

1. 2-out-of-4 Questions

For each question, please mark the *two* correct answers.

- (1) WPA2
- ☒ may use EAP and IEEE 802.1X during the authentication phase.
 - ☐ always encrypts data using RC4 for compatibility with older devices.
 - ☐ does not support pre-shared keys.
 - ☒ derives Pairwise Transient Key (PTK) from Pairwise Master Key (PMK).
- (2) DNSSEC provides
- ☐ protection against DNS amplification.
 - ☐ data confidentiality.
 - ☒ origin authenticity and data integrity of DNS replies.
 - ☒ backwards compatibility with DNS.
- (3) DomainKeys Identified Mail (DKIM) provides
- ☐ multi-factor user authentication.
 - ☒ compatibility with existing e-mail infrastructure.
 - ☐ encryption based on a combination of asymmetric- and symmetric-key cryptography.
 - ☒ detection of forged sender addresses in e-mail.
- (4) Pretty Good Privacy (PGP)
- ☐ authenticates e-mail servers with the help of DNS records.
 - ☐ protects against IP address spoofing.
 - ☒ can use “Web of Trust” for public-key distribution.
 - ☒ provides confidentiality.
- (5) Which security measures aim to provide confidentiality for payload data?
- ☒ WEP
 - ☐ MAC-address filtering
 - ☒ WPA
 - ☐ hidden SSID
- (6) Stateless firewalls
- ☒ apply rules to each incoming/outgoing packet.
 - ☒ can be used to create Demilitarized Zones (DMZ).
 - ☐ cannot inspect headers of higher-level protocols, such as UDP and TCP.
 - ☐ may need to keep track of every active connection.
- (7) SSL/TLS
- ☐ must authenticate both parties before establishing a session.
 - ☒ runs on top of a transport layer protocol (e.g., TCP).
 - ☒ supports Diffie-Hellman based key exchange.
 - ☐ supports Kerberos based key exchange.
- (8) Cross-site request forgery (CSRF) vulnerabilities
- ☐ may be prevented by filtering special HTML characters.
 - ☒ enable attackers to target servers that they cannot directly access.
 - ☐ are typically exploited by injecting malicious client-side scripts.
 - ☒ are typically exploited by tricking a user into requesting a special URL.
- (9) What can we use to sandbox code?
- ☒ virtual machines
 - ☐ executable space protection
 - ☐ IDS
 - ☒ Linux seccomp
- (10) Buffer-overflow vulnerabilities
- ☐ do not affect buffers that are dynamically allocated on the heap.
 - ☐ can be prevented by filtering characters based on a proper whitelist.
 - ☒ may be exploited by attackers to cause denial-of-service.
 - ☒ may be caused by unsafe functions for copying strings.
- (11) Which statements are true for typical vulnerabilities?
- ☒ Attackers may use integer-overflow vulnerabilities to cause buffer overflows.
 - ☒ Attackers may exploit format-string vulnerabilities to gain sensitive information.
 - ☐ Higher-level languages, such as Java and C#, are not susceptible to integer-overflow vulnerabilities.
 - ☐ Only local attackers (e.g., local users) can exploit race-condition vulnerabilities.
- (12) Mandatory Access Control (MAC)
- ☐ cannot be combined with Discretionary Access Control.
 - ☒ can be used to implement multilevel security.
 - ☒ enforces system-wide rules that are set by a central authority.
 - ☐ allows access rights to be propagated at the subjects' discretion.
- (13) Which statements are true for input validation?
- ☒ Short blacklists have a lower impact on usability than short whitelists.
 - ☒ List `ABC...Zabc...z012...9` may be used as a whitelist to prevent cross-site scripting (XSS).
 - ☐ Input that is provided by an authenticated user does not need to be validated.
 - ☐ Shorter whitelists tend to be less secure.

- (14) When set on a directory, what do these Unix access-control permission bits mean?
- ☒ Setgid bit: new files in the directory will inherit the group of the directory.
 - ☐ Write bit: enables modifying the contents of files in the directory.
 - ☒ Execute bit: enables accessing files in the directory.
 - ☐ Sticky bit: prevents users from accessing other users' files in the directory.
- (15) SQL injections
- ☒ cannot be mitigated securely by obscuring table and column names.
 - ☐ are vulnerabilities in client-side scripts (e.g., JavaScript).
 - ☐ may be exploited only by authenticated users.
 - ☒ may be used by an attacker to gain confidential information.
- (16) Parasitic malware
- ☒ cannot exist independently.
 - ☐ ask for payment in exchange for releasing a victim's files or system.
 - ☐ include worms.
 - ☒ include viruses.
- (17) Which statements are true for user authentication?
- ☒ Online password guessing is more challenging for the attacker than offline guessing.
 - ☐ Inherence factors include hardware tokens (e.g., RSA SecureID).
 - ☒ Multi-factor user authentication uses multiple authentication mechanisms.
 - ☐ Salt values must be stored securely to prevent password recovery attacks.
- (18) Which statements are true for Intrusion Detection Systems (IDS)?
- ☐ Signature-based IDS are trained to detect deviations from a known "normal" behavior.
 - ☐ Anomaly-based IDS are trained to detect samples of known attacks.
 - ☒ False-negative error means failure to detect an actual attack.
 - ☒ Network-based IDS may inspect the header and/or payload of a network packet.

2. Matching Questions

For each question, please fill out each ___ with the letter of the corresponding text or figure. Note that you have to use each letter exactly once in each question.

- (1) Attacks and countermeasures
- | | |
|--------------------------------------|--|
| b injecting and executing shellcodes | (a) salting |
| d eavesdropping wireless networks | (b) address space layout randomization |
| a brute-forcing many passwords | (c) PGP |
| e DDoS | (d) IEEE 802.11i |
| c tampering with e-mail | (e) upstream filtering |

3. Open-Ended Questions

For each question, please clearly indicate your final answer.

- (1) **Code Vulnerability** What software vulnerabilities can you identify in file `search.php`? Briefly explain where (i.e., on which lines) and how they occur!

File `english.php`:

```
0: $text = "Search: ";
```

File `spanish.php`:

```
1: $text = "Buscar: ";
```

File `search.php`:

```
...
2: $language = $_GET['language']; // english.php or spanish.php
3: $search = $_GET['query']; // search keyword
4: if ($language == "english.php") {
5:     include("english.php");
6: } else {
7:     include($language);
8: }
9: echo($text . $search);
10: $results = $db->query("SELECT * FROM posts WHERE text = '%" . $search . "%'");
... // echo results
```

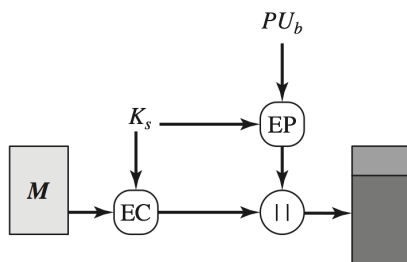
1. file inclusion on line 7 (input originally read on line 2): unfiltered / unvalidated input is used to include file
2. (reflected) cross-site scripting on line 9 (input originally read on line 3): unfiltered / unvalidated input is repeated in the response
3. SQL injection on line 10 (input originally read on line 3): unfiltered / unvalidated input is included as part of an SQL statement

How would you fix `search.php`? (You do not need to write code, just propose ideas and techniques.)

in general, all of these can be prevented through proper filtering, but that is not the best answer everywhere

1. good answer: do not use input directly for inclusion, use input only in condition (e.g., `if` or `case`); okay answer: filter out anything that could lead to the inclusion of unintended files (e.g., remove `..` and `/`), could be done either with blacklist or whitelist (latter is better)
2. good answer: escape special characters (e.g., `|` and `"`); okay answer: filter out special characters and HTML tags (could be either whitelist or blacklist)
3. good answer: prepared / parameterized statements or ORM framework; okay answer: filter out or escape special SQL characters (e.g., `"` and `;`)

- (2) **Pretty Good Privacy: Symmetric and Asymmetric Keys** The following figure shows how Pretty Good Privacy encrypts a message:



Notation:

- M : message
- K_s : symmetric key
- EC: symmetric-key encryption
- EP: asymmetric-key encryption
- ||: concatenation

How and when is K_s generated?

how: randomly by sender; when: before the message is encrypted (different key is generated for each message)

What is PU_b ? How can the recipient decrypt the message?

PU_b is the public key of the recipient; recipient can decrypt the message by first decrypting the symmetric key K_s using its private key and then decrypting the message with K_s