Note: JavaScript is an ecosystem and NodeJS is one part of that ecosystem.

The article I am using to seed my talk ( ).

Their GitHub site with a list of security best practices ( <https://github.com/i0natan/nodebestpractices/tree/security-best-practices-section/sections/security> ).

Security is a philosophy you follow or you don’t. Security through obfuscation doesn’t count as security, (for example having a steel door that bolts doesn’t work if you don’t lock it. Just having the steel door will keep an honest person honest. Thieves will try the lock and open your door.). Proper security takes extra time.

The Node.js community relies on you to keep it safe.

Sort these exploits by category, according to OWASP

**SLIDES IN ORDER**

1. OWASP
2. npm
3. Linters
4. Security Patches
5. SemVer
6. OWASP Top 10
7. 1 through 10-ish
8. Threats in the Wild
9. NodeGoat
10. Node.js Vulnerability Disclosure, “See something, Say something”
11. Acknowledgements

**SECURITY AS A PHILOSOPHY**

* Point out OWASP, and they have a meetup in our region also
  + The “Open Web Application Security” (OWASP)
  + website ( <https://www.owasp.org/> )
  + meetup in SF Bay Area ( <https://www.meetup.com/Bay-Area-OWASP/> )
  + on Twitter too ( <https://twitter.com/owasp> )
* npm has security features everyone should know about
  + Security Audits in npm. the docs are here ( <https://docs.npmjs.com/getting-started/running-a-security-audit> )
  + 2fa on packages. the docs are here ( <https://docs.npmjs.com/getting-started/using-two-factor-authentication> ).
    - use One Time Password Generators (Authy, Google Authenticator)
    - recover OTP and 2fa
  + also cover how to backup/recover 2fa
* Configure 2FA for npm and yarn (any package manager)
  + research how to recover 2fa from npm
* Linter security rules
  + source ( <https://github.com/i0natan/nodebestpractices/blob/master/sections/security/lintrules.md> )
  + we have tools let’s use them
  + TSLint ( <https://www.npmjs.com/package/tslint-config-security> ) read list of issues
  + ESLint ( <https://github.com/nodesecurity/eslint-plugin-security> )
* Security Patches
  + Bugs/issues/security flaws
  + Long-Term Support (LTS) will support you with patches if a security flaw is discovered.
    - ( <https://github.com/nodejs/Release> )
    - even number releases are LTS. Which is about 3 years from Current to EOL.
    - Bug fixes, security updates, non-semver-major npm updates,
    - Node 6.x EOL scheduled for April 2019
    - Node 10.x EOL scheduled for April 2021
    - Node 11.x EOL scheduled for June 2019
  + do not use End of Life (EOL) versions because there is no support even if there is a known security vulnerability.
    - note odd number releases will be EOL once the next “SemVer” realeases
  + bump dependency versions
* Semantic Versioning ( SemVer )
  + ( <https://semver.org/> )
  + Given a version number MAJOR.MINOR.PATCH, increment the:
  + MAJOR version when you make incompatible API changes,
  + MINOR version when you add functionality in a backwards-compatible manner, and
  + PATCH version when you make backwards-compatible bug fixes.
  + Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.
* Node.js Vulnerability Disclosure
  + “If you See something Say something”
  + report issues by email ( [security@nodejs.org](mailto:security@nodejs.org) ), or here at hackerone ( <https://hackerone.com/nodejs> ). main bug page for nodejs ( <https://nodejs.org/en/security/> )
  + There are bug bounties too, just another incentive to do the right thing
  + security.md file in each npm package
    - proposed standard
    - guidelines ( <https://github.com/securitytxt/security-txt> )
    - security disclosure policy for package
* Node Security Roadmap from Google ( <https://nodesecroadmap.fyi/> )
* Constantly and automatically inspect for vulnerable dependencies
  + use things like npm audit, nsp, or snyk
* NodeGoat
  + ( <https://github.com/OWASP/NodeGoat> )
  + OWASP initiative
  + app that can be exploited. They also guide devs in ways to fix the issues/vulnerabilities.
* use `npm-check` to see if modules are out of date.
  + ( <https://www.npmjs.com/package/npm-check> )
  + Looks a little different than npm-audit
* ~~scan your app for easy to find issues~~
  + ~~NodeJsScan (~~ [~~https://github.com/ajinabraham/NodeJsScan~~](https://github.com/ajinabraham/NodeJsScan) ~~)~~
  + ~~mainly regex style attacks~~

**THREATS**

* A1: Injection
  + **Solution**: make sure to validate the data somehow.
  + Prevent query injection vulnerabilities
    - ORM/ODM libraries like Mongoose have this feature
    - “Node.js Applicative DoS Through MongoDB Injection”   
      by Vladimir de Turckheim,   
      on YouTube ( <https://youtu.be/xJWZsoYmsIE> )
  + Avoid JavaScript `eval` statements and `new Function`
  + Avoid module loading `require(someVariable)` using a variable
* A2: Broken Authentication
  + Avoid using the Node.js crypto library for handling passwords, use Bcrypt
    - use bcrypt ( <https://www.npmjs.com/package/bcrypt> )
    - I am curious what is wrong with the crypto library
    - Thomas Hunter II has written articles about this too ( <https://medium.com/intrinsic/common-node-js-attack-vectors-the-dangers-of-malicious-modules-863ae949e7e8> )
  + Support blacklisting JWT tokens
    - JWTs Suck (and are stupid),   
      by Randall Degges,   
      slides here ( <https://speakerdeck.com/rdegges/jwts-suck-and-are-stupid> ),  
      on YouTube ( <https://youtu.be/JdGOb7AxUo0> )
  + Limit the allowed login request of each user
    - use express-brute ( <https://www.npmjs.com/package/express-brute> )
* A3: Sensitive Data Exposure
  + Extract secrets from config files or use packages to encrypt them
    - use environment variables for this
    - use `cryptr` ( <https://www.npmjs.com/package/cryptr> )
* A4: External Entities
  + Run unsafe code in a sandbox
    - use a dedicated child process
    - use a cloud serverless framework
    - use libraries like sandbox or vm2
      * <https://www.npmjs.com/package/sandbox>
      * <https://www.npmjs.com/package/vm2>
  + Take extra care when working with child processes
    - use `child\_process.execFile` if you are unsure but need to use it
* A5: Broken Access Control
  + Run Node.js as non-root user
* A6: Security Misconfiguration
  + Adjust the HTTP response headers for enhanced security
    - use helmet for express servers ( <https://www.npmjs.com/package/helmet> )
  + Hide error details from clients
    - set `NODE\_ENV` to production
  + Modify session middleware settings
* A7: Cross-Site Scripting (XSS)
  + Escape HTML, JS, and CSS output
    - use escape-html ( <https://github.com/component/escape-html> )
    - use node-esapi ( <https://github.com/ESAPI/node-esapi> )
* A8: Insecure Deserialization
  + Validate incoming JSON schemas
    - use jsonschema ( <https://www.npmjs.com/package/jsonschema> )
    - use joi ( <https://www.npmjs.com/package/joi> )
  + Limit payload size using a reverse proxy or middleware.
    - configure express body parser to accept only small-size payloads
* A9: Using Components with Known Vulnerabilities
  + npm audit fix
* A10: Insufficient Logging and Monitoring
  + use due diligence
* DDOS
  + Limit concurrent requests using a middleware
    - cloud load balancers, firewalls
    - `express-rate-limit` ( <https://www.npmjs.com/package/express-rate-limit> )
  + Avoid DOS attacks by explicitly setting when a process should crash
  + Prevent RegEx from overloading your single thread execution
* User input is a major vulnerability, please treat it like hostile code and sanitize it. Filter and validate user input.
  + gif from it crowd
* ? node-html-entities
  + ( <https://www.npmjs.com/package/html-entities> )
  + not quite sure what this is but it has 2M downloads/week

**THREATS IN THE WILD**

* Electron exploit
  + see if it is still an issue. Fixed in Electron version 2
  + <https://github.com/electron/electron/blob/master/docs/tutorial/security.md>
  + THIS IS WHAT GAVE ME THE IDEA FOR THE TALK
* CVE Details – a website I ran across during my research of Node.Js vulnerabilities
  + ( <https://www.cvedetails.com/vulnerability-list/vendor_id-12113/Nodejs.html> )
* Reverse Shell
  + patched and specific but interesting article on it here ( <https://wiremask.eu/writeups/reverse-shell-on-a-nodejs-application/> )

**CREDITS**

* “i0natan” - Their GitHub site with a list of security best practices ( <https://github.com/i0natan/nodebestpractices/tree/security-best-practices-section/sections/security> ).
* Thomas Hunter II - has written articles about crypto ( <https://medium.com/intrinsic/common-node-js-attack-vectors-the-dangers-of-malicious-modules-863ae949e7e8> )