PassQueen
a.k.a dumpsqueen.com

Fortinet NSE 5 - FortiWeb 8.0 Administrator

Fortinet NSE5_FWB_AD-8.0

Version Demo

Total Demo Questions: 5

Total Premium Questions: 52

Buy Premium PDF

https://passqueen.com

support@passqueen.com

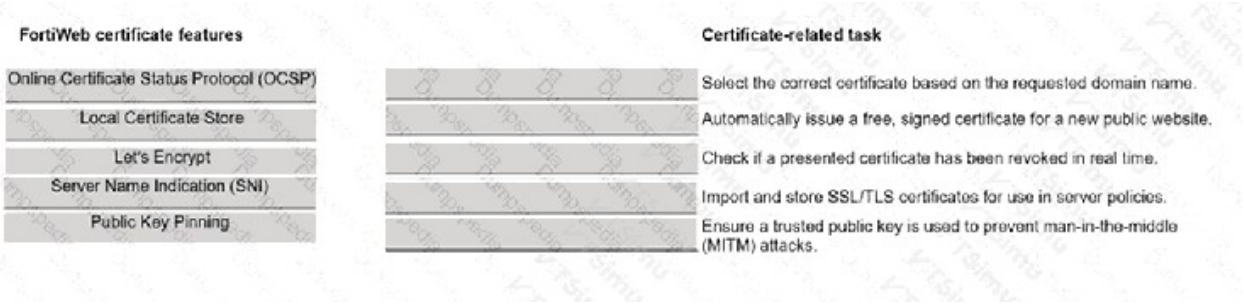
Topic Break Down

Topic	No. of Questions
Topic 1, Deployment and Configuration	18
Topic 2, Web Application Security	17
Topic 3, API Protection	5
Topic 4, Bot Mitigation	8
Topic 5, Authentication and Access Control	3
Topic 6, Troubleshooting	1
Total	52

QUESTION NO: 1 - (SIMULATION)

You are working on securing HTTPS communication across different services using FortiWeb. Your task is to configure and validate digital certificates for various traffic and communication needs.

Match each FortiWeb certificate feature to the certificate-related task that supports the feature.



ANSWER: See the explanation for the answer

Explanation:

Correct matches:

QUESTION NO: 2

You have configured parameter validation, file security, and machine learning (ML) anomaly detection for a web form, but some server-side request forgery tests are still succeeding. You need to advise the team on what to prioritize next to improve SSRF protection without compromising other parts of the application.

Which recommendation would best strengthen FortiWeb's ability to block remaining SSRF attempts?

A. Disable ML anomaly detection and rely solely on parameter inspection.

B. Review and refine input validation logic, as SSRF may be exploiting backend behavior or bypassing weak filters.

C. Offload all server-side request forgery (SSRF) protection to FortiGate and remove FortiWeb from the API flow.

D. Apply HTTPS inspection at the transport layer, which FortiWeb does not use to block SSRF.

ANSWER: B

Explanation:

Review and refine input validation logic, as SSRF may be exploiting backend behavior or bypassing weak filters. is the best next step because SSRF relies on attacker-controlled input being accepted by the application and then used by a server-side component to retrieve a resource. The relevant input can be a URL, hostname, URI scheme, redirect target, IP address, DNS-resolved destination, or a parameter that indirectly selects a backend service. Refining validation lets the team define the expected parameter format and tightly constrain permitted values for the specific form or API function, rather than broadly accepting arbitrary destinations.

For effective SSRF prevention, validation should use an allowlist of approved schemes, hostnames, paths, ports, and, where appropriate, fixed destination values. The application should also normalize and validate the final destination after parsing and redirects, and prevent access to private, loopback, link-local, and cloud-metadata address ranges unless the business function explicitly requires them. FortiWeb parameter validation and request-inspection controls can enforce application-specific constraints at the web-application boundary, while the development team should ensure equivalent server-side validation before outbound requests are made. This focused refinement preserves the existing layered controls while directly addressing the input paths that enable SSRF. See Fortinet's FortiWeb Administration Guide and OWASP's SSRF Prevention Cheat Sheet.

QUESTION NO: 3 - (DRAG DROP)

You are configuring the FortiWeb client-side protection feature to defend against browser-based attacks.

Based on the layered defense strategy, drag and drop each control to the corresponding stage of defense.

ANSWER:

Explanation:

The correct layered placement is Cross-Origin Resource Sharing (CORS) protection in the stage for preventing attacks before they happen, Subresource integrity check in the stage for detecting tampering at runtime, and HTTP header request in the stage for mitigating after the browser is compromised. This ordering reflects how browser-facing controls work together as defense in depth.

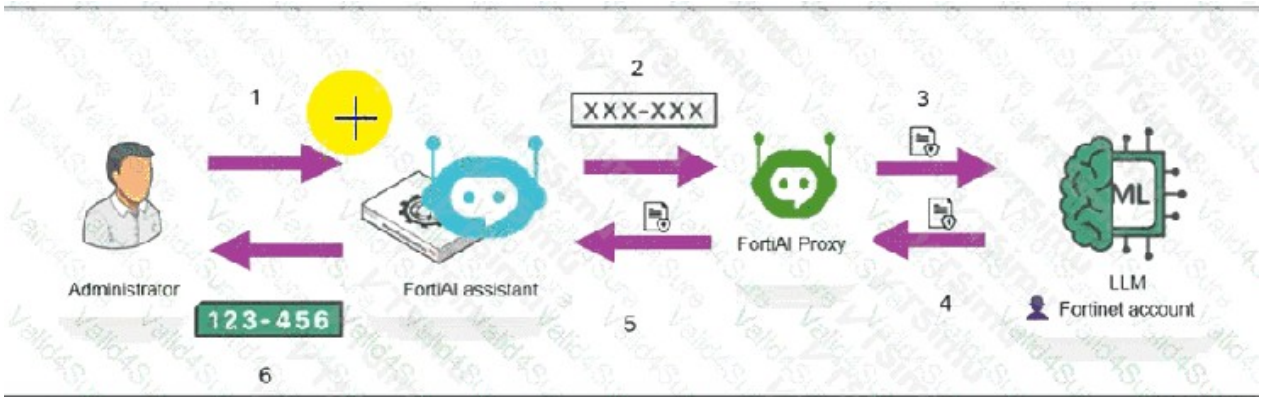
CORS is a browser-enforced access-control mechanism. By defining which origins are permitted to read or interact with application resources, it establishes a trust boundary before a cross-origin browser request can expose protected data to an untrusted site. That makes it an early preventive control. The MDN CORS guide describes CORS as the mechanism through which servers indicate the origins allowed to load resources.

Subresource Integrity is appropriately placed at runtime because the browser evaluates an integrity value when it fetches a referenced script or stylesheet. The resource is checked against the expected cryptographic hash, so a changed external resource can be identified at the point it is loaded. This gives active verification of third-party or externally hosted client-side resources. The MDN Subresource Integrity documentation explains that a browser uses the supplied hash to verify that a fetched resource has not been unexpectedly manipulated.

HTTP header request belongs in the mitigation stage because response-header-based browser controls can constrain browser behavior and limit what malicious code can load, execute, frame, disclose, or connect to after a client-side incident has occurred. These controls help contain consequences and reduce the practical capability of a compromise. Together, the three placements create a sequence of pre-attack access restrictions, in-browser integrity verification, and post-compromise impact reduction.

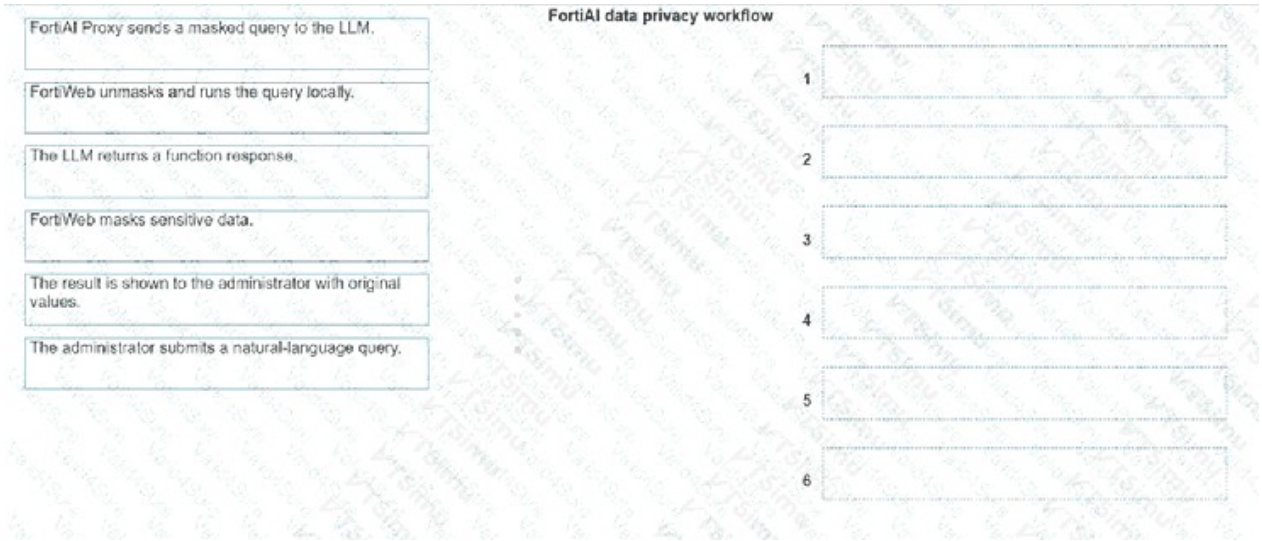
QUESTION NO: 4 - (HOTSPOT)

Refer to the exhibit.



You are a FortiWeb administrator reviewing how FortiAI protects sensitive data when interacting with a large language model (LLM).

Drag each label to the corresponding step in the FortiAI data privacy workflow.



ANSWER:

Explanation:

The displayed sequence correctly represents the FortiAI data privacy workflow. It begins when the administrator submits a natural-language request to the FortiAI assistant. Before any request is sent outside the protected FortiWeb environment, FortiWeb detects and masks the sensitive value. The exhibit illustrates this transformation by changing the local value, such as `123-456`, into a tokenized or masked representation, such as `XXX-XXX`. This lets the request retain the intended context while preventing the original sensitive value from being disclosed to the external model.

FortiAI Proxy then forwards the masked query to the LLM. The LLM can interpret the requested operation and return a function response, but it works only with the masked representation rather than the original protected value. When that response returns, FortiWeb performs the sensitive operation locally: it unmasks the protected value and runs the required query under local control. Finally, the administrator receives the completed result with the original values restored. The sequence therefore preserves the usability of LLM-assisted requests while keeping confidential data from being exposed externally. This follows Fortinet's FortiWeb administration guidance for using FortiAI capabilities and protecting application data; see the FortiWeb 8.0 Administration Guide.

QUESTION NO: 5

An API has an endpoint at `/api/orders`. The application requires normal clients to retrieve and create orders by using GET and POST, but DELETE requests must never reach the back-end servers.

Which two FortiWeb actions should the administrator take to enforce this requirement without blocking the permitted API operations? (Choose two.)

A. Create a URL access rule for `/api/orders` that permits only GET and POST.

B. Enable the URL access control in the protection profile applied to the API server policy.

C. Configure a request-rate threshold for `/api/orders`.

D. Enable URL encryption for `/api/orders`.

E. Add the current client IP addresses to a static blocklist.

ANSWER: A B

Explanation:

Create a URL access rule that matches `/api/orders` and permits only the required HTTP methods. This applies an application-specific restriction rather than a broad restriction across the entire site. The URL access control must also be enabled in the protection profile that is attached to the applicable server policy; otherwise FortiWeb will not enforce the rule for the protected API traffic.

Rate limiting can reduce abusive request volume but does not distinguish DELETE from GET or POST. URL encryption obscures links delivered to clients but does not control allowed methods. A static IP block would be overly broad and would not consistently prevent DELETE requests from other sources.