

Language Based Isolation of Untrusted JavaScript

Ankur Taly

Dept. of Computer Science, Stanford University

Joint work with Sergio Maffei (Imperial College London) and
John C. Mitchell (Stanford University)

Outline

- 1 Web 2.0 and the Isolation Problem
 - Web Mashups
 - Security issues in web mashups
 - FBJS and ADSafe
- 2 A bit about *JavaScript*
 - Formal Semantics of JavaScript
 - Formalizing the Isolation Problem
- 3 Solving the Isolation Problem
 - JavaScript facts
 - Achieving Isolation goals
 - Comparison with FBJS
- 4 Glimpse of Current Work
- 5 Conclusions and Future Work

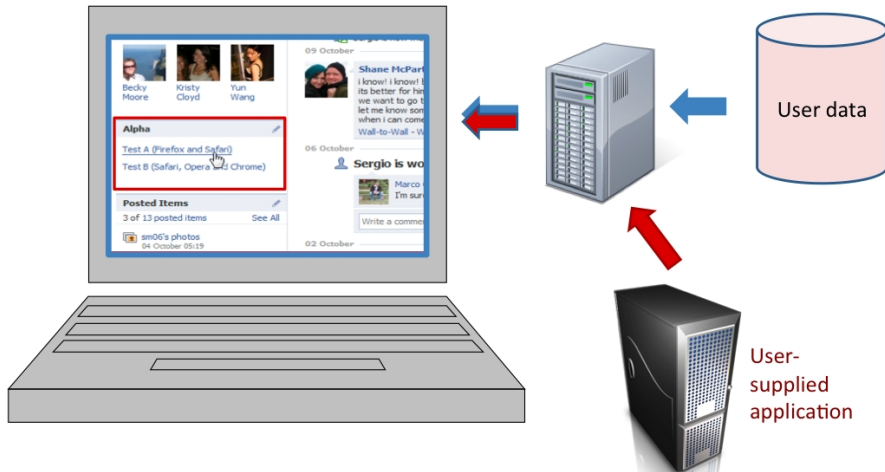
Web 2.0

*All about mixing and merging content (data and code) from multiple content providers in the users browser, to provide high-value applications known as **mashups***

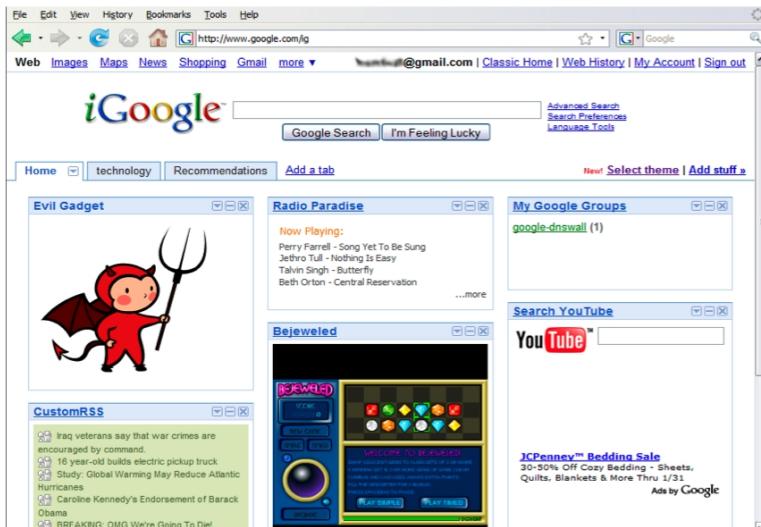
- Notation:
 - Individual contents being mixed - *Components*.
 - Content Providers - *Principals*.
 - Publisher of the mashup- *Host*.
- Execution environment- Web Browser.
- Web page (DOM) - Shared resource.
- Most common language for mashups- **JavaScript**.
- Examples:
 - Basic mashups: Any web page with advertisements, iGoogle.
 - More complex mashups: Yelp, Yahoo Newsglobe ...

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

Architecture



Example: Security Issue?



Example: Security Issue?

INDIANTAGS Home » Register

Sort news by: Recently Popular | [Top Today](#) | [Yesterday](#) | [Week](#) | [Month](#) | [Year](#) |

[Published News](#) | [Upcoming News](#)

Register

Register

Username:
 [Verify](#)

Email:
lowercase letters only
 [Verify](#)

Password:
five character minimum

Verify password:

Please enter the number provided in the image below. If you can not read the number you may refresh your browser.



[Continue](#)

What is INDIANTAGS?

INDIANTAGS is social news submitting site. vote for best

[read more](#)

Fabulous Festive Offers

SHOPPERS STOP
START SOMETHING NEW

- APPAREL *
- ACCESSORIES *
- HOME DÉCOR *
- FRAGRANCES *
- TOYS

SHOP NOW @
www.shoppersstop.com

Security Issues

- Execution of a mashup is not bound to or controlled by any specific principal
- Each principal owns part of the resources and has *integrity* and *confidentiality* constraints over them.
 - Yelp code: Google map scripts should not tamper with search results.
 - Google Map: Yelp code should not re-define any functions defined by google maps.
- A mashup should be designed such that the interests of all principals, including the host are protected.
- High risk associated: Credit card fraud, identity theft, loss of sensitive information
- Cannot afford to miss a single edge case- **Need a definitive proof of correctness.**

Our Model

Basic mashups.

- Two trust levels: *trusted* and *untrusted*.
 - Trusted: Code belonging to host.
 - Untrusted: All third-party code.
- Fix the language to *JavaScript*.
- Untrusted Components are sequentially composed and placed in a trusted context.
- Model fits the case of web pages with advertisements, iGoogle, Facebook Apps.

Isolation Problem

Design isolation mechanisms for untrusted components so that they cannot access security critical resources belonging to the host and also other untrusted components.

Browser Sandbox

- Placing all untrusted content in separate **IFrames** seems to be a safe solution .
 - Every IFrame has its own global object.
 - Same Origin Policy: Only frames displaying content from the same origin can **script** each other.
- Sometimes IFrames are too restrictive !
 - Framed applications are restricted to a confined region of the screen.
 - Full JavaScript can be too powerful (csrf, navigate top, browser exploits, etc.).
 - Interactions with other applications are severely restricted (fragment ids, PostMessage).

JavaScript sandboxing JavaScript

- A different approach.
 - Trusted and untrusted JavaScript run in the same execution environment.
 - Trusted code enforces a software sandbox on the untrusted code.
 - Server-side filtering, and rewriting.
 - Fine grained control on the interaction between applications.
 - Hard to get it right! Needs rigorous analysis.
- Websites using this approach: Facebook FBJS, Yahoo! Adsafe, Google Caja.

Quick Case study: Facebook FBJS

- Basics:
 - Facebook apps are either Iframed or **integrated**. We are interested in integrated apps.
 - Integrated FaceBook applications are written in **FBML/FBJS** Facebook subsets of HTML and JavaScript.
 - FBJS is served from Facebook, after **filtering and rewriting**.
 - Facebook libraries mediate access to the DOM.
- Security goals:
 - No direct access to the DOM.
 - No tampering with the execution environment
 - No tampering with Facebook libraries.
- Isolation approach:
 - **Blacklist variable names** that are used by containing page
 - Prevent access to global scope object, since property names cannot be renamed and variables are properties of scope objects

Quick Casestudy: Yahoo! ADsafe₀₇ (Douglas Crockford)

- Basics:
 - A **safe subset** of JavaScript to be used by **untrusted code not placed in an Iframe**.
 - The host page's server provides the ADSafe object to the untrusted ad code.
 - All interaction with the trusted code happens only using the methods in the ADSafe object.
- Security Goal: **Restrict untrusted code from directly accessing arbitrary elements of the Document Object Model (DOM) and/or the global object.**
- Isolation approach:
 - Force special idioms: ADSafe.get(o,p)
 - Untrusted code can be statically checked to ensure that it only calls methods of the ADSAFE object (**Tool : JSLint**).
 - ADSafe libraries mediate all access to the DOM.

Outline

- 1 Web 2.0 and the Isolation Problem
 - Web Mashups
 - Security issues in web mashups
 - FBJS and ADSafe
- 2 A bit about *JavaScript*
 - Formal Semantics of JavaScript
 - Formalizing the Isolation Problem
- 3 Solving the Isolation Problem
 - JavaScript facts
 - Achieving Isolation goals
 - Comparison with FBJS
- 4 Glimpse of Current Work
- 5 Conclusions and Future Work

A bit about JavaScript

- History :
 - Developed by Brendan Eich at Netscape.
 - Standardized for Browser Compatability : [ECMAScript 262-edition 3](#)
- First class functions, Prototype based language, re-definable object properties.
- Scope Objects/Stack frames can be first class JavaScript objects: Variable names $\Leftrightarrow$ Property names.
- Implicit type conversions which can trigger user code.

```
var y = "a"; var x = {valueOf: function(){ return y;}}  
x = x + 10;  
js> "a10"
```

Formal Semantics of JavaScript

Formalized all of [ECMA-262-3rd](#) edition ($JS_{ecma262}$).

- Small step style operational semantics.
 - Meaning of a program $\Leftrightarrow$ sequence of actions that are taken during its execution.
 - Specify sequence of actions as transitions of an [Abstract State machine](#)
- Developed formal semantics as basis for proofs ([APLAS'08](#))
 - Very long (70 pages of ascii).
 - DOM is just treated as a library object.
 - We experimented with available browsers and shells
 - Defining an operational semantics for a real programming language is hard: sheer size and JavaScript peculiarities.
- Subset of JS adequate for analyzing AdSafe, FBJS, Caja

A glimpse of the rules

State

Program state is represented as a triple $\langle H, l, t \rangle$.

- H : Denotes the Heap, mapping from the set of locations ($\mathbb{L}$) to objects. H_0 is used to denote the initial heap.
 - Objects are maps from property names ($\mathbb{P}$) to values (v).
 - l : Location of the current scope object (or current activation record).
 - t : Term being evaluated.
-
- Atomic transitions: $H, l, t \longrightarrow H', l', t'$
 - Contextual rules:
$$\frac{H, l, t \longrightarrow H', l', t'}{H, l, C[t] \longrightarrow H', l', C[t']}$$
 - Lots of internal terms introduced as part of evaluation trace of top-level user terms.

Formalizing Basic Mashups

Notation:

- Components are Program Terms.
- Resources are pairs of heap addresses and property names:
 $\mathbb{L} \times \mathbb{P}$.
- Actions: $\text{Resources} \times \{r, w\}$. We define $\text{Actions}(H)$ as the set of valid actions for a heap H .
- $\tau(S)$: Reduction trace of state S .
- $\text{Access}(S)$: Precise set of actions performed during single step transition of S (naturally extended to traces).

Mashup

Given components $t_1, \dots, t_n$ provided by principals $id_1, \dots, id_n$, $\text{Mashup}((t_1, id_1), \dots, (t_n, id_n))$ is the program term $\langle t_1 \rangle_{id_1}; \dots; \langle t_n \rangle_{id_n}$.

Formalizing the Isolation Problem

- Given a trace $\tau_{\text{mash}} = \tau(H, t_{\text{mash}})$ of a mashup, define $\text{States}(\tau_{\text{mash}}, id_k)$ as the sub-trace corresponding to id_k .
- CanInfluence**: For any two actions a_1 and a_2 ,
 $a_1 \triangleright a_2$ iff $\exists p : a_1 = (p, w) \wedge a_2 = (p, r)$.
- $\mathcal{A}_{\text{forbid}}$: Forbidden actions.
 Example: $(\text{window}, \text{"location"}, w); (\text{window}, \text{"eval"}, r) \dots$

Isolation Property

Given an initial heap H_{mash} and a mashup $t_{\text{mash}} = \text{Mashup}((t_1, id_1), \dots, (t_n, id_n))$, let $\tau_{\text{mash}} = \tau(H_{\text{mash}}, t_{\text{mash}})$. $\text{Isolation}(\mathcal{A}_{\text{forbid}}, H_{\text{mash}}, t_{\text{mash}})$ holds iff

- Isolating host**: For all i , $\text{Access}(\text{States}(\tau_{\text{mash}}, id_i)) \cap \mathcal{A}_{\text{forbid}} = \emptyset$
- Isolating untrusted components from each other**: For all $i \neq j$:
 $\text{Access}(\text{States}(\tau_{\text{mash}}, id_i)) \not\triangleright \text{Access}(\text{States}(\tau_{\text{mash}}, id_j))$

Isolation Problem

Find an initial heap H_{mash} and enforcement functions $\text{Enf}_1, \dots, \text{Enf}_n$ such that $\text{Isolation}(\mathcal{A}_{\text{forbid}}, H_{\text{mash}}, \text{Mashup}((\text{Enf}_1(t_1), id_1), \dots, (\text{Enf}_n(t_n), id_n)))$ holds.

Formalizing the Isolation Problem

- Given a trace $\tau_{\text{mash}} = \tau(H, t_{\text{mash}})$ of a mashup, define $\text{States}(\tau_{\text{mash}}, id_k)$ as the sub-trace corresponding to id_k .
- CanInfluence**: For any two actions a_1 and a_2 ,
 $a_1 \triangleright a_2$ iff $\exists p : a_1 = (p, w) \wedge a_2 = (p, r)$.
- $\mathcal{A}_{\text{forbid}}$: Forbidden actions.
 Example: (window,"location",w); (window, "eval",r) ...

Isolation Property

Given an initial heap H_{mash} and a mashup $t_{\text{mash}} = \text{Mashup}((t_1, id_1), \dots, (t_n, id_n))$, let $\tau_{\text{mash}} = \tau(H_{\text{mash}}, t_{\text{mash}})$. $\text{Isolation}(\mathcal{A}_{\text{forbid}}, H_{\text{mash}}, t_{\text{mash}})$ holds iff

- Isolating host**: For all i , $\text{Access}(\text{States}(\tau_{\text{mash}}, id_i)) \cap \mathcal{A}_{\text{forbid}} = \emptyset$
- Isolating untrusted components from each other**: For all $i \neq j$:
 $\text{Access}(\text{States}(\tau_{\text{mash}}, id_i)) \not\triangleright \text{Access}(\text{States}(\tau_{\text{mash}}, id_j))$

Isolation Problem

Find an initial heap H_{mash} and enforcement functions $\text{Enf}_1, \dots, \text{Enf}_n$ such that $\text{Isolation}(\mathcal{A}_{\text{forbid}}, H_{\text{mash}}, \text{Mashup}((\text{Enf}_1(t_1), id_1), \dots, (\text{Enf}_n(t_n), id_n)))$ holds.

Formalizing the Isolation Problem

- Given a trace $\tau_{\text{mash}} = \tau(H, t_{\text{mash}})$ of a mashup, define $\text{States}(\tau_{\text{mash}}, id_k)$ as the sub-trace corresponding to id_k .
- CanInfluence**: For any two actions a_1 and a_2 ,
 $a_1 \triangleright a_2$ iff $\exists p : a_1 = (p, w) \wedge a_2 = (p, r)$.
- $\mathcal{A}_{\text{forbid}}$: Forbidden actions.
 Example: (window,"location",w); (window, "eval",r) ...

Isolation Property

Given an initial heap H_{mash} and a mashup $t_{\text{mash}} = \text{Mashup}((t_1, id_1), \dots, (t_n, id_n))$, let $\tau_{\text{mash}} = \tau(H_{\text{mash}}, t_{\text{mash}})$. $\text{Isolation}(\mathcal{A}_{\text{forbid}}, H_{\text{mash}}, t_{\text{mash}})$ holds iff

- Isolating host**: For all i , $\text{Access}(\text{States}(\tau_{\text{mash}}, id_i)) \cap \mathcal{A}_{\text{forbid}} = \emptyset$
- Isolating untrusted components from each other**: For all $i \neq j$:
 $\text{Access}(\text{States}(\tau_{\text{mash}}, id_i)) \not\triangleright \text{Access}(\text{States}(\tau_{\text{mash}}, id_j))$

Isolation Problem

Find an initial heap H_{mash} and enforcement functions $\text{Enf}_1, \dots, \text{Enf}_n$ such that $\text{Isolation}(\mathcal{A}_{\text{forbid}}, H_{\text{mash}}, \text{Mashup}((\text{Enf}_1(t_1), id_1), \dots, (\text{Enf}_n(t_n), id_n)))$ holds.

Outline

- 1 Web 2.0 and the Isolation Problem
 - Web Mashups
 - Security issues in web mashups
 - FBJS and ADSafe
- 2 A bit about *JavaScript*
 - Formal Semantics of JavaScript
 - Formalizing the Isolation Problem
- 3 Solving the Isolation Problem
 - JavaScript facts
 - Achieving Isolation goals
 - Comparison with FBJS
- 4 Glimpse of Current Work
- 5 Conclusions and Future Work

Solving the problem

Approach:

- First focus on Goal 1 (Isolating host)
- **Blacklist**: $\mathcal{B} = \{p \mid \exists l : ((l, p), r/w) \in \mathcal{A}_{\text{forbid}}\}$
(Properties corresponding to all forbidden read actions).
- Design a sublanguage $J_{\text{safe}}(\mathcal{B}) \subseteq JS_{\text{ecma262}}$, an initial heap H_{mash} and an enforcement function Enf such that for all terms $t \in J_{\text{safe}}(\mathcal{B})$, $Enf(t)$ never accesses any property from the black-list $\mathcal{B}$.
- The Enf function is essentially a **source-to-source translation**.
- Developed in a series of papers: **CSF'09**, **W2SP'09**, **ESORICS'09**.

JavaScript facts

Property accesses

- ① Explicit property access:
 - As identifiers(x), via dot mechanism ($e.x$), or object indexing $e1[e2]$. Example: `window["loc" + "ation"]`.
- ② Implicit property access:
 - Properties not named in the term being evaluated, accessed as part of the evaluation semantics.
 - Example: `toString`, `valueOf`, ... Let $\mathcal{P}_{nat}$ be the set of all property names that can be implicitly accessed.

Dynamic code generation

Occurs only through native functions `eval`, `Function` (and indirectly `constructor` property).

Enforcement

Make use of [Filtering](#) and [Rewriting](#).

Filter 1

Filter all terms containing an identifier or property name from $\mathcal{B}_{mash} \cup \{eval, Function, constructor\}$ and also any \$-prefixed property name.

- The above filter takes care of explicit property access via x and $e.x$.
- To control $e1[e2]$ we adopt Facebook's approach - rewrite to $e1[IDX(e2)]$ ([runtime monitor](#)).
- IDX checks if $e2$ is a blacklisted string.
- What if $e2$ is an object ? - implicit type conversion

Enforcement

Make use of [Filtering](#) and [Rewriting](#).

Filter 1

Filter all terms containing an identifier or property name from $\mathcal{B}_{mash} \cup \{eval, Function, constructor\}$ and also any \$-prefixed property name.

- The above filter takes care of explicit property access via x and $e.x$.
- To control $e1[e2]$ we adopt Facebook's approach - rewrite to $e1[IDX(e2)]$ ([runtime monitor](#)).
- IDX checks if $e2$ is a blacklisted string.
- [What if \$e2\$ is an object ? - implicit type conversion](#)

FBJS IDX function

- **IDX(e): Semantics-**
 - 1 Evaluate e
 - 2 Convert to string
 - 3 If blacklisted return "bad", else return result(1).
- **Attack:** Pass an object as argument, which returns two different values when converted to a string successively.

Attack !!! (Safari)

```
var obj = GET_SCOPE;  
  
obj.toString=function(){this.toString = function(){return 'constructor'}}  
;return 'foo';  
  
var f=function(){}; f[obj]('alert(0)')();
```

Our IDX function

We need some auxiliary variables:

Init 1

```
var $String = String;
var $BL = {p1:true,...,pn:true, eval:true,...,$:true,...}
```

Rewrite 1

Rewrite every occurrence of $e1[e2]$ by $e1[IDX(e2)]$

```
IDX(e2) = ($=e2.toString;function()return ($=$String($),CHECK_$))
CHECK_$ = ($BL[$] ? "bad":
    ($ == "constructor" ? "bad": $ == "eval" ? "bad":
    ($ == "Function" ? "bad": ($[0] == "$" ? "bad": $))))
```

Our rewriting $e1[e2] \rightarrow e1[IDX(e2)]$ faithfully preserves the semantics: $e1[e2] \rightarrow va1[e2] \rightarrow va1[va2] \rightarrow o[va2] \rightarrow o[m]$

Evaluation

- Define $J_{safe}(\mathcal{B})$ as $JS_{ecma262}$ with *Filter* 1 applied.
- Let Enf_1 be the function *Rewrite* 1.
- Let H_1 be the heap obtained after executing *Init* 1 on the initial $JS_{ecma262}$ heap H_0 .

Theorem

For all user terms $t \in J_{safe}(\mathcal{B})$, $\tau(H_1, Enf_1(t))$ does not involve access to any resource corresponding to black-list $\mathcal{B}$.

- If the code does not access a blacklisted property, our enforcement is faithful to the intended semantics
- Goal 1 (Isolating Host) is achieved.

Isolating one component from another (Goal 2)

- **Main Issue:** All components execute in the same global scope.
- Separate the resources corresponding to the global object.
- If we restrict components from getting a handle to the global object, then the only resources accessible are the ones corresponding to the identifiers in the term.

Approach

- 1 Restrict getting a handle to the global object.
- 2 Rename all identifiers present in the term (except certain native property names).

Preventing access to Global Object

Global object can be accessed by

- 1 Using the keyword `this`.
- 2 Calling native methods of the form `function() ( ... return this)`.
- 3 Rewrite `this`, Wrap methods

Rewrite 1

Rewrite every occurrence of `this` to `NOGLOBALTHIS`.

```
NOGLOBALTHIS = (this==$g?null;this)
```

Save global object in `$g`.

Init 1

```
var $g = this;
```

Wrapping native methods

- Method `valueOf` of `Object.prototype` and `sort`, `concat`, `reverse` of `Array.prototype` can potentially return pointer to global object.

Init 2

```
$OPvalueOf = Object.prototype.valueOf;  
$OPvalueOf.call = Function.prototype.call;  
Object.prototype.valueOf =  
function(){var $= $OPvalueOf.call(this); return ($==$g?null:$)}
```

- Similarly *Init3*, 4, 5 for `sort`, `concat`, `reverse`.
- Wrapping `eval` and `Function`: doable, but need to define (and prove safe and correct!) a JavaScript expression that parses, filters and rewrites strings meant to representing JavaScript terms. (`constructor` will be the only thing left then!)

Evaluation

Rewrite 3

Rewrite every identifier x , which is not a native property name, in a term t from principal id_k by $id_k x$.

- Let H_2 be the heap obtained after executing $Init2, \dots, 5$.
- Let Enf_2 be Enf_1 composed with *Rewrite 2* and *Rewrite 3*.

Theorem

For all user terms $t \in J_{safe}(\mathcal{B})$, final value obtained on running $Enf_2(t)$ on H_2 is never the global object.

Comparison with FBJs

● Key features of FBJs enforcement mechanism.

- All application variables get prefixed by an application-specific identifier: `var x;` becomes `var a12345_x;`
- Global object isolated by analogous to NOGLOBAL check on this.
- Blacklisting enforced by filtering, and rewriting similar to `e1[IDX(e2)]`.
- Wrap `valueOf`, `sort`, `concat`, `reverse` and also many other DOM functions.

● Bugs reported

- We found many bugs in the Facebook IDX function and some wrapper functions (`Date`, `AJAX`).
- **Exploit:** A facebook app that could access the `window` object.
- Potential for damage is considerable.
 - Steal cookies or authentication credentials (attacker owns the whole Facebook page, facilitating phishing).
 - Impersonate user: deface or alter profile, query personal information, spam friends, spread virally.

Comparison with FBJS

- **Key features of FBJS enforcement mechanism.**

- All application variables get prefixed by an application-specific identifier: `var x;` becomes `var a12345_x;`
- Global object isolated by analogous to NOGLOBAL check on this.
- Blacklisting enforced by filtering, and rewriting similar to `e1[IDX(e2)]`.
- Wrap `valueOf`, `sort`, `concat`, `reverse` and also many other DOM functions.

- **Bugs reported**

- We found many bugs in the Facebook IDX function and some wrapper functions (`Date`, `AJAX`).
- **Exploit:** A facebook app that could access the `window` object.
- Potential for damage is considerable.
 - Steal cookies or authentication credentials (attacker owns the whole Facebook page, facilitating phishing).
 - Impersonate user: deface or alter profile, query personal information, spam friends, spread virally.

Comparison with FBJS

Our solution: Subset $J_{safe}(\mathcal{B})$, enforcement function Enf_2 and initial heap H_2 .

- Similar to the overall FBJS subset and enforcement.
- Systematically derived, proof of correctness.
- Increases confidence in the FBJS isolation mechanism.
- Limitation: we do not deal with DOM wrapping.

Recall Goal 2:

$Access(States(\tau_{mash}, id_i)) \not\vdash Access(States(\tau_{mash}, id_j))$

Have we achieved Goal 2 ?

No ! Untrusted components still share native resources

App1: `toString.channel = "message"`

App2: `var recv = toString.channel`

Comparison with FBJS

Our solution: Subset $J_{safe}(\mathcal{B})$, enforcement function Enf_2 and initial heap H_2 .

- Similar to the overall FBJS subset and enforcement.
- Systematically derived, proof of correctness.
- Increases confidence in the FBJS isolation mechanism.
- Limitation: we do not deal with DOM wrapping.

Recall Goal 2:

$Access(States(\tau_{mash}, id_i)) \not\vdash Access(States(\tau_{mash}, id_j))$

Have we achieved *Goal 2* ?

No ! Untrusted components still share native resources

App1: `toString.channel = "message"`

App2: `var recv = toString.channel`

Comparison with FBJS

Our solution: Subset $J_{safe}(\mathcal{B})$, enforcement function Enf_2 and initial heap H_2 .

- Similar to the overall FBJS subset and enforcement.
- Systematically derived, proof of correctness.
- Increases confidence in the FBJS isolation mechanism.
- Limitation: we do not deal with DOM wrapping.

Recall Goal 2:

$Access(States(\tau_{mash}, id_i)) \not\vdash Access(States(\tau_{mash}, id_j))$

Have we achieved *Goal 2* ?

No ! Untrusted components still share native resources

App1: `toString.channel = "message"`

App2: `var recv = toString.channel`

Outline

- 1 Web 2.0 and the Isolation Problem
 - Web Mashups
 - Security issues in web mashups
 - FBJS and ADSafe
- 2 A bit about *JavaScript*
 - Formal Semantics of JavaScript
 - Formalizing the Isolation Problem
- 3 Solving the Isolation Problem
 - JavaScript facts
 - Achieving Isolation goals
 - Comparison with FBJS
- 4 Glimpse of Current Work
- 5 Conclusions and Future Work

Back to Foundations

Recall: Isolation Property ($Isolation(\mathcal{A}_{forbid}, H_{mash}, t_{mash})$)

① Isolating host:

$$Access(States(\tau_{mash}, id_i)) \cap A_{forbid} = \emptyset$$

② Isolating untrusted components from each other:

$$Access(States(\tau_{mash}, id_i)) \not\supseteq Access(States(\tau_{mash}, id_j))$$

- Isolation Property can be stated in language independent manner.
- Need to somehow separate the set of privileges held by any two components.
- **Object Capabilities** seem like a promising approach !
 - Several systems and languages based on the model
 - **No formal definitions, proofs.**
- Separate the capabilities held by components (**Google Caja**).

Understanding the Object Capability Model

- Key components
 - **Resources**: Passive entities containing information (heap positions).
 - **Subject**: Active entities that access resources (program terms).
 - **Authority of a subject**: Over-approximation of the set of actions a subject can perform on the set of resources.
- Authority of a subject can change due to actions performed by other subjects.
- Two fundamental properties:
 - 1 *Only Connectivity begets Connectivity.*
 - 2 *No Authority Amplification*
- The above two property can be used to define a weaker *Authority-Safety* property which is sufficient for isolation.

Authority Safety

Authority

Given a heap H and a term t , $Auth(H, t)$ is any over-approximation of the set of actions performed in $\tau(H, t)$.

Authority-Safety

A language is said to be authority-safe if there exists an authority map such that:

- 1 (Connectivity begets connectivity) For all terms u

$$Access(H, t) \not\preceq Auth(H, u) \Rightarrow Auth(H, u) = Auth(K, u)$$

- 2 (No Authority Amplification) For all terms u ,

$$\begin{array}{ccc} Access(H, t) & \triangleright & Auth(H, u) \\ & \Downarrow & \\ Auth(K, u) & \subseteq & Auth(H, u) \cup Auth(H, t) \cup \\ & & (Actions(K) \setminus Actions(H)) \end{array}$$

Analysis

Authority Isolation

$AuthIsolation(\mathcal{A}_{forbid}, H, t_1, \dots, t_n)$ holds iff

- ① $\forall i : \forall j \neq i : Auth(H, t_i) \not\supseteq Auth(H, t_j)$ AND
- ② $\forall i : Auth(H, t_i) \cap \mathcal{A}_{forbid} = \emptyset$

Theorem

If $Auth$ is a valid authority map such that $AuthoritySafe(Auth)$ holds and $t_{mash} = Mashup((t_1, id_1), \dots, (t_n, id_n))$ is a mashed term then

$AuthIsolation(\mathcal{A}_{forbid}, H, t_1, \dots, t_n) \Rightarrow Isolation(\mathcal{A}_{forbid}, H, t_{mash})$

- Authority-isolation is closed under reduction
- Justifies the one-time source-to-source translation approach.

Further Work

- The enforcement function Enf_2 does not ensure complete authority isolation as communication channels are possible via native resources. **Fix:**
 - Make native resources *read-only*
 - Wrap native functions such that there are no implicit writes to global object (Array push and pop).
- **Authority Leaks:** True authority is more than expected authority.
 - All the FBJS attacks are essentially authority leaks.
 - Apps are isolated according to expected authority but not true authority.
- Formalized rigorous definition of **Object-Capability-Safety:**
 - Rigorous programming model for enforcing Authority-Safety.
 - Theorem: Object-Capability-Safety $\Rightarrow$ Authority-Safety.
- In submission to Oakland'09.

Further Work

- The enforcement function Enf_2 does not ensure complete authority isolation as communication channels are possible via native resources. **Fix:**
 - Make native resources *read-only*
 - Wrap native functions such that there are no implicit writes to global object (Array push and pop).
- **Authority Leaks:** True authority is more than expected authority.
 - All the FBJS attacks are essentially authority leaks.
 - Apps are isolated according to expected authority but not true authority.
- Formalized rigorous definition of **Object-Capability-Safety:**
 - Rigorous programming model for enforcing Authority-Safety.
 - Theorem: Object-Capability-Safety $\Rightarrow$ Authority-Safety.
- In submission to Oakland'09.

Further Work

- The enforcement function Enf_2 does not ensure complete authority isolation as communication channels are possible via native resources. **Fix:**
 - Make native resources *read-only*
 - Wrap native functions such that there are no implicit writes to global object (Array push and pop).
- **Authority Leaks:** True authority is more than expected authority.
 - All the FBJS attacks are essentially authority leaks.
 - Apps are isolated according to expected authority but not true authority.
- Formalized rigorous definition of **Object-Capability-Safety:**
 - Rigorous programming model for enforcing Authority-Safety.
 - Theorem: Object-Capability-Safety $\Rightarrow$ Authority-Safety.
- In submission to Oakland'09.

Conclusions and Future Work

- We used programming language techniques to study safe JavaScript subsets.
 - Results validated by experiment.
 - Found bugs in FBJS, ADSafe
 - Provably correct solutions (up to browser implementation).
- Ongoing work:
 - Formalized the notion of Object-capability-safety and Authority-safety.
 - First cut at a proof of concept for Google Caja.
- Future work:
 - We plan to write the *JavaScript* semantics in machine readable format so that the proofs can be automated.
 - Formalize the concept of *Defensive correctness* and its connection with Object-capability-safety .

Thank You !