

How do I Content Security Policy?

Kirk Jackson, RedShield
OWASP NZ Day 2019
22 Feb 19



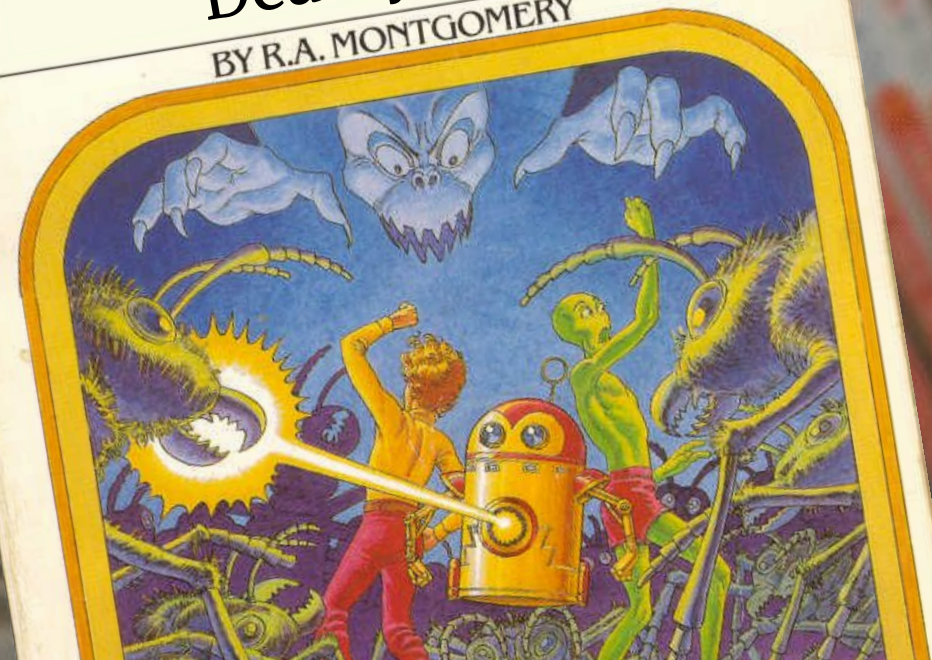
23661-X * \$1.95 * A BANTAM BOOK

CHOOSE YOUR OWN ADVENTURE®  **25**

YOU'RE THE STAR OF THE
STORY! CHOOSE FROM 28 POSSIBLE ENDINGS

The Attack of the Deadly XSS!

BY R.A. MONTGOMERY



Beware and warning!

This book is different from other books.

You and YOU ALONE are in charge of what happens in this story.

There are dangers, choices, adventures and consequences. YOU must use all of your numerous talents and much more of your enormous intelligence.

The wrong choice could end in disaster – even death.

But don't despair. At anytime, YOU can go back and make another choice, alter the path of your story, and change its result.

Turn to page 4.

The beginning

You are an intrepid web developer named Justice, aiming to protect your application as best as you can.

Armed with your trusty list of websites, you begin your quest to rid the world of XSS!

- Javascript was invented in 1995
- Cross-site scripting was invented shortly after

Do you understand XSS? Go to Page 9.

Want to learn more? Go to Page 5.

XSS is everywhere

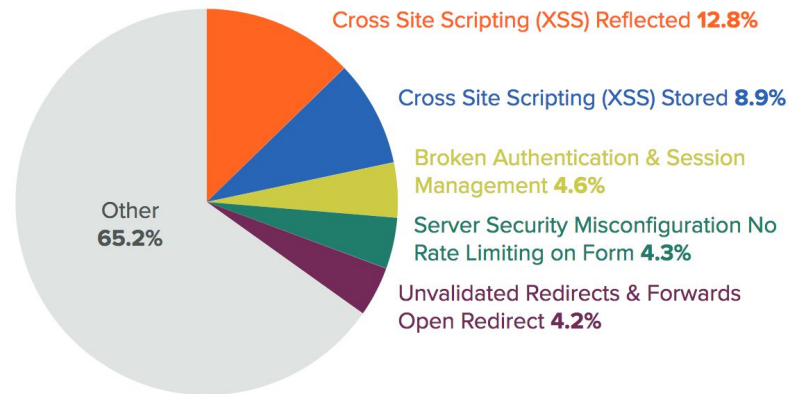
60% of bug payouts by Google

21.7% of bugs found on Bug Crowd
are cross-site scripting

Almost all sites have XSS*

Injected javascript can do *anything*
that your users can do

(* Source: anecdotal)



Bug Crowd's 2018 State of Bug Bounty report

Go to [Page 6](#).

HTML = code + data

A single HTML page mixes both code and data

It's all jumbled together

The browser doesn't know who wrote the HTML:

- The site creator?
- The end user?

```
<html>
<body>
<h1>Hello!</h1>
<script>
    var urchin_id = "61143";
</script>
</body>
</html>
```

Go to Page 7.

An attack

User enters their name:

Full Name

The page renders the name in an HTML context

```
<html>
<body>
<h1>
    Hello
    <script>
        alert("Hello!");
    </script>
</h1>
</body>
</html>
```

Go to Page 8.

An attack

The user's data can be inserted into many different contexts on a page:

- HTML element
- HTML attribute
- URL query parameter
- CSS value
- Javascript value
- ...
- or a combination of the above

```
<h1>Hello <%= name %></h1>
<input value="<%= name %>">
<a href="/?name=<%= name %>">
<style> h1 {color: <%= name %>; }
<script>
    var name='<%= name %>';
</script>
```

Go back to Page 4.

Fixing XSS

Fix the output:

Understand the context that you're outputting data:

- HTML
- URL
- Attribute
- Javascript
- ...

Encode data to make it safe in that context

Or, fix the input:

Restrict the input to your application to only safe characters

- Validation
- Whitelists

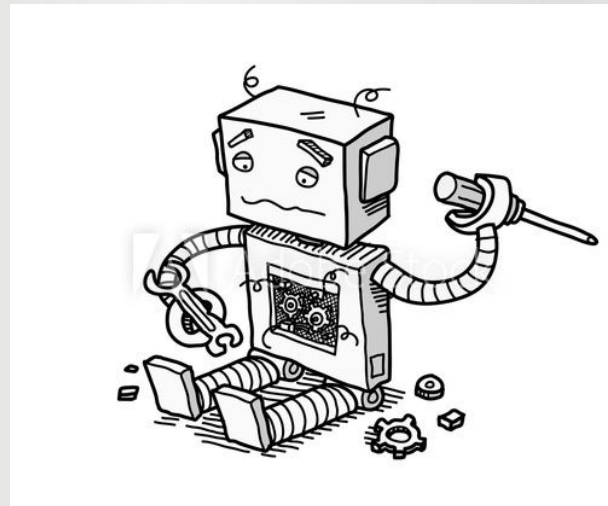
If you've fixed all your XSS, goto Page 11.

If you've got known XSS to fix, go to Page 10.

Keep on fixing!

Justice realises that CSP won't help them unless they fix all their known XSS first.

Across the kingdom, other browsers reign... and not all of them bow to the majesty of CSP.



Once you've finished fixing, go to Page 9.

A Defensive Mindset

Start by fixing all your known weaknesses

Layer on additional defences

Note: Each individual defence doesn't have to be perfect!



Go to Page 12.

You are ready for Content Security Policy!

Justice, you are ready for the
adventure of a lifetime. Riches and
rewards are ahead.

Do you have what it takes?



To learn CSP, go to Page 13.

To stop now, go to Page 37.

Content Security Policy

The goal:

Prevent execution of untrusted scripts*

How:

Separate code from data

Separate *your code* from the *attackers data*

Set an HTTP header to tell the browser what to do
(*CSP also does other things)

To get the maximum benefit from CSP, you will need to modify your application.

Rank	Site / App URL	Login?	Sensitive data?	Source code?	Active devt?
3	www.example.com	✗	✗	✓	✓
4	blog.example.com	✗	✗	✓	✗
1	finance.internal.example.com	✓	✓	✗	✗
2	shop.example.com	✓	✓	✓	✗

Building a brand new project? Go to Page 28.

If you have the source code and an active development team, go to Page 29.

If you don't, go to Page 14.

URL Whitelist Approach

Young traveller, you have worked hard but alas, you don't have the source code nor the ability to change your application.

You must use a URL whitelist to choose which locations are allowed to provide javascript. Anything more advanced requires the aid of a magician.

Set the HTTP header yonder, and instruct the browser henceforth.

Content-Security-Policy:

```
script-src 'self' google.com;  
object-src 'none';  
base-uri 'none';
```

If you would like to learn about script-src whitelists, go to [Page 17](#).

If you would like to learn about object-src and base-uri, go to [Page 15](#).

To proceed to more advanced topics, go to [Page 24](#).

Flash is dead!

A bolt of lightning comes down from the heavens above, and hits the remaining Flash apps.

Flash and Java Applets could be used to perform XSS attacks.

Luckily, they're dead.

Make sure you tell the browser you're not using them!

Content-Security-Policy:

```
script-src 'self' google.com;  
object-src 'none';  
base-uri 'none';
```

To learn about base-uri, go to Page 16.

All your base...

Take off every ZIG!!

The base tag is not used often. It changes the url of every relative path on your site.

```
<base href="http://attacker.com">  
<script src="/jquery.js">
```

If the attacker can inject a <base>, they can cause your javascript to be loaded from their site:

```
<script  
src="http://attacker.com/jquery.js">
```

Content-Security-Policy:

```
script-src 'self' google.com;  
object-src 'none';  
base-uri 'none';
```

Go back to Page 14.

script-src

Tell the browser where it is allowed to load scripts from:

- host-source
 - name, IP (optional: *, scheme, port)
- scheme-source
 - http:, https:, data:
- 'self'
 - The current website origin
- 'unsafe-inline'
- 'unsafe-eval'
- 'none'

Content-Security-Policy:

```
script-src <source> <source>;
```

```
script-src google.com 1.2.3.4  
*.example.com;
```

```
script-src 'self' https: data: ;
```

Go to Page 18.

Which external hosts do you use?

Alas, Justice – the All-Knowing Oracle does not know with whom your website communicates.

You could ask every knave in the land.

Or, you could consult the mighty wizard known to all as “Report URI”

```
Content-Security-Policy-Report-Only:  
  script-src 'none' 'report-sample';  
  object-src 'none';  
  report-uri https://xyz.com/csp;
```

This will report every use of script or plugins to your reporting url.

Report-only mode means users won't be blocked.

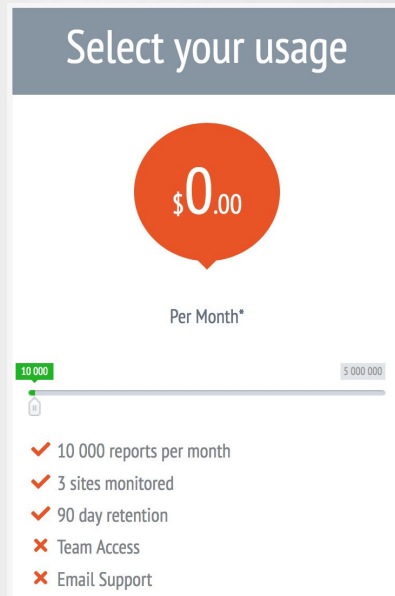
If you would like to see how report-uri works, go to Page 19.

Otherwise go to Page 24.

Report-uri.com

Report-uri.com is a tool to collect CSP reports (and more!)

It is written by @scotthelme, and while it's commercial it has a free tier



The screenshot shows a pricing selection interface. At the top, a dark blue header reads "Select your usage". Below this, a large orange circle contains the text "\$0.00". Underneath the circle, it says "Per Month*". A progress bar is visible, with a green segment on the left labeled "10 000" and a grey segment on the right labeled "5 000 000". Below the progress bar, there is a list of features with checkmarks and crosses:

- ✓ 10 000 reports per month
- ✓ 3 sites monitored
- ✓ 90 day retention
- ✗ Team Access
- ✗ Email Support

Go to [Page 20](#).

Policy #1

Create an account on
report-uri.com

Go to the CSP > Wizard and create
an entry for your site

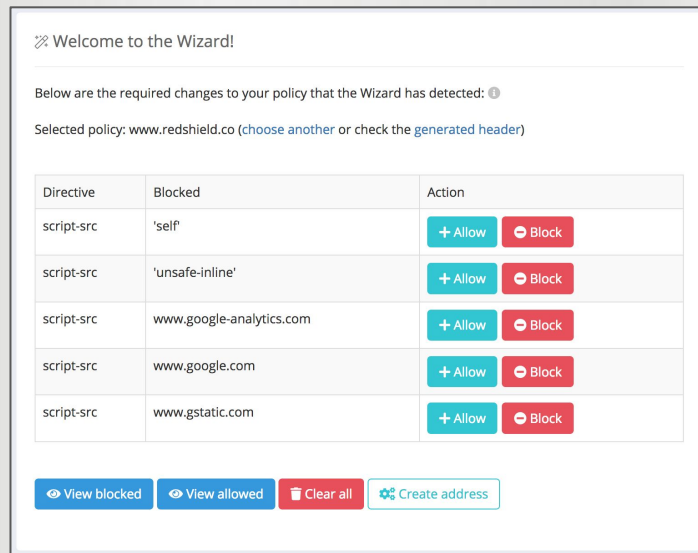
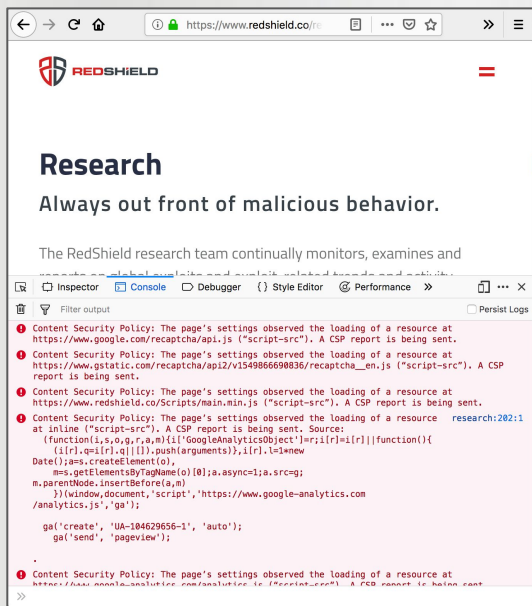
Add a header to your site

```
Content-Security-Policy-Report-Only:  
  script-src 'none' 'report-sample';  
  object-src 'none';  
  report-uri  
  https://redshield.report-uri.com  
  /r/d/csp/wizard;
```

Go to Page 21.

Browse around the site

Use the website for a while, and notice the violations being triggered by the browser



Go to Page 22.

Policy #2

Take the suggestions and add to your policy.



☰ My CSP policies

www.redshield.co  Used for wizard  

```
script-src 'self' 'unsafe-inline' www.google-analytics.com  
www.google.com www.gstatic.com
```

 + Add new

```
Content-Security-Policy-Report-Only:  
script-src 'self' 'unsafe-inline'  
www.google-analytics.com  
www.google.com  
www.gstatic.com  
'report-sample';  
object-src 'none';  
report-uri  
https://redshield.report-uri.com  
/r/d/csp/wizard;
```

*If you haven't browsed the whole site
yourself yet, go to Page 21.*

When you have a stable policy, go to Page 23.

Go live!

Put your CSP policy onto your production site, and see if anything new comes in to the wizard.

Rinse and repeat.

Once it's stable, consider switching from Report-Only mode to blocking.



```
Content-Security-Policy-Report-Only:  
  script-src 'self' 'unsafe-inline'  
    www.google-analytics.com  
    www.google.com  
    www.gstatic.com  
    'report-sample';  
  object-src 'none';  
  report-uri  
    https://redshield.report-uri.com  
    /r/d/csp/wizard;
```

You now have a whitelist-base CSP policy!
Return to Page 14.

What can go wrong?

You want to avoid the following entries in your whitelist:

http: https: or data:

These allow any url to be injected as a script source.

'unsafe-inline'

Allows inline <script>, javascript: urls, onclick handlers

'unsafe-eval'

Allows eval() and similar injections.

You might need to re-write some parts of your app to remove:

<script>...</script>

Go to Page 26.

Oh no!

You have walked into the slaverling
fangs of a lurking grue!

25



Go to Page 37.

The host-est with the most-est

You are whitelisting urls, and saying you trust all javascript from them.

However, it's common for CDNs to have unsafe javascript libraries or JSONP endpoints.

Do you really trust all of these hosts?

```
script-src 'self' 'unsafe-inline'  
          www.google-analytics.com  
          www.google.com  
          www.gstatic.com
```

```
<script  
src="https://www.googleadservices.com/  
pagead/conversion/1070110417/wcm?callb  
ack=alert(1337)"></script>
```

Go to Page 27.

Defense in depth?

A whitelist-based CSP policy isn't perfect, but it's better than nothing

If you don't have control of the app, this may be the best you can do

Factoring out evals and inline `<script>` will still be hard

If you need to rewrite parts of your app, you should aim for "Strict CSP" instead



*It's time to find out how to be strict.
Go to Page 13.*

Green fields?

If you're starting from scratch, it's the perfect time to adopt Strict CSP

You'll have to:

- Use a templating language that can insert nonces
- Remove inline event handlers
- Avoid eval



To learn about Strict CSP, go to Page 29.

See your other choices. Go to Page 13.

“Strict CSP”

Coined by Google researchers, Strict CSP avoids the pitfalls with whitelists

Use a random “nonce” to mark which parts of a page were written by you

Attacker parts stand out and are blocked

```
<script nonce="NDIK"  
    src="/lib.js">  
</script>
```

```
<script nonce="NDIK">  
    foo()  
</script>
```

```
<script>alert(1)</script>
```

**NO
NONCE-SENSE!**

Go to Page 30.

Generate a nonce

Server generates a random value and sends in the CSP header

Use a templating language to insert the nonce into your HTML

The nonce should be 128+ bits of cryptographically random data!

Content-Security-Policy:

```
script-src 'nonce-<%= mynonce %>'
```

...

```
<script nonce="<%= mynonce %>"  
    src="/lib.js">  
</script>
```

```
<script nonce="<%= mynonce %>">  
    foo()  
</script>
```

Go to [Page 31](#).

But wait!

Anything you mark as 'safe' with a nonce is now your responsibility

You have to make sure lib.js is safe from XSS

You have to be careful not to insert untrusted script inside your tags

```
<script nonce="NDIK"  
    src="/lib.js">  
</script>
```

```
<script nonce="NDIK">  
    foo()  
</script>
```

If your lib.js includes other scripts, you'll want to go to Page 32.

Otherwise, go to Page 33.

strict-dynamic

Any script library that itself inserts another script library into the page will break. It won't know the nonce!

The strict-dynamic directive allows the execution of scripts dynamically added to the page, as long as they were loaded by a safe, already trusted script.

Supported in CSP3+. Fallback mechanisms in older browsers.

Content-Security-Policy:

```
script-src 'nonce-<%= mynonce %>'  
        'strict-dynamic'
```

...

See <https://csp.withgoogle.com> to understand the nuances.

Go to Page 33.

Inline handlers must go!

Inline event handlers must be removed from your application

```
<a href="javascript:linkClicked()">foo</a>
```

```
<tag onerror="..." >
```

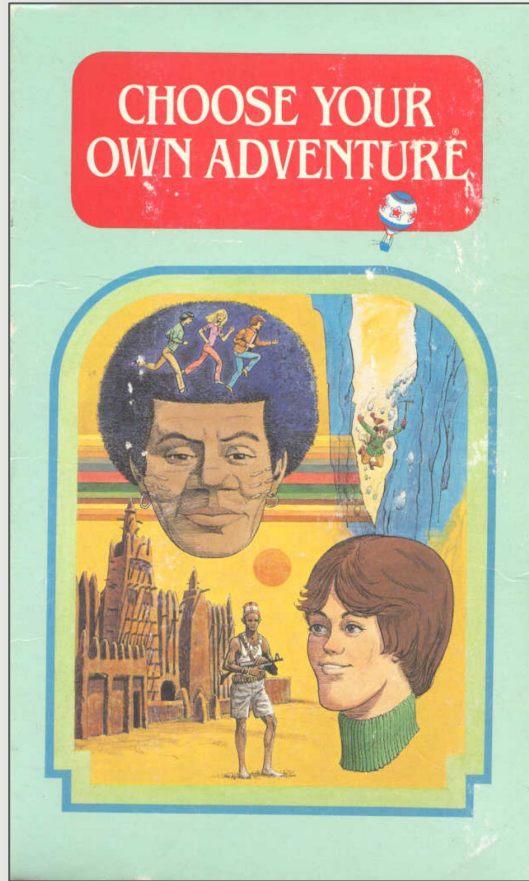
```
<tag onclick="..." >
```

They can be replaced with script block with a nonce.

```
<a id="foo">foo</a>
```

```
<script nonce="<%= mynonce %%">
document.addEventListener(
  'DOMContentLoaded',
  function () {
    document.getElementById('foo')
      .addEventListener(
        'click', linkClicked);
  });
</script>
```

*You now have a Strict CSP policy!
Go to Page 34.*



No source

CSP Whitelist

Active
development or
brand new

Strict CSP with
nonce

Go to Page 35.

> inventory

Every adventurer needs tools of the trade. Carry these in your inventory.

CSP Mitigator

Chrome plugin - A fast and easy CSP deployment analysis tool

CSP Evaluator

Check if a Content Security Policy serves as a strong mitigation against cross-site scripting attacks

Report-uri.com

Know what's happening on your site. A Wizard to help develop your policy.

<https://csp.withgoogle.com>

MDN CSP pages

Go to Page 36.

Well done Justice!

You have completed your quest, and arrived safely at your destination.

You have vanquished all the XSS as far as the eye can see, and have fortified your sanctuary to make it hard for XSS to return.

Rest well, knowing you have made the world a better place.

- XSS
- CSP
- Whitelist approach
- Strict CSP
- Tools

Rank	Site / App URL	Login?	Sensitive data?	Source code?	Active dev?
3	www.example.com	✗	✗	✓	✓
4	blog.example.com	✗	✗	✓	✗
1	finance.internal.example.com	✓	✓	✗	✗
2	shop.example.com	✓	✓	✓	✗

Go to Page 37.

Fin.