ADsafe?
Malicious JavaScript circulating in ad networks that serve over 50 percent of the Web can open the door to malware, without even clicking on the ad.

Internet security software and common sense can steer you away from the some of the Web’s skeeviest destinations to avoid malware, but when it’s embedded in ads from some of the biggest advertising networks out there, you may be looking at infection at every turn.

According to Avast, a manufacturer of antivirus software, major ad networks affiliated with Yahoo, Fox and Google have been distributing “poisoned ads” containing malicious code. The so-called JS:Potentially attack uses Javascript that can open the door to malware attacks and zero-day exploits, all without clicking on anything.

Because the infected networks serve more than 50 percent of all online advertisers, poisoned ads have appeared
There are at least 3 major efforts to solve this problem, but none verify their security claims.

I propose to verify the claims made by ADsafe.
To verify ADsafe, I need to:

1. Formalize \textit{ADsafety} at a high level;

2. Decompose this definition into small, provable properties, and;

3. Show ADsafety follows from their composition.
What claims made by ADsafe?
The client can't load new, unvetted code at runtime.

The widget can only interact with its subtree of the DOM.

No direct references to DOM nodes are exposed to the widget.
How does ADsafe try to guarantee this?
1. A runtime library, which contains guards against unwanted behavior.

```javascript
ADSAFE.get = function(obj,x) {
    if(reject(x))
        return error("Bad field name");
    else
        return obj[x];
}
```

(simplified)
2. Static restrictions on the JavaScript code the widget developer can write.

```javascript
obj[x];

ADSAFE.get(obj,x);

document.createElement("div");

dom.tag("div");

eval();
```
How can we prove whether this works?
What is ADSAFE.get ensuring?

ADSAFE.get = function(obj, x) {
    if(reject(x))
        return error("Bad field name");
    else
        return obj[x];
}

function reject(name) {
    return banned[name] ||
        name.charAt(0) === '_';
}

banned = {"eval" : true,
    ...
};

...that the client can't access fields

(1) named in banned,

(2) starting with '_'.

This should hold for all 1800 lines of ADsafe (not just ADSAFE.get).
Don't return the global window object.
Deny access to dangerous fields.
Never call:
  - `eval`
  - `setTimeout`
  - `setInterval`

Never return the result of various document function calls.

Only create tags from a whitelist.
Theoretical and practical support comes from:


Related JavaScript work:
