Exploring the World of ESI Injection

Exploring the World of ESI Injection - Sudhanshu Rajbhar, Medium.

- Published: 2023-01-03
- Original: <https://infosecwriteups.com/exploring-the-world-of-esi-injection-b86234e66f91>
- Preserved from: https://infosecwriteups.com/exploring-the-world-of-esi-injection-b86234e66f91 (stored) on 2026-08-09
- Capture timestamp: 20241007231329
- 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.

Exploring the World of ESI Injection

[[image: Sudhanshu Rajbhar]](https://sudhanshur705.medium.com/?source=post_page-----b86234e66f91----------------------------)[[image: InfoSec Write-ups]](https://infosecwriteups.com/?source=post_page-----b86234e66f91----------------------------)

Sudhanshu Rajbhar

Published in

InfoSec Write-ups

Heyyy Everyoneee,

In this writeup I will be sharing my findings related to ESI (***Edge Side Include***) Injection which me and my friend nytr0gen found on a Private bug bounty program. We found a bunch of them so make sure you read it till the last , as you won't miss the most interesting findings in that manner :)

If you aren't aware / haven't already heard about ***Edge Side Include Injection, ***I strongly recommend reading the below articles first , as Gosecure has already done a great job at explaining it and also you can checkout Alex Birsan tweet below to find out how to look for them last but not the least you can also you can watch the DEFCON talk:

Beyond XSS: Edge Side Include Injection - GoSecure ### Update: A new blog post has been published as a follow up to this article : ESI Part 2: Abusing specific... www.gosecure.net

ESI Injection Part 2: Abusing specific implementations - GoSecure ### This post is a follow up with items discovered after the first ESI publication. Those discoveries are attack vectors... www.gosecure.net

The Story Begins

These findings were on a private program so I will be referring to the target as **redacted.com**, to give you some idea about the company I will let you know that it's a very famous sport's organization.

I was looking for xss bugs and soon enough found one endpoint where it allowed `<>'"` characters as they were reflecting as it is in the response but I couldn't get xss there easily due to the *Akamai WAF* in place.

Going through the page source, I noticed a block of code which looked kinda suspicious to me:

I knew about ***ESI ****so instead of focusing on popping an alert box I started checking if the *caching server* will render my inputted esi tags,

I used the above payload and it was blocked by waf, the application had some more strict waf rules in place after further testing I realised that the waf looks for pattern like this `<esi:placeholderhere>` and blocks if it contains any of the valid *ESI *tags such as include, etc . Using just `<esi:>` returned 500 Internal Server Error ,so I knew the caching server was actually trying to render the *ESI *tags, I tried various characters such as %00,%0A,etc to check if any of them allows to bypass the check but nothing worked in the end.

At that time I was using ***Burp Pro Trial***, so I scanned this parameter using it and I received this output:

https://*globe.redacted.com/?showFooter=um6k%3c!--esi--%3e8omf%3c!--esx--%3eekdi*

Note: This issue was generated by the Burp extension: Active Scan++.

Issue detail

The application appears to support Edge Side Includes: The following probe was sent: um6k<!--esi-->8omf<!--esx-->ekd

So I was able to confirm the *ESI injection* using the comment tag, but still not very useful. As I had no further idea to try , I decided to move on.

I then saved the endpoint in my notes and moved on, in hope I will be able to escalate it in future.

A couple of months passed and I received a mail from the company's bbp, they had launched a ***Promotion Event*** and were paying **2x** for all high/crtical submissions.

Seeing this mail, I couldn't leave the endpoint in my notes any longer I needed to somehow escalate it, whatever it takes.

I was pretty sure I wouldn't be able to escalate it on my own as I don't have the required skillset for it so why not contact somebody else who has way more experience than me and would be able to come up with a solution to bypass the waf.

The program allowed ***collaborating*** on reports , so I could just contact someone from the `collaborator` section.

There were many people in this program even some big names were also there , so it was a tough call on who should I contact after some thoughts I contacted ***nytr0gen ,*** I already knew about him

as I have been playing ctfs. He is from *WreckTheLine *ctf team and trust me CTF players skillset are on a whole different level.

Collaboration Magic

In 30 minutes approximately nytr0gen came up with working ESI payload which can be used to popup a xss alert box.

Some more examples for you to understand how ESI tags can be use to bypass WAF:

Let me decode the payload for you to understand what actually happened

I was blown away when I first saw this as I didn't knew you can use variables inside the ESI comment tag also.

https://docs.oracle.com/cd/A97335_02/caching.102/a90372/esi.htm#633138

When the above payload will be rendered by caching server it will return only this on the page source:

`$(QUERY_STRING{countryCode})` returned the `countryCode` parameter value, ESI Tags are really very powerful you can do a bunch of things with them.

Another interesting thing was that the application session cookies were marked as ***httponly *** and it was scope to `.redacted.com ,` as the cookies were marked as ***httponly *** it wouldn't have been possible to steal the session cookies with a normal mere xss but with ESI tags it's possible :)

Well I even asked him how he came up with the idea of using the variables inside ESI comment, he told me he just read the docs

Edge Side Includes (ESI) Language Tags ### This chapter describes the Edge Side Includes (ESI) tags provided for content assembly of dynamic fragments. This... docs.oracle.com

A full blown account takeover POC was provided in the report , the issue got *triaged *and was rewarded with a* high severity* rating (thanks to the 2x multplier):

The story doesn't ends here as it turn out nytr0gen found another subdomain where he had xss upon using ESI tags, there also they were being rendered by the caching server, this was a great signal for us as this might mean that still there might be more of this ESI Injection bugs across the whole target.

This application was doing some wierd uppercase / lowercase transformation of our input so we were not able to use any variable like for eg `HTTP_COOKIE` as it was being transformed into uppercase / lowercase wierdly **http_COOKIE** , still nytr0gen was able to again read the docs and find another useful ESI function which allows to decode url encoded strings:

https://www.akamai.com/site/zh/documents/technical-publication/akamai-esi-developers-guide-technical-publication.pdf

He url encoded the payload 2times to make sure Akamai doesn't blocks it,

Moving on, as now I slowly slowly started to get a grasp of this ESI Injection by reading nytr0gen multiple times. I had already found a xss on another subdomain ,there were no signs of ESI tags in page source anywhere but I still decided to give it a shot `aaa<!--esi-->bbb<!--esx-->ccc ,` use this as the

payload and the server actually rendered it so I knew ESI injection bug was there used the same payload for account takeover

If you are interested in *account takeover POC* here it is (the js code is placed after the hash, I just placed it below to make it more readable):

nytr0gen did an awesome job here making it very easy for the triager to reproduce the actual impact using this poc. As you guys are now already aware what does the `HTTP_COOKIE` and `QUERY_STRING{x}` variables do so it should easy for you to understand the poc.

Btw it turned out that nytr0gen had already submitted this xss bug 10 months ago, we still decided to report it as this had much more impact compared to a normal xss.

Another HIGH severity bug, we are doing good here :)

Let's keep going, the upcoming ESI Injection bugs are super interesting so hold your seats tight, the best is yet to come.

After few days nytr0gen again found another new endpoint which was vulnerable .

Yeah you read that right, the Response `Content-Type` header value for this endpoint was `application/json` , the ESI tags were still rendered by caching server.

Although it was possible to include the ESI tag here, but how would you be able to steal the cookie which is in the page response as xss won't work with `Content-Type: application/json` or is it so `?`

Anyone would have have stopped here thinking that you can't do anything now due to the `Content-Type ,` but you shouldn't be worried when nytr0gen is there to cover your back :)

Here's the answer using the `$add_header()` method you can overwrite any response header.

https://www.akamai.com/site/zh/documents/technical-publication/akamai-esi-developers-guide-technical-publication.pdf

The below payload will change the response `content-type` to `text/html`

And here's the final payload we used to execute js code, we first used `HTTP_COOKIE` then `add_header` and then the `url_decode` function:

The `HTTP_COOKIE` variable returns the cookies in the response, `add_header` function changes the *content type* of the response and finally using `url_decode` function we put our xss payload (url encoded 2 times to avoid waf) using which we can steal the page response.

A great bug deserves a great bounty :)

We thought this must be the end of it and there will be no more ESI bugs, but we were wrong.....

The Story Continueeeees

We found another endpoint , it looked like a proxy which takes an url parameter as input and it then fetches the response of that url.

It only whitelisted few domains some of them were on our target redacted.com and few were on other 3rd party sites.

Due to the way it worked ,we started testing to see if ssrf is possible here. As it only allowed making requests to some whitelist hosts we began looking for any open redirect bug in those hosts.

We were able to find one in redacted.com but it only worked when the user is authenticated, btw this open redirect was a bit interesting.

Upon visiting this url: https://www.redacted.com/auth#login?url=http://evil.com

It loaded another url inside an iframe which then redirect to our url, I tried for xss but it didn't worked as the redirect was happening via the `Location` header.

We then moved onto the 3rd party sites xyz.com which were also in the whitelist, still we didn't find any open redirect which can be useful to us, although we find some xss on these 3rd party sites.

Ssrf looked so close here but still we weren't able to make arbitrary requests.

While playing around with this endpoint, I used the 3rd party site xss vulnerable endpoint in the url parameter and surprisingly the xss popup appeared in the context of redacted.com domain.

We had already reported an ESI Injection on this host before, so I just tried using an ESI payload and it really worked

So by using a xss on a different 3rd party domain we were able to get ESI injection working on our target domain, we couldn't get ssrf as we didn't find any open redirect after spending a couple of days but atleast we managed to escalate this using ESI Injection so we reported this.

We also tried one more thing which I think I should mention, as you already know about the `add_header` function of ESI, we thought we can create our own open redirect bug with a payload like this:

Using this payload on any of the previous endpoint which we found vulnerable to ESI injection will give us a sweet and simple open redirect.

It was working normally but when we used this open redirect url on the ***proxy *** endpoint, it didn't worked for some reason maybe Akamai waf or something we didn't knew.

And here comes the twist in the story, the bug was marked as duplicate by the *h1 triager , *we both were like wtf that's impossible.

If this was actually duplicate that would mean that there is someone else in this program who also knows about this ESI magic or is to so?

After further back and forth with the *h1 triager , *we got to know that the original reporter actually was able to escalate it to perform SSRF but without the ESI trick obviously.

We forgot about this report and moved on. Soon enough the *promotion *came to end and we decided that we will also stop for now and if in future they did the same 2x promotion we will look back at this, which we thought had very less probability.

Another Promotion

Soon enough after 4–5 months they again launched the same 2x multiplier promotion and we were ready to hunt those ESI injection bugs again

I tested out some *callback *parameters as they usually reflect the input in response and found some which even allowed `<>` characters but the caching server configuration for such hosts were different so the ESI tags didn't worked.

Some of old reports were fixed already , nytr0gen found that the same json api endpoint was now acccessible over a different path. To fix the original json api ESI bug they completely removed the endpoint but the same endpoint existed on another path so the bug was still there, using the same poc as before we were able to reproduce the bug.

Submitted it as the endpoint was different although it had the same api, triaged by the program and again rewarded as HIGH.

The team even asked us a question this time regarding mitigation :)

I am going to talk about the last ESI bug we found, this one is the best so many cool ways we came up with to actually to come up with a working exploit.

If you stayed this long to the blog thanks a ton, be ready as I am going to tell you about the best ESI bug we found.

I found another endpoint , the `Content-Type` for this was `application/json`

In the above screenshot you can see that the locale parameter value is reflected in the source, but the ESI tags didn't get rendered by the caching server. As we had already found an ESI Injection bug here before, this should have worked.

I send this url over to nytr0gen and according to him also this should have worked. After some time he messages me saying that he got it working but it's not useful.

In this screenshot you can see that ESI tags worked but can you spot the changes made to this request?

A trained eye would have already spot the url encoded `.` in the url `%2e`

The **Backend server** decoded the `%2e` in the url and returned the same response , the **Caching Server** didn't ignored the `%2e` so it didn't treated it as a *static file* anymore and hence it allowed the ESI tags to work.

It was working but sadly this isn't exploitablable :(

If you send this url to an victim, when he loads this url . The browser will auto decode the `%2e` , so a request is made to the original endpoint not the one we wanted it to thus the ESI tags don't get rendered.

We spent some more time on this but couldn't figured it out how to make it work on the browser also.

Left this endpoint and moved to finding some more endpoints to test.

After a whole month passed, I was testing another program where I had a potential client side path traversal bug for eg suppose in the below js code you had control over the `key` variable.

There was also a jsonp callback endpoint on the same host, so I was trying to trying to load it instead of the `/api.js` file, which will allow me to popup an alert

I was trying to do something like this , but the `key` variable value was taken from the query parameter so I couldn't use the `#` symbol .

As if I put this `#` in the parameter value, it will be consumed by the browser and won't be passed along in the `key` variable value.

If I urlencode %23 it then the browser while fetching the js file doesn't auto decoded it, so I was stuck here. I also couldn't use `?` to ignore the `/api.js` part due to this part of the code.

Inside the `getAllUrlParams` method noticed the first line `url.split('?')[1]` , so if I use ? in my url parameter value this would happen

The `?` will get completely ignored :(, due to the split method.

At that time I thought maybe if I load the url with the url encoded # inside an iframe , it will work. But sadly it didn't worked.

I though of using the same idea on the *Redacted target* also, if you are not short tempered remember this endpoint?

What if I load this url in an iframe would it work? Let's find out

It did actually work :)

When I tried to load the same code in Chrome browser it didn't worked :(

`%2e` was automatically decoded by Chrome

So wierdly this worked in Firefox but not in Chrome, I later confirmed that it also worked in the Safari the same way as Firefox.

I was so happy that it was finally working, I quickly contacted nytr0gen to give him the good news

Onto the POC part now, when I used the `HTTP_COOKIE` , I noticed that all the cookies weren't present there mainly the one used for session purpose. Asked ntr0gen about it and it turns out it was due to the page being loaded inside an iframe and the parent was of different origin.

So instead we used another xss (created using one of our old ESI Injection bug) and use it to iframe this `.json` url and voilla this time the cookie appeared.

When I tried using the `add_header` method to change the `application/json` content type to `text/html` the waf blocked the request, after shortening I found that the waf now was blocking `text/html` keyword . This additional rule might have been set from the fix of our previous report so I needed to comeup with a bypass.

Upon fuzzing I found that `%09` tab character (`\t`), was converted to `t` by the server.By using the below payload it was possible to change the `Content-Type` and then using xss steal the session cookie.

This wasn't rewarded with 2x multiplier because the asset in which we found the bug was not part of promotion and we agreed upon it as they still paid it as a HIGH severity even though it only worked in Firefox not in Chrome.

Another thing happened here at the last, the program manager asked us to stop testing further for ESI Injection bugs as they were going fine tune their WAF.

Finally we have come to an end , I hope you didn’t get bored and enjoyed reading it :) That’s all, thankyou very much for reading it till the last. Hope you would have enjoyed it.

Goodluck for 2023

Sya Everyoneeee
