[Skip to main content](https://lms.alnafi.com/xblock/block-v1:alnafi+DCCS102+2025_DCCS+type@vertical+block@189ca2fa355f43bc927d3f70ba3a1a0e?exam_access=&recheck_access=1&show_bookmark=0&show_title=0&view=student_view#main)

**Overview**

Web servers usually give developers the ability to add small pieces of dynamic code inside static HTML pages, without having to deal with full-fledged server-side or client-side languages. This feature is provided by [Server-Side Includes](https://owasp.org/www-community/attacks/Server-Side_Includes_%28SSI%29_Injection)(SSI). SSI can lead to a Remote Command Execution (RCE), however most webservers have the exec directive disabled by default.

**How to Test**

To test for exploitable SSI, inject SSI directives as user input. If SSI are enabled and user input validation has not been properly implemented, the server will execute the directive. This is very similar to a classical scripting language injection vulnerability in that it occurs when user input is not properly validated and sanitized.

First determine if the web server supports SSI directives. Often, the answer is yes, as SSI support is quite common. To determine if SSI directives are supported, discover the type of web server that the target is running using information gathering techniques (see [Fingerprint Web Server](https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server)). If you have access to the code, determine if SSI directives are used by searching through the webserver configuration files for specific keywords.

Another way of verifying that SSI directives are enabled is by checking for pages with the .shtml extension, which is associated with SSI directives. The use of the .shtml extension is not mandatory, so not having found any .shtml files doesn’t necessarily mean that the target is not vulnerable to SSI injection attacks.

The next step is determining all the possible user input vectors and testing to see if the SSI injection is exploitable.

First find all the pages where user input is allowed. Possible input vectors may also include headers and cookies. Determine how the input is stored and used, i.e if the input is returned as an error message or page element and if it was modified in some way. Access to the source code can help you to more easily determine where the input vectors are and how input is handled.

Once you have a list of potential injection points, you may determine if the input is correctly validated. Ensure it is possible to inject characters used in SSI directives such as <!#=/."-> and [a-zA-Z0-9]

The below example returns the value of the variable. The [references](https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/08-Testing_for_SSI_Injection#references) section has helpful links with server-specific documentation to help you better assess a particular system.

<!--#echo var="VAR" -->

When using the include directive, if the supplied file is a CGI script, this directive will include the output of the CGI script. This directive may also be used to include the content of a file or list files in a directory:

<!--#include virtual="FILENAME" -->

To return the output of a system command:

<!--#exec cmd="OS\_COMMAND" -->

If the application is vulnerable, the directive is injected and it would be interpreted by the server the next time the page is served.

The SSI directives can also be injected in the HTTP headers, if the web application is using that data to build a dynamically generated page:

Tools

* Web Proxy Burp Suite
* OWASP ZAP