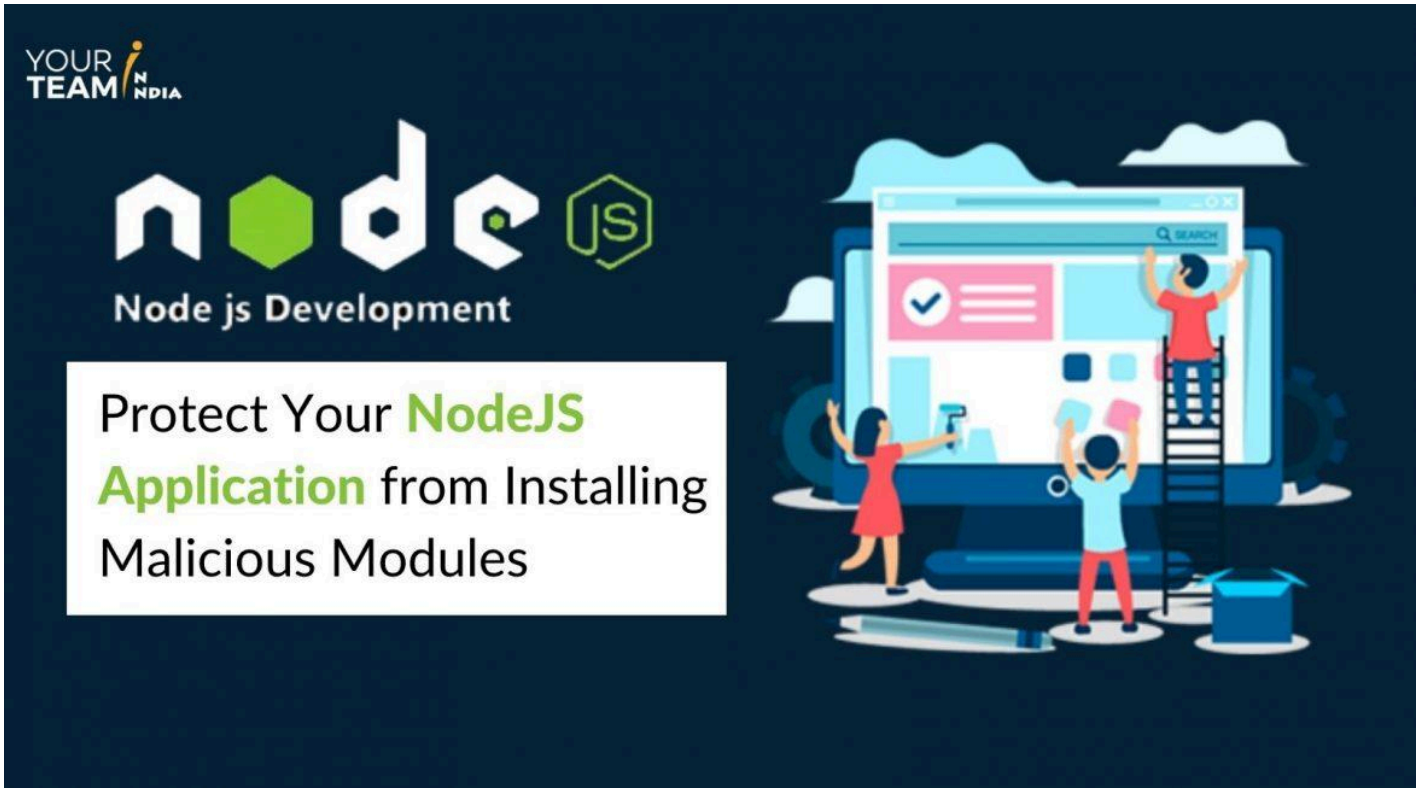
Guide to Protect Your NodeJS Application from Installing Malicious Modules



How would you feel if you figured out that nasty hackers have attacked your recently built Node.JS app? That didn't sound so pleasant, right! Well, hackers will always find ways to inject malicious code into your app- and not just through dependencies but also unintended vulnerabilities.

Being an expert Node.JS developer, you must always be ready with solutions for such unfavorable events.

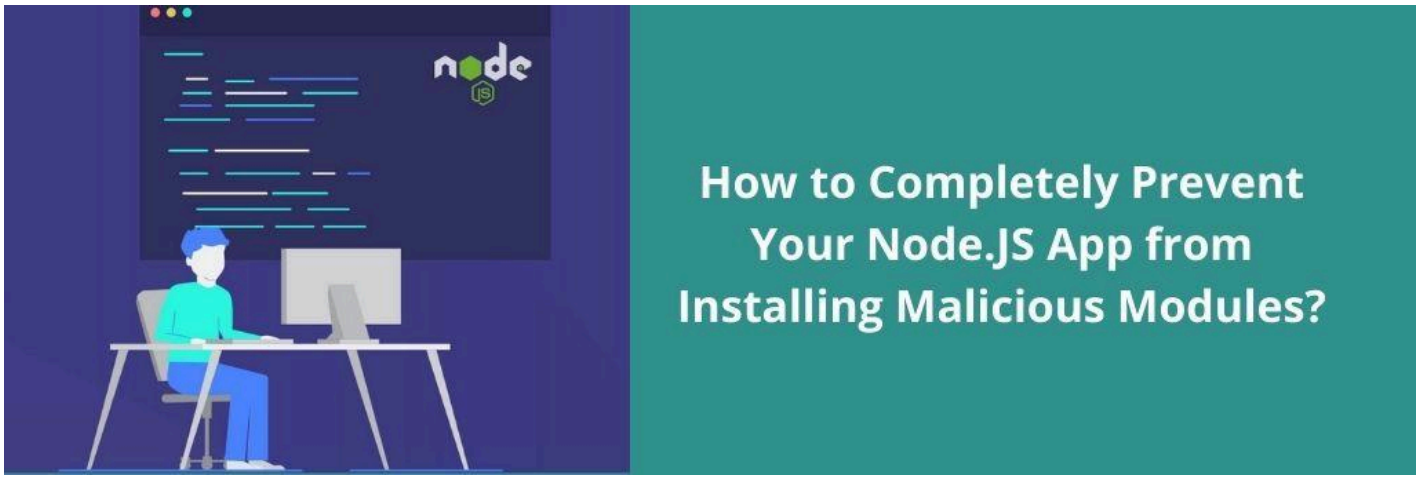
This post covers how you (being an expert node.js developer) can protect your node.js application from installing the malicious module.

Attackers are Not Stopping Any Soon!

You know how attackers work. It is their job to push the malicious modules into your app. They want access to your system, the information fed, and then send it out to the third party. They even intend to run destructive commands.

Beware of Typosquatting! The malicious modules will look so real, named exactly like the real ones, and you could be accidentally installing them yourself.

How to Completely Prevent Your Node.JS App from Installing Malicious Modules?



You can take these preventative measures and secure your app:

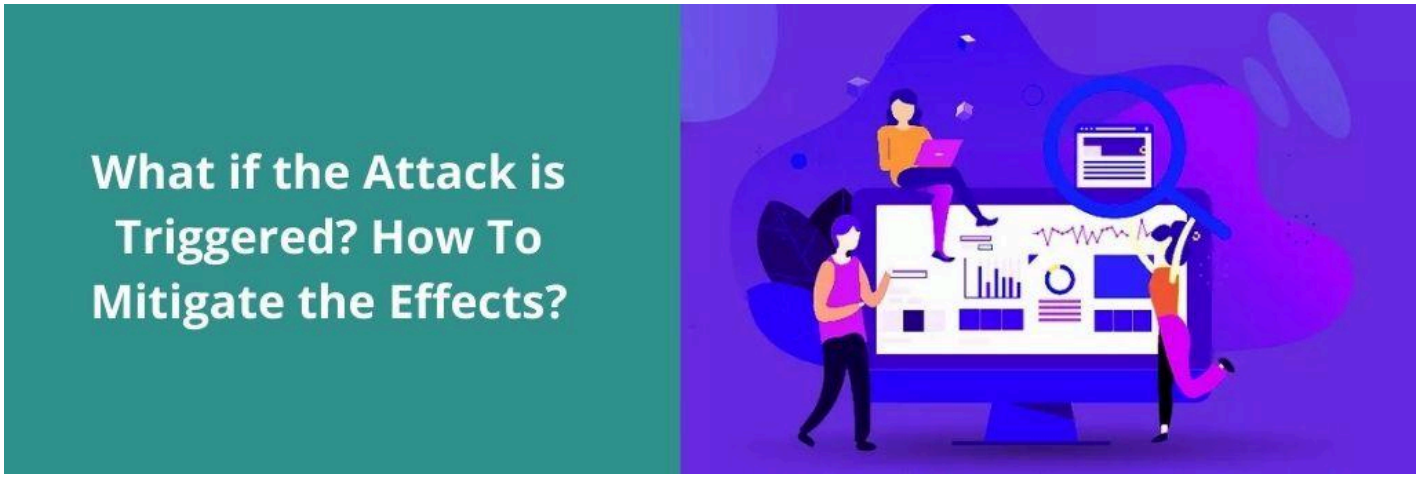
- **Lock your Dependencies**

You can easily lock the dependencies using package-lock.json or yarn. lock to avoid receiving automatic updates when you are doing npm/yarn install in your server. Locking dependencies reduces the chances of receiving malicious updates.

- **Use npm audit/GitHub Security Alerts**

Get notified about the security vulnerabilities in the dependencies using GitHub security alerts, npm audit, and Synk. **Hire Developers** to get an expert solutions.

What if the Attack is Triggered? How Can You Mitigate the Effects, Then?



If ever, your Node.JS application is injected with malicious code, we suggest you:
Restrict permissions (On Priority)

For example, limit the access to the filesystem. Or, simply configure iptables to restrict connecting the app to just a few trustworthy domains.

Now, What Can you Do Inside Node.JS?

Of course, being a developer, you would also want the **Node.JS framework** to be more secure to avoid such nuances in the future. However, we can just make the most of what we have on the table now.

Is there anything that we can do about such instances right now?

Yes, we do have options to control it as soon as it triggers.

As you know, Node.JS follows JavaScript (JS), the nature of which is pretty dynamic, allowing it to hack the runtime.

- Prefer running code in a sandboxed environment. Make sure to do it with vm module while passing a custom 'require' function to limit the access to specific modules. Basically, the core modules will be wrapped safely to avoid any unwanted access.
- You can even hijack the require() calls besides manipulating the code inside.
-
    - Override the core modules directly

Need Help? Talk To Our Node.Js Experts Now!

Try Overriding the Core Modules Directly

We are going to readFileSync here so that the access to specific modules is highly restricted.

```
// malicious.js const fs = require('fs') const secrets = fs.readFileSync('/sy
stem/secrets.txt', 'utf8') console.log(secrets);
```

To prevent malicious code, you will just have to implement a cage.js file to override the fs core module.

```
// cage.js const fs = require('fs') const path = require('path') const wrap =
(module, name, wrapper) => { const original = module[name] module[name] = wra
pper(original) } wrap(fs, 'readFileSync', (readFileSync) => (...args) => { co
nst [filepath] = args const fullpath = path.resolve(filepath) if (fullpath.st
artsWith('/system/')) { throw new Error('You do not have permissions to acces
s this file') } return readFileSync(...args) }) // Prevent further changes Ob
ject.freeze(fs)
```

So, once you have done this, and even if the malicious code has already run on the app, the malicious code will not have enough power to access the file's content. Simply put, any

potential error will be kicked out of your Node.js application, allowing users to use the app seamlessly.
Read our other post on **advantages of Node.js** how custom Node.js web application can benefit.

Apart from That, You can Consider these Points for Better Security

- **Add HTTP Headers**

With HTTP headers, both the client and server transfer information about the connection that they are trying to establish, where the resource is requested and even returned. If by any chance, you ignore it, you will only be alleviating leakage of sensitive information. You can use Helmet.js to secure HTTP headers, which have 12 Node modules for securing multiple HTTP headers.

- **Password Encryption**

You can convert your password into unreadable messages using the encryption key that both the recipient and the sender know. You can opt for either symmetric encryption or asymmetric one, where you and the recipient will have the same key for encryption and decryption or different keys, respectively. Using this technique is amongst the best ways to secure your Node.JS app from any unauthorized access.

- **Discard Sensitive Data (After Use)**

Sensitive data is the most common point of attack; hence it needs to be discarded after use. Attackers can easily steal secrets, user sessions, and important information. Storing sensitive information unnecessarily is not recommended at all as that increases the risk of data being stolen. To avoid such a situation, you should use stronger passwords.

Looking For Node.Js Experts? We Are Here To Help!

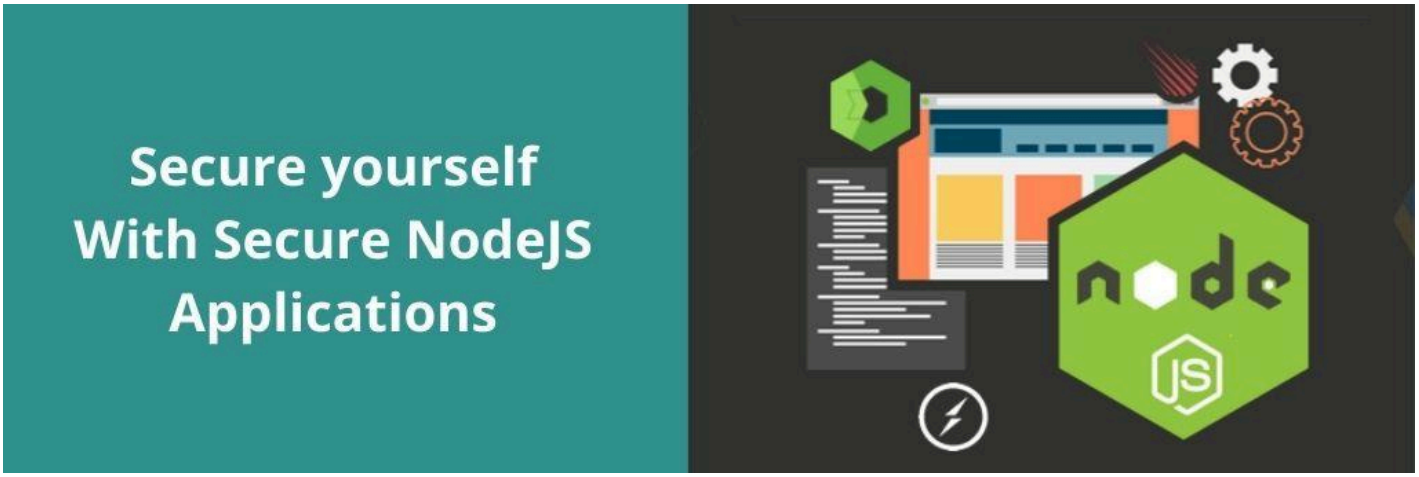
- **Sanitize All Incoming Inputs**

You might have heard about the XSS attack, where malicious JS code is executed on client-side facing apps. In this attack, input is received from an untrusted source from the client-side. The same is received and accepted by the backend to process further without any formal validation. So, when you try to send the response back to the user may contain malicious JS code. To stay safe against such attacks, every **Node.js developer** is advised to treat each incoming data request as unsafe to avoid further complications.

- **Use Environment File to Store All Credentials**

Credentials are significant when it comes to making an app secure. We have many credentials, i.e., secret key, database credentials, API key. We try not to embed these credentials hardcoded in the codebase. Developers must practice having an environment that is not shared over the version control to ensure that the credentials are safe.
Attackers can steal important information in the commit history, making a different environment with different environment files essential. In addition, environment files can assist programmers in safeguarding the security credentials in a much more efficient way.

Conclusion



Malicious code in Node.js is a growing concern, and we need to fix this concern to experience more secure applications in the future. The fact that the apps rely on more dependencies, making the attack surface larger than ever before. You can make your app code more secure by using tools like:

- GitHub security alerts
- NPM Audit

Remember, Node.js gives you enough space to try different techniques to secure yourself; all you need to do is what is demonstrated above. Hire expert Node.Js developers if you are a project manager and wish to hire the best developers to develop a secure app hassle-free.

Originally posted on YourTeam in India. Click here to read more.