Safe Wrappers and Sane Policies for Self Protecting JavaScript

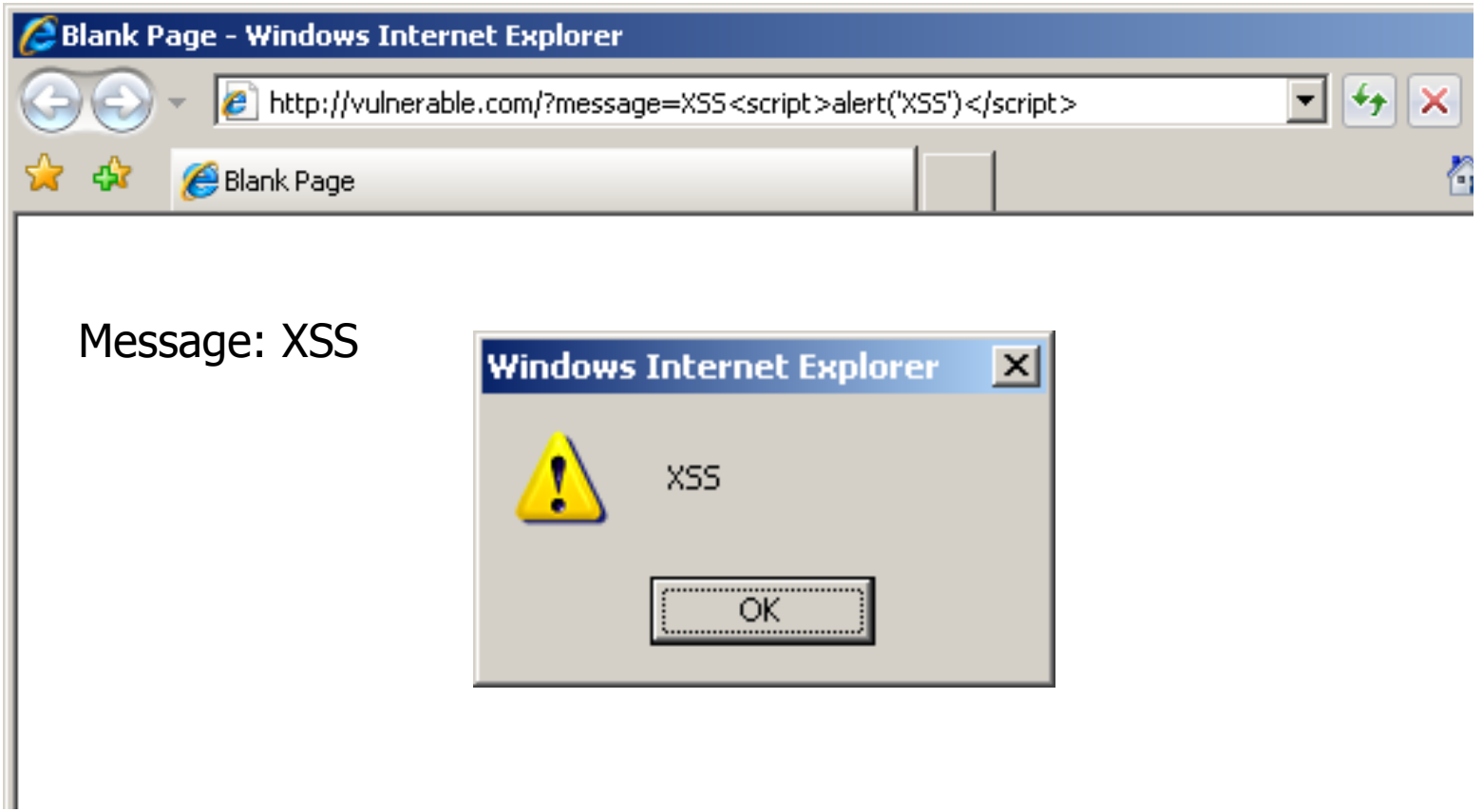
Jonas Magazinius, Phu Phung and David Sands

OWASP

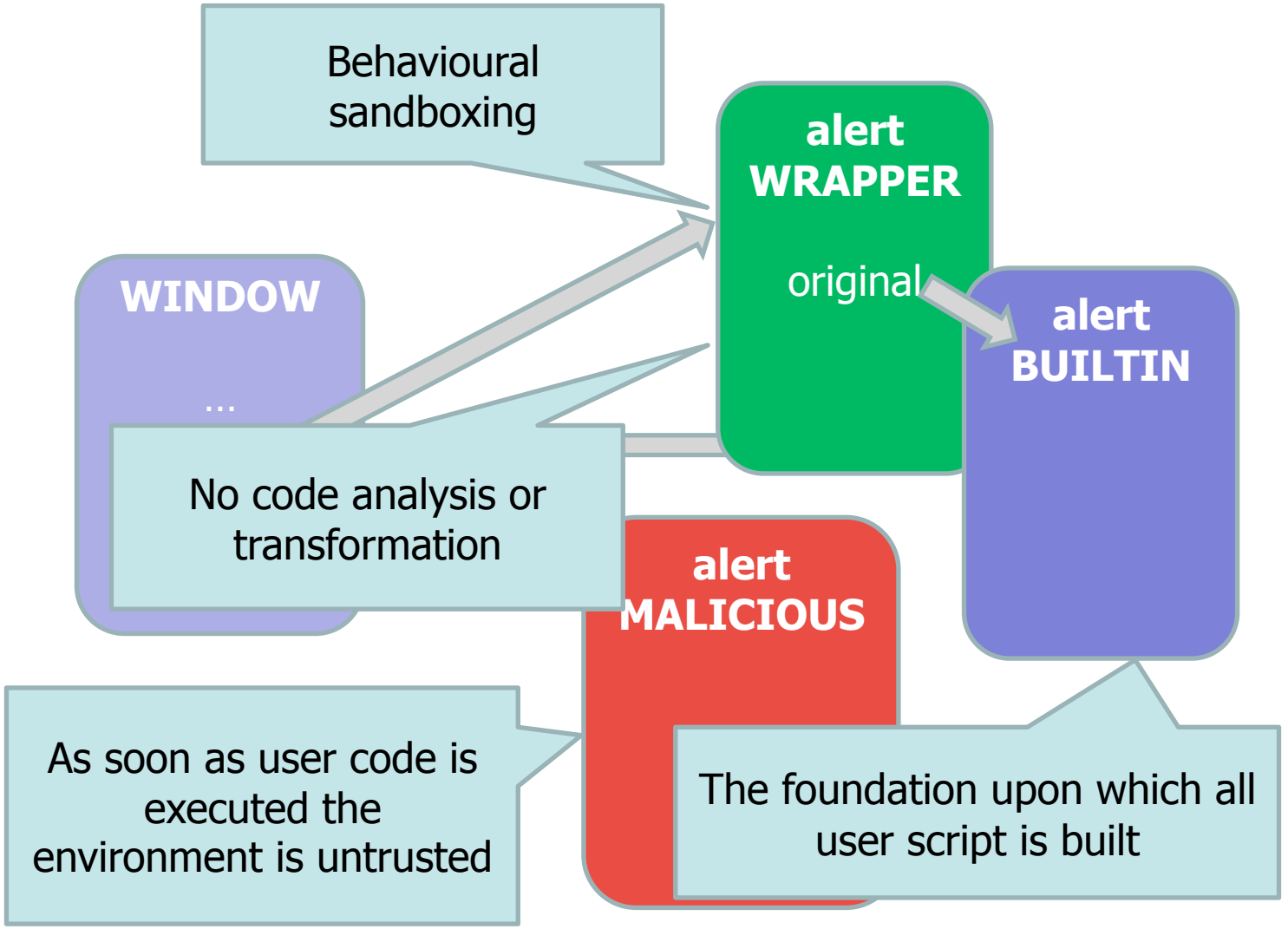
Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

The OWASP Foundation
http://www.owasp.org

Problem


Blank Page - Windows Internet Explorer
http://vulnerable.com/?message=XSS<script>alert('XSS')</script>
Blank Page
Message: XSS
Windows Internet Explorer
XSS
OK

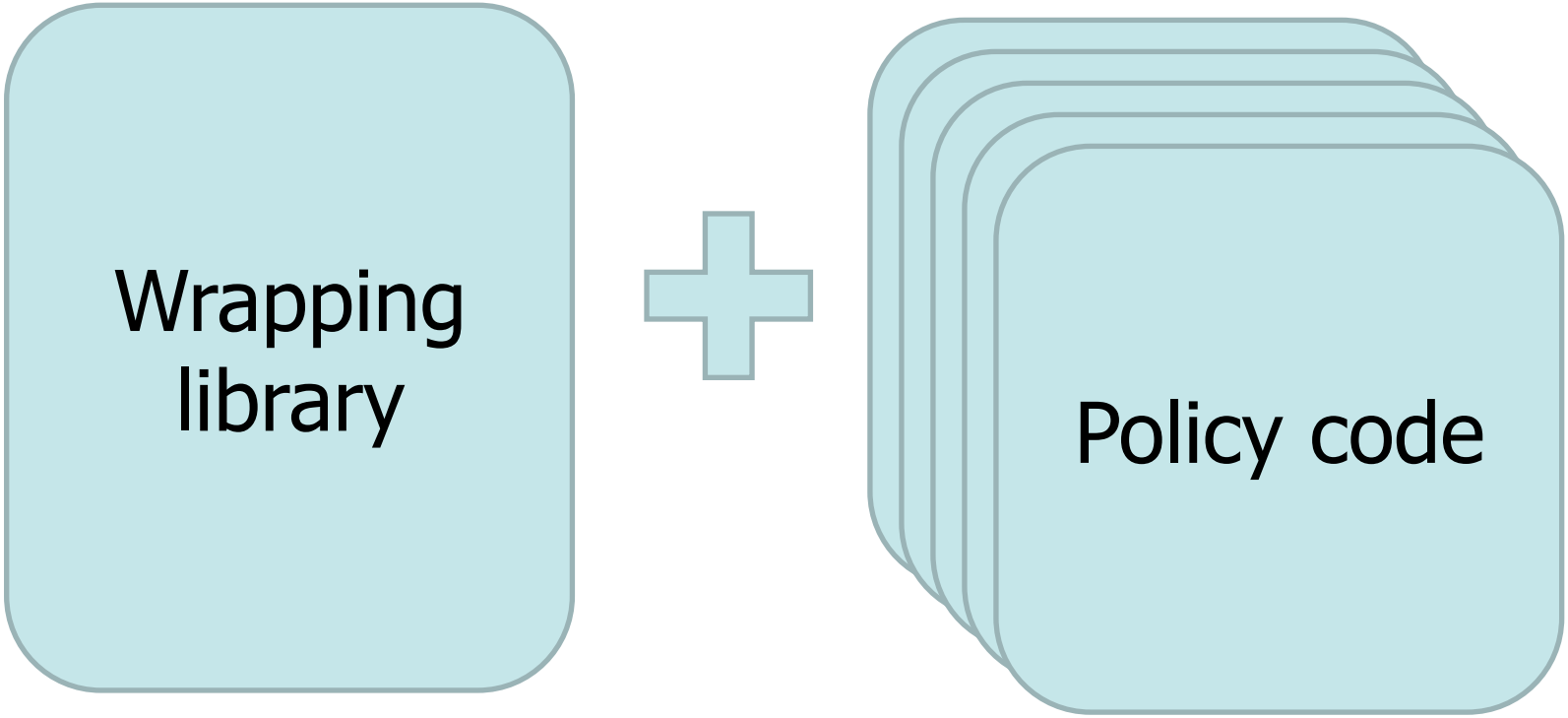
Light-Weight Self Protecting JavaScript



Wrapper + policy part

- Implementation problems
  - Safe wrappers
  - Sane policies
    - Don’t shoot yourself in the foot

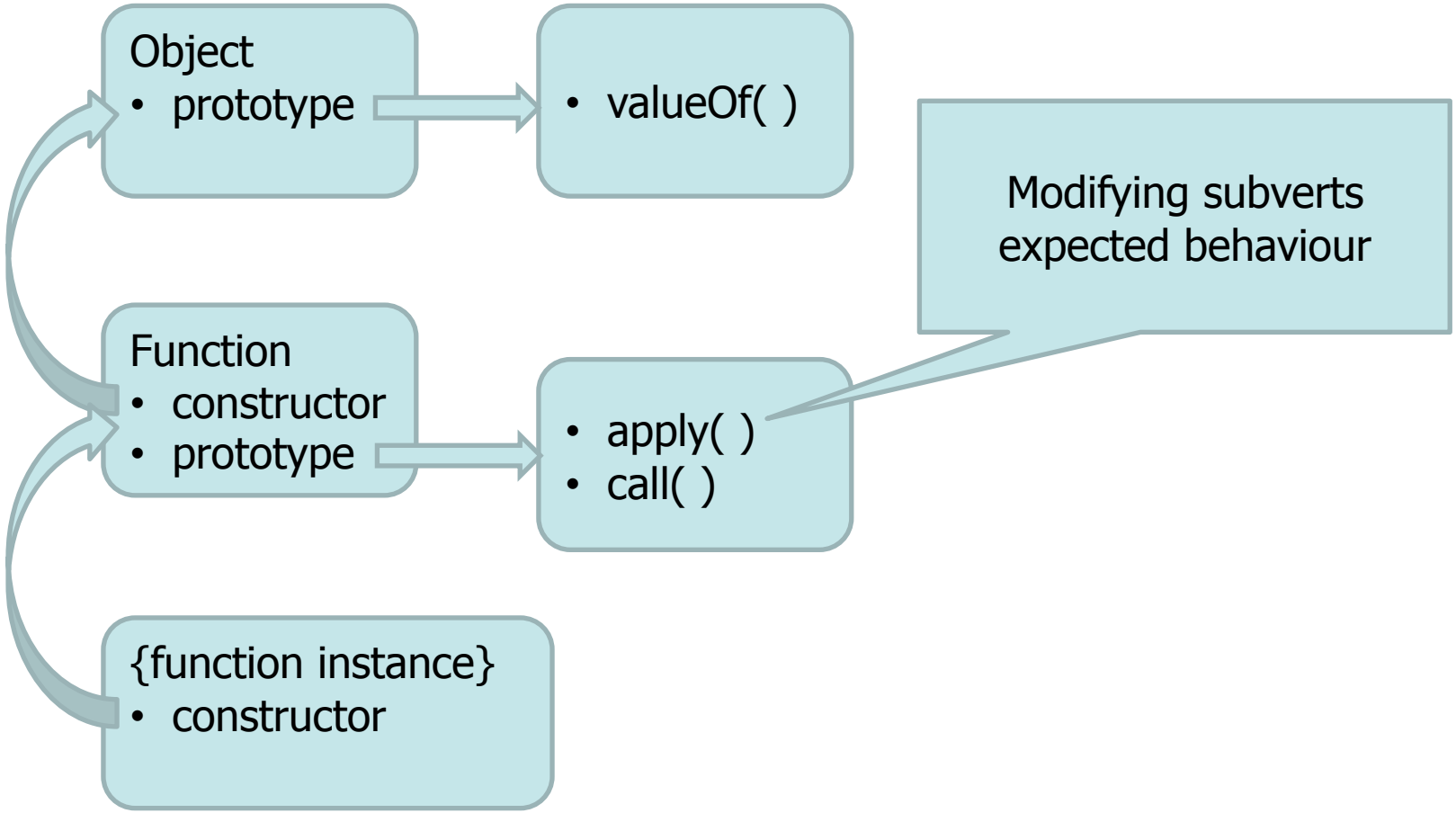
Light-Weight Self Protecting JavaScript



Breaking and Fixing the Wrapper

- Function and Object Subversion
- Global Setter Subversion
- Static and Dynamic Aliases

Function and object subversion



Function and Object Subversion

Wrapper code:

original.apply(this,args)

Subversion:

```
var org;
Function.prototype.apply =
    function(){ org = this}
```

Fixing the wrapper:

original.apply=apply

Global setter subversion

Function closure:

```
function x() {

    …

    var x = {secret: 1};

    …

}
```

Global setter for "secret":

```
function setSecret(sec) {
    // We have your secret!
    alert(sec);
}
```

Global Setter subversion

Wrapper code

```
policy({args: arguments,
        proceed: original})
```

Subversion

```
var org;
Object.prototype.
    __defineSetter__('proceed',
    function(o) { org = o })
```

Fixing the wrapper:

No temporary objects?

Use "safe" objects…

Change JavaScript: Don't execute setters upon instantiation

Aliases

Static aliases

- window.alert
- Window.prototype.alert
- window.__proto__.alert
- constructor.prototype.alert

Dynamic aliases

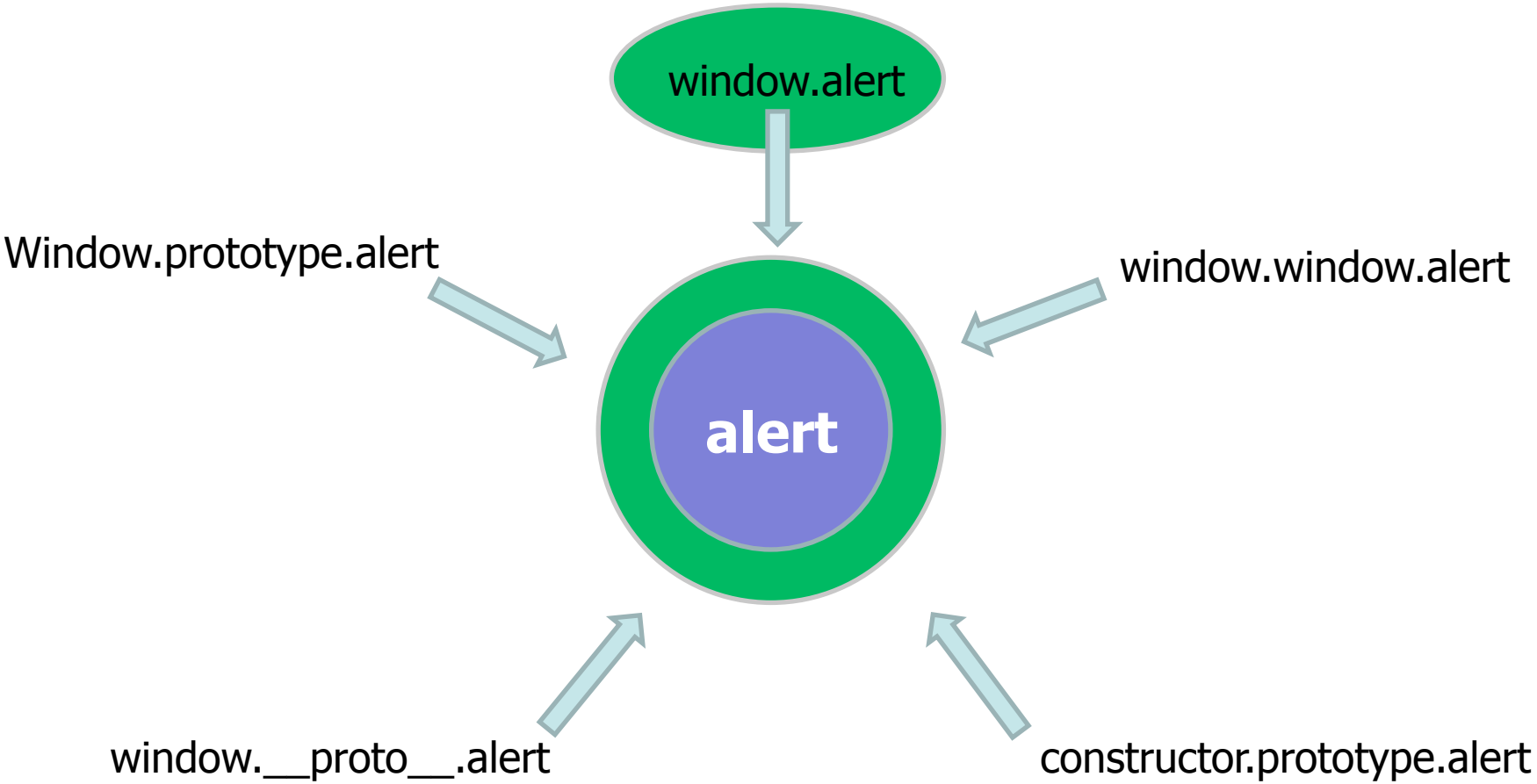
From iframe:

1. x=document.createElement('iframe')
2. document.appendChild(x)
3. alert=x.contentWindow.alert

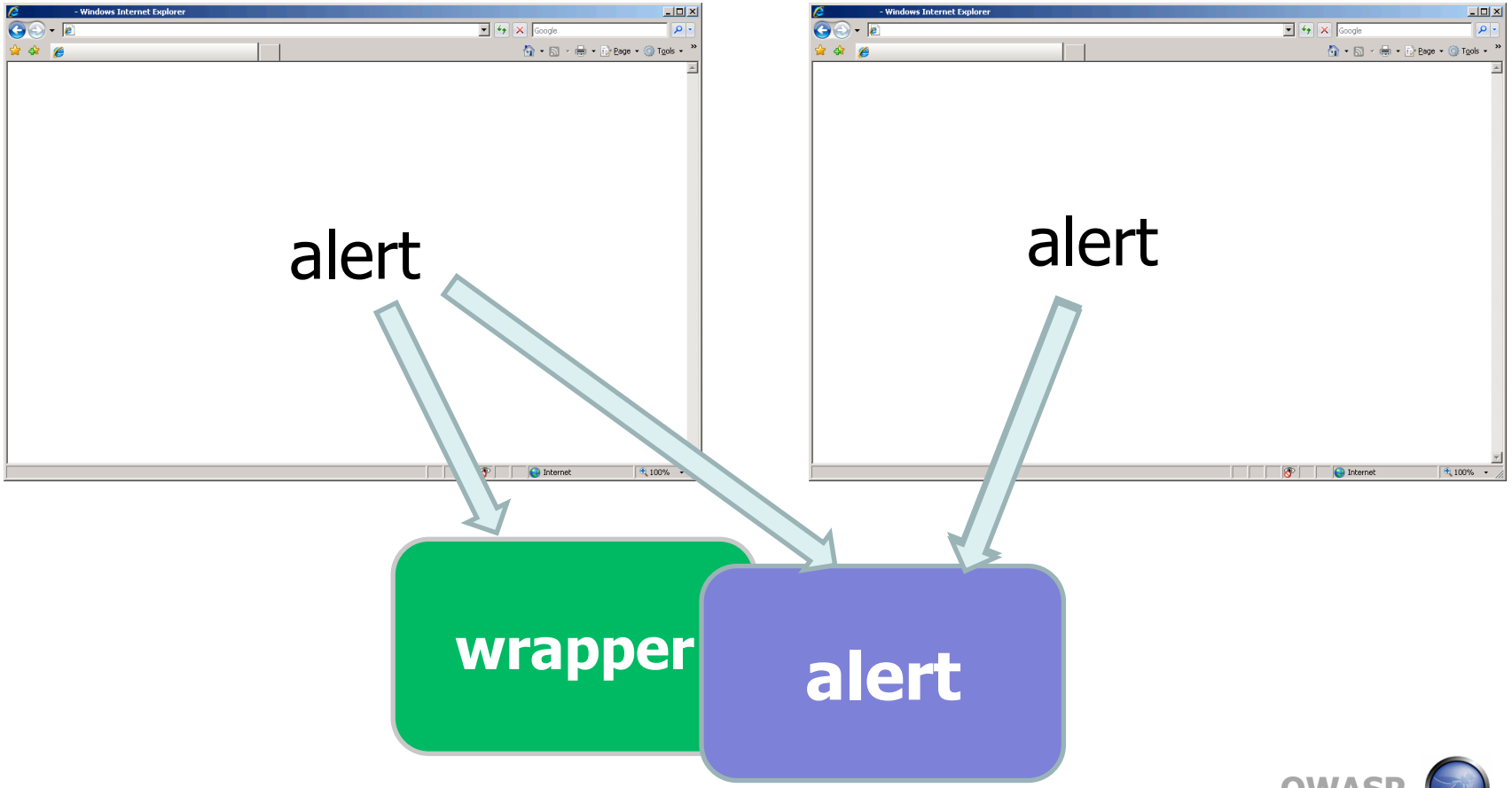
From other window

1. x=open("")
2. alert=x.alert

Static aliases



Dynamic aliases



Declarative Policies

- Function and Object Subversion for Policies
- Non-declarative policies vs. Declarative policies

Function and Object Subversion for Policies

Policy code

```
var whitelist =
        {"good.com":true,
        "good2.org":true}
if(whitelist[address.substr(...)])
```

Subversion

```
Object.prototype[‘evil.com’]= true

String.prototype.substr =
    function() { return ‘good.com’; }
```

Fixing subversion

- **hasLocalProperty()**
- **Use “safe” objects…**

The policy writer should not have to remember this…

"Safe" objects

- safe() function
  - Creates a blank object which **does not** inherit from the prototype-chain
    - {__proto__: null}
  - Recursively copies all fields from the input object to the newly created copy

- Limitations
  - Sometimes inheritance is required

Non-declarative vs. declarative policies

Policy code

```
if (whitelist[address])
    img.src = address;
```

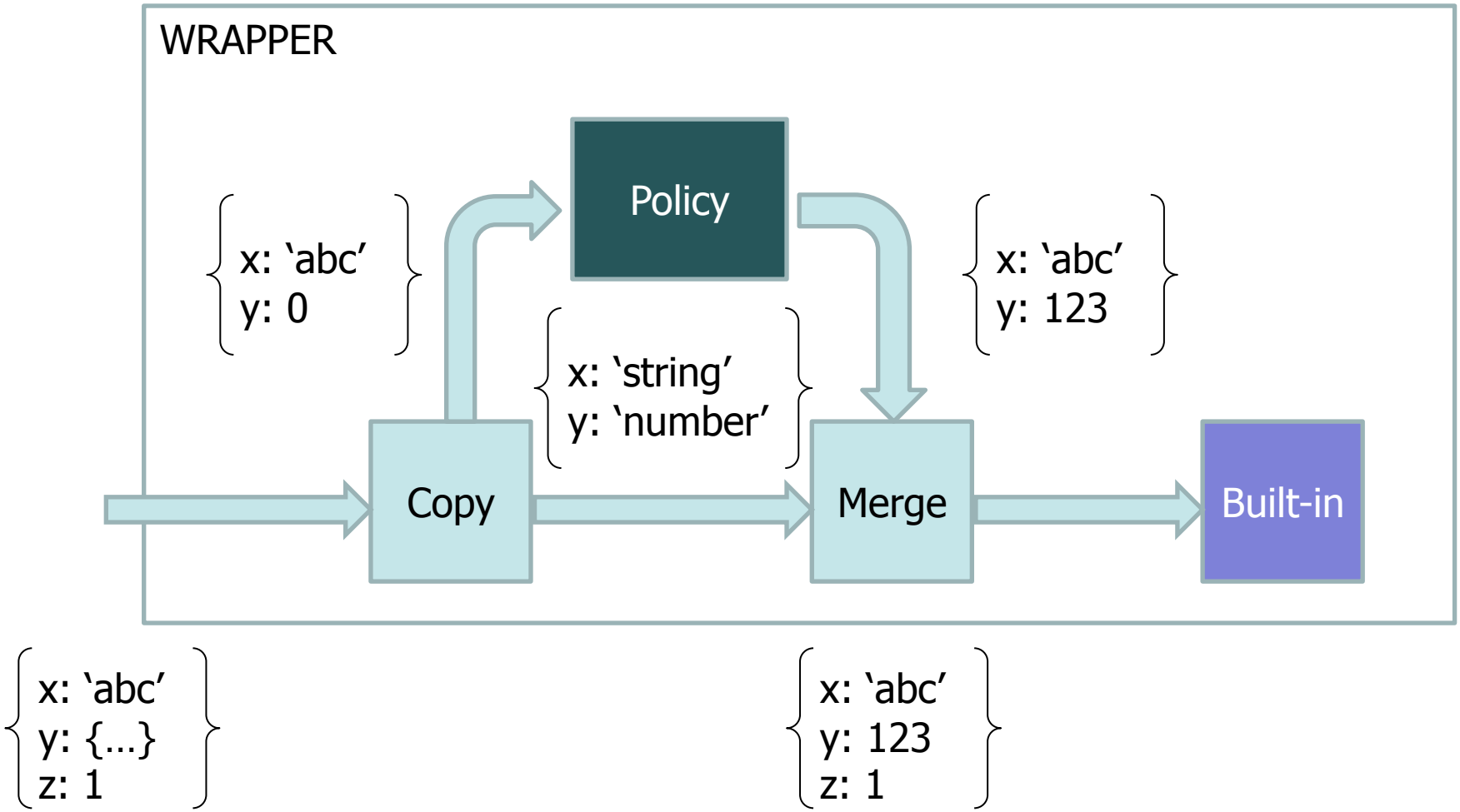
Fixing problem

Policy declare which types it expects in a type language and monitor enforces it

Attack

```
x = {toString: function() {
    this.toString=
        function()'bad.com';
    return  'good.com';
    }
}
```

Declarative policies



More information

- The paper is available at:
  - http://www.cse.chalmers.se/~phung/projects/jss