CreatingRegularExpressions

Creating Regular Expressions

Regular expressions can be a bit daunting at first, so here is a suggested procedure to help create them.

Getting Started

Ensure you have an exact copy of the source document that you want to operate on.

One way to do this is to attach a Save Responses to a file Listener to the Sampler, and run the test to create a copy of the resource.

You can then use the HTTP Sampler with the "file:" protocol to retrieve the page at any time.

Extract the section you are interested in

Find the variable part that you want to extract (in the file, or in the Tree View Listener), and start by using that as the regular expression.

For example, suppose you want to find the value from the following snippet:

```
input type="hidden" name="secret" value="CAFEBABE.12345(3)"
```

Start with exactly that as the regular expression, and check that it works, for example in the Tree View Listener Regex tester panel.

If not, examine the expression for any meta-characters (characters that have a special meaning in regexes).In this case, the only special characters are the "." and the parentheses "(" and ")".These need to be escaped, by being prefixed with "\". We now have:

```
input type="hidden" name="secret" value="CAFEBABE\.12345\(3\)"
```

This should now match the whole phrase.

The next stage is to tell the regex processor which part of the section you want to use. This is easy, just enclose the characters in parentheses.

So assume you want to match just

```
CAFEBABE.12345
```

Your regular expression then becomes:

```
input type="hidden" name="secret" value="(CAFEBABE\.12345)\(3\)"
```

Fix the expression so it matches variable text

Of course, the previous expression is not much use, as the text it matches is already known. We want to match variable text, so we have to replace the fixed characters of the target text with meta-character expressions that will match all possible variations of the target.

This requires knowledge (or a good guess) as to what possible characters can be used in the target.

In this case, it looks as if there is a string of hex characters followed by a number, followed by a digit in parentheses.

A digit is easy, that's "`\d`", so a number is "`\d+`", where the "+" means one or more of the previous item.

A hex character can be represented by "`[0-9A-Za-z]`". The enclosing "`[]`" create a *character class* which matches one of the specified characters. The "`-`" here means a range

So putting that together, we get:

```
input type="hidden" name="secret" value="([0-9A-Za-z]+\.\d+)\(d\)"
```

Now suppose we wanted to match the whole of the value. We could move the closing capture parenthesis to the end of the value. This would be suitable if there were other values with different patterns that we did not want to match.

However, if we just wanted to capture the quoted value, then we could use:

```
input type="hidden" name="secret" value="([^"]+)"
```

The character class in this case is `[^"]` which means any character except double-quote. The "+" suffix means we want as many as there are in succession. This will take us up to the end of the value.

Hints and tips

If the expression matches more than once in the source, you have a choice:

- specify which match to use
- extend the expression to include more context at the beginning or end so the match is unique