CONTENT SECURITY POLICY

Many thanks to this awesome tutorial http://www.html5rocks.com/en/tutorials/security/content-security-policy/ for some of the inputs in this class

CONTENT SECURITY POLICY

Modern browsers provide a **new defense methodology** which can significantly reduce the risk, impact and exposure of websites to XSS

CONTENT SECURITY POLICY

This is via a header called the

Content-Security-Policy

CONTENT SECURITY POLICY

Content-Security-Policy

Headers are sent from the server to the client with every response

And from the client to the server with every request

CONTENT SECURITY POLICY

Content-Security-Policy

Headers contain **metadata** about the request or the response

CONTENT SECURITY POLICY

Content-Security-Policy

The main issue behind XSS is the browser's **inability to determine** whether code is malicious or not

CONTENT SECURITY POLICY

The main issue behind XSS is the browser's **inability to determine** whether code is malicious or not

The browser **assumes** that all code embedded in the HTML, CSS, JS in a site is trusted code - it is **willing** to download and execute **any code** requested by a page

CONTENT SECURITY POLICY

The main issue behind XSS is the browser's **inability to determine** whether code is malicious or not

The browser **assumes** that all code embedded in the HTML, CSS, JS in a site is trusted code - it is **willing** to download and execute **any code** requested by a page

The Content-Security-Policy header is a way to tell the browser what sites you trust - a **whitelist** of trusted sites

CONTENT SECURITY POLICY

The main issue behind XSS is the browser's **inability to determine** whether code is malicious or not

The browser **assumes** that all code embedded in the HTML, CSS, JS in a site is trusted code - it is **willing** to download and execute **any code** requested by a page

The Content-Security-Policy header is a way to tell the browser what sites you trust - a **whitelist** of trusted sites

Instead of blindly rendering everything a page requests - the browser now has **rules** on what can be trusted!

CONTENT SECURITY POLICY

This only works for browsers which **support** the Content-Security-Policy header - all modern browsers do!

CONTENT SECURITY POLICY

Example6-XSS-contentSecurityPolicy.php

CONTENT SECURITY POLICY

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

The Content-Security-Policy header is made up of directives and source expressions

CONTENT SECURITY POLICY

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

The **script-src** is the directive which indicates trusted sources for scripts

CONTENT SECURITY POLICY

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

All sources from the same domain i.e. ‘self’ are trusted

CONTENT SECURITY POLICY

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

As are APIs from apis.google.com

These are the source expressions

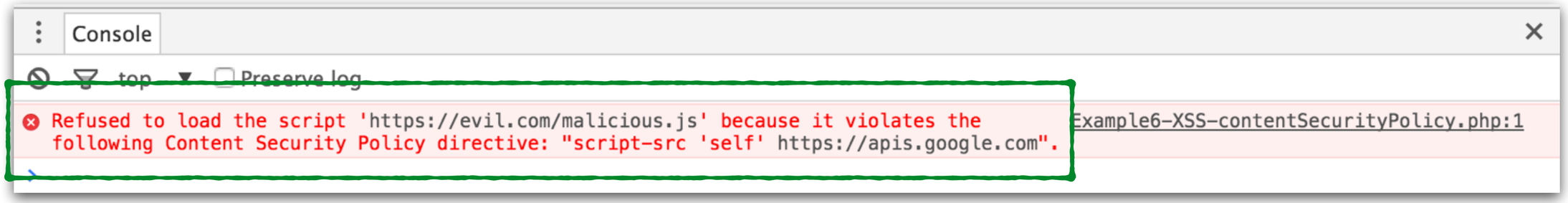
CONTENT SECURITY POLICY

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

The browser will reject any script which does not belong to these trusted sources

CONTENT SECURITY POLICY

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```



CONTENT SECURITY POLICY

`Content-Security-Policy: child-src https://youtube.com`

The **child-src** directive specifies which origins can be used to embed iframes and workers

CONTENT SECURITY POLICY

```
Content-Security-Policy: child-src https://youtube.com
```

Here YouTube videos can be embedded
but not videos from other sites

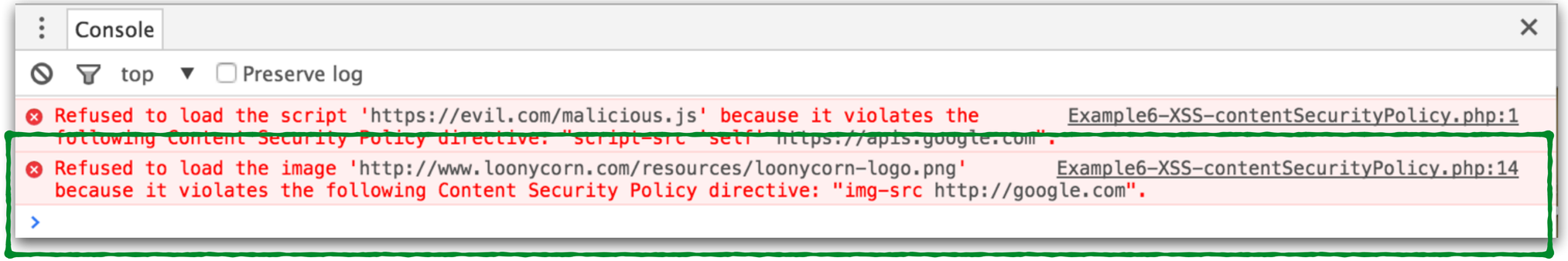
CONTENT SECURITY POLICY

```
Content-Security-Policy: img-src *
```

The **img-src** directive specifies where images can be loaded from. The * attribute indicates that all sources are valid for images

CONTENT SECURITY POLICY

```
Content-Security-Policy: img-src http://google.com
```



CONTENT SECURITY POLICY

```
"Content-Security-Policy: script-src 'self' https://
apis.google.com; child-src https://youtube.com; img-src *"
```

Here is how you specify **multiple** directives in the header

CONTENT SECURITY POLICY

```
"Content-Security-Policy: script-src 'self' https://
apis.google.com; child-src https://youtube.com; img-src *"
```

Here are the directives

CONTENT SECURITY POLICY

```
"Content-Security-Policy: script-src 'self' https://
apis.google.com; child-src https://youtube.com; img-src *"
```

Separated by commas

CONTENT SECURITY POLICY

font-src

Specifies origins that can serve fonts e.g. https://fonts.googleapis.com

form-action

Lists valid endpoints for submission of form contents

object-src

Allows control over flash and other plugins

style-src

Specifies where stylesheets can be loaded from

CONTENT SECURITY POLICY

font-src

Specifies origins that can serve fonts e.g. https://fonts.googleapis.com

form-action

Lists valid endpoints for submission of form contents

report-uri

object-src

Allows control over flash and other plugins

style-src

Specifies where stylesheets can be loaded from

plugin-types

media-src

connect-src

CONTENT SECURITY POLICY

When **no** Content-Security-Policy
is specified for a directive it's like
the **gate is wide open**

Code, content etc. can be loaded
and executed from **any** location

CONTENT SECURITY POLICY

When **no** Content-Security-Policy is specified for a directive it's like the **gate is wide open**

Code, content etc. can be loaded and executed from **any** location

This is equivalent to specifying

***** for every available directive in this header

CONTENT SECURITY POLICY

Having to give specific values for every directive makes things onerous

`default-src`

CONTENT SECURITY POLICY

`default-src`

The default-src directive allows you specify default expressions for all directives which end with -src

CONTENT SECURITY POLICY

```
default-src http://www.mysite.com
```

This means images, scripts, stylesheets, fonts and everything that can be specified by a "-src" attribute can only load from mysite.com

CONTENT SECURITY POLICY

```
default-src http://www.mysite.com
```

Just remember that directives like form-action, plugin-types etc do **not** fallback on this default

CONTENT SECURITY POLICY

```
script-src http://mysite.com:8080
```

This allows scripts to be loaded from that exact origin including the port

CONTENT SECURITY POLICY

```
script-src http://mysite.com:*
```

The script can be loaded from mysite.com and **any** port

CONTENT SECURITY POLICY

```
script-src http://*.mysite.com
```

The script can be loaded from **any subdomain** in mysite.com

CONTENT SECURITY POLICY

```
script-src https://*
```

The script can be loaded from any site provided it is over https

CONTENT SECURITY POLICY

Some special **keywords** which are allowed as source expressions

CONTENT SECURITY POLICY

```
media-src ‘none’
```

The source expression ‘none’ does not allow any resources of that type i.e. no video or audio download from any origin is allowed

CONTENT SECURITY POLICY

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

Only scripts from the host which served the page

CONTENT SECURITY POLICY

```
script-src 'unsafe-inline'
```

Allow resources such as <script> tags embedded in the page

CONTENT SECURITY POLICY

```
script-src 'unsafe-eval'
```

Allow the use of the Javascript **eval** function

CONTENT SECURITY POLICY

Note that all the keywords 'none', 'self', 'unsafe-eval' and 'unsafe-inline' are all in quotes

Otherwise they will be identified as hostnames rather than the special keywords

CONTENT SECURITY POLICY

One note: IE does not support the Content-Security-Policy header yet, IE 10 supports it partially

CONTENT SECURITY POLICY

The policy is defined on a page by page basis

Which means each page can re-define the policy for that page by specifying a different Content-Security-Policy header in that page's response

CONTENT SECURITY POLICY

If a certain page **loads images** from a partner site - only allow that particular page to do so

Do not give other pages the same privilege!

If a page needs to **embed YouTube videos** only set the policy for that page

CONTENT SECURITY POLICY

The **sandbox** directive works a little differently

It places restrictions on the **actions** that the page can **take** rather than on the **resources** that the page can **load**

CONTENT SECURITY POLICY

Using the sandbox directive you can

1. Force the page into a unique origin (different from the rest of the site)
2. Prevent forms submission
3. Prevent loading scripts

etc.

STAY AWAY FROM INLINE CODE

CONTENT SECURITY POLICY

Example7-XSS-disallowInline.php

Example7-XSS-disallowInline.js

STAY AWAY FROM INLINE CODE

Inline script injection is the most serious of all XSS threats

A browser has no way to distinguish between a legit inline script and a malicious one

STAY AWAY FROM INLINE CODE

Content-Security-Policy places a **blanket ban** on all inline scripts

no inline **<script>** tags

no inline **event handlers**

no **javascript:** scripts

STAY AWAY FROM INLINE CODE

```
<script>
  function handleButtonClick() {
    alert("You clicked the button!");
  }
</script>
<button onclick="handleButtonClick()"> Click me! </button>
```

This displays a simple button and which has a click handler

STAY AWAY FROM INLINE CODE

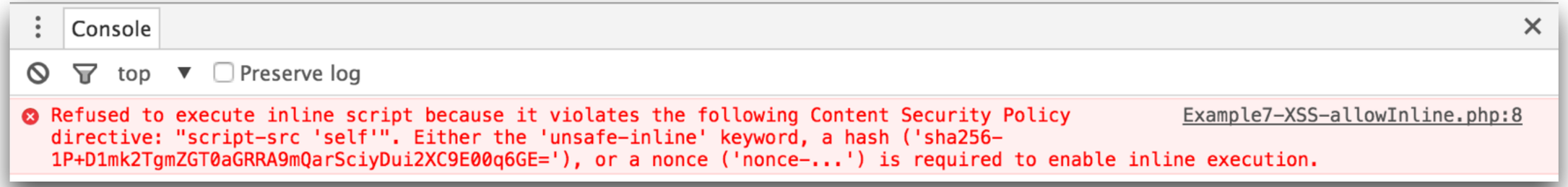
```
<script>
  function handleButtonClick() {
    alert("You clicked the button!");
  }
</script>
<button onclick="handleButtonClick()"> Click me! </button>
```

If you have Content-Security-Policy set to say 'self' then this is not allowed!

STAY AWAY FROM INLINE CODE

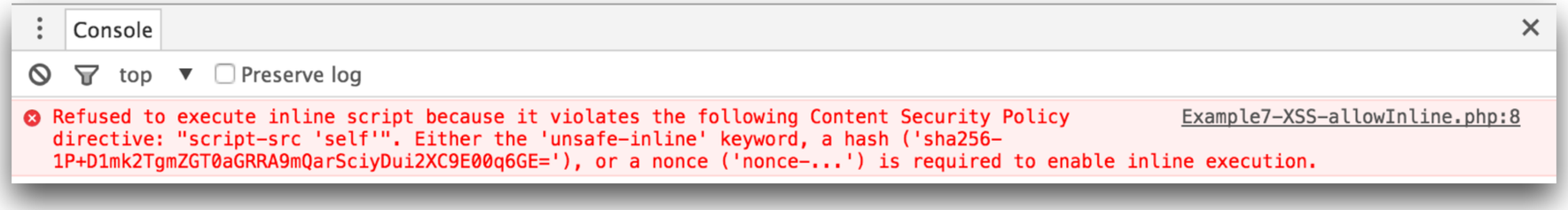
```
<script>
  function handleButtonClick() {
    alert("You clicked the button!");
  }
</script>
<button onclick="handleButtonClick()"> Click me! </button>
```

If you have Content-Security-Policy set to say 'self' then this is not allowed!



STAY AWAY FROM INLINE CODE

```
<script>
  function handleButtonClick() {
    alert("You clicked the button!");
  }
</script>
<button onclick="handleButtonClick()"> Click me! </button>
```



Use script-src 'unsafe-inline' to allow inline scripts

STAY AWAY FROM INLINE CODE

```
<script>
  function handleButtonClick() {
    alert("You clicked the button!");
  }
</script>
<button onclick="handleButtonClick()"> Click me! </button>
```

Or even better this code should be rewritten

STAY AWAY FROM INLINE CODE

Or even better this code should be rewritten

HTML file

```php
<?php
 header("Content-Security-Policy: script-src 'self'");
?>
<!DOCTYPE html>
<html lang="en">
<body>
 <button id="button-id"> Click me! </button>
 <script src='Example7-XSS-disallowInline.js'></script>
</body>
</html>
```

JS file

```js
window.onload = addListener();

function handleButtonClick() {
    alert('You clicked the button!');
}

function addListener() {
    var el = document.getElementById("button-id");
    el.addEventListener("click", handleButtonClick);
}
```

All event handling code should be in the JS file

STAY AWAY FROM INLINE CODE

HTML file

```php
<?php
 header("Content-Security-Policy: script-src 'self'");
?>
<!DOCTYPE html>
<html lang="en">
<body>
 <button id="button-id"> Click me! </button>
 <script src='Example7-XSS-disallowInline.js'></script>
</body>
</html>
```

JS file

```javascript
window.onload = addListener();

function handleButtonClick() {
    alert('You clicked the button!');
}

function addListener() {
    var el = document.getElementById("button-id");
    el.addEventListener("click", handleButtonClick);
}
```

Reference the JS file from the HTML file

STAY AWAY FROM INLINE CODE

HTML file

```php
<?php
 header("Content-Security-Policy: script-src 'self'");
?>
<!DOCTYPE html>
<html lang="en">
<body>
 <button id="button-id"> Click me! </button>
 <script src='Example7-XSS-disallowInline.js'></script>
</body>
</html>
```

JS file

```js
window.onload = addListener();

function handleButtonClick() {
    alert('You clicked the button!');
}

function addListener() {
    var el = document.getElementById("button-id");
    el.addEventListener("click", handleButtonClick);
}
```

The listener for the button click

STAY AWAY FROM INLINE CODE

HTML file

```php
<?php
 header("Content-Security-Policy: script-src 'self'");
?>
<!DOCTYPE html>
<html lang="en">
<body>
 <button id="button-id"> Click me! </button>
 <script src='Example7-XSS-disallowInline.js'></script>
</body>
</html>
```

JS file

```javascript
window.onload = addListener();

function handleButtonClick() {
    alert('You clicked the button!');
}

function addListener() {
    var el = document.getElementById("button-id");
    el.addEventListener("click", handleButtonClick);
}
```

Add the event listener in JS

CONTENT SECURITY POLICY

Refactoring the JS code into a separate file is a good practice which has other advantages!

1. It splits up presentation from business logic
2. External resources are easier to cache
3. JS can be compiled and minified

CONTENT SECURITY POLICY

Avoiding inline scripts goes a long way towards making your website XSS secure - stick to it if possible!

THE NONCE ATTRIBUTE

THE NONCE ATTRIBUTE

Browsers which support CSP level 2 (latest Chrome and Firefox) **allow** specific inline scripts to be whitelisted under **certain** conditions

Using something called a **nonce** attribute on the script tag

THE NONCE ATTRIBUTE

```
<script nonce="Xiojd98">
  function doSomething() {
    alert("This alert is inline with the nonce");
  }
</script>
```

The nonce attribute uses a secure random id to identify those script tags which we expect on a page

THE NONCE ATTRIBUTE

```
<script nonce="Xiojd98">
  function doSomething() {
    alert("This alert is inline with the nonce");
  }
</script>
```

The nonce should change for every response!

THE NONCE ATTRIBUTE

```
<script nonce=“Xiojd98”>
  function doSomething() {
    alert("This alert is inline with the nonce");
  }
</script>
```

The page indicates that this script is acceptable by specifying the nonce values in the Content-Security-Policy header

THE NONCE ATTRIBUTE

```
<script nonce=“Xiojd98”>
  function doSomething() {
    alert("This alert is inline with the nonce");
  }
</script>
```

```
Content-Security-Policy: script-src 'nonce-Xiojd98'
```

THE NONCE ATTRIBUTE

Example8-XSS-nonce.php

THE NONCE ATTRIBUTE

```php
<?php
  $nonce = sha1(uniqid('n', true));
  header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>XSS - Nonce</title>
</head>
<body>
  <script nonce=<?php echo $nonce ?>>
    window.onload = doSomething();
    function doSomething() {
      alert("This alert is inline with the nonce");
    }
  </script>
</body>
</html>
```

Generate a nonce for every response

THE NONCE ATTRIBUTE

```php
<?php
  $nonce = sha1(uniqid('n', true));
  header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>XSS - Nonce</title>
</head>
<body>
  <script nonce=<?php echo $nonce; ?>>
    window.onload = doSomething();
    function doSomething() {
      alert("This alert is in the nonce");
    }
  </script>
</body>
</html>
```

A nonce is generated each time this page is requested

THE NONCE ATTRIBUTE

```php
<?php
  $nonce = sha1(uniqid('n', true));
  header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>XSS - Nonce</title>
</head>
<body>
  <script nonce="<?= $nonce ?>">
    window.onload = doSomething();
    function doSomething() {
      alert("This alert was triggered with the nonce");
    }
  </script>
</body>
</html>
```

Include the nonce in the Content-Security-Policy header

THE NONCE ATTRIBUTE

Make sure the nonce is included in the script tag

```
<?php
  $nonce = base64_encode(random_bytes(16));
  header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>XSS - Nonce</title>
</head>
<body>
  <script nonce=<?php echo $nonce ?>>
    window.onload = doSomething();
    function doSomething() {
      alert("This alert is inline with the nonce");
    }
  </script>
</body>
</html>
```

THE NONCE ATTRIBUTE

```php
<?php
  $nonce = sha1(uniqid('n', true));
  header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>XSS - Nonce</title>
</head>
<body>
  <script nonce=<?php echo $nonce ?>>
    window.onload = doSomething();
    function doSomething() {
      alert("This alert is inline with the nonce");
    }
  </script>
</body>
</html>
```

Now the inline script should work!

THE NONCE ATTRIBUTE

```php
<?php
  $nonce = sha1(uniqid('n', true));
  header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>XSS - Nonce</title>
</head>
<body>
  <script nonce=<?php echo $nonce ?>>
    window.onload = doSomething();
    function doSomething() {
      alert("This alert is inline with the nonce");
    }
  </script>
</body>
</html>
```

Remove the nonce from the inline script and you'll find that it does not execute

THE NONCE ATTRIBUTE

```php
<?php
  $nonce = sha1(uniqid('n', true));
  header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>XSS - Nonce</title>
</head>
<body>
  <script nonce=<?php echo $nonce ?>>
    window.onload = doSomething();
    function doSomething() {
      alert("This alert is inline with the nonce");
    }
  </script>
</body>
</html>
```

Generate a nonce for every response, the nonce should not be simple or guessable

THE SCRIPT HASH

THE SCRIPT HASH

An alternative to the nonce is to use a **hash** to **uniquely identify** the contents of a script tag

Specify this hash as part of the **script-src** in the Content-Security-Policy

THE SCRIPT HASH

An alternative to the nonce is to use a **hash** to **uniquely identify** the contents of a script tag

Specify this hash as part of the **script-src** in the Content-Security-Policy

Inline scripts which **match** the hash are allowed to execute on that page

THE SCRIPT HASH

Example9-XSS-hash.php

THE SCRIPT HASH

```
<script>
  window.onload = doSomething();
  function doSomething() {
    alert("This alert is inline with the hash");
  }
</script>
```

If this is the inline script you want to execute on the page

THE SCRIPT HASH

```
<script>
  window.onload = doSomething();
  function doSomething() {
    alert("This alert is inline with the hash");
  }
</script>
```

If this is the inline script you want to execute on the page

```
Content-Security-Policy: script-src 'sha256-iUH3eQYYa+2VDoL59Xvzys4n7OTwJrE2iKsyC+r4pao='
```

THE SCRIPT HASH

```
<script>
  window.onload = doSomething();
  function doSomething() {
    alert("This alert is inline with the hash");
  }
</script>
```

```
Content-Security-Policy: script-src 'sha256-iUH3eQYYa+2VDoL59Xvzys4n7OTwJrE2iKsyC+r4pao='
```

The sha256 indicates the encryption algorithm used for the hash - CSP supports sha384 and sha512 as well

THE SCRIPT HASH

```
<script>
  window.onload = doSomething();
  function doSomething() {
    alert("This alert is inline with the hash");
  }
</script>
```

```
Content-Security-Policy: script-src 'sha256-iUH3eQYYa+2VDoL59Xvzys4n7OTwJrE2iKsyC+r4pao='
```

You can generate this hash using online tools or can do it programmatically

THE SCRIPT HASH

```
<script>
  window.onload = doSomething();
  function doSomething() {
    alert("This alert is inline with the hash");
  }
</script>
```

```
Content-Security-Policy: script-src 'sha256-iUH3eQYYa+2VDoL59Xvzys4n7OTwJrE2iKsyC+r4pao='
```

Using a version of Chrome newer than 40 gives you the hash when you open up Chrome developer tools

THE SCRIPT HASH

CSP Using hash for inline scripts

STAY AWAY FROM EVAL AS WELL

STAY AWAY FROM EVAL AS WELL

eval() is a function in Javascript which **evaluates a string** as though it was an expression i.e. code and returns a result

STAY AWAY FROM EVAL AS WELL

Basically an string can have code in it - it's a major source of XSS issues!

It's not just eval(), functions like **setTimeout()**, **setInterval()**, **new Function()** all accept strings to be evaluated and executed

STAY AWAY FROM EVAL AS WELL

Content-Security-Policy simply
blocks this completely

STAY AWAY FROM EVAL AS WELL

```
setTimeout(
    “document.getElementById('some-id').style.display = 'none';”,
    10);
```

This is actually a string which has code inside it

STAY AWAY FROM EVAL AS WELL

```
setTimeout(
    “document.getElementById('some-id').style.display = 'none';",
    10);
```

DO NOT USE STRINGS WHICH CAN BE EVALUATED AS CODE!

STAY AWAY FROM EVAL AS WELL

```
setTimeout(function () {
  document.getElementById(‘some-id').style.display = 'none';
}, 10);
```

Use actual code instead:-)

STAY AWAY FROM EVAL AS WELL

```
setTimeout(function () {
  document.getElementById('some-id').style.display = 'none';
}, 10);
```

Use actual code instead:-)

In summary use Content-Security-Policy for your website

In summary use Content-Security-Policy for your website

Typically companies roll it out in report only mode - and then turn it on completely

report only mode

Here violations are reported but the restrictions are not enforced

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

report only mode

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

A URL on your server which can track the violations so they can be fixed

In summary use Content-Security-Policy for your website

Typically companies roll it out in report only mode - and then turn it on completely

In summary use Content-Security-Policy for your website

Typically companies roll it out in report only mode - and then turn it on completely

For an existing site which has inline scripts and lots of external dependencies this will be a major rewrite

In summary use Content-Security-Policy for your website

Typically companies roll it out in report only mode - and then turn it on completely

For an existing site which has inline scripts and lots of external dependencies this will be a major rewrite

In summary use Content-Security-Policy for your website

Typically companies roll it out in report only mode - and then turn it on when ready

For an existing site which has inline scripts and lots of external dependencies this will be a major rewrite

But totally worth it in the long run!