All.Net Analyst Report and Newsletter

Welcome to our Analyst Report and Newsletter

A note on the Javascript confinement problem

I asked my AI to ’add javascript to translate the text on the page into upside down appearance and use onmouseover to any p or value change the whole page to have different the color, font, fonts size, bullet type, and appearance’ (see The Resulting Script after the Conclusions).

Of course it did it, and it will do lots of other stuff you likely do not want. But the problem is not the AI doing what I asked it to do. That was just easier than looking stuff up. The problem was (and is) what to do about it.

Warning! technical minutia ahead.

JS ways of confining things

Apparently the programmers in the JS world think that the goal of confinement is to allow modularization of programs from their environment. But they do not seem to mind corrupting the environment from within the modules. In particular, when you place a JS program within a Web page, or change the ‘style’ within a Web page, it effects the whole page. While you would think there would be a way to confine it to a region of the page, like a ‘div’ or some other displayed area, sorry…

Except…

There is, it turns out, a heavy weight way to accomplish this. You can use an iFrame, which is effectively an entire Web page confined to an area within another Web page. But iFrames are heavy weight, and accessing internal content is more than just a pain…

If you want to try to control (read, write, and actively work selectively with) one or two iFrames within a Web page, it’s reasonably doable. But try it for anything complicated, like a Web page with a few hundred different components that can come from combinations of internal and external sources; or imports and exports and cuts and pastes, and drags and drops, and you will start to suffer massive slow-downs, memory overruns, and other similar challenges.

So how does this likely effect you?

Here’s the problem. Any time you paste, drop, or otherwise put content into an area of a Web page as HTML (that is interpreted by the application / browser), unless that part of the page somehow confines the content to prevent altering the rest of the page, each part of the page can effect all the others. While making things look strange across the page is not a big problem, the problem of confinement can be much bigger. Here are some of the things that might happen…

- Things you should see could be made invisible or covered up by other things
  - Example: the ‘Yes/No’ or ‘Allow/Deny’ buttons in an application could be covered up with content making ‘Yes’ look like ‘No’ and ‘No’ look like ‘Yes’ and of course ‘Allow’ look like ‘Deny’ and ‘Deny’ look like ‘Allow’
- All of your keystrokes and mouse movements and cuts and pastes, and perhaps camera(s) and microphone(s), etc. could be captured, altered, and made available to the Web site(s) you are connected to.

    - It’s not just your passwords we will be collecting. We will be changing the things you enter, perhaps in ways you cannot see by looking at them. With AI...
- Suppose your system starts flipping out. I mean in all sorts of ways.
    - Screen maximization, seemingly full takeover of your system, till you regain control, or do you? The flashes of light you see may not be your own. And the sounds emitted may order your devices to do things you don’t want them to do, at frequencies you cannot even hear.

So how do we stop it?

I have taken what is possibly the stupidist approach possible, after exhausting all the others. The details are provided under the “Safer Mode Code” at the end of this article. After spending however many hours and days trying different things I found on the Web (much of the factual information being incorrect about how things actually work), I finally settled on the most ridiculous way to do things. I intercepted the relevant calls and made my code replace the bad things with things that made them go away.

Just to be clear, I don’t know that I found all the bad things that can happen… but I think I found enough so that the AI-generated versions of bad things are unlikely to come up with any. In essence, there are only 2 ways code can get executed in HTML, and only one way for styles to propagate upward in the domain tree.

- <style... seems to be the only way a style can be added or changed upward in the DOM tree. You can still use styles inline of course, but they seem to be local to the element the inline is included in. So be not allowing <style… I can eliminate that, but of course it means the hierarchy of styles from whatever the rest of the page is will be inherited unless overridden inline..
- <script… seems to be one way you can add javascript from HTML and of course eliminating that gets rid of a lot of the bad stuff – and the good stuff… but I don’t care to keep the good stuff at the risk of having the bad stuff...
- onWHATEVER is the inline way scripts get to run. For example,

  <div onmouseover=”script stuff “>whatever</div>

  will run the script stuff whenever the mouse passes over the area of the whatever, which by the way could be the entire window area if I want.

I replaced <script… with <div style=display:none> and</script… with </div>, did the same with onWHATEVER changing it to noWHATEVER (onmouseover becomes nomouseover), and it seems to cure the problem… except of course it disables lots of stuff you might want.

Conclusions

The real problem is the lack of sensible confinement mechanisms in the most popular operational code component of Web technology. They fail to reasonably address ‘security’ issues. They actually had some mechanisms to limit the scope of javascript and styles but deprecated the lightweight ones in favor of a heavyweight one - iFrames. Someone in the programming world should someday learn something about security…

Nah… Then I would have nothing to do...

The resulting script

```
<style>
body {  transform: rotate(180deg); /* Upside down text */}

p, span {  cursor: pointer;}

p:hover, span:hover {  font-family: 'Arial Black', sans-serif; /* Example font */
  font-size: 2em; /* Example font size */
  color: blue; /* Example color */
  list-style-type: square; /* Example bullet type */
  background-color: lightgreen; /* Example background color */
}

/* Style for the span to handle mouseover/mouseout separately */
span {  background-color: white;
  padding: 5px;
  border: 1px solid black;
}

span:hover {  background-color: yellow;}
</style>

<script>
document.addEventListener('DOMContentLoaded', function() {
  const elements = document.querySelectorAll('p, span');
  elements.forEach(element => {
    element.addEventListener('mouseover', function() {
      document.body.style.fontFamily = 'Impact, Charcoal, sans-serif';
      document.body.style.fontSize = '1.5em';
      document.body.style.color = 'purple';
      document.body.style.backgroundColor = 'lightyellow';
      document.body.style.listStyleType = 'disc';
    });
    element.addEventListener('mouseout', function() {
      document.body.style.fontFamily = 'Arial, sans-serif';
      document.body.style.fontSize = '1em';
      document.body.style.color = 'black';
      document.body.style.backgroundColor = 'white';
      document.body.style.listStyleType = 'none';
    });
  });
});
</script>
```

Safer mode code

```
document.addEventListener('paste', function(e) {e.preventDefault(); // Prevent the
default paste behavior
        let pastedText = (e.clipboardData || window.clipboardData).getData('text');
// Get the pasted text from the clipboard
        let modifiedText = FixString(pastedText); // Modify the pasted text
        e.target.innerHTML = modifiedText;});
document.addEventListener('drop', function(e) {e.preventDefault(); // Prevent the
default drop behavior
        let data=e.dataTransfer.getData("Text")
        let modifiedText = FixString(data); // Modify the dropped text
        e.target.innerHTML = modifiedText;});

// ============= DANGER MODE
let DangerMode=0;
/* Per https://www.w3schools.com/tags/ref_eventattributes.asp */
let OnList=["onmousedown", "onmouseup", "onclick", "ondblclick", "onmousemove",
"onchange", "onmouseover", "onmouseout", "onwheel", "onabort", "oncanplay",
"oncanplaythrough", "oncuechange", "ondurationchange", "onemptied", "onended",
"onerror", "onloadeddata", "onloadedmetadata", "onloadstart", "onpause", "onplay",
"onplaying", "onprogress", "onratechange", "onseeked", "onseeking", "onstalled",
"onsuspend", "ontimeupdate", "onvolumechange", "onwaiting", "ontoggle", "ondrag",
"ondragend", "ondragenter", "ondragleave", "ondragover", "ondragstart", "ondrop",
"onscroll", "oncopy", "onpaste", "oncut", "onbeforeprint", "onafterprint",
"onbeforeunload","onerror", "onfocus", "onblur", "oncontextmenu", "onfocus",
"oninvalid", "onresent", "onsearch", "onselect", "onsubmit", "onkeydown",
"onkeypress", "onkeyup", "onhashchange", "onload", "onmessage", "onoffline",
"ononline", "onpagfehide", "onpageshow", "onpopstate", "onresize", "onstorage",
"onunload", "oninput"];

function FixString(S)
        {if (DangerMode == 1) {return(S);}
        let ResultString=S.replaceAll(/<style/gi,"<div style='display:none;'");
        ResultString=ResultString.replaceAll(/<\/style/gi,"</div style");
        ResultString=ResultString.replaceAll(/<script/gi,"<div
style='display:none;'");
        ResultString=ResultString.replaceAll(/<\/script/gi,"</div script");
        for (i=0;i<OnList.length;i++) {let from=OnList[i];let
to=from.replace(/on/i,"no");from=new
RegExp(from,"gi");ResultString=ResultString.replaceAll(from, to);}
        return(ResultString);}
```