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. 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.io

WANT THIS HANDBOOK AS A PDF? GO TO:
https://www.sqreen.io/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 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

Use publish-please package:
https://www.npmjs.com/package/publish-please
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 get into your application should be validated and escaped to avoid various kinds of injections.

Learn more about MongoDB injections:

https://blog.sqreen.io/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.

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.
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

Modern browsers support a set of headers dedicated to blocking certain types of attacks. Make sure you have properly implemented all security headers. Don't forget about the Content Security Policy.

https://www.npmjs.com/package/helmet

https://myheaders.sqreen.io/

https://securityheaders.io/

https://blog.appcanary.com/2017/http-security-headers.html

✔ 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 Burp.

https://portswigger.net/burp/

http://www.devsecops.org/blog/2015/12/10/red-team-pwning-the-hearts-and-minds-one-ticket-at-a-time

---

✔ Run security linters on your code

Static Application Security Testing (SAST) is an easy and fast way to find unsafe patterns in your code. You can enforce SAST security checks with a pre or post-commit hook, but be aware of the high number of false positives.

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.

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

Third-party libraries can put your application at risk. Make sure you track your vulnerable packages and update them regularly.

https://snyk.io/

https://www.sqreen.io/

https://nodesource.com/products/certified-modules

✔ Enforce a secure code review checklist

Security should always be kept in mind while coding. Enforce security reviews for pull requests.

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 separately in order to prevent them accidentally being shared or exposed. This allows a clear separation between your environments (typically development, staging and production).

https://12factor.net/

Use a configuration file/en variable
Use a configuration management module:
https://www.npmjs.com/package/config

✔ 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.

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

INFRASTRUCTURE

✔ Automatically configure & update your servers

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

Chef: https://learn.chef.io/tutorials/

Puppet: https://www.digitalocean.com/community/tutorials/how-to-install-puppet-4-in-a-master-agent-setup-on-ubuntu-14-04

Ansible: http://docs.ansible.com/ansible/intro_getting_started.html

Salt: https://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html

✔ Backup regularly

Your data is likely to be your business's most precious asset. Be sure not to lose it. Implement proper backups and check for backup integrity.

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

✔ Check your SSL / TLS configurations

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

https://observatory.mozilla.org/

https://www.ssllabs.com/

https://diogomonica.com/2015/12/29/from-double-f-to-double-a/

✔ 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.

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 case an attacker successfully attacks 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

Infrastructure logs and application logs are 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.

https://qbox.io/blog/welcome-to-the-elk-stack-elasticsearch-logstash-kibana

https://www.loggly.com/

✔ 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.

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.

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.

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.

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.

https://martinfowler.com/bliki/ImmutableServer.html

https://hackernoon.com/configuration-management-is-an-antipattern-e677e34be64c#.n68b1i3eo

---

✔ 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!

https://www.ssllabs.com/

https://serverlesscode.com/post/ssl-expiration-alerts-with-lambda/

✔ Monitor your authorizations

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

http://techblog.netflix.com/2017/03/netflix-security-monkey-on-google-cloud.html

https://cloudsploit.com/events

http://ossec.github.io/

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.

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

PROTECTION

✔ Protect your applications against breaches

Detect and block attacks in real-time using a protection solution. At least all the OWASP top-10 vulnerabilities (SQL injections, NoSQL injections, cross-site scripting attacks, code/command injections, etc.) should be covered.

https://www.sqreen.io/

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

✔ Enforce Two-factor authentication (2FA)

Enforce 2FA on all the services used (whenever possible).

https://duo.com/

https://auth0.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 bounties program set rewards in place. You need security-aware people inside your development teams to evaluate the reports you receive.

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/

✔ Have a public security policy

This is a page on your corporate website describing how you plan to respond to external security reports. This page should say that you support responsible disclosure. Keep in mind that most of the reports that you receive probably won't be relevant.

https://www.intercom.com/security

https://www.zendesk.com/product/zendesk-security/

https://www.apple.com/support/security/

✔ Protect against Denial Of Service (DoS)

DoS attacks are meant to break your application and make it unavailable to your customers. Use a specific service to protect your app against Denial of Service attacks.

https://www.akamai.com/

https://www.cloudflare.com/ddos/

https://www.ovh.com/us/news/articles/a1171.protection-anti-ddos-service-standard

✔ 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 setup tools to keep these scanners away from your servers.

https://www.sqreen.io/

https://www.digitalocean.com/community/tutorials/how-to-protect-ssh-with-fail2ban-on-ubuntu-14-04

✔ 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.

https://www.sqreen.io/

https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks

https://security.stackexchange.com/questions/94432/should-i-implement-incorrect-password-delay-in-a-website-or-a-webservice

✔ Keep your containers secure

Use Docker (or Kubernetes), and ensure that they are patched and secure. Use tools to automatically update and scan your containers for security vulnerabilities.

https://blog.sqreen.io/docker-security/

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.

https://stripe.com/

https://www.braintreepayments.com

https://www.pcisecuritystandards.org/pdfs/pciscc_ten_common_myths.pdf

https://medium.com/@folsen/accepting-payments-is-getting-harder-1b2f342e4ea#.897akko4q

✔ Ensure Compliance with Relevant Industry Standards

Comply with standards to ensure you follow industry best practices and answer your customer needs. But simple compliance will never protect your apps.

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

You will be attacked. 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.

https://www.sqreen.io/

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.

http://docs.aws.amazon.com/general/latest/gr/aws-security-audit-guide.html

http://searchenterpriselinux.techtarget.com/tip/Creating-an-inventory-with-nmap-network-scanning

https://github.com/Netflix/security_monkey

✔ Detect attackers early

The most important attacks will come from attackers who have acquired larger attack surfaces. Those can be attackers with regular user accounts or users having

gained access to privileged user accounts. Make sure you monitor your users to detect attackers early.

https://www.sqreen.io/

✔ Monitor third party vendors

You're likely to use third-party products to manage your servers / payrolls / logs or even just social media. Just like you're likely to be hacked, they can be too. Make sure you follow the news and react immediately after a breach.

https://haveibeenpwned.com/

https://twitter.com/SecurityNewsbot

www.sqreen.io