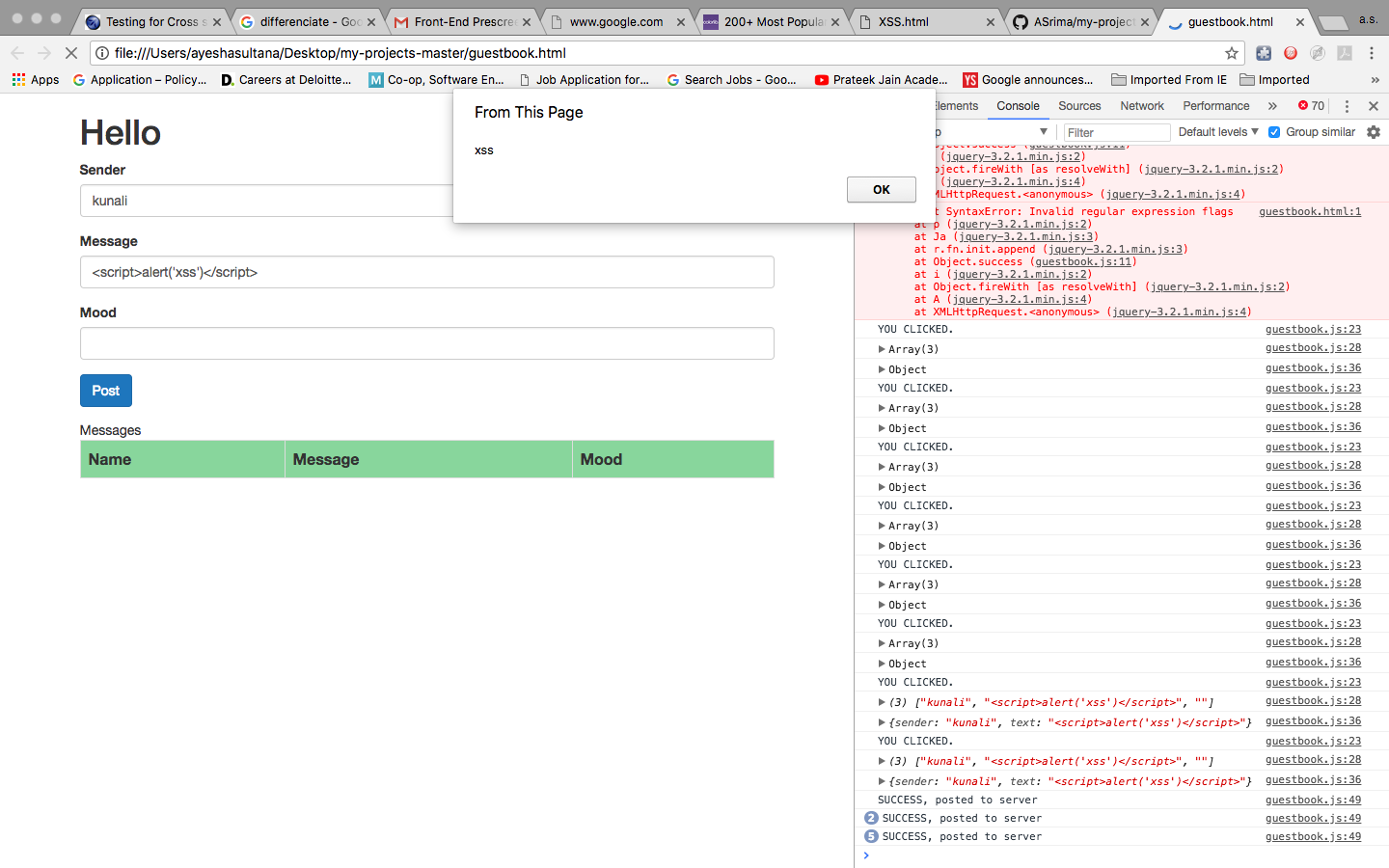
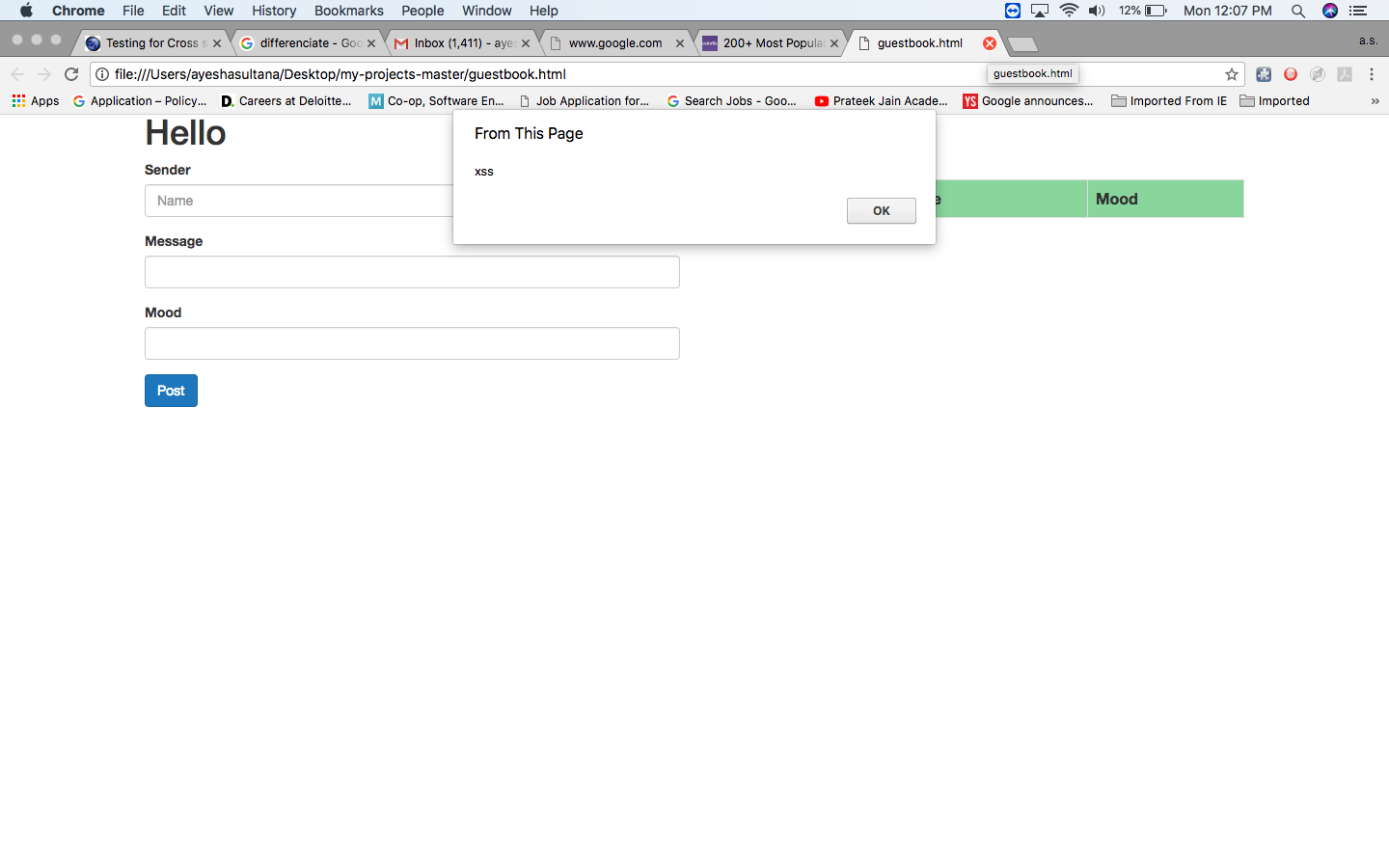
**What is an XSS attack? Explain what can lead to one and how can it be prevented?**

Answer:

XSS is known as a Client-Side code vulnerability that allows a user or hacker to inject code on websites and make the website vulnerable with that code to get valuable information’s, cookies, session tokens and other personal information’s of any user using the injected websites. During my web development class there was a time when I wrote a JavaScript code in our professor’s website to test if I can inject the site. For that project, we were supposed to enter some input in a form where it was required to write our name, mood and a small message. So, when I wrote JavaScript code with an alert message instead of texts in the message box and submitted it the code got executed and affected the whole website as it was a persistent attack. By persistent attack it means that it affects the whole website and not just the current session and it (malicious url, alert message) stays there until the administrator removes the code and write functions to prevent XSS attacks. The main reason why the code got executed is because the server could not differentiate the website’s actual code and the user’s input so it executed that JavaScript code thinking it’s the websites actual code. We were not able to submit any input in that form until the website was fixed. The professor did not realize there was some flaw in his code which made that website XSS vulnerable until I showed him what I was able to do and could anyone else in that class can inject the website direct the users to some malicious websites.



The professor used string escaping for preventing any future JavaScript injection from getting executed. The professor fixed the website and showed us ways how we can defend our own websites from any future XSS attack. One way was by using X-XSS-Protection, so if anyone tries to write JavaScript in the text box of that website it will enable XSS filtering and prevent rendering of the page and also block the user from taking any further action and will not execute the code. X-XSS-Protection prevents unauthorized user’s from executing code on any website.

Some Http header examples from X-XSS-Protection MDN source page are:

X-XSS-Protection: 0

X-XSS-Protection: 1

X-XSS-Protection: 1; mode =block

X-XSS-Protection: 1; report=<reporting-uri>

Content-Security-Policy

X-WebKit-CSP: default-src ‘self’

Therefore, if a website is XSS (Cross Site Scripting) vulnerable and the website has a form to input text that will allow anyone to write HTML and JavaScript codes. If a hacker writes some sort of codes for example JavaScript code (<body bgcolor= “blue”>, <script>alert (“this website is fake”);</script>) that code will get executed in that website. It changes the websites current session or sometimes the whole websites background color/ pop an alert message depending on the persistent (permanent and affects the whole website) and non-persistent (temporary only affects the current session) attack, this is known as XSS attack which can cause security issues. If the hacker uses JavaScript for XSS attack, then the system gets compromised because JavaScript interacts with cookies and the browser saves the users password and other valuable information’s in the cookies. For example, the hacker can do phishing using DOM-based-Cross-Site-Scripting where the hacker can use a fake URL on a particular website such as online banking website’s URL. So, when the user opens his bank account in that website it will redirect the user to the URL the hacker wants the user to click which will open a fake URL and the website will look exactly like the back account website the user is trying to log in to. So, the moment the user puts his/her username and password it gets compromised as the hacker now can easily access to that user’s private bank account as the information gets stored where the hacker wants them to be displayed and saved. This is how many people’s credit card, social security information gets stolen and they can also become a victim of identity theft.

More information’s can be found about the XSS attack prevention rules in the link below. <https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet#A_Positive_XSS_Prevention_Model>

Even though XSS attack is very common in present days most websites now-a-days is protected from XSS attack. For testing purpose, I have examined many local websites to see if I can find any XSS vulnerabilities in them or not and found out they were safe from XSS vulnerabilities.

There were two more techniques my professor mentioned to prevent cross site scripting is which are 1) Input Validation and 2) Input Transformation

Input validation: It puts restrictions on user input by specifying a list of characters known as “Blacklist” where one can prevent the user from using characters like <, >, “”, and symbols and some dangerous character such as <script> </script> tags in their website. So, the user cannot execute any malicious codes while searching and commenting in a website. But the downside of “Blacklist” characters is that one has to list each and every

dangerous characters and string that can allow XSS attack otherwise there is still chances of XSS vulnerabilities exploitation.

Input Transformation: It uses encoding which transforms users input directly that means the server does not interpret the input as code and prevents from code execution. To do that some characters need to be encoded for example <, > special symbols and it can be done by using HTML character entities that corresponds to special symbols (&It, &gt). These symbols help’s the browser to identify if it’s the code on the website or just a user input which needs to be prevented. I have attached the screenshot of the local website I was testing to see how they protect websites from XSS attacks and it shows that the used encoding to transform user input. Therefore, when I typed my JavaScript code it encoded < and > special symbols, the browser identified that it’s not the website’s original code and prevented the code from getting executed. It did it by escaping inputs that are untrusted to JavaScript and CSS to avoid XSS attacks. Escaping handles special characters (“”) in a proper way by preventing them from being parsed as code (by using \ before “”).