json-sanitizer

Given JSON-like content, The JSON Sanitizer converts it to valid JSON.

Getting Started - Contact

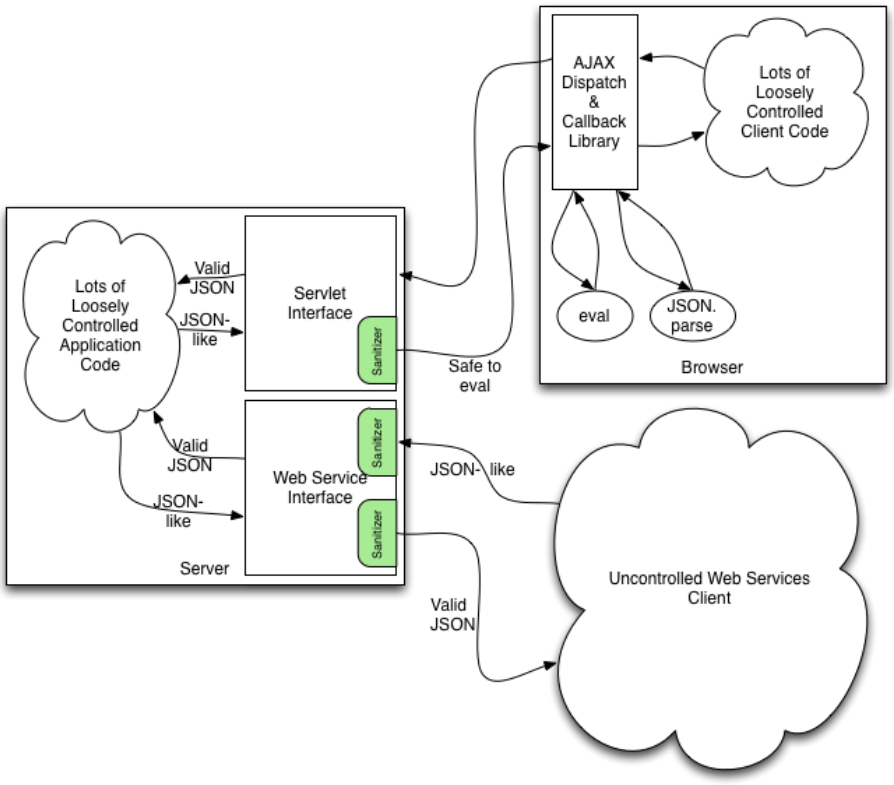
This can be attached at either end of a data-pipeline to help satisfy Postel's principle:

> be conservative in what you do, be liberal in what you accept from others

Applied to JSON-like content from others, it will produce well-formed JSON that should satisfy any parser you use.

Applied to your output before you send, it will coerce minor mistakes in encoding and make it easier to embed your JSON in HTML and XML.

Motivation



Many applications have large amounts of code that uses ad-hoc methods to generate JSON outputs.

Frequently these outputs all pass through a small amount of framework code before being sent over the network. This small amount of framework code can use this library to make sure that the ad-hoc outputs are standards compliant and safe to pass to

(overly) powerful deserializers like Javascript's `eval` operator.

Applications also often have web service APIs that receive JSON from a variety of sources. When this JSON is created using ad-hoc methods, this library can massage it into a form that is easy to parse.

By hooking this library into the code that sends and receives requests and responses, this library can help software architects ensure system-wide security and well-formedness guarantees.

Input

The sanitizer takes JSON like content, and interprets it as JS eval would. Specifically, it deals with these non-standard constructs.

Construct	Policy
`'...'`	Single quoted strings are converted to JSON strings.
`\xAB`	Hex escapes are converted to JSON unicode escapes.
`\012`	Octal escapes are converted to JSON unicode escapes.
`0xAB`	Hex integer literals are converted to JSON decimal numbers.
`012`	Octal integer literals are converted to JSON decimal numbers.
`+.5`	Decimal numbers are coerced to JSON's stricter format.
`[0,,2]`	Elisions in arrays are filled with `null`.
`[1,2,3,]`	Trailing commas are removed.
`{foo:"bar"}`	Unquoted property names are quoted.
`//comments`	JS style line and block comments are removed.
`(...)`	Grouping parentheses are removed.

The sanitizer fixes missing punctuation, end quotes, and mismatched or missing close brackets. If an input contains only white-space then the valid JSON string `null` is substituted.

Output

The output is well-formed JSON as defined by RFC 4627. The output satisfies these additional properties:

- The output will not contain the substring (case-insensitively) `"</script"` so can be embedded inside an HTML script element without further encoding.
- The output will not contain the substring `"]]>"` so can be embedded inside an XML CDATA section without further encoding.
- The output is a valid Javascript expression, so can be parsed by Javascript's `eval` builtin (after being wrapped in parentheses) or by `JSON.parse`. Specifically, the output will not contain any string literals with embedded JS newlines (U+2028 Paragraph separator or U+2029 Line separator).
- The output contains only valid Unicode scalar values (no isolated UTF-16 surrogates) that are allowed in XML unescaped.

Security

Since the output is well-formed JSON, passing it to `eval` will have no side-effects and no free variables, so is neither a code-injection vector, nor a vector for exfiltration of secrets.

This library only ensures that the JSON string → Javascript object phase has no side effects and resolves no free variables, and cannot control how other client side code later interprets the resulting Javascript object. So if client-side code takes a part of the parsed data that is controlled by an attacker and passes it back through a powerful interpreter like `eval` or `innerHTML` then that client-side code might suffer unintended side-effects.

```
var myValue = eval(sanitizedJsonString);  // safe
var myEmbeddedValue = eval(myValue.foo);  // possibly unsafe
```

Additionally, sanitizing JSON cannot protect an application from Confused Deputy attacks

```
var myValue = JSON.parse(sanitizedJsonString);
addToAdminstratorsGroup(myValue.propertyFromUntrustedSource);
```

Performance

The sanitize method will return the input string without allocating a new buffer when the input is already valid JSON that satisfies the properties above. Thus, if used on input that is usually well formed, it has minimal memory overhead.

The sanitize method takes O(n) time where n is the length of the input in UTF-16 code-units.