OWASP / java-html-sanitizer Public

Takes third-party HTML and produces HTML that is safe to embed in your web application. Fast and easy to configure.

View license

772 stars 201 forks Activity

Star Watch

Code Issues 94 Pull requests 12 Actions Projects Security Insigh

main

mikesamuel Create codeql.yml … on Apr 7 518

View code

OWASP Java HTML Sanitizer

coverage 93% openssf best practices passing maven central 20220608.1

A fast and easy to configure HTML Sanitizer written in Java which lets you include HTML authored by third-parties in your web application while protecting against XSS.

The existing dependencies are on guava and JSR 305. The other jars are only needed by the test suite. The JSR 305 dependency is a compile-only dependency, only needed for annotations.

This code was written with security best practices in mind, has an extensive test suite, and has undergone adversarial security review.

Table Of Contents

README.md

Getting Started

Getting Started includes instructions on how to get started with or without Maven.

Prepackaged Policies

You can use prepackaged policies:

```
PolicyFactory policy = Sanitizers.FORMATTING.and(Sanitizers.LINKS);
String safeHTML = policy.sanitize(untrustedHTML);
```

Crafting a policy

The tests show how to configure your own policy:

```
PolicyFactory policy = new HtmlPolicyBuilder()
    .allowElements("a")
    .allowUrlProtocols("https")
    .allowAttributes("href").onElements("a")
    .requireRelNofollowOnLinks()
    .toFactory();
String safeHTML = policy.sanitize(untrustedHTML);
```

Custom Policies

You can write custom policies to do things like changing `h1` s to `div` s with a certain class:

```
PolicyFactory policy = new HtmlPolicyBuilder()
    .allowElements("p")
    .allowElements(
        (String elementName, List<String> attrs) -> {
          // Add a class attribute.
          attrs.add("class");
          attrs.add("header-" + elementName);
          // Return elementName to include, null to drop.
          return "div";
        }, "h1", "h2", "h3", "h4", "h5", "h6")
    .toFactory();
String safeHTML = policy.sanitize(untrustedHTML);
```

Please note that the elements "a", "font", "img", "input" and "span" need to be explicitly whitelisted using the `allowWithoutAttributes()` method if you want them to be allowed through the filter when these elements do not include any attributes.

Attribute policies allow running custom code too. Adding an attribute policy will not water down any default policy like `style` or URL attribute checks.

```
new HtmlPolicyBuilder = new HtmlPolicyBuilder()
    .allowElement("div", "span")
    .allowAttributes("data-foo")
        .matching(
            (String elementName, String attributeName, String value) -> {
              // Return value for the attribute or null to drop.
            })
        .onElements("div", "span")
    .build()
```

Preprocessors

Preprocessors allow inserting text and large scale structural changes.

```
new HtmlPolicyBuilder = new HtmlPolicyBuilder()
    // Use a preprocessor to be backwards compatible with the
    // <plaintext> element which
    .withPreprocessor(
        (HtmlStreamEventReceiver r) -> {
          // Provide user with info about links before they click.
          // Before:                       <a href="https://example.com/...">
          // After:  (https://example.com) <a href="https://example.com/...">
          return new HtmlStreamEventReceiverWrapper(r) {
            @Override public void openTag(String elementName, List<String> attrs) {
              if ("a".equals(elementName)) {
                for (int i = 0, n = attrs.size(); i < n; i += 2) {
                  if ("href".equals(attrs.get(i)) {
```

```
                String url = attrs.get(i + 1);
                String origin;
                try {
                  URI uri = new URI(url);
                  String scheme = uri.getScheme();
                  String authority = uri.getRawAuthority();
                  if (scheme == null && authority == null) {
                    origin = null;
                  } else {
                    origin = (scheme != null ? scheme + ":" : "")
                           + (authority != null ? "//" + authority : "");
                  }
                } catch (URISyntaxException ex) {
                  origin = "about:invalid";
                }
                if (origin != null) {
                  text(" (" + origin + ") ");
                }
              }
            }
          }
          super.openTag(elementName, attrs);
        }
      };
    }
.allowElement("a")
...
.build()
```

Preprocessing happens before a policy is applied, so cannot affect the security of the output.

Telemetry

When a policy rejects an element or attribute it notifies an HtmlChangeListener.

You can use this to keep track of policy violation trends and find out when someone is making an effort to breach your security.

```
PolicyFactory myPolicyFactory = ...;
// If you need to associate reports with some context, you can do so.
MyContextClass myContext = ...;

String sanitizedHtml = myPolicyFactory.sanitize(
    unsanitizedHtml,
    new HtmlChangeListener<MyContextClass>() {
      @Override
      public void discardedTag(MyContextClass context, String elementName) {
        // ...
      }
      @Override
      public void discardedAttributes(
          MyContextClass context, String elementName, String... attributeNames) {
        // ...
      }
    },
    myContext);
```

Note: If a string sanitizes with no change notifications, it is not the case that the input string is necessarily safe to use. Only use the output of the sanitizer.

The sanitizer ensures that the output is in a sub-set of HTML that commonly used HTML parsers will agree on the meaning of, but the absence of notifications does not mean that the input is in such a sub-set, only that it does not contain elements or attributes that were removed.

See "Why sanitize when you can validate" for more on this topic.

Questions?

If you wish to report a vulnerability, please see AttackReviewGroundRules.

Subscribe to the mailing list to be notified of known Vulnerabilities and important updates.

Contributing

If you would like to contribute, please ping @mvsamuel or @manicode.

We welcome issue reports and PRs. PRs that change behavior or that add functionality should include both positive and negative tests.

Please be aware that contributions fall under the Apache 2.0 License.

Credits

Thanks to everyone who has helped with criticism and code

Releases 6

Release 20220608.1 Latest
on Jun 8, 2022

+ 5 releases

Packages

No packages published

Used by 2.5k

+ 2,501

Contributors 18

+ 7 contributors

Languages

● **Java** 50.9% ● **JavaScript** 35.1% ● **HTML** 13.6% ● **Shell** 0.4%