Most Repeated Errors in COM-301 Final 2024

Last edit: 10/02/2025

General Mistakes

- Inconsistent answers. Some of the questions in this exam have several parts, e.g., Q1 asked for an attack, the answers of Q2-4 are based on the attack written in Q1. Some of the answers in Q2-4 did not match or address the attack described in Q1.
- Not giving a direct answer to the “yes or no” question. Or alternatively answering both yes AND no.

Q1

- **Stating that BGP hijacking alone can prevent HR from accessing the seehowiparty website**. BGP hijacking by itself does not drop nor modify packets, only reroutes them but they still arrive at the correct destination (Peach). Hence it is not enough to achieve Paul’s attack goal. Answers that only mention BGP hijacking received no point.

Q2 - Q4

- **Not specifying which DNS attack when arguing over defenses.** DNS spoofing includes DNS hijacking and DNS cache poisoning. DNS hijacking actively tampers with the DNS response, e.g., by responding with a fake IP address to the DNS request or not giving any DNS response at all; while DNS cache poisoning replaces the cached answer in the DNS resolver’s storage with a value decided by the attacker even before the DNS query is made. These two types of attacks result in different answers when it comes to defense mechanisms, so it was important to specify which of the two was considered.

Q3

- **Not mentioning the HTTPS certificate when arguing about whether HTTPS protects from DNS-related attacks.** In HTTPS, there is a certificate check when connecting to a website. Some answers neglected that, if a fake IP address is given to the HR’s DNS request as part of a DNS hijacking attack, when HR connects to the website at this fake IP address, this website’s certificate will not

match with the one expected for seehowiparty. In this case, HR will notice that something went wrong by the alert caused by the mismatched certificate and detect when a DNS-related attack is happening. Since detecting helps toward protecting from a DNS attack, answers that argued yes (detecting and alerting the user is enough to protect) and no (detecting is not enough to protects) were given points, but justifications that do not mention the certificate (for either yes or no) received no points.

Q5 - Q7

- **Stating that ASLR shuffles the position of the variables on the stack.** ASLR randomizes the page mappings, not how the variables are allocated on the stack. If we consider the program given in this exercise, the address of the stack base and functions will be different per execution; but relatively, x and y will always be placed contiguously on the stack.
- **Pointing out the lack of sanitization of b as vulnerability.** The fact that b is not sanitized is not relevant in our case. Similarly, b is, by definition, an int64_t (which is a 64-bit integer). It is not possible to "buffer overflow" such a value, because it is an integer. Hence this is not a vulnerability.
- **Misusing the terminology.** Many answers referred to "memory leak": a memory leak in software refers to an improper allocation of resources (e.g., `malloc` without `free`.) Similarly, about "stack overflow" / "stack frame": in this case, the attack is not about a stack overflow but rather a buffer overflow. In the context of the exercise, it can be unclear to what a "stack overflow" refers to. A stack frame contains variables, arguments, return address and the stack base pointer. The attack does not overflow the stack frame.
- **Using a wrong format for the value b**. Some answers confuse the usage of ampersand (&) or asterisk (*) in C, which results in incorrect format when answering the value of b. In line 10, it returns *(y[0]): the asterisk (*) means that we are returning the *value* pointed by the pointer y[0]. The value of y[0] is *an address of a value* which y[0] points to. As a reminder of how ampersand (&) or asterisk (*) are used in C, an example explanation is https://stackoverflow.com/a/2094715; As a reminder of how C pointer works (its own value and the value it points to), an example explanation is https://www.geeksforgeeks.org/c-pointers/.
- **Stating that canaries protect the buffers x, y.** Some answers assumed that stack canaries protect all buffers in the stack, they do not. As we have seen in the class, the canaries are only before the return address (and therefore protect only the return address.) However, there exists extensions for compilers to do that, but by default, it is not enabled.

Q8 - Q9

- **Declaring that Meta is an Internet Service Provider or controls an Internet Service provider.** Multiple answers contained statements that Meta is either an ISP or controls an ISP, and therefore it is a global adversary. Such assumptions are not based on any information in the question nor in real world. Indeed, Meta is a global adversary in this question, but it operates on the application level and not on the level of the internet infrastructure. Meta neither can nor needs to read all the packets on the internet, since all the packets are moving through their servers anyway.
- **Using unjustified assumptions.** For example, multiple answers considered scenarios when corporate WiFi is used to communicate via Tor. However, there is no information in the question pointing to it; moreover, using the corporate hardware for sending the sensitive information outside is not the most straightforward option. We removed points depending on how realistic/critical the assumption is. It is completely valid to use assumptions if one of the following is present in the answer: 1) an explanation why this assumption holds in reality; 2) what happens if the assumption does not hold.

Q10

- **Justification does not address the particularities of CSRF.** Multiple answers justifications only suggest that general attack types are possible and not specifically CSRF. Such answers do not discuss the reasons why CSRF is *specifically* possible (e.g. having no checks for origin) or how a CSRF attack can be executed against the server-side code presented. This includes generic answers that throw keywords like “ambient authority”, “same origin policy”, “confused deputy”, etc. without linking them to the problem at hand. These answers received partial points, depending on the lack in justification.
- **Claiming the professor’s action *fully* protects them.** Many answers do not make the distinction between *fully* protecting the professor and *partially* protecting them *sometimes*. When the question mentions “… would protect him …” without any adjectives, it is necessary to assume the requirement to be *full protection***.** Answers that argue for full protection did not receive full points.
- **Mentioning the student “steals” the cookie and/or performs actions unrelated to CSRF.** Some answers describe an adversarial model that is not typical of CSRF, demonstrating a lack of understanding of CSRF. Some of these answers mention that the student can actively steal the cookie or send the request from their own browser which describes orthogonal and most often incoherent attacks. These answers only got partial points.

- **Saying the cookie is insufficient because it includes the username which is publicly known.** Some answers demonstrate a lack of understanding in the role of session cookies, describing them as only a storage location for public information like username and email. These answers only got partial points.
- **Confusing CSRF with XSS.** Some answers confuse cross-site scripting and cross-site request forgery either by explicitly naming the attack incorrectly or providing a description or justification for XSS. These answers did not receive points.

Q11

- **Not taking into consideration that line 6 will exit the code if username is incorrect.** Most answers describe a code injection attack at line 14 by modifying the username without addressing in any way or form that line 6 will cause the code to exit. Answers that present assumptions about how line 6 can be bypassed with a malicious username value were partially graded depending on whether the assumption is reasonable. Unreasonable assumptions include mentioning that "username=<script>..." will still bypass line 6. Such answers did not receive points.
- **Mentioning OS code injection or SQL injection.** Some answers that still point to line 6 reasonably describe attacks infeasible through exploiting line 14 in isolation like OS code injection or SQL injection. These answers did not receive points.
- **Orthogonal answers and mentioning defenses.** Some answers suggest defenses – which we don't ask for. In most cases, we did not reward the defense with any points, but if the defense is canonically wrong, we removed points depending on the severity of the error.
- **Mentioning attacks that exploit lines other than line 14.** Some answers suggest attacks to exploit other lines than the requested one. These answers did not receive points as they don't answer the question.
- **Thinking that the PHP code is executed over time.** Some answers suggest the idea that the PHP code for the webpage is executed slowly as the user visits the page, implying that "we can change username after we execute line 6". This is incorrect as the totality of the PHP code is run on the server before the response containing the rendered webpage HTML is returned to the client. These answers did not receive points.

Q12

- **Proposing to plug the USB stick in a factory-reset computer without disconnecting from the network.** Since the malware could be spreading over

the network, it is also important to disconnect the computer from the Wi-Fi (and network in general).

- **Proposing to only disconnect Alex's laptop from the network.** This is good to stop the spreading of the malware when plugging the USB stick but will not stop Alex's laptop from being infected. Additionally, the malware can eventually spread once reconnected to the network, which is bound to happen, since it's most likely not a computer Alex can just reset or throw away.

Q13

- **Stating that the malware is a virus because it needs a host, and that the host is Alex's laptop.** A virus needs a host application/program, as opposed to malware that are self-contained programs like worms. But both need to be on a host machine.
- **Indicating two types of malwares when just one was asked.** This makes correcting Q14 impossible, as the answer depends on the type of malware.
- **Stating the malware is a botnet.** A botnet is NOT a type of malware, it is a machine infected and belonging to a network of compromised hosts. A botnet could be considered the "payload" of a malware, i.e., what it does once it has compromised a host.

Q14

- **Redefining the firewall to answer the question.** The simple firewall was described in the question, so assuming the firewall functioned differently than stated, by performing deep packet inspection for example, was not counted as a valid justification for an answer.
- **Stating that a worm can infect known applications to spread over the network.** A worm is its own stand-alone application that could compromise or use another application to spread (through emails for example), however it does not **infect** another application.

Q15

- **Answering that the user will always enter the same time** because the key does not change. Both the IV and the message change with time, so the ciphertext will also be a new pseudo-random value at each epoch.

Q16

- **Recovering the key from the IV** as an additional vulnerability. In the token case, the IV is not sent over the network, only computed by the client and server locally. Thus, an adversary cannot recover the key, as it never sees the IV.
- **Stating that there is an IV reuse vulnerability**, since the time can wrap around modulo 2^256. While technically true, 30 seconds * 2^{256} is roughly 10^{71} years (the universe is roughly 10^{10} years old).
- **Stating that the IV is predictable, without arguing why this is an issue**, for many ciphers (including AES modes) predictable IVs are fine as long as they are unique. Stating that the IV is predictable could lead to a vulnerability depending on the exact cipher mode used is an acceptable answer.

Q17

- **Not giving a counterexample**. While this is stressed in previous MREs, many did not fully read the question statement and did not answer accordingly. As asked in the question, you must give a counterexample when justifying no. Justifications consisting of “No because it’s easy to find (k’, t') such that h(k, t) = h(k’, t’)” without a counterexample were not given any point.
- **Choosing and fixing the message (k, t) or the hash h when giving a counterexample for first and second preimage resistance.** For example, for 2^{nd} preimage resistance, (k, t) and h are arbitrary. When choosing a value for the counterexample, e.g., “assuming (k, t) = (0, 0), then (1, 1) is a counterexample”, what you are doing is providing a counterexample against collision resistance, not 2^{nd} preimage resistance. In 2^{nd} preimage resistance, x (here x = (k, t)) is arbitrary. An example of valid counterexample against 2^{nd} preimage resistance is "(pad(t), k) has the same hash as (k, t)".
- **Assuming k or t is always known.** The fact that in the application XORHash is used, the second input to XORHash t is the time indeed implies that one can argue that it is known, assuming that the challenges are fresh. However, this is a characteristic of the application, not of the function. These answers did not receive full points. Answers assuming k was known did not receive any point because even in the context of the application the function XORHash is used in, assuming a secret key is known is wrong.