标题 NBSS协议之NTLM认证 第一篇

0 在命令提示符中输入net use \\192.168.1.13 /user:lyc lyc后按回车
1 检查输出是否是"命令成功完成."(中文操作系统),如果不是的话请跳转至最后两步
2 打开wireshark 1.2.6 在Filter:后输入smb
/*
这个版本如果找不到我也可以发给你们
此后在Windows任务管理器中要一直有wireshark.exe这个映像名称
*/
3 选择菜单栏的Capture,点击Stop
4 再次选择菜单栏里的File,点击Save
5 在弹出的窗口中看到Packet Range,勾选Displayed,默认为Captured
6 在文件名后的输入框中输入一个文件名123
7 保存
8 选择菜单栏里的File,点击Open
9 在弹出的窗口中选中123.pcap,点击打开
/*
这是默认的保存类型
现在你应该看到了8个数据报文(如果不是的话不要紧,我可以发给你们)
我们只关注2个数据报文
*/
10 在Filter:后依次输入ntlmssp.messagetype==0x00000002和ntlmssp.messagetype==0x00000003
/*
将分别显示第4个和第5个(MS-NLMP的解释是Challenge(挑战)消息和Authenticate(认证)消息)
*/
11 在文件资源管理器里,按住Shift,同时按右键,在此处打开命令提示符窗口并输入python后按回车
12 输入以下内容

```
import pyshark
a=pyshark.FileCapture('123.pcap')
b=a[4]
c=a[3]
print("auth username:",b.smb.ntlmssp_auth_username)
print("auth hostname:",b.smb.ntlmssp_auth_hostname.lower())
print("auth lan manager response:",b.smb.ntlmssp_auth_lmresponse.replace(':',''))
print("auth ntlm response",b.smb.ntlmssp_auth_ntresponse.replace(':',''))
print("ntlm server challenge:",c.smb.ntlmssp_ntlmserverchallenge.replace(':',''),end='')
```

13 第二种方法输入exit()后按回车,再次输入

```
tshark -r 123.pcap -Tfields -e ntlmssp.auth.username -e ntlmssp.auth.hostname
-e ntlmssp.auth.lmresponse -e ntlmssp.auth.ntresponse -e ntlmssp.ntlmserverchallenge
```

/*
复制的时候要注意回车符
*/
14.0 在192.168.1.13这个主机上(XP系统)按Windows键,再按C键,打开控制面板
14.1 选择一个类别:外观和主题
14.2 选择一个控制面板图标:文件夹选项
14.3 在弹出的窗口中点击查看这一栏
14.4 取消勾选 使用简单文件共享 (推荐)这个选项
14.5 然后按确定
/*
接下来有两种方法
*/
15.0 第一种方法:在命令提示符中输入netsh firewall set opmode mode = DISABLE,输出应该是"确定"
15.1 然后返回1检查输出
15.2 第二种方法:倘若你已经执行了第一种方法,那你得
在命令提示符中输入netsh firewall set opmode mode = ENABLE,输出也应该是"确定"
15.3 然后继续输入net stop SharedAccess,输出应该包含"已成功停止"
15.4 然后返回1检查输出
16 过滤表达式为ntlmssp.negotiatentlm2==1,应该显示3,4,5一共3个数据报文(对于4.0.17版本以下的tshark)
17 第二种确定是否With Extended Session Security的方法
对于4.2.0版本以上的tshark
过滤表达式为ntlmssp.negotiateextendedsessionsecurity==1,还是显示3,4,5一共3个数据报文

这里是分割线 以下内容不适合脚本小子

参考Github上用户hashcat的仓库

hashcat/blob/master/src/modules/module_05500.c

```
static void transform_netntlmv1_key (const u8 *nthash, u8 *key)
{
  key[0] =                    (nthash[0] >> 0);
  key[1] = (nthash[0] << 7) | (nthash[1] >> 1);
  key[2] = (nthash[1] << 6) | (nthash[2] >> 2);
  key[3] = (nthash[2] << 5) | (nthash[3] >> 3);
```

```
  key[4] = (nthash[3] << 4) | (nthash[4] >> 4);
  key[5] = (nthash[4] << 3) | (nthash[5] >> 5);
  key[6] = (nthash[5] << 2) | (nthash[6] >> 6);
  key[7] = (nthash[6] << 1);

  key[0] |= 0x01;
  key[1] |= 0x01;
  key[2] |= 0x01;
  key[3] |= 0x01;
  key[4] |= 0x01;
  key[5] |= 0x01;
  key[6] |= 0x01;
  key[7] |= 0x01;
}
```

Python 自研

```
def transform_netntlmv1_key(nthash,key):
    nthash = [l & 0xff for l in nthash]
    key[0] =                    (nthash[0] >> 0)
    key[1] = ((nthash[0] << 7) | (nthash[1] >> 1)) & 0xff
    key[2] = ((nthash[1] << 6) | (nthash[2] >> 2)) & 0xff
    key[3] = ((nthash[2] << 5) | (nthash[3] >> 3)) & 0xff
    key[4] = ((nthash[3] << 4) | (nthash[4] >> 4)) & 0xff
    key[5] = ((nthash[4] << 3) | (nthash[5] >> 5)) & 0xff
    key[6] = ((nthash[5] << 2) | (nthash[6] >> 6)) & 0xff
    key[7] = (nthash[6] << 1)
    key[0] |= 0x01
    key[1] |= 0x01
    key[2] |= 0x01
    key[3] |= 0x01
    key[4] |= 0x01
    key[5] |= 0x01
    key[6] |= 0x01
    key[7] |= 0x01
    return key
```

Java 来自SOURCEFORGE

```
private static Key createDESKey(byte[] bytes, int offset) {
        byte[] keyBytes = new byte[7];
        System.arraycopy(bytes, offset, keyBytes, 0, 7);
        byte[] material = new byte[8];
        material[0] = keyBytes[0];
        material[1] = (byte) (keyBytes[0] << 7 | (keyBytes[1] & 0xff) >>> 1);
        material[2] = (byte) (keyBytes[1] << 6 | (keyBytes[2] & 0xff) >>> 2);
        material[3] = (byte) (keyBytes[2] << 5 | (keyBytes[3] & 0xff) >>> 3);
        material[4] = (byte) (keyBytes[3] << 4 | (keyBytes[4] & 0xff) >>> 4);
        material[5] = (byte) (keyBytes[4] << 3 | (keyBytes[5] & 0xff) >>> 5);
        material[6] = (byte) (keyBytes[5] << 2 | (keyBytes[6] & 0xff) >>> 6);
        material[7] = (byte) (keyBytes[6] << 1);
        oddParity(material);
        return new SecretKeySpec(material, "DES");
    }
public static byte[] getNTLM2SessionResponse(String password,
            byte[] challenge, byte[] clientNonce) throws Exception {
        byte[] ntlmHash = ntlmHash(password);
        MessageDigest md5 = MessageDigest.getInstance("MD5");
        md5.update(challenge);
        md5.update(clientNonce);
        byte[] sessionHash = new byte[8];
        System.arraycopy(md5.digest(), 0, sessionHash, 0, 8);
        return lmResponse(ntlmHash, sessionHash);
    }
private static byte[] lmResponse(byte[] hash, byte[] challenge)
            throws Exception {
        byte[] keyBytes = new byte[21];
        System.arraycopy(hash, 0, keyBytes, 0, 16);
        Key lowKey = createDESKey(keyBytes, 0);
        Key middleKey = createDESKey(keyBytes, 7);
        Key highKey = createDESKey(keyBytes, 14);
        Cipher des = Cipher.getInstance("DES/ECB/NoPadding");
        des.init(Cipher.ENCRYPT_MODE, lowKey);
        byte[] lowResponse = des.doFinal(challenge);
        des.init(Cipher.ENCRYPT_MODE, middleKey);
        byte[] middleResponse = des.doFinal(challenge);
        des.init(Cipher.ENCRYPT_MODE, highKey);
        byte[] highResponse = des.doFinal(challenge);
```

```
        byte[] lmResponse = new byte[24];
        System.arraycopy(lowResponse, 0, lmResponse, 0, 8);
        System.arraycopy(middleResponse, 0, lmResponse, 8, 8);
        System.arraycopy(highResponse, 0, lmResponse, 16, 8);
        return lmResponse;
    }
```

Python 自研 注释来自Ryan Grunsten

```
def netntlmv1_empty_checker(netntlm_t, password):
    hc_token_t = netntlm_t.split(':')
    lmResponse = bytes.fromhex(hc_token_t[3])
    ntresponse = bytes.fromhex(hc_token_t[4])
    challenge = bytes.fromhex(hc_token_t[5])
    clientNonce = lmResponse[0:8]
    """
    the 8 byte server challenge is concatednated
    with the 8 byte client challenge,yielding the "session nonce"
    """
    sessionHash = hashlib.md5(challenge)
    sessionHash.update(clientNonce)
    sessionHash = sessionHash.digest()[:8]
    """
    the session nonce is MD5 hashed,yielding a 16 byte hash
    the 16 byte hash is truncated to 8 bytes,yielding the NTLMv1 ESS hash.
    """
    hash = hashlib.new('md4', password.encode('utf-16le')).digest()
    """
    the 16 byte NT hash(AKA MD4 hash of user password) is null-padded to 21 bytes.
    """
    keyBytes = hash.ljust(21, b'\x00')
    """
    The 16-byte NTLM hash is null-padded to 21 bytes.
    """
    lowKey = keyBytes[0:7]
    middleKey = keyBytes[7:14]
    highKey = keyBytes[14:21]
    """
    The value is split into three 7-byte thrids
    """
    lowResponse = transform_netntlmv1_key(lowKey,bytearray(8))
    middleResponse = transform_netntlmv1_key(middleKey,bytearray(8))
    highResponse = transform_netntlmv1_key(highKey,bytearray(8))
    """
    These values are used to create three DES keys
    (one from each 7-byte third)
    """
    return new(lowResponse, MODE_ECB).encrypt(sessionHash)+new(middleResponse, MODE_ECB).encrypt(sessionHash)+
    new(highResponse, MODE_ECB).encrypt(sessionHash)==ntresponse
    """
    each third is used as a DES key to encrypt the NTLMv1 ESS hash
    resulting in three 8 byte ciphertext values
    These three 8 byte ciphertext values are concatednated
    resulting in the 24 byte NTLMv1 ESS Response.
    """
```

可以修改RawHash(获取方式在12-13步里)和lyc的值

```
if __name__ == "__main__":
    RawHash = "lyc::SXSYMXSYY-PC:4bf50c3614e725760000000000000000000000000000000000
    :59e47b73556d0dbfdb2705eb111eb1e0af0ecfc8a0281d63:bd756289919e57e6"
    lyc = "lyc"
    """
    lyc是我的朋友
    你们可以给她01luyicheng@gmail.com发邮件
    对文章有疑问也可以给我c27344098@gmail.com发邮件
    """
    print(netntlmv1_empty_checker(RawHash, lyc))
    """
    应该是True
    """
```