Information Security Notes

Patrick Adams

November 1, 2020

Note: This is a draft copy of notes generated by free code camp.
https://www.freecodecamp.org/

Contents

1 Information Security With Helmetjs

1.1 Install and Require Helmet

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub. Helmet helps you secure your Express apps by setting various HTTP headers.

1.2 Hide Potentially Dangerous Information Using helmet.hidePoweredBy()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub. Hackers can exploit known vulnerabilities in Express/Node if they see that your site is powered by Express. X-Powered-By: Express is sent in every request coming from Express by default. The helmet.hidePoweredBy() middleware will remove the X-Powered-By header. You can also explicitly set the header to something else, to throw people off. e.g. app.use(helmet.hidePoweredBy({ setTo: 'PHP 4.2.0' }))

1.3 Mitigate the Risk of Clickjacking with helmet.frameguard()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
Your page could be put in a <frame> or <iframe> without your consent. This can result in clickjacking attacks, among other things. Clickjacking is a technique of tricking a user into interacting with a page different from what the user thinks it is. This can be obtained executing your page in a malicious context, by mean of iframing. In that context a hacker can put a hidden layer over your page. Hidden buttons can be used to run bad scripts. This middleware sets the X-Frame-Options header. It restricts who can put your site in a frame. It has three modes: DENY, SAMEORIGIN, and ALLOW-FROM.
We don't need our app to be framed.

1.4 Mitigate the Risk of Cross Site Scripting (XSS) Attacks with helmet.xssFilter()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
Cross-site scripting (XSS) is a frequent type of attack where malicious scripts are injected into vulnerable pages, with the purpose of stealing sensitive data like session cookies, or passwords.
The basic rule to lower the risk of an XSS attack is simple: "Never trust user's input". As a developer you should always sanitize all the input coming from the outside. This includes data coming from forms, GET query urls, and even from POST bodies. Sanitizing means that you should find and encode the characters that may be dangerous e.g. <, >.
Modern browsers can help mitigating the risk by adopting better software strategies. Often these are configurable via http headers.
The X-XSS-Protection HTTP header is a basic protection. The browser detects a potential injected script using a heuristic filter. If the header is enabled, the browser changes the script code, neutralizing it.
It still has limited support.

1.5 Avoid Inferring the Response MIME Type with helmet.noSniff()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
Browsers can use content or MIME sniffing to adapt to different datatypes coming from a response. They override the Content-Type headers to guess and process the data. While this can be convenient in some scenarios, it can also lead to some dangerous attacks. This middleware sets the X-Content-Type-Options header to nosniff. This instructs the browser to not bypass the provided Content-Type.

1.6 Prevent IE from Opening Untrusted HTML with helmet.ieNoOpen()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
Some web applications will serve untrusted HTML for download. Some versions of Internet Explorer by default open those HTML files in the context of your site. This means that an untrusted HTML page could start doing bad things in the context of your pages. This middleware sets the X-Download-Options header to noopen. This will prevent IE users from executing downloads in the trusted site's context.

1.7 Ask Browsers to Access Your Site via HTTPS Only with helmet.hsts()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
HTTP Strict Transport Security (HSTS) is a web security policy which helps to protect websites against protocol downgrade attacks and cookie hijacking. If your website can be accessed via HTTPS you can ask user's browsers to avoid using insecure HTTP. By setting the header Strict-Transport-Security, you tell the browsers to use HTTPS for the future requests in a specified amount of time. This will work for the requests

coming after the initial request.

1.8 Disable DNS Prefetching with helmet.dnsPrefetchControl()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
To improve performance, most browsers prefetch DNS records for the links in a page. In that way the destination ip is already known when the user clicks on a link. This may lead to over-use of the DNS service (if you own a big website, visited by millions people...), privacy issues (one eavesdropper could infer that you are on a certain page), or page statistics alteration (some links may appear visited even if they are not). If you have high security needs you can disable DNS prefetching, at the cost of a performance penalty.

1.9 Disable Client-Side Caching with helmet.noCache()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
If you are releasing an update for your website, and you want the users to always download the newer version, you can (try to) disable caching on client's browser. It can be useful in development too. Caching has performance benefits, which you will lose, so only use this option when there is a real need.

1.10 Set a Content Security Policy with helmet.contentSecurityPolicy()

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
This challenge highlights one promising new defense that can significantly reduce the risk and impact of many type of attacks in modern browsers. By setting and configuring a Content Security Policy you can prevent the injection of anything unintended into your page. This will protect your app from XSS vulnerabilities, undesired tracking, malicious frames, and much more. CSP works by defining an allowed list of content sources which are trusted. You can configure them for each kind of resource a web page may need (scripts, stylesheets, fonts, frames, media, and so on...). There are multiple directives available, so a website owner can have a granular control. See HTML 5 Rocks, KeyCDN for more details. Unfortunately CSP is unsupported by older browser.
By default, directives are wide open, so it's important to set the defaultSrc directive as a fallback. Helmet supports both defaultSrc and default-src naming styles. The fallback applies for most of the unspecified directives.

1.11 Configure Helmet Using the 'parent' helmet() Middleware

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
app.use(helmet()) will automatically include all the middleware introduced above, except noCache(), and contentSecurityPolicy(), but these can be enabled if necessary. You can also disable or configure any other middleware individually, using a configuration object.
Example:
"'js
app.use(helmet({
frameguard: { // configure
action: 'deny'
},
contentSecurityPolicy: { // enable and configure
directives: {
defaultSrc: ["self"],
styleSrc: ['style.com'],
}

},
dnsPrefetchControl: false // disable
}))
‘‘‘

We introduced each middleware separately for teaching purposes and for ease of testing. Using the ‘parent’ helmet() middleware is easy to implement in a real project.

1.12 Understand BCrypt Hashes

For the following challenges, you will be working with a new starter project that is different from the previous one. You can find the new starter project on Repl.it, or clone it from GitHub.
BCrypt hashes are very secure. A hash is basically a fingerprint of the original data- always unique. This is accomplished by feeding the original data into an algorithm and returning a fixed length result. To further complicate this process and make it more secure, you can also salt your hash. Salting your hash involves adding random data to the original data before the hashing process which makes it even harder to crack the hash.
BCrypt hashes will always looks like $2a$13$ZyprE5MRw2Q3WpNOGZWGbeG7ADUre1Q8QO.uUUtcbqloU0yvzavOm which does have a structure. The first small bit of data $2a is defining what kind of hash algorithm was used. The next portion $13 defines the cost. Cost is about how much power it takes to compute the hash. It is on a logarithmic scale of 2^cost and determines how many times the data is put through the hashing algorithm. For example, at a cost of 10 you are able to hash 10 passwords a second on an average computer, however at a cost of 15 it takes 3 seconds per hash... and to take it further, at a cost of 31 it would takes multiple days to complete a hash. A cost of 12 is considered very secure at this time. The last portion of your hash $ZyprE5MRw2Q3WpNOGZWGbeG7ADUre1Q8QO.uUUtcbqloU0yvzavOm, looks like one large string of numbers, periods, and letters but it is actually two separate pieces of information. The first 22 characters is the salt in plain text, and the rest is the hashed password!

1.13 Hash and Compare Passwords Asynchronously

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
As hashing is designed to be computationally intensive, it is recommended to do so asynchronously on your server as to avoid blocking incoming connections while you hash. All you have to do to hash a password asynchronous is call
‘‘‘js
bcrypt.hash(myPlaintextPassword, saltRounds, (err, hash) => {
/*Store hash in your db*/
});
‘‘‘

1.14 Hash and Compare Passwords Synchronously

As a reminder, this project is being built upon the following starter project on Repl.it, or cloned from GitHub.
Hashing synchronously is just as easy to do but can cause lag if using it server side with a high cost or with hashing done very often. Hashing with this method is as easy as calling
‘‘‘js
var hash = bcrypt.hashSync(myPlaintextPassword, saltRounds);
‘‘‘

Add this method of hashing to your code and then log the result to the console. Again, the variables used are already defined in the server so you won’t need to adjust them. You may notice even though you are hashing the same password as in the async function, the result in the console is different- this is due to the salt being randomly generated each time as seen by the first 22 characters in the third string of the hash.
Now to compare a password input with the new sync hash, you would use the compareSync method:

“‘js
var result = bcrypt.compareSync(myPlaintextPassword, hash);
“‘
with the result being a boolean true or false.

2 Python For Penetration Testing

2.1 Introduction and Setup

2.2 Understanding Sockets and Creating a TCP Server

2.3 Creating a TCP Client

2.4 Developing an Nmap Scanner part 1

2.5 Developing an Nmap Scanner part 2

2.6 Developing a Banner Grabber

2.7 Developing a Port Scanner

3 Information Security Projects

3.1 Stock Price Checker

Build a full stack JavaScript app that is functionally similar to this: https://stock-price-checker.freecodecamp.rocks/. Since all reliable stock price APIs require an API key, we've built a workaround. Use https://stock-price-checker-proxy.freecodecamp.rocks/ to get up-to-date stock price information without needing to sign up for your own key.
Working on this project will involve you writing your code on Repl.it on our starter project. After completing this project you can copy your public Repl.it URL (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but must be publicly visible for our testing.
Start this project on Repl.it using this link or clone this repository on GitHub! If you use Repl.it, remember to save the link to your project somewhere safe!

3.2 Anonymous Message Board

Build a full stack JavaScript app that is functionally similar to this: https://anonymous-message-board.freecodecamp.rocks/.
Working on this project will involve you writing your code on Repl.it on our starter project. After completing this project you can copy your public Repl.it URL (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing.
Start this project on Repl.it using this link or clone this repository on GitHub! If you use Repl.it, remember to save the link to your project somewhere safe!

3.3 Port Scanner

Create a port scanner using Python.
You can access the full project description and starter code on Repl.it.
After going to that link, fork the project. Once you complete the project based on the instructions in 'README.md', submit your project link below.
We are still developing the interactive instructional part of the Python curriculum. For now, here are some videos on the freeCodeCamp.org YouTube channel that will teach you some of the Python skills required for this project:
Python for Everybody Video Course (14 hours)

Learn Python Video Course (2 hours)

3.4 SHA-1 Password Cracker

For this project you will learn about the importance of good security by creating a password cracker to figure out passwords that were hashed using SHA-1.
You can access the full project description and starter code on Repl.it.
After going to that link, fork the project. Once you complete the project based on the instructions in 'README.md', submit your project link below.
We are still developing the interactive instructional part of the Python curriculum. For now, here are some videos on the freeCodeCamp.org YouTube channel that will teach you some of the Python skills required for this project:
Python for Everybody Video Course (14 hours)

Learn Python Video Course (2 hours)

3.5 Secure Real Time Multiplayer Game

Develop a 2D real time multiplayer game using the HTML Canvas API and Socket.io that is functionally similar to this: https://secure-real-time-multiplayer-game.freecodecamp.rocks/.
Working on this project will involve you writing your code on Repl.it on our starter project. After completing this project you can copy your public Repl.it URL (to the homepage of your app) into this screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing.
Start this project on Repl.it using this link or clone this repository on GitHub! If you use Repl.it, remember to save the link to your project somewhere safe!