
3.9. Securing your application

You may have noticed that this book doesn't have a specific chapter on security. We'd rather leave this extensive and rapidly changing topic to the experts, but we can't write a book on web applications without touching on how GWT can help secure your application.

If you read GWT's "Security for GWT Applications,"[7] you'll see that it summarizes four vectors of attack to which the GWT team feels GWT applications are vulnerable:

[7] "Security for GWT Applications": http://mng.bz/pq07.

- JavaScript on your host page that's unrelated to GWT
- Code you write that sets `innerHTML` on GWT widget objects
- Using the JSON API to parse untrusted strings (which ultimately calls the JavaScript `eval` function)
- JSNI code that does something unsafe

For JavaScript on your host page that's unrelated to GWT, such as a third-party library you're using, you have to make sure you trust that third-party code to do nothing dangerous to your application—maybe an OK assumption for well-known libraries.

This chapter's example is a good demonstration of the second point—using `innerHTML`. Although not explicitly constrained to DOM manipulation, it's always wise to consider if you need to make safe any HTML you're handling to minimize against cross-site scripting (XSS) attacks.

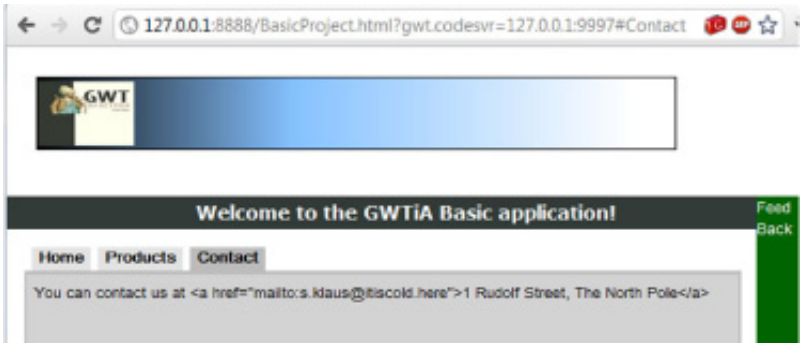
GWT provides a `SafeHtml` object that allows you to sanitize text against cross-site scripting attacks. In `BasicProject`, you use that to sanitize the content you pull out of the HTML and put into content tabs. You might feel safe in `BasicProject`, because you control the HTML file that's manipulated, but it's better to be safe than sorry and guard against anyone sitting in the middle or hacking your HTML file.

At  back in listing 3.4 you take the precaution of using `SafeHtml` on your extracted content:

```
SafeHtml sfHtml = SimpleHtmlSanitizer.sanitizeHtml(toReturn);
```

The impact of using `SimpleHtmlSanitizer` is to stop the contact in the HTML from being a live URL, as you can see in figure 3.12—sometimes safety can get in the way of functionality.

Figure 3.12. The result of using the `SafeHTML` object on the manipulated HTML in `BasicProject`—note that the mailto URL of the contact has been sanitized to plain text and is no longer an active URL (to not use the `SafeHTML` approach, comment out the two lines of code in the example).



As you go through the chapters you'll see time and again that GWT allows you to guard against attacks—although you have to proactively use the

techniques available to you:

- Chapter 4 talks about using `SafeHTML`, `SafeHtmlBuilder`, `SafeHtmlUtils`, `SimpleHtmlSanitizer`, and `SafeHtmlTemplates`—all of which you can use when creating new widgets to protect against hack attacks.
- Chapter 4 also looks at `UriUtils` to ensure a URI is safe; it also notes that you can use `SafeStylesBuilder` and `SafeStylesUtils` to help protect against attacks contained within any injected stylesheets.
- Chapter 10 looks at how you can use `SafeHtmlTemplates` to create new `cell` widgets.
- Chapter 13 looks at how to use `SafeHtml` in internationalization messages to help protect against attacks. In the example it's possible to inject a malicious image tag that runs some JavaScript (fires up an alert box)—a simple example of what and how to protect against.

Unfortunately, XSS isn't the only attack you need to be careful about. Your server-side communication may (will?) need protecting against cross-site request forgery (often referred to as XSRF or CSRF):

- Using `RequestBuilder` (chapter 12) you can set a custom header in the request (using the `setHeader` method) to contain the value of a cookie originally set by the server. On the server side you can compare the value of the cookie sent in the request to the copy of the cookie in the header—if they don't match, then you're being attacked.
- You can also protect RequestFactory (chapter 8) in a similar way by extending the `DefaultRequestTransport` object to include a custom header.
- In chapter 7 we look in detail at how to use GWT's `XsrfProtectedServiceServlet` instead of `RemoteServiceServlet` to gain access to GWT's built-in XSRF protection for GWT-RPC.

We've arrived at the end of this chapter with an application resembling the mock-up from the start—figure 3.1 (and it would probably look even prettier if we were designers and not developers).