CS 3240 - F21 / Guided Practice I: Django Security Check

Instructions:

- Complete this guided practice either during lecture on Tuesday, November 23, or outside of class.
- **For this Guided Practice, you must work with members of your team! Your team can submit just one guided practice for all of you or can split your team into smaller groups if you want, but you MAY NOT work with anyone that is not a member of your project team!**
- You must submit a PDF of this GP into Gradescope by 11:59 PM, Tuesday, November 23.
- Make sure to select the pages with the answers to the questions!
- If you work in a group of two or three, one person should submit the GP to Gradescope and then use the "View or edit group" option on the right side of the screen to select the other members of your team.

Your Team ID:
A-21
Herokuapp URL of your project:
a-21-assignment-organizer.herokuapp.com
The goal of this Guided Practice is for your team to look at some security features of Django and run some automated security checks on your apps.

Step 1) Everyone should read the information at these two links: https://www.laurencegellert.com/2019/01/tips-and-tools-for-securing-django/ and https://docs.djangoproject.com/en/3.1/topics/security/

Step 2) Run these automated security checks on your app: https://djcheckup.com/, https://observatory.mozilla.org/, and also run `python manage.py check --deploy` from your command line with your project.

Step 3) Answer the following questions:

1) What are some steps that these tools told your team you should take to secure your app? How hard would it be to implement them?

DJ checkup gave our site a pretty good rating, only giving us one thumbs-down, saying that we do not have HSTS enabled. HSTS ensures that users can only communicate with our website with the HTTPS protocol. Interestingly, to fix an issue with Google Login a while ago, we added a setting to our settings.py that redirects users trying to access our site through HTTP to HTTPS, since Google Login did not allow HTTP URLs to sign in. So, we ended up implementing this feature in a non-standard way. That being said, enabling HSTS is a very simple procedure, since we already have the SecurityMiddleware in our settings. All we have to do is set a value for SECURE_HSTS_SECONDS (which I set to 1 year) in settings.py. Running DJ checkup again on the site after this change gave us a perfect score.

Mozilla observatory, on the other hand, gave us a slightly lower score. Along with DJ Checkup, it said we need to have HSTS implemented, but it also had a few other things that DJ checkup did not mention. After implementing HSTS, the score did go up, but it still said we were missing CSP implementation and X-XSS protection. Implementing this feature was, again, fairly simple, and running the test afterwards gave us a perfect score using both tools. However, in implementing CSP, a lot of our user interface broke, and there did not seem to be any easy workaround I could find. Bootstrap essentially became unusable for us. So, I disabled it, settling with a perfect score from DJ checkup and a B- from Mozilla observatory.

2) Find one simple fix that you can do and then re-run the tools. Did the results change? (If you cannot find a simple fix, explain why you cannot.)

Both tools recommended that we enable HSTS, which ensures that our site can only be accessed through HTTPS. This rule can protect a user's information, especially when they are using insecure, or public networks to communicate with our website. Enabling this feature was rather easy, and only required setting a variable SECURE_HSTS_SECONDS in our settings.py to 1 year. After enabling this, DJ Checkup gave us a perfect score, and the score from Mozilla observatory went up as well.

3) What is the main problem with using automated security checking tools? Describe your reasoning.

Automatic security analyzing tools are great, in that they are a quick and easy way to check for some basic security flaws in a website to be implemented. Creating tools to check for common security flaws for any website allows the entire web ecosystem to become more secure, since developers can easily check the security of their website before making it public, and web-users can check the security of websites before deciding to trust that website with their private information. However, these tools do come with a detriment. There are many ways for a security vulnerability to exist, and the cybersecurity climate changes drastically over time. Since these security checks can naturally only check for a few different kinds of vulnerabilities, there may be more niche, possibly even more devastating vulnerabilities that were not able to be checked by these tools. For instance, we used two tools to check

our website. If a website with more private information, such as credit card information, decided to only use DJ checkup, they might be told that their website was completely secure, when Mozilla Observer would have told them that there are more vulnerabilities they should fix. Because of the fact that there are many ways to have a vulnerability in a website, automated security analyzing tools are prone to false positives, ensuring developers that their website is secure when it may not be. This is why it is beneficial to hire a white-hat group to do vulnerability checking over a long period of time on your application. This is a better simulation of the real world, since a system with private information will likely be hacked by humans, trying many niche and creative ways to get into a system.