The State of the Art is (Still) Regexp-based Validation

Roger L. Costello
May 2013

In the 2005 paper "*Guns and Butter: Towards Formal Axioms of Input Validation*" [1] the authors bemoan the fact that at that time most input validation was done using regular expressions:

> The state of the art, as far as implementation
> goes, is regexp-based validation.

Sadly, things haven't changed in the 8 years since that paper was written. In 2013 XML-formatted data is still validated using regular expressions. Let's see why.

In a previous post [2] I showed how to construct XML languages with increasing computational power – Regular Languages, Deterministic Context-Free Languages, Non-deterministic Context-Free Languages, Recursive Languages, and Recursively Enumerable Languages. It is possible to validate these languages using XML Schema 1.1. So how can I claim that the state of the art is still regexp-based validation? Answer: XML Schema 1.1 provides good support for validating *markup*, but the primary mechanism for validating *data* is still regular expressions. For example, suppose we want to constrain the value of this element:

```
<Surname>_______________</Surname>
```

The most powerful mechanism that XML Schema provides for constraining that value is a regular expression in a pattern facet, for example:

```
<element name="Surname">
    <simpleType>
        <restriction base="string">
            <pattern value="[a-zA-Z' \.-]+" />
        </restriction>
    </simpleType>
</element>
```

Hopefully I have convinced you that in 2013 the state of the art for validating data is (still) regular expressions.

Now we must address the question: So what?

It is important to understand why this is a problem. Consider this scenario: The value of an element is a SQL expression. As we've just seen, the most powerful tool XML Schema provides for checking that SQL

expression is a regular expression. However, regular expressions have the weakest computational power and are not powerful enough to deal with SQL expressions, which have (at least) the power of a Recursive language. In this fabulous paper [3] Dartmouth security researchers talk about the security risk with parsing input that has strength A using a parser that has strength B, where A is greater than B. That is:

> Input language power > Parser language power

Here is what they write regarding validating SQL using a regular expression [3]:

> SQL injection attacks have been extremely successful, due to both the complicated syntax of SQL and application developers' habit of sanitizing SQL inputs by using regular expressions to ban undesirable inputs, whereas regular expressions are not powerful enough to validate non-regular languages.

That is just one example. There are countless other examples. The problem is the input data has more computational power than the regular expressions used to validate the data.

Now you may argue, "Ah, but XML Schema 1.1 has assertions and assertions use XPath and with XPath one can supplement the regular expression language with additional computing power." No, not really. XPath is great for markup-level processing but it's not a very good text processing language. Not only that, consider the primary text-processing functions in XPath: matches(), tokenize(), and replace(). Guess what, they all use regular expressions!

Protect Your System

The bottom line is this:

> ***Validating XML with regexp-based XML Schemas leaves your system vulnerable to input processing attacks.***

Here is how to protect your system: Design the XML Schema so that every string-based element and attribute is constrained using enumeration facets. Here is an example schema element declaration:

```
<element name="Surname">
    <simpleType>
        <restriction base="string">
            <enumeration value="Smith" />
            <enumeration value="Jones" />
            <enumeration value="Costello" />
        </restriction>
```

```
        </simpleType>
    </element>
```

With that approach you have taken away computational power in the data. Another way of stating this is that with this approach you have defined a sublanguage which only generates safe XML. If you require additional computational power then express it in markup (as I describe in [2]).

References

[1] The paper *Guns and Butter: Towards Formal Axioms of Input Validation* may be found at this URL: http://www.blackhat.com/presentations/bh-usa-05/BH_US_05-Hansen-Patterson/HP2005.pdf

[2] The xml-dev post titled *Computational Power Required Of XML Languages And The Insecurity Thereof* may be found at this URL: http://www.stylusstudio.com/xmldev/201304/post60000.html

[3] The paper *The Halting Problems of Network Stack Insecurity* may be found at this URL: http://www.cs.dartmouth.edu/~sergey/langsec/papers/Sassaman.pdf