Web Security and Untrusted JavaScript

John Mitchell
Stanford University

Stanford Computer Security Lab

Outline

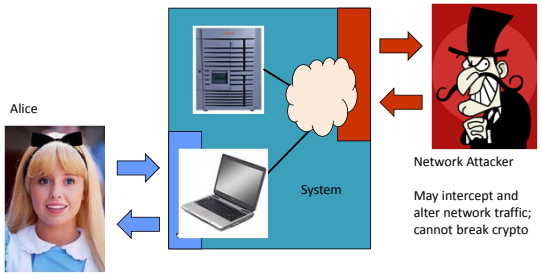
- Background: computer security
- Web security
- JavaScript isolation
  - How can trusted and untrusted code be executed in the same environment, without compromising functionality or security?
- Three parts
  - Isolate untrusted application from hosting page
  - Isolate one untrusted application from another
  - Mediated access: reference monitor for critical resources
- Looking ahead
  - Improvements in JavaScript standards, tools
  - Foundations for Web security

Computer Security

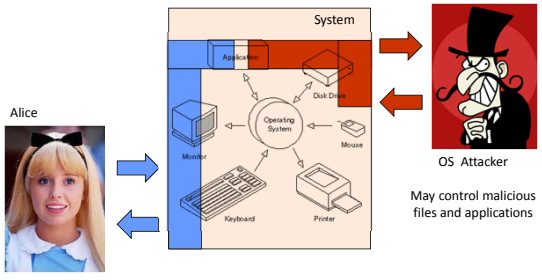
Computer Security

- Security model
  - A system of interest
  - Desired properties of the system
  - Interface and capabilities of an attacker
- Security analysis
  - Can system design and security mechanism it includes guarantee desired the properties, in spite of attacker?

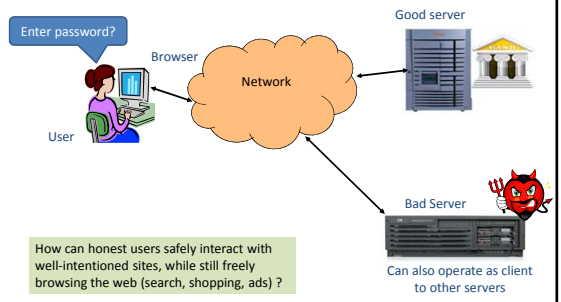
Network security



Operating system security

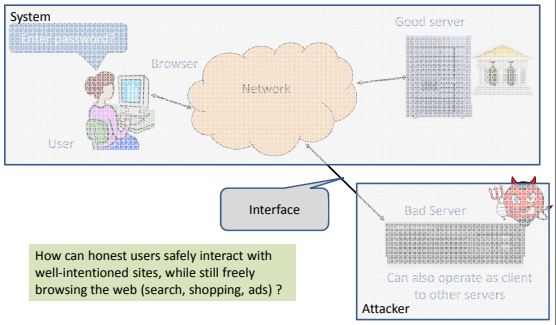


Web Security



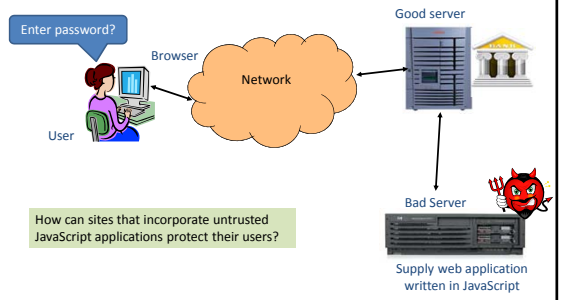
How can honest users safely interact with well-intentioned sites, while still freely browsing the web (search, shopping, ads) ?

Web Security



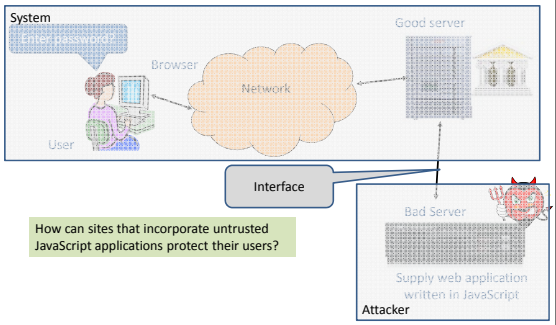
How can honest users safely interact with well-intentioned sites, while still freely browsing the web (search, shopping, ads) ?

Specific problem for this talk



How can sites that incorporate untrusted JavaScript applications protect their users?

Specific problem for this talk



How can sites that incorporate untrusted JavaScript applications protect their users?

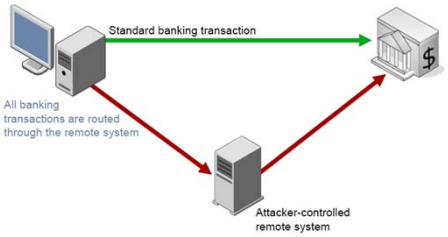
Web Security Challenges

Online Identity Theft

- Password phishing
  - Forged email and fake web sites steal passwords
- Password theft
  - Criminals break into servers and steal password files
- Spyware
  - Keyloggers steal passwords, product activation codes, etc.
- Botnets
  - Networks of compromised end-user machines spread SPAM, launch attacks, collect and share stolen information
- Magnitude
  - $$$ billions in direct loss per year
  - Significant indirect loss
    - Loss of confidence in online transactions
    - Inconvenience of restoring credit rating, identity

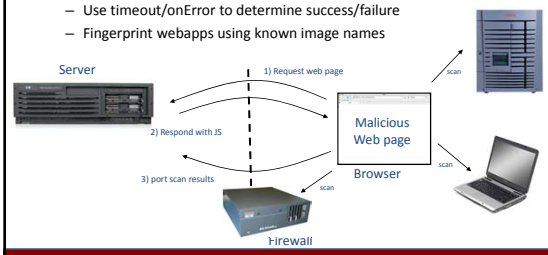
Current trend

- Why ask the user to do something if you can write JavaScript to do it automatically?

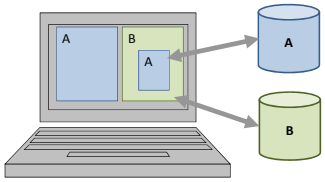


Port scanning behind firewall

- JavaScript can:
  - Request images from internal IP addresses
    - Example:
  - Use timeout/onError to determine success/failure
  - Fingerprint webapps using known image names

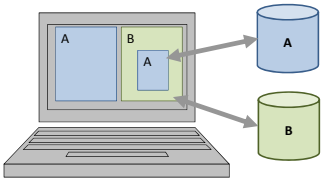


Browser security mechanism



- Each frame of a page has an origin
  - Origin = protocol://host:port
- Frame can access its own origin
  - Network access, Read/write DOM, Storage (cookies)
- Frame cannot access data associated with a different origin

Browser security mechanism



CSRF Attack: Code can make a request to any origin, transmitted as if typed in by user

- Each frame of a page has an origin
  - Origin = protocol://host:port
- Frame can access its own origin
  - Network access, Read/write DOM, Storage (cookies)
- Frame cannot access data associated with a different origin

Library import

```
<script src=https://seal.verisign.com/getseal?
                        host_name=a.com></script>
```

- Embedded script has privileges of frame, NOT source server
- Can script other pages in this origin, load more scripts
- Other forms of importing

Analogy

Operating system

- Primitives
  - System calls
  - Processes
  - Disk
- Principals: Users
  - Discretionary access control
- Vulnerabilities
  - Buffer overflow
  - Root exploit

Web browser

- Primitives
  - Document object model
  - Frames
  - Cookies / localStorage
- Principals: "Origins"
  - Mandatory access control
- Vulnerabilities
  - Cross-site scripting
  - Cross-site request forgery
  - Cache history attacks
  - …

Frame and iFrame

- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken
- Why not use frames?
  - Limit interaction between different sections of code
  - Performance cost
- Analogy
  - We use both OS isolation and PL isolation in systems
  - We use both Frame isolation and PL isolation on Web

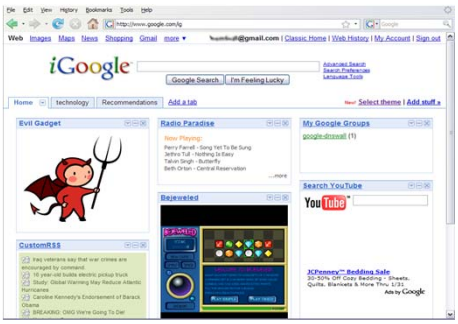
Advertisements

Advertisements

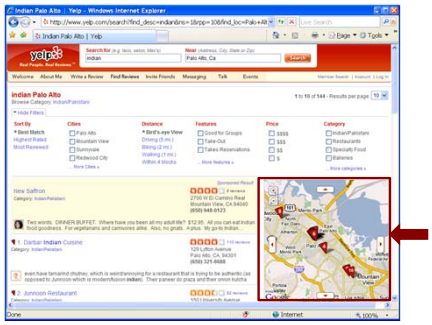
- Ad network, publisher have incentives to show ads
  - Could place ads in iframe
  - Rules out more profitable floating ads, etc.
- Ad network and publisher can try to screen ads
  - Example: Yahoo! AdSafe
- Some limitations in current web
  - Ads may contain links to "images" that are part of ad

- Important to remember
  - This is a very effective way to reach victims: $30-50 per 1000
  - User does not have to click on anything to run malicious code

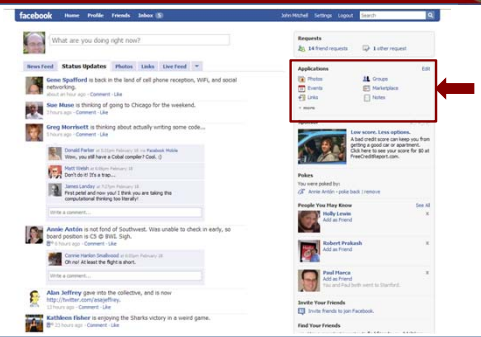
Mashups



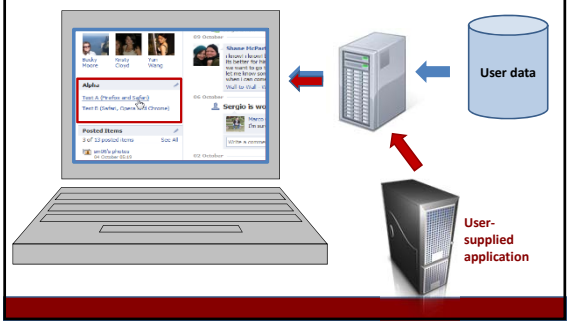
Maps



Social Networking Sites

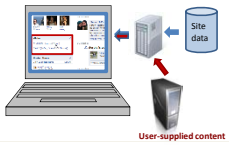


Third-party content



Secure Web Mashups

- Challenge
  - How can trusted and untrusted code be executed in the same environment, without compromising functionality or security?



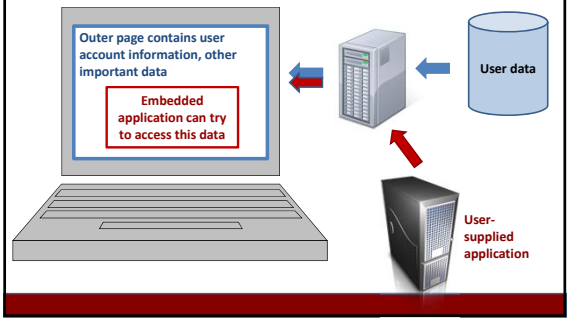
- Approach
  - Programming language semantics
    - Mathematical model of program execution
  - Secure sublanguages and security tools
    - Filtering, Rewriting, Wrapping
    - Object-capability model
    - Improved JavaScript standards
    - Automated code analysis tool(s)
- Test cases and paradigms
  - Facebook JavaScript (FBJS)
    - Allow user-supplied applications
  - Yahoo! ADSafe
    - Screen ad content before publisher
  - Google Caja
    - Mathematical foundations of object-capability languages
    - Isolation, defensive consistency, …

Hosting page isolation

Protect hosting page from untrusted applications

(executed in the same browser frame)

Hosting page isolation



Facebook FBJS

- Facebook applications either "iframed" or integrated on page
  - We are interested in integrated applications
- Integrated 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 Document Object Model
- Security goals
  - No direct access to the Document Object Model (DOM)
  - No tampering with the execution environment
  - No tampering with Facebook libraries
- Basic approach
  - FBJS restricts "tricky" parts of JavaScript
  - Blacklist variable names that are used by containing page
  - Prevent access to global scope object, because variables are properties of scope objects

Four "FBJS" Theorems

- **Theorem 1:** Subset J(B) of ES-3 prevents access to chosen blacklist B (assuming $B \cap P_{nat} = \varnothing$)
- **Theorem 2:** Subset $J(B)_G$ of J(B) prevents any expression from naming the global scope object
- **Theorem 3:** Subset $J(B)_S$ of $J(B)_G$ of prevents any expression from naming any scope object
- **Theorem 4:** A specific "wrapping" technique preserves Theorem 3 and allows previously blacklisted functions to be safely used

JavaScript Challenges

- Mutable objects with implicit self parameter:
  - `o={b:function(){return this.a}}`
- Prototype-based object inheritance:
  - `Object.prototype.a="foo";`
- Scope can be a first-class object:
  - `this.o === o;`
- Can convert strings into code:
  - `eval("o + o.b()");`
- Implicit type conversions, which can be redefined.
  - `Object.prototype.toString = o.b;`

JavaScript can be tricky

- Which declaration of g is used?

```
var f = function(){ var a = g();
                    function g() { return 1;};
                    function g() { return 2;};
                    var g = function() { return 3;}
                    return a;}
var result = f();
                    // has as value 2
```

- Implicit conversions

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

- Use of *this* inside functions

```
var b = 10;
var f = function(){ var b = 5;
                    function g(){var b = 8; return this.b;};
                    g();}
var result = f();
                    // has as value 10
```

- String computation of property names

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

for (p in o){....}, eval(...), o[s]
allow strings to be used as code and vice versa

JavaScript modularity

- Modularity: variable naming and scope
- JavaScript local variables are not "local"
  - Activation records are objects
  - A program can get access to these objects
    - Properties (local variables) can be added, removed
  - These objects have prototypes
    - Properties (local variables) can be added, removed
- Traditional JavaScript (ECMA 2.6.2-3) does not support modularity with information hiding

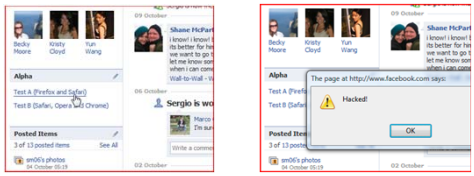
Operational Semantics [APLAS'08]

- 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 : $\frac{\langle Premise \rangle}{S \xrightarrow{t} S'}$
- Atomic Transitions : Rules which do have another transition in their premise
- Context rules : Rules to apply atomic transitions in presence of certain specific contexts.

Basis for JavaScript Isolation

1. All explicit property access has form `x, e.x`, or `e1[e2]`
2. The implicitly accessed property names are: `0,1,2,…, toString, toNumber, valueOf, length, prototype, constructor, message, arguments, Object, Array, RegExpg`
3. Dynamic code generation (converting strings to programs) occurs only through `eval, Function,` and indirectly `constructor`
4. A pointer to the global object can only be obtained by: `this`, native method `valueOf` of Object.prototype, and native methods `concat, sort` and `reverse` of Array.prototype
5. Pointers to local scope objects through `with, try/catch`, "named" recursive functions `var f = function g(..){… g(..)…`

Sample Facebook vulnerability



- FBJS `e1[IDX(e2)]` did not correctly convert objects to strings
- Exploit: we built an FBJS application able to reach the DOM.
- Disclosure: we notified Facebook; they promptly patched FBJS.
- Potential for damage is considerable.
  - Steal cookies or authentication credentials
  - Impersonate user: deface or alter profile, query personal information, spam friends, spread virally.

The run time monitor IDX

- We need some auxiliary variables: we prefix them with $ and include them in our blacklist B

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

- Rewrite `e1[e2]` to `e1[IDX(e2)]`, where

```
IDX(e) =
($=e,{toString:function(){
                  return($=$String($),
                  $B[$]?"bad":$)
                  }})
```

  - Blacklisting can be turned into whitelisting by inverting the check above (`$B[$]?$:"bad"`).
- Our rewriting faithfully emulates the semantics

```
e1[e2] -> va1[e2] -> va1[va2] -> l[va2] -> l[m]
```

Improving our solutions by wrapping

- No need to blacklist `sort, concat, reverse, valueOf`.
  - We can wrap them as follows

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

  - This variant is provably correct.
- Wrapping `eval` and `Function` : possible in principle

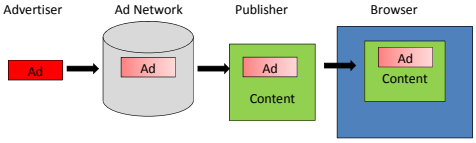
Four "FBJS" Theorems [CSF'09… ESORICS'09]

- **Theorem 1:** Subset J(B) of ES-3 prevents access to chosen blacklist B (assuming $B \cap P_{nat} = \emptyset$)
- **Theorem 2:** Subset $J(B)_G$ of J(B) prevents any expression from naming the global scope object
- **Theorem 3:** Subset $J(B)_S$ of $J(B)_G$ of prevents any expression from naming any scope object
- **Theorem 4:** A specific "wrapping" technique preserves Theorem 3 and allows previously blacklisted functions to be safely used

We can prove isolation for a language very similar to FBJS. Success!! ?

Yahoo! AdSafe

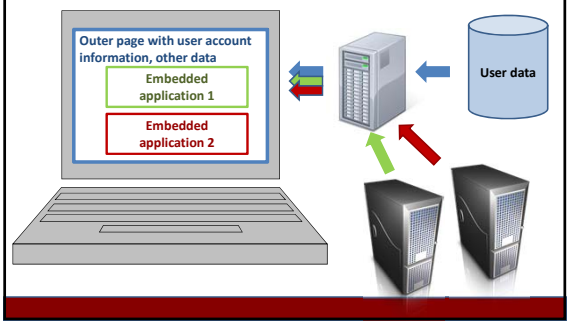
- Goal: Restrict access to DOM, global object



- This is a *harder* problem than SNS applications
  - Advertising network must screen advertisements
  - Publishing site is not under control of ad network

Isolation *Between* Untrusted Applications

Isolation between applications



FBJS limitations

- Authority leak
  - Can write/read properties of native objects
    - var Obj = {};
    - var ObjProtToString = Obj.toString;
- Communication between untrusted apps
  - First application
    - Obj.toString.channel = ”message”;
  - Second application
    - var receive_message = Obj.toString.channel;

Defeat Sandbox

```
<a href="#" onclick="break()">Attack FBJS!</a> <script>
function break(){
   var f = function(){};
   f.bind.apply =
       (function(old){return function(x,y){
              var getWindow = y[1].setReplay;
              getWindow(0).alert("Hacked!");
              return old(x,y)}
       })(f.bind.apply)
}</script>
```

- Redefine bind method used to Curry functions
- Interferes with code that uses f.bind.apply(e)

How to isolate applications?

- Capability-based protection
  - Traditional idea in operating systems
  - Capability is “ticket” granting access
  - Process can only access through capabilities given
- If we had a capability-safe subset of JavaScript:
  - Give independent apps disjoint capabilities
- Problem: Is there a capability-safe JavaScript?

Foundations for object-capabilities

[S&P 2010]

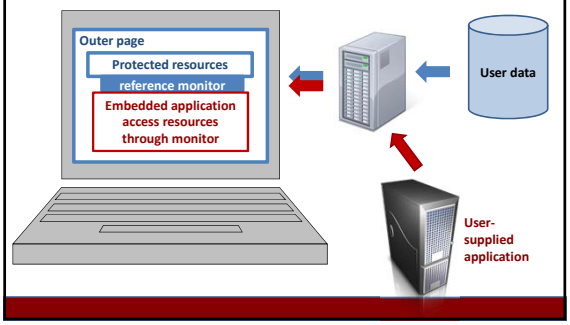
- Object-capability model [Miller, …]
  - Intriguing, not formally rigorous
  - Examples: E (Java), JoeE (Java), Emily (Ocaml), W7 (Scheme)
- Authority safety
  - Safety conditions sufficient to prevent
    - Authority leak (“only connectivity begets connectivity”)
    - Privilege escalation (“no authority amplification”)
  - Preserved by program execution
    - **Eliminates basis for our previous attacks**
- Capability safety
  - Access control model sufficient to imply authority safety
- **Theorems:** Cap safety ⇒ Auth safety ⇒ Isolation
  - Accepted examples satisfy our formal definitions

Google Caja

Mediated Access

How can trusted code provide access to security-critical resources?

Mediated access



ECMA Script 5 Strict Mode

- Restricted subset of JavaScript with "safer" semantics, e.g.
  - Assignment to an undeclared identifier does not create a property in the global object
  - Strict mode eval code cannot instantiate variables or functions in the variable environment of the caller to eval.
  - A *this* value of null or undefined is not converted to the global object …
  - Strict mode code may not include a WithStatement
  - …
- Goals
  - Provide language framework for code isolation
  - Protect trusted strict code against untrusted unstrict code

Language standard captures many properties explored in our research

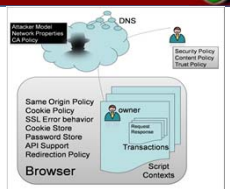
Research Results [S&P 2011]

- Use JavaScript sublanguage SES_{light}
  - ECMA-5 Strict mode, *with* scope-restricted *eval* and *without* setters/getters
- Prove language properties
  - Local variables can be renamed safely
  - Only variables of outer scope are accessible
- Develop automated code analysis tool
  - Given *trusted* code isolating a resource, determine if confinement is guaranteed
- Test tool on sample isolation libraries
  - Found new bug in well-tested Yahoo! ADSafe code

General Foundations for Web Security

Broader Foundations for Web Security

- Problem: Web platform and application security are not based on precise model
- Solution: Foundational model of web macro-platform supporting rigorous analysis
  - Apply formal modeling techniques and tools, e.g., network security ⇒ web
  - Precise threat models: web attacker, active network attacker, gadget attacker
  - Support trustworthy design of browser, server, protocol, web application mechanisms



- Initial case studies
  - Origin header
  - Cross-Origin Resource Sharing
  - Referer Validation,
  - HTML5 forms
  - WebAuth
- Find attacks, verify repairs

Goals and Challenges Ahead

- Language-based isolation
  - Better understanding of object-capability model
  - Apply to JavaScript and other languages: E, Joe-E, Emily, W7, ES 3 ⇒ ES 5
  - Better tools for working with secure JavaScript
  - Wider recognition and deployment through standards, browser implementations
- Web platform security
  - Formalize additional properties of web platform
    - Browser same-origin
    - Cookie policies
    - Headers, …
  - Prove correctness of accepted defenses
  - Improve design of central components
  - Guide design of emerging features (e.g., native client)

Conclusions

- The web is an exciting area for Computer Science
- Isolating untrusted JavaScript
  - Isolate untrusted application from hosting page
  - Isolate one untrusted application from another
  - Confinement: mediate access to critical resources
- Many more Web security problems
  - Define precise model of web application platform
  - Analyze protocols, conventions, attacks, defenses
    - Are http-only cookies useful? Is CSRF prevented?

References

- With A. Taly, S. Maffeis:
  - Operational semantics of ECMA 262-3 [APLAS'08]
  - Language-Based Isolation of Untrusted JavaScript [CSF'09]
  - Run-Time Enforcement of Secure JavaScript Subsets [W2SP'09]
  - Isolating JavaScript with Filters, Rewriting, and Wrappers [ESORICS'09]
  - Object Capabilities and Isolation of Untrusted Web Applications [S&P'10]
  - Automated Analysis of Security-Critical JavaScript APIs [S&P'11] (with T. + Google group)

Additional related work

[Yu,Chander,Islam,Serikov'07] *JavaScript instrumentation for browser security.*
Rewriting of JavaScript to enforce security policies based on edit-automata.

[Sands,Phung,Chudnov'09] *Lightweight, self protecting JavaScript.*
Aspect-oriented wrapping of DOM to enforce user-defined safety policies.

[Jensen,Møller,Thiemann'09] *Type analysis for JavaScript.*
Abstract-interpretation based analysis to detect basic type errors.

[Chugh,Meister,Jhala,Lerner'09] *Staged information flow for JavaScript.*
Static information flow analysis plus run-time checks for integrity and confidentiality.

[Livshits, Guarnieri'09] *GateKeeper: Mostly static enforcement of security and reliability policies for JavaScript code.*
Enforcing policies by filtering and rewriting based on call-graph and points-to analysis.

Web Sandbox (Scott Isaacs). Based on BrowserShield.
Rewriting and run-time monitoring with performance penalty.