Cheat sheet: Django security tips

snyk

1. Know your version and use a secure one

What version of Django are you using? The choice of version determines what known vulnerabilities are present, and potentially exploitable, in your application. Learn more about known Django vulnerabilities from our vulnerability database or you scan your project with Snyk.

2. Throttle user authentications

Django provides a lot of security features baked in, but the authentication system does not inherently protect against brute force attacks. It is important for you to write your own code to prevent this, or use one of many open source solutions (like Django Defender).

3. Protect your source code

Make sure your source code is not included in your web server's root directory. Use a private repository if your project is sensitive. Never check your secrets into version control, even if you are using a private repository.

4. Use raw queries and custom SQL with caution

While it may be tempting to write raw sql queries and custom SQL, doing so may open the door for an attack. A user attempting to perform an sql injection (execute arbitrary sql on a database) is going to find it much harder, if you always use the ORM.

5. Use HTTPs

Regardless of your framework of choice, it is always preferable to deploy behind HTTPS. Doing so prevents malicious users from intercepting information sent between the client and the server.

6. Watch your headers

When the site is served via https, the referer request header is utilized by Django to help prevent cross site request forgery (CRSF) attacks. If you are too strict with your referer-policy header, you disable the functionality of Django's CRSF protection. In the end you need to weigh the privacy benefits of using a strict referer-policy header with the benefits of the CRSF protection.

7. Be careful with your cookies

Some cookies are more secure than others — the default cookie behavior is to connect over http. However, since we already established that you need to use https, you want to make sure your cookies are only being sent over https as well. To prevent leaking cookies, be sure to set your SESSION_COOKIE_SECURE and CSRF_COOKIE_SECURE settings to True.

8. Carefully handle user uploads

If your web application allows users to upload files, you are opening yourself to an attack vector and the upload logic should therefore be handled carefully. It is important to validate all uploaded files to be sure they are what you expect (for instance, an image file and not a PHP script!).

9. Understand all of your dependencies

Indirect dependencies are as likely to introduce risk as direct dependencies, but these risks are less likely to be recognized. A tool like Snyk helps you understand your entire dependency tree, and now that Snyk offers fix pull requests for Python, fixing problems (even in indirect dependencies) is easier than ever.

10. Don't let the perfect get in the way of the good

Every security step you take is a step in the right direction. Django may be for perfectionists with deadlines, but code doesn't have to be perfect to reap security benefits.