Vulnerabilities and Flapjax

Søren Krogh - 20105661
Emil Madsen - 20105376
K. Rohde Fischer - 20052356

December 10, 2014

Exploiting Gruyere

Cross-Site Scripting

The cross site scripting is really a set of attacks, because if executed correctly. The attacks can be used to achieve any number of goals. Most of them are based of taking completely control of the site. For instance the fact that file upload can be used to upload websites enables us to set up a fake site that in all ways looks authentic, because it by all means of validations is part of the actual page.

Quite a few of the attacks uses the way data is inserted into the site to inject JavaScript, for instance by exploiting the way attributes are written to insert a harmful script. For example if the color is set to #" $onmouseover = "alert(42)$ the tag for the user name will have an onmouseover-attribute that can execute harmful code.

Also sending users URLs can enable execution of harmful code and also the AJAX calls can be exploited because they execute the code returned (that should be clean JSON, but might not be).

The XSS attacks provides a quite dangerous set of attacks as they potentially provide full control over what the client does from the second it is executed. The attack can basically just add a script tag providing the full attack code.

Client-State manipulation

Manipulating the client stage provides is a way to forge requests, the typical usage would be either escalating the users privileges such as in the Gruyere example or for manipulating sites to have a wrong behavior, such as a shop thinking the total price of a shopping basket it 0.

Cross-Site Request Forgery

The XSRF attack is a bit similar to the XSS, as it also provides a way to make the user execute unintended actions. The Gruyere example is deleting a snippet, but could also be to make users upload intended things.

An interesting detail in the type of attack is that it can for instance be used in cases where you don't have admin rights, but need the admins permissions to do other attacks such as XSS. Also due to the fact the the simplest way to

perform an XSRF is by the get parameters, a lot of developers is wrongly let to believe post is more secure. This however is not true as the attacking site could have a hidden form that submits to the site under attack and then having a JavaScript that automatically submits.

Path Traversal

Path traversal is used to either reveal secret info or to place/replace files in the system. This can be anything from replacing a list of users to replacing for instance the index-page of a site or even worse in the cases where the uploads are not properly checked even uploading files that will be executed by the server. This can in worst case be used to replace the entire code base of a site with the code intended by a malicious person.

XSS and XSRF in node.js

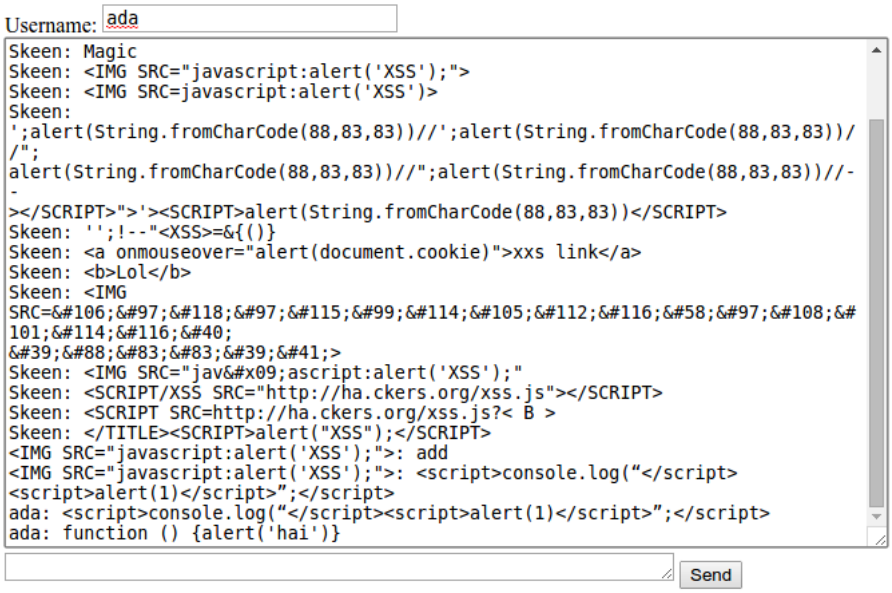
Node.js chat application

We havn't been able to break our chat application. We're not accepting user input, via query arguments and hence the only input method is the chat-username and chat-messages.

Username and message are transfered as JSON, which removes the possibility of directly sending functions to break the server. We've tried several approaches to break the server code, but as it interacts minimally with the send json, there's not a lot of room for error. In essence it just forwards one users JSON message to other users.

Back at the users end, we may now be recieving script tags and whatnot from other users. Once we recieve a message, we do nothing with it, but to append it to the textarea's value. This property is resposible for escaping all attempts at tags, be it scripts or html.

Several of our attempts can be seen on the picture below;



We invite the reviewer to attempt to break the system.

The system does have a somewhat staggering weakness to phishing, as there's no mechanism in place to differentiate between users impersonating one another.

Say Alice is chatting with Bob, and Carol is passivly following the chat. Carol decides to mess with Alice, so she changes her username to Bob, and sends a message, as if she was him. Bob will be able to see this message no problem, but before he gets to warn Alice that the message wasn't send by him, the damage may already have been invoked.

Extensions

A logical extension to avoid the above weakness to phishing would be the introduction of user accounts. This would however introduce all of the XSS weaknesses commonly known to roam in those waters.

And alternative extension which could be troublesome, is to display messages in a different manner, say though the use of lists, and via the `.innerHTML` property. This way we'll have to handle escaping dangerous tags ourselves, which can be error-prone. An argument for doing this, would be to allow for some limited html markup within the messages.

Dart Picture Browser

We've been unable to get our client for the Dart Picture Browser to run. And henceforth unable to test for weaknesses in it's implementation.

Flapjax

Flapjax is a language designed for contemporary web applications. With a little effort, Flapjax can also be used as JavaScript library. This will be covered later in this section.

Flapjax is build on top of JavaScript, this means that it works with existing JavaScript code, and runs in unmodified browsers. As mentioned above, It can be used either as a language or which is then compiled to JavaScript, or it can be used as a library to JavaScript.

One might aske why it was decided not to design a language from scratch but instead to engieneer Flapjax ontop of JavaScript. In the paper "Flapjax: A Programming Language for Ajax Applications" they point out three important benefits of JavaScript; First, it is found in all modern browsers which means that it has become a common language for web applications. Secondly, it reifies the entire content of the current web page into a single data structure called the (DOM)Document Object Model, so that developers con address and modify all aspects of the current page. This also includes the visual style. Thirdly, it provides a primitive, XMLHttpRequest, that permits asynchronous communication without reloading the current page.

Falpjax is a reactive language, which means that it automatically trackes datadependencies and propagates updates along those dataflows. In effect, if a developer defines y = f(x) and if the value x then changes, y is automatically recomputed, which is nice! It does this by basically augmenting JavaScript with two new kinds of data. A behavior is more or less like a variable, it always has a value, but in contrast to a variable changes to a behavior propagates automatically. An event stream is potentially an infinite stream of events whose new events trigger additional computation.

One thing that Flapjax gets rid of is callbacks. To better show this, we present two versions of the same program, the first in JavaScript and the second in Flapjax. The program displays a counter that increments every second until the reset button is pressed.

```
1  var timerID = null;
2  var elapsedTime = 0;
3
4  function doEverySecond() {
5          elapsedTime += 1;
6          document.getElementById("curTime")
7                              .innerHTML = elapsedTime; }
8  function startTimer() {
9          timerId = setInterval("doEverySecond()", 1000); }
10 function resetElapsed() {
11         elapsedTime = 0; }
12
13 <body onload='startTimer()'>
14 <input id="reset" type="button" value="Reset"
15                 onclick="resetElapsed()"/>
16 <div id="curTime" >  </div>
17 </body>
```

In the program above, the variable elapsedTime is set a total of three times, and only used once. The problem here isn't JavaScript, but rather the use of callbacks and the effect they have on the structure of the program. Callbacks are invoked by a generic event loop which has no knowledge of the application's

logic, so it would be meaningless for a callback to compute and return a non-trivial value.

```
var nowB = timer(1000);
var startTm = nowB.valueNow();
varclickTmsB = \$E("reset","click").snapshot(nowB)
                                    .startsWith(startTm);

var elapsedB = now - clickTmsB;
insertValueB(elapsedB, "curTime", "innerHTML");

<body onload="loader()" >
<input id="reset" type="button" value="Reset"/>
<div id="curTime" >  </div>
</body>
```

In the Flapjax version above, we see that callbacks are now absent. The developer simply expresses the dependencies between expressions, and leaves it to the language to schedule updates. Said in other words, the developer has left the maintenance of consistency to the language.

The above is an example of buttons understod as event streams and clocks as behaviors. Flapjax makes it possible for developers to treat all other components of Ajax programs in these terms.

As we mentioned earlier, Flapjax can, with a little extra work, be directly used as a library to JavaScript. With Flapjax as a language, the compiler compiles files containing HTML, JavaScript and Flapjax. The Flapjax code is identified by:

```
<script type="text/flapjax" >
```

The compiler transforms Flapjax code into JavaScript, and produces a standard web page of just HTML and JavaScript. Whitout the compiler, this transformation has to be performed manually. A drawback to using the compiler is, that it adds an additional step to every update during development. Whitout the compiler the developer simply just saves the project and refreshes the browser.