Staged Information Flow for JavaScript

Notes by Matthew Milano and Xiang Long

December 2, 2013

1 Summary

- Static approach for dynamic code
- Mechanism
  - Split programs into stages
  - Context: main program
  - Holes: dynamically loaded portions
- Traditional static analysis not feasible due to dynamically loaded code not known ahead of time.
- Also there are other JS problems: everything is loaded in the same scope, including remote code. No private fields or encapsulation.
- Solution is to divide into stages as above.
- Observation: you can do static flow control without a strongly-typed language.
- We're not enforcing what the *user* thinks of as security, we're enforcing what the *website* thinks of as security.
- Andrew: it's really a richer version of the same-origin policy.
- Some more motivation straight from the paper. Definition of policies. Examples:
  - Do not want remote code to affect browser location or variables flowing to location. Also should not access cookies.
  - Integrity policy: holes cannot write to location.
  - Confidentiality policy: cookies cannot flow to holes.
- The constraint rules: if you squint, it looks a lot like the typing rules we've traditionally seen.

- It is surprising that the "disallow x.cookie bans cookie" policy works in practice.
- It's not clear how coarse their metrics are. They punt on using array syntax to access object fields (like document["cookie"]), suggesting that they "could" just reject it dynamically.
- Summary of results
  - Efficient yet imprecise, it's unclear how useful this is. Performance evaluated on 100 websites.
  - If you make mistakes in the context that's your problem, tool won't help with that.
  - They only track flows to and from holes.
  - Syntax analysis used to prevent renaming in hole code. E.g. all fields named "cookie" are made secret. Obvious problems with that.

2 Discussion

- Any prior work?
  - Yes, Sabelfeld and Russo did some dynamic analysis. More dynamic than current paper, but cheap so not to harm use experience.
  - Yes, #21 in references.
  - Yes, #6 in references.
  - Yes, Andrew's papers on SIF and SWIFT, which targets IS.
- Q: Are these holes particular scripts or any scripts?

  A: Any. Would be hard to make specific.
- Q: Is this also why we have a blacklist instead of a whitelist?

  A: Probably not - we're talking safety here, so in some ways a blacklist is more natural.

  It's not clear [a whitelist] is not just syntactic sugar (as long as you can quantify over the namespace). Also problems with conflicting policies.

  [A whitelist] might be easier for scaling because of the small root set (?), but Owen doesn't really think of it in that way.

  What would whitelist corruption mean? Un-tainted?

  Tom concedes.
- Q: So do you execute "state" returned before checking RPS?

  A: Yes; it's imperative.

- Q: How do others handle the dictionary-style field lookup in objects?

  A: In S3, Arjun uses prefix with lookup indirection.

  Observation: Unless you have a lot of secret fields, this isn't usually a problem.

  Observation: Overwriting the prototype field ("__proto__") is not handled well by this framework (and is an awful thing to do if you're a JavaScript developer). Unfortunately, big frameworks (ex.: Dart compiler) do overwrite this field (Sam knows this because he wrote the code that does it).

- Results discussion
  - The actual enforcement mechanism is mostly access control. Fine for integrity, limiting for confidentiality.
  - A lot of the work is on Ads, which are separable and therefore represent an easy case.
  - There's a "shadow DOM" proposed spec to address this; can specify certain nodes as invisible through DOM. Still seems like access control, not proper information flow.
  - Considering the "this is inferring an access control policy," it's just me and one other person who I don't trust at all. I wish they had taken this step.
  - Fabric does this. need to move from isolated co-located code to code you interact with. Can go way beyond Fabric and this paper. Declassification is hard.
  - Are we fighting a losing battle here? Take the long view: languages come and go. But Tom's worried that JavaScript is becomeing the C of web applications, and so we will be stuck with it for decades to come.