Automated Encapsulation Analysis of Security-Critical APIs

Ankur Taly
Stanford University

Joint work with
John C. Mitchell, Ulfar Erlingsson, Mark S. Miller and Jasvir Nagra
Web 2.0 – webpages with third-party code

- Lots of client-side JavaScript, AJAX
- High Impact: Millions of users, loads of e-commerce, $$$. 

Security Issues: JavaScript is too powerful

<script src="http://adpublisher.com/ad1.js"></script>

Unrestricted DOM manipulation
- `top.location = http://evil.com`
- `var c = document.getElementsByName("cookie")`
- `var p = document.getElementsByName("password")[0]`

Must sandbox untrusted code to reduce its privileges!

Manipulates features of the page via the JavaScript DOM API

Escape same-origin policy

5/5/2011
Browser based Sandbox: IFrames

- Like Process Isolation: Each frame has a separate execution environment.
- Too restrictive:
  - cannot communicate objects across frame boundaries
  - Iframes cannot move around.
- No control over (malicious) code in frames.

\[ \text{Not the mechanism studied in this talk} \]

Language based Sandbox (This work)

A.com
Protected resources
Dom Wrapper API
B.com – sandboxed JS code

JS Filter and Rewriter (sandboxing mechanism)

Facebook FBJS, Yahoo! ADSafe, Google Caja
Approach: Security API + Sandbox

**Protected Resources:** Typically elements of the DOM.

**Security API:** Provide mediated access to critical resources thereby preserving a security policy.

**Sandbox:** Restrict untrusted code providing it access ONLY to the API.

**Security Goal:** Untrusted code must NOT obtain direct access to any security critical resource.

Motivated by Object-Capability based Security.
Simple Example: Write-only Log

- Hosting page wants to provide a *write only* Log service to untrusted code.

```javascript
var privLog = [<critical data>]

var api = [push: function(x){
    privLog.push(x)
}]
```

- Sandbox ensures that untrusted code can ONLY access the `global variable “api”`. Also all built-ins are non-writable.
Things can go wrong very easily

- Hosting page wants to provide a *write only* Log service to untrusted code.

  ```javascript
  var privLog = [<critical data>]
  var api = [
    push: function(){
      privLog.push(x)
    },
    store: function(i,x){
      privLog.store(i,x)
    }
  ]
  ```

**Need Automated Analysis of APIS:** check for ALL possible method invocations

Is the API still safe?  **No !!**

  ```javascript
  var steal;
  api.store("push",function(){
    steal = this[0]};
  api.push();
  ```
Quick Case study: Yahoo Adsafe

Mechanism for safely embedding untrusted advertisements.

- **ADSAFE object (API)**: Provides mediated access to the DOM.
  
  ```javascript
  ADSAFE.dom.fragment = function () {
    reject_global(this);
    var frag = document.createFragment();
    return new Bunch([frag]);
  }
  ```

- **JSLint (Sandbox)**: Static filter for JS.
  - Disallows `eval`, `this`, `with`, `o[p]`.
  - Allows untrusted code to only have a single global variable “ADSAFE”.

- **Security Goal**: No direct access to any DOM element.

**This Work**: Found a new vulnerability in the ADSAFE API.
Two Problems

**Sandbox Security Problem:** Design a sandbox for untrusted code that ensures that all external mutable access is derived from the API only.

**API Confinement Problem:** Verify that no sandboxed untrusted program can use the API to obtain direct access to a critical resource.
State of JavaScript

• ES3: JS based on ECMA-262 3rd edition spec.
• ES5: JS based on ECMA-262 5th edition spec (released in Dec 2009).
• ES5S: strict mode of the ES5 language.
• SESlight: Security ECMAScript, a subset of ES5S.
  – This Work!
  – SES is currently under proposal by the ECMA committee.
Enemies of Static Analysis in ES3

• No static scoping.

```javascript
Object.prototype.x = 42
var x = 24;
var f = function foo(){return x;};
f();

What is the value returned by f()? 42
```

• No Closure-based Encapsulation

```javascript
function trusted(untrusted, secret){
    if (untrusted() === secret){ <do something>}
}

Can untrusted steal secret? Yes
```

```javascript
untrusted = function(){steal = untrusted.caller.arguments[1]}
```

To our rescue: SES\textsubscript{light} subset of ES\textsubscript{5S}.
• Describe the language $\text{SES}_{\text{light}}$
• Describe a simple sandbox for $\text{SES}_{\text{light}}$ code.
• Describe a procedure for solving the API security problem for $\text{SES}_{\text{light}}$.

Recall: $\text{SES}_{\text{light}}$ is a subset of ES5S.
From ES3 to ES5S

<table>
<thead>
<tr>
<th>Restriction</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>No <code>delete</code> on variable names</td>
<td>Achieving Static Scoping</td>
</tr>
<tr>
<td>No prototypes for scope objects</td>
<td>Achieving Static Scoping</td>
</tr>
<tr>
<td>No <code>with</code></td>
<td>Achieving Static Scoping</td>
</tr>
<tr>
<td>No <code>this</code> coercion</td>
<td>Plugging Encapsulation Leaks</td>
</tr>
<tr>
<td>No <code>.callee</code>, <code>.caller</code> on argument objs</td>
<td>Plugging Encapsulation Leaks</td>
</tr>
<tr>
<td>No <code>.callee</code>, <code>.arguments</code> on function objs</td>
<td>Plugging Encapsulation Leaks</td>
</tr>
<tr>
<td>No arguments and formal parameters aliasing</td>
<td>Plugging Encapsulation Leaks</td>
</tr>
</tbody>
</table>

**Property:** ES5S is a lexically scoped language
ES5S to SES$_{\text{light}}$ (Secure ECMAScript)

**Sandbox Goal:** *Sandboxed code must derive all it external mutable access from the API.*

**Remaining Hurdles:**

- Default Mutable Access to all built-in objects.
  
  ```javascript
  Object.prototype.toString = '< evil function >
  ```
  
  (OR **Lack of Loader Isolation**)

- Dynamic code execution using eval.
ES5S to SES$_{\text{light}}$ (Secure ECMAScript)

- **Immutable Built-ins**: No built-in object can be mutated or extended.

- **Annotated Eval**:
  - Syntax: `eval(t, x_1,...,x_n)
  - Semantic Restriction:
  - Example: `eval("var x = y + z","y")` throws TypeError.
  - Example: `eval("var x = y + z",y)` terminates normally but `eval("var x = y + z","y")` throws TypeError.

- **No Setters/Getters**: Removed purely for ease of analysis (will be there in next paper !)
Simple Sandbox for SES\textsubscript{light}

- Trusted page code creates an API object and stores it in the variable “api”.
- Sandbox:
  - Only accessible trusted code global variable must be “api”.
  - Also provide a variable “test” as a challenge variable.
  - Untrusted code wins if it sets “test” to a critical object.

**Sandbox**: Rewrite untrusted code \( s \) to \( \text{eval}(s,"\text{api"},"\text{test"}) \)

- Much \textit{simpler} than JSLint, FBJS, Caja
- Thanks to the cleaner ES5S semantics!
Formulating the API Confinement Problem

Only involves analysis of trusted code, assuming worst case for untrusted code.
Solving the API Security Problem
Technique

• Points-to Analysis
  – Context Insensitive: Single Activation Record per function.
  – Flow Insensitive: Insensitive to the order of statements.

• Only track references, ignore primitive values.

• Expressed in Datalog (Whaley et al)
  – Collect Facts about the program: Assign(x,y), Load(x,y,z),...
  – Encode semantics of the program as Horn clauses:
    Stack(x,l) :- Assign(x,y), Stack(y,l)
  – Generate all possible consequence facts under the rules.
  – Tricky Case: eval, capturing implicit code execution.
    – Scalable and well studied technique for C, Java.
Simple Example

Program $t$:

```plaintext
var y = {}; 
var z = {}; 
var x = y; 
x.f = z; 
x = y.f;
```

Facts:

- Stack($y, l_y$)
- Stack($z, l_z$)
- Assign($x, y$)
- Store($x, "f", z$)
- Load($x, y, "f"$)

Consequence Facts:

- Stack($y, l_y$), Stack($z, l_z$), Stack($x, l_y$), Stack($x, l_z$), Heap($l_y, "f", l_z$)

$\text{Pointsto}(x, t) = \{l_y, l_z\}$
## Datalog Predicates

<table>
<thead>
<tr>
<th>Predicates for encoding the term</th>
<th>Predicates for encoding the heap-stack</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Assign(x, y)</code></td>
<td><code>Heap(l, x, m)</code></td>
</tr>
<tr>
<td><code>Load(x, y, z)</code></td>
<td><code>Stack(x, l)</code></td>
</tr>
<tr>
<td><code>Catch(l, x)</code></td>
<td><code>Prototype(l, m)</code></td>
</tr>
<tr>
<td><code>Store(x, y, z)</code></td>
<td><code>FuncType(l)</code></td>
</tr>
<tr>
<td><code>TP(l, x)</code></td>
<td><code>ObjType(l)</code></td>
</tr>
<tr>
<td><code>FormalArg(l, i, x)</code></td>
<td><code>ArrayType(l)</code></td>
</tr>
<tr>
<td><code>Actual(x, i, z, y, l)</code></td>
<td><code>NotBuiltin(l)</code></td>
</tr>
</tbody>
</table>

- $x, y, z$ are variable/property names
- $l, m, n$ are object creation site labels (abstract locations)
Encoding Eval

Encoding $\texttt{eval}(s,x_1,\ldots,x_n)$

- Assume \textit{worst case} that all possible points-to relationships are created between all free and bound variables.
- Summarize all bound variables by a single variable $x_{\text{eval}}$.
- Summarize all locally created object by a single label $l_{\text{eval}}$

Let $X := \{x_1, \ldots, x_n\} \cup \{x_{\text{eval}}\}$ and $L := \{l_{\text{eval}}, l_{\text{outer}}\}$

\[
\begin{align*}
\{\text{Assign}(x_1, x_2) \mid x_1, x_2 \in X\} & U \{\text{Load}(x_1, x_2, "\text{\$All}\") \mid x_1, x_2 \in X\} U \\
\{\text{Store}(x_1, "\text{\$All}\", x_2) \mid x_1, x_2 \in X\} & U \\
\{\text{NotBuiltin}(l_{\text{eval}})\} & U \\
\{\text{FuncType}(l_{\text{eval}})\} & U \\
\{\text{Actual}(x_1, i, x_2, x_3, l) \mid x_1, x_2, x_3 \in X; l \in L\} & U \\
\{\text{FormalArg}(l_{\text{eval}}, i, x) \mid x \in X\} & U \\
\{\text{Instance}(l_{\text{eval}}, x) \mid x \in X\} & U \\
\{\text{Catch}(l, v) \mid x \in X; l \in L\} & U \\
\{\text{Throw}(l, v) \mid x \in X; l \in L\} & U
\end{align*}
\]
Inference Rules

Assignment, Load, Store

\[ \text{Stack}(x, l) \quad ::= \quad \text{Stack}(y, l), \text{Assign}(x, y) \]
\[ \text{Stack}(x, n) \quad ::= \quad \text{Load}(x, y, f), \text{Stack}(y, l), \text{Prototype}(l, m), \text{Heap}(m, f, n) \]
\[ \text{Heap}(l, f, m) \quad ::= \quad \text{Store}(x, f, y), \text{Stack}(x, l), \text{Stack}(y, m), \text{NotBuiltin}(l) \]
\[ \text{Prototype}(l, n) \quad ::= \quad \text{Prototype}(l, m), \text{Prototype}(m, n) \]

Function Calls

\[ \text{Assign}(z, x) \quad ::= \quad \text{Actual}(f, i, x, y, k), \text{Stack}(f, l), \text{FormalArg}(l, i, z) \]
\[ \text{Assign}(y, z) \quad ::= \quad \text{Actual}(f, i, x, y, k), \text{Stack}(f, l), \text{FormalRet}(l, z) \]

ToPrimitive Conversions

\[ \text{Actual}(n, 0, x, 'd', l) \quad ::= \quad \text{TP}(x, l), \text{Stack}(x, l), \text{Prototype}(l, m), \text{Heap}(m, "toString", n) \]
\[ \text{FuncType}(n) \]
\[ \text{Actual}(n, 0, x, 'd', l) \quad ::= \quad \text{TP}(x, l), \text{Stack}(x, l), \text{Prototype}(l, m), \text{Heap}(m, "valueOf", n) \]
\[ \text{FuncType}(n) \]
Procedure

Procedure D(t, P):

1. Pick any \( s \) in \( \text{SES}_{\text{light}} \) and set \( t_{\text{new}} \leftarrow t \); eval(s, “api”, “test”)
2. \( D_1 \leftarrow \text{Encode}(t_{\text{new}}) + \text{Encoding of built-in objects.} \)
3. \( D_2 \leftarrow \text{All possible consequences of } D_1 \)
4. Return CONFINED iff there is NO \( l \in P \) such that \( \text{Stack(“test”,l)} \in D_2 \)

Theorem: Procedure \( D(t,P) \) is SOUND
If the procedure says CONFINED then the API is actually safe.

Paper at Oakland’ 11
Applications

ENCAP: http://code.google.com/p/es-lab/trunk/src/util/EncapsulationAnalysis

1. Yahoo! ADSafe DOM API – This talk
2. Mark Miller’s Mint Library – see paper
3. Sealers and Unsealers Library – see paper
Yahoo! ADSafe

- ADSafe API does not use setters/getters
- can be de-sugared to SES\textsubscript{light}
- Added Annotations on Property lookups:
  - $\texttt{NotReject}$: Added to patterns of the form
    \[
    \text{if}(!\texttt{reject}(\texttt{name})){ \ ...\texttt{object[name, NotReject]}\ ...
    }\]
- Approx 2000 LOC.

**On Running ENCAP:** We obtained the result NOT CONFINED if the methods ADSAFE.lib and ADSAFE.go were exposed to untrusted code.
This turned out to be a real Attack!

```html
<div id="test">
  <script>
"use strict";
ADSAFE.lib("___nodes___",
  function(lib){
    var o = [[appendChild, function()]{
      var f = frag.value;
      // f now points to the value method of the dom library
      lib.v = f;
      lib.v();
    }]
  };
</script>
</div>
```

- ADSafe API protects DOM objects by hiding them behind "___nodes___" property of certain fake object (Bunch objects).

- **Assumption**: Sandboxed code can never write to "___nodes___" property of any object.

Pass `adsafe_lib` as Bunch object to DOM wrapper.
Fixing the Attack

- Replace ADSafe.lib with the following

```javascript
ADSafe.lib = function(name, f){
  if(!reject_name(name)){
    adsafe_lib[name] =
    f(adsafe_lib)}
}
```

On running ENCAP: We obtain that the API is CONFINED.

**ADSafe API is confined under the provided annotations and the SESlight threat model.**

Currently adopted by ADSafe.
Summary

• $\text{SES}_{\text{light}}$ is more amenable to static analysis than ES3

• Confinement analysis only involves analyzing trusted code.

• Precision can be improved by further restricting (trusted) developers to a more static analysis-friendly subset.
Future Work

• Setters/Getters
• Improving Context-sensitivity of the analysis
  • Explore CFA2, Object-sensitivity.
• Provenance tracking in Datalog.
• Multiple untrusted components instead of one.
  — Collusion between components can be a problem.
Operational Semantics of SES\textsubscript{light}

- Small step style Operational Semantics.
  - Describes intermediate execution states.
  - Based on ECMA262 5\textsuperscript{th} edition spec.
- Key differences with ES3 semantics.
  - Activation record modeled as store data-structure instead of scope objects.
  - Simpler and more standard rules for most constructs.
  - 20 pages instead of 70!

Properties:
- Free variables can be statically defined.
- A term derives all its external mutable access from its free variables.
Previous Work (with S. Maffeis, J. Mitchell)

• Operational Semantics for ES3 (APLAS’08)
• Secure subsets of ES3 (CSF’00, W2SP’09, ESORICS’09)
  • Subset $J_{safe}(B)$ for a blacklist $B$ of property names.
  • An enforcement function $enf: J_{safe}(B) \rightarrow J_{safe}(B)$ based on filtering, rewriting and wrapping of code.

**Theorem:** For all programs $t$ in $J_{safe}(B)$, $enf(t)$ never accesses a black-listed property names and never obtains direct access to the global object.

• Sandbox Security Problem can be solved for ES3.
• API Security Problem still remains unsolved for ES3
  • static analysis of ES3 is challenging!