The TLS Handshake

a	Briefly, what are the goals of a TLS handshake?	Overall, the aim is to create a secure channel by authenticating the client and server involved in the connection and agreeing on a master secret.
b	Why isn't a Diffie-Hellman key exchange sufficient to meet these goals?	A DH key exchange alone has a weakness in the lack of authentication, i.e. there is not an authentication procedure, so it's vulnerable to a MITM attack.
c	In our "Protocol 2.2," explain carefully why you believe this protocol to achieve the TLS handshake goals. Alternatively, if you think Protocol 2.2 is not up to the task, explain carefully why not, with a detailed example of what could go wrong.	The protocol ensures that Alice knows she is talking with Bob because Bob should use his secret key to encrypt the random number, and Bob can do the same thing to authenticate he's talking to Alice. If there were interference, Alice would not be able to decrypt the message back given Bob's ID when receiving the response by a man in the middle, Eve.
d	For each of the elements of Protocol 2.2, identify where in RFC 8446 that element or its analogue occurs during a TLS handshake.	Outlined on page 11–12 of RFC 8446, the key exchange phase contains a "random nonce" that should be R, as nonces are used for challenge-response authentication processes. Also in the same phase is "a list of symmetric cipher/HKDF hash pairs" which symbolize the shared or public key, or g^x mod p (in section 4.2.8.1) in Protocol 2.2. These are within "key_share" for ClientHello and ServerHello. The secret key should be generated during the key exchange phase and a challenge is issued in RFC 8446 as CertificateRequest, Certificate, and CertificateVerify (where Bob's private key would be used), exchanging the Authentication messages between the client and server.

A little pen-testing

a	Use gobuster to try to find all the subfolders of this website's top-level folder (i.e., the folder that contains index.php). List all the subfolders that gobuster finds.	/.git/HEAD, /index.php, /secrets and /uploadedimages
b	Use the gobuster results to help you find a file named kinda_secret.txt. Tell me where you found this file, and what the secret is.	It's in the secrets directory. Its content is "Congratulations! (ASCII art of a frog) by Joan Stark, http://www.asciiart.eu/animals/frogs

c	In what folder are the uploaded files stored? How do you know?	They're stored in uploadedimages because when uploading an image, it states the target as uploadedimages.
d	Are you able to view the uploaded files? If so, how? If not, why not?	When I try to open http://danger.jeffondich.com/uploadedimages subfolder, it gives me a 403 Forbidden message, so I can't view files outside my own (plus some other files like a picture of a goat or moose).
e	Will the website allow you to upload files that are not image files?	Yes.
f	I wonder what would happen if I uploaded a PHP file instead of an image file...maybe you could show me a particular PHP file, modeled on the class notes from November 11, that I could upload for nefarious purposes.	(I forgot to include my username in some of the stuff I uploaded, but I just uploaded files somewhere along the lines of some-code.php.)
g	Assuming you now have a sufficiently nefarious PHP file living on the danger.jeffondich.com server, show me how you can list all the subfolders of the web root folder, not just the ones that the gobuster output gave you.	I just copied and pasted the code from the class notes. I then typed a random URL followed by ";" with command line prompts. Typing "google.com;ls /var/www" could lead me to the directory of the web root.
h	Find a file named secret.txt, tell me which folder it's in, and give me its contents.	/var/www/danger.jeffondich.com/youwontfindthiswithgobuster/secret.txt. The contents are "Congratulations! (ASCII art of two birds on land) https://www.asciiart.eu/animals/birds-land"
i	Find a file named supersecret.txt somewhere on this server, tell me which folder it's in, and tell me its contents.	It's on /home/jeff/supersecret.txt. It says "Congratulations! (ASCII art of a moose) https://www.asciiart.eu/animals/moose"

Same-origin policy

a	Assume that the browser is *not* using the same-origin policy. Show an information flow illustrating how an attacker could harm the user of the browser.	The user would be prone to a cross site request forgery (CSRF) attack. **1.** Mal (attacker) forges a request… for example, a form **2.** Mal embeds this into a hyperlink and

		sends it to visitors who may be logged in to the site **3.** When the user submits a form on the site, for example, the data or request gets inadvertently sent to the attacker
b	Now assume that the browser is using the same-origin policy. Show the step(s) in the flow from your previous answer at which this policy thwarts the attack. Explain clearly why the attack fails.	The step is thwarted at the last step when the user submits a form. With same-origin policy (and a little JavaScript doing a double submit with onClick), the cookie value gets added to the form and gets read by JavaScript. Now, the form can only run from within the origin, or JavaScript couldn't read the cookie.
c	Suppose a website wanted to provide web pages from port 443 and also an API from port 8888. For example, suppose there's a search form at https://tapirsunlimited.com/search/, but when the user clicks the Submit button on that page, the Javascript on the search page issues a query to https://tapirsunlimited.com:8888/ to get the search results. This architecture separates the user interface from the database access in a fairly natural way, so you can imagine why a web developer might want to organize things this way.	—
c1	What about this system architecture would violate the same-origin policy on the user's browser?	The port.
c2	In specific detail, what would the tapirsunlimited.com developer need to do to take advantage of Cross-Origin Resource Sharing (CORS) to enable the two-port architecture to work?	The developer could use a XML HTTP request in JavaScript to make a GET request to https://tapirsunlimited.com:8888/. On the browser side, a GET request gets sent to https://tapirsunlimited.com:8888/ with an extra header called ORIGIN set to https://tapirsunlimited.com/search/. https://tapirsunlimited.com:8888/ then responds back with an Access Control Allow Origin header set to *.

Fun for Jeff

Sky: Children of the Light is a free-to-play fun, relaxing game that you can play solo or with other people. The world is really beautiful and the characters are cute, with its story remaining relatively self-interpretative.