

# Ajax Security

Douglas Crockford

Yahoo! Inc.

[http://crockford.com/codecamp/  
ajaxsecurity.ppt](http://crockford.com/codecamp/ajaxsecurity.ppt)

# The SAMY Saga

MySpace (2005)

# In a week Samy learned

- How to hide JavaScript in a URL in CSS.
- How to generate MySpace crumbs.
- How to submit a message to MySpace to update the friends list.
- How to submit a message that contains a script that sends a message containing a script.

His script added this line to  
the friends list of every user  
who looked at a page

But most of all, Samy is my  
hero

# The number of accounts corrupted doubled every hour

- Within 20 hours, his script was in one million accounts.
- He immediately stepped forward because he intended no harm.
- He was charged with a felony.
- Three years probation.
- 90 days community service.
- Banned from the internet.

What can an attacker do if  
he gets some script into  
your page?

An attacker can request additional scripts from any server in the world.

Once it gets a foothold, it can obtain all of the scripts it needs.

An attacker can read the  
document.

The attacker can see  
everything the user sees.

An attacker can make requests of your server.

Your server cannot detect that the request did not originate with your application.

If your server accepts SQL queries, then the attacker gets access to your database.

An attacker has control over the display and can request information from the user.

The user cannot detect that the request did not originate with your application.

An attacker can send  
information to servers  
anywhere in the world.

The browser does not  
prevent any of these.

Web standards require these  
weaknesses.

The consequences of a  
successful attack are horrible.

Harm to customers.

Loss of trust.

Legal liabilities.

Cross site scripting attacks  
were invented in 1995.

We have made no progress on  
the fundamental problems  
since then.

# Will the web ever reach the Threshold of Goodenoughness?

- + Discovery of vulnerabilities leads to corrections.

# Will the web ever reach the Threshold of Goodenoughness?

- + Discovery of vulnerabilities leads to corrections.
- + If the rate at which correcting vulnerabilities introduces new vulnerabilities, eventually goodenoughness should be achieved.

# Will the web ever reach the Threshold of Goodenoughness?

- + Discovery of vulnerabilities leads to corrections.
- + If the rate at which correcting vulnerabilities introduces new vulnerabilities, eventually goodenoughness should be achieved.
- Adding new features tends to introduce vulnerabilities: Unintended consequences.
  - For example: HTML5

# Will the web ever reach the Threshold of Goodenoughness?

- + Discovery of vulnerabilities leads to corrections.
- + If the rate at which correcting vulnerabilities introduces new vulnerabilities, eventually goodenoughness should be achieved.
- Adding new features tends to introduce vulnerabilities: Unintended consequences.
- If the fundamental assumptions are faulty, incremental correction never converges onto goodenoughness.

We are compiling an  
evergrowing corpus of  
hazards.

Ultimately, perfect knowledge  
of all of the vulnerabilities  
might never be obtained.

# Perfection is not an option.

It is unreasonable to require  
developers to have an adequate  
understanding of the current  
model.

# Is the web too big to fail?

Enter Flash and Silverlight.

The web came closer to  
getting it right than  
everything else.

But first: What goes wrong?

# The Standard Mistake

"We will add security in 2.0."

# The Itty Bitty -ity Committee

Quality Modularity Reliability  
Maintainability Security

You can't add security.  
You must remove insecurity.

# Confusion of Cryptography and Security.

Digital Living Room

# Confusion of Identity and Authority.

# Blame the Victim

# Confusion of Interest

The interests of a program  
may not be the same as  
the interests of the user.

# Confusion of Interest

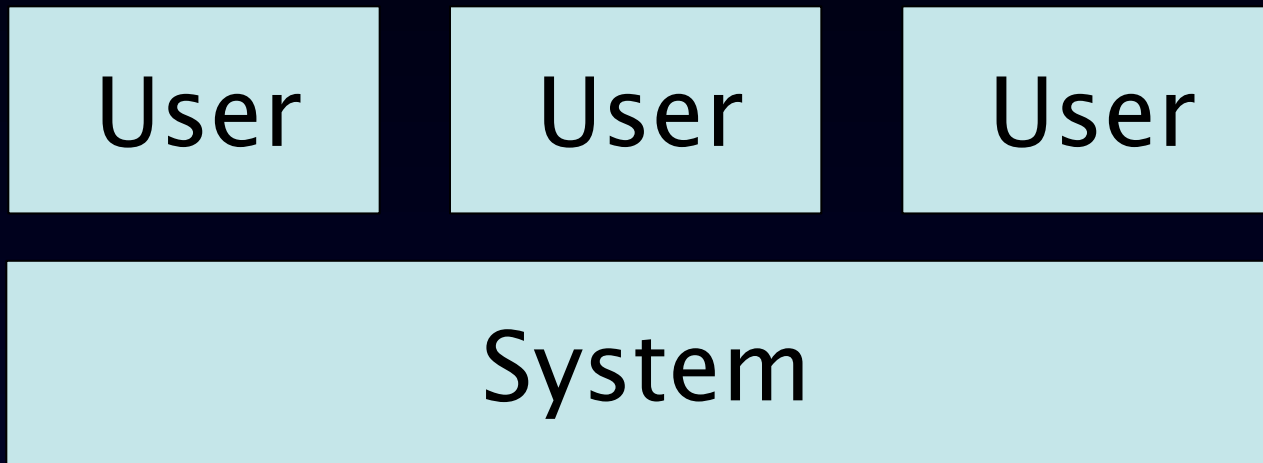
Computer

# Confusion of Interest

User

System

# Confusion of Interest



# Confusion of Interest

CP/M MS-DOS MacOS Windows

The system cannot distinguish  
between the interests of the  
user and the interests of the

This mostly works when software is  
expensive and intentionally  
installed.

It is not unusual for the purpose or use or scope of software to change over its life. Rarely are the security properties of software systems reexamined in the context of new or evolving missions. This leads to insecure systems.

On the web we have casual,  
promiscuous, automatic,  
unintentional installation of

The interests of the user and  
of the program must be  
distinguished.

The browser successfully  
distinguishes the interests  
of the user and the interests

# Confusion of Interest

The browser is a significant improvement, able to distinguish the interests of users and sites (usually).

Site

Site

Site

User

Browser

But within a page,  
interests are confused.

An ad or a widget or an Ajax  
library gets the same rights as  
the site's own scripts.

# Turducken



This is not a Web 2.0  
problem.

All of these problems came  
with Netscape 2 in 1995.

We are mashing things up.

There are many more  
interested parties represented  
in the page.

A mashup is a self-inflicted  
XSS attack.

(Advertising is a mashup.)

JavaScript got close  
to getting it right.

A secure dialect is obtainable.  
ADsafe and Caja leading the  
way.

We need a new security  
model:  
Object Capabilities.

Robust Composition, Mark  
Miller

[http://erights.org/talks/  
thesis/](http://erights.org/talks/thesis/)

Cooperation under  
mutual suspicion.

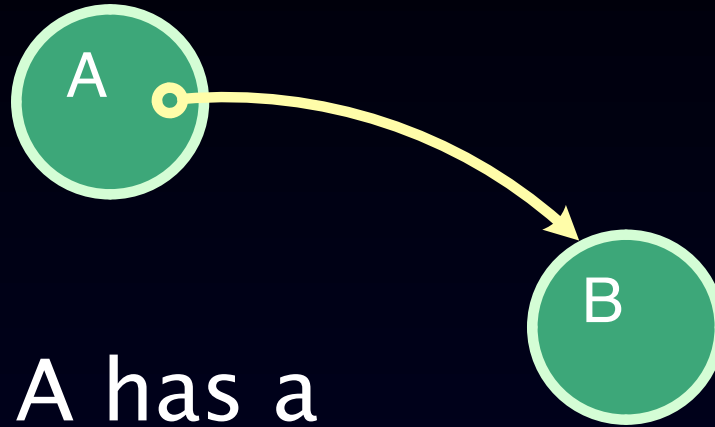
# An Introduction to Object Capabilities

A is an Object.

Object A has  
state and  
behavior.



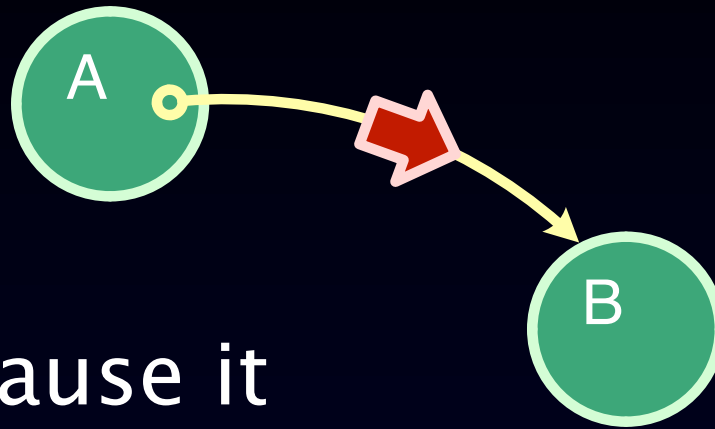
# has-a



Object A has a  
reference to  
Object B.

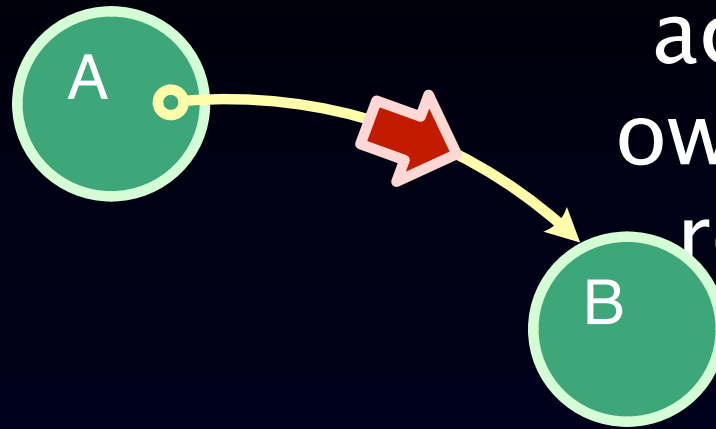
An object can have  
references to other  
objects.

Object A can  
communicate  
with Object B...



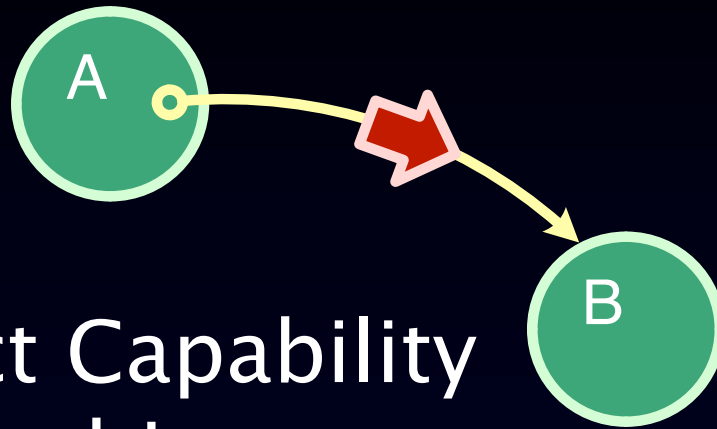
...because it  
has a reference  
to Object B.

Object B provides an interface that constrains access to its own state and references.



Object A does not get access to Object B's innards.

Object A does not have a reference to Object C, so Object A cannot communicate with Object C.



In an Object Capability System, an object can only communicate with objects that it has references to.



An Object Capability System is produced by constraining the ways that references are

A reference cannot be obtained simply by knowing the name of a global variable or a public class.

# There are exactly three ways to obtain a reference.

1. By Creation.
2. By Construction.
3. By Introduction.

# 1. By Creation

If a function creates an object,  
it gets a reference to that  
object.

## 2. By Construction

An object may be endowed by its constructor with references.

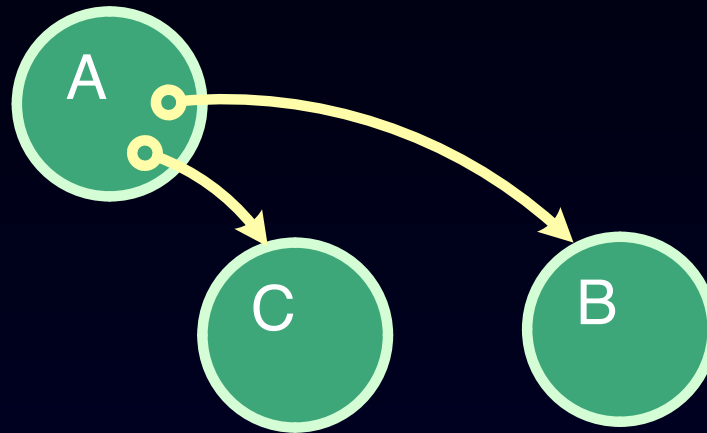
This can include references in the constructor's context and inherited references.

# 3. By Introduction

A has a references to B and C.

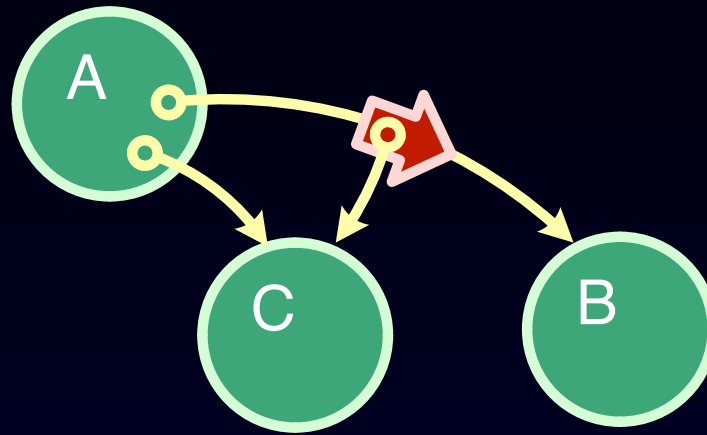
B has no references, so it cannot communicate with A or C.

C has no references, so it cannot communicate with A or B.



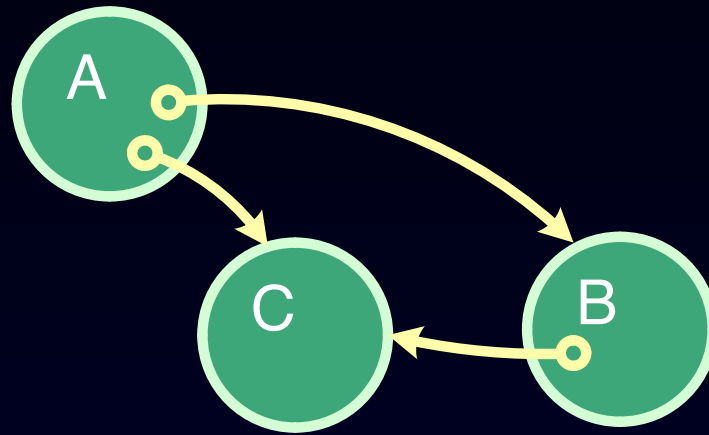
# 3. By Introduction

A calls B, passing a reference to C.



# 3. By Introduction

B is now able to communicate with C.



It has the capability.

If references can only be  
obtained by Creation,  
Construction, or Introduction,

If references can be  
obtained in any

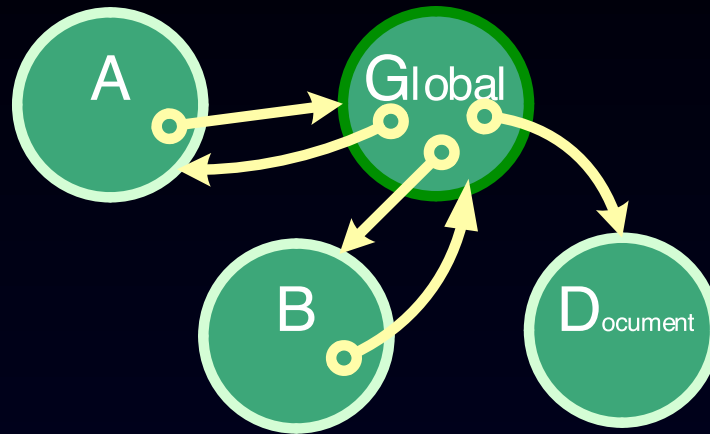
# Potential weaknesses include

1. Arrogation.
2. Corruption.
3. Confusion.
4. Collusion.

# 1. Arrogation

- To take or claim for oneself without right.
- Global variables.
- public static variables.
- Standard libraries that grant powerful capabilities like access to the file system or the network or the operating system to all programs.
- Address generation.

JavaScript's Global object gives powerful capabilities to every object.



There is too much ambient authority.

## 2. Corruption

It should not be possible to tamper with or circumvent the system or other objects.

# 3. Confusion

It should be possible to create objects that are not subject to confusion. A confused object can be tricked into misusing its capabilities.

# 4. Collusion

- It must not be possible for two objects to communicate until they are introduced.
- If two independent objects can collude, they might be able to pool their capabilities to cause harm.
- For example, I can give gasoline to one object, and matches to another. I need to be confident that they cannot collude.

# Rights Attenuation

- Some capabilities are too dangerous to give to guest code.
- We can instead give those capabilities to intermediate objects that will constrain the power.
- For example, an intermediate object for a file system might limit access to a particular device or directory, or limit the size of files, or the number of files, or the longevity of files, or the types of files.

Ultimately, every object  
should be given exactly the  
capabilities it needs to do its

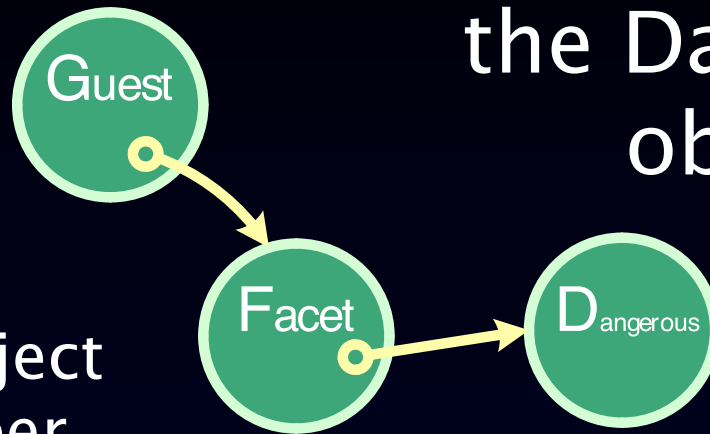
Capabilities should be granted on a  
need-to-do basis.

Information Hiding – Capability  
Hiding.

Intermediate objects, or  
**facets**, can be very light  
weight.

Class-free languages can be  
especially effective.

The Facet object limits the Guest object's access to the Dangerous object.

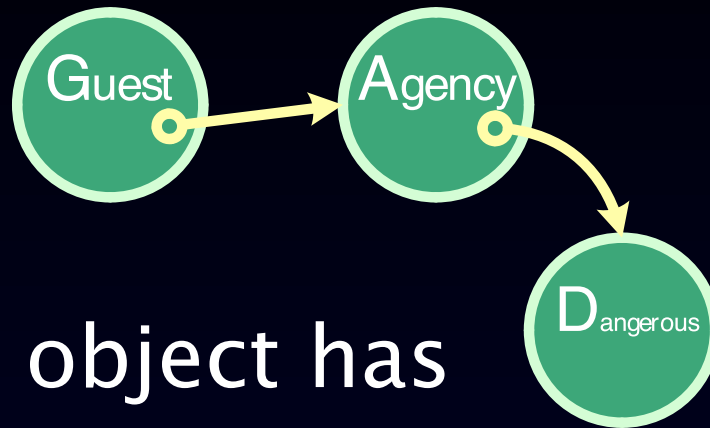


The Guest object cannot tamper with the Facet to get a direct reference to the Dangerous object.

References are not  
revocable.

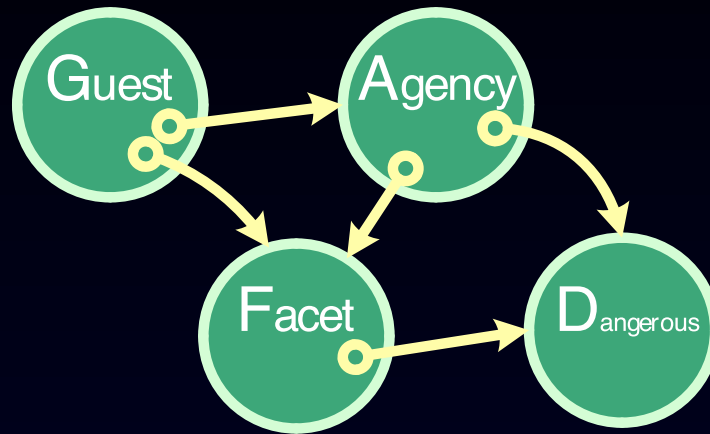
Once you introduce an object,  
you can't ask it to forget it.

You can ask, but you should not depend  
on your request being honored.



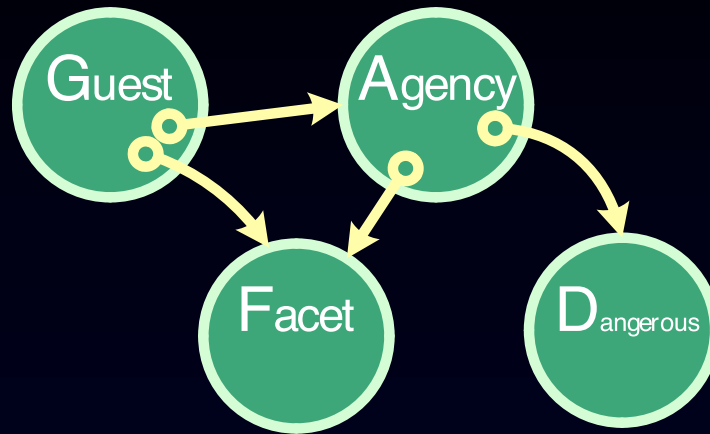
The Guest object has a reference to an Agency object. The Guest asks for an introduction to the Dangerous object.

The Agency object makes a Facet,  
and gives it to the Guest.



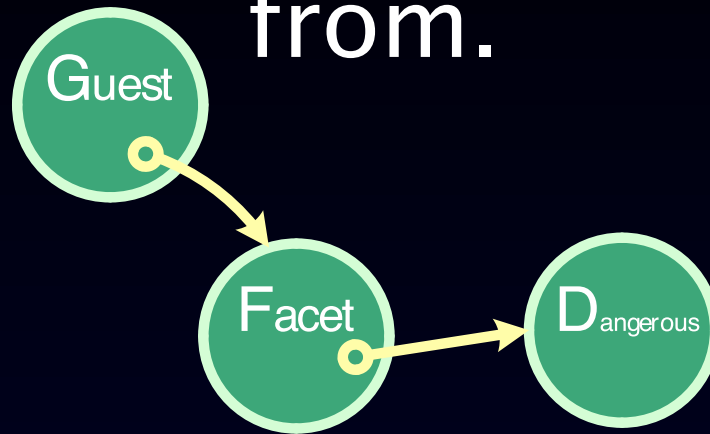
The Facet might be a simple pass  
through.

When the Agency wants to revoke the capability, it tells the Facet to forget its capability.



The Facet is now useless to the Guest.

A Facet can mark requests so that the Dangerous object can know where the request came from.



# Facets

- Very expressive.
- Easy to construct.
- Lightweight.
- Power Reduction.
- Revocation.
- Notification.
- Delegation.
- The best OO patterns are also capability patterns

# Good Object Capability Design is

# The DOM is dangerous

- But the Ajax libraries are converging on a much better API.
- We need to replace the DOM with something that is more portable, more rational, more modular, and safer.
- We need to replace the DOM with something that is less complicated, less exceptional, less grotesque.

W3C is moving  
in the opposite direction

HTML5 needs to be reset.

Or W3C needs to be abolished.

The advertising problem  
is serious.

The publishers must go as one to  
the advertising industry and  
demand basic standards of ad  
quality.



A system for safe web advertising.

<http://www.ADsafesafe.org/>

# ADsafe

- ADsafe makes it possible to package fully interactive ads that are denied the capability to do harm.
- ADsafe defines a safe subset of JavaScript which can be statically verified without rewriting.
- The static verification can be performed at every stage in the ad delivery pipeline, from creative development to post-consumer compliance testing.
- Unlike Caja, ADsafe imposes no runtime performance penalties.

# ADsafe

- ADsafe is a JavaScript subset that adds capability discipline by deleting features that cause capability leakage.
- No global variables or functions may be defined.
- No global variables or functions can be accessed except the `ADSAFE` object.
- No use of `this`.
- These words cannot be used: `arguments` `callee` `caller` `constructor` `eval` `prototype` `unwatch` `valueOf` `watch`
- Words starting with `_` cannot be used.
- Use of the `[]` subscript operator is restricted.

# The DOM

- Document Object Model API.
- The DOM provides no containment at all.
- A document is represented as a tree of nodes.
- Every node has a capability to the root.
- The root has a capability
  - to every node in the tree
  - to load dangerous scripts from any server in the world
  - to send data to any server in the world
- ADsafe does not permit direct access to the DOM.
- ADsafe's API is minimal but adequate.

# ADsafe relies on static validation

- ADsafe does not modify the widget.
- Debugging is much easier.
- No performance penalty.
- Validation can be performed at every stage in the ad pipeline, from creative to post-consumer.
- JSLint.

# Unfortunately

- It is extremely unlikely that existing code will run under ADsafe. New code must be written.
- Dangerous, but popular, practices are not allowed. (`document.write`)
- ADsafe cannot protect the widget from the page. It can only protect the page from the widget and the widgets from each other.

# ADsafe DOM Interface

- Light weight.
- Query-oriented.
- Scope of queries is strictly limited to the contents of a widget's `<div>`.
- Guest code cannot get direct access to any DOM node.

```
<div id="WIDGETNAME_">
    html markup required by the widget
<script>
    "use strict";
    ADSAFE.id("WIDGETNAME_");
</script>
<script src="ADsafe approved url"></script>
<script>
    "use strict";
    ADSAFE.go("WIDGETNAME_", function (dom, lib) {

        // This is where the code for the widget is placed. It can access
        // the document through the dom parameter, allowing it indirect
        // access to html elements, allowing it to change content, styling,
        // and behavior.

        // Each library file can give itself a name. This script can access
        // the library file as lib.name.

    });
</script>
```

We have gone as far as we can  
go on luck and good  
intentions.

We need, at very long last,  
to get it right.

Fixing the web will be very  
hard.

Not doing this will be even  
harder.

# ECMAScript

- The JavaScript standard is called ECMAScript.
- This year: The Fifth Edition.
- The new edition corrects some of the problems that make ADsafe and Caja so difficult.
- Object hardening.
- These features are a direct consequence of Yahoo's participation in the standards process.
- We are looking at defining a safe subset within the language itself. If we are successful in getting this specified and adopted, then we will essentially have Caja-like safety and generality built into the browser itself with no performance penalty.

# The DOM

- The DOM is insecure, inefficient, incomplete, complicated, irregular, and difficult to use.
- W3C's HTML5 committee is actively enlarging the DOM API, making it less secure, more complicated, and slower.
- Piling on of features does not ultimately produce good systems.
- Complexity is the enemy of security.