

Forum:[Hack Forums](#)

Thread: [The Red Team Playbook: C2 & RAT Deployment with .bat and .vbs](#)

CyberXero - September 21, 2026, 9:54 pm

The Red Team Playbook Hiding in Plain Sight with .bat and .vbs

If you are trying to deploy a RAT or C2 beacon on a modern OS, dropping a compiled .exe to disk is going to get you caught instantly. I've been researching how professional operators bypass this, and I want to share this methodology with the community. I started by wondering: "How do red teams use .bat and .vbs to bypass AV and EDR?"

That was when I came to learn that red teams frequently use .bat (Batch) and .vbs (Visual Basic Script) files as Living-off-the-Land (LotL) tools and initial droppers. Because these legacy script formats are natively trusted and executed by the Windows OS (via cmd.exe and wscript.exe / cscript.exe), red teams leverage them to blend malicious actions into normal administrative activity, bypass static signatures, delay analysis, and invoke more powerful execution engines under the radar.

How Scripts Evade Detection (5 Examples)

Abusing Native Trust & Reputation (BYOR): Security controls and cloud filters (like Microsoft SmartScreen) often grant higher reputation trust to native extensions like .bat, .cmd, or .vbs because they are standard files used daily in enterprise environments. By wrapping malicious logic inside a trusted format, red teams can bypass reputation prompts, initial file-hash blocklists, and basic static analysis.

Obfuscation and Dynamic Assembly: Static AV scanners look for known malicious strings, API calls, or byte signatures. Batch files and VBScript can be heavily obfuscated using string concatenation, environment variable expansion, XOR encoding, or reverse-string functions. The script looks like garbled text on disk; it only deobfuscates or reconstructs the payload dynamically in memory at the exact moment of execution.

Bypassing PowerShell Execution Policies: Many environments restrict direct PowerShell execution (powershell.exe -ExecutionPolicy Restricted). Red teams use .bat or .vbs as wrappers to invoke PowerShell or download payloads via alternative paths (e.g., using mshta, rundll32, or hidden flags like -NoProfile -WindowStyle Hidden -Command), effectively bypassing standard logging and block policies.

Disabling Security Controls (Tampering): Scripts run with whatever privileges the user context allows. If an operator gains local administrator rights, a batch script can execute rapid-fire commands to blind defenses. This includes stopping specific services, deleting shadow copies, modifying firewall rules, or running sc stop / net stop against known AV/EDR agents before the secondary tooling drops.

Staging and Multi-Stage Droppers: Dropping a heavy, fully compiled executable (.exe or .dll) onto a disk triggers immediate EDR behavioral blocking. Instead, the .bat/.vbs acts as a lightweight loader. It uses native utilities like certutil.exe or curl.exe to fetch the real payload from a remote C2 directly into memory, or unpacks it from a harmless-looking text file embedded in the script, drastically

reducing the disk footprint.

How Defenders Detect This Activity

Modern EDR solutions counter these techniques by focusing on behavioral telemetry and parent-child process relationships rather than just the file extension.

Unusual Process Spawning: Monitoring when `cmd.exe` or `wscript.exe` spawns unexpected child processes like `powershell.exe`, `whoami.exe`, or network utilities (`net.exe`, `certutil.exe`).

Command-Line Logging: Capturing full command-line arguments to spot obfuscated strings, base64 encoding, or remote URL requests.

Script Block Logging: Enabling advanced script block logging (AMSI / Event ID 4104) to catch the deobfuscated payload regardless of whether it started via a `.bat` or `.vbs` file.

Loaders vs. Packers: What They Do

While `.bat` and `.vbs` files handle the initial system interaction, loaders and packers handle the packaging and memory delivery.

Loaders: A loader's primary job is execution and memory injection. It reads a payload (often encrypted shellcode), allocates a secure space in the computer's memory, decrypts the payload, and executes it without ever writing the unencrypted malicious file to the hard drive.

Packers: A packer's primary job is obfuscation and compression on disk. It takes an existing executable, compresses or encrypts it, and wraps it in a new executable container (a stub). When run, the stub decompresses the original program directly into memory.

How do modern EDRs counter loaders?

Because loaders must eventually place the payload into memory to execute it, modern security tools focus heavily on memory forensics: **Memory Scanning (Fluctuating Protections):** EDRs look for memory regions marked as `RX` (Read/Execute) or `RWX` (Read/Write/Execute) that do not map back to a legitimate file on disk.

AMSI (Antimalware Scan Interface): Even if a loader obfuscates a payload, AMSI intercepts it at the exact moment it is decrypted in memory, just before execution.

API Hooking: EDRs monitor critical Windows API functions (like `VirtualAllocEx` or `CreateRemoteThread`) that loaders commonly use to request memory space.

How Scripts Handoff to a C2 (OPSEC Rules)

To get an Overlord C2 beacon to run under the watch of an aggressive EDR, the initial script wrapper acts purely as an orchestrator rather than doing the heavy lifting.

Evading AMSI: When a `.vbs` file runs via `wscript.exe`, Windows routes the script text through AMSI. If known C2 domains are seen in clear text, it blocks it instantly. Red teams bypass this by dynamically slicing strings with complex math or environment variables rather than basic Base64 (which EDRs decode automatically).

Living off the Land (LoLBins) for Staging: Spawning an unknown executable directly from a script triggers a high-fidelity alert. Instead, scripts invoke trusted binaries. A `.bat` might use a native utility to reconstruct an encrypted file in a temp folder, while a `.vbs` might use the `Wscript.Shell` object to silently call Microsoft-signed tools to map a remote network share and grab the next stage.

Does a Red Team Actually Require a Loader?

Yes. While you can execute basic commands purely out of a script, dropping a comprehensive C2 agent requires a sophisticated loader (like Jinx C2 or a custom build in C, C++, or Rust) for three main reasons:

Scripts Cannot Unhook EDR: EDRs monitor systems by injecting user-mode "hooks" into standard Windows system libraries (like ntdll.dll). Scripts cannot interact with memory closely enough to remove these. A compiled loader can manually read a fresh copy of ntdll.dll from disk and rewrite the hooked functions in memory, "blinding" the EDR.

Advanced Injection Techniques: To blend in, an Overlord beacon should run inside the memory space of a legitimate Windows process (e.g., explorer.exe or svchost.exe). Loaders handle the heavy mathematical and system-level operations required, such as Indirect Syscalls (talking directly to the OS kernel to bypass APIs) and API Hashing (hiding which APIs the malware uses).

In-Memory Sleep Obfuscation: Once an Overlord beacon is inside memory, it periodically "beacons" back to its server. During quiet periods, a routine YARA memory scan will spot it. A loader wraps the C2 agent with Sleep Obfuscation. When it sleeps, the loader encrypts the agent's memory region and changes permissions from Executable (RX) to Read-Only (RO), making it look like harmless data until it wakes up.

The Modern Attack Pipeline

In a professional engagement, the process follows a highly modular factory pipeline. You do not just deploy a raw RAT/C2 agent; you wrap it in multiple protective layers.

Step 1: The Core Agent (e.g., Overlord C2)

The operator uses the framework to generate the beacon as raw shellcode (a stream of machine code instructions). If saved directly to disk, standard AV would block its static signature instantly.

Step 2: The Crypter (Obfuscation)

The raw shellcode is passed through a crypter, encrypting it using algorithms like AES, RC4, or a custom XOR routine. The shellcode is transformed into completely randomized, harmless-looking data with no recognizable signature.

Step 3: The Loader (Execution Engine)

This encrypted data is embedded inside a Loader (like Jinx). The loader contains the de-crypter key. When executed, it unhooks the EDR, allocates hidden RAM, decrypts the shellcode inside that memory space, and points the CPU to execute it.

Stage 0: The Dropper (Where the .bat or .vbs fits in!)

Because modern email filters and browsers block compiled .exe loaders immediately, red teams use the script file as Stage 0 to sneak the loader onto the system. The script's only job is to execute a native command (like curl or certutil) to download the encrypted loader, or assemble it from hidden fragments on the disk, and then run it.

Disclaimer: The techniques and workflows detailed in this playbook are strictly for educational purposes, defensive telemetry research, and authorized red team engagements. Do not apply these methods to systems you do not own or have explicit permission to test.

If you found this playbook valuable and want to support future research, you can purchase the attached Supporter Token. All Bytes go directly toward funding the next volume.

Nice thread, I will read it all properly later but im pretty much up to date with this anyway.
But nice share

CyberXero - September 21, 2026, 10:08 pm

(Sep 21, 2026 10:04 PM)7 Wrote: [\[CITE\]](#) Nice thread, I will read it all properly later but im pretty much up to date with this anyway.

But nice share

I am quite fresh to the Scene but my brain is putting together the process smoothly, I thought I would share what i have learned, it took me hours of study as well as testing to get to my current point in a short time thanks for popping in [7](#)

roker8866 - September 21, 2026, 10:18 pm

This is such an excellent and detailed article! I'm going to take the time to read through every point carefully.

Thanks for the post!

CyberXero - September 21, 2026, 10:25 pm

(Sep 21, 2026 10:18 PM)roker8866 Wrote: [\[CITE\]](#) This is such an excellent and detailed article! I'm going to take the time to read through every point carefully.

Thanks for the post!

Thank you for popping in and taking the time to browse through, I hope this thread helps you out [roker8866](#)

DramaKing - September 22, 2026, 3:45 am

This is good material for the most part, but I wonder if you were studying up-to-date material. VBS support is either completely removed from Windows 11 now or is about to be. Also, even if running a script in an elevated context and knowing the relevant service name, trying to stop a service should work on about 0 EDRs. And you mention patching AMSI in-memory (which seems to trigger detection now, no matter what) but state that a script cannot unhook APIs, which is a contradiction.

I thought, too, that Jinx didn't do any encryption/decryption, except maybe with an encrypted channel.

CyberXero - September 22, 2026, 12:50 pm

(Sep 22, 2026 03:45 AM)DramaKing Wrote: [\[CITE\]](#) This is good material for the most part, but I wonder if you were studying up-to-date material. VBS support is either completely removed from Windows 11 now or is about to be. Also, even if running a script in an elevated context and knowing the relevant service name, trying to stop a service should work on about 0 EDRs. And you mention patching AMSI in-memory (which seems to trigger detection now, no matter what) but state that a script cannot unhook APIs, which is a contradiction.

I thought, too, that Jinx didn't do any encryption/decryption, except maybe with an encrypted channel.

Really appreciate the feedback!

I had to look into what you have said and you're entirely right about the VBS deprecation timeline as well as

how brutal EDR behavioral monitoring is on service manipulations now. Good catch on the AMSI/unhooking logic slip as well

I will definitely be restructuring those notes once I study more! Thanks for taking the time to pick it apart!

Even a shove in the right path would be beneficial to myself and others,

I just noticed .vbs and .bat are local script files that are indeed dying out, was looking into cryptify

which is why i went this direction,
whereas LotC is a methodology of using trusted cloud infrastructure and native legitimate binaries
to blend in right?