

Russian AI Slopsquatting Publishes 700+ Malicious NPM Packages

 opensourcemalware.com/blog/russian-ai-slopsquatting-npm-campaign

c0a15726-c5b1-4b0d-85e6-fe15553df9e2

August 6, 2026



Over the course of 48 hours a threat actor has published more than 700 malicious packages to the NPM registry. These packages appear to use AI slop squatted, or randomly generated typo-squatting package names, but all of them deliver a powerful RAT and infostealer payload. We're calling this campaign "WEL1DROPPER," and you can see the associated threat reports [here](#).

Update: This campaign is also being tracked under the name "Flooding Dropper."

The NPM packages do not use a preinstall or postinstall script. It doesn't need one. The README tells developers to load the library with `require("checkout-mobile-bnpl")`, and that single call starts the infection chain.

The downloader supports Windows, Linux, and macOS. It rotates through three Cloudflare Workers hosts for its primary payload delivery and falls back to reconstructing the payload from DNS TXT records hosted under `well.ru`.

That DNS fallback is the interesting part. Even if defenders block the obvious HTTPS hosts, the package has a second route for delivering the same executable.

TL;DR

- **Threat type:** Cross-platform downloader and dropper
- **Affected ecosystem:** npm
- **Malicious package:** `checkout-mobile-bnpl@35.6.9`
- **Execution trigger:** Importing the package with `require()`
- **Primary delivery:** HTTPS from three `workers.dev` hosts
- **Fallback delivery:** Base64-encoded payload chunks in DNS TXT records
- **Impact:** Downloads and silently executes native malware; the macOS payload establishes persistence and retrieves an additional beacon
- **Severity:** Critical

The OpenSourceMalware research team has analyzed multiple packages from this campaign. There are minor differences in the workers.dev C2, but otherwise the payloads are exactly the same.

A Mobile SDK That Doesn't Do Much Mobile SDK Work

At first glance, the example package we chose, checkout-mobile-bnpl, presents itself as a small mobile SDK. Its public API exposes a class with `init()`, `version()`, and `configure()` methods. None of those methods perform any meaningful checkout or buy-now-pay-later functionality.

The real behavior is tucked underneath the export at the bottom of `index.js`:

```
module.exports = {  
  CheckoutMobileBnpl,  
  create: (opts) => new CheckoutMobileBnpl(opts),  
  VERSION  
};
```

```
try { require("./_helpers"); } catch (_) {}
```

That last line loads `_helpers.js`. The helper file calls its own `run()` function immediately, so a developer does not need to initialize the alleged SDK or call a suspicious method. Importing the package is enough.

This is a useful reminder that npm malware does not require lifecycle scripts to execute. Install-script controls help, but they do not protect an application that later imports a malicious dependency during development, testing, or production startup.

What Does the Package Do?

Once loaded, the package identifies the operating system and CPU architecture and maps the victim to one of four payload paths:

Victim platform	Payload path
Linux x64	/pkg/package
Linux ARM64	/pkg/package-arm64
macOS x64 or ARM64	/pkg/loader_mac
Windows x64 or x86	/pkg/package.exe

It shuffles three attacker-controlled hosts and attempts an HTTPS GET against each one:

```
oob-worker.cf103-070.workers.dev  
oob-worker.cf102-baf.workers.dev  
oob-worker.cf99-9b3.workers.dev
```

The request is forced over IPv4, times out after 15 seconds, and uses `node-fetch/2.6` as its User-Agent. A response is accepted only when the server returns HTTP 200 and more than 1,000 bytes.

There is no signature check, trusted certificate pinning, expected hash, or other payload verification. Whatever those servers return is treated as an executable.

When HTTPS Fails, It Downloads the Payload Through DNS

If all three HTTPS hosts fail, the package switches to a platform-specific domain:

Victim platform DNS payload domain

Linux x64	sdk.dl.well.ru
-----------	----------------

Linux ARM64	ext.dl.well.ru
-------------	----------------

macOS	pkg.dl.well.ru
-------	----------------

Windows	net.dl.well.ru
---------	----------------

The package first requests a TXT record from `c.<domain>`. It parses the response as the number of payload chunks, accepting a value between 1 and 2,000.

It then requests numbered TXT records such as:

```
0.sdk.dl.well.ru
1.sdk.dl.well.ru
2.sdk.dl.well.ru
```

The returned strings are joined together and Base64-decoded into a binary buffer:

```
const cnt = await _dnsQuery("c." + domain);
const n = parseInt(cnt, 10);

// Queries 0.<domain> through (n-1).<domain>

return Buffer.from(parts.join(""), "base64");
```

This is not ordinary DNS service discovery. The TXT records are a secondary payload-delivery channel. DNS monitoring that only looks for known tunneling tools may miss this because the traffic consists of ordinary-looking TXT lookups to a small set of predictable names.

How the Payload Is Executed

After obtaining at least 1,001 bytes through HTTPS or DNS, the downloader generates a random eight-character hexadecimal identifier and writes the payload to disk.

On Linux and macOS, it uses:

```
/var/tmp/.cache_<8 hex characters>
```

The file is made executable and launched through:

```
/bin/sh -c "/var/tmp/.cache_<id> &"
```

On Windows, it uses:

```
%TEMP%\dotnet_diag_<8 hex characters>.exe
```

It launches the file in the background with:

```
cmd.exe /c start /b %TEMP%\dotnet_diag_<id>.exe
```

Both execution paths are detached and have their standard input and output ignored. The filenames—`.cache_` on Unix and `dotnet_diag_` on Windows—are designed to look uninteresting among legitimate temporary files.

The package also creates a marker file to avoid running again for 21,760 seconds, or just over six hours:

```
/tmp/.analytics_state
%TEMP%\analytics_state
```

The telemetry naming is camouflage. This file does not store analytics; it is a rate-limit marker for the downloader.

Is This a Multi-Stage Loader?

The first-stage JavaScript from the NPM packages behave like a downloader. It makes HTTPS GET requests and DNS TXT queries, but there are no active POST request, credential collection routine, or host-data exfiltration path in `_helpers.js`. That's what the second stage is for: The second-stage payload can perform additional discovery, credential theft, persistence, or command-and-control activity that is not visible in the npm package itself.

The second stage comes in 4 platform specific versions, and the OSM team analyzed all four.

There are also no hardcoded IP addresses in the package. The Cloudflare Workers and `well.ru` hostnames resolve dynamically, so defenders should prioritize the domain names, DNS queries, process behavior, and dropped-file patterns over a static IP blocklist.

The 80 KB Telemetry Decoy

The package contains a second file, `lib/telemetry.js`, weighing in at roughly 80 KB. It implements a large, plausible-looking telemetry SDK, but buried inside it is another version of the same native downloader design: HTTPS retrieval, DNS TXT reconstruction, temporary-file creation, permission changes, and detached execution.

The package entry point does not import this file, and it contains no additional hardcoded infrastructure. Its network destinations have to be supplied by a caller. The active infection path is the much smaller `_helpers.js` file.

The oversized telemetry implementation appears intended to add noise and make the malicious behavior look like native profiling or analytics functionality during a quick review.

Inside the Native Payloads

The first-stage JavaScript is only the beginning of the chain. The OSM research team analyzed two binaries served by the infrastructure: a Linux x86-64 executable and a universal macOS Mach-O containing both Intel and Apple Silicon versions.

Sample	Format	SHA-256
<code>oob-worker.cf103-070.workers.dev.linux-second-stage.payload</code>	Linux x86-64 ELF, statically linked and UPX- packed	<code>7e486657f30594afda379b97030252a09a19fe8055e25c9e371544f59bd8e9e3</code>
<code>oob-worker.cf99-9b3.workers.dev.second.stage.payload</code>	Universal macOS Mach-O, x86-64 and ARM64	<code>c214746c74cae8ece8bdaf69aa05da4db6ce013f9e77452d1eed1a002fd9ba00</code>

The macOS Payload Downloads Another Payload

The macOS binary is not the final stage. It contains an additional download path named:

```
/pkg/beacon_mac.bin
```

It uses `libcurl` for HTTPS and `libresolv` for direct DNS queries. Its imports include `curl_easy_perform`, `res_9_query`, `fork`, `setsid`, `execl`, `chmod`, `ptrace`, and `sysctlbyname`. Together, these support HTTPS and DNS delivery, file installation, detached execution, and anti-debugging.

The likely macOS attack chain is:

1. Look for debuggers, instrumentation, packet capture, and VMware.
2. Query the machine's memory size as a likely sandbox heuristic.
3. Test remote or local health endpoints.
4. Retrieve `/pkg/beacon_mac.bin` through a Cloudflare Workers proxy.
5. Fall back to DNS TXT delivery if HTTPS fails.
6. Write the payload beneath `~/local/share/runtime`.
7. Create and load a LaunchAgent for persistence.
8. Start the executable in a detached process.

More Configuration Hidden With XOR

The macOS binary hides its infrastructure strings with a single-byte XOR operation using the key `0x9c`. Static decoding recovered five more Cloudflare Workers hosts:

```
package-proxy[.]cf5oobworker.workers.dev  
package-proxy[.]cf6oobworker.workers.dev  
package-proxy[.]cf7oobworker.workers.dev  
package-proxy[.]cf8oobworker.workers.dev  
package-proxy[.]cf11oobworker.workers.dev
```

Combined with the embedded `/pkg/beacon_mac.bin` path, these hosts appear to be third-stage payload proxies. The binary builds requests with the template `https://s:%d%s`, so the connection can include an explicit port. The exact configured port could not be recovered confidently from strings alone.

The macOS loader also contains another DNS delivery domain:

```
dl[.]well.ru
```

Its query format strings are:

```
c.%s.%s  
%d.%s.%s
```

This suggests a count query followed by numbered payload chunks, likely resembling:

```
TXT c.<session-or-host-id>.dl.well.ru  
TXT 0.<session-or-host-id>.dl.well.ru  
TXT 1.<session-or-host-id>.dl.well.ru
```

That is the same basic delivery technique used by the JavaScript first stage, with an extra session or host label in the query name.

Health Checks and Possible Decoy Infrastructure

Three other XOR-hidden domains are embedded in both architecture slices:

```
nexus[.]tcsbank.ru
repo-linux[.]tcsbank.ru
alertmanager[.]cloudpayments.ru
```

The binary contains the formatter `https://%s/health`, so these appear to be health checks, connectivity tests, fallback services, or decoy traffic. Static evidence alone does not prove that every embedded domain is attacker-owned. Defenders should treat them as investigation leads and correlate them with the process and filesystem indicators in this post.

The binary also probes:

```
http://127.0.0.1:4444/health
```

This is the only literal IP address found in either native sample. Because it is the local loopback address, it is not an external C2. It may test whether another local malware component is already active.

macOS Persistence

The loader constructs the following directory and file paths beneath the victim's home directory:

```
~/./local/share/runtime
~/./local/share/runtime/.lock
~/./local/share/runtime/com.apple.runtime
```

It also creates:

```
~/Library/LaunchAgents/com.apple.windowserver.helper.plist
```

The embedded plist identifies itself as `com.apple.windowserver.helper`, runs its configured executable at login, and keeps restarting it after unsuccessful exits. Both standard output and standard error are redirected to `/dev/null`, and restart attempts are throttled to once every 60 seconds.

The loader activates the persistence entry with:

```
launchctl load -w '<path>/com.apple.windowserver.helper.plist' 2>/dev/null
```

The names `com.apple.runtime` and `com.apple.windowserver.helper` are camouflage. Neither artifact is a legitimate Apple component in these locations.

Anti-Analysis Checks

The macOS payload explicitly looks for several analysis tools and virtualization artifacts:

```
lldb
debugserver
dtrace
frida
/usr/local/bin/wireshark
/Applications/VMware Fusion.app
/Library/Application Support/VMware Tools
```

It imports `ptrace`, `sysctl`, and `sysctlbyname` and queries `hw.memsize`. This combination indicates debugger detection, instrumentation detection, VMware detection, and a likely low-memory sandbox check.

The exact reaction to each detected artifact—such as exiting, sleeping, or suppressing network activity—would require further control-flow reconstruction. The anti-analysis intent itself is clear from the combined strings and imports.

The Linux Payload Is UPX-Packed

The Linux sample is a statically linked, non-PIE x86-64 ELF with its section header table removed. It contains multiple UPX markers and identifies its packer as UPX 3.96.

The original program's code, strings, and configuration are compressed. There is no dynamic section or useful import table, so ordinary static string and import analysis cannot expose its inner behavior. The visible `/proc/self/exe` string is consistent with the UPX loader reading its own image and should not be mistaken for an IOC by itself.

This file downloads a second set of binaries from one of a number of Cloudflare Worker URLs:

```
https://oob-worker[.]cf99-9b3.workers.dev/pkg/package
https://oob-worker[.]cf99-9b3.workers.dev/pkg/package-arm64
https://oob-worker[.]cf99-9b3.workers.dev/pkg/loader_mac
https://oob-worker[.]cf99-9b3.workers.dev/pkg/package.exe
```

This last stage served by the Cloudflare URL delivers what another researcher is describing as a Sliver implant. Sliver is an open-source command and control framework, written in Go, and maintained by Bishop Fox. It is intended to be used by red teams and offensive security practitioners. However, Sliver is often used by malicious threat actors as well. We have not confirmed that this last stage is Sliver yet, but will update this section if we do.

Attribution

The use of the `wel1[.]ru` domain to host the TXT records could indicate that the actor is Russian. Additionally, the malware mentions several Russian financial institutions including `cloudpayments[.]ru` and `tcsbank[.]ru` which both appear to be legitimate.

We think this campaign might be an evolution of the Moika tech malware campaign that saw 250+ NPM packages published to the registry in April/May of this year.

You can see the OSM list of Moika packages [HERE](#).

The Moika and `wel1dropper` campaigns share some tradecraft including:

- The use of the "oob" string in file and server names
- The focus on Russian financial institutions and mobile payments
- A focus on fake telemetry camouflage to validate legitimacy
- Moika and `wel1dropper` both have kill switches that work in similar ways

Indicators of Compromise (IOCs)

Malicious NPM Packages

We are currently tracking 788 packages in this campaign. You can see them all [HERE](#).

We are also maintaining a list of affected packages on [GitHub](#).

HTTPS Payload Hosts

```
oob-worker[.]cf103-070.workers.dev
oob-worker[.]cf102-baf.workers.dev
oob-worker[.]cf99-9b3.workers.dev
```

```
package-proxy[.]cf5oobworker.workers.dev
package-proxy[.]cf6oobworker.workers.dev
package-proxy[.]cf7oobworker.workers.dev
```

```
package-proxy[.]cf8oobworker.workers.dev
package-proxy[.]cf11oobworker.workers.dev
```

DNS Payload Domains

```
sdk[.]dl.well.ru
ext[.]dl.well.ru
pkg[.]dl.well.ru
net[.]dl.well.ru
dl[.]well.ru
```

Defenders should also search for TXT queries matching:

```
c.<domain>
<integer>.<domain>
c.<session-or-host-id>.dl.well.ru
<integer>.<session-or-host-id>.dl.well.ru
```

Additional Embedded Domains

```
nexus[.]tcsbank.ru
repo-linux[.]tcsbank.ru
alertmanager[.]cloudpayments.ru
```

These domains are embedded in the macOS sample and appear related to health checks. Static analysis does not establish whether they are attacker-owned, compromised, or used as decoys.

Payload URL Patterns

```
https://oob-worker[.]cf103-070.workers.dev/pkg/package
https://oob-worker[.]cf103-070.workers.dev/pkg/package-arm64
https://oob-worker[.]cf103-070.workers.dev/pkg/loader_mac
https://oob-worker[.]cf103-070.workers.dev/pkg/package.exe
```

```
https://oob-worker[.]cf102-baf.workers.dev/pkg/package
https://oob-worker[.]cf102-baf.workers.dev/pkg/package-arm64
https://oob-worker[.]cf102-baf.workers.dev/pkg/loader_mac
https://oob-worker[.]cf102-baf.workers.dev/pkg/package.exe
```

```
https://oob-worker[.]cf99-9b3.workers.dev/pkg/package
https://oob-worker[.]cf99-9b3.workers.dev/pkg/package-arm64
https://oob-worker[.]cf99-9b3.workers.dev/pkg/loader_mac
https://oob-worker[.]cf99-9b3.workers.dev/pkg/package.exe
```

```
https://package-proxy[.]cf5oobworker.workers.dev:<port>/pkg/beacon_mac.bin
https://package-proxy[.]cf6oobworker.workers.dev:<port>/pkg/beacon_mac.bin
https://package-proxy[.]cf7oobworker.workers.dev:<port>/pkg/beacon_mac.bin
```

```
https://package-proxy[.]cf8oobworker.workers.dev:<port>/pkg/beacon_mac.bin
https://package-proxy[.]cf11oobworker.workers.dev:<port>/pkg/beacon_mac.bin
```

File and Process Indicators

```
/tmp/.analytics_state
/var/tmp/.cache_<8 hex characters>
%TEMP%\analytics_state
%TEMP%\dotnet_diag_<8 hex characters>.exe
/bin/sh -c /var/tmp/.cache_<id> &
cmd.exe /c start /b %TEMP%\dotnet_diag_<id>.exe
```

```
~/.local/share/runtime/.lock
~/.local/share/runtime/com.apple.runtime
~/Library/LaunchAgents/com.apple.windowserver.helper.plist
launchctl load -w '<path>/com.apple.windowserver.helper.plist'
```

http://127.0.0.1:4444/health

SHA-256 Hashes

README.md

0fc30f82e1fa5e51a6c0c43f3ed7f13592ea731cb331e43a4d085df60a4db8b6

_helpers.js

94ef6b1c4a9d31f78f446d053048bcef34fd88f4376a1a46f7f777a9e9c83a29

index.js

b74c5675725911c62091bdf40714df760cc2af7a88360d21065f4e1c878aa8f0

package.json

e2650e9aa2f924433ba422857b22ee7c5996b5ad306f3f903283f6a13e248935

lib/telemetry.js

a3e2ffb440b779d30da3ff282affd649731088e8570df7b1aa72742d995b782c

Linux x86-64 native payload

7e486657f30594afda379b97030252a09a19fe8055e25c9e371544f59bd8e9e3

macOS universal native payload

c214746c74cae8ece8bdaf69aa05da4db6ce013f9e77452d1eed1a002fd9ba00

What Should Defenders Do?

If you identify that any of these 800+ NPM packages has been installed in your environment, appears in a lockfile, software bill of materials, package-manager cache, build log, or deployed application, do not assume that removing the dependency is sufficient.

Determine whether the package was ever imported. Review DNS logs for TXT lookups under `well1.ru`, proxy logs for all eight Cloudflare Workers hosts, and endpoint telemetry for the dropped-file and process patterns above. On macOS, specifically hunt for the fake runtime executable and LaunchAgent. Because the final beacon remains unavailable for analysis, a confirmed execution should be handled as a potential host compromise.

Rotate credentials accessible to affected developer workstations or CI runners, including npm tokens, GitHub tokens, cloud credentials, signing keys, and deployment secrets. Rebuild affected systems from a known-good state where appropriate.

Conclusion

This campaign is noisy, but it also packs a punch. It hides behind a minimal SDK facade, executes on import instead of using an install script, supports the three major desktop operating systems, rotates its HTTPS infrastructure, and uses DNS TXT records as a fallback payload channel. Its macOS payload adds anti-analysis checks, user-level persistence, five more download proxies, and another beacon-delivery stage.

The most important lesson is simple: blocking npm lifecycle scripts does not eliminate npm malware. Dependency code can wait until the application imports it, then do exactly the same damage.

If you encounter similar packages or suspicious activity, please report them to [OpenSourceMalware.com](https://www.opensourcemalware.com).

Stay safe out there.