Automated verification of string-manipulating programs

Programs written in scripting languages with heavy use of string variables can easily lead to programming errors, with potentially serious security consequences. A new project will tackle these issues by developing algorithms to analyse how strings are manipulated in a program, and decision procedures to automate this reasoning. Professor Anthony W Lin explains.

Strings have been a fundamental data type in programming languages since the inception of Computer Science. This is true now more than ever with the past decade having witnessed the rapidly growing popularity of scripting languages (for example JavaScript, Python and PHP). JavaScript, for instance, was ranked as the most used programming language worldwide ever by back-end developers in a recent developer survey conducted by Stack Overflow.

Programs written in scripting languages tend to make heavy use of string variables, which are difficult to reason about and could easily lead to programming errors. In some cases, such mistakes could have serious security consequences. For example, in the case of client-side web applications, cross-site scripting (XSS) attacks could lead to a security breach by a malicious user.

The top ten classes of web application security vulnerabilities today include XSS, based on OWASP's (Open Web Application Security Project) most recent studies. These vulnerabilities are typically caused by improper handling of user inputs and submitted requests (in the form of strings, for example 'Donald Duck') by the web applications.

The goal of the project 'Algorithmic Verification of String Manipulating Programs' (for which I am the Principal Investigator) is to develop a constraint language for reasoning about how strings are manipulated in a program, along with decision procedures for automating this reasoning. For instance, the boxed example below requires us to have logic for reasoning about concatenation and finite-state transductions applied to strings. This is a challenging problem in both theory and practice since it requires us to solve some long-standing open problems about constraint solving over strings in the most general case.

The European Research Council is funding the project (November 2017 to October 2022), which aims to make scientific advances towards this in both theory and tool implementation, and apply the results to challenging real-life problems, including the analysis of XSS vulnerabilities in web applications.

Example of an XSS vulnerability

The following JavaScript code snippet is adapted from the paper 'Securing the tangled web' by Christopher Kern in 2014.

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);
catElem.innerHTML = '<button onclick= "createCatList
(\" + y + '\')">' + x + '</button>';
```

The code creates a DOM element catElem by assigning HTML markup. The value for the category cat is provided by an untrusted third party. The code attempts to first sanitise the value of cat. This is done via the Closure Library, string functions, htmlEscape and escapeString. Inputting the value 'Flora & Fauna' into cat gives the desired HTML markup:

```
<button onclick="createCatList('Flora &amp; Fauna')
">Flora &amp; Fauna</button>
```

On the other hand, inputting the value `');attackScript();//` into cat, results in the HTML markup:

```
<button onclick="createCatList('&#39;);
attackScript();//')">&#39;);attackScript();//')</button>
```

When this is inserted into the DOM via innerHTML, an implicit browser transduction will take place, ie, first HTML-unescaping the value inside the onclick attribute and invoking the attacker's script attackScript() after createCatList. This subtle XSS bug (a type of mutation XSS) is due to calling the appropriate escaping functions in the wrong order.