Eliminating XSS: Context-Sensitive Auto-Sanitization in PHP

Joseph Connor
@josconno

Jared M. Smith
@jaredthecoder

Howdy!

I am Jared Smith

I do R&D at Oak Ridge National Laboratory.

You can find me at @jaredthecoder on Twitter.

Hi there!

I am Joseph Connor

I hack into stuff at Cisco Systems.

You can find me at @josconno on Twitter.

47%

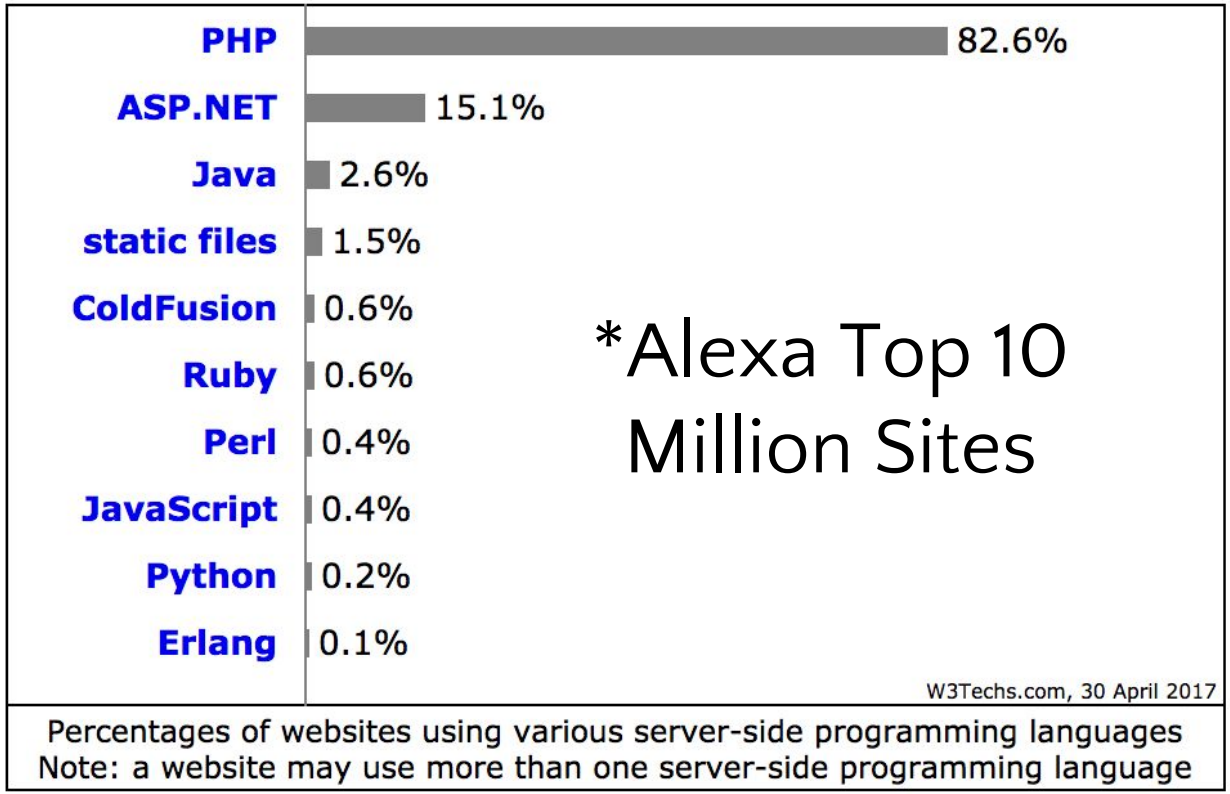
Likelihood of XSS in an Application in 2015

Source: White Hat Information Security

#3

2017 OWASP Top 10 Application Vulnerabilities

Source: OWASP



*Alexa Top 10 Million Sites

Existing Solutions

- PHP Taint, a module to mark data as unsafe, doesn't take action to sanitize data
- Manual sanitization functions rely on the programmer to be smart about security

Our Solution: PHP-CSAS

- We provide a PHP extension that **automatically and transparently to the user and developer** effectively sanitizes user-input **based on context**
- We add **less than 2% overhead** at runtime
- Our solution is compatible with PHP 5.X, works as a PECL extension, and is works with Wordpress, MediaWiki, and other major PHP applications

Why does this matter?

1

AppSec is hard...

Let’s face it. Securing your stuff isn’t easy.

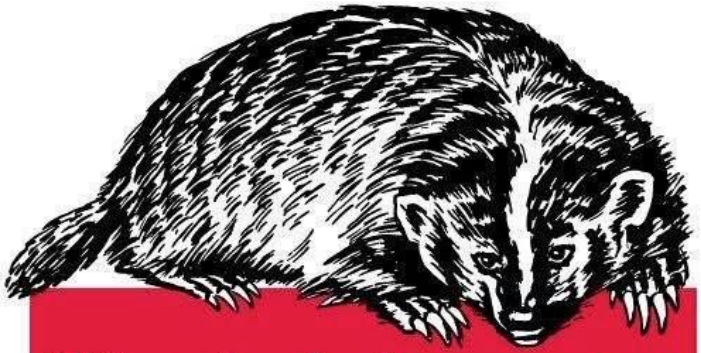
“The only truly secure system is one that is powered off, cast in a block of concrete, and sealed in a lead-lined room with armed guards.” - Gene Spafford

2

Developers suck.

You can’t trust developers to write secure code.

The definitive guide for all project managers



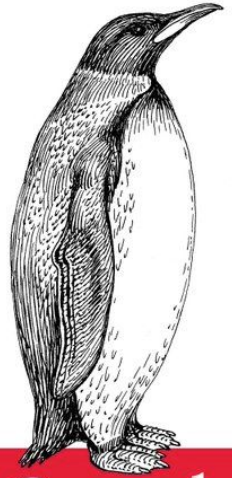
What the fuck is security

How to ignore it and deliver your project

O'RLY

Awn Thyme

Letting your baby out of the nest — for better or worse



Good Enough to Ship

The Definitive Guide

O RLY?

@ThePracticalDev

3

Users suck more.

You *definitely* can't trust your users.


shutterstock

Approach

Prior Work

- Python's Django Web Framework
  - Auto-escaping by default in Django templates since 2007
- Google's Closure Templates for Java and JS
  - Samuel et al, *Context-Sensitive Auto-Sanitization in Web Templating Languages Using Type Qualifiers*
- CTemplate
  - Optional auto-escaping for many contexts

Approach

- Context-Sensitive Auto-Sanitization (CSAS) with a PHP extension:
  - Track trustworthiness of data
  - Sanitize untrusted data automatically in a **context-sensitive** manner

Why context-sensitive?

- **`htmlspecialchars()`** isn't always enough...

Why context-sensitive?

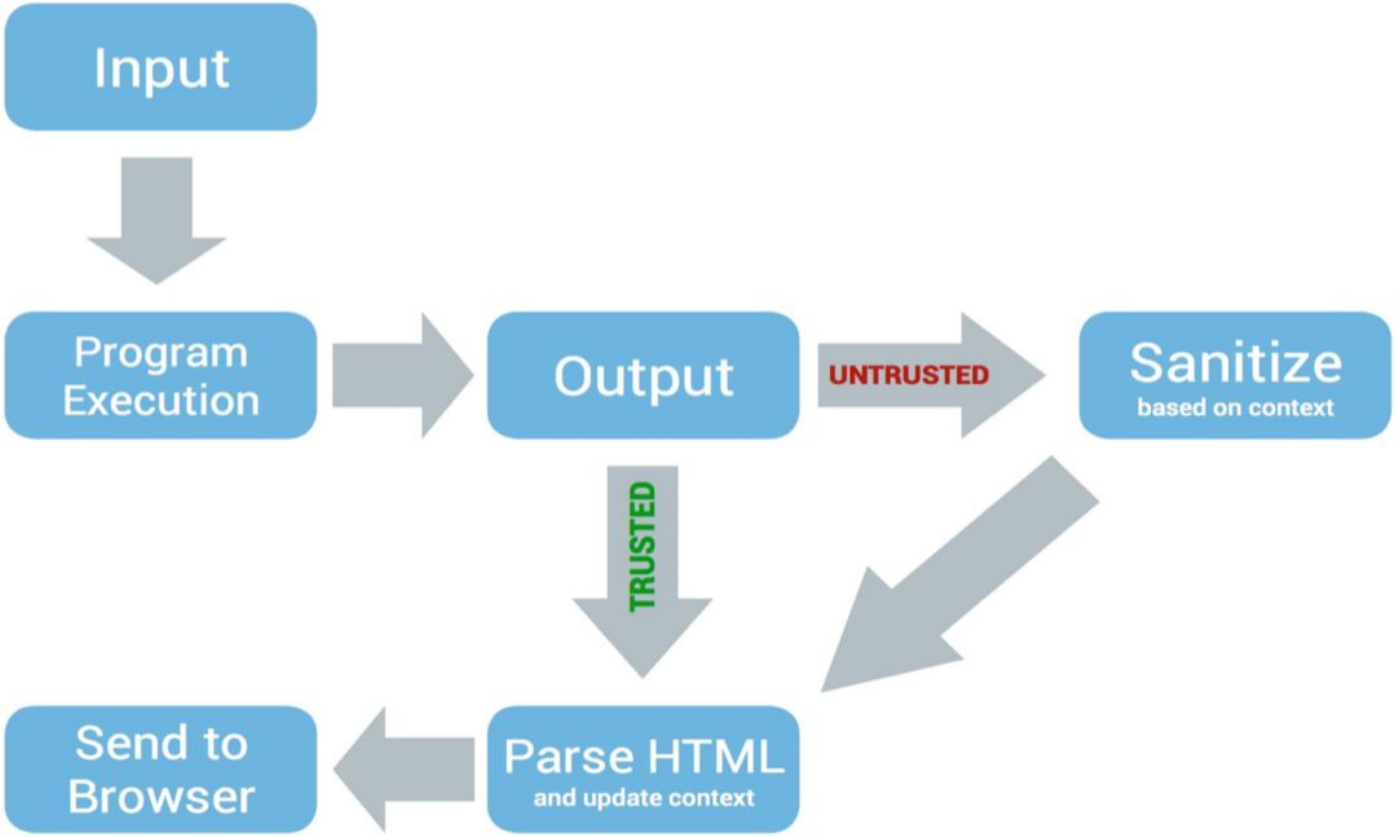
```
<a href="<?php echo htmlspecialchars($url); ?>" >
```

```
$url = "javascript:alert()"
```

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

How context-sensitive?

- Solution:
  - Use an HTML parser to parse our own output
  - Select sanitizer based on parser state:
    - HTML text, tag, attribute
    - Javascript string
    - CSS


Input
Program Execution
Output
UNTRUSTED
Sanitize
based on context
TRUSTED
Send to Browser
Parse HTML
and update context

Input

- Tag input trustworthiness by source
- What's trusted?
  - String literals
- Untrusted?
  - Request parameters, headers (duh)
  - SQL query results
  - Data read from files
  - Pretty much everything else

Tracking Trust

- Track data safety across string operations
- For most, this is a GLB of operand safety:
  - safety(result) = safety(op1) & safety(op2)
- But not all:
  - htmlspecialchars()

Parsing Output

- Feed all output to internal HTML parser (post-sanitization)
- Keep track of HTML context:
  - Ordinary HTML
  - Inside a tag
  - Inside a tag attribute
  - JS
  - CSS

Sanitization

- For all output data:
  - Check HTML parser state
  - Is data marked safe for current context?
    - If so, output it as-is
    - If not, sanitize it with appropriate sanitizer for context, then output
  - Send result to HTML parser

Implementation

Implementation

- Override lots of PHP internals:
  - Request initialization (tag request data as unsafe)
  - Custom handlers for PHP VM opcodes
  - Patch PHP internal function and class tables

```
if (zend_hash_find(CG(function_table), name, len, (void **)&func) == SUCCESS) {

    if (stash) {
        *stash = func->internal_function.handler;
    }

    func->internal_function.handler = handler;
}
```

Implementation

- Override:
  - Input functions (SQL, file reading, ...)
  - String manipulation (concatenation, substrings, ...)
  - Output (echo, printf, ...)

Implementation

- Tag strings — without changing zval struct
  - Just store tag after the string!

```
typedef union _zvalue_value {
    ...
    struct {
        char *val;
        int len;
    } str;
    ...
} zvalue_value;
```

```
#define PHP_CSAS_SET_SAFETY_(zv, mark) \
     *((unsigned *)(Z_STRVAL_P(zv)\
     + Z_STRLEN_P(zv) \
     + 1 + sizeof(uint))) = (mark)
```

Example: csas_mysqli_result_fetch_array

```
PHP_FUNCTION(csas_mysqli_result_fetch_array) {
   CSAS_O_FUNC(mysqli_result_fetch_array)(INTERNAL_FUNCTION_PARAM_PASSTHRU);

   if (return_value && Z_TYPE_P(return_value) == IS_ARRAY) {
       php_csas_mark_strings(return_value, PHP_CSAS_UNSAFE, 1 TSRMLS_CC);
   }
}
```

Example: php_csas_concat_handler

```
safety = PHP_CSAS_SAFE_ALL;
if (op1 && IS_STRING == Z_TYPE_P(op1)) {
    safety &= php_csas_get_safety(op1);
}
if (op2 && IS_STRING == Z_TYPE_P(op2)) {
    safety &= php_csas_get_safety(op2);
}

concat_function(result, op1, op2 TSRMLS_CC);
if (IS_STRING == Z_TYPE_P(result)) {
    Z_STRVAL_P(result) = erealloc(Z_STRVAL_P(result),
            Z_STRLEN_P(result) + 1 +PHP_CSAS_MAGIC_LENGTH);
    php_csas_set_safety(result, safety);
}
```

Example: php_csas_safe_write

```
static int php_csas_safe_write(char *s, int len, uint safety TSRMLS_DC) {
   int rval = 0;
   char *s_safe = sanitize_for_context(s, safety, &len);

   if (len > 0)  {
       rval = PHPWRITE(s_safe, len);
       htmlparser_update_context(htmlparser, s_safe, len);
   }

   return rval;
}
```

Results

Results: Compatibility

Platform	Tested Version	Compatible
Wordpress	4.5	**Yes***
MediaWiki	1.26.2	**Yes**
RoundCube Webmail	1.15	**Yes**

*except for some HTMLPC data. For example, <script> was automatically sanitized, but <b> was not for a comment form.

Results: Limitations

- Incurs 1.9x overhead in page load time
- Potential for over-sanitization

Future Work

- Compatibility with web development frameworks in PHP:
  - Symfony
  - Laravel
  - CakePHP
- Test with PHP 7.x
- Add sanitization functions for additional contexts
- Extension added to the official PECL repositories

Conclusions

- XSS is still a major problem
- PHP is widely used
- Built a system to mitigate XSS with no developer or user effort required
- Compatible with major PHP frameworks
- Incurs some overhead in page load time
- Project Code: http://php-csas.github.io/php-csas/

Thanks!

Any questions ?

Find Joseph:

- @josconno
- joecon.net
- josephc346@gmail.com

Find Jared:

- @jaredthecoder
- jaredthecoder.com
- jared@jaredthecoder.com