Goal of the policy

A popup security policy example: allow at most 2 popup windows that its URL must be in the whitelist and the new popup window must has location bar

Policy to disallow iframe creation

Policy preventing leakage of information through loading of new images

Policy controlling the redirection of a webpage.

Policy preventing impersonation attacks using XMLHttpRequest object.

Policy preventing potential forgery attacks.

Policy preventing potential forgery attacks by injecting bad links.

Policy to disable methods that might cause resource abuse.

The intended policy is to limit the content of a frame to URLs specified by a whitelist

Bob may read the amount property and invoke the deposit method

A policy that restricts a subtree to read-only if the root’s class name includes example. Meth- ods named shake may also be invoked.

Object x has two properties, one of which (secret) is not meant to be readable by the recipient of the wrapper.

a policy, which also allows subdomains of the domains in the whitelist

The first part, ’a.?’, gives read/write access to all properties of the object in the a property, but a itself is read-only. The second part, ’b∗’, allows read and write access to an arbitrarily long chain of properties named b.

The policy in line 2 allows the ad script read-only access to the email message body. The policy in line 5 permits the ad script write access to
the sidebar on the right of the email message body.

Policy to simply disallow any code from being introduced after a certain point, such as after the main library loads. We encode disabling scripts by returning the empty string back to the JavaScript interpreter.

This policy aims to prevent inline scripts: if a script’s context is not the global object, it is an inline script, so the empty string is returned to the interpreter.

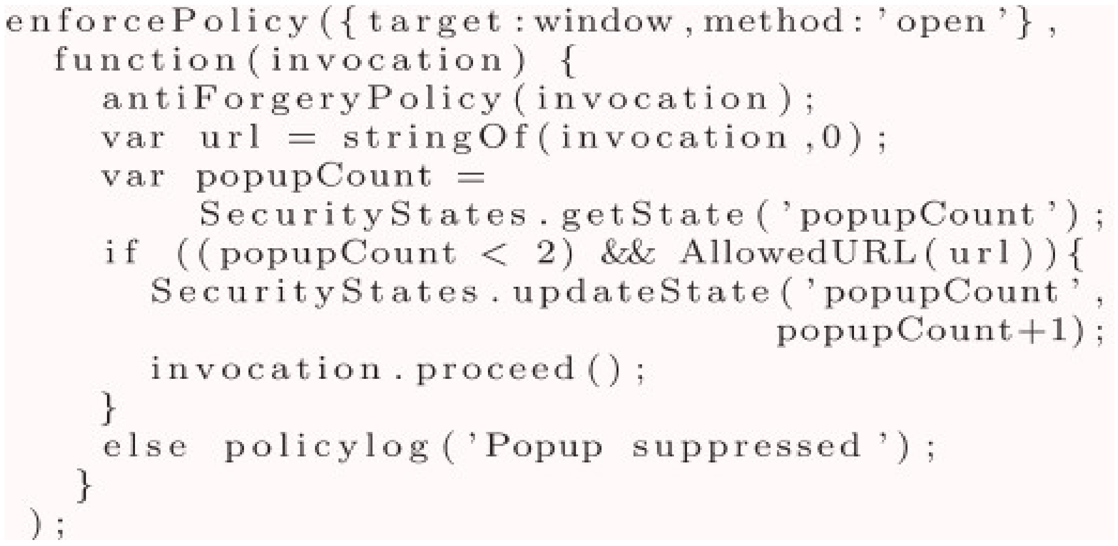
We can ensure that statically loaded script tags have a source listed in whitelist w.

the following policy prevents inline scripts from being loaded as descendants of a <noinlinescript> tag:

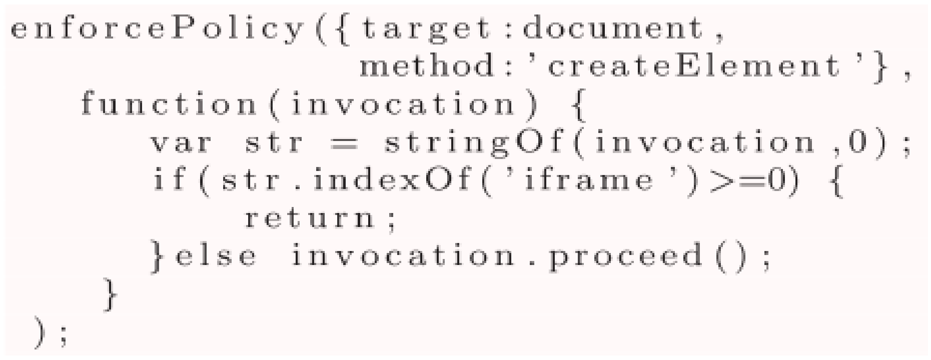
The following policy ensures, if a username and password is supplied, that the connection is over HTTPS.

Policy to disable JavaScript access to cookies.

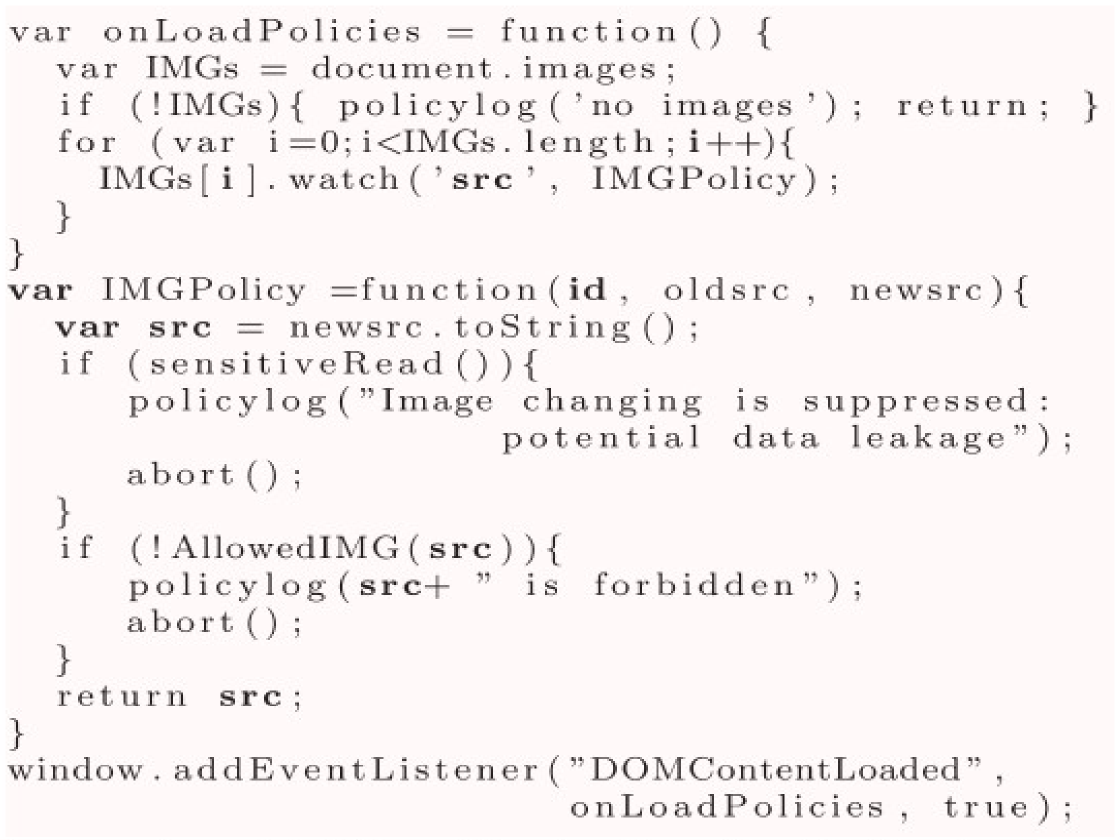
How is it expressed in the paper?



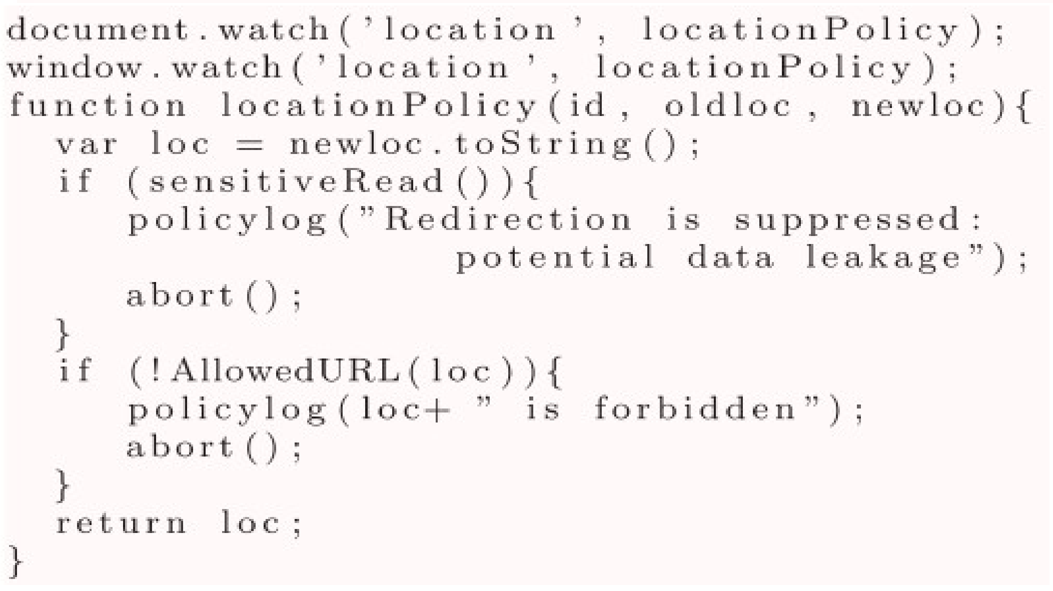
```
enforcePolicy({target:window,method:'open'},
  function(invocation) {
    antiForgeryPolicy(invocation);
    var url = stringOf(invocation,0);
    var popupCount =
        SecurityStates.getState('popupCount');
    if ((popupCount < 2) && AllowedURL(url)){
      SecurityStates.updateState('popupCount',
                                  popupCount+1);
      invocation.proceed();
    }
    else policylog('Popup suppressed');
   }
 );
```



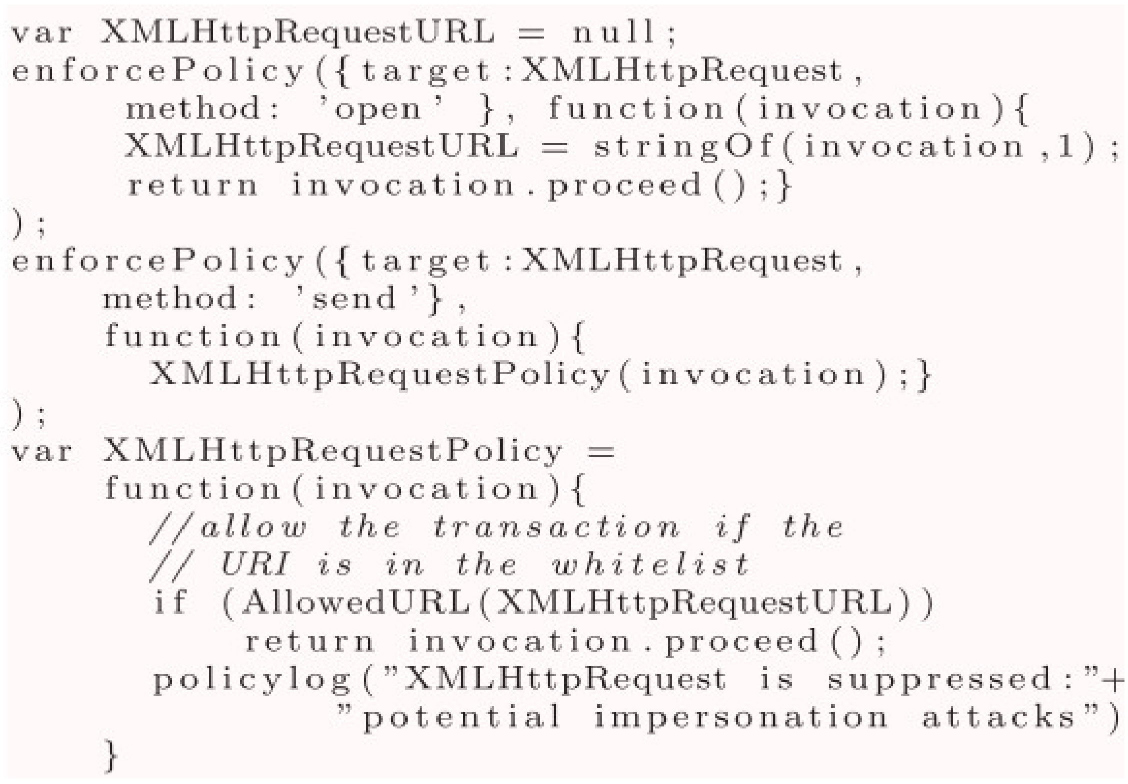
```
enforcePolicy({target:document,
               method:'createElement'},
   function(invocation) {
      var str = stringOf(invocation,0);
      if(str.indexOf('iframe')>=0) {
          return;
      }else invocation.proceed();
    }
 );
```



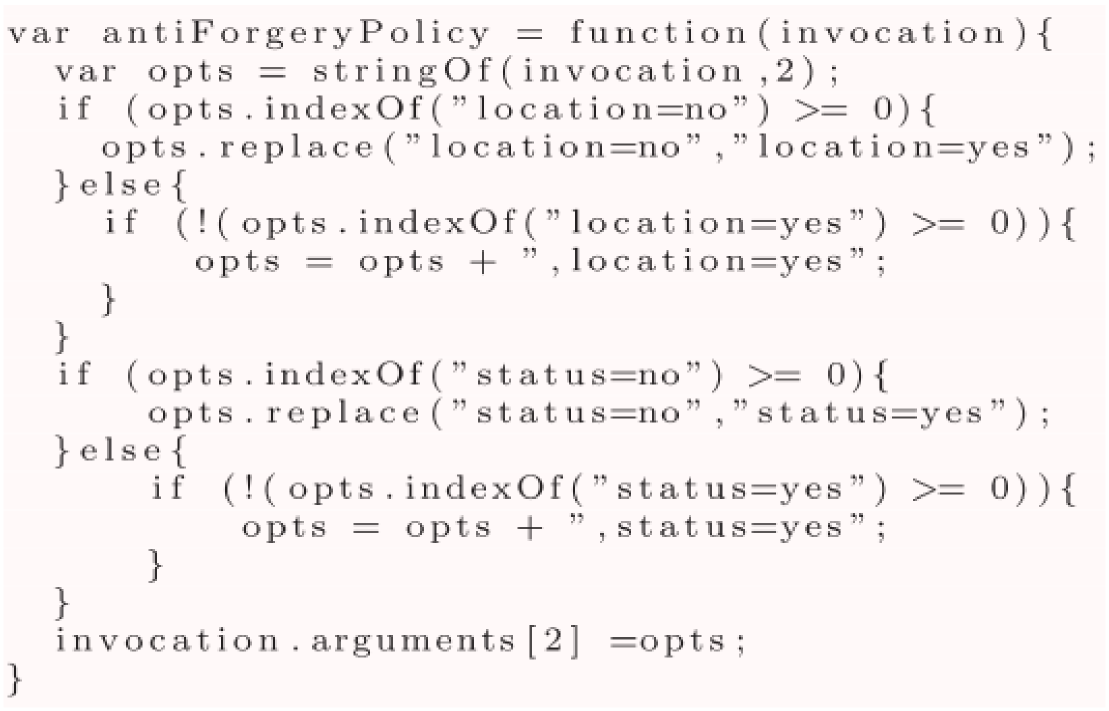
```
var onLoadPolicies = function() {
  var IMGs = document.images;
  if (!IMGs){ policylog('no images'); return; }
  for (var i=0;i<IMGs.length;i++){
    IMGs[i].watch('src', IMGPolicy);
  }
}
var IMGPolicy =function(id, oldsrc, newsrc){
  var src = newsrc.toString();
  if (sensitiveRead()){
     policylog("Image changing is suppressed:
                          potential data leakage");
     abort();
  }
  if (!AllowedIMG(src)){
     policylog(src+ " is forbidden");
     abort();
  }
  return src;
}
window.addEventListener("DOMContentLoaded",
                            onLoadPolicies, true);
```



```
document.watch('location', locationPolicy);
window.watch('location', locationPolicy);
function locationPolicy(id, oldloc, newloc){
  var loc = newloc.toString();
  if (sensitiveRead()){
     policylog("Redirection is suppressed:
                          potential data leakage");
     abort();
  }
  if (!AllowedURL(loc)){
     policylog(loc+ " is forbidden");
     abort();
  }
  return loc;
}
```



```
var XMLHttpRequestURL = null;
enforcePolicy({target:XMLHttpRequest,
     method: 'open' }, function(invocation){
     XMLHttpRequestURL = stringOf(invocation,1);
     return invocation.proceed();}
);
enforcePolicy({target:XMLHttpRequest,
    method: 'send'},
    function(invocation){
      XMLHttpRequestPolicy(invocation);}
);
var XMLHttpRequestPolicy =
    function(invocation){
      //allow the transaction if the
      // URI is in the whitelist
      if (AllowedURL(XMLHttpRequestURL))
          return invocation.proceed();
      policylog("XMLHttpRequest is suppressed:"+
              "potential impersonation attacks")
    }
```



```
var antiForgeryPolicy = function(invocation){
  var opts = stringOf(invocation,2);
  if (opts.indexOf("location=no") >= 0){
    opts.replace("location=no","location=yes");
  }else{
    if (!(opts.indexOf("location=yes") >= 0)){
        opts = opts + ",location=yes";
    }
  }
  if (opts.indexOf("status=no") >= 0){
      opts.replace("status=no","status=yes");
  }else{
      if (!(opts.indexOf("status=yes") >= 0)){
          opts = opts + ",status=yes";
      }
  }
  invocation.arguments[2] =opts;
}
```

```
var checkLinks = function(){
  var links = document.links;
  if (!links){ policylog('no links'); return; }
  for(var i = 0; i < links.length; i++) {
      if (!AllowedLinks(links[i].href))
         links[i].href ="javascript:"+
                         "alert('disabled link')"
var deniedPolicy = function(invocation){
    debug('Method is disabled');
    return null;
}
enforcePolicy({target:window,method:'alert'},
    function(invocation) {
      deniedPolicy(invocation);
    }
);
enforcePolicy({target:window,method:'prompt'},
   function(invocation) {
     deniedPolicy(invocation);
   }
);
//...
```

```
<head><script>
(function () {
  var orig = frame1.location.assign;
  var wlist = {"msn.com": true};
  frame1.location.assign = function (url) {
      if (wlist[url])
        orig.call(this, url); };
})();</script> ... </head>
```

```
var account = { deposit: function(v){ }, amount: 200 };
var account_control = makeView(account);
var permitGet = function (obj, prop) { return obj[prop]; };
account_control.definePolicy(account,
  {getters: {amount: permitGet, deposit: permitGet}});
var permitCall = function (f, t, a) { return f.apply(t, a); }
account_control.definePolicy(account.deposit,
  {funCall: permitCall});
Bob.send(account_control.view);
```

```
var m = makePolicyView(makeView(document));
var policy = [{"selector": "(//*[@class='example'])
                  | (//*[@class='example']//*)",
   "enabled": true,
   "defaultFieldActions": {read: permit},
   "fields": {shake: {methCall: permit}}}];
m.applyPolicy(policy);
return m.view;
```

```
var x = {y: function () {}, secret: "secret"};
var c = makeView(x);
var permitSet = function (o, p, rhs) { o[p] = rhs; };
c.definePolicy(x, {getters: {y: permitGet},
                   setters: {y: permitSet}}]]);
mallory.send(c.view); // gives Mallory the view of x
```

```
var l = url.lastIndexOf('.',url.length-5) + 1;
if (whitelist[url.substring(l)] === true) { ... }
```

```
var protected =
  __APC.permit('(a.?+b*)', {a:{a:3,b:5},b:{a:7,b:11}});
```

```
<div id="MessageBody"
  policy="read-access: subtree;">
  Message body text here...      </div>
<div id="Advertisement"
  policy="write-access: subtree;"></div>
```

```
<script src="main.js" policy="
  aroundScript(function () { return ''; }); "/>
```

```
let glbl : K = this;
aroundScript(function (src) {
    return glbl == this ? src : ""; });
```

```
let glbl : K = this;
let getScripts : K = document.getElementsByTagName;
let doc : K = document;
let w : {"good.js": K} = {"good.js": true};
aroundScript(function (load : K, src : U) {
    if (this == glbl) {
      let scripts : U = uCall(doc, getScripts, "script");
      return hasProp(w, scripts[scripts.length - 1].src) ?
        load(src) : "";
    } });
```

```
let glbl : K = this;
let getScripts : K = document.getElementsByTagName;
let doc : K = document;
aroundScript(function (load : K, src : U) {
    if (this == glbl) return src;
    else {
      let n : U = this;
      while (n)
        if (n.tagName == "NOINLINESCRIPT") return "";
        else n = n.parentNode;
      return src; } });
```

```
let substr : K = String.prototype.substr;
around((new XMLHttpRequest()).open,
  function (o : K, m : U, u : U, a : U, nm : U, pw: U)
  {
    let name : K = toPrimitive(nm);
    let password : K = toPrimitive(pw);
    let url : K = toPrimitive(u);
    if ((name || password)
        && uCall(url, substr, 0, 8) != "https://") {
      curse(); throw "Use HTTPS for secure a XHR.";
    } else
      return uCall(this, o, m, url, a, name, password);
    });
```

```
let httpOnly : K -> K = function (_ : K) {
    curse(); throw "HTTP-only cookies"; };
around(getField(document, "cookie"), httpOnly);
around(setField(document, "cookie"), httpOnly);
```

| How do I express it | Problems | What type of policy is it |
|---|---|---|
| new policyWithState(0)<br>.allow('create')<br>.from(popup)<br>.condition(  ((input) => {return input <= 2}) )<br>.install();<br>// Allow to create max two pop ups<br><br>NewPolicy<br>.allow(create(url))<br>.condition(url == whiteList)<br>.from(popup)<br>.install() | express location bar , I need to be able to give condition to allowed objects or functions | Statefull access control |
| newPolicy()<br>.deny('create')<br>.from(iframe)<br>Install() | | access control |

;

;

| paper name |
|---|
| Lightweight Self-Protecting JavaScript |
| Lightweight Self-Protecting JavaScript |

Lightweight Self-Protecting JavaScript

Lightweight Self-Protecting JavaScript

Lightweight Self-Protecting JavaScript

Lightweight Self-Protecting JavaScript

Lightweight Self-Protecting JavaScript

Lightweight Self-Protecting JavaScript

Object Views: Fine-Grained Sharing in Browsers

Object Views: Fine-Grained Sharing in Browsers

Object Views: Fine-Grained Sharing in Browsers

Object Views: Fine-Grained Sharing in Browsers

Safe Wrappers and Sane Policies for Self Protecting JavaScript

Efficient Dynamic Access Analysis Using JavaScript Proxies

AdJail: Practical Enforcement of Confidentiality and Integrity Policies onWeb Advertisements