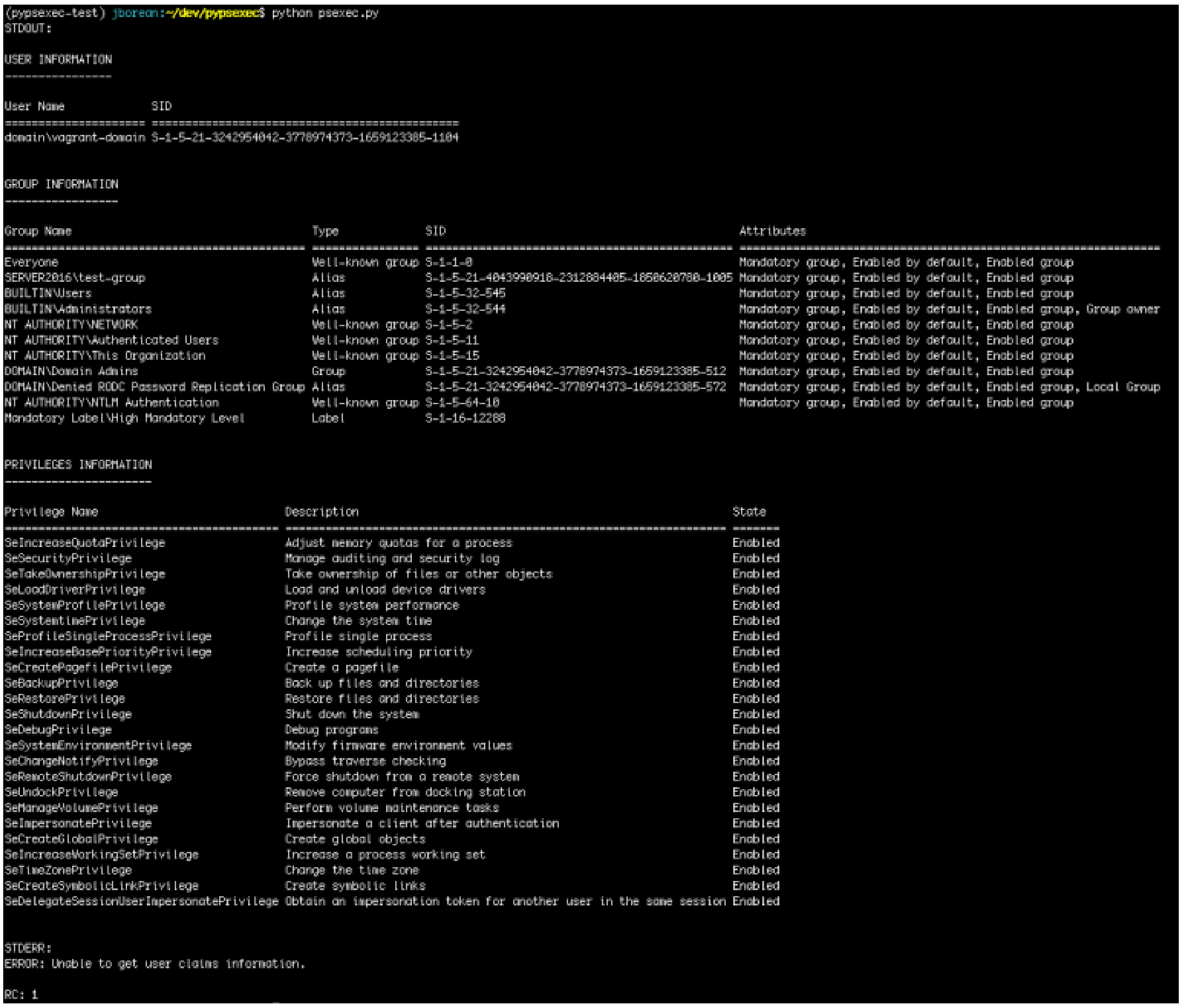
```
(pypsexec-test) jborean:~/dev/pypsexec$ pip install pypsexec
Requirement already satisfied: pypsexec in /Users/jborean/dev/pypsexec
Collecting smbprotocol (from pypsexec)
  Using cached smbprotocol-0.1.0-py2.py3-none-any.whl
Collecting six (from pypsexec)
  Using cached six-1.11.0-py2.py3-none-any.whl
Collecting ntlm-auth (from smbprotocol->pypsexec)
  Using cached ntlm_auth-1.1.0-py2.py3-none-any.whl
Collecting cryptography>=2.0 (from smbprotocol->pypsexec)
  Using cached cryptography-2.1.4-cp36-cp36m-macosx_10_6_intel.whl
Collecting pyasn1 (from smbprotocol->pypsexec)
  Using cached pyasn1-0.4.2-py2.py3-none-any.whl
Collecting cffi>=1.7; platform_python_implementation != "PyPy" (from cryptography>=2.0->smbprotocol->pypsexec)
  Using cached cffi-1.11.5-cp36-cp36m-macosx_10_6_intel.whl
Collecting idna>=2.1 (from cryptography>=2.0->smbprotocol->pypsexec)
  Using cached idna-2.6-py2.py3-none-any.whl
Collecting asn1crypto>=0.21.0 (from cryptography>=2.0->smbprotocol->pypsexec)
  Using cached asn1crypto-0.24.0-py2.py3-none-any.whl
Collecting pycparser (from cffi>=1.7; platform_python_implementation != "PyPy"->cryptography>=2.0->smbprotocol->pypsexec)
Installing collected packages: six, ntlm-auth, pycparser, cffi, idna, asn1crypto, cryptography, pyasn1, smbprotocol
Successfully installed asn1crypto-0.24.0 cffi-1.11.5 cryptography-2.1.4 idna-2.6 ntlm-auth-1.1.0 pyasn1-0.4.2 pycparser-2.18 six-1.11.0 smbprotocol-0.1.0
```

One simple command and it's done

Once installed we need to create a simple Python script to call the library and tell it what commands to run. This is a very basic template you can use which runs the command `whoami.exe /all` under a specific account.

```
from pypsexec.client import Client

server = "server2016.domain.local"
username = "vagrant-domain@DOMAIN.LOCAL"
password = "VagrantPass1"
executable = "whoami.exe"
arguments = "/all"

c = Client(server, username=username, password=password,
           encrypt=True)

c.connect()
try:
    c.create_service()
    result = c.run_executable(executable, arguments=arguments)
finally:
    c.remove_service()
    c.disconnect()

print("STDOUT:\n%s" % result[0].decode('utf-8') if result[0] else "")
print("STDERR:\n%s" % result[1].decode('utf-8') if result[1] else "")
print("RC: %d" % result[2])
```

From there I can simply run the Python script with just `python psexec.py` and here is the result from one of my servers



```
(pypsexec-test) jborean:~/dev/pypsexec$ python psexec.py
STDOUT:

USER INFORMATION
----------------

User Name             SID
===================== ==============================================
domain\vagrant-domain S-1-5-21-3242954042-3778974373-1659123385-1104


GROUP INFORMATION
-----------------

Group Name                                    Type             SID                                            Attributes
============================================= ================ ============================================== ===============================================================
Everyone                                      Well-known group S-1-1-0                                        Mandatory group, Enabled by default, Enabled group
SERVER2016\test-group                         Alias            S-1-5-21-4043990918-2312884405-1850620780-1005 Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                                 Alias            S-1-5-32-545                                   Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators                        Alias            S-1-5-32-544                                   Mandatory group, Enabled by default, Enabled group, Group owner
NT AUTHORITY\NETWORK                          Well-known group S-1-5-2                                        Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users              Well-known group S-1-5-11                                       Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization                Well-known group S-1-5-15                                       Mandatory group, Enabled by default, Enabled group
DOMAIN\Domain Admins                          Group            S-1-5-21-3242954042-3778974373-1659123385-512  Mandatory group, Enabled by default, Enabled group
DOMAIN\Denied RODC Password Replication Group Alias            S-1-5-21-3242954042-3778974373-1659123385-572  Mandatory group, Enabled by default, Enabled group, Local Group
NT AUTHORITY\NTLM Authentication              Well-known group S-1-5-64-10                                    Mandatory group, Enabled by default, Enabled group
Mandatory Label\High Mandatory Level          Label            S-1-16-12288


PRIVILEGES INFORMATION
----------------------

Privilege Name                            Description                                                        State
========================================= ================================================================== =======
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Enabled
SeSecurityPrivilege                       Manage auditing and security log                                   Enabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Enabled
SeLoadDriverPrivilege                     Load and unload device drivers                                     Enabled
SeSystemProfilePrivilege                  Profile system performance                                         Enabled
SeSystemtimePrivilege                     Change the system time                                             Enabled
SeProfileSingleProcessPrivilege           Profile single process                                             Enabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Enabled
SeCreatePagefilePrivilege                 Create a pagefile                                                  Enabled
SeBackupPrivilege                         Back up files and directories                                      Enabled
SeRestorePrivilege                        Restore files and directories                                      Enabled
SeShutdownPrivilege                       Shut down the system                                               Enabled
SeDebugPrivilege                          Debug programs                                                     Enabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Enabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Enabled
SeUndockPrivilege                         Remove computer from docking station                               Enabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Enabled
SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled
SeCreateGlobalPrivilege                   Create global objects                                              Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Enabled
SeTimeZonePrivilege                       Change the time zone                                               Enabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Enabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Enabled


STDERR:
ERROR: Unable to get user claims information.

RC: 1
```

Output is just like it is locally

This is a basic example of running a process but pypsexec gives you control over multiple options like;

- The number of processors it can run on
- Whether to run the command asynchronously and not wait for a response, it will continue to run in the background
- Whether to run it as an interactive process and what session to show that process on (the application will run on that session's desktop)
- Running as a different account than what was used in the connection process
- Running as the local SYSTEM account to get godlike privileges on the process
- Change the working directory
- Set the priority of the process
- Send bytes through the stdin pipe in case the remote process requires input
- Set a timeout on the remote process

All these options and more can be found on the pypsexec Github page.

One extra feature that is not included by default is the ability to use Kerberos to authenticate and run a process as that user. This requires some Kerberos bindings to be installed on the host as well as the Python Kerberos packages. The system Kerberos bindings are dependent on the distro that is being used and once installed, needs to be configured. The Python packages can be installed by running `pip install smbprotocol[kerberos]`. This means that in the SMB authentication process, it will automatically attempt to authenticate with Kerberos if possible and continue on from there.

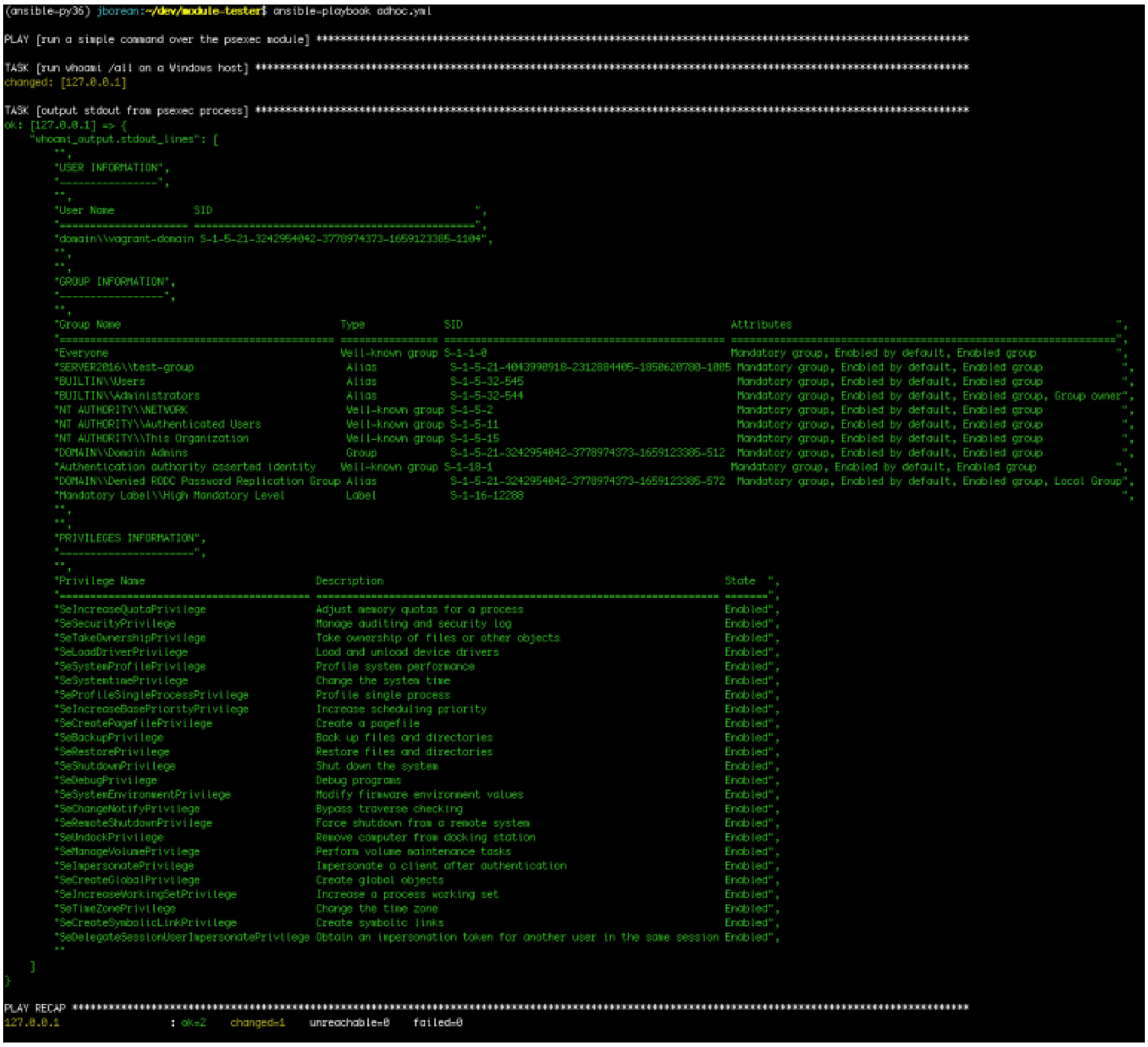
As I mentioned earlier, one of the reasons why I wanted to do this work was to add in a new way to run commands on a Windows host through Ansible. While, as of writing this blog post, it hasn't been merged into the Ansible repository I have created a PR that you can start using today and try it out. This PR can be found here and any tests or feedback is greatly appreciated. To use this module in your own Ansible setup you will have to;

- Install `pypsexec` and optional Kerberos dependencies as per usual on the host the module will run on
- Copy down the `psexec.py` file from that PR into a folder called `library` that is adjacent to your playbook or in a role directory

There are multiple examples in that PR on how you can use it but I will show you a simple example like the one above;

```
- name: run a simple command over the psexec module
  hosts: localhost
  gather_facts: no
  tasks:
  - name: run whoami /all on a Windows host
    psexec:
      hostname: server2016.domain.local
      connection_username: vagrant-domain@DOMAIN.LOCAL
      connection_password: VagrantPass1
      executable: whoami.exe
      arguments: /all
    register: whoami_output
    failed_when: whoami_output.rc not in [0, 1]

  - name: output stdout from psexec process
    debug:
      var: whoami_output.stdout_lines
```

Here is what it looks like when run;



```
(ansible-py36) jborean:~/dev/module-tester$ ansible-playbook adhoc.yml

PLAY [run a simple command over the psexec module] *****************************************************************************************

TASK [run whoami /all on a Windows host] ***************************************************************************************************
changed: [127.0.0.1]

TASK [output stdout from psexec process] ***************************************************************************************************
ok: [127.0.0.1] => {
    "whoami_output.stdout_lines": [
        "",
        "USER INFORMATION",
        "----------------",
        "",
        "User Name             SID                                           ",
        "===================== ==============================================",
        "domain\\vagrant-domain S-1-5-21-3242954042-3778974373-1659123385-1104",
        "",
        "",
        "GROUP INFORMATION",
        "-----------------",
        "",
        "Group Name                                    Type             SID                                           Attributes                                                      ",
        "============================================= ================ ============================================= ===============================================================",
        "Everyone                                      Well-known group S-1-1-0                                       Mandatory group, Enabled by default, Enabled group             ",
        "SERVER2016\\test-group                         Alias            S-1-5-21-4043990918-2312884405-1850620780-1005 Mandatory group, Enabled by default, Enabled group             ",
        "BUILTIN\\Users                                 Alias            S-1-5-32-545                                  Mandatory group, Enabled by default, Enabled group             ",
        "BUILTIN\\Administrators                        Alias            S-1-5-32-544                                  Mandatory group, Enabled by default, Enabled group, Group owner",
        "NT AUTHORITY\\NETWORK                          Well-known group S-1-5-2                                       Mandatory group, Enabled by default, Enabled group             ",
        "NT AUTHORITY\\Authenticated Users              Well-known group S-1-5-11                                      Mandatory group, Enabled by default, Enabled group             ",
        "NT AUTHORITY\\This Organization                Well-known group S-1-5-15                                      Mandatory group, Enabled by default, Enabled group             ",
        "DOMAIN\\Domain Admins                          Group            S-1-5-21-3242954042-3778974373-1659123385-512  Mandatory group, Enabled by default, Enabled group             ",
        "Authentication authority asserted identity    Well-known group S-1-18-1                                      Mandatory group, Enabled by default, Enabled group             ",
        "DOMAIN\\Denied RODC Password Replication Group Alias            S-1-5-21-3242954042-3778974373-1659123385-572  Mandatory group, Enabled by default, Enabled group, Local Group",
        "Mandatory Label\\High Mandatory Level          Label            S-1-16-12288                                                                                                  ",
        "",
        "",
        "PRIVILEGES INFORMATION",
        "----------------------",
        "",
        "Privilege Name                            Description                                                        State  ",
        "========================================= ================================================================== =======",
        "SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Enabled",
        "SeSecurityPrivilege                       Manage auditing and security log                                   Enabled",
        "SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Enabled",
        "SeLoadDriverPrivilege                     Load and unload device drivers                                     Enabled",
        "SeSystemProfilePrivilege                  Profile system performance                                         Enabled",
        "SeSystemtimePrivilege                     Change the system time                                             Enabled",
        "SeProfileSingleProcessPrivilege           Profile single process                                             Enabled",
        "SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Enabled",
        "SeCreatePagefilePrivilege                 Create a pagefile                                                  Enabled",
        "SeBackupPrivilege                         Back up files and directories                                      Enabled",
        "SeRestorePrivilege                        Restore files and directories                                      Enabled",
        "SeShutdownPrivilege                       Shut down the system                                               Enabled",
        "SeDebugPrivilege                          Debug programs                                                     Enabled",
        "SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Enabled",
        "SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled",
        "SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Enabled",
        "SeUndockPrivilege                         Remove computer from docking station                               Enabled",
        "SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Enabled",
        "SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled",
        "SeCreateGlobalPrivilege                   Create global objects                                              Enabled",
        "SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Enabled",
        "SeTimeZonePrivilege                       Change the time zone                                               Enabled",
        "SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Enabled",
        "SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Enabled",
        ""
    ]
}

PLAY RECAP *********************************************************************************************************************************
127.0.0.1                  : ok=2    changed=1    unreachable=0    failed=0
```

One of the benefits I spoke about of using this module is that you can provision a Windows host and use `pypsexec` to provision the WinRM listeners so Ansible can communicate with it normally. With this module you can easily do this by adding in the following task

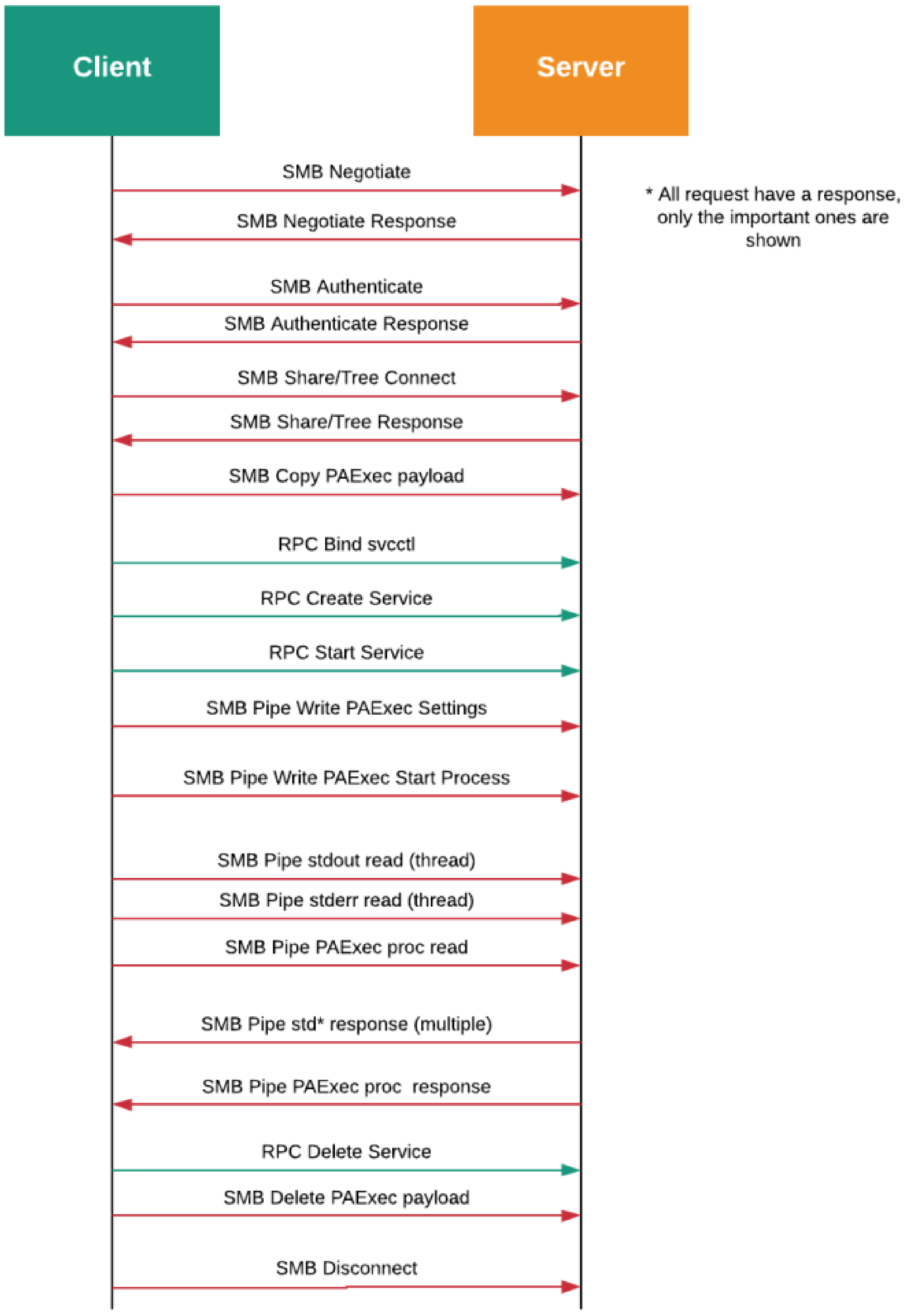
```
- name: Download and run ConfigureRemotingForAnsible.ps1 to setup WinRM
  psexec:
    hostname: windows-pc
    connection_username: Administrator
    connection_password: Password01
    executable: powershell.exe
    arguments: '-'
    stdin: |
      $ErrorActionPreference = "Stop"
      $sec_protocols = [Net.ServicePointManager]::SecurityProtocol -bor
[Net.SecurityProtocolType]::SystemDefault
      $sec_protocols = $sec_protocols -bor [Net.SecurityProtocolType]::Tls12
      [Net.ServicePointManager]::SecurityProtocol = $sec_protocols
      $url =
"https://github.com/ansible/ansible/raw/devel/examples/scripts/ConfigureRemotingFo
rAnsible.ps1"
      Invoke-Expression ((New-Object Net.WebClient).DownloadString($url))
      exit
```

This will run a PowerShell that we pass in through the `stdin` option that downloads Ansible's ConfigureRemotingForAnsible.ps1 and runs it on that Windows host. Once complete, your Ansible playbook can switch over to using the standard WinRM listener and continue as usual.

The next steps from here would be to look into turning this into a connection plugin within Ansible so that you aren't limited to running commands but you can do things like copy and fetch files to the host as well as use all the Ansible PowerShell modules. This isn't the be all to end all as the overhead required to run the process would make this quite slow and impractical to use over WinRM.

How it works

Now we know how to install and run this, it's time to get down into some protocol details and how it all stacks together. Here is a basic process flow of how this all fits together.


Client
Server
SMB Negotiate
SMB Negotiate Response
* All request have a response, only the important ones are shown
SMB Authenticate
SMB Authenticate Response
SMB Share/Tree Connect
SMB Share/Tree Response
SMB Copy PAExec payload
RPC Bind svcctl
RPC Create Service
RPC Start Service
SMB Pipe Write PAExec Settings
SMB Pipe Write PAExec Start Process
SMB Pipe stdout read (thread)
SMB Pipe stderr read (thread)
SMB Pipe PAExec proc read
SMB Pipe std* response (multiple)
SMB Pipe PAExec proc response
RPC Delete Service
SMB Delete PAExec payload
SMB Disconnect

Complex but gets the job done

As you can see, the majority of the network packets sent are done through SMB and in fact the RPC packets are encapsulated inside a specific SMB packet itself. The only part that is hard to describe in the process flow is the reading of the stdout and stderr pipes.

What pypsexec does is runs those read requests as part of a separate thread while it is blocked waiting for the main PAExec pipe to return the process exit info. There can be multiple responses from the server during this process and these threads will continue to run until the remote process is finished. With the basic flow out of the way, let's drill even deeper into each of the protocols that are used.

SMB

SMB standards for Server Message Block and depending on who you ask, is also known as CIFS or Samba. SMB is the actual protocol name while CIFS is an older dialect used by Microsoft historically. Samba is a suite of programs for Linux or Unix that is designed to inter operate with various Microsoft products like SMB or Active Directory. It is a protocol that is used for providing shared access to file, printer and pipes that operates on the OSI Application layer.

It can send data over numerous transports;

- Directly over TCP with port 445
- NetBIOS over TCP on ports 137 and 139
- Other legacy protocols like NBF, IPC/SPX

We will only focus on the direct TCP transport over port 445 as that is what is most commonly used today and provided the largest packet sizes. Some key terms in used within the SMB protocol are;

- `Connection`: Refers to the main connection to the server over TCP, this is the level in which the negotiate process occurs and there is typically one Connection per server
- `Session`: Refers to an authenticated session of a Connection, there is typically one Session per user per server
- `Tree`: Refers to a connected SMB share, like `ADMIN$` and is run over a Session
- `Open`: Refers to an open handle of a file, directory, printer, or pipe. This Open can govern what rights are allowed by a file operation based on the initial Open message and is run over a Tree
- `Dialect`: The version of SMB that is supported and controls what features are available and in some limited scenarios, the format of a message

Dialects

There have been numerous revisions and changes to the SMB protocol which ultimately results in a new dialect being created. The dialect controls what features are available to an SMB connection and can control what structure the messages ultimately takes. Here are some of the main SMB dialects that are still in use today

- `PC NETWORK PROGRAM 1.0`, `LANMAN1.0`, `Windows for Workgroups 3.1a`, `LM1.2X002`, `LANMAN2.1`, `NT LM 0.12`: All SMBv1 dialects and are not used at all by `smbprotocol`
- `2.0.0`: Introduced with Server 2008 and Windows Vista

- `2.1.0`: Server 2008 R2 and Windows 7
- `3.0.0`: Server 2012 and Windows 8
- `3.0.2`: Server 2012 R2 and Windows 8.1
- `3.1.1`: Server 2016 and Windows 10

Starting with the `2.0.0` dialect, the structure of the SMB messages have remained consistent and is supported by all currently supported Windows versions. This means the benefits of supported the older SMB 1.0 dialects is quite minimal and can open a user up to more attack vectors which is something we want to avoid.

One of the biggest changes that affects end users of this project is the addition of message encryption in the `3.x` dialects. This means that only Windows hosts based on Server 2012 or Windows 8 and newer support the encryption of messages sent to and from the clients. In today’s environment, this is definitely something we want to have and it is enabled by default on `pypsexec`.

Who knows what Microsoft will introduce in newer dialects but currently `smbprotocol` supports dialects `2.0.0` to `3.1.1` and most of the features in each dialect.

Messages

There are numerous types of messages in the SMBv2 protocol which I’ll briefly explain the major ones that are in use by `pypsexec`;

- SMB2 Packet Header: The header of all requests and responses, it contains the metadata around the request and response
- SMB2 NEGOTIATE: Used to negotiate the capabilities of the client and the server such as the dialect and encryption setup
- SMB2 SESSION_SETUP: Used to authenticate a user and setup an SMB Session
- SMB2 TREE_CONNECT: Used to connect to a Tree/Share on the remote host
- SMB2 CREATE: Used to create an open handle to a file, directory, printer, or pipe
- SMB2 READ: Used to read bytes from a file or pipe
- SMB2 WRITE: Used to write bytes to a file or pipe
- SMB2 IOCTL: Used to issue an implementation-specific FSCTL or IOCTL command across the network like an RPC bind
- SMB2 TRANSFORM_HEADER: Used as the header for an encrypted message and can contain 1 or multiple SMB2 Packet Headers

The `smbprotocol` library exposes a function that can be used to create and send each of these messages based on a few input parameters. This makes it quite simple to send an `SMB2 Read` request by only passing in the file handle and the offset to read from. In most circumstances, a single packet is sent to the server but the SMB protocol allows compounded packets to be sent in one request.

Authentication

One very important part of this process is to authenticate with a valid Windows account. This is most commonly done in SMB using the SPNEGO protocol to negotiate an authentication mechanism supports by both the client and the server. Currently `smbprotocol` can authenticate a local or domain account with either `NTLM` or `Kerberos` where `Kerberos` is the preferred option of the 2. The authentication process occurs straight after the negotiation response is received with the server's SPNEGO token. This token contains a list of authentication mechanisms that are supported which `smbprotocol` compares against its own setup. If the Kerberos requirements are installed and setup up correctly and the remote host indicates it supports Kerberos in the `SPNEGO` token, `smbprotocol` will first attempt to authenticate with Kerberos before falling back to NTLM.

Once authenticated, both the NTLM and Kerberos protocols supply a unique session key which is then used by SMB to derive both the signing and encryption keys. The process to compute these keys are based on the dialect that was negotiated where newer dialects have a more complicated process for greater security. Once these keys are computed and an SMB Session is created, any future messages using that Session will be encrypted using that authenticated user context.

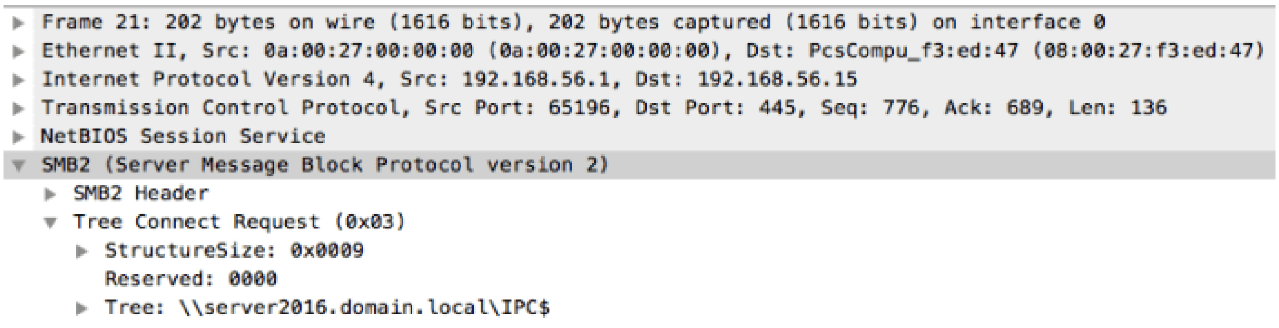
Encryption

If the `3.0.0` or newer dialect is negotiated then SMB allows messages to be encrypted to ensure confidentiality of the data sent to and from the server. Unlike other Microsoft protocols which typically uses the GSSAPI/SSPI `Wrap` and `Unwrap` functions based on an authenticated context, SMB relies on it's own process for encryption. Currently there are two different types of encryption that are supported in SMB;

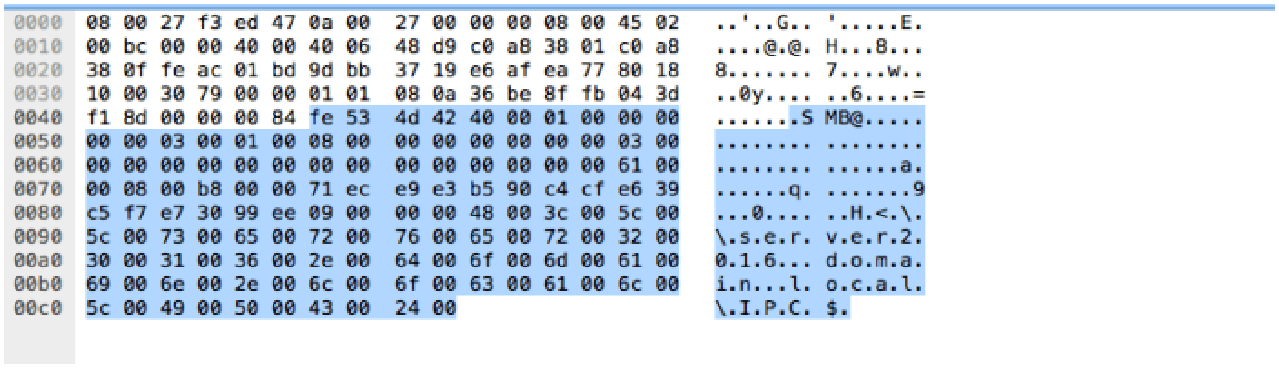
- AES 128-bit CCM
- AES 128-bit GCM (Dialect `3.1.1` only)

In Dialect `3.1.1`, the encryption cipher that is used is negotiated in the initial `SMB2 NEGOTIATE` message otherwise `AES 128-bit CCM` is used. Some servers require encryption on all shares and is set as a global setting otherwise it can be set as an individual share setting. This cannot be controlled by the client but rest assured, `smbprotocol` should support each scenario.

As an example, here is a Tree Connect message sent without encryption;

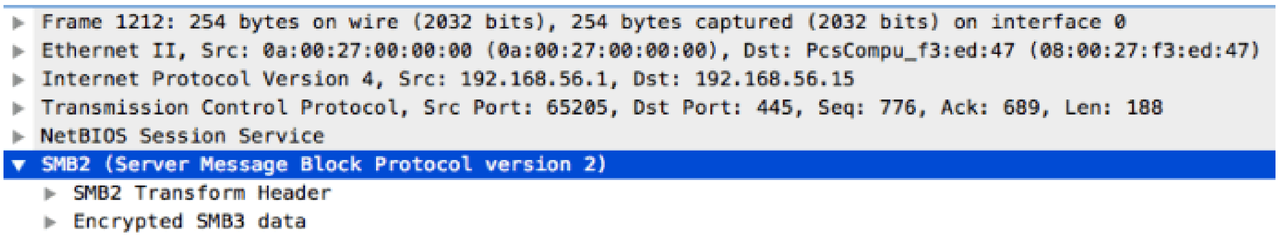


```
▶ Frame 21: 202 bytes on wire (1616 bits), 202 bytes captured (1616 bits) on interface 0
▶ Ethernet II, Src: 0a:00:27:00:00:00 (0a:00:27:00:00:00), Dst: PcsCompu_f3:ed:47 (08:00:27:f3:ed:47)
▶ Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.15
▶ Transmission Control Protocol, Src Port: 65196, Dst Port: 445, Seq: 776, Ack: 689, Len: 136
▶ NetBIOS Session Service
▼ SMB2 (Server Message Block Protocol version 2)
   ▶ SMB2 Header
   ▼ Tree Connect Request (0x03)
      ▶ StructureSize: 0x0009
        Reserved: 0000
      ▶ Tree: \\server2016.domain.local\IPC$
```

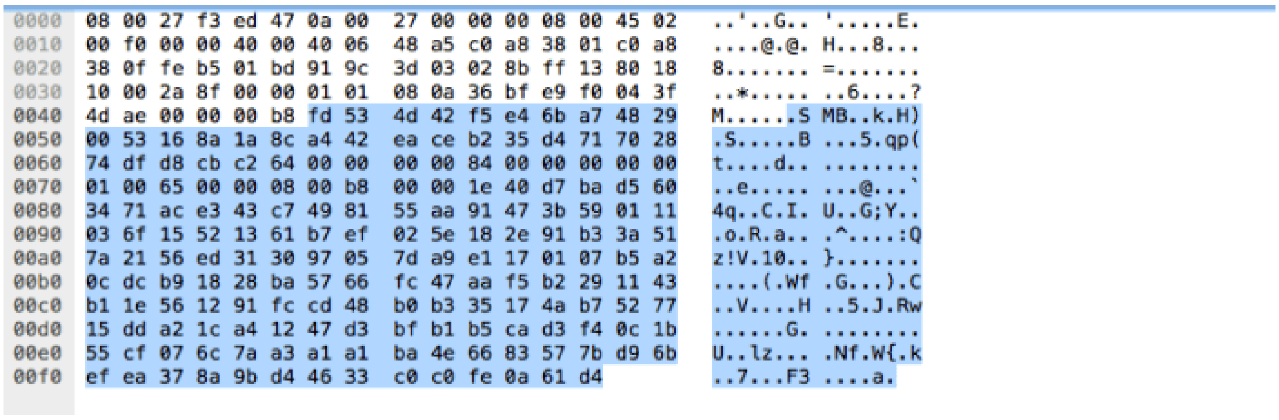


```
0000  08 00 27 f3 ed 47 0a 00  27 00 00 00 08 00 45 02   ..'..G.. '.....E.
0010  00 bc 00 00 40 00 40 06  48 d9 c0 a8 38 01 c0 a8   ....@.@. H...8...
0020  38 0f fe ac 01 bd 9d bb  37 19 e6 af ea 77 80 18   8....... 7....w..
0030  10 00 30 79 00 00 01 01  08 0a 36 be 8f fb 04 3d   ..0y.... ..6....=
0040  f1 8d 00 00 00 84 fe 53  4d 42 40 00 01 00 00 00   .......S MB@.....
0050  00 00 03 00 01 00 08 00  00 00 00 00 00 00 03 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 61 00   ........ ......a.
0070  00 08 00 b8 00 00 71 ec  e9 e3 b5 90 c4 cf e6 39   ......q. .......9
0080  c5 f7 e7 30 99 ee 09 00  00 00 48 00 3c 00 5c 00   ...0.... ..H.<.\.
0090  5c 00 73 00 65 00 72 00  76 00 65 00 72 00 32 00   \.s.e.r. v.e.r.2.
00a0  30 00 31 00 36 00 2e 00  64 00 6f 00 6d 00 61 00   0.1.6... d.o.m.a.
00b0  69 00 6e 00 2e 00 6c 00  6f 00 63 00 61 00 6c 00   i.n...l. o.c.a.l.
00c0  5c 00 49 00 50 00 43 00  24 00                     \.I.P.C. $.
```

Here is the same message sent with encrypted;



```
▶ Frame 1212: 254 bytes on wire (2032 bits), 254 bytes captured (2032 bits) on interface 0
▶ Ethernet II, Src: 0a:00:27:00:00:00 (0a:00:27:00:00:00), Dst: PcsCompu_f3:ed:47 (08:00:27:f3:ed:47)
▶ Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.15
▶ Transmission Control Protocol, Src Port: 65205, Dst Port: 445, Seq: 776, Ack: 689, Len: 188
▶ NetBIOS Session Service
▼ SMB2 (Server Message Block Protocol version 2)
   ▶ SMB2 Transform Header
   ▶ Encrypted SMB3 data
```



```
0000  08 00 27 f3 ed 47 0a 00  27 00 00 00 08 00 45 02   ..'..G.. '.....E.
0010  00 f0 00 00 40 00 40 06  48 a5 c0 a8 38 01 c0 a8   ....@.@. H...8...
0020  38 0f fe b5 01 bd 91 9c  3d 03 02 8b ff 13 80 18   8....... =.......
0030  10 00 2a 8f 00 00 01 01  08 0a 36 bf e9 f0 04 3f   ..*..... ..6....?
0040  4d ae 00 00 00 b8 fd 53  4d 42 f5 e4 6b a7 48 29   M......S MB..k.H)
0050  00 53 16 8a 1a 8c a4 42  ea ce b2 35 d4 71 70 28   .S.....B ...5.qp(
0060  74 df d8 cb c2 64 00 00  00 00 84 00 00 00 00 00   t....d.. ........
0070  01 00 65 00 00 08 00 b8  00 00 1e 40 d7 ba d5 60   ..e..... ...@...`
0080  34 71 ac e3 43 c7 49 81  55 aa 91 47 3b 59 01 11   4q..C.I. U..G;Y..
0090  03 6f 15 52 13 61 b7 ef  02 5e 18 2e 91 b3 3a 51   .o.R.a.. .^....:Q
00a0  7a 21 56 ed 31 30 97 05  7d a9 e1 17 01 07 b5 a2   z!V.10.. }.......
00b0  0c dc b9 18 28 ba 57 66  fc 47 aa f5 b2 29 11 43   ....(.Wf .G...).C
00c0  b1 1e 56 12 91 fc cd 48  b0 b3 35 17 4a b7 52 77   ..V....H ..5.J.Rw
00d0  15 dd a2 1c a4 12 47 d3  bf b1 b5 ca d3 f4 0c 1b   ......G. ........
00e0  55 cf 07 6c 7a a3 a1 a1  ba 4e 66 83 57 7b d9 6b   U..lz... .Nf.W{.k
00f0  ef ea 37 8a 9b d4 46 33  c0 c0 fe 0a 61 d4         ..7...F3 ....a.
```

In the message without encryption, I can easily see that I am connecting to the share `\\server2016.domain.local\IPC$` whereas the encryption example I cannot even see what type of SMB message is being sent. While hiding what share I am connecting to can be important, encryption becomes even more useful when reading and writing on files and pipes so that a nefarious lurker can't see the data.

RPC

RPC stands for Remote Procedure Call and is a way of running a procedure remotely but is coded like it would when running locally. Unfortunately the whole part of calling a procedure remotely like it would be done locally is lost when it comes to this process. This is because the usual RPC layer that handles this abstraction does not support the Windows specific functions. This means that the `pypsexec` library needs to implement that RPC layer when calling the functions that are required. For `pypsexec`, we use RPC to interact with the Windows Service Control Manager Remote (SCMR) API so that we can manage the Windows service that runs our remote payload. The RPC process is as follows;

- A new SMB Open is created on the `IPC$` tree for the pipe `svcctl`
- An SMB Write packet is sent to the opened pipe that contains the DCE/RPC Bind PDU structure
- The Bind Acknowledgement response is parsed to ensure the Bind didn't fail
- Any SCMR calls will then send an SMB IOCTL request that contains the method and parameters to invoke on the remote host
- Once all the SCMR tasks are complete, the SMB Open is closed which also closes the binding

This was a complicated protocol to understand and I only really just scratched the surface to get the Python library working with SCMR. I'm sure there are a lot of major details I am missing or misunderstood but so far it is working and I don't have a full need to move past it.

SCMR

SCMR stands for Service Control Manager Remote and is a protocol that is used to remotely manage Windows services. It can do things remotely like;

- Enumerate services
- Start/stop services
- Create/delete services
- Modify services
- Lots and lots more, see the MS-SCMR docs for a full list of functions available

It is run over RPC which in turn is run over SMB pipes and on a typical Windows setup, this is all abstracted with the local RPC implementation on that host. This implementation would marshal the data that is used in the function into a byte structure and send that through the network as well as parse and marshal the responses back to the client. As mentioned in the RPC section, this is unavailable on a non-Windows host and so we have to do all this work ourselves. The current `pypsexec` code only has to deal with 2 different types of variables, integers and strings. Integers are packed like normally in Python as a little-endian byte but strings are a bit more complex. String have a structure like

```
RCP string
{
    Int32 ReferenceID - A unique ID to set for the string, the uniqueness is not
really implemented in pypsexec and we just set it as 1 if required otherwise it is
a 0 byte value
    Int32 MaxCount - The numbers of chars in the Bytes field when returned by the
server, this is just set to ActualCount
    Int32 Offset - The offset of the Bytes value
    Int32 ActualCount - The number of chars (not Bytes) of the Bytes field when
encoded including the NULL terminator
    Bytes Bytes - The string that is encoded as a byte string, typically this is
UTF-16-LE encoded with a null terminator
    Bytes Padding - The Bytes field must align to a 4-byte boundary so this is
just some NULL bytes to pad the length
}
```

Now that the basic data marshaling is covered, `pypsexec` must add support for invoking the required functions in SCMR. This is done by creating a Request PDU as defined in the RCP/DCE 1.1 documentation and send that over as a `FSCTL_PIPE_TRANSCEIVE` SMB IOCTL Request. Each function has a particular operation number (opnum) that is set on the Request PDU and the data is just the marshaled bytes of the function's input parameters. The response contains at least the return code that identifies the result of the invocation and can also contain other return values based on the function that was called.

As an example, let's dive into the ROpenServiceW function and show what happens with the data being passed to and from the client. The function is defined in MS-SCMR as;

```
DWORD ROpenServiceW(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpServiceName,
  [in] DWORD dwDesiredAccess,
  [out] LPSC_RPC_HANDLE lpServiceHandle
 );
```

This means it takes in 3 input parameters `hSCManager`, `lpServiceName`, `dwDesiredAccess` and return 2 values; `lpServiceHandle` and a `DWORD/Int32` that indicates the function result. If we wanted to open a handle to the service called `Test Service` with the rights to query, start, and stop a service here is what it would look like;

The `hSCManager` was created as a unique handle as part of a previous call to `SCMR`, in this example we will just pretend it is 20 bytes of `0xFF`. The string `Test Service` does not need a unique/referent ID and the marshaled string structure would look like

```
MaxCount: 0D 00 00 00
Offset: 00 00 00 00
ActualCount: 0D 00 00 00
Bytes: 54 00 65 00 73 00 74 00 20 00 53 00 65 00 72 00 76 00 69 00 63 00 65 00 00
00
Padding: 00 00
```

The `MaxCount` and `ActualCount` is equal to `0x0D` which is 13 in decimal form while the string is encoded as a UTF-16-LE string with a null terminator. Because the UTF-16-LE encoded string is 26 bytes long, we need to pad it with 2 null bytes so it is aligned to the 4-byte boundary.

The `dwDesiredAccess` requires 3 flags that are set to get the query, start, and stop rights which are;

- `SERVICE_QUERY_STATUS`: 0x00000004
- `SERVICE_START`: 0x00000010
- `SERVICE_STOP`: 0x00000020

When combined this results in an integer of 52 and the packed bytes value for this is `34 00 00 00`. Putting this all together, the byte structure that is sent with the RPC `Request PDU` is;

```
FF FF FF FF
FF FF FF FF
FF FF FF FF
FF FF FF FF
FF FF FF FF
0D 00 00 00
00 00 00 00
0D 00 00 00
54 00 65 00
73 00 74 00
20 00 53 00
65 00 72 00
76 00 69 00
63 00 65 00
00 00 00 00
34 00 00 00
```

When sent with the RPC Request PDU, the opnum is set to `16` and that packed as a Int16 is `10 00`. The server would then receive the request, unpack the data and execute the function and finally return the result under an RPC Response PDU. This response would look like;

```
AA AA AA AA
AA AA AA AA
AA AA AA AA
AA AA AA AA
AA AA AA AA
00 00 00 00
```

The first 20 bytes is the handle for the service `Test Service`, in this case is 20 bytes of `0xAA` while the return code is 0 which means it was successful.

PAExec

While all this is done on the client side, we still need some executable to run as the Windows service and execute the requested process. Unfortunately while PsExec is free it is not licensed for distribution which means I can't legally use the PsExec executable to run as the service payload. In comes PAExec which is an open source alternative and can be distributed in other projects without a fee. PAExec is designed to be a drop in replacement to PsExec and offers the same features as well as some others that PsExec doesn't supply and for this project I am just using the service component. If you are interested in delving into the code for PAExec you can find it on its own Github page.

The process around PAExec is;

- Copy the PAExec payload to `C:\Windows\PAExec-<localhostname>-<localpid>.exe`
- Create a service that calls `C:\Windows\PAExec-<localhostname>-<localpid>.exe -service` to start the PAExec service
- This will create a Named Pipe on the remote host called `PAExec-<localhostname>-<localpid>.exe`
- Send the PAExec process settings (contains info such as the executable to run) to that Named Pipe
- Send the PAExec start message to the main Named Pipe
- PAExec will start the process and create three more Named Pipes, `PaExecOut<localhostname><localpid>`, `PaExecErr<localhostname><localpid>`, `PaExecIn<localhostname><localpid>` which is the `stdout`, `stderr` and `stdin` for the remote process
- The client will create a separate thread for both the stdout and stderr and continually read from that pipe
- The client will also create a read request for the main Named Pipe to get the process output
- This read on the main Named Pipe will wait until the remote process finishes, during this time the stdout and stderr thread will have stored each read response in a buffer
- Now the process has ended, the client has the stdout and stderr bytes as well as the return code from the remote process.

This can be repeated multiple times and once everything is run, the client can then cleanup the payload and service that remains on the Windows host. One of the biggest challenges I faced when creating this process was how to read from the stdout and stderr pipe as the main named pipe will not complete until those buffers are read and empty. To ensure we don't block any process in case there is no more data to be read from the pipes. What happens now is that the stdout and stderr pipes will keep on polling the remote pipe until it has been closed (the process is finished) as a separate thread. There is bound to be some more optimisations that can occur in this space but for a 1.0 product it works for me.

The biggest downfall of PAExec is that the initial settings end over the main Named Pipe is not encrypted but just encoded using a simple XOR pass. This means for Windows hosts that cannot use SMB encryption, any of these details can be viewed by anyone.

For the future

Currently the `pypsexec` and `smbprotocol` gives you the ability to run a process on a Windows host but it is not perfect. In the future I am looking to add in the following features;

- I still occasionally see some deadlocks when running a command, this needs to be solved but it is quite hard to debug and reproduce on demand
- An interactive shell for `pypsexec` that takes input from `stdin` and outputs the responses to `stdout` and `stderr`
- Simple script bundled with `pypsexec` to take in arguments and run the command like the `PsExec.exe` executable
- Add a higher level API to `smbprotocol` that does things like create/deleting files or reading/writing data to a file using simple functions
- Find ways to improve the speed of `smbprotocol`
- Move beyond a simple Ansible module and turn it into a connection plugin so it can be used to run all the existing Windows modules instead of just commands

Some of this stuff is easy to do but others are quite complex and depend on other things to be in place to be considered viable. If you feel up to the challenge or just come across a bug, feel free to submit an issue or a pull request on Github.