Projet Ajacs

Deliverable WP1

Identification of a sub-language of JavaScript to implement secure libraries

June 2016

A goal of WP1, in the context of the JSCert framework, was to explore what it means to encode sublanguages of JavaScript or intermediate languages targeting JavaScript in Coq and reason about their properties using JSCert. Given the complexity of JSCert and, more generally, the complexity of reasoning in a formal setting like Coq, and for this reason it is necessary to determine an appropriate "sufficiently simple" first candidate sublanguage.

Our first choice was Defensive JavaScript (DJS). DJS is a statically typed subset of JavaScript, introduced by Bhargavan et al. in [1], the purpose of which is to allow programmers to write defensive JavaScript code, i.e. code whose functional behaviour cannot be tampered with even if it is loaded by and executed within a malicious environment under the control of the attacker. Such defensive code would be well-suited, for instance, for writing cryptographic applications, single sign-on widgets, and bookmarklets.

To achieve defensiveness, DJS excludes the bulk of the dynamic features of JavaScript: scoping is static; there is no prototypical inheritance; there are no implicit coercions; objects and arrays are not extensible; dynamic accessors are designed so that the index always falls within bounds, and they are not allowed for property access. To provide full isolation, DJS programs are designed as wrapped functions that only export scalar (`string` $\rightarrow$ `string`) APIs.

On the level of syntax, DJS includes JavaScript literals, a number of unary and binary operators, property access only using the `e.x` notation, several dynamic accessors for arrays and strings that ensure that the index never falls out of bounds, the `with`, `if` − `then` − `else`, `while` and sequence statements, as well as function and program definitions.

This (relatively) simple syntax of DJS, combined with the fact that the semantics of DJS do not depart fom the semantics of JavaScript, make DJS an excellent first candidate for formalisation.

Formalising DJS in Coq

Thus far, we have formalised the following in Coq:[1]

(1) The complete syntax of DJS,

(2) The complete type system of DJS,

(3) The mapping from DJS terms to JSCert terms,

(4) A predicate describing allowed JSCert terms,

(5) A proof that the mapping of (3) and the predicate of (4) are equivalent.

We have done a deep encoding (with respect to JSCert) of the syntax and the type system for DJS. In order to derive proper induction principles, it was necessary for us to define size for DJS types, DJS terms, and JSCert terms. All of the proofs then proceed either by using these induction principles, or by induction on the appropriate size.

[1] Coq v8.4pl6 + TLC library (`http://www.chargueraud.org/softs/tlc/`).

DJS contains in its syntax rules of variable length, and we have used two strategies for encoding them: for function definitions, due to the design of the rule, it was necessary to use a pretty-big-step style; for the rules concerning array and object creation, we have used the `Forall` constructor of the TLC library. Currently, Coq cannot generate proper induction principles for definitions using `Forall`, but this is a non-issue as we are deriving induction principles ourselves.

When defining the mapping from DJS terms to JSCert terms, as well as when describing the allowed JSCert terms, we have taken particular care to follow the specification of DJS, akin to our approach in the construction of JSCert itself.

Our next steps will involve understanding how to relate the typing environments of DJS with the environment records of JSCert, how to describe properties of the JSCert heap, possibly instrumenting JSCert with additional information required for tracing function calls, and proving type safety for DJS programs. Ultimately, what we would like to prove is encapsulation and, most importantly, the defensiveness property.

References

[1] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Sergio Maffeis. Defensive javascript - building and verifying secure web components. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 88–123. Springer, 2013.