1. Before we sent the passwords to the JTR tool, we generated and saved the MD5 hashes of the passwords in the password lists. This was done because servers and services don’t (or at least shouldn’t) store raw passwords. Passwords should only be stored in a hashed format. This is done to take advantage of the desired properties of one-way hash functions. Mainly, that it should be infeasible to find a password M that will match a given hashed value simply from knowing the hashed value. The overall effect is to make unauthorized access significantly harder, even when a password file has been successfully stolen.
2. C: High; Confidentiality has a number of requirements. First is that someone who simply finds or steals the card should not be able to access the account. In the example, this is taken care of with the additional 2FA in the form of the PIN. Another requirement is that interactions between the authorized user and their account need to be secure. For example, the web traffic should be secure to counter data being intercepted.  
   I: High; as much as a certain game has made generations of people hope for bank errors in our favor, we generally want the actions that the user takes to be carried out as intended. An example of insuring integrity is to have all transactions have atomicity.  
   A: Med; This is probably the most subjective area for this example. An ATM needs to work when users want it to work (otherwise, it has no point of being there in the first place), but I personally would rather have a secure, but occasionally broken ATM to an unsecure, but always working ATM. An example of an availability requirement would be something like a minimum uptime, or something to the effect of 99.99% of the attempted transactions should be successful.
3. Encryption: Bob should encrypt the message with Alice’s public key  
   Decryption: Alice will decrypt the message using her own private key
4. Encryption: Bob should encrypt the message with his own private key  
   Decryption: Alice will decrypt the message using Bob’s public key
5. A. Hash Function  
   B. Private Key  
   C. Encryption  
   D. Message Digest  
   E. Public Key  
   F. Decryption
6. My C: 11  
   n = pq  
   35 factors into 5 and 7  
   p = 5, q = 7  
   ϕ(n) = (5 – 1)(7 – 1) = 24  
   ed mod ϕ(n) = 1  
   5d mod 24 = 1  
   d = 5  
   M = 16
7. Regex: Basically, regex is used to validate input by checking whether the input follows an expected pattern.  
   Encode and escape: encoding sees special characters are translated into other, non-dangerous characters. Basically, the input is mangled so that it is no longer functional code. Escaping adds an escape character before specific characters to prevent misinterpretation, such as ensuring that a “ character is interpreted as the string character, rather than as the end of a string in some code.  
   CVE-2024-7485  
   The Traffic Manager plugin for WordPress  
   A screenshot of a computer

   Description automatically generated  
   A screenshot of a computer

   Description automatically generated  
   A specific parameter (‘page’) in the ‘UserWebStat’ AJAX function is vulnerable to a cross-site scripting attack due to insufficient input sanitization and output escaping.
8. I thought that the password strength lab was far and away the most informative lab we did this semester. All the other labs placed us firmly in the role of the defender, setting up defenses and confirming that they worked. This lab had us in the role of the attacker, and I thought the perspective was incredibly valuable. This was both the most important and the most interesting thing, allowing us to see how an attacker would use these various password cracking tools, and specifically in weighing the advantages and disadvantages of each of the different methods. I will use this knowledge in the future by reminding myself that on the other end of whatever program I write or cybersecurity defenses I put in place, is a human. A fairly famous book once said, “If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle.” This is perhaps a little cliché, but I thought it was an important learning point. A different way to look at it is by inverting one of the principles of secure design, namely psychological availability. In its standard usage, this principle states that our security systems should not overly burden users, otherwise they will seek to circumvent them. Inverting this principle would suggest that one way of deterring attackers is to make our system psychologically unavailable to attack.
9. Honestly, and somewhat depressingly, the most surprising thing was how incredibly easy it is to leave vulnerabilities in our software systems. Buffer overflows can result from simply using the wrong function (particularly in C type languages) to get user input. As mentioned in question 7, simply forgetting to validate and mangle input can also lead to a variety of vulnerabilities. And we probably covered another dozen or so examples of this sort of thing during the course, and I’m sure this course only touched on a small fraction of the possible vulnerabilities. This insight was surprising because it implies how monumental a challenge it is to create secure software systems, and produces genuine amazement that people are actually able to do so. The primary use of this information to me is has started the process of creating and learning a list of things that I need to look out for when writing software. But more importantly, this knowledge will be used to remind me that the list I currently have in my mind is in no way even remotely close to being all encompassing. This means that if I ever want to actually get proficient at writing software and creating secure software systems, or even if I just want to break into the cybersecurity field, I’m basically signing up for a lifetime of learning, where I would need to continuously add simple things to look out for to my ever-growing list of things which can lead to possible vulnerabilities.
10. Rule 1: Allows systems in the internal protected network (application and database servers and workstation) to send HTTP (port 80), HTTPS (port 443), and HTTP proxy (port 8080) requests to the web proxy located in the internal DMZ network.  
    Rule 2: Allows the web proxy to send HTTP (port 80), HTTPS (port 443), and HTTP proxy (port 8080) requests to the external internet. Basically, forwarding the requests from rule 1.  
    Rule 3: Allows external web users to send HTTP (port 80), HTTPS (port 443), and HTTP proxy (port 8080) requests to the web server located in the internal DMZ network. Basically, allows external users to access the business’s website.  
      
    A web proxy is a system which provides a gateway between users and the internet, basically an intermediary which helps prevent attackers from accessing a private network.
11. Of the two, I would select S/MIME. This was selected due to PGP being unsuitable for use at an enterprise. It is designed for individuals, not companies. Key revocations are still unresolved. And it has poor usability due to being unable to integrate into email clients. S/MIME on the other hand does not run into these issues, and most importantly from a cybersecurity point of view, it has a centralized trust model which allows for certificate revocations and issuance.
12. SSH: application  
    TLS: application, presentation, session, and transport  
    IPSec: network  
    ARP Spoofing: data link  
    SYN Flood: transport  
    Ping of death: network