2ND EDITION

The Node.JS Security Handbook

sqreen

INTRODUCTION

Damn, but security is hard.

It's not always obvious what needs doing, and the payoffs of good security are at best obscure. Who is surprised when it falls off our priority lists? We'd like to offer a little help if you don't mind. And by "help" we don't mean "pitch you our product"—we genuinely mean it. Sqreen's mission is to empower engineers to build secure, reliable web applications. We've put our security knowledge to work in compiling an actionable list of best practices to help you get a grip on your security priorities for Node.js environments. It's all on the following pages. We hope you find it useful. If you do, share it with your network. And if you don't, please take to Twitter to complain loudly—it's the best way to get our attention.

The Sqreen Team
@SqreenIO
howdy@sqreen.com

WANT THIS HANDBOOK AS A PDF? GO TO:

https://www.sqreen.com/resources/nodejs-security-handbook

CODE

☑ Use a prepublish/pre-commit script to protect yourself

Before committing your code or publishing your package to a repository, you should ensure that no sensitive data will be shipped. Using a pre-commit hook or a pre-publish script helps to prevent such leaks. You should particularly look for database credentials, API keys, or configuration files.
A few npm packages can help placing pre-commit hooks: https://www.npmjs.com/package/pre-commit

You can also use publish-please package: https://www.npmjs.com/package/publish-please, and add a pre-publish script in your package.json file: https://docs.npmjs.com/misc/scripts

☑ When using a templating engine, do not use unsafe methods

When using a templating engine, you should know which syntax can introduce XSS vulnerabilities. For instance, Pug (formerly, Jade) escapes all inputs by default unless you use the '!' symbol.

Check Pug's documentation: https://pugjs.org/language/code.html
Mustache documentation: https://mustache.github.io/mustache.5.html

☑ Perform data validation on everything you don't control

All user data that gets into your application should be validated and escaped to avoid various kinds of injections.

Learn more about MongoDB injections:

https://blog.sqreen.com/mongodb-will-not-prevent-nosql-injections-in-your-node-js-app/
Use Joi to perform data validation: https://www.npmjs.com/package/joi
Learn more about SQL injections: https://en.wikipedia.org/wiki/SQL_injection
Learn more about code injections in Node.js: https://ckarande.gitbooks.io/owasp-nodegoat-tutorial/content/tutorial/a1_-_server_side_js_injection.html

☑ Avoid using fs, child_process and vm modules with user data

The fs module allows access to the file system. Using it with unsafe data can allow a malicious user to tamper with the content of your server. The child_process module is used to create new processes. Using it can allow a malicious user to run their own commands on your server. The vm module provides APIs for compiling and running code within V8 Virtual Machine contexts. If not used with a sandbox, a malicious user could run arbitrary code within your web application.

Read more:
Node.js fs module documentation: https://nodejs.org/api/fs.html
Node.js child_process module documentation: https://nodejs.org/api/child_process.html
Node.js vm module documentation: https://nodejs.org/api/vm.html

☑ Don’t implement your own crypto

The problem with cryptography is that you don’t know you are wrong until you are hacked. So don’t do your own crypto. Use standards instead. For most crypto-related operations, the ‘crypto’ core module can help you.

Read more:
https://nodejs.org/dist/latest-v8.x/docs/api/crypto.html
https://en.wikipedia.org/wiki/Bcrypt
http://crypto.stackexchange.com/questions/43272/why-is-writing-your-own-

encryption-discouraged
https://blogs.dropbox.com/tech/2016/09/how-dropbox-securely-stores-your-passwords/

☑ Ensure you are using security headers

Websites are exposed to many different classes of vulnerabilities, and some may be prevented by appropriately configuring the server. Best practices include adding headers such as HSTS, X-Frame-Options, X-Content-Type-Options, etc. Add in a Content Security Policy if possible.

Read more:
https://www.npmjs.com/package/helmet
https://www.sqreen.com/scanner
https://securityheaders.com
https://www.ssllabs.com/

☑ Go hack yourself

Once in a while, the entire technical team should sit together and spend time targeting all parts of the application, looking for vulnerabilities. This is a great time to test for account isolation, token unicity, unauthenticated paths, etc... You will heavily rely on your browser's web console, curl, and 3rd party tools such as Zap (https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project).

The benefit of doing these test sessions yourselves is that your team has the best understanding of your application, and likely where the weak points are. Showing that they can be exploited (or not) is valuable feedback for the team. These sessions complement external pentests quite well.

Read more:
https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents

☑ Run security linters on your code

Pre-production analysis tools like static code analysis (SAST) can help identify some of your low-hanging security fruits. They also improve the overall security awareness of your team when the checks are automatically integrated into the code review process. But keep in mind that these tools generate a lot of false positives that can quickly overwhelm you with meaningless alerts. The best practice is to make them part of your process, but not too rely too heavily on them.

Some good tools:
https://www.owasp.org/index.php/Source_Code_Analysis_Tools
http://eslint.org/ with https://github.com/nodesecurity/eslint-plugin-security
https://github.com/mre/awesome-static-analysis

☑ Integrate security scanners in your CI pipeline

Integrate a Dynamic Application Security Testing (DAST) tool in your CI, but just like SAST be aware of the high number of false positives.

Read more:
http://www.arachni-scanner.com/
https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
https://www.acunetix.com/vulnerability-scanner/

☑ Keep your dependencies up to date

Applications are built using dozens of third party libraries. A single flaw in any of these libraries may put your entire application at risk. According to OWASP, one of the most common application security risks is using dependencies with known vulnerabilities. Some tools allow you to check your dependencies for vulnerabilities and ensure that they are up-to-date:

https://docs.npmjs.com/cli/audit

https://snyk.io/
https://www.sqreen.com/

☑ Enforce a secure code review checklist

Security should always be kept in mind while coding. Pull request reviews should be performed with security in mind as well. Depending on where the code is, the checks should be different. Dealing with user entry is one thing, dealing with business structures is another – the concerns are related to the context.

In addition to common sense, keep in mind typical security flaws. For example, many code snippets from places like StackOverflow have not been written with security in mind. If your team pulls code snippets from the Internet, make sure they double check them for security before deploying them.

Security competency is also a good topic to ask about when interviewing a candidate.

Read more:
https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project

☑ Keep secrets away from code

Never commit secrets in your code. They should be handled and stored separately in order to prevent them from accidentally being shared or exposed. This keeps a clear layer of separation between your environments (typically development, staging, and production).

Pass secrets to the application through environment variables or through a configuration file.
You can use a configuration management module for this:
https://www.npmjs.com/package/config

Read more:
https://www.infosecurity-magazine.com/opinions/comment-tips-for-private-key-management/
https://www.digitalocean.com/community/tutorials/an-introduction-to-managing-secrets-safely-with-version-control-systems
https://www.vaultproject.io/

☑ Use a secure development life cycle

The secure development lifecycle is a process that helps tackle security issues at the beginning of a project. While rarely used as is, it provides good insights at all stages of the project, from the specification to the release. It will allow you to enforce good practices at every stage of the project life.

Read more:
https://en.wikipedia.org/wiki/Systems_development_life_cycle
https://www.owasp.org/images/7/76/Jim_Manico_(Hamburg)_-_Securiing_the_SDLC.pdf

☑ Avoid merging methods with un-sanitized with user data

Using user data, such as HTTP body, with a merging method can lead to prototype pollution. Before doing anything with user data, one must ensure there was no value injected in `__proto__`.

Read more:
https://github.com/HoLyVieR/prototype-pollution-nsec18/blob/master/paper/JavaScript_prototype_pollution_attack_in_NodeJS.pdf
https://hackerone.com/reports/310443

Scan your project for dangerous regular expressions

Some regular expressions can be prone to catastrophic backtracking. In a Node.js context, this usually can lead to a Denial of Service. Ideally, one should avoid writing complex regular expressions themselves.

Read more:
https://www.regular-expressions.info/catastrophic.html
https://github.com/davisjam/vuln-regex-detector

☑ Evaluate the impact of introducing a new dependency

When adding a dependency to a project, you also add all the modules upon which this dependency relies. If you're not conscientious, doing so could introduce outdated or malicious packages.

Read more:
A tool to view the real impact of adding a new dependency: https://npm.anvaka.com/#/

INFRASTRUCTURE

☑ Automatically configure & update your servers

An automated configuration management tool helps you ensure that your servers are updated and secured.

Read more:
Chef: https://learn.chef.io/tutorials/
Puppet: https://www.linode.com/docs/applications/configuration-management/getting-started-with-puppet-6-1-basic-installation-and-setup/
Ansible: http://docs.ansible.com/ansible/intro_getting_started.html
Salt: https://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html

☑ Backup regularly, test your backups, then backup again

Backup all your critical assets. Ensure that you attempt to restore your backups frequently so you can guarantee that they're working as intended. S3 is a very cheap and effective way to backup your assets.

Read more:
MongoDB Backup: https://docs.mongodb.com/manual/core/backups/
Postgresql: https://www.postgresql.org/docs/current/static/backup.html
Linux: http://www.tecmint.com/linux-system-backup-tools/
https://www.dataone.org/best-practices/ensure-integrity-and-accessibility-when-making-backups-data
https://aws.amazon.com/getting-started/backup-files-to-amazon-s3/

☑ Check your SSL / TLS configurations

Use free tools to scan your infrastructure regularly and make sure the SSL configurations are correct.

Read more:
https://observatory.mozilla.org/
https://www.ssllabs.com/

☑ Control access on your cloud providers

The best way to protect your services (database, file storage) is to not use passwords at all. Use the built-in Identity and Access Management (IAM) functions to securely control access to your resources.

Read more:
http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html
https://cloud.google.com/compute/docs/access/create-enable-service-accounts-for-instances

☑ Run it unprivileged

In the case that an attacker does successfully attack your application, having it running as a user with restricted privileges will make it harder for the attacker to take over the host and/or to bounce to other services. Privileged users are root on Unix systems, and Administrator or System on Windows systems.

☑ Log all the things, and centralize them

Infrastructure logs and application logs are some of your most precious allies for investigating a data breach. Make sure your logs are stored somewhere safe and central. Also make sure you whitelist or blacklist specific incoming data to avoid storing personally identifiable information (PII) data.

Don’t forget, you need to take care that the system time configured on each of your machines is in sync so that you can easily cross-correlate logs. You’ll have a much harder time if they’re not.

Read more:
https://qbox.io/blog/welcome-to-the-elk-stack-elasticsearch-logstash-kibana
https://www.loggly.com/
https://en.wikipedia.org/wiki/Network_Time_Protocol

☑ Manage secrets with dedicated tools and vaults

When you need to store cryptographic secrets (other than database password, TLS certificate, etc.) and perform encryption with them, you should use dedicated tools. This way the cryptographic secret never leaves the tool and you get auditing features.

Read more:
https://www.vaultproject.io/
https://github.com/square/keywhiz
https://aws.amazon.com/cloudhsm/
https://aws.amazon.com/kms/

☑ Store encrypted passwords in your configuration management

Storing passwords (like for your database) can be done on a dedicated database with restricted access. The other solution is to store them encrypted in your Source Code Management (SCM) system. That way, you just need the master key to decrypt them.

Read more:

Chef: https://github.com/chef/chef-vault
Puppet: https://puppet.com/blog/encrypt-your-data-using-hiera-eyaml
Salt: https://docs.saltstack.com/en/latest/ref/renderers/all/salt.renderers.gpg.html
Ansible: http://docs.ansible.com/ansible/playbooks_vault.html

☑ Upgrade your servers regularly

Server packages and libraries are often updated when security vulnerabilities are found. You should update them as soon as a security vulnerability is found.

Read more:
https://www.ubuntu.com/usn/
https://help.ubuntu.com/community/AutomaticSecurityUpdates
https://access.redhat.com/security/vulnerabilities

☑ Encrypt all the things

SSL performance problems are a myth and you have no good reason not to use SSL on all your public services. Encrypting communications is not only about privacy, but also about your users' safety, since it will prevent most attempts at tampering with what they receive.

Read more:
https://letsencrypt.org/
https://certbot.eff.org/
https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-14-04
https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-14-04

☑ Use an immutable infrastructure

Use immutable infrastructure to avoid having to manage and update your servers.

Read more:
https://martinfowler.com/bliki/ImmutableServer.html
https://hackernoon.com/configuration-management-is-an-antipattern-e677e34be64c#.n68b1i3eo
https://www.digitalocean.com/community/tutorials/what-is-immutable-infrastructure

---

☑ Renew your certificates on time

You should be using TLS certificates. It can be a hassle to configure and monitor, but don't forget to renew them!

Read more:
https://www.ssllabs.com/
https://serverlesscode.com/post/ssl-expiration-alerts-with-lambda/

☑ Monitor your authorizations

Be proactive and get alerted when authorizations or keys binary are changed in production.

Read more:
http://techblog.netflix.com/2017/03/netflix-security-monkey-on-google- cloud.html
https://cloudsploit.com/events
https://www.ossec.net/
https://security.stackexchange.com/a/19386

☑ Monitor your DNS expiration date

Just like TLS certificates, DNS can expire. Make sure you monitor your DNS expiration automatically.

Read more:
https://github.com/glensc/monitoring-plugin-check_domain

☑ Keep Node.js up to date and only use LTS versions in production

Node.js has a public support planning. Non-LTS versions (those with odd major version numbers) are supported only for a few months. LTS versions are supported for three and a half years.

Read more:
https://github.com/nodejs/Release

PROTECTION

☑ Protect your applications against breaches

Detect and block attacks in real time using an application security management solution, or a suite of protection layers. At least all the OWASP Top 10 vulnerabilities (SQL injections, NoSQL injections, cross-site scripting attacks, code/ command injections, etc.) should be covered.

Read more:
https://www.sqreen.com/
https://www.sqreen.com/web-application-security/what-is-rasp
https://en.wikipedia.org/wiki/Web_application_firewall

☑ Enforce Two-factor authentication (2FA)

Enforce 2FA on all the services you use (whenever possible). Internally, your company should all use two-factor authentication. By adding 2FA, you add an extra layer of security. Should someone's password get stolen, the attacker would still be locked out unless they have access to the second factor (e.g. phone app or text) as well. Phones are the most commonly used device for second factors, and thus have to be secured accordingly (e.g. with codes or biometry). Another option is to use purpose-built hardware-based 2FA, like Yubikeys.

As you get higher profile customers, you will be required to implement stronger security practices. This includes offering them 2FA, role-based account management, SSO, etc. as well. Often times, these features are entry level requirements for more enterprise deals.

Read more:
https://duo.com/

https://auth0.com/
https://www.yubico.com/
https://nakedsecurity.sophos.com/2016/08/18/nists-new-password-rules-what-you-need-to-know/

☑ Have a public bug bounty program

A bug bounty program will allow external hackers to report vulnerabilities. Most of the bug bounty programs set rewards in place. You need security-aware people inside your development teams to evaluate any reports you receive, so make sure that you have the right internal resources before you set up such a program.

Read more:
https://www.tripwire.com/state-of-security/vulnerability-management/launching-an-efficient-and-cost-effective-bug-bounty-program/
https://www.hackerone.com/
https://www.bugcrowd.com/
https://cobalt.io

☑ Have a public security policy

This is a page on your corporate website describing how you secure your users and their data, and how you plan to respond to external bug reports. You should advise that you support responsible disclosure. Keep in mind that you will likely receive reports of varying impact, so having a process for prioritizing them is important.

Read more:
https://www.sqreen.com/resources/security-page
https://www.airbnb.com/security
https://www.apple.com/support/security/

☑ Protect against Distributed Denial Of Service (DDoS)

DDoS attacks are meant to break your application and make it unavailable to your customers. Basic DDoS protections can easily be integrated with a CDN, but there are purpose-built DDoS protection tools available as well.

Read more:
https://www.akamai.com/
https://www.cloudflare.com/ddos/
https://www.techradar.com/news/best-ddos-protection

☑ Protect your servers and infrastructure from scanners

Your servers will be scanned in order to fingerprint your application and locate open services, misconfiguration, etc. You can set up tools to keep these scanners away from your servers.

Read more:
https://www.sqreen.com/
https://www.digitalocean.com/community/tutorials/how-to-protect-ssh-with-fail2ban-on-ubuntu-14-04
https://docs.microsoft.com/en-us/azure/information-protection/deploy-aip-scanner

☑ Protect your users against account takeovers

Account takeovers or brute-force attacks are easy to set up. You should make sure your users are protected against account takeovers.

Read more:
https://www.sqreen.com/
https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks
https://security.stackexchange.com/questions/94432/should-i-implement- in-

correct-password-delay-in-a-website-or-a-webservice
https://blog.sqreen.com/most-common-types-of-ato-attacks/

☑ Keep your containers secure

If you use Docker (or Kubernetes), ensure that they are patched and secure. Use tools to automatically update and scan your containers for security vulnerabilities. If you use a PAAS provider (Heroku, AWS Beanstalk, etc...), they will take care of this for you. If not, you will need to do it yourself. Ideally, automate this process if possible.

Read more:
https://blog.sqreen.com/docker-security/
https://blog.sqreen.com/kubernetes-security-best-practices/
https://docs.docker.com/docker-cloud/builds/image-scan/

☑ Don't store credit card information (if you don't need to)

Use third-party services to store credit card information to avoid having to manage and protect them.

Read more:
https://stripe.com/
https://www.braintreepayments.com
https://www.pcisecuritystandards.org/pdfs/pciscc_ten_common_myths.pdf

---

☑ Ensure compliance with relevant industry standards

Comply with standards to ensure that you follow industry best practices and answer your customer needs. But simple compliance alone will not be enough to protect your apps.

Read more:

https://cloudsecurityalliance.org/

https://en.wikipedia.org/wiki/ISO/IEC_27001:2013

https://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard

MONITORING

☑ Get notified when your app is under attack

Attacks happen, and a percentage of those will actually hit a vulnerability. Make sure you have a monitoring system in place that will detect security events targeting your application before it's too late. Knowing when your application is starting to get massively scanned is key to stop more advanced attacks.

Read more:
https://www.sqreen.com/
https://www.linode.com/docs/security/using-fail2ban-for-security#email-alerts
http://alerta.io/

☑ Audit your infrastructure on a regular basis

With cloud providers, it's easy to start instances and forget about them. You will need to create and maintain a list of your assets (servers, network devices, services exposed, etc.), and review it regularly to determine if you still need them, keep them up to date, and ensure that they benefit from your latest deployments.

Read more:
http://docs.aws.amazon.com/general/latest/gr/aws-security-audit-guide.html
https://searchsecurity.techtarget.com/IT-security-auditing-Best-practices-for-conducting-audits
https://cloud.google.com/asset-inventory/docs/overview

☑ Detect attackers early

The most impactful attacks will come from attackers that have acquired larger attack surfaces. Those can be attackers with regular user accounts or users that have gained access to privileged user accounts. Make sure you monitor your users for suspicious behavior to detect attackers early.

Read more:
https://www.sqreen.com/

☑ Monitor your third-party vendors

You're likely to use third-party products to manage your servers / payrolls / logs or even just social media. Third-party vendors are susceptible to breaches just like everyone else. Make sure you follow the news and react immediately after a breach.

Read more:
https://haveibeenpwned.com/
https://twitter.com/SecurityNewsbot

NOTES

Want this handbook as a PDF?
Scan the QR-code, or go to:
https://www.sqreen.com/checklists/nodejs-security-handbook