Language Based isolation of Untrusted JavaScript

Ankur Taly

Dept. of Computer Science, Stanford University

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

1. Web 2.0 and the Isolation Problem
2. Case Study : FBJS
    - Design
    - Attacks and Challenges
3. Formal Semantics of JavaScript
4. Achieving the Isolation goal
5. Ongoing and Future Work

Web 2.0 and the Isolation Problem

Web 2.0 : *All about mixing and merging content (data and code) from multiple content providers in a users browser, to provide high-value applications*

- Extensive Client-side scripting - lots of JavaScript.
- Systems have complex trust boundaries.
- Security Issues

This work

- Focus on the simple case where content providers are either trusted or untrusted : Third party Advertisements , Widgets, Social Networking site - applications.
- Assume the publisher has access to untrusted content before it adds it to the page.
- Focus on JavaScript content present in untrusted code.

Web 2.0 and the Isolation Problem

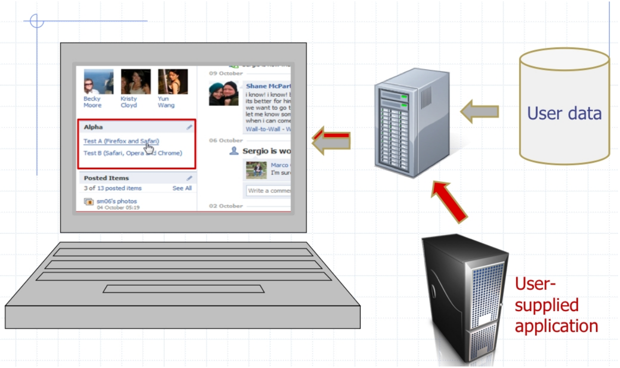
Web 2.0 : *All about mixing and merging content (data and code) from multiple content providers in a users browser, to provide high-value applications*

- Extensive Client-side scripting - lots of JavaScript.
- Systems have complex trust boundaries.
- Security Issues

This work

- Focus on the simple case where content providers are either trusted or untrusted : Third party Advertisements , Widgets, Social Networking site - applications.
- Assume the publisher has access to untrusted content before it adds it to the page.
- Focus on JavaScript content present in untrusted code.

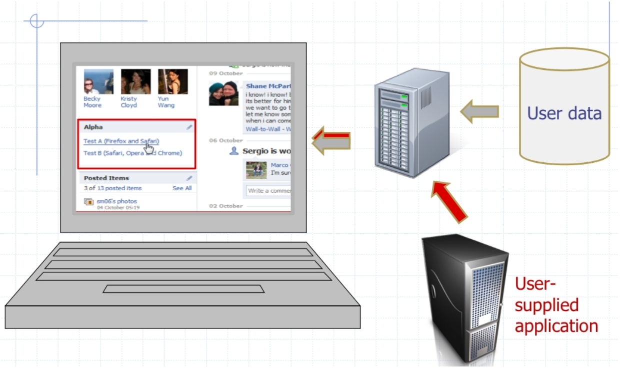
Web 2.0 and the Isolation Problem



Isolation Problem

Design security mechanisms which allow untrusted code to perform valuable interactions and at the same time prevent intrusion and malicious damage.

Web 2.0 and the Isolation Problem



Isolation Problem

Design security mechanisms which allow untrusted code to perform valuable interactions and at the same time prevent intrusion and malicious damage.

IFrames

- Placing all untrusted content in separate IFrames seems to be a safe solution .
- Social network site applications and Ads : IFrames are sometimes too restrictive
  - Restricts the ad to a delineated section of the page.
  - Social network applications need more permissive interaction with the host page.
- Some publishers prefer to *not use* IFrames
  - Gives better control over untrusted code.
  - Easier to restrict same-origin untrusted code.

This Work

Design isolation mechanisms for untrusted code not placed in separate IFrames.

Program Analysis Problem

Program Analysis Problem

Given an untrusted JavaScript program P and a Heap H (corresponding to the trusted page), design a procedure to either statically or dynamically via run time checks, guarantee that P does not access any security critical portions of the Heap.

- Design static analysis and/or code instrumentation techniques
- Very hard problem to solve for whole of JavaScript as all code that gets executed may not appear textually !

```
var m = "toS"; var n = "tring";
 Object.prototype[m + n] = function(){return undefined};
```

Approach

Solve the above problem for subsets of JavaScript that are more amenable to static analysis.

Program Analysis Problem

Program Analysis Problem

Given an untrusted JavaScript program P and a Heap H (corresponding to the trusted page), design a procedure to either statically or dynamically via run time checks, guarantee that P does not access any security critical portions of the Heap.

- Design static analysis and/or code instrumentation techniques
- Very hard problem to solve for whole of JavaScript as all code that gets executed may not appear textually !

```
var m = "toS"; var n = "tring";
 Object.prototype[m + n] = function(){return undefined};
```

Approach

Solve the above problem for subsets of JavaScript that are more amenable to static analysis.

Case Study : *FBJS*

- *FBJS* is a subset of JavaScriptfor writing Facebook applications which are placed as a subtree of the page.

Restrictions Applied

Filtering : Application code must be written in *FBJS*

- Forbid eval, Function constructs.
- Disallow explicit access to properties (via the dot notation -o.p) __parent__, constructor, ...

Rewriting

- this is re-written to ref(this)
  - ref(x) is a function defined by the host (Facebook) in the global object such that ref(x) = x if x ≠ window else ref(x) = null
  - Prevents application code form accessing the global object.

Case Study : *FBJS*

- *FBJS* is a subset of JavaScriptfor writing Facebook applications which are placed as a subtree of the page.

Restrictions Applied

Filtering : Application code must be written in *FBJS*

- Forbid eval, Function constructs.
- Disallow explicit access to properties (via the dot notation -o.p) __parent__, constructor, ...

Rewriting

- this is re-written to ref(this)
  - ref(x) is a function defined by the host (Facebook) in the global object such that ref(x) = x if x $\neq$ window else ref(x) = null
  - Prevents application code form accessing the global object.

Case Study : *FBJS*

Rewriting (contd):

- o[p] is rewritten to o[idx(p)] : Controls access to dynamically generated property names.
  - idx(p) is a function defined by the host (Facebook) in the global object such that idx(p) = bad if p ∈ Blacklist else idx(p) = p.
  - Blacklist contains sensitive property names like __parent__, constructor, . . .
- Add application specific prefix to all top-level identifiers.
  - Example : o.p is renamed to a1234_o.p
  - Separates effective namespace of an application from others.
  - Facebook provides libraries, accessible within the application namespace, to allow safe access to certain parts of the global object.

Case Study : *FBJS*

Rewriting (contd):

- o[p] is rewritten to o[idx(p)] : Controls access to dynamically generated property names.
  - idx(p) is a function defined by the host (Facebook) in the global object such that idx(p) = bad if p ∈ Blacklist else idx(p) = p.
  - Blacklist contains sensitive property names like __parent__, constructor, . . .
- Add application specific prefix to all top-level identifiers.
  - Example : o.p is renamed to a1234_o.p
  - Separates effective namespace of an application from others.
  - Facebook provides libraries, accessible within the application namespace, to allow safe access to certain parts of the global object.

An attack on FBJS (Nov'08)

Goal of the Attack

Get a handle to the global object in the application code.

Main Idea : Get a handle to the current scope object and shadow the ref method.

1. Getting the current scope: GET_SCOPE.

   ```
   try {throw (function(){return this;});}
   catch (f){ curr_scp = f();}
   ```

   Other tricks : Use named recursive functions (see our CSF'09 paper)

2. Shadow ref : `curr_scp.ref = function(x){return x;}`.
3. this will now evaluate to the global object !

Another attack on FBJS (Mar'09)

Goal of the attack

Access a black-listed property name

Main Idea

- The Facebook IDX(e) does the following check :
  1. Evaluate e2.
  2. Convert result(1) to string and check it is blacklisted
  3. If result(2) is false, return result(1) else return "bad".
- Observe e2 will get converted to string twice.

Almost works

e := {toString : function(){this.toString = function(){return 'constructor'} ;return 'foo'}}

FBJS has a check e instanceOf Object ? *"bad"*

Attack contd

In Safari, scope objects have a null prototype and hence they escape the instanceOf check.

Attack !!! (Safari)

```
var obj = GET_SCOPE;

obj.toString=function(){this.toString = function(){return 'constructor'}
;return 'foo'};

var f=function(){}; f[obj]('alert(0)')();
```

Vulnerabilities Disclosed

- To defend against the first attack, Facebook renamed idx and ref methods to $FBJS.idx and $FBJS.ref .
- To defend against the second attack, Facebook modified idx function to check the browser and decide if the object can escape the "instanceOf" check.
- Does this fix the problem once and for all ?
- Are more attacks possible on these lines ?

Summary of our analysis of FBJS

We realize the following three fundamental issues :

1. The ultimate goal is to ensure that a piece of untrusted code (that satisfies a certain syntactic criterion), does not access certain global variables.
2. There are a number of subtleties related to the expressiveness and complexity of JavaScript.
3. Finding temporary fixes to the currently known attacks is NOT sufficient.
4. Several million users : Impact value of a single attack is VERY high.

Formal Analysis !!

It is important to do a formal analysis based on traditional programming language foundations to design provable secure isolation techniques

1 Web 2.0 and the Isolation Problem

2 Case Study : FBJS
- Design
- Attacks and Challenges

3 Formal Semantics of JavaScript

4 Achieving the Isolation goal

5 Ongoing and Future Work

A bit about JavaScript

Key language features

- First class functions, Prototype based language, redefinable object properties.
- Can convert string to code :eval, Function
- Implicit type conversions

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

- ECMA262-3 : Standardized for browser compatibility. Does not include DOM and other browser extensions.
- Sufficient for 'understanding' the language but insufficient for rigorously proving properties about it.
- We need a formal semantics for representing the meaning of every possible JavaScript program.

Our Approach

For now, focus on ECMA-262-3rd edition. This is already quite non-trivial !

1. Convert Informal semantics(ECMA262-3) into a Formal semantics. (APLAS'08)
   - Specifies meaning in a Mathematically rigorous way.
   - The very act of formalization revealed subtle aspects of the language and helped us devise attacks on FBJS.
2. Systematically design subsets of JavaScript to achieve the isolation goal.
3. Use the formal semantics to rigorously prove that the isolation goal is attained for all programs within the subset (CSF'09, W2SP'09 and Ongoing) .

Structural 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

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.
- l : Location of the current scope object (or current activation record).
- t : Term being evaluated.

Semantic Rules

Small step style semantics (Gordon Plotkin)

- Three semantic functions $\xrightarrow{e}$, $\xrightarrow{s}$, $\xrightarrow{P}$ for expressions, statements and programs.
- Small step transitions : A semantic function transforms one state to another if certain conditions (premise) are true.
- General form : $\dfrac{\langle Premise \rangle}{S \xrightarrow{t} S'}$
- Atomic Transitions : Rules which do have another transition in their premise (Transition axioms).
- Context rules : Rules to apply atomic transitions in presence of certain specific contexts.
- Complete set of rules (in ASCII) span 70 pages.

1 Web 2.0 and the Isolation Problem

2 Case Study : FBJS
- Design
- Attacks and Challenges

3 Formal Semantics of JavaScript

4 Achieving the Isolation goal

5 Ongoing and Future Work

Back to Isolation Problem

Isolation Problem

Ensure that a piece of untrusted code written in a safe subset does not access certain security-critical global variables.

Let $Access(P)$ be the set of property names accessed when program P is executed.
Reduce the isolation problem to the following 2 sub problems.

Problem 1 (Isolation from library code)

Given a blacklist $\mathcal{B}$, design a meaningful sublanguage and an enforcement mechanism so that for all enforced programs P in the sublanguage, $Access(P) \cap \mathcal{B} \neq \emptyset$

Isolating host library methods : Create a blacklist $\mathcal{B}$ of all security critical methods in the library code .

Back to Isolation Problem

Isolation Problem

Ensure that a piece of untrusted code written in a safe subset does not access certain security-critical global variables.

Let $Access(P)$ be the set of property names accessed when program P is executed.

Reduce the isolation problem to the following 2 sub problems.

Problem 1 (Isolation from library code)

Given a blacklist $\mathcal{B}$, design a meaningful sublanguage and an enforcement mechanism so that for all enforced programs P in the sublanguage, $Access(P) \cap \mathcal{B} \neq \emptyset$

Isolating host library methods : Create a blacklist $\mathcal{B}$ of all security critical methods in the library code .

Isolation from other untrusted code ?

Key Idea : Rename identifiers to separate namespace of untrusted code.

But does this preserve the semantics ? Not for *Jt*.

- **Issue** : Variables are essentially properties of the current scope object (activation object).
  - var x = 42; this.x returns 42 in the global scope.
  - var a123_x = 42; this.x returns "reference error x not defined".
  - Disallow access to scope object !

Problem 2 (Isolating scope objects)

Define a meaningful sublanguage so that no program *P* in the sublanguage can return a pointer to a scope object.

Isolation from other untrusted code ?

Key Idea : Rename identifiers to separate namespace of untrusted code.

But does this preserve the semantics ? Not for *Jt*.

- **Issue** : Variables are essentially properties of the current scope object (activation object).
  - var x = 42; this.x returns 42 in the global scope.
  - var a123_x = 42; this.x returns "reference error x not defined".
  - Disallow access to scope object !

Problem 2 (Isolating scope objects)

Define a meaningful sublanguage so that no program *P* in the sublanguage can return a pointer to a scope object.

Plan

Isolating	Solution 1 (Static)	Solution 2 (Static + Runtime)
Blacklist (Problem 1)		
Scope (Problem 2)		

- Solution 1 is a sublanguage with pure static enforcement for achieving the goals in problem 1 and 2.
- Solution 2 is a sublanguage with static and runtime enforcement for achieving the goals in problem 1 and 2.

Isolating blacklist with syntactic enforcement

Design a sublanguage such that for any program P, all property names that can potentially be accessed appear textually in the code.

- **Fundamental issue** : Strings (m), Property Names (pn) and Identifiers (x) are implicitly converted to each other
- Terms whose reduction trace involves conversion from

  Strings ⟶ Property names (like e[e])

  Strings ⟶ Code (*like* eval)

  are evil. Get rid of them !

Subset *Jt*

Jt is defined as ECMA-262 MINUS: all terms containing the identifiers eval, Function, hasOwnProperty, propertyIsEnumerable, constructor and expressions e[e], e in e; the statement for (e in e) s;

Isolating blacklist with syntactic enforcement

Design a sublanguage such that for any program P, all property names that can potentially be accessed appear textually in the code.

- **Fundamental issue** : Strings (m), Property Names (pn) and Identifiers (x) are implicitly converted to each other
- Terms whose reduction trace involves conversion from

$$\text{Strings} \longrightarrow \text{Property names} \quad (\text{like } e[e])$$
$$\text{Strings} \longrightarrow \text{Code} \quad (\textit{like } \text{eval})$$

are evil. Get rid of them !

Subset *Jt*

Jt is defined as ECMA-262 MINUS: all terms containing the identifiers eval, Function, hasOwnProperty, propertyIsEnumerable, constructor and expressions e[e], e in e; the statement for (e in e) s;

Results

Isolating	Solution 1 (Static)	Solution 2 (Static + Runtime)
Blacklist	Subset *Jt* Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$	
Scope		

- Id(P) : Set of identifiers in P.
- Some property names are accessed implicitly (Recall type conversions). Denote these property names by $\mathcal{P}_{nat}$. Includes {toString, toNumber, valueOf }, Object, Array, RegExp}

Result

Any property name accessed by a program P in Jt when executed with respect to the initial heap is either contained in Id(P) or in $\mathcal{P}_{nat}$.

- Can also enforce whitelists !

Isolating scope object with syntactic enforcement

Isolating the scope object

- For initial empty heap state, global object is only accessible via @scope and @this properties
- Dereferencing @this is the only way of returning the current scope object.
- Object.prototype.valueOf, Array.prototype.sort/concat/reverse can potentially deference the @this property.

Subset *Js*

The subset *Js* is defined as *Jt*, MINUS: all terms containing the expression this; all terms containing the identifiers valueOf, sort, concat and reverse;

$Js \subset Jt$: Sufficient for imposing the restriction that properties valueOf, sort, concat and reverse are never accessed

Isolating scope object with syntactic enforcement

Isolating the scope object

- For initial empty heap state, global object is only accessible via @scope and @this properties
- Dereferencing @this is the only way of returning the current scope object.
- Object.prototype.valueOf, Array.prototype.sort/concat/reverse can potentially deference the @this property.

Subset *Js*

The subset *Js* is defined as *Jt*, MINUS: all terms containing the expression this; all terms containing the identifiers valueOf, sort, concat and reverse;

$Js \subset Jt$: Sufficient for imposing the restriction that properties valueOf, sort, concat and reverse are never accessed

Results

Isolating	Solution 1 (Static)	Solution 2 (Static + Runtime)
Blacklist (Problem 1)	Subset *Jt* Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$	
Scope (Problem 2)	Subset *Js* $\subseteq$ *Jt* Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$	

Result

No program in the language *Js* when executed with respect to the initial heap evaluates to the address of a scope object.

Isolating blacklist with runtime enforcement

Jt is fairly restrictive.

- Disallows [] operator altogether $\Rightarrow$ No array access
- In principle, solution to problem 1 should allow o[p] where $p \notin \mathcal{B}$.

Runtime Check : e1[e2] $\longrightarrow$ e1[IDX(e2)] (along the lines of FBJS)
How do we design for IDX which enforces property that

- No property name from blacklist $\mathcal{B}$ ever gets accessed.
- Semantics is preserved for all programs P for which $Access(P) \cap \mathcal{B} \neq \emptyset$.

Subset Jt^{run}

e1[e2] ⟶ va1[e2] ⟶ va1[va2] ⟶ o[va2] ⟶ o[m]

- Observe that first e1 and e2 are converted to a value and only then e2 is converted to a string.
- Ideally, IDX(e2) should return a value which on being converted to a string, checks if the string obtained from e2 is outside the blacklist and returns it.

IDX

($=e2,{toString:function(){return ($=TOSTRING($),FILTER($))}}) where TOSTRING($) = (new $String($)).valueOf() FILTER($) = ($blacklist[$]?"*bad*":$)

Subset Jt^{run}

The subset Jt^{run} is defined as as Jt plus e[e] minus all terms with identifiers beginning with $

Subset Jt^{run}

e1[e2] ⟶ va1[e2] ⟶ va1[va2] ⟶ o[va2] ⟶ o[m]

- Observe that first e1 and e2 are converted to a value and only then e2 is converted to a string.
- Ideally, IDX(e2) should return a value which on being converted to a string, checks if the string obtained from e2 is outside the blacklist and returns it.

IDX

($=e2,{toString:function(){return ($=TOSTRING($),FILTER($))}})
where TOSTRING($) = (new $String($)).valueOf() FILTER($) = ($blacklist[$]?"*bad*":$)

Subset Jt^{run}

The subset $Jt^{run}$is defined as as Jtplus e[e] minus all terms with identifiers beginning with $

Subset Jt^{run}

Isolating	Solution 1 (Static)	Solution 2 (Static + Runtime)
Blacklist (Problem 1)	Subset Jt Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$	Subset Jt^{run} Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$ e1[e2] $\rightarrow$ e1[IDX(e2)]
Scope (Problem 2)	Subset Js Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$	

Result

For all programs P in Jt^{run} such that Id(P) $\cap \mathcal{B} \neq \emptyset$, the program $String=String; Rew(P) when executed with respect to the initial heap does not access any property from $\mathcal{B}$.

Isolating global object with runtime enforcement

Js disallows this

- Heavily used in object oriented programming.
- In principle, solution to problem 2 must allow this if it does not point to a scope object.

Runtime check : this $\longrightarrow$ NOSCOPE(this)

- How can we check if a given object is a scope object ?
- Not straightforward in general,
  Use NOGLOBAL(this) = (this==$?null;this).
- NOSCOPE(this) is definable for Firefox, see paper.

Subset Js^{run}

Define the subset Js^{run} as *Js* plus : all terms containing this minus all terms with identifiers beginning with $

Isolating global object with runtime enforcement

Js disallows this

- Heavily used in object oriented programming.
- In principle, solution to problem 2 must allow this if it does not point to a scope object.

Runtime check : this ⟶ NOSCOPE(this)

- How can we check if a given object is a scope object ?
- Not straightforward in general,
  Use NOGLOBAL(this) = (this==$?null;this).
- NOSCOPE(this) is definable for Firefox, see paper.

Subset Js^{run}

Define the subset Js^{run} as *Js* plus : all terms containing this minus all terms with identifiers beginning with $

Results

Isolating	Solution 1 (Static)	Solution 2 (Static + Runtime)
Blacklist (Problem 1)	Subset Jt Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$	Subset Jt^{run} Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$ e1[e2] $\rightarrow$ e1[IDX(e2)]
Global Object (Problem 2 weak)	Subset Js Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$	Subset Js^{run} Filter P if Id(P) $\cap \mathcal{B} \neq \emptyset$ e1[e2] $\rightarrow$ e1[IDX(e2)] this $\rightarrow$ NOGLOBAL(this)

Result

For all programs P in Js^{run} such that Id(P) $\cap \mathcal{B} \neq \emptyset$, the program \$=window; Rew(P) when executed with respect to the initial heap, never evaluates to the global object and does not access any blacklisted property.

Results

Isolating	Solution 1 (Static)	Solution 2 (Static + Runtime)
Blacklist (Problem 1)	Subset *Jt* Filter P if $Id(P) \cap \mathcal{B} \neq \emptyset$	Subset Jt^{run} Filter P if $Id(P) \cap \mathcal{B} \neq \emptyset$ e1[e2] → e1[IDX(e2)]
Global Object (Problem 2 weak)	Subset *Js* Filter P if $Id(P) \cap \mathcal{B} \neq \emptyset$	Subset Js^{run} Filter P if $Id(P) \cap \mathcal{B} \neq \emptyset$ e1[e2] → e1[IDX(e2)] this → NOGLOBAL(this)

Result

For all programs P in Js^{run} such that $Id(P) \cap \mathcal{B} \neq \emptyset$, the program $=window; Rew(P) when executed with respect to the initial heap, never evaluates to the global object and does not access any blacklisted property.

Solution for FBJS

- Define $FBJS.ref and $FBJS.IDX in a different name-space.
- Use the version of IDX proposed by us.
  - Preserves semantics.
  - Prevents access to blacklisted properties
- Given a library blacklist $\mathcal{B}$, use subset Js^{run}.
- Appropriately rename all identifiers
- Finally, parse the text of the code to disallow identifier names beginning with "$" or any blacklisted identifiers.

Ongoing and Future Work

- Design suitable run-time checks for eval, Function.
- Given a set of sensitive property names, design a procedure to analyze the library code and automatically generate the minimal blacklist which will guarantee property isolation.
- Write the semantics in machine readable format so that the proofs can be automated.
- Extend the above results to apply to JavaScript supported by various browsers which include features beyond the ECMA-262 spec, such as getter, setters, __proto__ etc.

Thank You !