WTF, XSS? CSP!

or simply: use a safe by default stack and “strict-dynamic”

Menu du jour

We will talk about:

- WTF, XSS?
- A little rant about our current stack
- Silver bullets (not)
- CSP!
- ???

WTF, XSS?

XSS = Cross-site scripting

- One of the most common vulnerability with (sql, ...) injections
- You are able to inject arbitrary javascript in pages viewed by other users
- Should not be considered a feature

Why should we care?

- For your users
- For your business
- For you

As the complexity of our applications/services goes up, the number of mistakes goes up, as we are (still) not robots

Typical reflected XSS :)

```
<?php

echo $_GET['param'];

?>
```

You can add some variations

- Persisted (for example, the payload is saved in a database)
- DOM based (when everything is done client side, without intervention of the server)

Escaping is hard

That’s why we use a template engine

Right?

Well, unfortunately most of them have do not have a safe default

Auditing every templates is time consuming

Let’s be realist: most of us don’t have a dedicated security team

Mistakes

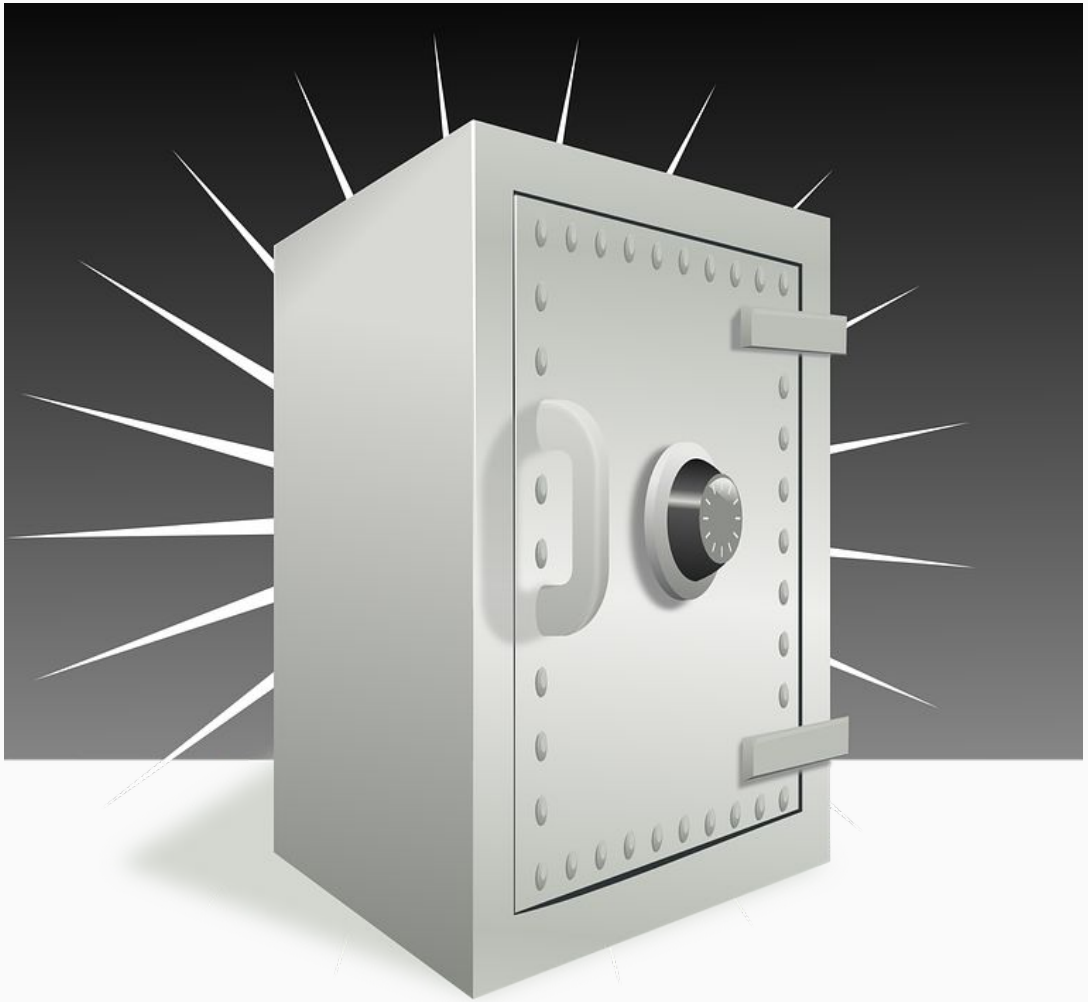
Everybody does it

<rant start=”true”>

Our tools often are:

We need to build:

With a level of security equivalent to:



Escaping is hard, escaping is contextual

From twig documentation: http://twig.sensiolabs.org/doc/filters/escape.html
you need to use the correct strategy manually, error prone, and somebody

will make an error (most likely me!)

Escaping should be done automa{gi,ti}cally

- Closure template: https://developers.google.com/closure/templates/
- Golang “html/template”: https://golang.org/pkg/html/template/
- XHP: https://docs.hhvm.com/hack/XHP/introduction

(thank you google&facebook)

On the client side:

It’s easy to make mistakes too:

- A misplaced and forgotten .innerHTML / .html() call somewhere deep in your code (or worse, in a library!)
- Maybe a document.write() is used for good measure? :D
- Using eval() with user supplied data? Or passing a string to setTimeout/setInterval?

Should we pretend/hope a safe by default approach?

the answer is obviously:

YES

</rant>

Disclaimer

No silver bullets here

No miracle pill here

CSP is a defense-in-depth mechanism

Paranoia++

- CSP help mitigate a good amount of XSS
- but it does not mean you can lower your shield: all your stack should be “safe”!
- SQL injection will happen if you are not using bind parameters
- Session hijacking will happen if you are not careful enough
- Account handling is hard too

CSP: Content Security Policy

- A HTTP header for controlling what can be loaded and executed
- Can block inline javascript (major source of XSS) and use of eval
- Extremely fine grained (too much actually):
  - Styles
  - Images
  - Fonts
  - Connect-src (where xmlhttprequest&co are allowed to connect)
  - Child-src (frames, …)

A policy should look likes that:

```
Content-Security-Policy:
  object-src 'none';
  script-src 'nonce-{random}' 'unsafe-inline' 'unsafe-eval' 'strict-dynamic' https: http:;
  report-uri https://your-report-collector.example.com/
```

The policy is “simple” and clever

- No whitelist of domains to maintain
- Care only about javascript
- Work to it’s full extent with modern browsers, degrade to a no-op for the old one

Your scripts should look like that

```
<script nonce="${nonce}" src="script.js"></script>
```

A Nonce

- Must be a hard to guess number
- Which change at every page reload
- 16 bytes from /dev/urandom is enough

Additional work must be done

- Refactor inline event handlers: no more onclick=”myFunction();”
- Cleanup document.write(‘<script>’+...) to convert them in createElement call with the nonce setted as an attribute
- Remove eval&co :)

Deploy your policy as report-only first

So you can see what will be broken:

```
Content-Security-Policy-Report-Only: <your policy here>
```

Use tool like CSP-Mitigator (chrome extension) during development to see what you need to fix

Even with this policy, what could go wrong?

- You are using a template engine which does not escape contextually and automatically
- Beware of dynamically generated javascript
- Do you use jsonp endpoints?
- Did I just say to avoid dynamically generated javascript?
- Data exfiltration without javascript is possible
- ...

Set your priority

- Evaluate where you should put the effort
- Secure your stack first!
- Adopt CSP
- ???
- Profit!

Questions and blames?

Tools and resources and sources:

- https://csp.withgoogle.com/docs/index.html
- https://csp-evaluator.withgoogle.com/
- https://developers.google.com/web/fundamentals/security/csp/
- https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting
- https://www.w3.org/TR/CSP3/#strict-dynamic-usage