Fundamentals of Cybersecurity (CS 4222/6222)
Final Examination Practice Questions – Solutions

Instructor: Olusesi Balogun
Department of Computer Science
Georgia State University
Fall 2025

PART A – Multiple-Choice Solutions

1. C (Availability)
2. B (Integrity)
3. B (Confidentiality)
4. C (Accounting)
5. B (Authentication)
6. B (Insider misuse)
7. B (Denial-of-Service / DDoS)
8. B (Social engineering)
9. C (White-box testing)
10. B (Security testing / security assessment)
11. B (Give each user only the access needed)

12. B (Complete mediation)
13. B (Security by obscurity)
14. B (Scales better for large organizations)
15. B (Access Control List)
16. B (Larger character set and longer length)
17. B (Rainbow table attack)
18. B (Salt prevents identical hashes and precomputation)
19. C (Buffer overflow)
20. B (Virus)
21. B (Worm)
22. C (Black box)
23. C (Be proportional to resources of attackers)
24. A (Least Privilege Principle)
25. C (More efficient bandwidth usage and flexibility)
26. B (ARP)
27. B (DNS)
28. C (Network layer)
29. C (Jam then random backoff)
30. B (SYN, SYN+ACK, ACK)
31. B (Exponential increase from small value)
32. B (Stateful firewall)
33. B (Encrypted tunnel over untrusted network)
34. D (Link layer)

35. C (Broadcast ARP for DNS server MAC)
36. A (ARP replies from both devices confuse mapping)
37. B (5001)
38. C (Add random unique salt per user)
39. A (Broadcast storm)
40. B (Congestion control)
41. C (UDP)
42. A (Change private internal IPs into a single public IP)
43. B (Return address and frame pointer)
44. C (Random canary value)
45. A (Base rate fallacy)
46. A (Cross-origin script access in the browser)
47. B (Cross-Site Scripting, XSS)
48. A (Cross-Site Request Forgery, CSRF)
49. A (SQL Injection)
50. B (Stored XSS in the browser)
51. C (Replace (a, b) with $(b, a \bmod b)$)
52. B (Their greatest common divisor is 1)
53. B ((n, e) where $n = pq$)
54. B (Verify origin and integrity)
55. B (Quasi-identifiers)
56. A (k-anonymity)
57. B (Group distribution close to overall distribution)

58. B (Forensic imaging)
59. B (Prevent modifications to original evidence)
60. B (Baiting)
61. A (Spear phishing)
62. B (Carefully perturbed input causing misclassification)
63. B (Fairness and bias in AI)
64. B (Understand/challenge decisions for trust & accountability)
65. B (Defense in depth)

PART B – Problem-Solving and Short-Answer Solutions

1. **CIA Goals in Context**

   a) **Confidentiality** is violated: unauthorized users can see students' grades without logging in.

   b) **Integrity** is violated: the legitimate patch is replaced with malicious code, so software no longer reflects the intended, correct state.

   c) **Availability** is impacted: the LMS is down for 6 hours during a critical period, so legitimate users cannot access needed resources.

2. **Threat Types and Attack Classification**

   a) **Denial-of-service**: dropping 30% of VoIP packets degrades service until it becomes unusable.

   b) **Masquerading**: the intruder uses stolen credentials to act as the CEO.

   c) **Correlation**: the agency infers relationships from timing and volume of encrypted traffic without seeing contents.

3. **Access Control Matrix, ACL, and RBAC Design**

   Let subjects: Alice (Manager), Bob (Engineer), Eve (Intern). Objects: HR_Records, Source_Code, Public_Docs. Rights: R, W, X.

   a) **Sample Access Control Matrix**

	HR_Records	Source_Code	Public_Docs
Alice	R,W	R	R
Bob	–	R,W	R
Eve	–	–	R

   b) **ACL Representation**

   - HR_Records: { (Alice: R,W) }
   - Source_Code: { (Alice: R), (Bob: R,W) }

- Public_Docs: { (Alice: R), (Bob: R), (Eve: R) }

c) **RBAC Roles**

Define roles and assign permissions to those roles based on the given context:

- **Manager**: HR_Records (R,W), Source_Code (R), Public_Docs (R)
- **Engineer**: Source_Code (R,W), Public_Docs (R)
- **Intern**: Public_Docs (R)

Then, assign users to the roles:

- Alice → Manager
- Bob → Engineer
- Eve → Intern

This role-permission mapping reproduces the original matrix.

4. **Password Space and Entropy**

a) Alphabet size = 36 (26 letters + 10 digits), length = 10.
Total passwords:

$$N = 36^{10} \approx 3.66 \times 10^{15}.$$

Entropy in bits:

$$H = \log_2(36^{10}) = 10 \log_2(36) \approx 10 \times 5.17 \approx 51.7 \text{ bits}.$$

b) Increasing the length multiplies the search space (exponentially in length), while increasing the character set increases the base of the exponent. Together they make brute-force attacks require many more guesses and thus much more time and computational power.

5. **Password Vulnerabilities and Salting**

a) **Examples:**

- *Reuse*: user uses the same password for email, banking, and university account; if one site is breached, all are at risk.

- *Weak composition*: password is `password123`; easily guessed via dictionary or simple brute force.

b) **Salting process and benefit:**

- For each user, generate a unique random salt s.
- Store s and the hash $h = H(s\|\text{password})$.
- When a user logs in, recompute $H(s\|\text{password\_input})$ and compare to stored h.
- Because salts differ per user, the same password produces different hashes; precomputed rainbow tables for unsalted hashes no longer apply, and an attacker must brute-force each account separately.

6. **Buffer Overflow Root Cause and Mitigation**

a) **Overflow explanation:** `buffer` is 16 bytes, `input` can hold 64 bytes. `strcpy(buffer, input)` copies until a null byte without checking length, so if `input` holds more than 15 characters plus terminator, bytes beyond `buffer` overwrite adjacent memory on the stack (saved frame pointer, saved return address, local variables), potentially allowing control-flow hijacking.

b) **Safer version (example):**

```
void process() {
    char buffer[16];
    char input[64];

    /* ensure input is safely obtained first, e.g., via fgets */
    if (fgets(input, sizeof(input), stdin) != NULL) {
        /* copy at most sizeof(buffer) - 1 characters */
        strncpy(buffer, input, sizeof(buffer) - 1);
        buffer[sizeof(buffer) - 1] = '\0';  // ensure null-termination
    }
}
```

c) **Defenses:**

- Compile-time: stack protection (e.g., `-fstack-protector`) inserts canaries and checks them before returning.

- Run-time: Address Space Layout Randomization (ASLR) randomizes memory layout so an attacker cannot easily predict the address of injected code or return targets.

7. **Malware Scenario and Botnets**

   a) Periodic connections to an unknown IP on a non-standard port followed by coordinated traffic spikes match classic **botnet** behavior: infected hosts contact a command-and-control (C2) server, receive instructions, then participate in attacks (such as DDoS).

   b) Likely initial infection vectors:
   - Users opened a malicious email attachment or link (phishing).
   - Drive-by download from a compromised web site exploiting browser/plugin vulnerabilities.

   c) **Defensive measures (examples):**
   - Deploy network IDS/IPS and egress filtering to detect and block suspicious outbound C2 traffic.
   - Maintain up-to-date patching and endpoint protection (anti-malware/EDR) on workstations.
   - Use DNS filtering and web proxies to block known malicious domains.

8. **Intrusion Detection System and Malware**

   a) **Firewall vs IDS:**
   - A **firewall** primarily controls traffic flow based on rules (IP/port/protocol, sometimes application-level), blocking or allowing connections.
   - An **IDS** monitors traffic or host activity for signs of attacks or policy violations and generates alerts; it is usually passive (detection, not blocking) unless integrated as IPS.

   b) **Virus, Trojan, Worm:**
   - **Virus**: attaches to legitimate files/programs and spreads when the host file is executed; often requires user action.
   - **Trojan**: appears benign or useful but secretly performs malicious actions; does not self-replicate by exploiting vulnerabilities.

- **Worm**: self-replicating malware that spreads over networks autonomously by exploiting vulnerabilities, without user action.

c) **White-list vs Black-list Firewall:**

- **Whitelist**: "default deny": only explicitly allowed traffic is permitted; all else blocked.
- **Blacklist**: "default allow": all traffic is allowed except explicitly blocked addresses/ports/services.

d) **Penetration Test (P) vs Security Test (S):**

#	**Activity**	**P**	**S**
i	Attempting to exploit a misconfigured web server to gain unauthorized access (authorized engagement).	X	
ii	Reviewing password policies vs standards.		X
iii	Evaluating whether incident-response procedures are followed.		X
iv	Controlled vulnerability scan to find outdated software.		X
v	Simulating a phishing attack against employees to test susceptibility.	X	

9. **DNS Resolution and Attack Surface**

a) **Main steps (client view):**

i) Browser checks its own cache / OS cache for `www.bank.com`.

ii) If not cached, the OS sends a DNS query to the configured resolver (often via UDP/53).

iii) The resolver (recursive DNS server) checks its cache; if needed, it queries root, then TLD (e.g., `.com`) servers, then the authoritative server for `bank.com`.

iv) The resolver gets the answer (IP address), caches it, and returns it to the client.

v) The client uses that IP to establish a TCP/TLS connection to the server.

b) **Two attack points (examples):**

- **DNS cache poisoning**: attacker injects fake DNS responses into the resolver's cache so `www.bank.com` resolves to an attacker-controlled IP, leading to credential theft.
- **Rogue DNS server / DHCP poisoning**: attacker tricks the client into using a malicious DNS server that always returns attacker-controlled IPs, enabling large-scale redirection.

10. **IPv4 Subnetting**

Network `192.168.10.0/27`:

a) **Subnet mask**: /27 → `255.255.255.224`.

b) Number of host bits: $32-27=5$. Total addresses $2^5=32$. Usable hosts:

$$32-2=30.$$

c) Block size: 32 addresses. Network address: `192.168.10.0`. Broadcast address: `192.168.10.31`. Usable hosts: `192.168.10.1`–`192.168.10.30`.

d) To support at least 50 usable hosts:

$$2^{(32-\text{prefix})}-2\geq 50.$$

Try 6 host bits: $2^6-2=64-2=62\geq 50$. So need 6 host bits ⇒ prefix $32-6=26$. A `/27` (30 hosts) is insufficient; `/26` is required.

For **198.168.9.120/25**:

(a) **Subnet mask**: /25 → `255.255.255.128`.

(b) Host bits: $32-25=7$. Total addresses $2^7=128$; usable hosts $128-2=126$.

(c) Address `198.168.9.120` is in the first subnet. Network address: `198.168.9.0`, broadcast: `198.168.9.127`. Usable host range: `198.168.9.1`–`198.168.9.126`.

11. **ARP Poisoning Scenario**

a) A malicious host sends forged ARP replies telling the victim, "The gateway's IP maps to my MAC," and telling the gateway, "The victim's IP maps to my MAC." Both endpoints then send their packets to the attacker, who can forward them on to keep the connection working, achieving a man-in-the-middle.

b) The victim's ARP cache will have incorrect entries: the IP of the gateway mapped to the attacker's MAC; similarly, the gateway will map the victim's IP to the attacker's MAC. Routing still appears normal at the IP layer, but frames are actually sent to the attacker first.

c) Mitigations:

- **Dynamic ARP Inspection (DAI)** with trusted ports: the switch validates ARP replies against DHCP snooping tables; higher complexity/cost and careful configuration needed.
- **Static ARP entries** for critical hosts (e.g., default gateway): prevents spoofing but does not scale well and increases administrative overhead.
- Network segmentation / use of secure switching (e.g., private VLANs, port security) to reduce attack surface.

Specific ARP table state after poisoning (Node 1/Node 2 example):

```
Node 1 ARP (after poisoning):
192.168.5.98  ->  00:15:20:48:20:10   (attacker's MAC for Node 2 IP)

Node 2 ARP (after poisoning):
192.168.6.96  ->  00:15:20:48:20:10   (attacker's MAC for Node 1 IP)
```

Mitigations here are the same: e.g., DAI, static ARP for critical peers, strong network segmentation.

12. **Flow Control vs. Congestion Control**

a) **Flow control** is end-to-end between sender and receiver: it prevents the sender from overwhelming the receiver's buffer. **Congestion control** protects the network: it prevents too much traffic from overloading routers and links.

b) Examples:

- Flow control: TCP sliding window / advertised receive window (rwnd) that limits how much unacknowledged data the sender can have in flight.
- Congestion control: TCP congestion window (cwnd) with slow start and AIMD (additive increase, multiplicative decrease) adjusting the sending rate based on perceived congestion.

13. **Base Rate Fallacy in IDS**

a) Since only 1% of connections are malicious, most traffic is benign. Even with a high true positive rate, a modest false positive rate applied to the large benign population can yield many false alerts. Therefore, the *posterior* probability that an alert is a real attack can be relatively low.

b) Out of 100,000 connections:

- Malicious: 1% $\rightarrow$ 1,000. True positives: 99% of 1,000 $\rightarrow$ 990 alerts.
- Benign: 99% $\rightarrow$ 99,000. False positives: 5% of 99,000 $\rightarrow$ 4,950 alerts.

Total alerts $\approx 990 + 4{,}950 = 5{,}940$. Only $990/5{,}940 \approx 16.7\%$ of alerts correspond to real attacks. Operationally, analysts spend most of their time on false alarms, causing alert fatigue and increasing the chance that real attacks are missed or ignored.

14. **Cookies and Domains**

Recall that, as we looked at the policy in class, a cookie is sent when the request host domain suffix-matches the cookie's `domain` and the request path starts with the cookie `path`.

- **cookie1**: domain = `account.microsoft.com`, path = `/`
- **cookie2**: domain = `microsoft.com`, path = `/my/account`
- **cookie3**: domain = `microsoft.com`, path = `/`

URLs:

(a) `support.microsoft.com`

- cookie1: NO (host is not `account.microsoft.com`).
- cookie2: NO (path "/" does not start with `/my/account`).
- cookie3: YES (domain suffix matches `microsoft.com`, path starts with `/`).

(b) `account.microsoft.com/my/account`

- cookie1: YES (domain match, path / covers /my/account).
- cookie2: YES (host ends with `microsoft.com`, path starts with `/my/account`).
- cookie3: YES (domain match, path /).

(c) `microsoft.com.com/my/page`

- cookie1: NO.
- cookie2: NO (host `microsoft.com.com` does not have `microsoft.com` as a proper suffix domain; different label boundary).
- cookie3: NO (same reason).

(d) `microsoft.com/my/home`

- cookie1: NO.
- cookie2: NO (path `/my/home` does not start with `/my/account`).
- cookie3: YES (domain `microsoft.com`, path /).

Note: When we have path = `/`, it means that path is a root directory and so satisfies all other paths.

15. **Euclidean Algorithm and Modular Inverse**

a) Compute $\gcd(276, 345)$:

$$345 = 276 \cdot 1 + 69$$
$$276 = 69 \cdot 4 + 0$$

So $\gcd(276, 345) = 69$.

b) Solve $17x \equiv 1 \pmod{72}$.
Use extended Euclid:

$$72 = 4 \cdot 17 + 4, \quad 17 = 4 \cdot 4 + 1.$$

Back-substitute:

$$1 = 17 - 4 \cdot 4 = 17 - 4(72 - 4 \cdot 17) = 17 - 4 \cdot 72 + 16 \cdot 17 = 17 \cdot 17 - 4 \cdot 72.$$

So $1 \equiv 17 \cdot 17 \pmod{72}$. Thus $x \equiv 17 \pmod{72}$. One solution is $x = 17$.

16. **Small RSA Example**

First example: $p = 13$, $q = 19$, $e = 5$, $M = 7$.

a) $N = pq = 13 \cdot 19 = 247$. $\phi(N) = (p-1)(q-1) = 12 \cdot 18 = 216$.

b) Find d with $5d \equiv 1 \pmod{216}$. We need inverse of 5 mod 216. Since $216 \equiv 1 \pmod{5}$,

$$1 = 216 - 43 \cdot 5 \Rightarrow 1 \equiv -43 \cdot 5 \pmod{216}.$$

So $d \equiv -43 \equiv 216 - 43 = 173 \pmod{216}$. Take $d = 173$.

c) Encrypt $M = 7$: $C \equiv M^e \pmod{N} = 7^5 \pmod{247}$.

$$7^2 = 49, \quad 7^4 = 49^2 = 2401 \equiv 178 \pmod{247}$$

(since $247 \cdot 9 = 2223$, $2401 - 2223 = 178$).

$$7^5 = 7^4 \cdot 7 \equiv 178 \cdot 7 = 1246 \equiv 11 \pmod{247}$$

($247 \cdot 5 = 1235$, $1246 - 1235 = 11$). So $C = 11$.

d) Decrypt: $M' \equiv C^d \pmod{N} = 11^{173} \pmod{247}$. By construction of RSA, $M' = 7$. (One can verify by modular exponentiation if desired.)

Second example: $p = 17$, $q = 29$, $e = 3$, $M = 100$.

(a) $N = pq = 17 \cdot 29 = 493$. $\phi(N) = (17-1)(29-1) = 16 \cdot 28 = 448$.

(b) Find d with $3d \equiv 1 \pmod{448}$. Since $448 = 3 \cdot 149 + 1$, we have:

$$1 = 448 - 3 \cdot 149 \Rightarrow 1 \equiv -149 \cdot 3 \pmod{448}.$$

So the inverse of 3 is $-149 \equiv 299 \pmod{448}$. Thus $d = 299$.

(c) Encrypt: $C \equiv M^e \pmod{N} = 100^3 \pmod{493}$.

$$100^2 = 10,000 \equiv 140 \pmod{493}$$

(because $493 \cdot 20 = 9,860$, $10,000 - 9,860 = 140$).

$$100^3 = 100 \cdot 100^2 \equiv 100 \cdot 140 = 14,000 \equiv 196 \pmod{493}$$

($493 \cdot 28 = 13,804$, $14,000 - 13,804 = 196$). So $C = 196$.

(d) Decrypt: $M' \equiv C^d \pmod{N} = 196^{299} \pmod{493}$. Because (e, d) were chosen correctly, RSA guarantees $M' = 100$.

17. **k-Anonymity Transformation**

a) 3-anonymity means that, for any combination of quasi-identifiers (ZIP, Age, Gender), each record is indistinguishable from at least 2 other records – i.e., each equivalence class has size at least 3 with respect to the QIs.

b) One possible anonymized table:

ZIP Code	Age	Gender	Disease
4767*	20–40	*	Flu
4767*	20–40	*	Cancer
4767*	20–40	*	Flu
4760*	40–50	*	Diabetes
4760*	40–50	*	Cancer
4760*	40–50	*	Flu

Here:

- ZIP generalized to first 3 digits (4767*, 4760*).
- Age binned into ranges (20–40, 40–50).
- Gender suppressed (*).
- Each QI group has at least 3 records.

c) Trade-off: the more we generalize/suppress (coarser ZIP, age ranges, hide gender), the harder it becomes for attackers to re-identify individuals, but analysts lose detail. For example, they can no longer study exact age or fine-grained geographic patterns. In other words, the more the anonymity, the less the utility.

18. **Cyber Forensics Process and Evidence Handling**

    a) **Main phases (applied to this case):**

    - **Identification**: determine which systems (employee PC, email server, USB devices) may contain evidence.
    - **Acquisition**: create forensic images of the suspect's workstation, mail server mailbox, and any recovered USB drives using write-blockers; capture logs.
    - **Preservation**: securely store images and originals, document handling; maintain integrity (hashes).
    - **Analysis**: examine file access times, USB insertion logs, email content/headers, and data transfers to confirm whether customer data was copied and exfiltrated.
    - **Reporting**: produce a formal report summarizing methods, findings, and timelines for management and potential legal proceedings.

    b) **Chain of custody and hashes:**

    - Chain of custody documents who handled evidence, when, where, and why, demonstrating it was not tampered with or substituted.
    - Cryptographic hashes (e.g., SHA-256) of disk images taken at acquisition and re-verified later prove that the evidence analyzed is bit-for-bit identical to the original, supporting admissibility in court.

19. **Social Engineering Email Analysis**

    a) **Red flags (examples):**

    - Urgent threat: "within 24 hours or payment will be cancelled."
    - Requests sensitive information (bank details, login credentials) via a link.
    - Suspicious domain: link points to `gsu-payments.com`, not official `gsu.edu`.
    - Generic greeting (e.g., "Dear Student") not personalized.

    b) **Social engineering principles:**

- Urgency / fear: short deadline pressures the victim into acting quickly.
- Authority: pretending to be the Financial Aid Office invokes institutional authority.
- Scarcity / loss aversion: threat of losing funds exploits fear of missing out or losing money.
- Deception via look-alike domain: exploits users' inattentional blindness and trust in brand names.

c) **Controls:**

- *Technical*: email filtering and phishing detection; DMARC/SPF/DKIM to authenticate mail; URL filtering and browser warnings for suspicious domains.
- *User-awareness*: regular phishing awareness training; simulated phishing campaigns; clear policies to verify messages via official portals and never click unknown links in unsolicited emails.

20. **Security of Machine Learning Applications**

a) The modified stop sign with stickers is visually still a stop sign to humans, but the small perturbations cause the model to misclassify it as a speed-limit sign. This is an **adversarial example**: a carefully crafted input that fools the model. Safety impact: the vehicle may fail to stop, potentially causing accidents.

b) Possible defenses:

- **Adversarial training**: include adversarially perturbed images during training so the model learns to classify them correctly.
- **Multi-sensor fusion**: combine camera input with other sensors (LIDAR, radar, HD maps) or multiple camera views, so a single perturbed sign is less likely to determine the final decision.
- Additional options: input preprocessing/denoising; ensembles of diverse models; runtime anomaly detection for inputs far from the training distribution.

c) **FATE considerations:**

- **Fairness**: ensure performance is consistent across environments (lighting, weather, regions) and demographic groups, so one population is not disproportionately harmed.
- **Accountability**: clear responsibility for failures, audit logs of decisions, and compliance with safety/ regulatory standards.
- **Transparency / Explainability**: providing interpretable reasons (e.g., highlighting regions used for classification) can help debug failures and increase trust from regulators and the public.