

Programming with Impacket - Working with SMB

blog.spookysec.net/Programming-with-Impacket.md

01 Aug 2023

Impacket by Fortra (formerly SecureAuth Corp) is probably best known for it's example scripts, they're a really awesome set of tools that allow you to do a **ton** of things. This can be as complex as Kerberoasting/ASREP-Roasting using `GetUserSPNs.py/GetNPUsers.py` or as simple as checking if a host is alive with `ping.py`.

Lots of people often forget that Impacket, in addition to all of the example scripts provided within is a full fledge library for interacting with and manipulating protocols typically used within a Windows environment. So, where is this blog post going? Well, as the title of the post suggests, we're going to be programming with the Impacket Library today!

This posts aims to get you some exposure to the Library, how to use the code in the example scripts to build your own and develop your own toolkit. Today we'll be working with and rebuilding some of the functionality that exists within `smbclient.py` as its the codebase I'm the most familiar with.

Installing Impacket

In case this is your first time using Impacket, we're going to cover the installation process real quick. It's important to note that I am using Kali 2023.1, this process may differ slightly from operating system to operating system and version to version.

To start, we need to clone the repository from Github. We can do this with the following command:

```
git clone https://github.com/fortra/impacket.git /opt/impacket
```

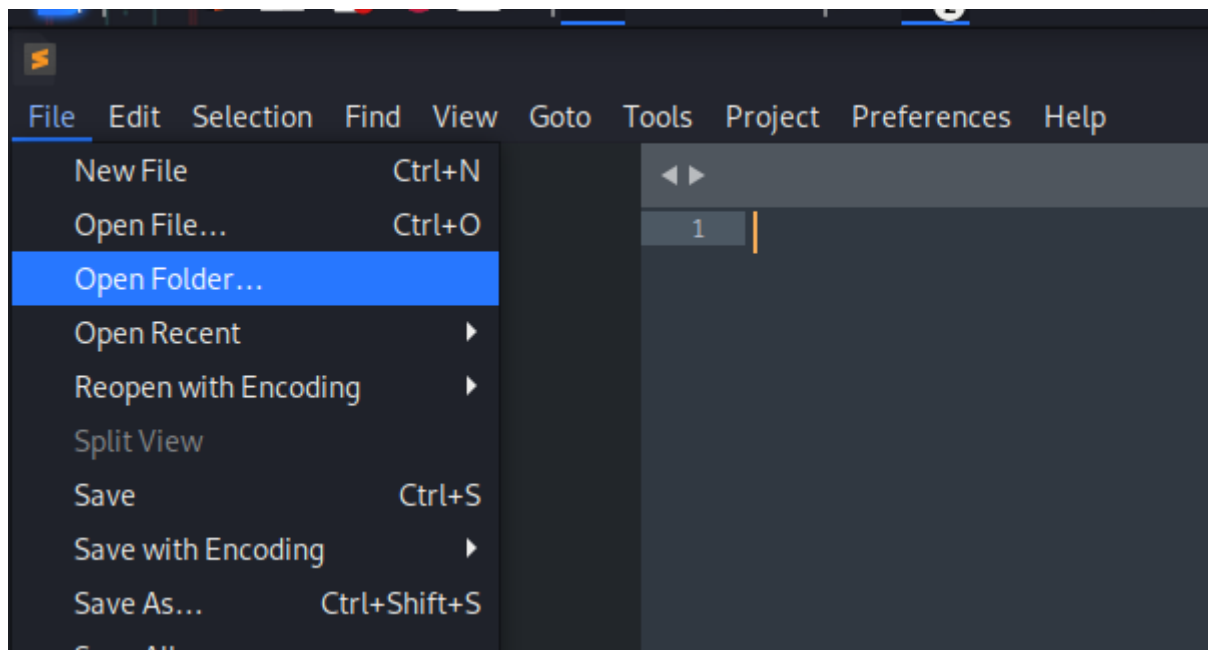
This command will download Impacket into the `/opt/impacket` folder, after it's complete, you'll want to `cd` into `/opt/impacket` and execute `python3 setup.py install`.

```
cd /opt/impacket/  
python3 setup.py install
```

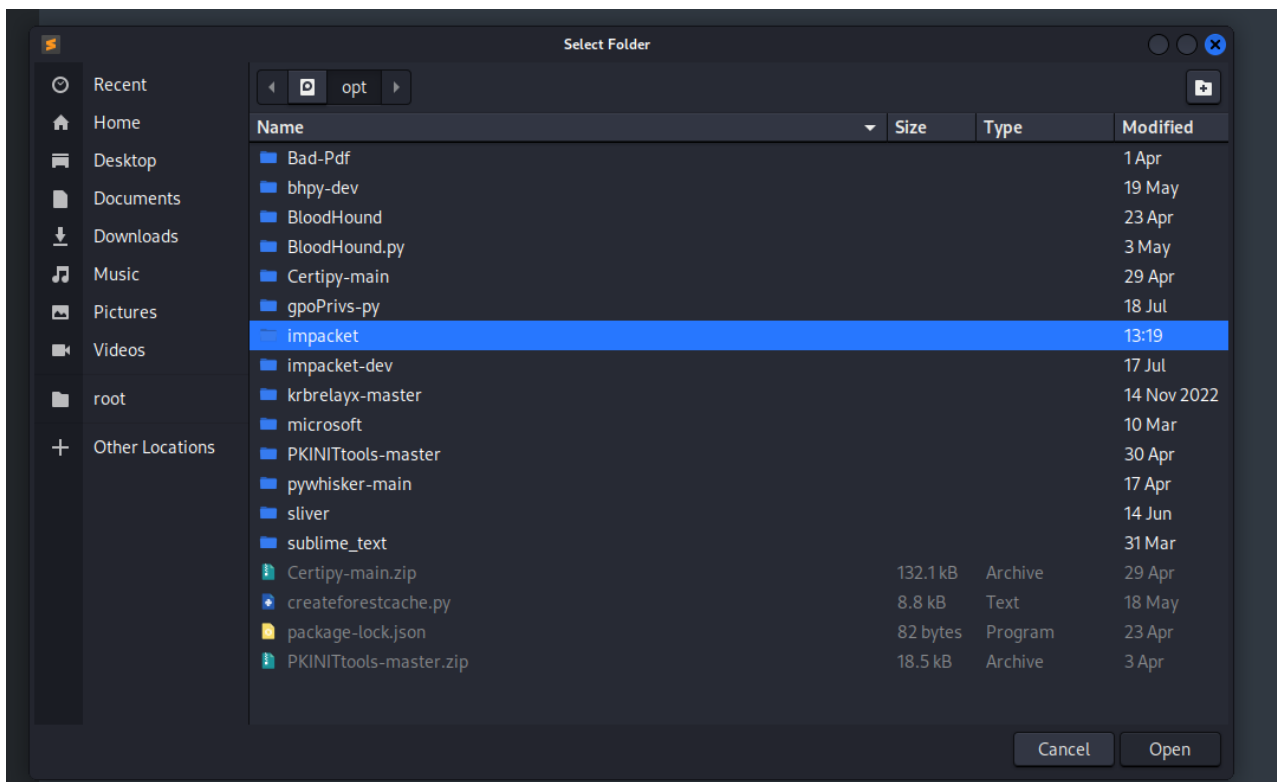
This will build and install Impacket and put it into your path. It's important to note that if you make any changes to the library itself (in `/opt/impacket`) you'll need to reinstall Impacket using the above command. After the install is finished we can move along!

Sublime and You

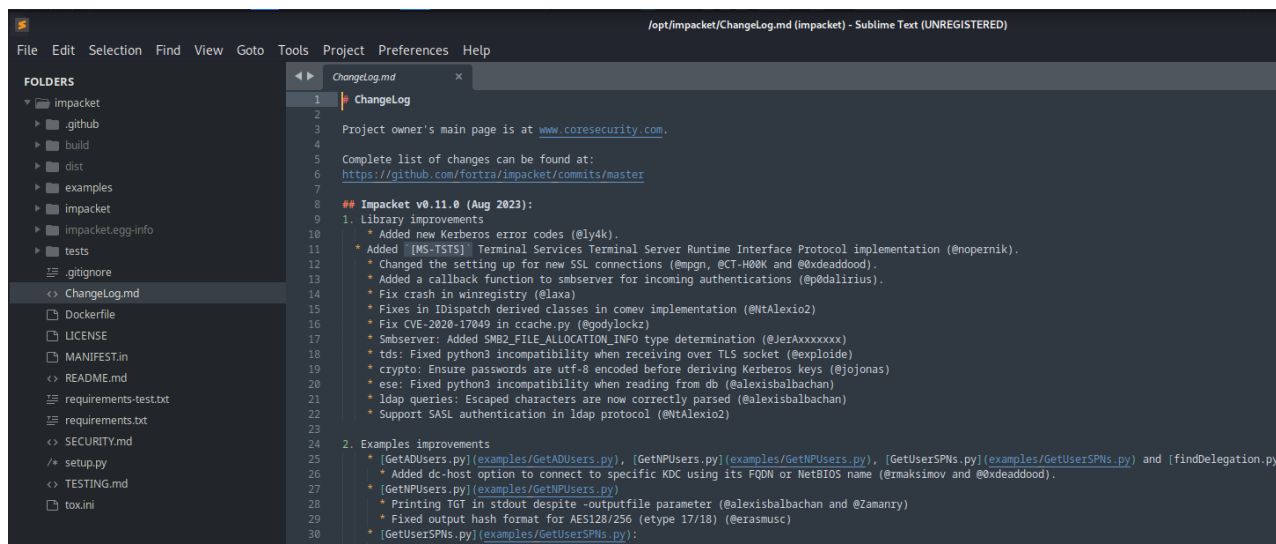
Sublime Text is my personal favorite text editor of choice, any IDE should have similar features as to what I'm going to demo. First, you'll want to select the file drop-down and select "Open Folder".



Navigate to /opt/, highlight “Impacket” and select “Open”.

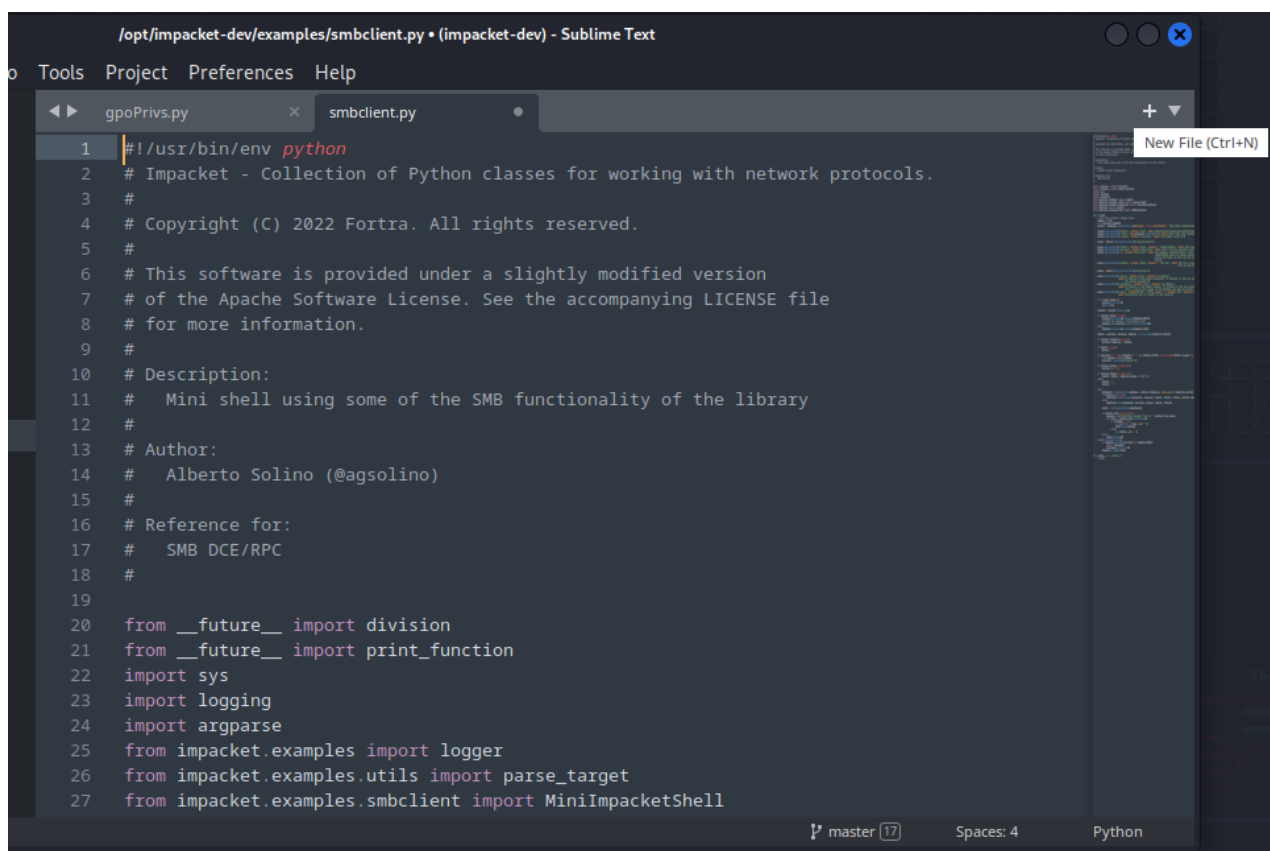


After, you should now see a list of files and folders on the left pane. As we navigate our way through the library, this will help keep us organized.



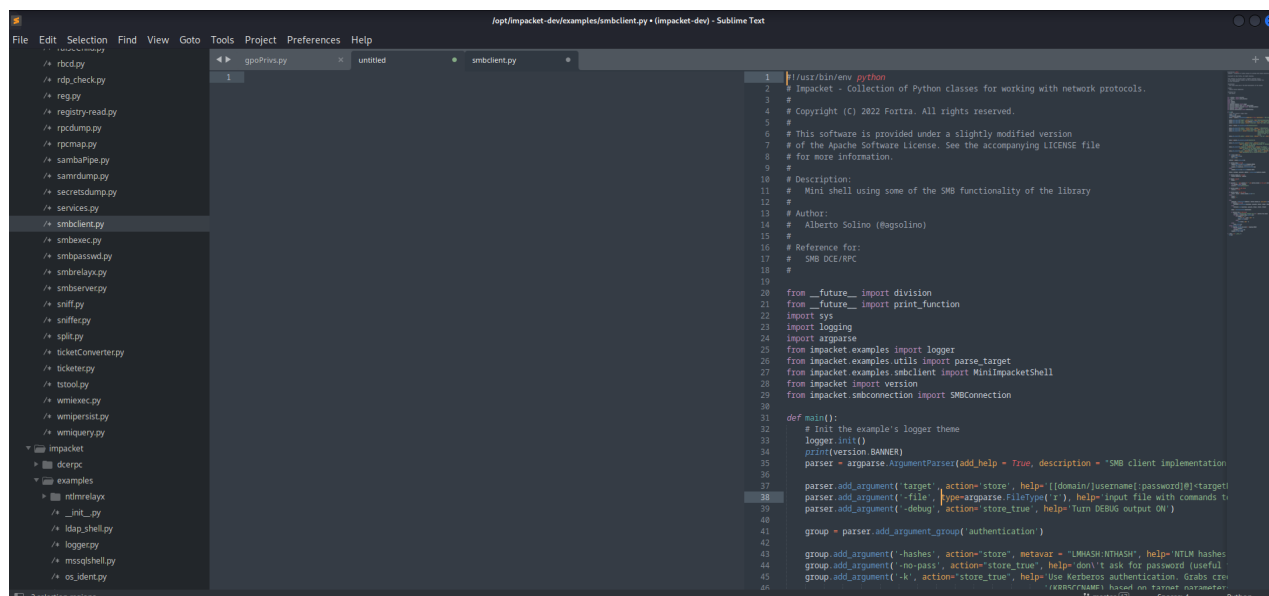
Examples Scripts and the Library

Now that we have our workspace established, opened and created, we're now going to open up the examples dropdown and select `smbclient.py` - you should now have the source code of `SMBClient.py` open on your screen. If you do not have a untitled blank document that you can use for writing code, now would be a good time to create one. This can be done by selecting `Ctrl+N`, or by double clicking any of the "white space" in the same bar that holds all of the open files, or by selecting the `+` icon.



Once the new file is created, you can move the untitled tab to the left or right by holding left click and dragging it to the left or right. You can also drag it out of Sublime to create a new window entirely. If you hold "Control" and click, you can open multiple windows at

once. This can be very helpful when you're writing code and reading the library to understand what it's doing.



So, here's generally where we should start structuring the program. Like most programs we're going to start with importing some libraries (Ex: ArgParse for ease of use) and grab some basic arguments, like Username, Password, IP Address, and Share. If you're unfamiliar with Argument Parsing, I recommend reading the [ArgParse documentation from Python.org](https://docs.python.org/3/library/argparse.html). We won't focus too much on it since this isn't the scope for this blog post.

To understand what should happen next, reading the SMBClient.py source code can help out majorly. Somehow we have to authenticate to this server, so let's trace where the Username and Password get supplied at in SMBClient.py.

```
37:     parser.add_argument('target', action='store',
help='[[domain/]username[:password]@]<targetName or address>')
```

It appears that the username, domain, password and target are stored in the *target* variable. Let's search for any references to Target.

```
76:     domain, username, password, address = parse_target(options.target)
```

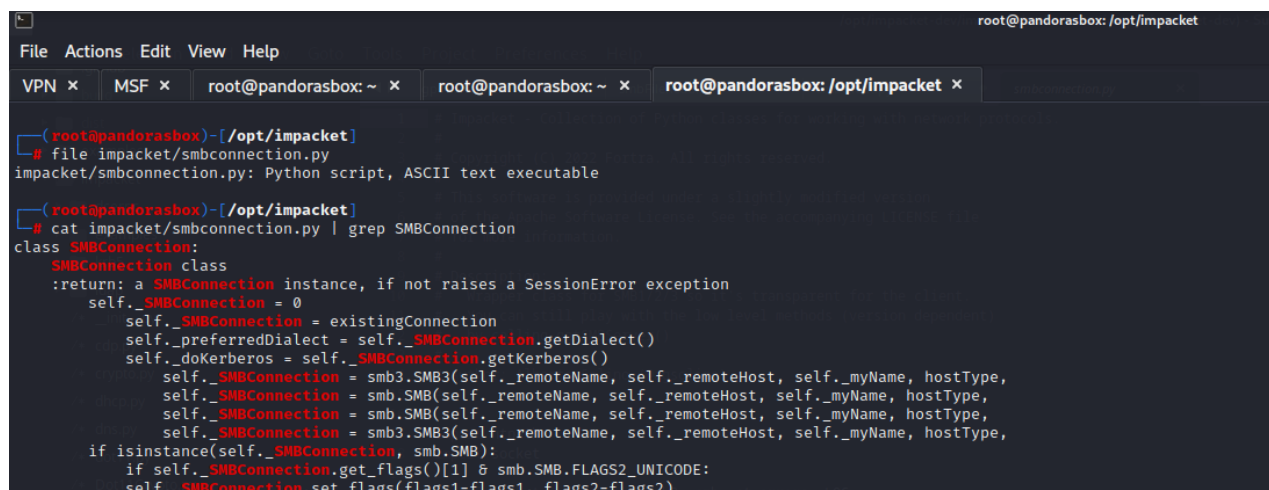
We can see that it's split into their own independent variables in line 76. Now we can search for any of the arguments and we should be able to find it.

```
97:     try:
98:         smbClient = SMBConnection(address, options.target_ip,
sess_port=int(options.port))
99:         if options.k is True:
100:             smbClient.kerberosLogin(username, password, domain, lmhash,
nthash, options.aesKey, options.dc_ip )
101:         else:
102:             smbClient.login(username, password, domain, lmhash, nthash)
103:
104:         shell = MiniImpacketShell(smbClient)
```

In line's 98-102 we actually have to connect to the host before we can authenticate to it. This is being handled by the SMBConnection function. Let's see if we can identify this in the source code of the Library to see what arguments can be supplied. By going back to the top of SMBClient.py, we can see all the imports. This will tell us where we need to look:

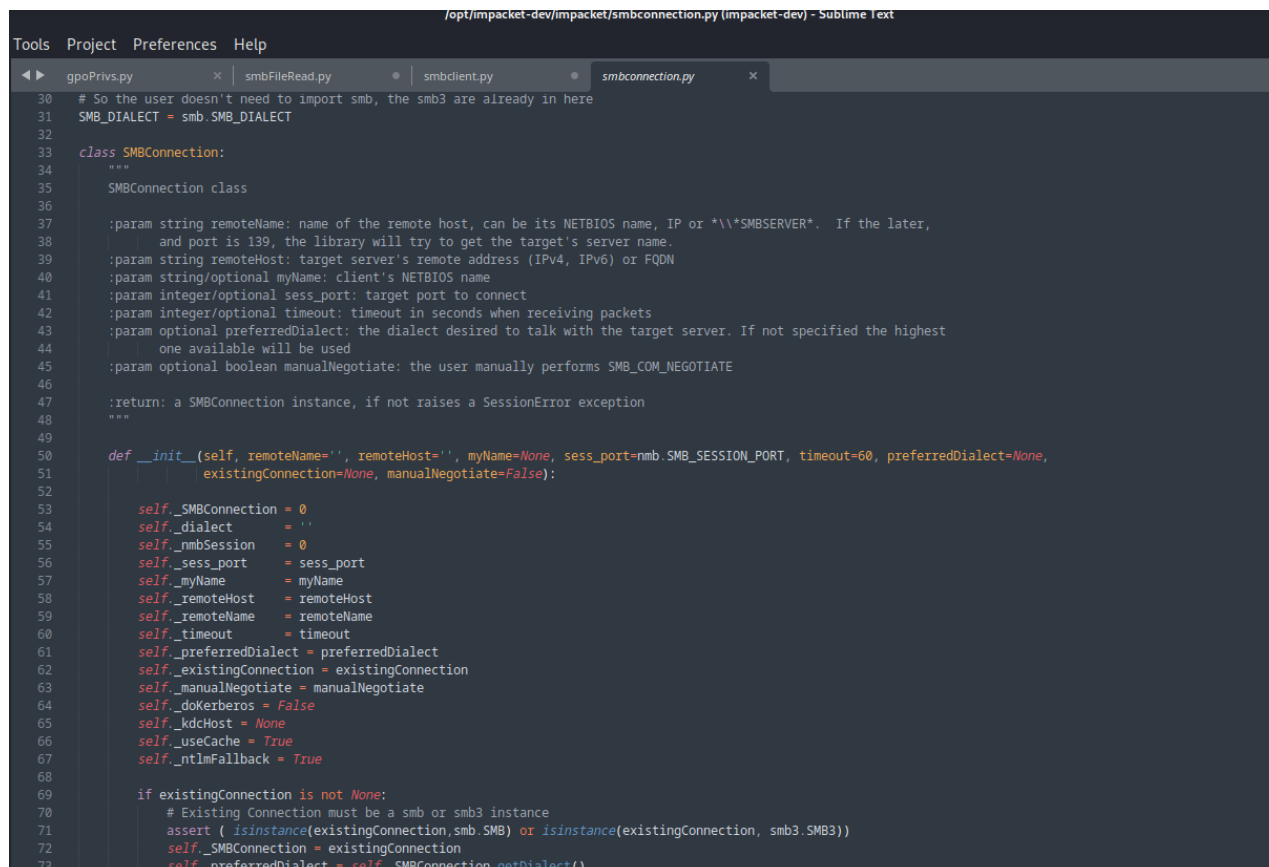
```
29: from impacket.smbconnection import SMBConnection
```

We should be able to find this in the Impacket/smbconnection.py file. We can validate that this file exists in either Sublime or the terminal (I chose the terminal for funsies)



```
root@pandorasbox: /opt/impacket
File Actions Edit View Help
VPN x MSF x root@pandorasbox: ~ x root@pandorasbox: ~ x root@pandorasbox: /opt/impacket x
root@pandorasbox)-[/opt/impacket]
# file impacket/smbconnection.py
impacket/smbconnection.py: Python script, ASCII text executable
root@pandorasbox)-[/opt/impacket]
# cat impacket/smbconnection.py | grep SMBConnection
class SMBConnection:
    SMBConnection class
    :return: a SMBConnection instance, if not raises a SessionError exception
        self._SMBConnection = 0
        self._SMBConnection = existingConnection
        self._preferredDialect = self._SMBConnection.getDialect()
        self._doKerberos = self._SMBConnection.getKerberos()
        self._SMBConnection = smb3.SMB3(self._remoteName, self._remoteHost, self._myName, hostType,
        self._SMBConnection = smb.SMB(self._remoteName, self._remoteHost, self._myName, hostType,
        self._SMBConnection = smb.SMB(self._remoteName, self._remoteHost, self._myName, hostType,
        self._SMBConnection = smb3.SMB3(self._remoteName, self._remoteHost, self._myName, hostType,
    if isinstance(self._SMBConnection, smb.SMB):
    if self._SMBConnection.get_flags()[1] & smb.SMB.FLAGS2_UNICODE:
        self._SMBConnection.set_flags(flags1=flags1, flags2=flags2)
```

We can see that there is a SMB Connection class exists. This looks like what we're searching for. If we open the file up in Sublime, we can see that sure enough, SMBConnection is the first class declared within the file.



```
/opt/impacket-dev/impacket/smbconnection.py (impacket-dev) - Sublime Text
Tools Project Preferences Help
gpoPrivs.py x smbFileRead.py x smbclient.py x smbconnection.py x
30 # So the user doesn't need to import smb, the smb3 are already in here
31 SMB_DIALECT = smb.SMB_DIALECT
32
33 class SMBConnection:
34     """
35     SMBConnection class
36
37     :param string remoteName: name of the remote host, can be its NETBIOS name, IP or *\\*SMBSERVER*. If the later,
38     and port is 139, the library will try to get the target's server name.
39     :param string remoteHost: target server's remote address (IPv4, IPv6) or FQDN
40     :param string/optional myName: client's NETBIOS name
41     :param integer/optional sess_port: target port to connect
42     :param integer/optional timeout: timeout in seconds when receiving packets
43     :param optional preferredDialect: the dialect desired to talk with the target server. If not specified the highest
44     one available will be used
45     :param optional boolean manualNegotiate: the user manually performs SMB_COM_NEGOTIATE
46
47     :return: a SMBConnection instance, if not raises a SessionError exception
48     """
49
50     def __init__(self, remoteName='', remoteHost='', myName=None, sess_port=smb.SMB_SESSION_PORT, timeout=60, preferredDialect=None,
51                 existingConnection=None, manualNegotiate=False):
52
53         self._SMBConnection = 0
54         self._dialect = ''
55         self._nmbSession = 0
56         self._sess_port = sess_port
57         self._myName = myName
58         self._remoteHost = remoteHost
59         self._remoteName = remoteName
60         self._timeout = timeout
61         self._preferredDialect = preferredDialect
62         self._existingConnection = existingConnection
63         self._manualNegotiate = manualNegotiate
64         self._doKerberos = False
65         self._kdcHost = None
66         self._useCache = True
67         self._ntlmFallback = True
68
69         if existingConnection is not None:
70             # Existing Connection must be a smb or smb3 instance
71             assert ( isinstance(existingConnection, smb.SMB) or isinstance(existingConnection, smb3.SMB3))
72             self._SMBConnection = existingConnection
73             self._preferredDialect = self._SMBConnection.getDialect()
```

We can see that it takes in several arguments, these being the following:

- remoteName
- remoteHost
- myName
- sess_port
- timeout
- preferredDialect
- existingConnection
- manualNegotiation

That's quite the list of arguments to supply to SMBConnection. Fortunately, not all of them are required and even have defaults, it's still nice to know that we have the ability to customize them if needed. There are even some interesting ones like existingConnection. Perhaps this is used in NTLMRelayx.

Try the skills you learned in the previous section (analyzing the source code in the example script (NTLMRelayx.py) to find the library file that contains the relevant code) to see if you can identify what file is depicted below.

```
25
26 PROTOCOL_ATTACK_CLASS = "SMBAttack"
27
28 class SMBAttack(ProtocolAttack):
29     """
30     This is the SMB default attack class.
31     It will either dump the hashes from the remote target, or open an interactive
32     shell if the -i option is specified.
33     """
34     PLUGIN_NAMES = ["SMB"]
35     def __init__(self, config, SMBClient, username):
36         ProtocolAttack.__init__(self, config, SMBClient, username)
37         if isinstance(SMBClient, smb.SMB) or isinstance(SMBClient, smb3.SMB3):
38             self.__SMBConnection = SMBConnection(existingConnection=SMBClient)
39         else:
40             self.__SMBConnection = SMBClient
41             self.__answerTMP = bytearray()
42             if self.config.interactive:
43                 #Launch locally listening interactive shell
44                 self.tcpshell = TcpShell()
45             else:
46                 self.tcpshell = None
47                 if self.config.exeFile is not None:
48                     self.installService = serviceinstall.ServiceInstall(SMBClient, self.config.exeFile)
49
50     def __answer(self, data):
51         self.__answerTMP += data
52
```

Back on track! Now that we know what argument the SMBConnection class expects, we can add our first real bit of code that uses Impacket. But first, don't forget to import the library! At a minimum, we must specify the remoteName and the remoteHost argument or else it may not connect properly. we can see some conditional if statements regarding the two very early on in the SMBConnection class.

```

# If port 445 and the name sent is *SMBSERVER we're setting the name to the IP. This is to help some old
# applications still believing
# *SMBSERVER will work against modern OSes. If port is NETBIOS_SESSION_PORT the user better know about i
# *SMBSERVER's limitations
if self._sess_port == nmb.SMB_SESSION_PORT and self._remoteName == '*SMBSERVER':
    self._remoteName = self._remoteHost
elif self._sess_port == nmb.NETBIOS_SESSION_PORT and self._remoteName == '*SMBSERVER':
    # If remote name is *SMBSERVER let's try to query its name.. if can't be guessed, continue and hope for the best
    nb = nmb.NetBIOS()
    try:
        res = nb.getnetbiosname(self._remoteHost)
    except:
        pass
    else:
        self._remoteName = res

if self._sess_port == nmb.NETBIOS_SESSION_PORT:
    negoData = '\x02NT LM 0.12\x00\x02SMB 2.002\x00'

```

Code Checkpoint - If you've made it this far in the post, that means it's time for a code checkpoint! Here's everything we have so far:

```
#!/usr/bin/python3
import argparse
from impacket.smbconnection import SMBConnection

args = argparse.ArgumentParser(description="A basic tool for reading files off of
SMB Shares", formatter_class=argparse.RawTextHelpFormatter,
usage=argparse.SUPPRESS)

args.add_argument('-u', '--username', dest='username', required=True,
default=None,help='Username to use for SMB connections.')
args.add_argument('-p', '--password', dest='password', required=True,
default=None,help='Password to use for SMB connections.')
args.add_argument('-d', '--domain', dest='domain', required=True,
default=None,help='Domain to use for SMB connections.')
args.add_argument('-i', '--ipaddress', dest='ip', required=True,
default=None,help='The IP Address or Hostname of the host to connect to.')
args.add_argument('-s', '--share', dest='share', required=True,
default=None,help='The SMB Share you would like to connect to.')
args.add_argument('-f', '--file', dest='file', required=True,
default=None,help='The file which you would like to read.')
args.add_argument('-v', '--verbose', dest='verbose', required=False,
default=False, action=argparse.BooleanOptionalAction, help='This option will
enable the program to be more or less verbose.')

args = args.parse_args()

if(args.verbose == True):
    print("[DEBUG] Domain: " + args.domain)
    print("[DEBUG] Username: " + args.username)
    print("[DEBUG] Password: " + args.password)
    print("[DEBUG] IP Address: " + args.ip)
    print("[DEBUG] Share: " + args.share)
    print("[DEBUG] File: " + args.file)
print("Connecting to " + args.ip)

try:
    smbConn = SMBConnection(remoteName=args.ip, remoteHost=args.ip)
except Exception as e:
    print("Failed to connect to " + args.ip + "\nReason: " + str(e))
    exit(1)
```

We're going to continue expanding on this, if Python isn't your strong suit and you're having difficulties, use this!

Back to Programming - Now we have to login to the server that we just established a connection to. Looking at the smbconnection.py file, on line 259 we have a login function.


```

258
259 def login(self, user, password, domain = '', lmhash = '', nthash = '', ntlmFallback = True):
260     """
261     logins into the target system
262
263     :param string user: username
264     :param string password: password for the user
265     :param string domain: domain where the account is valid for
266     :param string lmhash: LMHASH used to authenticate using hashes (password is not used)
267     :param string nthash: NTHASH used to authenticate using hashes (password is not used)
268     :param bool ntlmFallback: If True it will try NTLMv1 authentication if NTLMv2 fails. Only available for SMBv1
269
270     :return: None
271     :raise SessionError: if error
272     """
273     self._ntlmFallback = ntlmFallback
274     try:
275         if self.getDialect() == smb.SMB_DIALECT:
276             return self._SMBConnection.login(user, password, domain, lmhash, nthash, ntlmFallback)
277         else:
278             return self._SMBConnection.login(user, password, domain, lmhash, nthash)
279     except (smb.SessionError, smb3.SessionError) as e:
280         raise SessionError(e.get_error_code(), e.get_error_packet())
281

```

It takes in the following arguments:

- Username
- Password
- Domain
- LM/NTLM hash
- NTLM Fallback

We're primarily concerned with Username, Password and Domain here. Let's add the login function. We must call the variable we assigned to the SMBConnection class (smbConn) and specify the function (smbConn.login) and supply the arguments. I highly recommend you continue to handle errors as it's very crucial to know when you've made a mistake in either entering the command, the servers down and cannot authenticate for some reason, or if there's a fault in the Library. Reason can vary, it is helpful none the less!

```

(root@pandorasbox)~# python3 smbFileRead.py -u Administrator -p 'P@ssw0rd123!' -d contoso.com -i dc.contoso.com -s C$ -f /Windows/System32/Drivers/etc/hosts -v
[DEBUG] Username: Administrator
[DEBUG] Password: P@ssw0rd123!
[DEBUG] IP Address: dc.contoso.com
[DEBUG] Share: C$
[DEBUG] File: /Windows/System32/Drivers/etc/hosts
Connecting to dc.contoso.com
Authenticating to dc.contoso.com

(root@pandorasbox)~# python3 smbFileRead.py -u Administrator -p 'P@w0rd123!' -d contoso.com -i dc.contoso.com -s C$ -f /Windows/System32/Drivers/etc/hosts -v
[DEBUG] Username: Administrator
[DEBUG] Password: P@w0rd123!
[DEBUG] IP Address: dc.contoso.com
[DEBUG] Share: C$
[DEBUG] File: /Windows/System32/Drivers/etc/hosts
Connecting to dc.contoso.com
Authenticating to dc.contoso.com
Failed to authenticate to dc.contoso.com
Reason: SMB SessionError: STATUS_LOGON_FAILURE(The attempted logon is invalid. This is either due to a bad username or authentication information.)

(root@pandorasbox)~#

```

Next, we have to select the share we want to use. If we follow along in the SMBClient.py source code, we notice after authentication is handled/finished, it passes it over to MinilmpacketShell. We're not really down to re-use this as this is a programming exercise, so let's continue reading SMBConnection.py.

On line's 350-360, a function is created called "connectTree". If you've ever analyzed a SMB Packet Capture, this may stand out to you as a Tree Connect is what's used to connect to a share.

```

349
350     def connectTree(self, share):
351         if self.getDialect() == smb.SMB_DIALECT:
352             # If we already have a UNC we do nothing.
353             if ntpath.ismount(share) is False:
354                 # Else we build it
355                 share = ntpath.basename(share)
356                 share = '\\\\' + self.getRemoteHost() + '\\\\' + share
357         try:
358             return self._SMBConnection.connect_tree(share)
359         except (smb.SessionError, smb3.SessionError) as e:
360             raise SessionError(e.get_error_code(), e.get_error_packet())
361

```

We can see that this function takes in only one argument - Share. Let's try invoking this function as well.

```

(root@pandorasbox)-[~]
# python3 smbFileRead.py -u Administrator -p 'P@ssw0rd123!' -d contoso.com -i dc.contoso.com -s C$ -f /Windows/System32/Drivers/etc/hosts -v
[DEBUG] Username: Administrator
[DEBUG] Password: P@ssw0rd123!
[DEBUG] IP Address: dc.contoso.com
[DEBUG] Share: C$
[DEBUG] File: /Windows/System32/Drivers/etc/hosts
Connecting to dc.contoso.com
Authenticating to dc.contoso.com
Connecting to \\dc.contoso.com\C$

(root@pandorasbox)-[~]
# python3 smbFileRead.py -u Administrator -p 'P@ssw0rd123!' -d contoso.com -i dc.contoso.com -s Test$ -f /Windows/System32/Drivers/etc/hosts -v
[DEBUG] Username: Administrator
[DEBUG] Password: P@ssw0rd123!
[DEBUG] IP Address: dc.contoso.com
[DEBUG] Share: Test$
[DEBUG] File: /Windows/System32/Drivers/etc/hosts
Connecting to dc.contoso.com
Authenticating to dc.contoso.com
Connecting to \\dc.contoso.com\Test$
Failed to connect to \\dc.contoso.com\Test$
Reason: SMB SessionError: STATUS_BAD_NETWORK_NAME({Network Name Not Found} The specified share name cannot be found on the remote server.)

```

Code Checkpoint - Progress is being made! Which means it's time for another code checkpoint. This is continuing from the line we left off on in the last Code Checkpoint. You should be able to copy and paste the code below:

```

print("Authenticating to " + args.ip)
try:
    smbConn.login(user=args.username, password=args.password,
domain=args.domain)
except Exception as e:
    print("Failed to authenticate to " + args.ip + "\nReason: " + str(e))
    exit(1)

print("Connecting to \\\\" + args.ip + "\\" + args.share)
try:
    smbConn.connectTree(share=args.share)
except Exception as e:
    print("Failed to connect to \\\\" + args.ip + "\\" + args.share +
"\nReason: " + str(e))
    exit(1)

```

Alright, we're making good progress - Let's try to add in our file reading functionality now. Skipping down to line 754 in smbconnection.py, we can find the definition of the getFile function. This will be handy for reading files as it only takes in three arguments - a shareName, pathName and a "callback".

```

753
754 def getFile(self, shareName, pathName, callback, shareAccessMode = None):
755     """
756     downloads a file
757
758     :param string shareName: name for the share where the file is to be retrieved
759     :param string pathName: the path name to retrieve
760     :param callback callback: function called to write the contents read.
761     :param int shareAccessMode:
762
763     :return: None
764     :raise SessionError: if error
765     """
766     try:
767         if shareAccessMode is None:
768             # if share access mode is none, let's the underlying API deals with it
769             return self._SMBConnection.retr_file(shareName, pathName, callback)
770         else:
771             return self._SMBConnection.retr_file(shareName, pathName, callback, shareAccessMode=shareAccessMode)
772     except (smb.SessionError, smb3.SessionError) as e:
773         raise SessionError(e.get_error_code(), e.get_error_packet())
774

```

Callback isn't exactly self explanatory, so it may be a good idea to read into MiniShell and see what it's doing. If you're lazy - I can give you a quick answer. It wants a byte stream that it can write the data out to.

You have two methods here that you could use. You can write the data out to the disk, or you can write it to a memory stream. I recommend writing it out to disk because its slightly easier and I'm sure most people with an intermediate working level of python know how to do this already. If you have a more advanced knowledge set, or advanced use cases, you may want to write this into memory, do whatever is needed, then continue on with your day. Especially if you're not trying to save this data for a long time thing.

So, with all that said, we can create a file handle where we write our byte stream out to, invoke the getFile function from the SMBConnection class, specify the share and the file path, then lastly specify the callback to write the data onto disk. After that operation is finished, we can close the file handle, and open it with a read only flag, then print all the data to the user.

Code Checkpoint - This operation sounds fairly complex but is really straight forward. The addition of the file stream is what makes it a bit more complicated.

```

print("Downloading File...")
try:
    fh = open("temp.txt", "wb")
    smbConn.getFile(shareName=args.share, pathName=args.file,
callback=fh.write)
except Exception as e:
    print("Failed to download file: " + str(e))
    fh.close()
    exit(1)
fh.close()
print("Reading file:\n\n")
fh = open("temp.txt", "r")
print(fh.read())

```

Wrapping up & Testing

So, for my example, I want to read the /etc/hosts file on Windows. For those of you who don't know, it's stored in C:\Windows\System32\Drivers\etc\hosts. This should be accessible from the C\$ share. So, if we run the command with these arguments, we should get the contents of the file:

```
python3 smbFileRead.py -u Administrator -p 'SuperSecretAdminPassword1337!' -d contoso.com -i dc.contoso.com -s C$ -f /Windows/System32/Drivers/etc/hosts
```

If we wrote the code correctly, we should receive the contents of /etc/hosts back. And giving it our final test...

```
(root@pandorasbox)~# python3 smbFileRead.py -u Administrator -p 'P@ssw0rd123!' -d contoso.com -i dc.contoso.com -s C$ -f /Windows/System32/Drivers/etc/hosts
Connecting to dc.contoso.com
Authenticating to dc.contoso.com
Connecting to \\dc.contoso.com\C$
Downloading File...
Reading file:

# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#     102.54.94.97     rhino.acme.com          # source server
#     38.25.63.10     x.acme.com             # x client host

# localhost name resolution is handled within DNS itself.
#
#     127.0.0.1       localhost
#     ::1             localhost
# Flag{31337-Impacket-Programmer!}

(root@pandorasbox)~#
```

We received the contents back! Looking at the file on the Windows side, we can see we have the same results:

```
hosts - Notepad
File Edit Format View Help
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#     102.54.94.97     rhino.acme.com          # source server
#     38.25.63.10     x.acme.com             # x client host

# localhost name resolution is handled within DNS itself.
#
#     127.0.0.1       localhost
#     ::1             localhost
# Flag{31337-Impacket-Programmer!}
```

So that's about it. That's how you can build your own file reading utility with Impacket. I know it's a bit different from my standard blog posts, I don't really do much programming, but I thought this might make for an interesting topic. You can find the source code [here](#).

Feel free to hit me up on X if you're having any issues, I'd be happy to try to help out!

As always, thanks for reading. I hope to see you all soon :)

Comments
