JavaScript Analysis Challenges

Dynamic characteristics
- dynamic code generation
- function variadicity
- constructor polymorphism
- dynamic typing
- prototype-based inheritance

Software engineering challenges
- program understanding
- optimization

Examples of key features
- reflective mechanisms:
  - `for(n=1; n<20; n++) { x++ = 'y.p' + n + ':' + myUe(s.prop + n + ')'; eval(x); }`
  - other forms of eval: Function, setTimeout, Write, etc.
  - `setTimeout('sendRequest('+action+','+validate+'','1000)');`
- dynamic object behavior
  - a JavaScript object == {local property | inherited property}
  - `v.p = new A(); delete v['p']; //v.p is undeclared
    v.p = new C();`

Our Solution for `eval`:
**JavaScript Blended Analysis Framework**

Design goal: a practical general-purpose combination of dynamic and static analysis capable of capturing the effects of the dynamic features of JavaScript.

Security Application: **Blended Taint Analysis for JavaScript Websites**

Our Solution for Dynamic Object Behavior:
**State-sensitive Points-to Analysis**

- Partially flow-sensitive analysis via State-Preserving Block Graph (SPBG)
- Calling context: an approximation of the object state of the receiver object
- Extended points-to graph with annotations
- Points-to analysis transfer functions

Program Understanding Application: **REF Analysis Results**

- REF analysis calculates the set of objects returned by property lookup at a property read statement (i.e., `x = y.p`) or call statement (i.e., `x = y.p(...)`).
- The results of REF analysis can be used for building software IDEs supporting smart code completion.
- Comparison: correlation-tracking vs. state-sensitive analysis

Blended Taint Analysis Results

- Static Taint+: JavaScript library + application code
- Static Taint-: application code only

**Blended vs. Static Analysis**

1. Pure Static Analysis
2. 1
3. 2
4. 3
5. Blended Analysis

**Blended Taint Analysis for JavaScript Websites**

**References:**