[Skip to main content](https://lms.alnafi.com/xblock/block-v1:alnafi+DCCS102+2025_DCCS+type@vertical+block@936a126764784304a0ea41674381763a?exam_access=&recheck_access=1&show_bookmark=0&show_title=0&view=student_view#main)

**Summary**

The Session Tokens (Cookie, SessionID, Hidden Field), if exposed, will usually enable an attacker to impersonate a victim and access the application illegitimately. It is important that they are protected from eavesdropping at all times, particularly whilst in transit between the client browser and the application servers.

The information here relates to how transport security applies to the transfer of sensitive Session ID data rather than data in general, and may be stricter than the caching and transport policies applied to the data served by the site.

Using a personal proxy, it is possible to ascertain the following about each request and response:

* Protocol used (e.g., HTTP vs. HTTPS)
* HTTP Headers
* Message Body (e.g., POST or page content)

Each time Session ID data is passed between the client and the server, the protocol, cache, and privacy directives and body should be examined. Transport security here refers to Session IDs passed in GET or POST requests, message bodies, or other means over valid HTTP requests.

**Test Objectives**

* Ensure that proper encryption is implemented.
* Review the caching configuration.
* Assess the channel and methods’ security.

**How to Test**

**Testing for Encryption & Reuse of Session Tokens Vulnerabilities**

Protection from eavesdropping is often provided by SSL encryption, but may incorporate other tunneling or encryption. It should be noted that encryption or cryptographic hashing of the Session ID should be considered separately from transport encryption, as it is the Session ID itself being protected, not the data that may be represented by it.

If the Session ID could be presented by an attacker to the application to gain access, then it must be protected in transit to mitigate that risk. It should therefore be ensured that encryption is both the default and enforced for any request or response where the Session ID is passed, regardless of the mechanism used (e.g., a hidden form field). Simple checks such as replacing https:// with http:// during interaction with the application should be performed, together with modification of form posts to determine if adequate segregation between the secure and non-secure sites is implemented.

Note that if there is also an element to the site where the user is tracked with Session IDs but security is not present (e.g., noting which public documents a registered user downloads) it is essential that a different Session ID is used. The Session ID should therefore be monitored as the client switches from the secure to non-secure elements to ensure a different one is used.

Every time the authentication is successful, the user should expect to receive:

* A different session token
* A token sent via encrypted channel every time they make an HTTP Request

**Testing for Proxies & Caching Vulnerabilities**

Proxies must also be considered when reviewing application security. In many cases, clients will access the application through corporate, ISP, or other proxies or protocol aware gateways (e.g., Firewalls). The HTTP protocol provides directives to control the behavior of downstream proxies, and the correct implementation of these directives should also be assessed.

In general, the Session ID should never be sent over unencrypted transport and should never be cached. The application should be examined to ensure that encrypted communications are both the default and enforced for any transfer of Session IDs. Furthermore, whenever the Session ID is passed, directives should be in place to prevent its caching by intermediate and even local caches.

The application should also be configured to secure data in caches over both HTTP/1.0 and HTTP/1.1 – RFC 2616 discusses the appropriate controls with reference to HTTP. HTTP/1.1 provides a number of cache control mechanisms. Cache-Control: no-cache indicates that a proxy must not re-use any data. Whilst Cache-Control: Private appears to be a suitable directive, this still allows a non-shared proxy to cache data. In the case of web-cafes or other shared systems, this presents a clear risk. Even with single-user workstations the cached Session ID may be exposed through a compromise of the file-system or where network stores are used. HTTP/1.0 caches do not recognise the Cache-Control: no-cache directive.

The Expires: 0 and Cache-Control: max-age=0 directives should be used to further ensure caches do not expose the data. Each request/response passing Session ID data should be examined to ensure appropriate cache directives are in use.

**Testing for GET & POST Vulnerabilities**

In general, GET requests should not be used, as the Session ID may be exposed in Proxy or Firewall logs. They are also far more easily manipulated than other types of transport, although it should be noted that almost any mechanism can be manipulated by the client with the right tools. Furthermore, [Cross-site Scripting (XSS)](https://owasp.org/www-community/attacks/xss/) attacks are most easily exploited by sending a specially constructed link to the victim. This is far less likely if data is sent from the client as POSTs.

All server-side code receiving data from POST requests should be tested to ensure it does not accept the data if sent as a GET. For example, consider the following POST request (http://owaspapp.com/login.asp) generated by a log in page.

POST /login.asp HTTP/1.1  
Host: owaspapp.com  
[...]  
Cookie: ASPSESSIONIDABCDEFG=ASKLJDLKJRELKHJG  
Content-Length: 51  
 Login=Username&password=Password&SessionID=12345678

If login.asp is badly implemented, it may be possible to log in using the following URL: http://owaspapp.com/login.asp?Login=Username&password=Password&SessionID=12345678

Potentially insecure server-side scripts may be identified by checking each POST in this way.

**Testing for Transport Vulnerabilities**

All interaction between the Client and Application should be tested at least against the following criteria.

* How are Session IDs transferred? e.g., GET, POST, Form Field (including hidden fields)
* Are Session IDs always sent over encrypted transport by default?
* Is it possible to manipulate the application to send Session IDs unencrypted? e.g., by changing HTTP to HTTPS?
* What cache-control directives are applied to requests/responses passing Session IDs?
* Are these directives always present? If not, where are the exceptions?
* Are GET requests incorporating the Session ID used?
* If POST is used, can it be interchanged with GET?