Advanced Open Source Development Final Project

Brian Colantonio

New England Institute of Technology

Advanced Open Source Development Final Project

Technology affords us incredible feats which were unimaginable merely 20 years ago. Rapid growth in technology naturally requires our software to keep up with the pace. Fast forward to present day and the basic operation of our entire civilization rests on a foundation built in software. Now, everything from managing your bank account, watching a movie, and recording musical projects, to the operation of our traffic grid, and energy systems like that which provides our electricity, rely on software for instructions on how to operate. Quick thinking and strong problem solving is required to prevent or recover from vulnerabilities of which are uncovered and exploited by savvy troublemakers all around the world. Vulnerabilities in software pose such a risk for problems of detrimental proportions that the multibillion-dollar industry of software/cyber security has since been established in the ongoing effort to mitigate their effects.

As students, we have received various introductions to methods and practices utilized to secure our code. We have also realized by now that our field of expertise is so vast that it would be impossible to receive a formal education which covers every aspect of software development. Therefore it is important never to cease learning through our own research and investigation.

An opportunity has been provided to dig deeper into aspects of Node.JS, MongoDB, and the many modules and tools which have led us to this point. With the great importance of software and data security, I have decided to investigate further into the best practices while securing our Node.js projects.

A relatively easy effort in beginning to reinforce the security of your project would be to ensure that the utilized modules and dependencies are up to date. According to express.js documentation, security and performance issues in former versions will go unfixed and discontinuation of their use is emphasized. It is also good practice to examine the log of security updates for associated tools before considering which version to implement. Naturally, new versions can include flashy new capabilities which can be rather tempting to work with, but production is not recommended with vulnerable code and one should implement the latest stable versions as best efforts to avoid the exploitation of bugs.

So, you’ve built the strongest code in the world? —Congratulations. But have you considered network vulnerabilities? Some of the most common software hacks are done by wrongfully inspecting the packets sent while servers and clients are communicating. The Ajax and post transmissions you once thought to be safe and hidden can be exposed relatively easily using packet sniffing utilities. If your program deals with or transmits sensitive data, it is important to use an encryption service in effort to prevent these types of attacks. Secure Socket Layer (SSL) is a standard security technology which provides an encrypted link between a server and the browser (Info.ssl.com.) The latest progression of SSL encryption however, is Transport Layer Security (TLS;) which encrypts data before it is sent from the client to the server. It is recommended by express.js security documentation to migrate from SSL to TLS with use of useful tools to configure free TLS certificates such as ‘Let’s encrypt.’ Which is a free, open sourced, and open certificate authority provided by the Internet Security Research Group.

Another important security consideration to be made is the setting of headers. Node.js affords us some tools which help to avoid well-known web vulnerabilities by appropriately setting HTTP headers for us; one of which, is ‘Helmet.’ Instead of just one tool, however, Helmet can actually be thought of as just a collection of smaller middleware functions that set security-related HTTP response headers. These middleware functions include, but are not limited to: ‘xssFilter,’ which sets x-xss-Protection to enable the Cross-site scripting (XSS) filter in most recent web browsers, ‘noSniff,’ which sets X-Content-Type-Options to prevent browsers from MIME-sniffing a response away from the declared content type, ‘ieNoOpen,’ which sets X-Download-Options for IE8+, and many more.

Following these guidelines in consideration to your project’s security will have you well on your way to providing a safer experience for your end-user. There are countless more strategies used in the effort to secure Node.js, but it is up to us to be aware and stay vigilant. Keep an eye out for ‘Node Security Project’ or ‘Snyk’ advisories that affect the modules that your app utilizes; both databases are great resources when it comes to Node security. Stay informed and familiarize yourself with known web vulnerabilities, such as: SQL/NoSQL Injection, Broken Authentication, and Sensitive Data Exposure; --and do your best while taking precautions to avoid them.

References

OWASP FOUNDATION. (2017). OWASP Top 10 Application Security Risks - 2017. Retrieved from https://www.owasp.org/index.php/Top\_10-2017\_Top\_10

StrongLoop, IBM, and other expressjs.com contributors. (2017) Production Best Practices: Security. Retrieved from https://expressjs.com/en/advanced/best-practice-security.html

DSSL.com. What is SSL? Retrieved from info.ssl.com/article.aspx?id=10241

Last Name, F. M. (Year). *Book Title.* City Name: Publisher Name.

Last, F. M. (Year Month Date Published). Article title [Type of blog post]. Retrieved from URL.