Web 2.0 applications written in JavaScript handle sensitive information. Information flow analysis is an important security problem.

Information flow...  
...can be direct  
\[ \text{if}(x) \{h\} \text{ else } \{ \text{false} \} \rightarrow h \text{ or } \text{false} \]  
\( h \) directly affects the result

...can be indirect  
\[ \text{if}(h) \{ \text{true} \} \text{ else } \{ \text{false} \} \rightarrow \text{true} \ (\equiv h) \text{ or } \text{false} \ (\equiv h) \]  
\( h \) indirectly affects the result

We want to determine **statically** which subexpressions can affect the result.

**JavaScript is hard!**

- **eval** lets programs run strings as code
  - So use **staged metaprogramming** instead:
    - limit manipulation to splicing well-formed code templates
    - still have to handle static and dynamic scoping and loss of alpha equivalence

**Our Analysis**

1. Mark expressions of interest  2. CFA  3. Generate information flow constraints

\[
((\text{fun}(x)\{x + 1\})
\]  
...evaluates like  
\[
\text{eval}("(\text{fun}(x)\{" + "x + 1" + ")")")
\]  
...with staged metaprogramming  
\[
\text{run } (\text{box } (\text{fun}(x)\{\text{unbox } (\text{box } (x + 1))\}))
\]  
...evaluates like  
\[
((\text{fun}(x)\{1\})
\]  
...with staged metaprogramming  
\[
((\text{fun}(x)\{1\})\{y \rightarrow x\}
\]  
...with staged metaprogramming  
\[
((\text{fun}(x)\{y \rightarrow x\})\{x \rightarrow y\}
\]  
...with staged metaprogramming  
\[
((\text{fun}(x)\{y \rightarrow x\})\{x \rightarrow y\})\{x \rightarrow y\}
\]  
...with staged metaprogramming

**What we have done...**

- first analysis of its kind
- an implementation in OCaml
- a soundness proof in Coq

**Still to do...**

- automate transformation to staged metaprogramming
- improve precision of analysis of strings, numbers...
- support full JavaScript: mutable state, exceptions...