Secure Application Development

The ins and outs of Client-side XSS

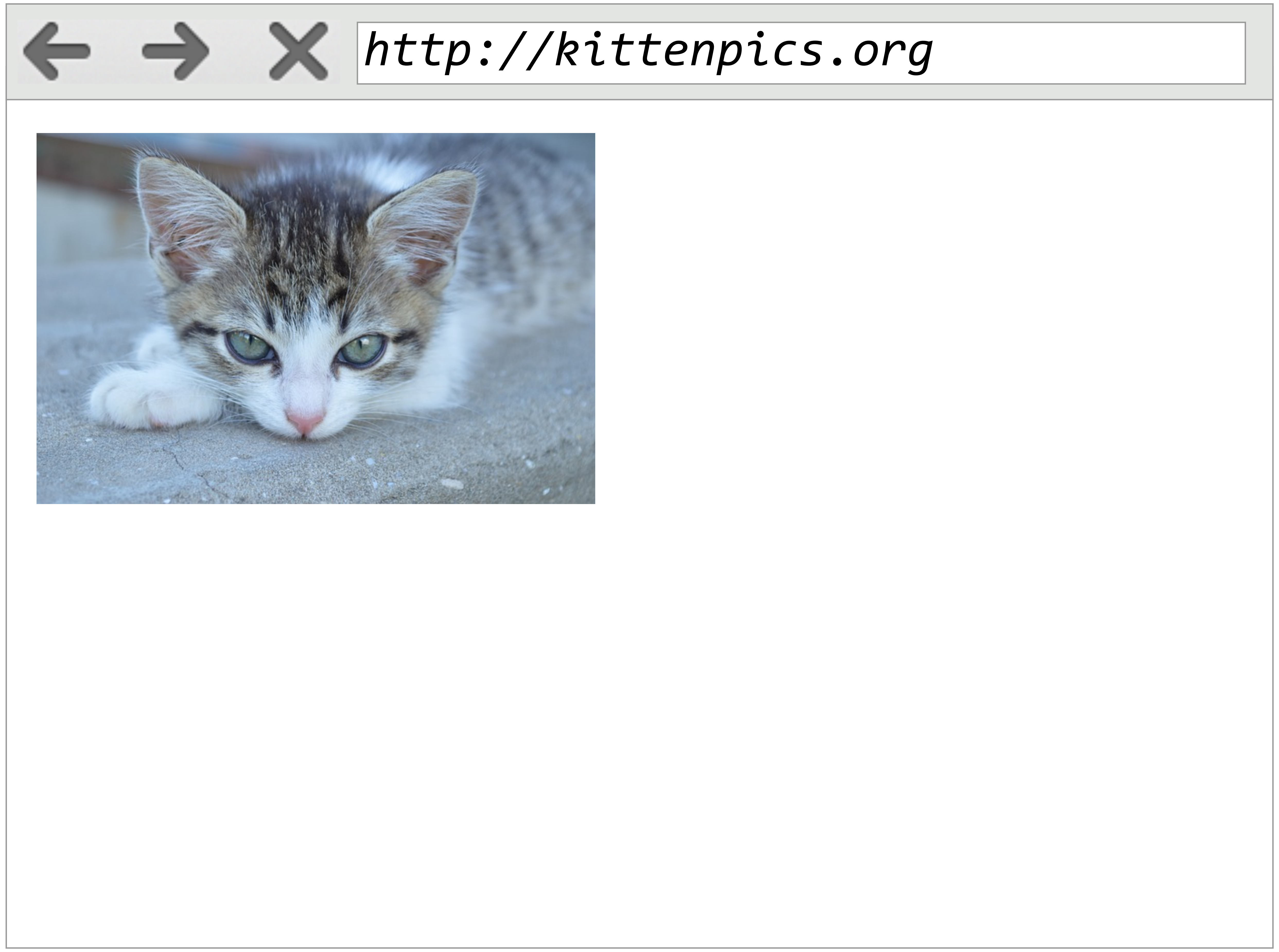
Martin Johns
m.johns@tu-braunschweig.de
SecAppDev 2019

IAS INSTITUTE FOR APPLICATION SECURITY

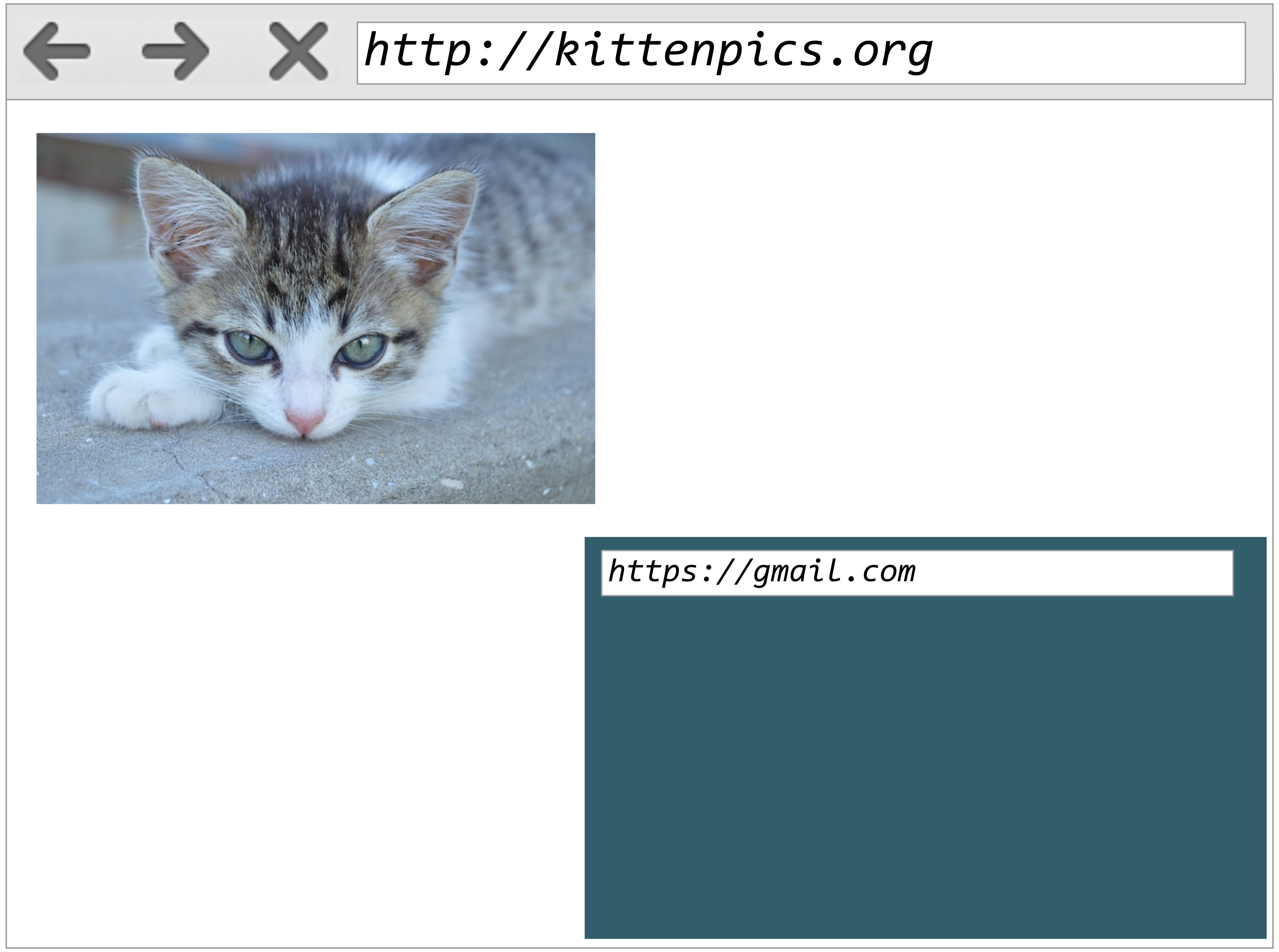
Me, myself and I

- Prof. Dr. Martin Johns
  - TU Braunschweig, Institute for Application Security (IAS)
  - Since April 2018
- Before rejoining the wonderful world of academia (2009 - 2018)
  - 9 years at SAP Security Research, Germany
  - Lead for application and web security research
- PhD on Web Security at University of Passau (2004 - 2009)
- Tons of development jobs during the Web 2.0 times (1998 - 2003)

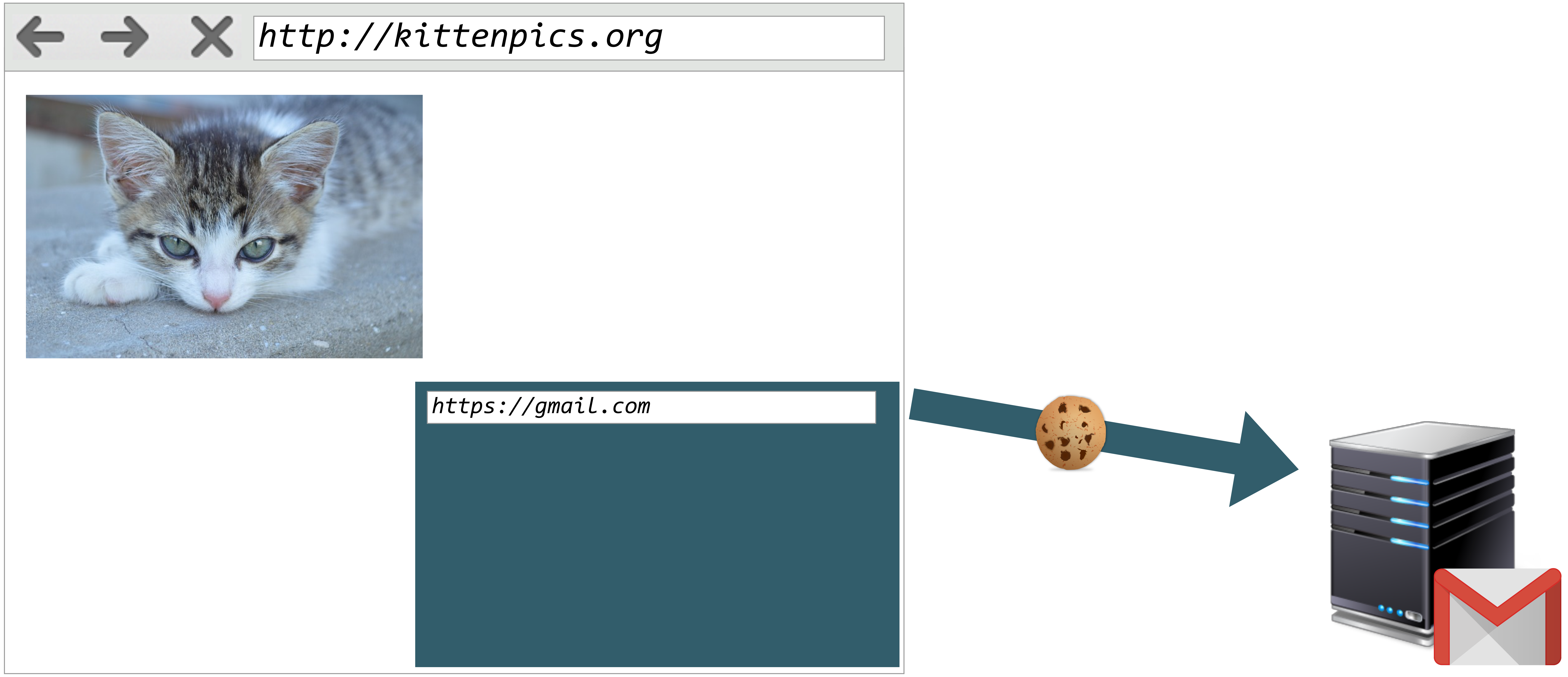
Same-Origin Policy in Action



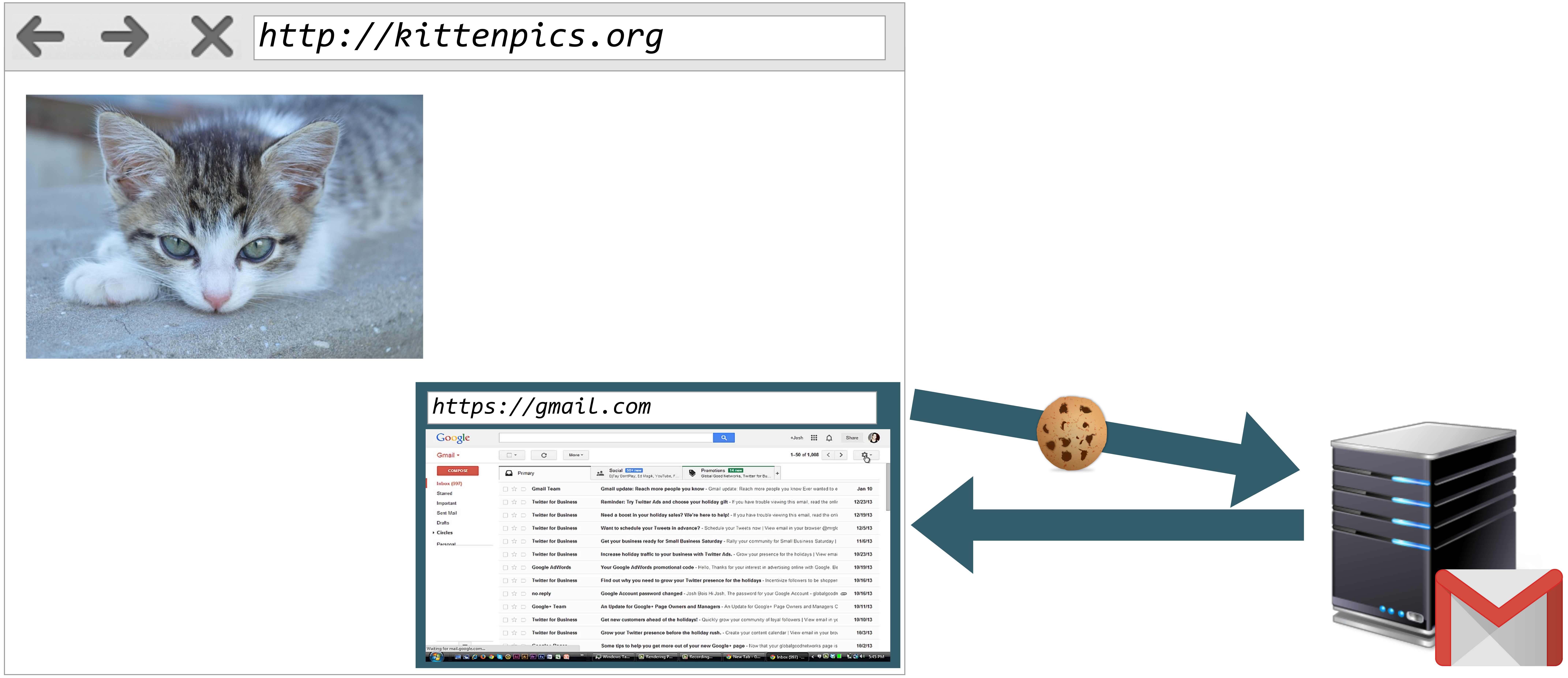
Same-Origin Policy in Action



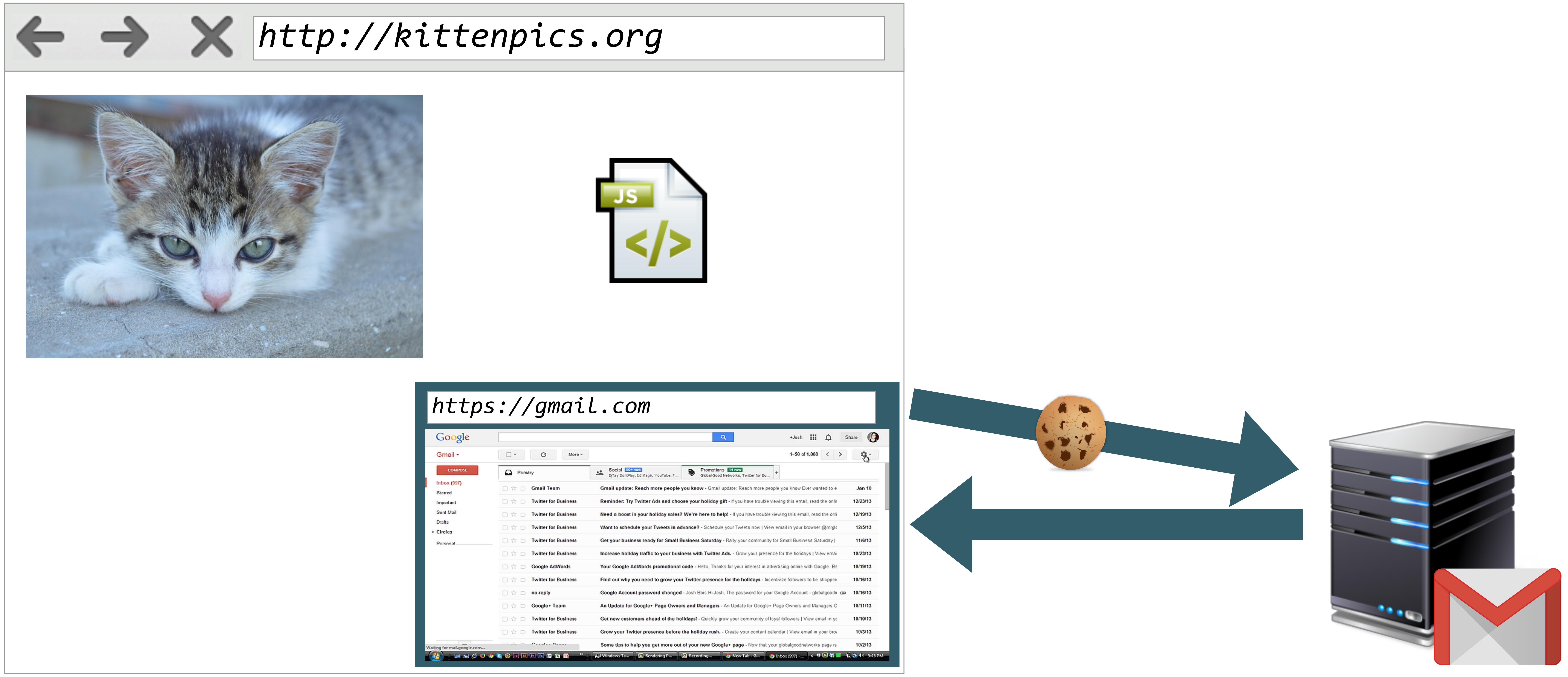
Same-Origin Policy in Action



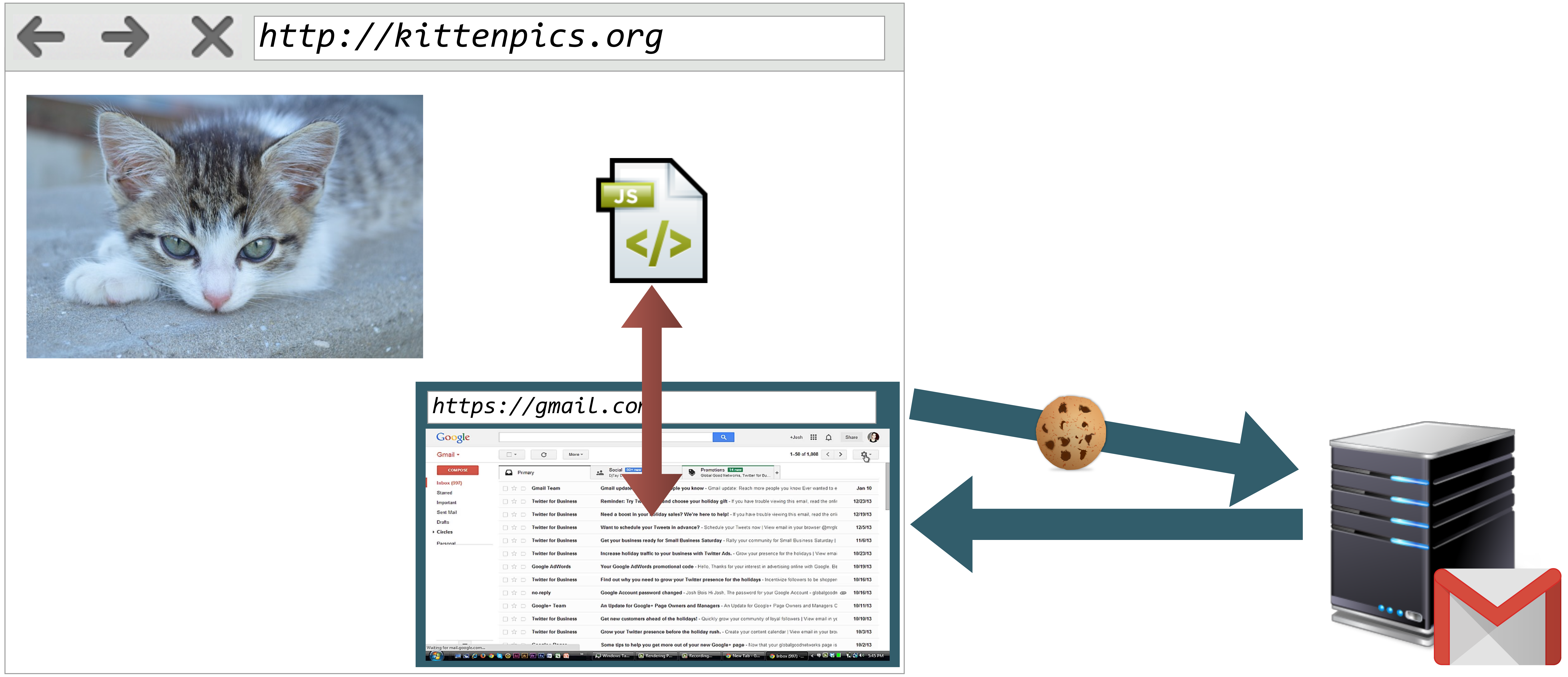
Same-Origin Policy in Action



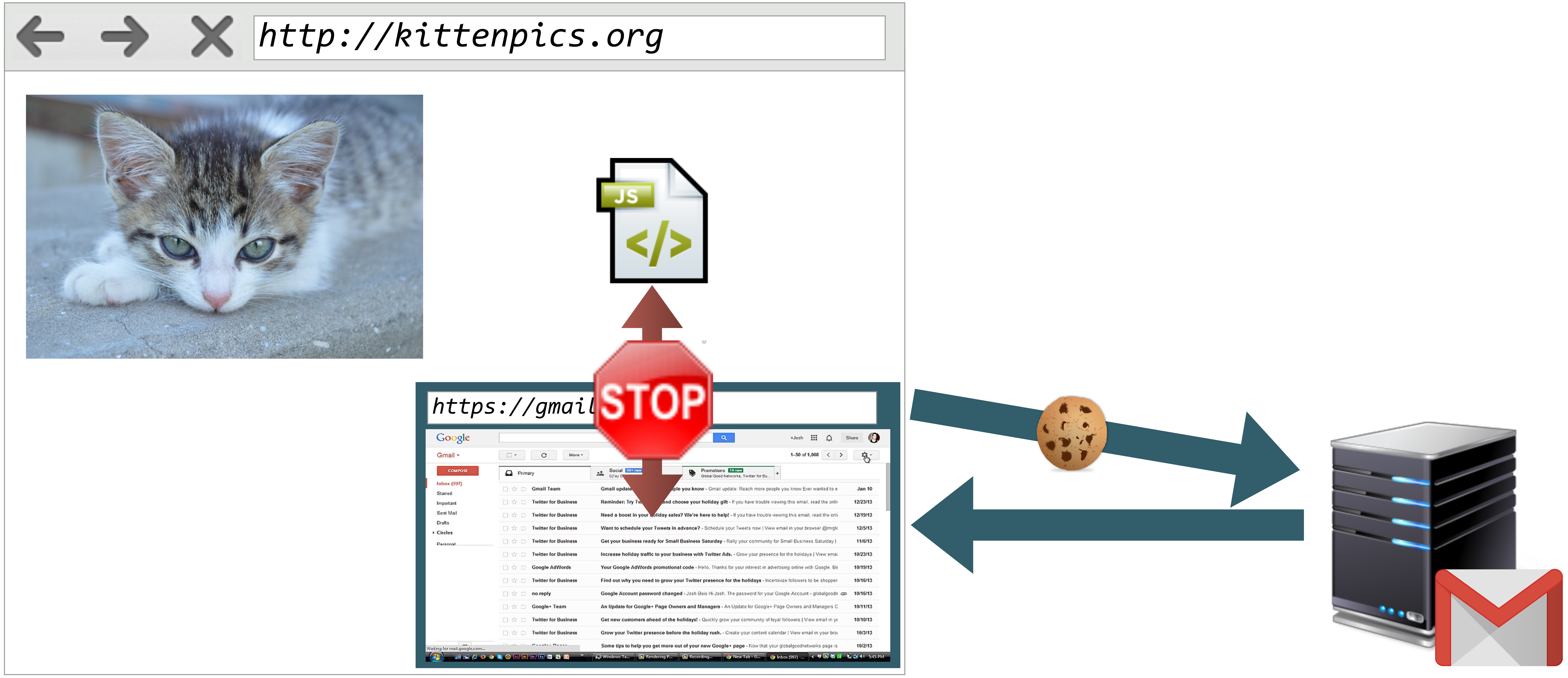
Same-Origin Policy in Action



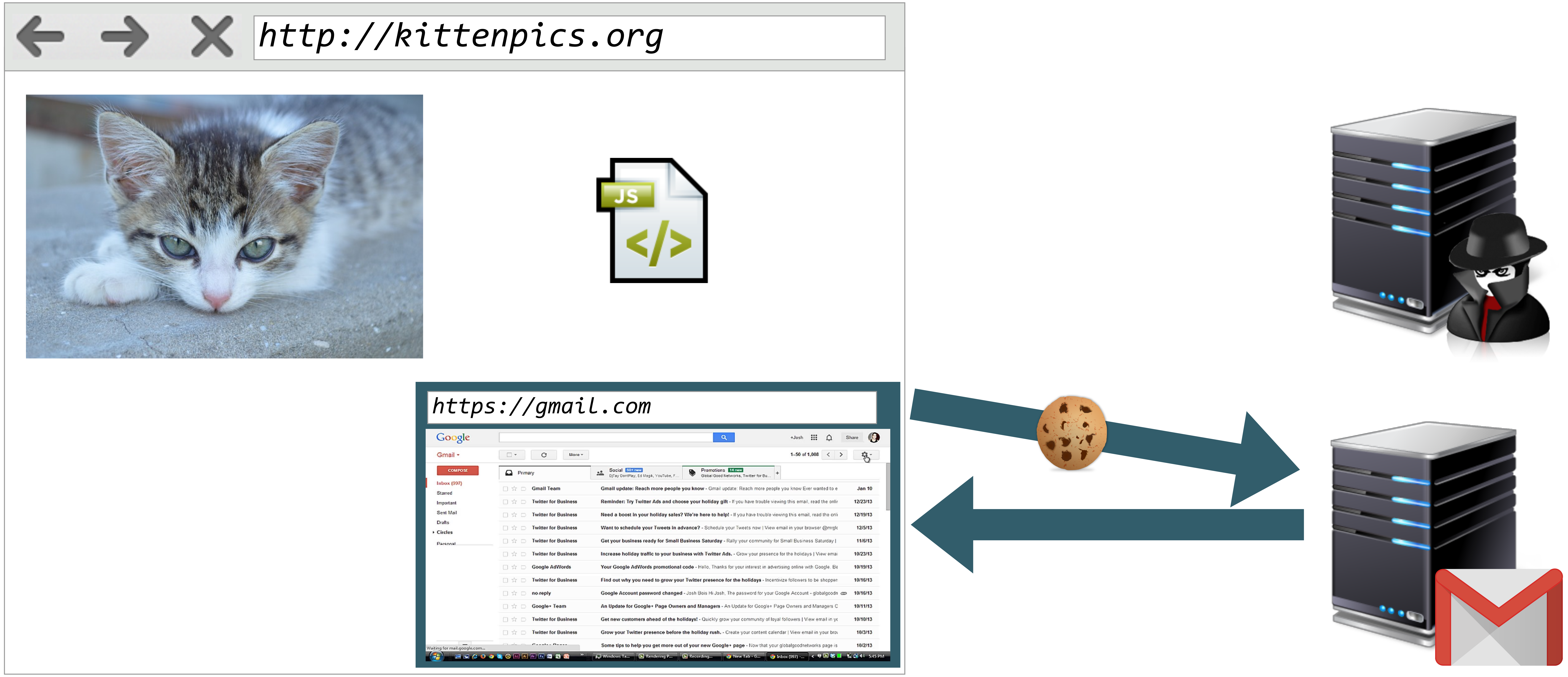
Same-Origin Policy in Action



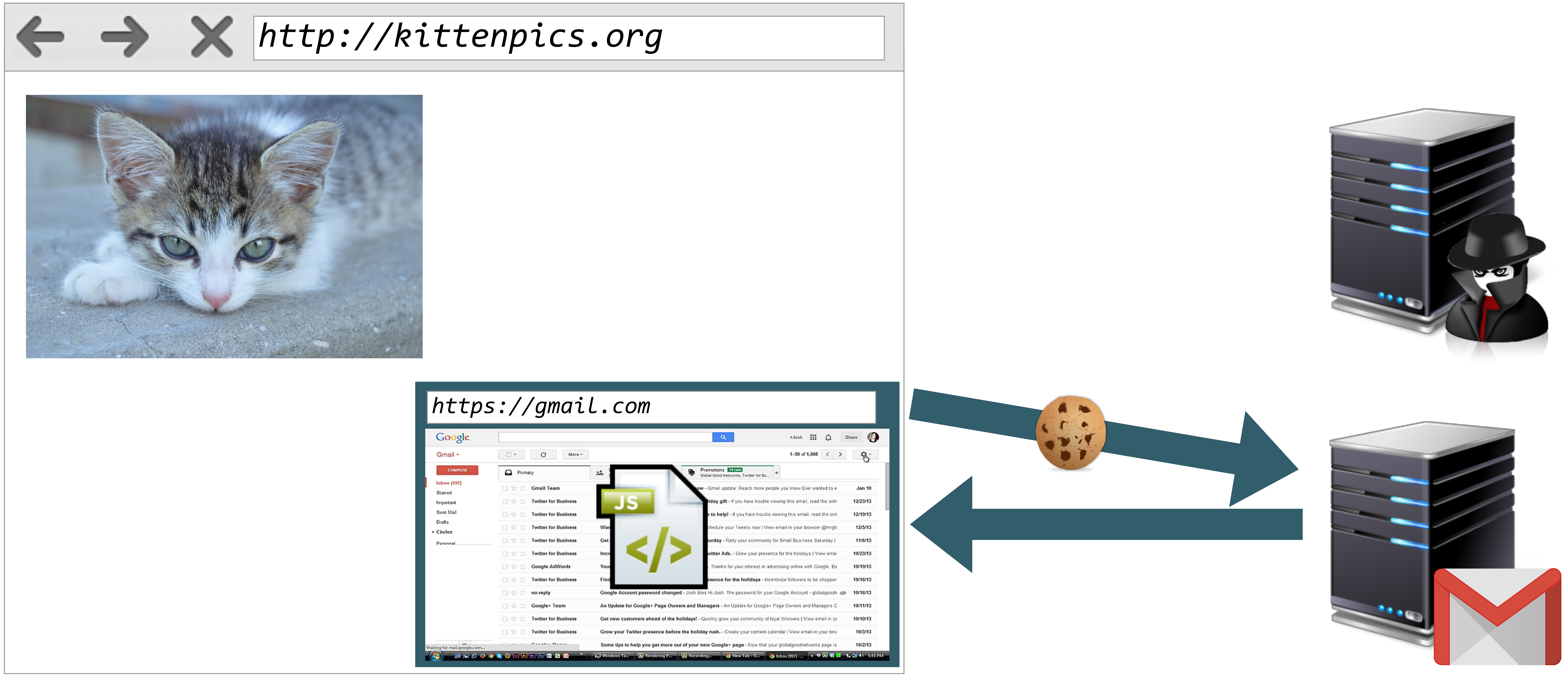
Same-Origin Policy in Action



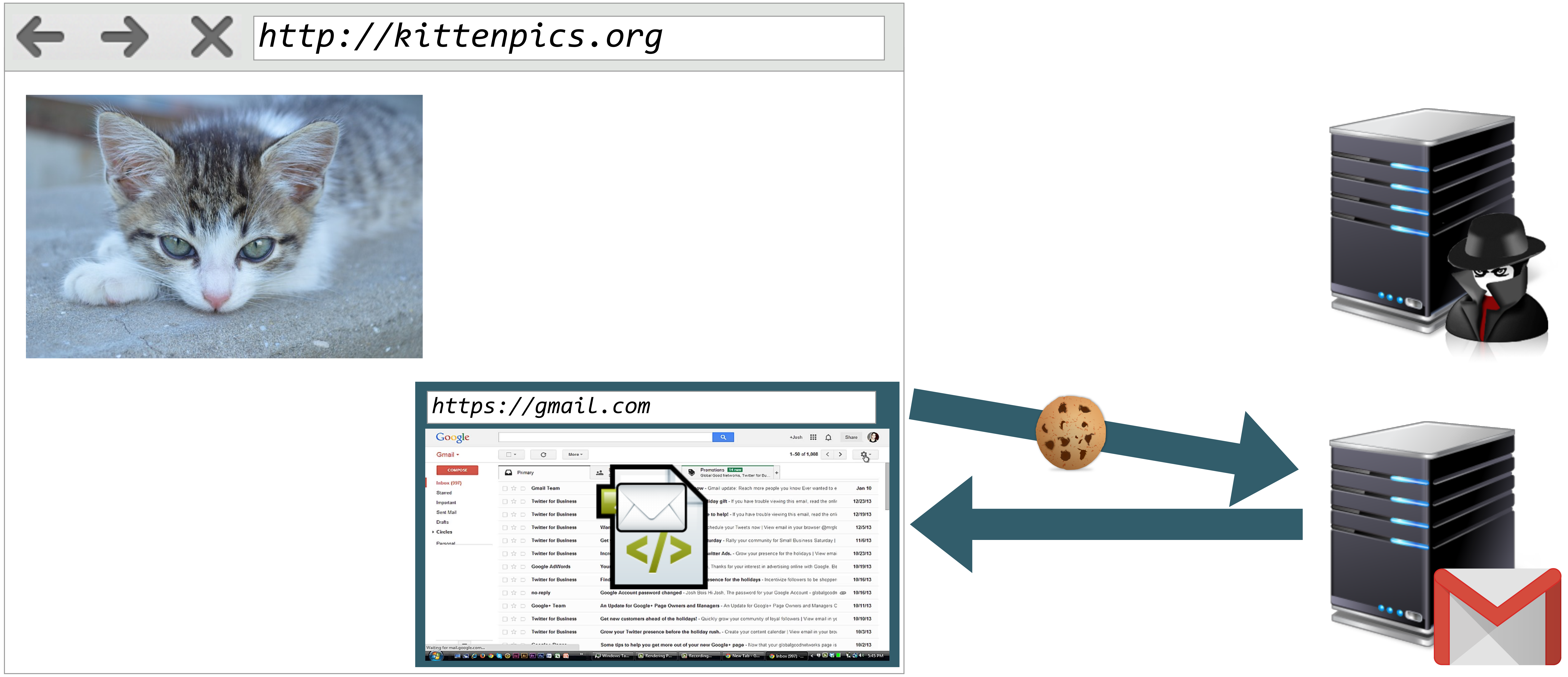
Bypassing the SOP with Code Injection



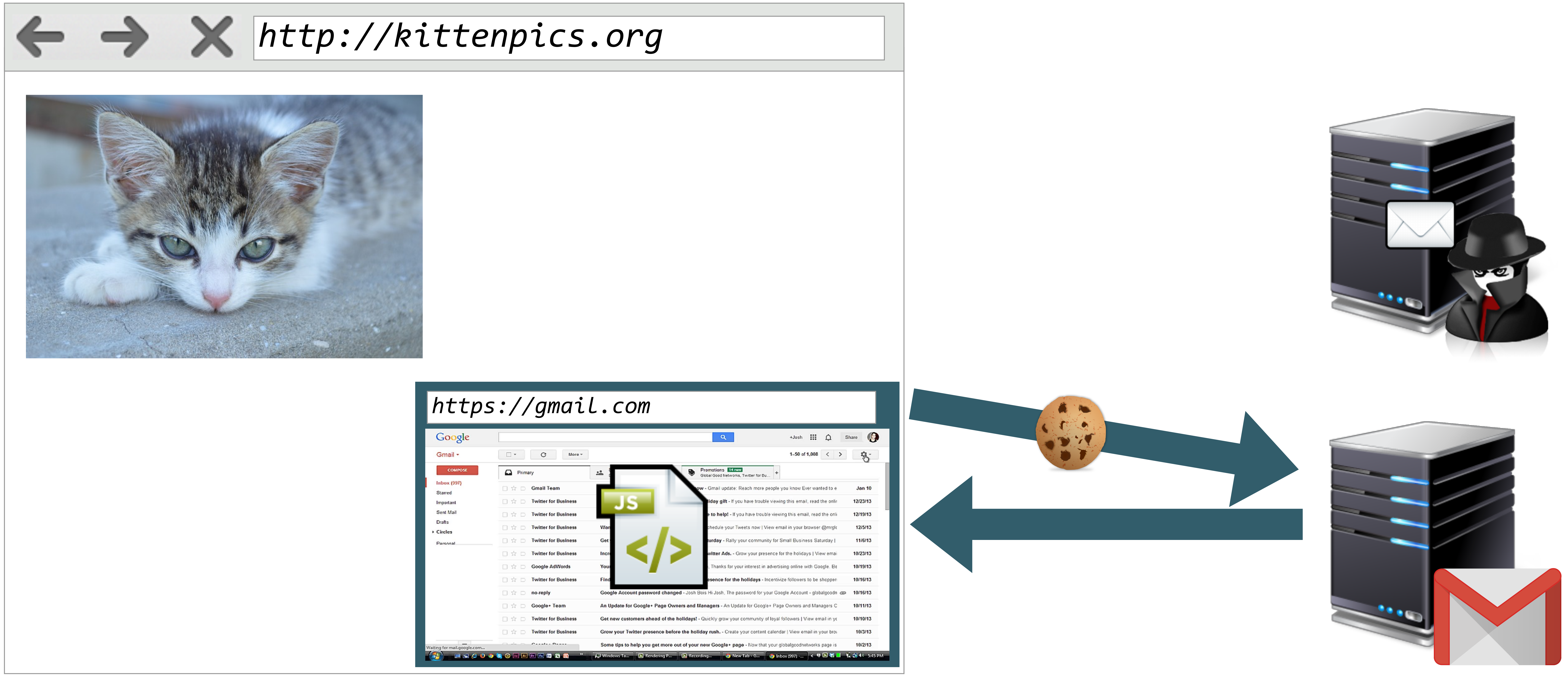
Bypassing the SOP with Code Injection



Bypassing the SOP with Code Injection



Bypassing the SOP with Code Injection



Cross-Site Scripting

- Attacker can inject his own **script** into another site (**cross-site**)
  - actually, might have to inject HTML markup
  - ... which contains JavaScript code
- Injected code runs in origin of vulnerable page
- Has roughly two orthogonal dimensions
  - Location of vulnerable code (server or client)
  - Persistence of attack payload (reflected or persistent)

XSS Examples

XSS Examples

- Tag injection

XSS Examples

- Tag injection

```
Hello <b>$user</b>
```

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="$mypicture">
```

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="foo.jpg" onload="...">
```

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="foo.jpg" onload="...">
```

- JavaScript-URLs

XSS Examples

IAS INSTITUTE FOR APPLICATION SECURITY

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="foo.jpg" onload="...">
```

- JavaScript-URLs

```
<iframe src="$mysite">
```

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="foo.jpg" onload="...">
```

- JavaScript-URLs

```
<iframe src="javascript:...">
```

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="foo.jpg" onload="...">
```

- JavaScript-URLs

```
<iframe src="javascript:...">
```

- In-script injection

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="foo.jpg" onload="...">
```

- JavaScript-URLs

```
<iframe src="javascript:...">
```

- In-script injection

```
<script>
var a = $foo;
</script>
```

XSS Examples

- Tag injection

```
Hello <b><script>...</script></b>
```

- Breaking out of attributes

```
<img src="foo.jpg" onload="...">
```

- JavaScript-URLs

```
<iframe src="javascript:...">
```

- In-script injection

```
<script>
var a = a; evilcode();
</script>
```

Impact of Cross-Site Scripting vulnerabilities

- JavaScript has in the contexts of *its* document powerful capabilities
  - Full control over the documents DOM
  - Full access to the associated browser state
  - Read/write HTTP requests to targets within the SOP

Impact of Cross-Site Scripting vulnerabilities

- JavaScript has in the contexts of *its* document powerful capabilities
  - Full control over the documents DOM
  - Full access to the associated browser state
  - Read/write HTTP requests to targets within the SOP
- This allows JavaScript execution allows attacker to pretend to be
  - ... user towards the server (e.g., posting content in social network)
  - ... server towards the user (e.g., by modifying the look of a page

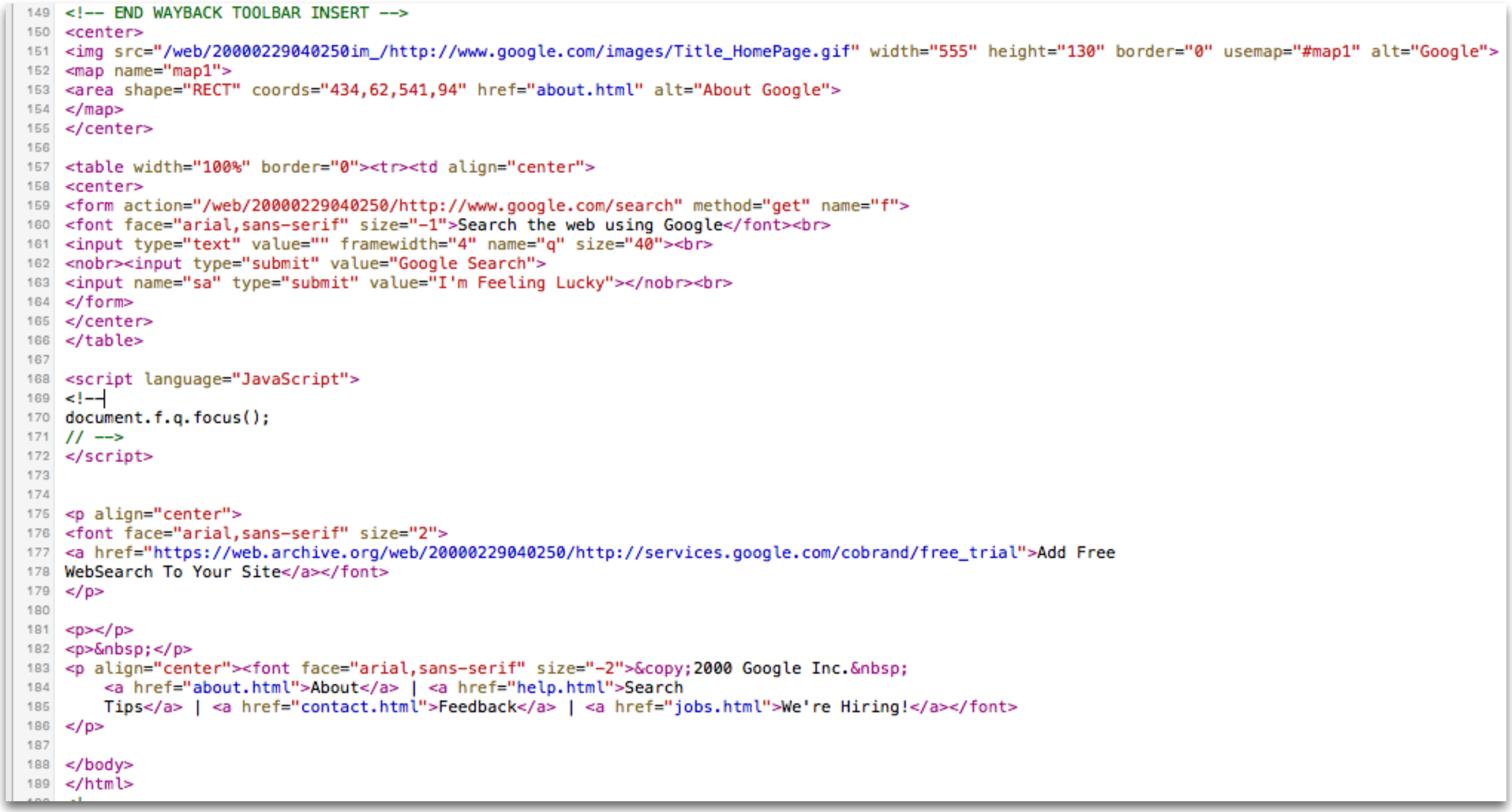
XSS — a server-side problem

- XSS was initially “discovered” in 2000
  - Even though the actual vulnerability is probably as old as dynamic web pages
- In the mid-2000s XSS is high on the radar
  - Both in the academic as well as the practitioner’s communities
  - E.g., thanks to the efforts of OWASP
- However, XSS is perceived to be a security problem caused by server-side code
  - HTML is assembled on the server
  - Attacker data is used insecurely in this process
  - The attacker is able to insert his own markup / JavaScript
- Thus, all proposed defences focused on server-side code

The rise of JavaScript

Google.com in the year 2000

IAS INSTITUTE FOR APPLICATION SECURITY



```html
<!-- END WAYBACK TOOLBAR INSERT -->
<center>
<img src="/web/20000229040250im_/http://www.google.com/images/Title_HomePage.gif" width="555" height="130" border="0" usemap="#map1" alt="Google">
<map name="map1">
<area shape="RECT" coords="434,62,541,94" href="about.html" alt="About Google">
</map>
</center>

<table width="100%" border="0"><tr><td align="center">
<center>
<form action="/web/20000229040250/http://www.google.com/search" method="get" name="f">
<font face="arial,sans-serif" size="-1">Search the web using Google</font><br>
<input type="text" value="" framewidth="4" name="q" size="40"><br>
<nobr><input type="submit" value="Google Search">
<input name="sa" type="submit" value="I'm Feeling Lucky"></nobr><br>
</form>
</center>
</table>

<script language="JavaScript">
<!--
document.f.q.focus();
// -->
</script>


<p align="center">
<font face="arial,sans-serif" size="2">
<a href="https://web.archive.org/web/20000229040250/http://services.google.com/cobrand/free_trial">Add Free
WebSearch To Your Site</a></font>
</p>

<p></p>
<p>&nbsp;</p>
<p align="center"><font face="arial,sans-serif" size="-2">&copy;2000 Google Inc.&nbsp;
    <a href="about.html">About</a> | <a href="help.html">Search
    Tips</a> | <a href="contact.html">Feedback</a> | <a href="jobs.html">We're Hiring!</a></font>
</p>

</body>
</html>
```

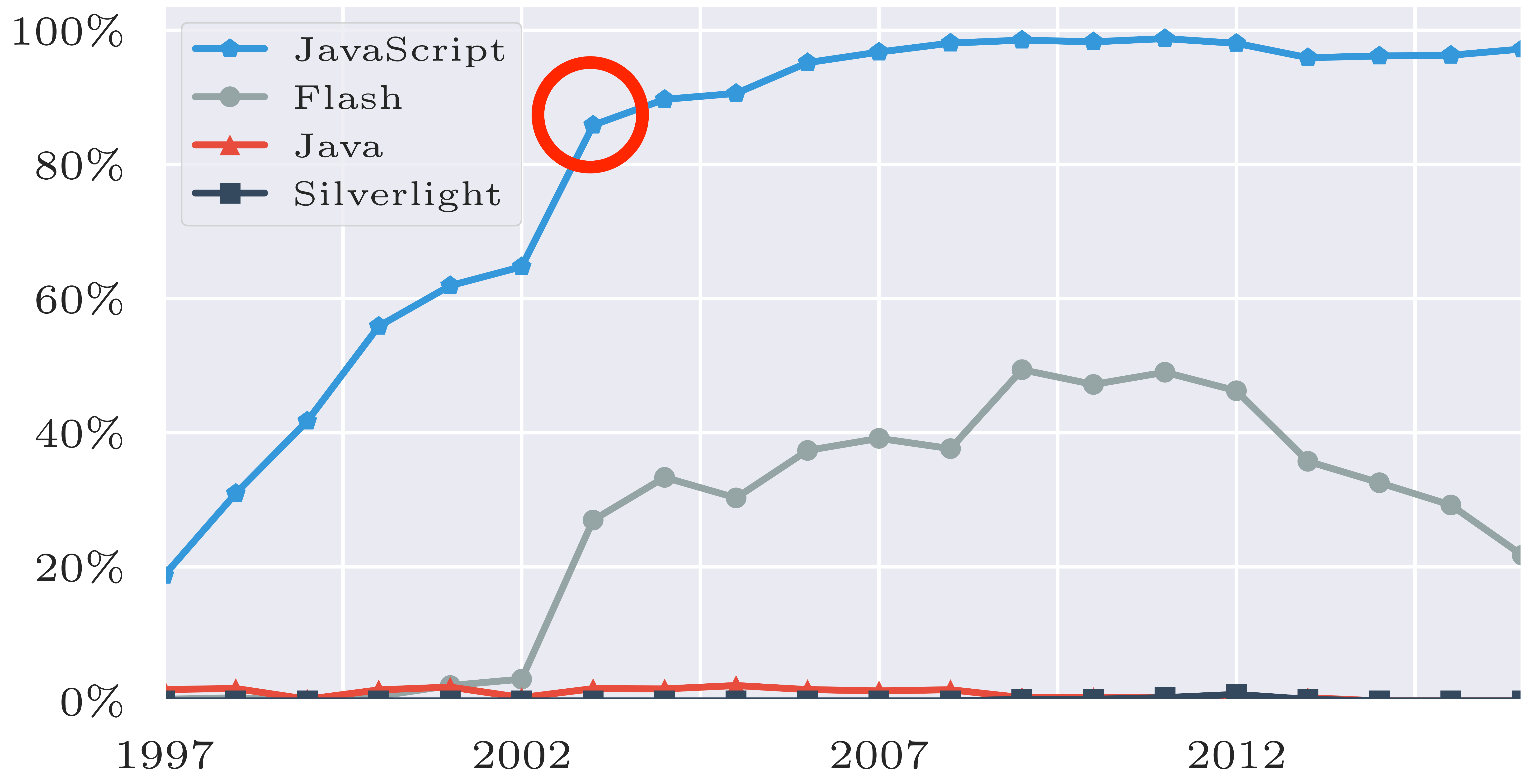
Google.com in the year 2019

```
"clickmod";else{var f=b.which||b.keyCode||b.key;D&&3==f&&(f=13);if(13!=f&&32!=f)f=!1;else{var m=B(b),l=(m.getAttribute("role")||m.type||m.tagName).toUpperCase(),g;(g="keydown"!=b.type)||("getAttribute"in m?(g=(m.getAttribute("r
da&&"A"!=n&&!G(l)&&!H(l)||"BUTTON"==n);l&&(b.preventDefault?b.preventDefault():b.returnValue=!1);if("mouseenter"==c||"mouseleave"==c)if(l=b.relatedTarget,!("mouseover"==b.type&&"mouseenter"==c||"mouseout"==b.type&&"mouseleave"=
"",f.actionElement=null;e=f;a.i&&(q=R(e.eventType,e.event,e.targetElement,e.action,e.actionElement,e.timeStamp),"clickonly"==q.eventType&&(q.eventType="click"),a.i(q,!0));if(e.actionElement){"A"!=e.actionElement.tagName||"click
e.event.type&&e.event._mouseEventsPrevented){b=e.event;for(var oa in b);k()}}}},R=function(a,d,b,c,e,f){return{eventType:a,event:d,targetElement:b,action:c,actionElement:e,timeStamp:f||k()}},ja={},la=function(a,d){return functi
c,e));return{o:c,m:e,capture:f}}},O=function(a,d){if(!a.l.hasOwnProperty(d)){var b=ka(a,d),c=la(d,b);a.l[d]=b;a.s.push(c);for(b=0;b<a.g.length;++b){var e=a.g[b];e.h.push(c.call(null,e.g))}"click"==d&&O(a,"keydown")}};P.prototyp
(0<a.j.length&&d(a.j),a.j=null)},ma=function(a){this.g=a;this.h=[]},S=function(a,d){for(var b=a.g,c=d;b!=c&&c.parentNode;)c=c.parentNode;return b==c},W=function(a,d){for(var b=0;b<d.length;++b)if(d[b].g!=a.g&&S(d[b],a.g))return
try{
var aa,da,ja,ka,la,ya,za;aa=function(a){var c=0;return function(){return c<a.length?{done:!1,value:a[c++]}:{done:!0}}};_.ba=function(a){var c="undefined"!=typeof Symbol&&Symbol.iterator&&a[Symbol.iterator];return c?c.call(a):{r
if("function"==typeof Object.setPrototypeOf)da=Object.setPrototypeOf;else{var ea;a:{var fa={Og:!0},ha={};try{ha.__proto__=fa;ea=ha.Og;break a}catch(a){}ea=!1}da=ea?function(a,c){a.__proto__=c;if(a.__proto__!==c)throw new TypeEr
ka="undefined"!=typeof window&&window===this?this:"undefined"!=typeof global&&null!=global?global:this;la=function(a,c){if(c){var d=ka;a=a.split(".");for(var e=0;e<a.length-1;e++){var f=a[e];f in d||(d[f]={});d=d[f]}a=a[a.lengt
la("String.prototype.startsWith",function(a){return a?a:function(a,d){if(null==this)throw new TypeError("b`startsWith");if(a instanceof RegExp)throw new TypeError("c`startsWith");var c=this.length,f=a.length;d=Math.max(0,Math.m
var ma=function(){ma=function(){};ka.Symbol||(ka.Symbol=na)},na=function(){var a=0;return function(c){return"jscomp_symbol_"+(c||"")+a++}}(),pa=function(){ma();var a=ka.Symbol.iterator;a||(a=ka.Symbol.iterator=ka.Symbol("iterat
c)};
la("WeakMap",function(a){function c(){}function d(a){qa(a,f)||ja(a,f,{value:new c})}function e(a){var e=Object[a];e&&(Object[a]=function(a){if(a instanceof c)return a;d(a);return e(a)})}if(function(){if(!a||!Object.seal)return!
0,h=function(a){this.da=(g+=Math.random()+1).toString();if(a){a=_.ba(a);for(var c;!(c=a.next()).done;)c=c.value,this.set(c[0],c[1])}};h.prototype.set=function(a,c){d(a);if(!qa(a,f))throw Error("d`"+a);a[f][this.da]=c;return thi
la("Map",function(a){if(function(){if(!a||"function"!=typeof a||!a.prototype.entries||"function"!=typeof Object.seal)return!1;try{var c=Object.seal({x:4}),d=new a(_.ba([[c,"s"]]));if("s"!=d.get(c)||1!=d.size||d.get({x:4})||d.se
this.b=g();this.size=0;if(a){a=_.ba(a);for(var c;!(c=a.next()).done;)c=c.value,this.set(c[0],c[1])}};d.prototype.set=function(a,c){a=0===a?0:a;var d=e(this,a);d.list||(d.list=this.o[d.id]=[]);d.Qa?d.Qa.value=c:(d.Qa={next:this.
a.Qa.Ib,a.Qa.head=null,this.size--,!0):!1};d.prototype.clear=function(){this.o={};this.b=this.b.Ib=g();this.size=0};d.prototype.has=function(a){return!!e(this,a).Qa};d.prototype.get=function(a){return(a=e(this,a).Qa)&&a.value};
e;!(e=d.next()).done;)e=e.value,a.call(c,e[1],e[0],this)};d.prototype[Symbol.iterator]=d.prototype.entries;var e=function(a,d){var e=d&&typeof d;"object"==e||"function"==e?c.has(d)?e=c.get(d):(e=""+ ++h,c.set(d,e)):e="p_"+d;var
d.next,{done:!1,value:c(d)};d=null}return{done:!0,value:void 0}})},g=function(){var a={};return a.Ib=a.next=a.head=a},h=0;return d});
la("Set",function(a){if(function(){if(!a||"function"!=typeof a||!a.prototype.entries||"function"!=typeof Object.seal)return!1;try{var c=Object.seal({x:4}),e=new a(_.ba([c]));if(!e.has(c)||1!=e.size||e.add(c)!=e||1!=e.size||e.ac
if(a){a=_.ba(a);for(var c;!(c=a.next()).done;)this.add(c.value)}this.size=this.b.size};c.prototype.add=function(a){a=0===a?0:a;this.b.set(a,a);this.size=this.b.size;return this};c.prototype["delete"]=function(a){a=this.b["delet
c.prototype[Symbol.iterator]=c.prototype.values;c.prototype.forEach=function(a,c){var d=this;this.b.forEach(function(e){return a.call(c,e,e,d)})};return c});la("Object.entries",function(a){return a?a:function(a){var c=[],e;for(
la("Array.from",function(a){return a?a:function(a,d,e){d=null!=d?d:function(a){return a};var c=[],g="undefined"!=typeof Symbol&&Symbol.iterator&&a[Symbol.iterator];if("function"==typeof g){a=g.call(a);for(var h=0;!(g=a.next()).
_.p=function(a){return"string"==typeof a};_.sa=function(a){return"number"==typeof a};_.t=function(a,c){a=a.split(".");c=c||_.m;for(var d=0;d<a.length;d++)if(c=c[a[d]],null==c)return null;return c};_.ta=function(){};_.ua=functi
_.va=function(a){var c=typeof a;if("object"==c)if(a){if(a instanceof Array)return"array";if(a instanceof Object)return c;var d=Object.prototype.toString.call(a);if("[object Window]"==d)return"object";if("[object Array]"==d||"nu
else if("function"==c&&"undefined"==typeof a.call)return"object";return c};_.wa=function(a){return"array"==_.va(a)};_.xa="closure_uid_"+(1E9*Math.random()>>>0);ya=function(a,c,d){return a.call.apply(a.bind,arguments)};za=functi
_.u=function(a,c,d){Function.prototype.bind&&-1!=Function.prototype.bind.toString().indexOf("native code")?_.u=ya:_.u=za;return _.u.apply(null,arguments)};_.v=Date.now||function(){return+new Date};_.w=function(a,c){a=a.split(".
_.x=function(a,c){function d(){}d.prototype=c.prototype;a.J=c.prototype;a.prototype=new d;a.prototype.constructor=a;a.Ok=function(a,d,g){for(var e=Array(arguments.length-2),f=2;f<arguments.length;f++)e[f-2]=arguments[f];return
var Aa=function(a,c,d){this.A=a;this.o=!1;this.b=c;this.w=d};Aa.prototype.Pa=function(a){if(this.o)throw Error("e`"+this.b);try{a.apply(this.A,this.w),this.o=!0}catch(c){}};_.z=function(){this.ia=this.ia;this.ab=this.ab};_.z.pr
Ba.prototype.Pa=function(){for(var a=this.b.length,c=this.b,d=[],e=0;e<a;++e){var f=c[e].b;a:{var g=this.w;for(var h=f.split("."),l=h.length,q=0;q<l;++q)if(g[h[q]])g=g[h[q]];else{g=null;break a}g=g instanceof Function?g:null}if
_.Ca=function(a){if(Error.captureStackTrace)Error.captureStackTrace(this,_.Ca);else{var c=Error().stack;c&&(this.stack=c)}a&&(this.message=String(a))};_.x(_.Ca,Error);_.Ca.prototype.name="CustomError";_.Da=Array.prototype.index
_.Fa=Array.prototype.filter?function(a,c,d){return Array.prototype.filter.call(a,c,d)}:function(a,c,d){for(var e=a.length,f=[],g=0,h=_.p(a)?a.split(""):a,l=0;l<e;l++)if(l in h){var q=h[l];c.call(d,q,l,a)&&(f[g++]=q)}return f};_
_.Ha=Array.prototype.reduce?function(a,c,d){return Array.prototype.reduce.call(a,c,d)}:function(a,c,d){var e=d;(0,_.Ea)(a,function(d,g){e=c.call(void 0,e,d,g,a)});return e};_.Ia=Array.prototype.some?function(a,c){return Array.p
var Pa;_.Ja=function(){this.b={};this.o={}};_.ua(_.Ja);_.La=function(a,c){a.V=function(){return _.Ka(_.Ja.V(),c)};a.Hj=function(){return _.Ja.V().b[c]||null}};_.Na=function(a,c){var d=_.Ja.V();if(a in d.b){if(d.b[a]!=c)throw ne
var Sa;_.Qa="bbh bbr bbs has prm sngw so".split(" ");Sa=new Ba(_.m);_.Na("api",Sa);
for(var Ta="addExtraLink addLink aomc asmc close cp.c cp.l cp.me cp.ml cp.rc cp.rel ela elc elh gpca gpcr lGC lPWF ldb mls noam paa pc pca pcm pw.clk pw.hvr qfaae qfaas qfaau qfae qfas qfau qfhi qm qs qsi rtl sa setContinueCb s
var Ya;_.Xa=String.prototype.trim?function(a){return a.trim()}:function(a){return/^[\s\xa0]*([\s\S]*?)[\s\xa0]*$/.exec(a)[1]};
_.Za=function(a,c){var d=0;a=(0,_.Xa)(String(a)).split(".");c=(0,_.Xa)(String(c)).split(".");for(var e=Math.max(a.length,c.length),f=0;0==d&&f<e;f++){var g=a[f]||"",h=c[f]||"";do{g=/(\d*)(\D*)(.*)/.exec(g)||["","","",""];h=/(\
a:{var ab=_.m.navigator;if(ab){var bb=ab.userAgent;if(bb){_.$a=bb;break a}}_.$a=""}_.A=function(a){return-1!=_.$a.indexOf(a)};var eb;_.cb=function(){return _.A("Trident")||_.A("MSIE")};_.db=function(){return _.A("Firefox")||_.A
Ab=function(){var a=_.m.document;return a?a.documentMode:void 0};a:{var Cb="",Db=function(){var a=_.$a;if(_.nb)return/rv:([^\);]+)(\)|;)/.exec(a);if(_.lb)return/Edge\/([\d\.]+)/.exec(a);if(_.B)return/\b(?:MSIE|rv)[: ]([^\);]+)(
_.Lb=_.db();_.Mb=fb()||_.A("iPod");_.Nb=_.A("iPad");_.Ob=_.A("Android")&&!(eb()||_.db()||_.A("Opera")||_.A("Silk"));_.Qb=eb();_.Rb=_.A("Safari")&&!(eb()||_.A("Coast")||_.A("Opera")||_.A("Edge")||_.db()||_.A("Silk")||_.A("Androi
_.E=function(a,c,d,e,f,g){a.b=null;c||(c=d?[d]:[]);a.ia=d?String(d):void 0;a.B=0===d?-1:0;a.w=c;a:{d=a.w.length;c=-1;if(d&&(c=d-1,d=a.w[c],!(null===d||"object"!=typeof d||_.wa(d)||Ub&&d instanceof Uint8Array))){a.C=c-a.B;a.A=d;
Wb=function(a){var c=a.C+a.B;a.w[c]||(a.A=a.w[c]={})};_.F=function(a,c){if(c<a.C){c+=a.B;var d=a.w[c];return d===_.Vb?a.w[c]=[]:d}if(a.A)return d=a.A[c],d===_.Vb?a.A[c]=[]:d};_.G=function(a,c,d){a=_.F(a,c);return null==a?d:a};_
Yb=function(a){if(a.b)for(var c in a.b){var d=a.b[c];if(_.wa(d))for(var e=0;e<d.length;e++)d[e]&&d[e].Oa();else d&&d.Oa()}};_.C.prototype.Oa=function(){Yb(this);return this.w};
```

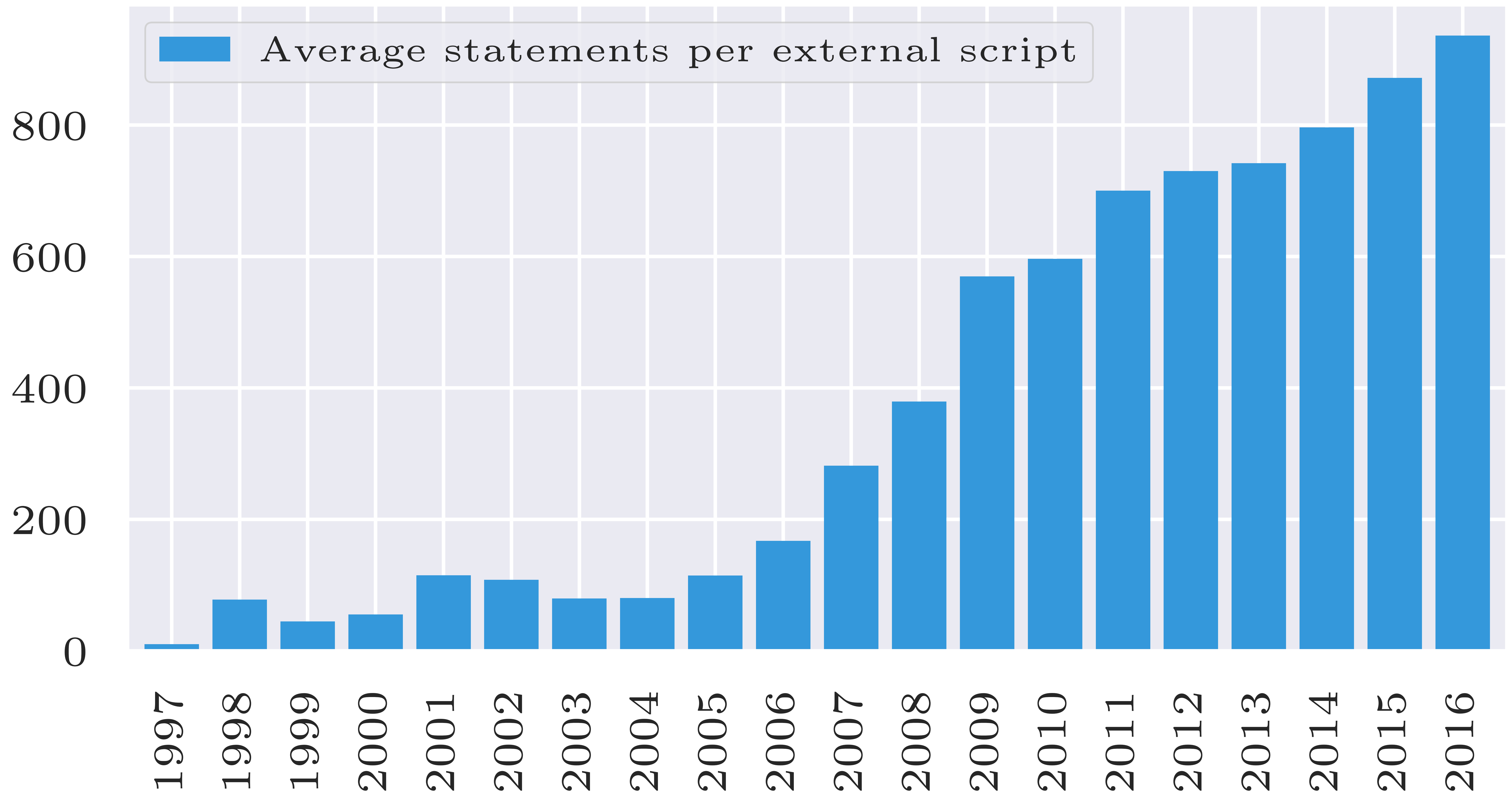
- …this goes on for another 290 lines of code

INSTITUTE FOR APPLICATION SECURITY

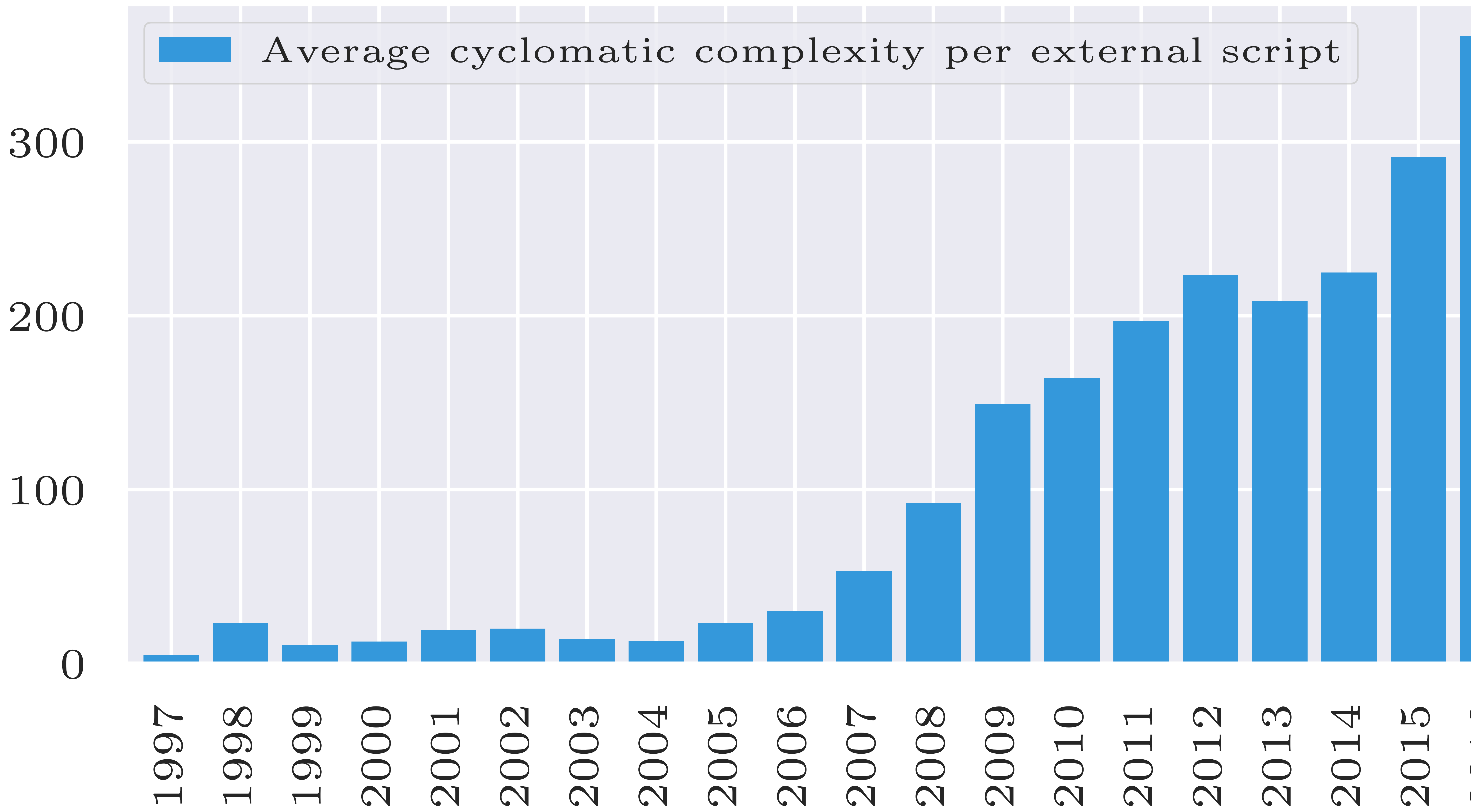
Technologies used by the top 500 sites



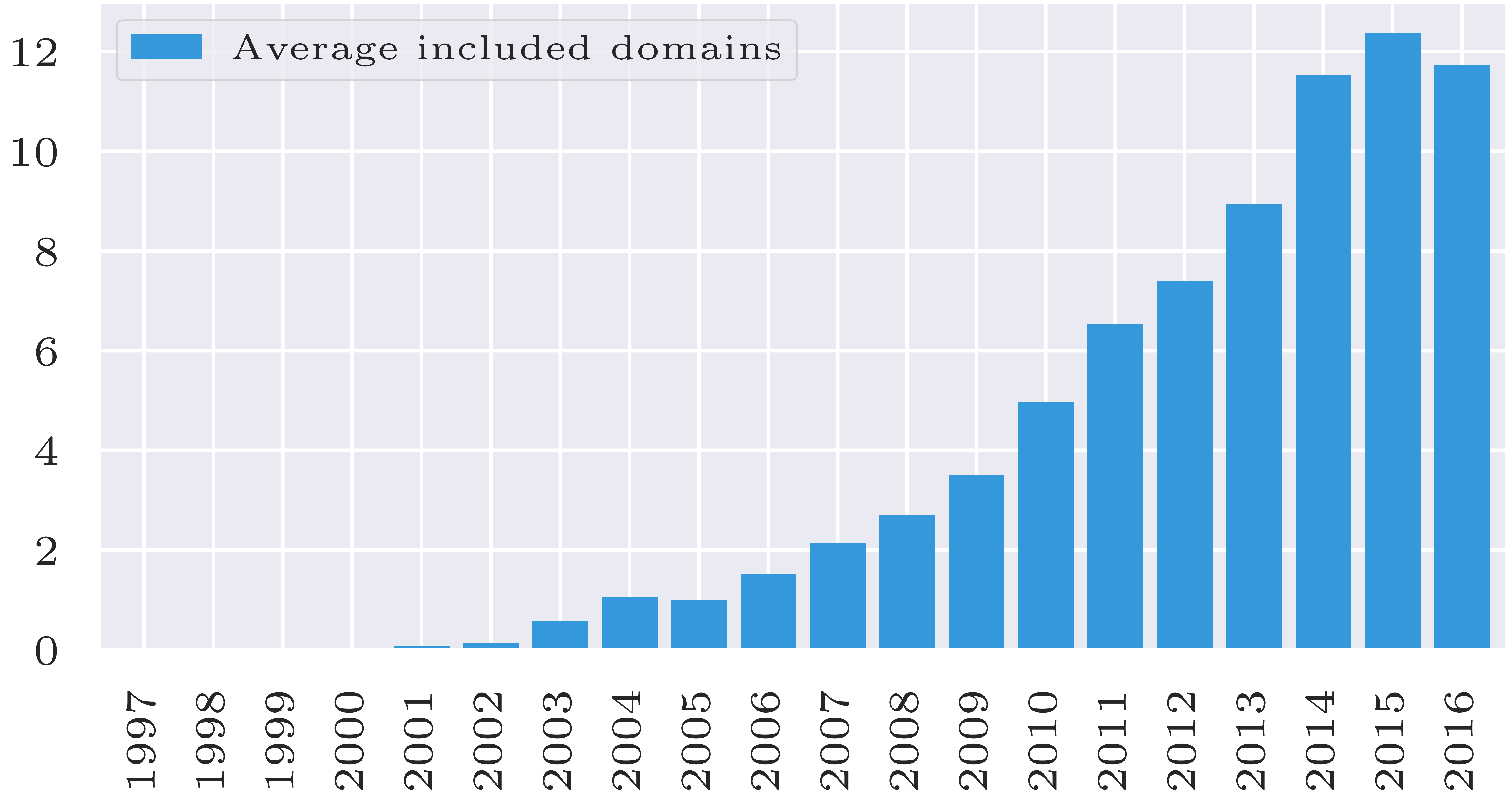
JavaScript code size on the rise



JavaScript code complexity on the rise



Multiple parties contribute JavaScript code



Client-side XSS

A (mostly) overlooked facet of XSS

A (mostly) overlooked facet of XSS

- We push more and more JavaScript code to the browser

A (mostly) overlooked facet of XSS

- We push more and more JavaScript code to the browser
- This JavaScript can
  - create new HTML code
  - introduce new DOM elements
  - convert strings into JavaScript code

A (mostly) overlooked facet of XSS

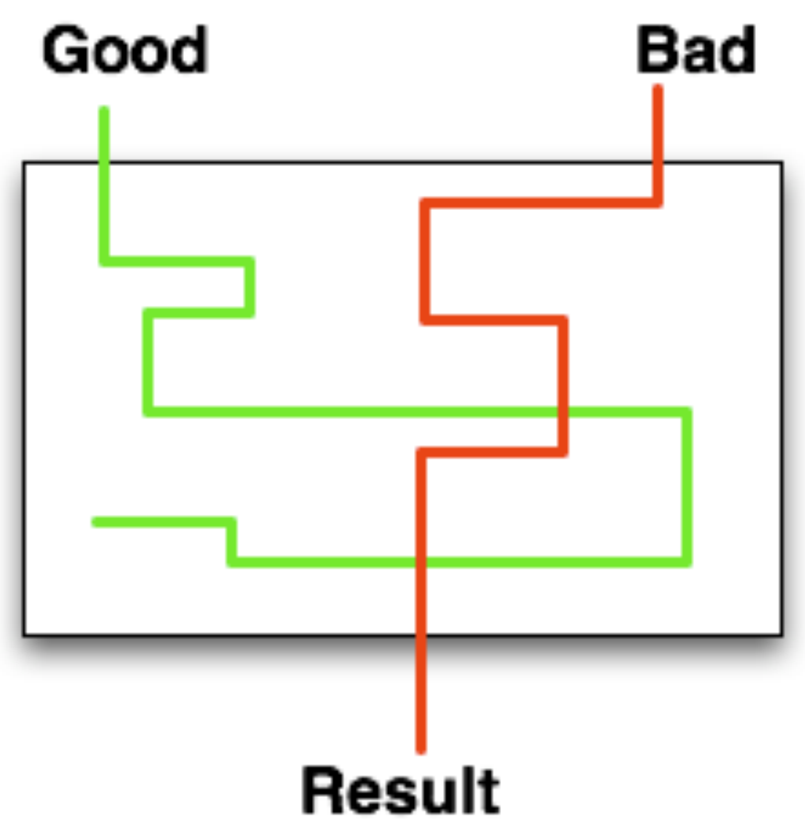
- We push more and more JavaScript code to the browser
- This JavaScript can
  - create new HTML code
  - introduce new DOM elements
  - convert strings into JavaScript code
- Thus, has all means necessary to create XSS problems

A (mostly) overlooked facet of XSS

- We push more and more JavaScript code to the browser
- This JavaScript can
  - create new HTML code
  - introduce new DOM elements
  - convert strings into JavaScript code
- Thus, has all means necessary to create XSS problems
- This problem was initially discussed by Amit Klein in 2005
  - “DOM Based Cross Site Scripting or XSS of the Third Kind”
  - …however only little attention was paid to this vulnerability class

How do XSS problems occur?

- XSS is always rooted in an insecure data flow
  - The adversary’s attack payload enters the application as a string through a **source**
  - It traverses the application without proper sanitization
  - It ends up in a **sink API**, which transforms the attacker’s string into computer code
    - i.e., into HTML or JavaScript



- So, what are the sources and sinks of client-side XSS?

Sinks for Client-Side Cross-Site Scripting

- *`document.write, document.writeln`*
  - Can write new script tags which will be executed
- *`eval, setTimeout, setInterval`*
  - Directly executes JavaScript code
- *`innerHTML, outerHTML`*
  - will not execute script elements, but event handlers work
  - *`<img src="does-not-exist" onerror="attack()" />`*
- *`document.location`* and other URL attribute
  - Script execution via `javascript:`-URLs

Library sinks for client-side XSS

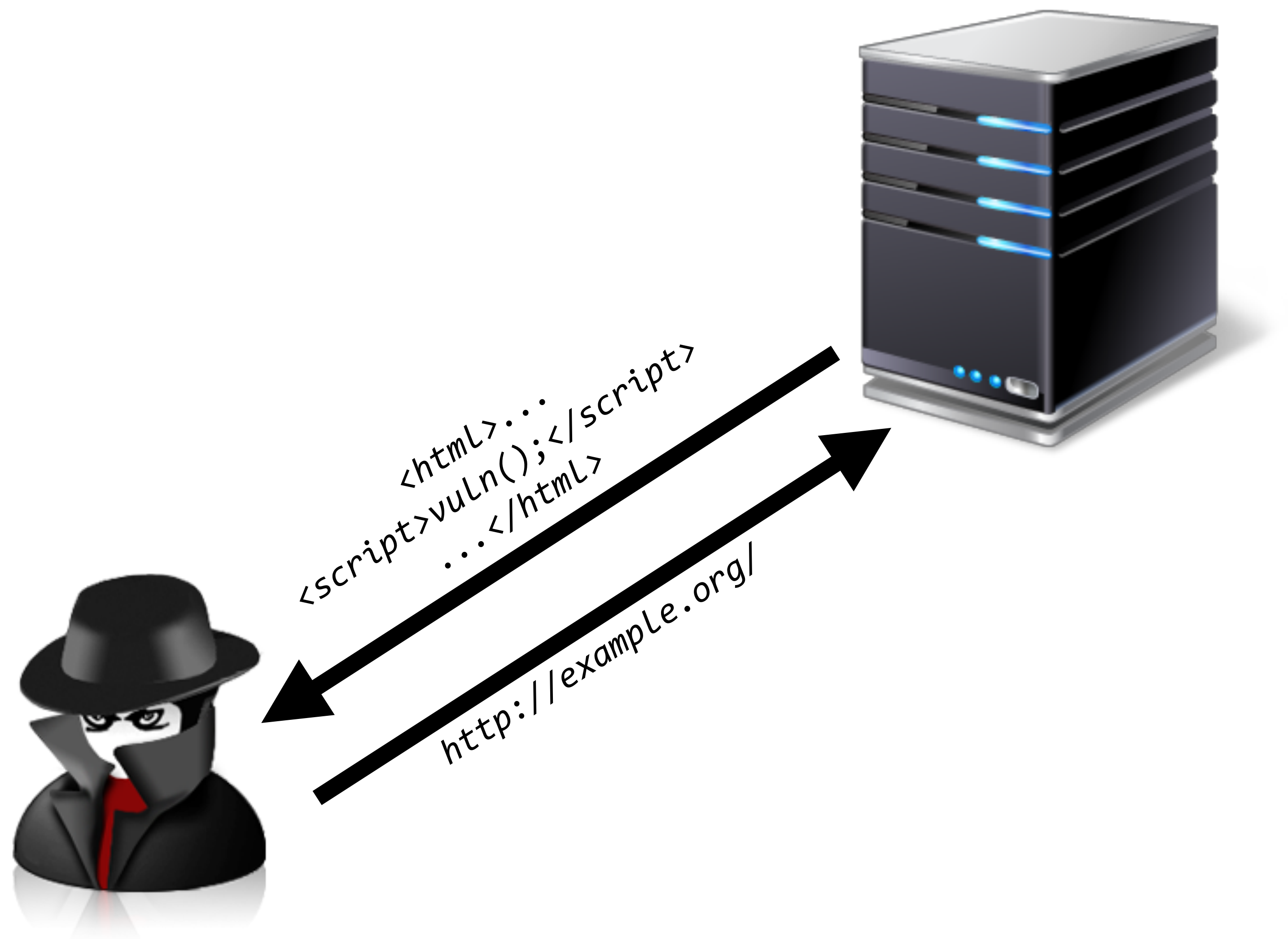
- Nowadays, only few people still write “vanilla” JavaScript
- Instead, using JavaScript frameworks and libraries is commonplace
- Especially relevant in this context is *JQuery*
  - Most notably the .html() API
  - Unlike the DOM’s innerHTML API $.html even executes <script>-tags
- Old versions of JQuery even contained unintended CXSS
  - Queries for non-existing elements created such elements
  - Thus, $(location.hash) could cause JS injection

Sources in Client-side XSS

- A XSS source is controlled by the attacker
  - Only few entry points for attacker data in the browser
- `document.location` and it’s aliases
  - Especially the URL query and hash-part
- `window.name`
  - Can be set for new windows and frames
  - Retains its value after cross-origin navigation
- `document.referer`
  - Requires cross-origin navigation
- Data in `postmessage` events

Attack scenario

1. Attacker analyzes client-side JavaScript code for vulnerabilities
   - searches for unfiltered usage of attacker-controllable data (e.g., the URL)
   - such data may be contained in URL fragment
     - Important: the fragment is not sent to the server

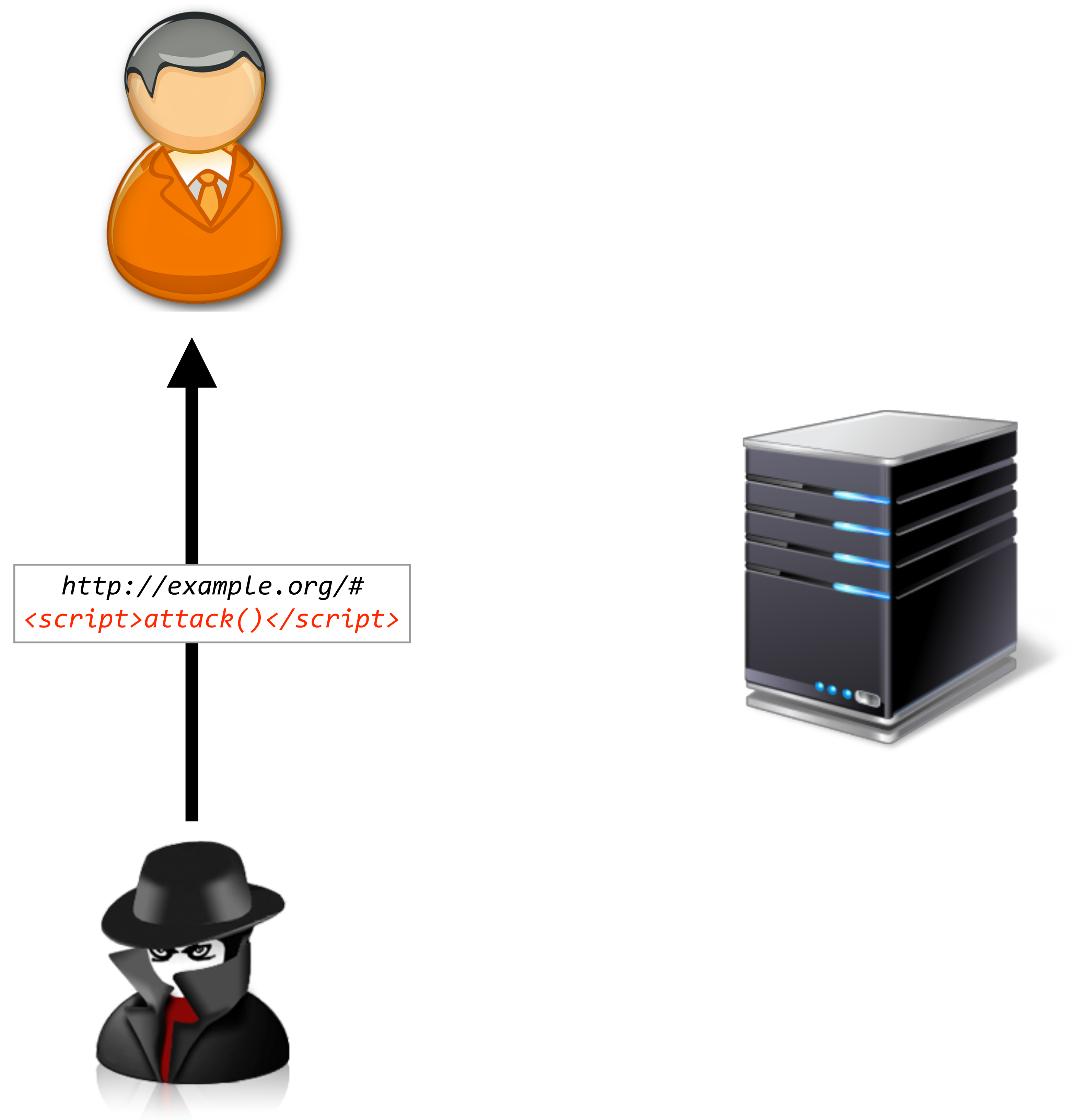


INSTITUTE FOR APPLICATION SECURITY

Attack scenario

2. Attacker tricks victim into visiting URL with payload, e.g., in fragment

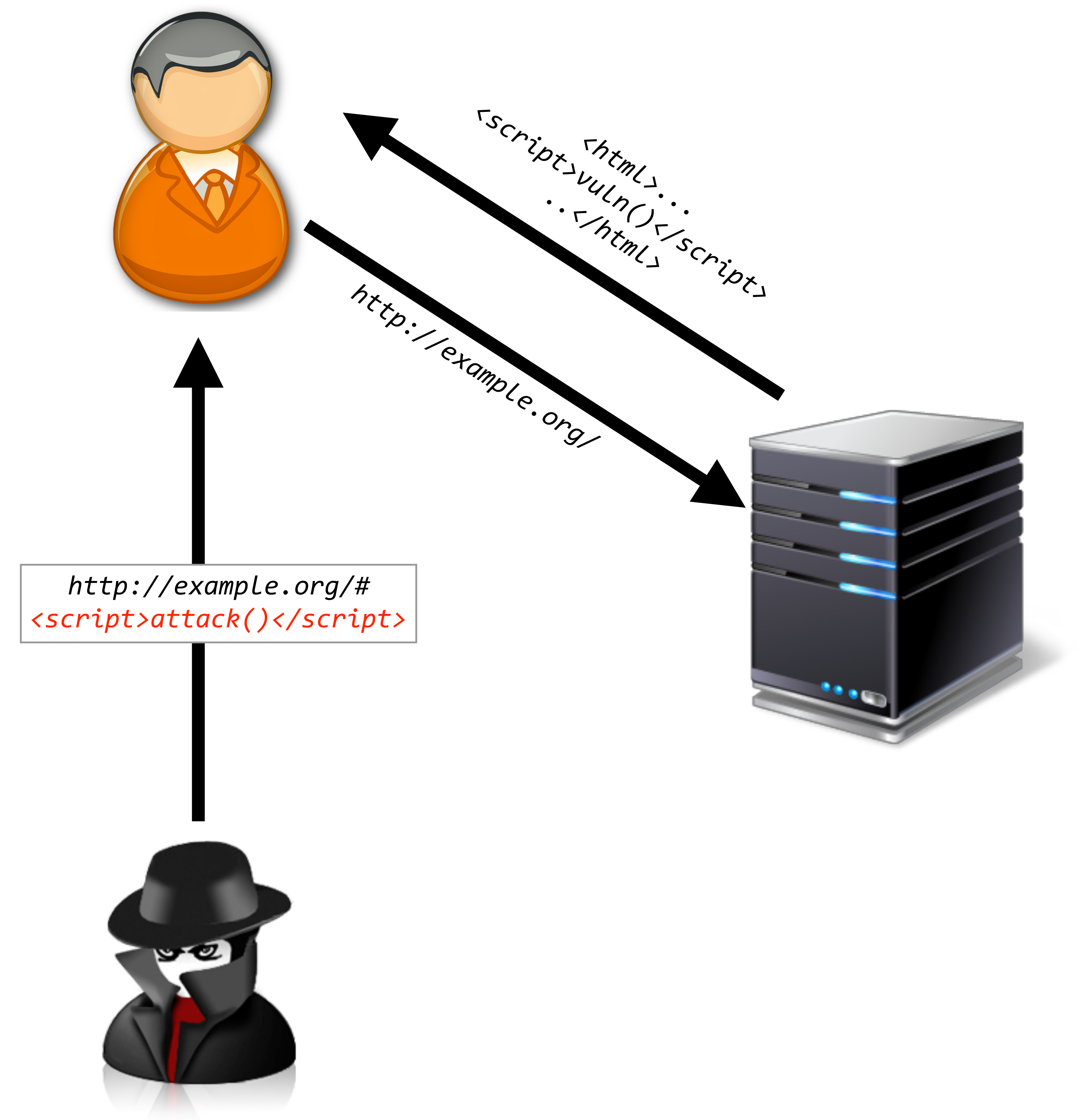
- Vulnerable JavaScript is delivered to client
- Vulnerable JavaScript accesses the attacker controlled data
- Exploit triggered
    - Potentially without payload being sent to server (if in fragment)



Attack scenario

2. Attacker tricks victim into visiting URL with payload, e.g., in fragment

- Vulnerable JavaScript is delivered to client
- Vulnerable JavaScript accesses the attacker controlled data
- Exploit triggered
    - Potentially without payload being sent to server (if in fragment)



Reasons why client-side XSS flew under the radar

Reasons why client-side XSS flew under the radar

- A XSS source partially or fully controlled by the attacker

Reasons why client-side XSS flew under the radar

- A XSS source partially or fully controlled by the attacker
- In case of server-side XSS, the application scenario mandates the direct interaction with the attacker's data
  - Source: HTTP request
  - Sink: HTTP response

Reasons why client-side XSS flew under the radar

- A XSS source partially or fully controlled by the attacker
- In case of server-side XSS, the application scenario mandates the direct interaction with the attacker's data
  - Source: HTTP request
  - Sink: HTTP response
- This is not the case with client-side XSS
  - No mandatory case why the (very limited) amount of sources should influence the various sinks
  - …actually, why would you want to do this?

Study: Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

Study: Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Recall: JavaScript is highly dynamic
  - sound static analysis pretty much dies with eval()
  - prototype chaining increases difficulty even further

Study: Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Recall: JavaScript is highly dynamic
  - sound static analysis pretty much dies with eval()
  - prototype chaining increases difficulty even further
- On abstract level, XSS is insecure data flow
  - from attacker-controllable sources
    - e.g., URL, referrer, cookies, window.name, postMessage, ...
  - to security-critical sinks
    - e.g., document.write (writes HTML), eval (executes JavaScript), ...

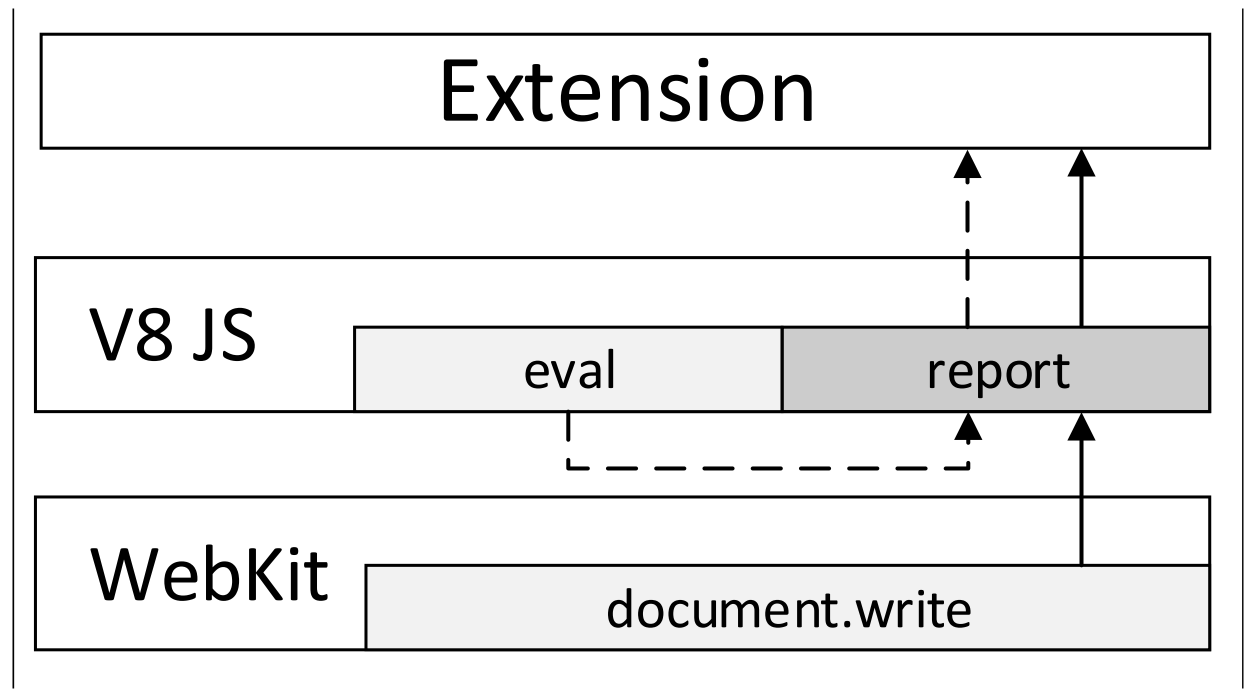
Study: Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Recall: JavaScript is highly dynamic
  - sound static analysis pretty much dies with eval()
  - prototype chaining increases difficulty even further
- On abstract level, XSS is insecure data flow
  - from attacker-controllable sources
    - e.g., URL, referrer, cookies, window.name, postMessage, ...
  - to security-critical sinks
    - e.g., document.write (writes HTML), eval (executes JavaScript), ...
- We can use dynamic analysis for detection
  - precisely: taint tracking

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- We implemented byte-level tainting in Chromium
  - able to taint strings, understand usage of encoding functions
  - on access to security-critical sink, string and taint info reported to backend

- Conducted large-scale study on data flows
  - Alexa Top 5000 shallow crawl
  - 504,275 URLs, 4,358,031 frames in total
  - **24,474,306** data flows
    - only JavaScript/HTML flows: **4,948,264**
    - only directly controllable sources: **1,825,598**
    - only unfiltered flows: **313,794**



Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Not every unencoded flow is vulnerable

```
<script>
  if (/^[a-z][0-9]+$/.test(location.hash.slice(1)) {
    document.write(location.hash.slice(1));
  }
</script>
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Not every unencoded flow is vulnerable

```
<script>
  if (/^[a-z][0-9]+$/.test(location.hash.slice(1)) {
    document.write(location.hash.slice(1));
  }
</script>
```

- `http://example.org/#top`
  - measurable data flow

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Not every unencoded flow is vulnerable

```
<script>
  if (/^[a-z][0-9]+$/.test(location.hash.slice(1)) {
    document.write(location.hash.slice(1));
  }
</script>
```

- `http://example.org/#top`
  - measurable data flow
- `http://example.org/#<script>alert(1)</script>`
  - does not pass regular expression

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Not every unencoded flow is vulnerable

```
<script>
  if (/^[a-z][0-9]+$/.test(location.has
    document.write(location.hash.sl
  }
</script>
```

- *http://examp*
  - measurabl
- *http://exampl*   *(1)</script>*
  - does not pass r   on

Approach:
Verify vulnerabilities via successful exploits

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Exploitation is dependent on the context

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Exploitation is dependent on the context
  - HTML context requires <script> tags

```
document.write("<input value='"
  + location.hash.slice(1) + "'>");
```

```
'><script>alert(1);</script><textarea>
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Exploitation is dependent on the context
  - HTML context requires <script> tags

```
document.write("<input value='"
  + location.hash.slice(1) + "'>");
```

```
'><script>alert(1);</script><textarea>
```

  - JavaScript context only requires new JavaScript statements

```
eval("var x = '" + location.hash + "'");
```

```
'; alert(1);//
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Exploitation is dependent on the context
    - HTML context requires <script> tags

```
document.write("<input value='"
  + location.hash.slice(1) + "'>");
```

```
'><script>alert(1);</script><textarea>
```

- JavaScript context only requires new JavaScript statements

```
eval("var x = '" + location.hash + "'");
```

```
'; alert(1);//
```

- URL context requires javascript: URL

```
var frame = document.createElement("iframe");
frame.src = location.hash.slice(1) + "/test.html";
```

```
javascript:alert(1);//
```

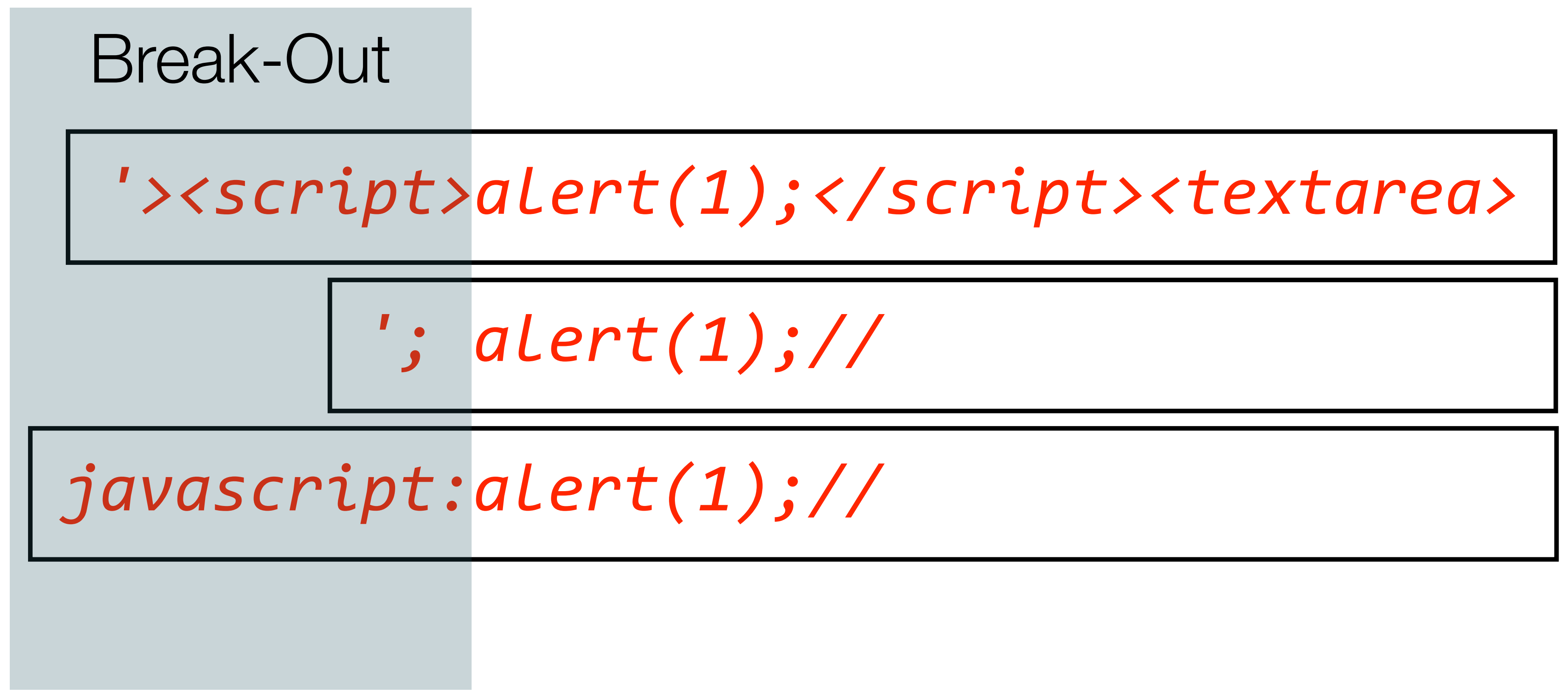
Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
'><script>alert(1);</script><textarea>
```

```
'; alert(1);//
```

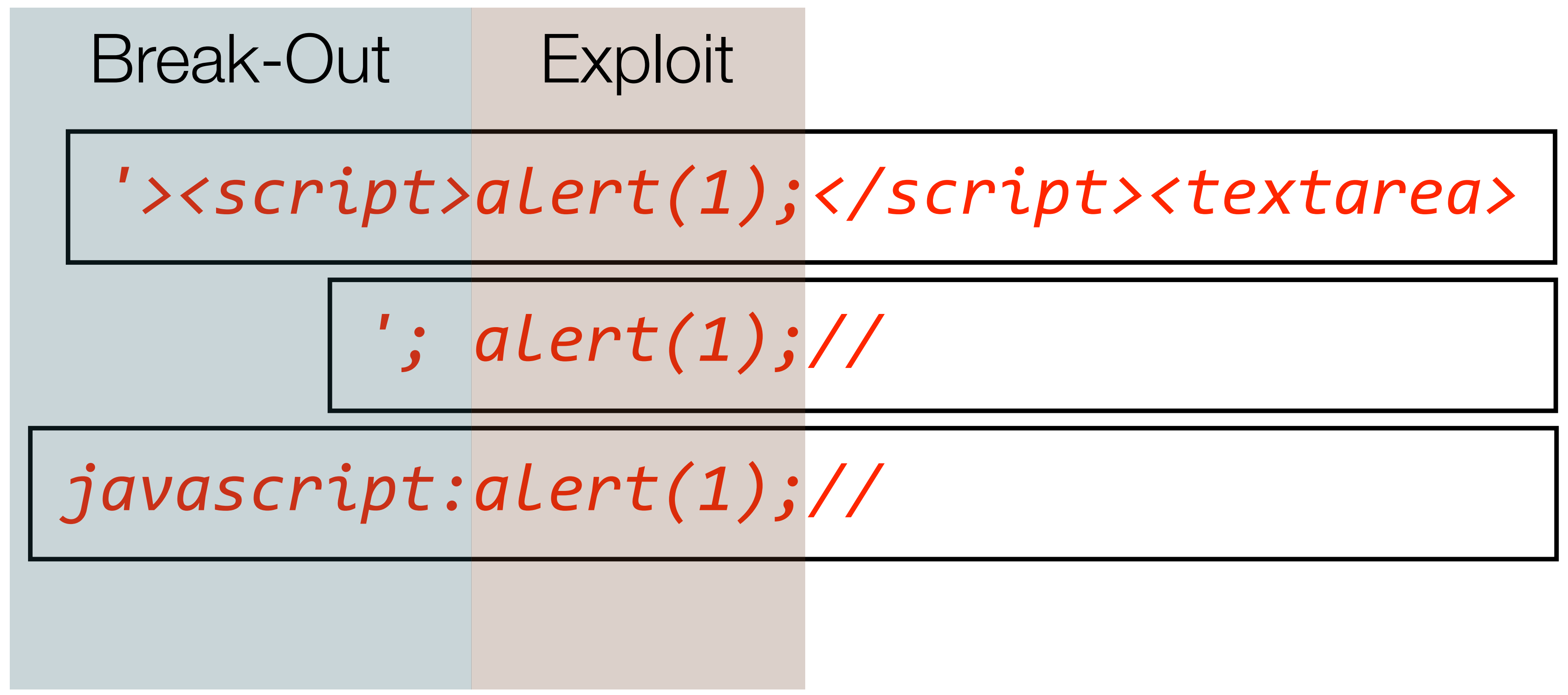
```
javascript:alert(1);//
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



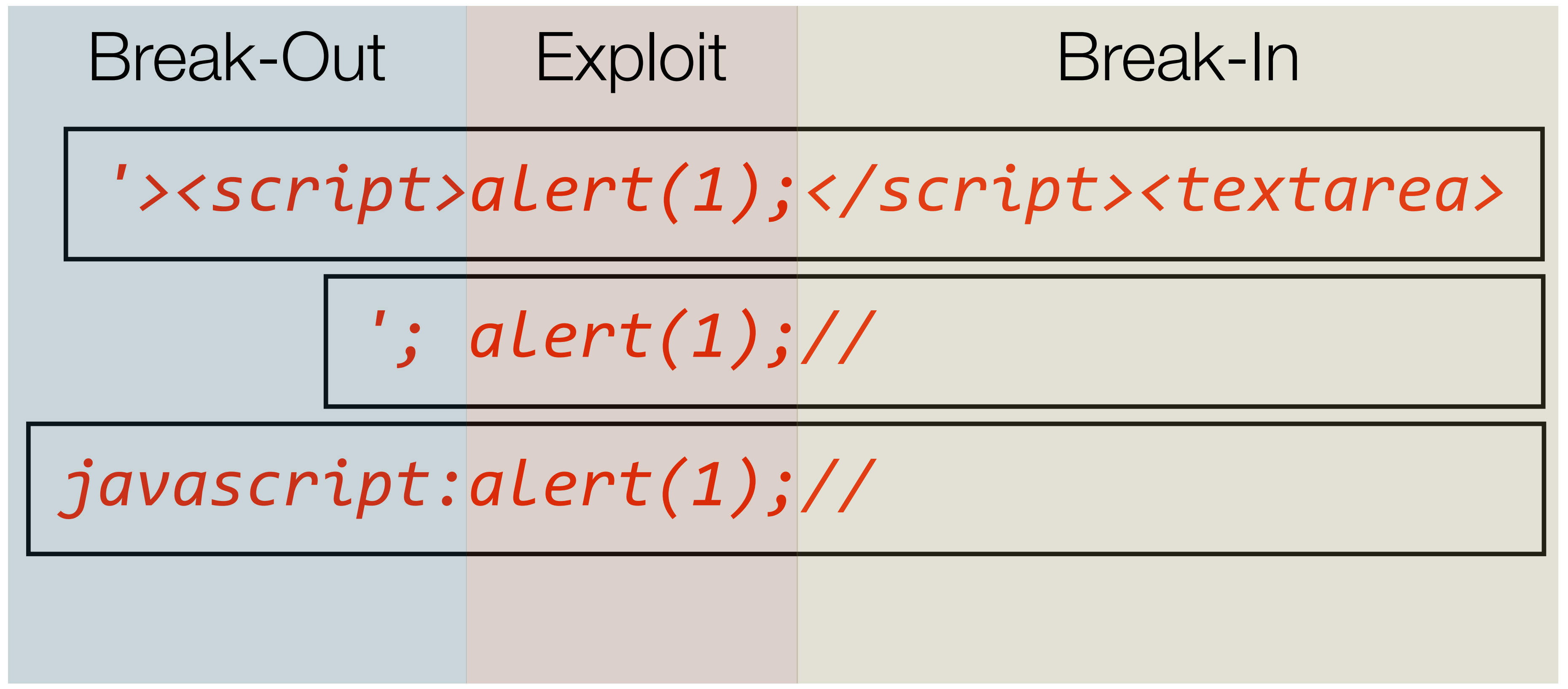
```
'><script>alert(1);</script><textarea>
'; alert(1);//
javascript:alert(1);//
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



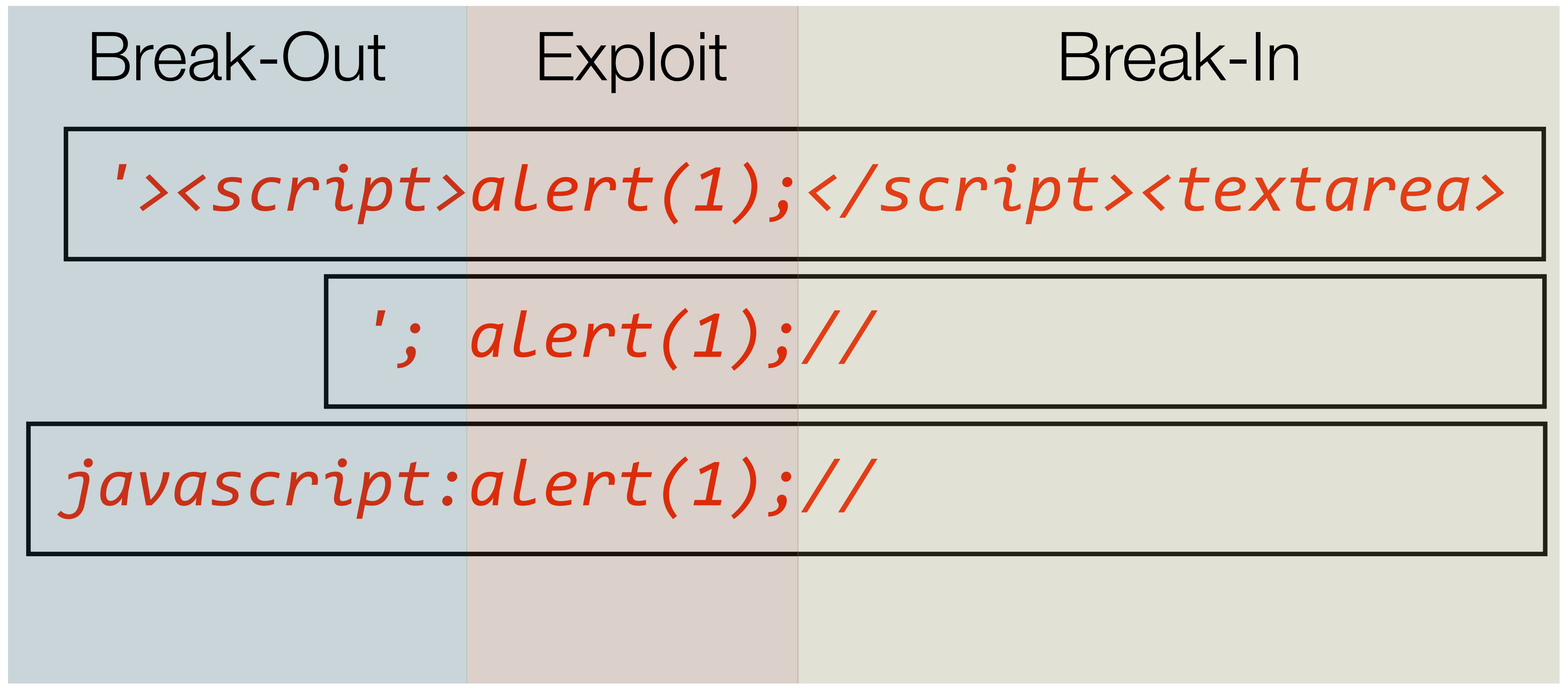
```
'><script>alert(1);</script><textarea>
'; alert(1);//
javascript:alert(1);//
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



```
'><script>alert(1);</script><textarea>
'; alert(1);//
javascript:alert(1);//
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



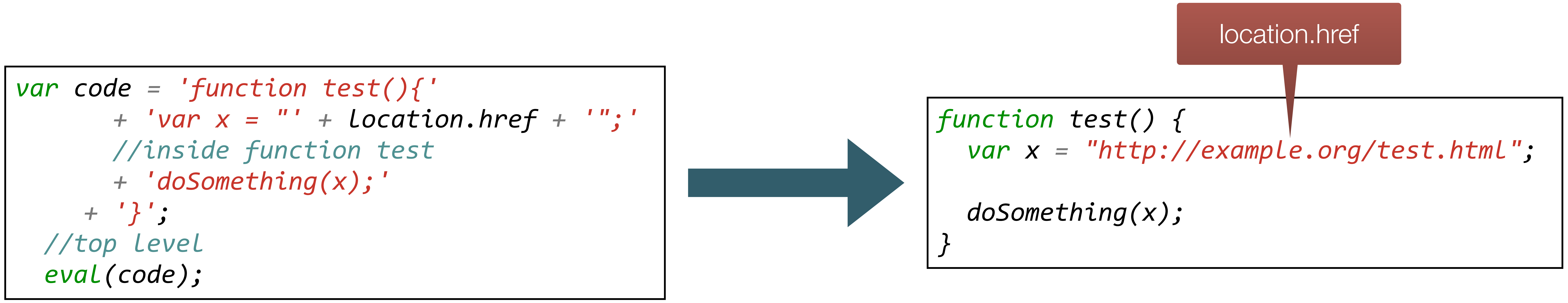
```
'><script>alert(1);</script><textarea>
'; alert(1);//
javascript:alert(1);//
```

- Break-Out depends on context and point of injection
- Exploit is same regardless of context/injection
- Break-In only depends on the context

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
       + 'var x = "' + location.href + '";'
       //inside function test
       + 'doSomething(x);'
     + '}';
  //top level
  eval(code);
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



```
var code = 'function test(){'
        + 'var x = "' + location.href + '";'
        //inside function test
        + 'doSomething(x);'
      + '}';
  //top level
  eval(code);
```

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
       + 'var x = "' + location.href + '";'
       //inside function test
       + 'doSomething(x);'
     + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
       + 'var x = "' + location.href + '";'
       //inside function test
       + 'doSomething(x);'
     + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

INSTITUTE FOR APPLICATION SECURITY

```
var code = 'function test(){'
       + 'var x = "' + location.href + '";'
       //inside function test
       + 'doSomething(x);'
     + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

- End string literal: "

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
        + 'var x = "' + location.href + '";'
        //inside function test
        + 'doSomething(x);'
      + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

- End string literal: "
- End declaration ;

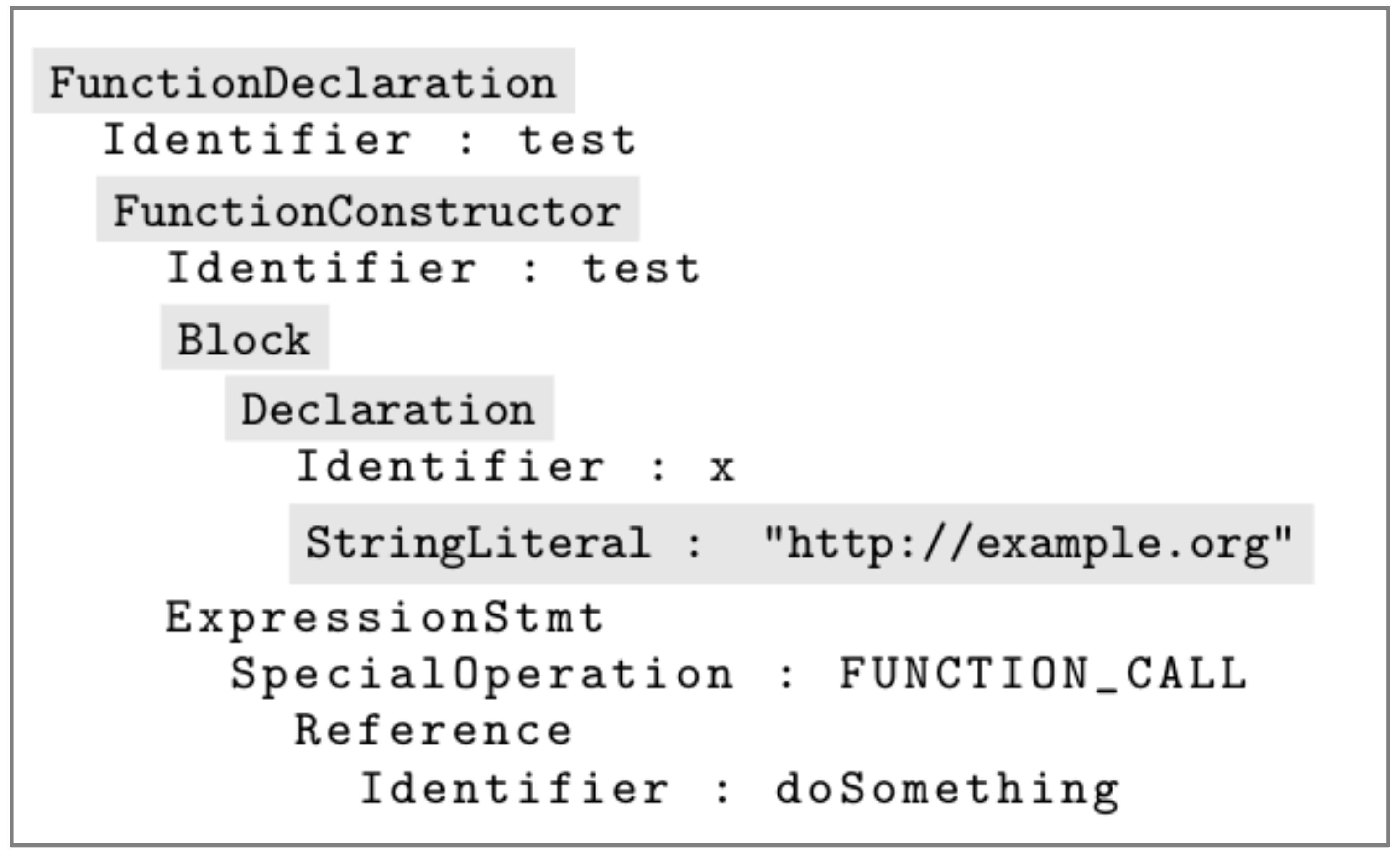
Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
        + 'var x = "' + location.href + '";'
        //inside function test
        + 'doSomething(x);'
      + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

- End string literal: "
- End declaration ;
- End block }



```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
       + 'var x = "' + location.href + '";'
       //inside function test
       + 'doSomething(x);'
     + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

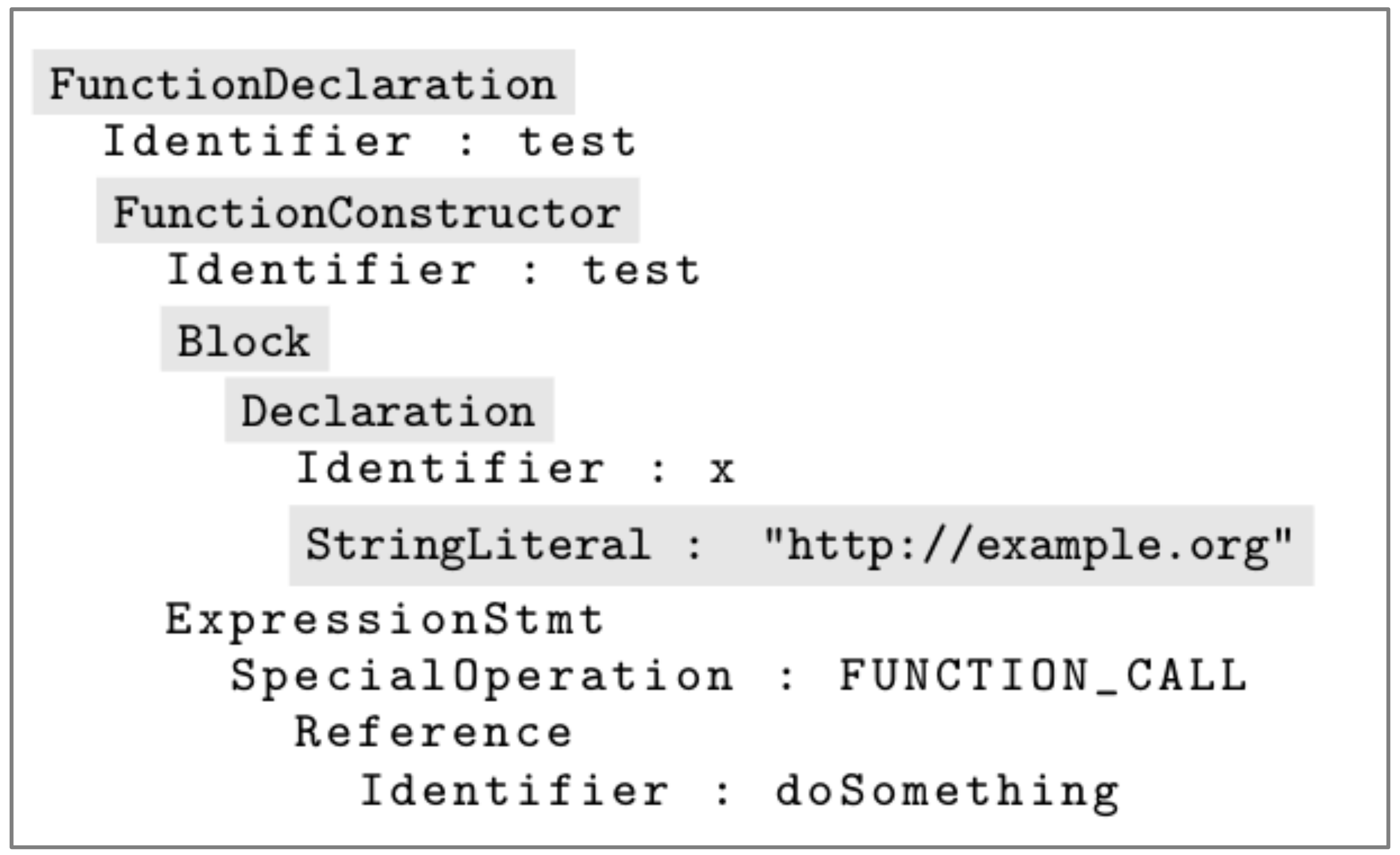
- End string literal: `"`
- End declaration `;`
- End block `}`
- Exploit `alert(1);`

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
        + 'var x = "' + location.href + '";'
        //inside function test
        + 'doSomething(x);'
      + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```



```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

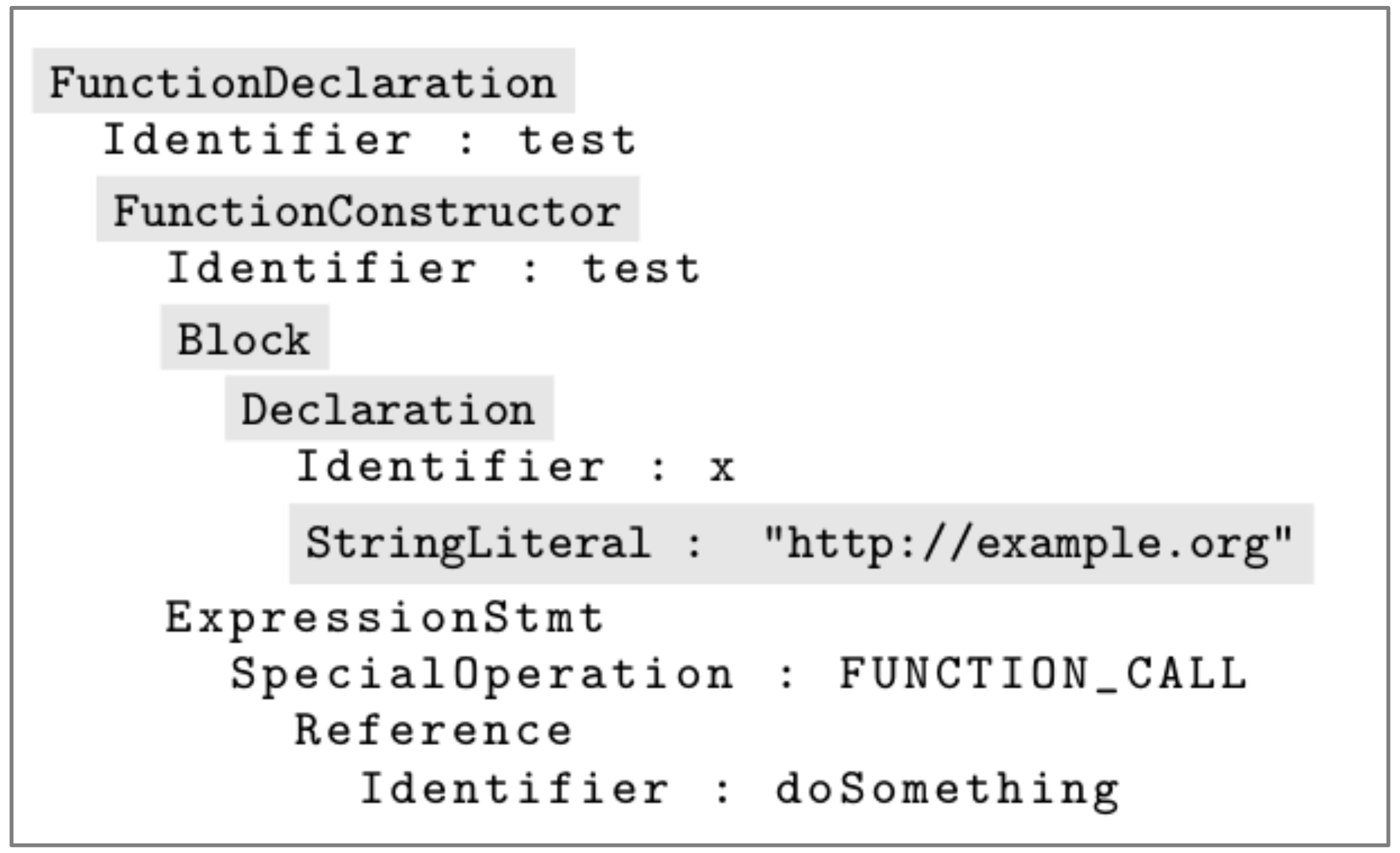
End string literal:	"
End declaration	;
End block	}
Exploit	*alert(1);*
Break-In	//

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

```
var code = 'function test(){'
       + 'var x = "' + location.href + '";'
       //inside function test
       + 'doSomething(x);'
     + '}';
  //top level
  eval(code);
```

location.href

```
function test() {
  var x = "http://example.org/test.html";

  doSomething(x);
}
```

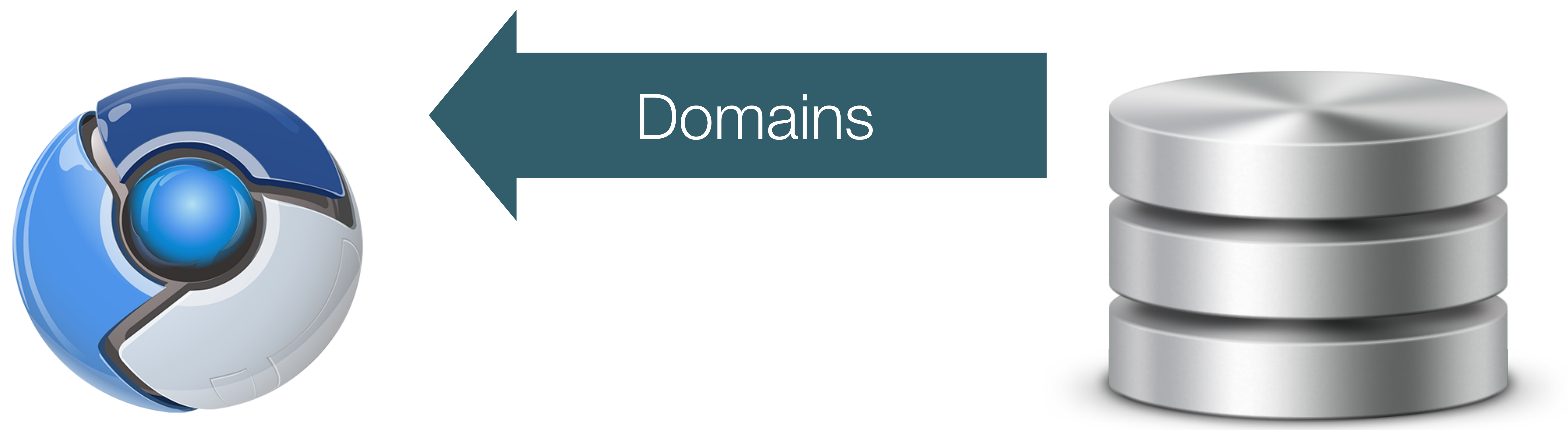


```
FunctionDeclaration
  Identifier : test
  FunctionConstructor
    Identifier : test
    Block
      Declaration
        Identifier : x
        StringLiteral : "http://example.org"
    ExpressionStmt
      SpecialOperation : FUNCTION_CALL
        Reference
          Identifier : doSomething
```

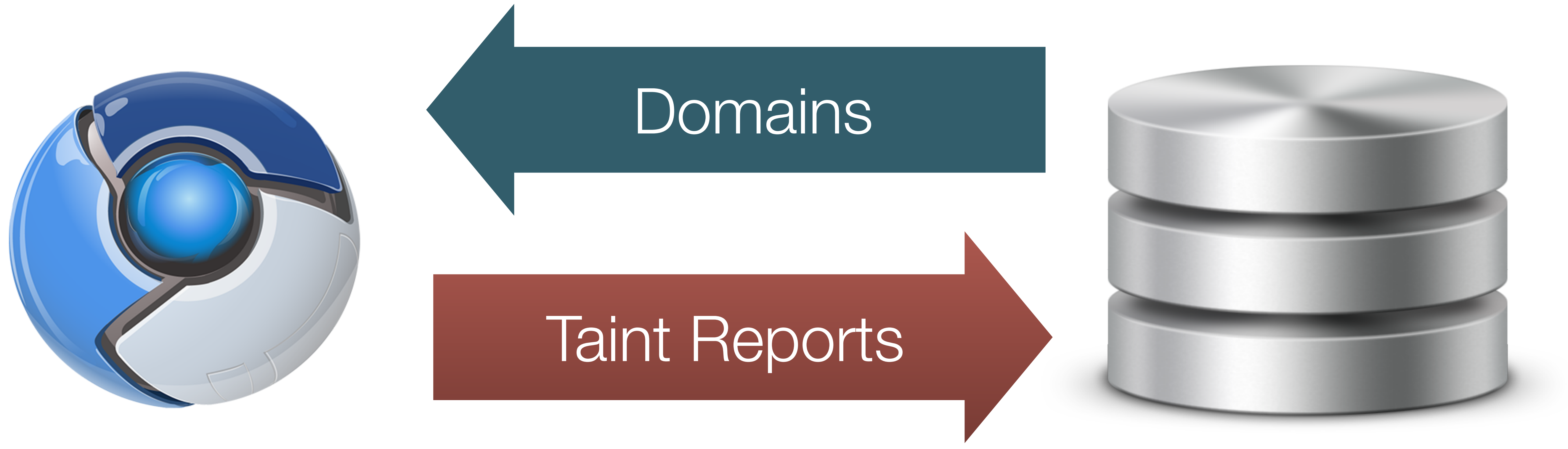
- End string literal: *"*
- End declaration *;*
- End block *}*
- Exploit *alert(1);*
- Break-In *//*
- Final exploit: *";} alert(1); //*

Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

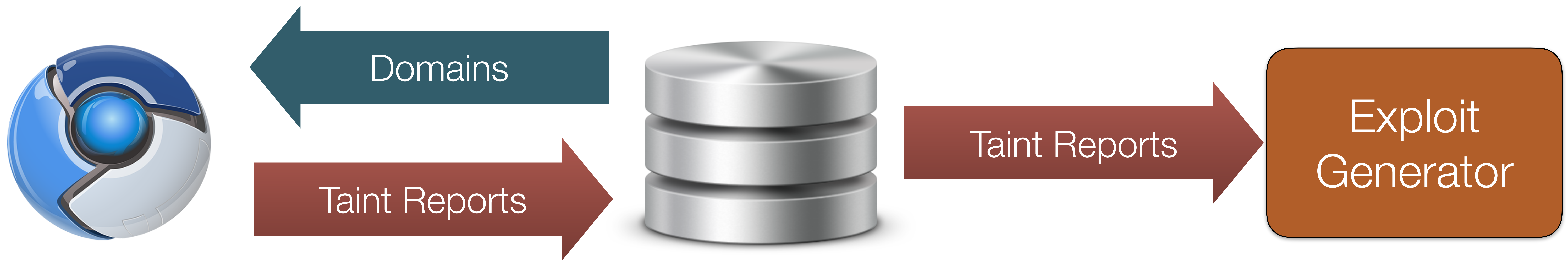
Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

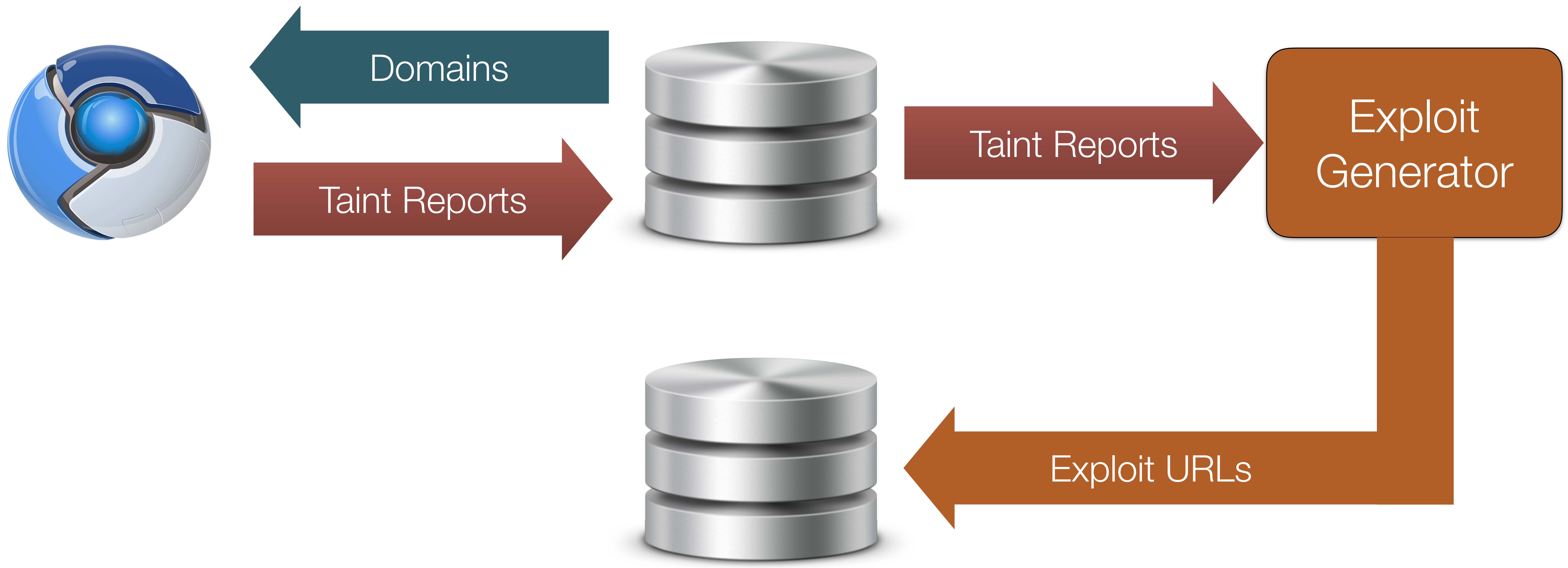


Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

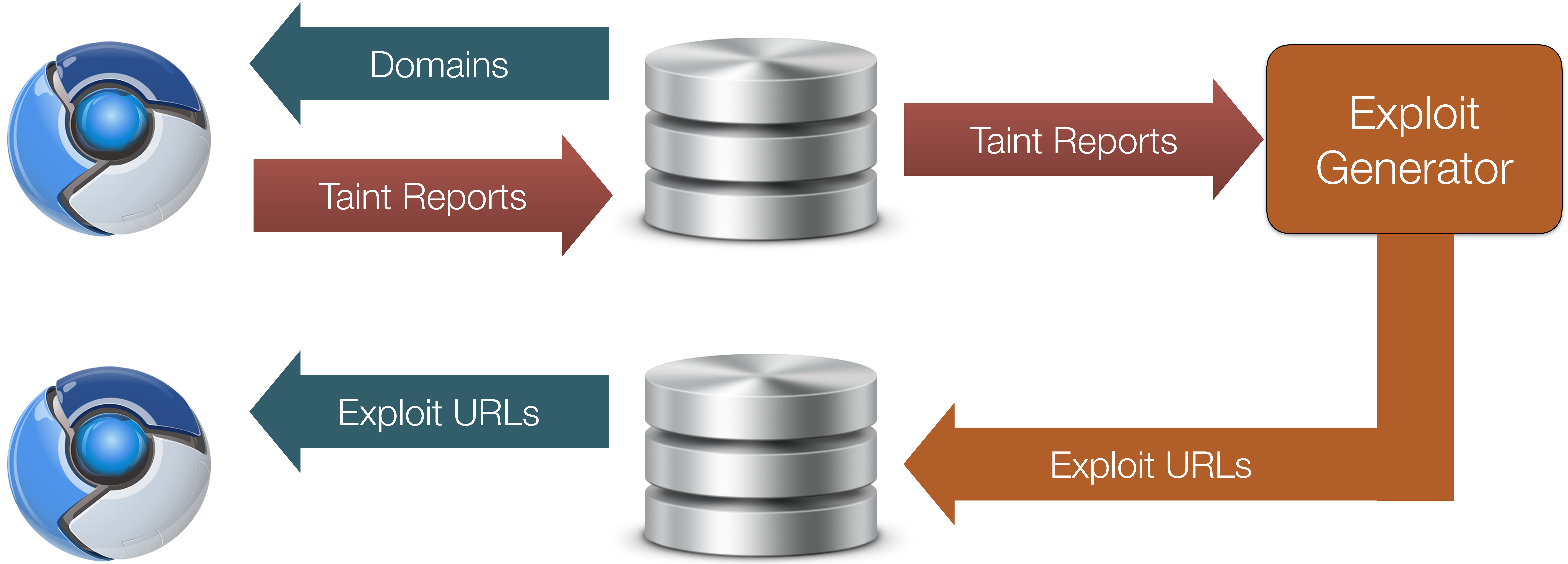


IAS INSTITUTE FOR APPLICATION SECURITY

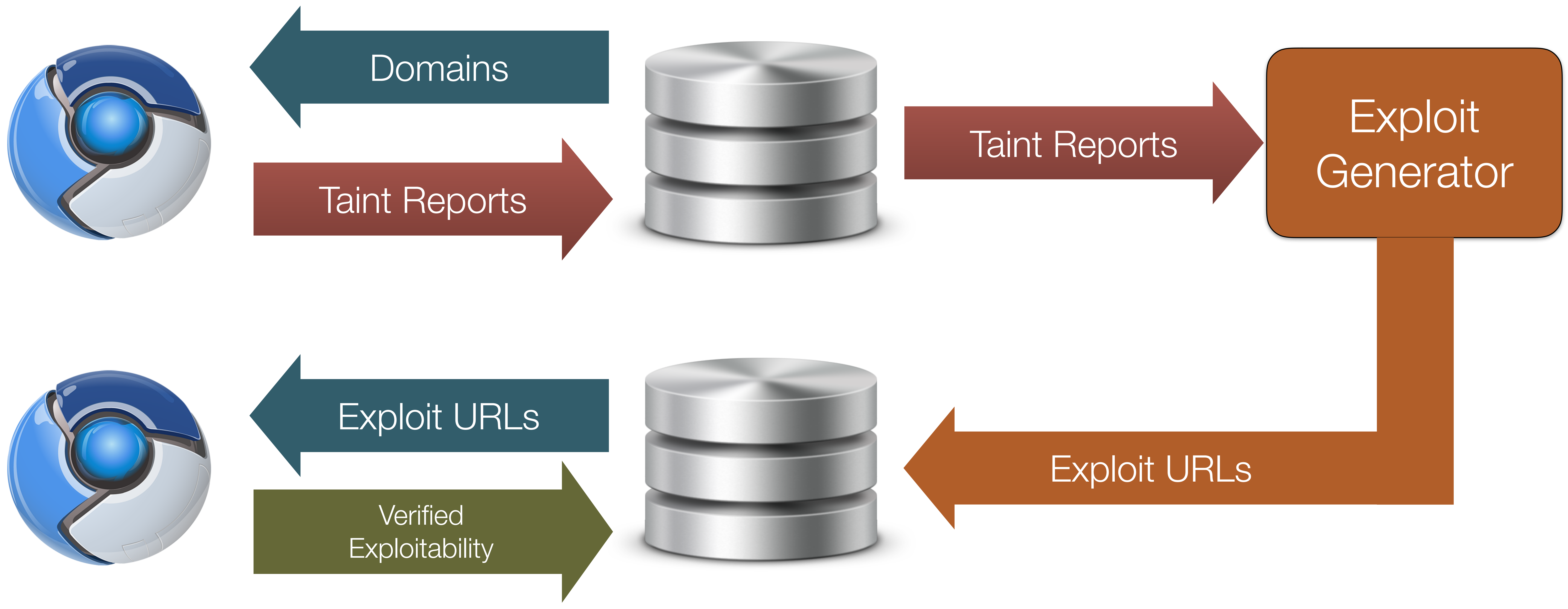
Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

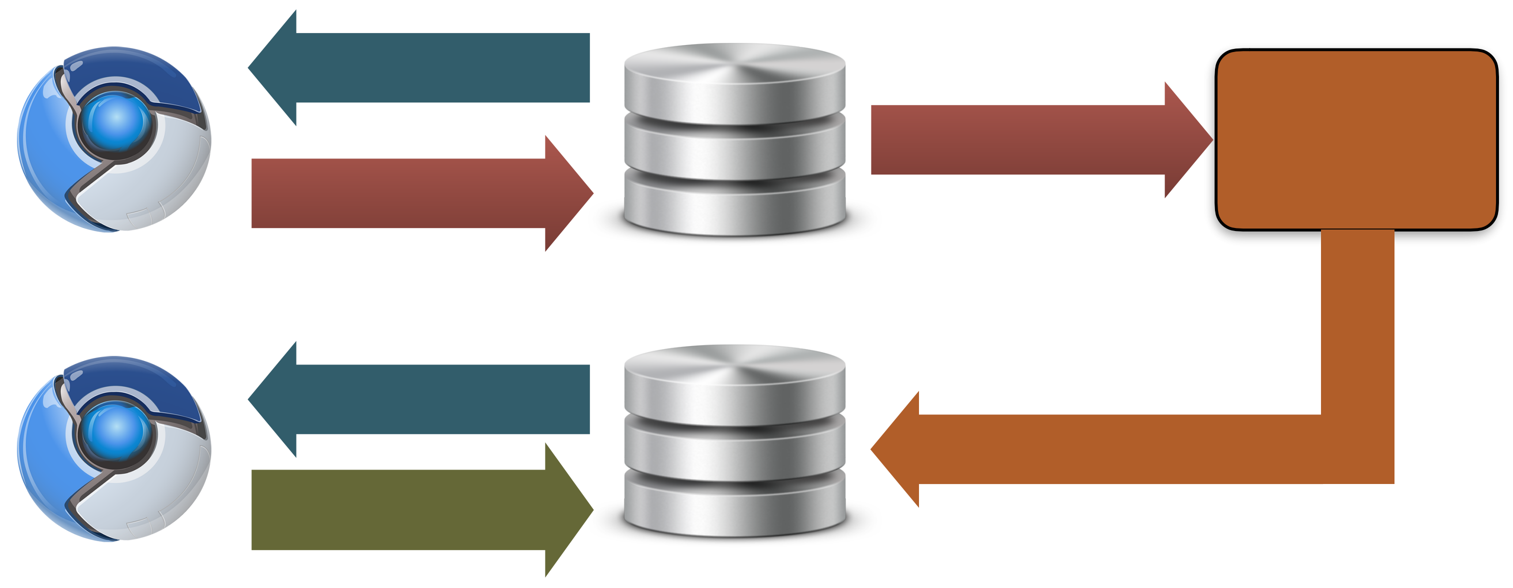


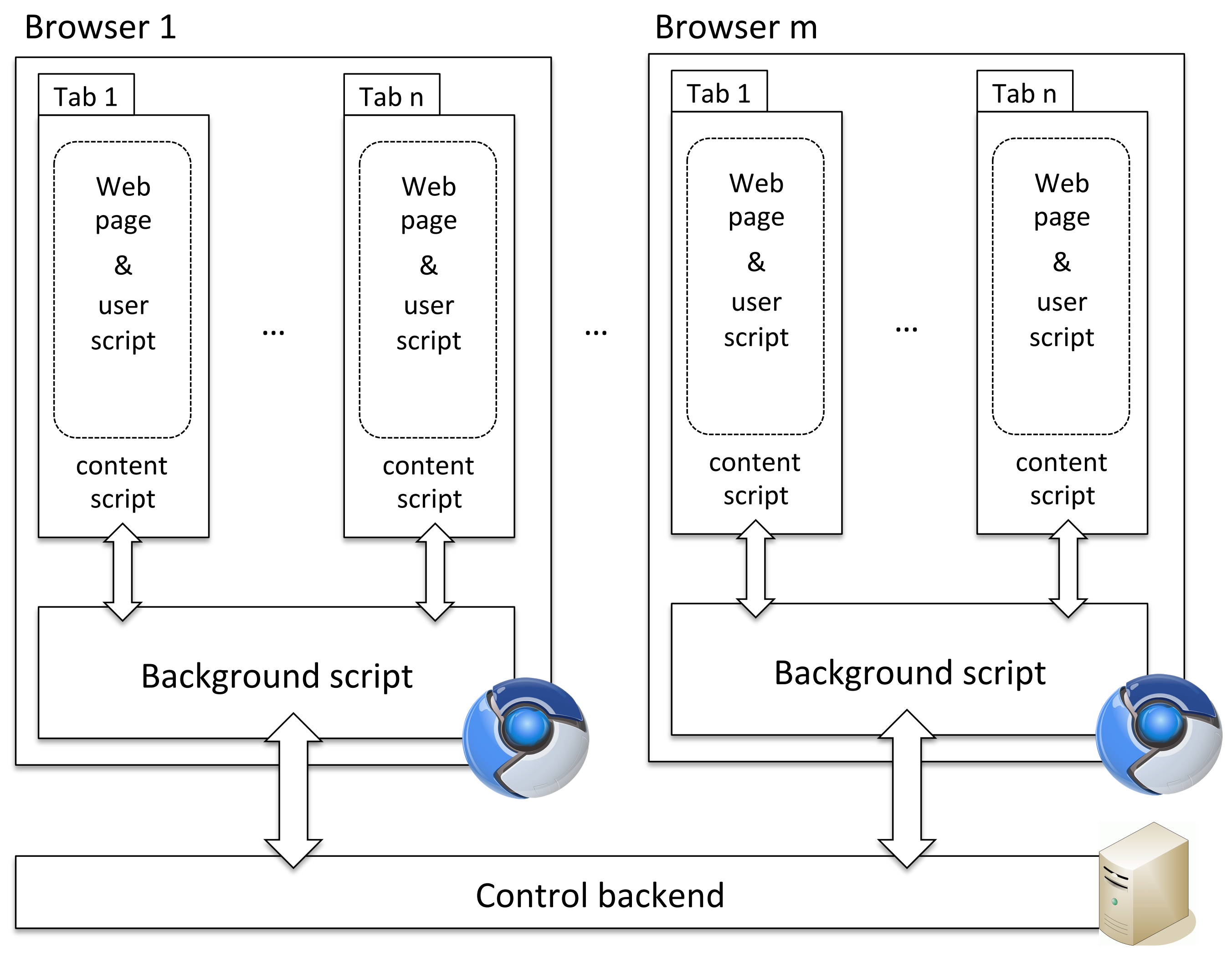
Detecting Reflected Client-Side Cross-Site Scripting [CCS13]



Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

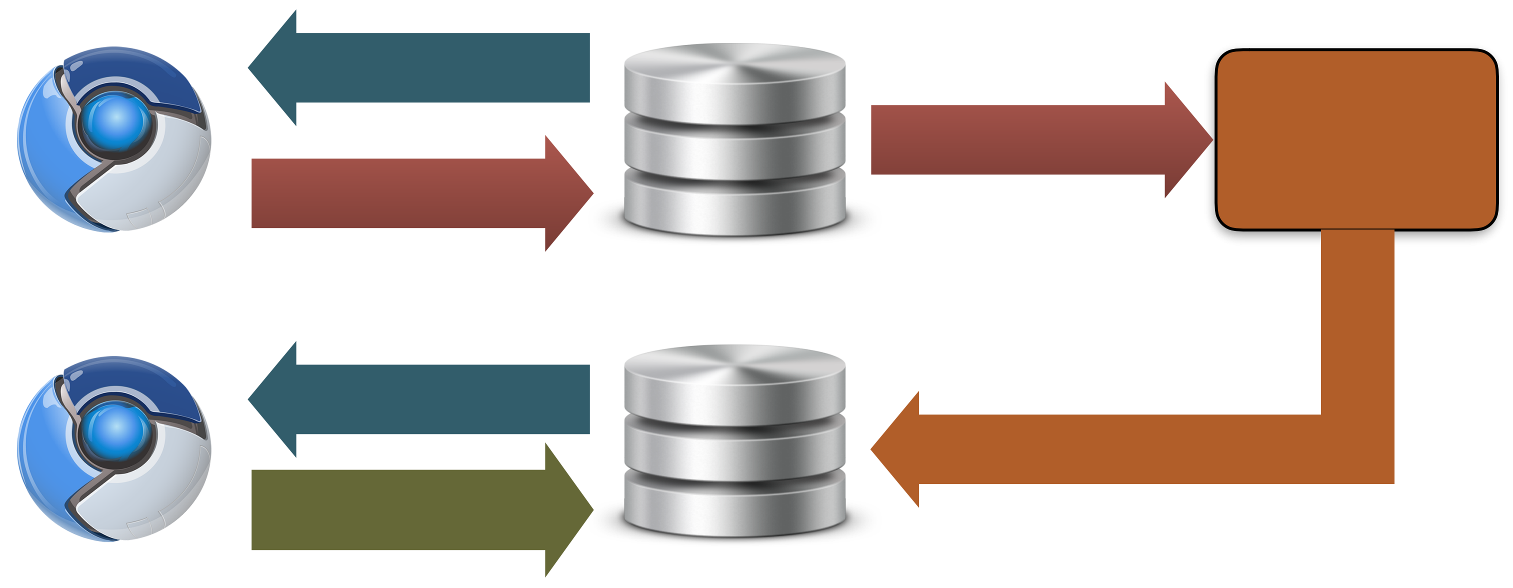
- Conducted large-scale study on data flows
  - Alexa Top 5000 shallow crawl
  - 504,275 URLs, 4,358,031 frames in total
  - **24,474,306** data flows
    - only JavaScript/HTML flows: **4,948,264**
    - only directly controllable sources: **1,825,598**
    - only unfiltered flows: **313,794**

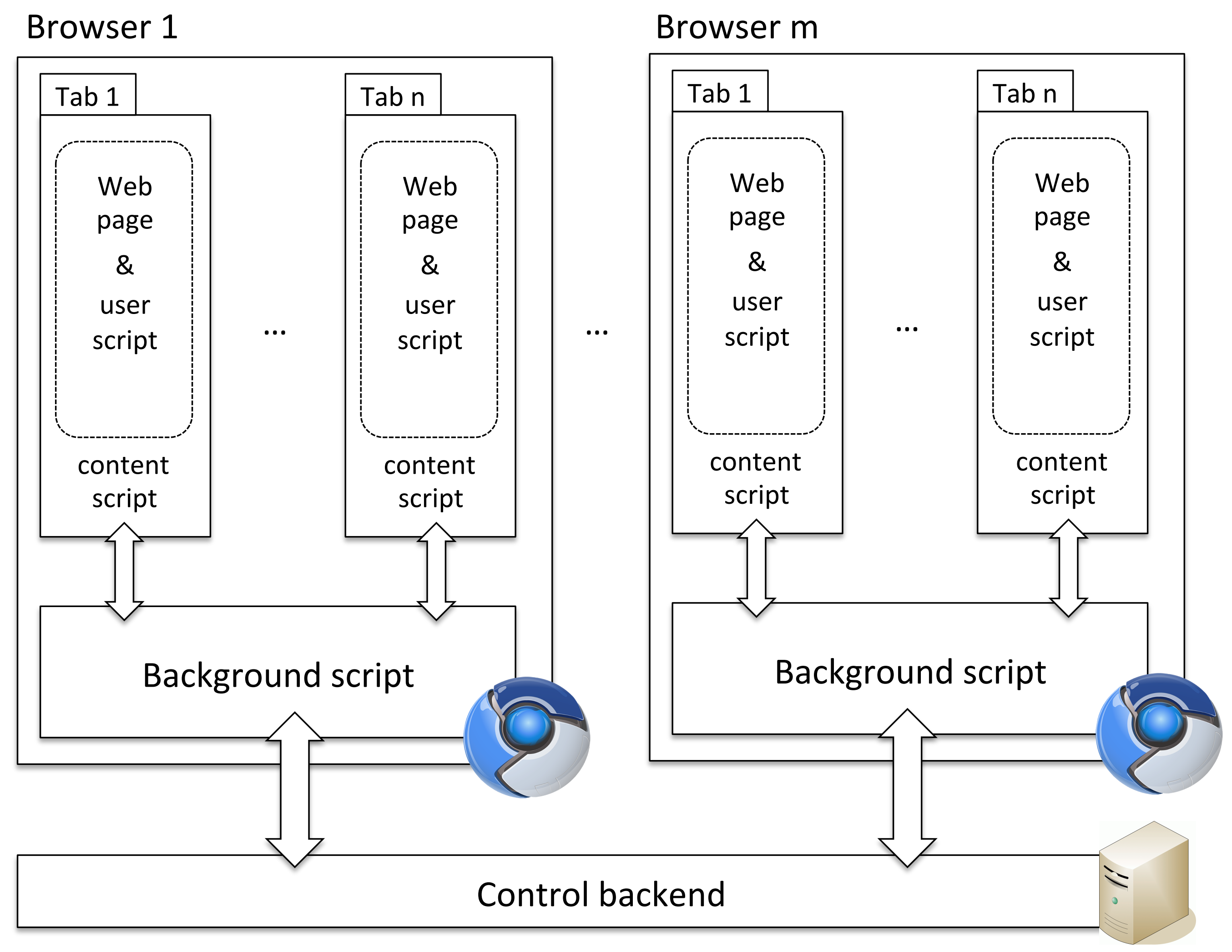




Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

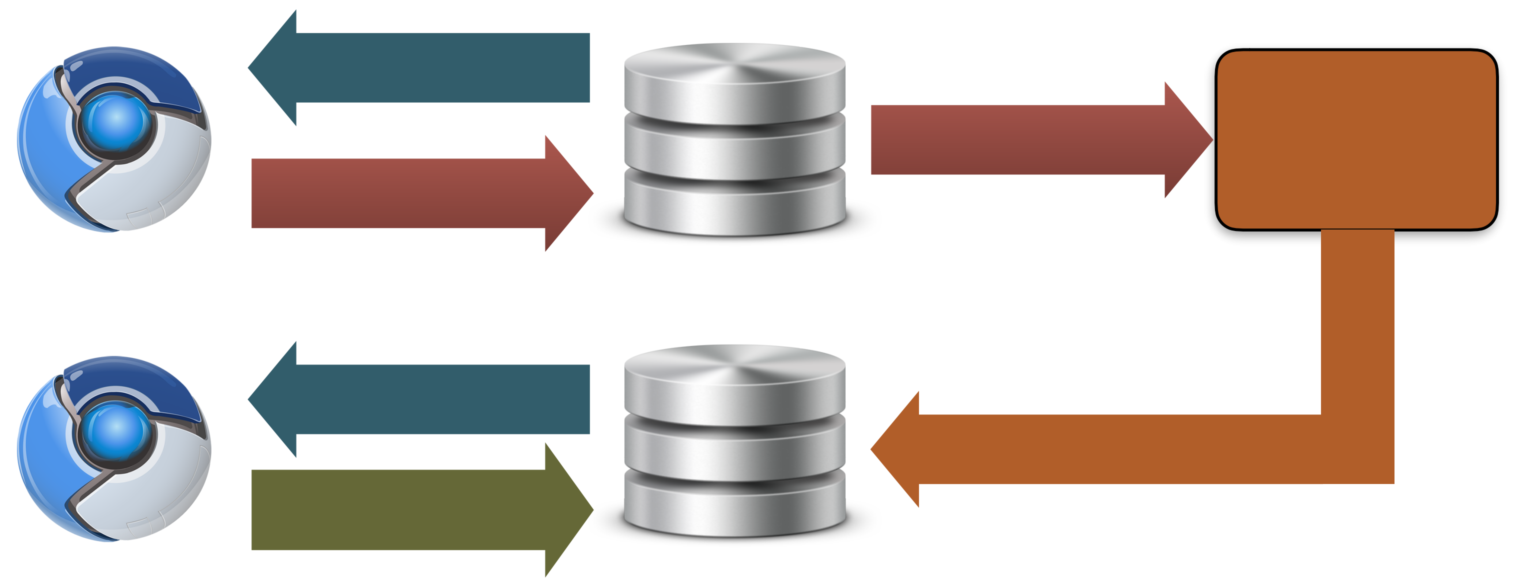
- Conducted large-scale study on data flows
  - Alexa Top 5000 shallow crawl
  - 504,275 URLs, 4,358,031 frames in total
  - **24,474,306** data flows
    - only JavaScript/HTML flows: **4,948,264**
    - only directly controllable sources: **1,825,598**
    - only unfiltered flows: **313,794**
  - **181,238** unique test cases
    - others were duplicate combinations of URLs + payloads

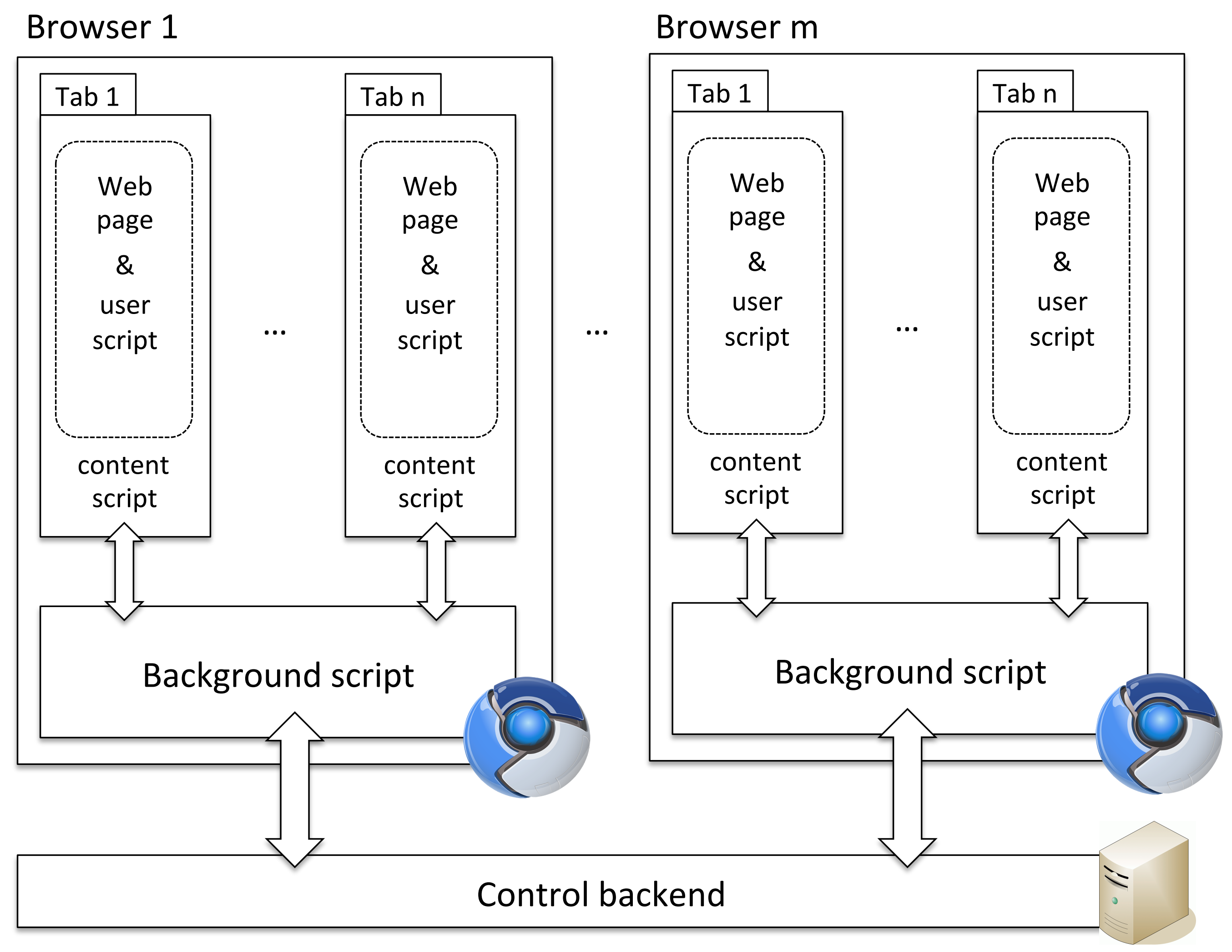




Detecting Reflected Client-Side Cross-Site Scripting [CCS13]

- Conducted large-scale study on data flows
  - Alexa Top 5000 shallow crawl
  - 504,275 URLs, 4,358,031 frames in total
  - **24,474,306** data flows
    - only JavaScript/HTML flows: **4,948,264**
    - only directly controllable sources: **1,825,598**
    - only unfiltered flows: **313,794**
  - **181,238** unique test cases
    - others were duplicate combinations of URLs + payloads
  - **69,987** successful exploits
    - affected **701** domains in total
    - ... and **480** in top 5000 domains





So, we learned that CXSS is very common

So, we learned that CXSS is very common

- But, why?

So, we learned that CXSS is very common

- But, why?

So, we learned that CXSS is very common

- But, why?
- Theory one:
  - JavaScript is written by frontend developers
  - Frontend dev lack the security background and cause obvious mistakes

So, we learned that CXSS is very common

- But, why?
- Theory one:
  - JavaScript is written by frontend developers
  - Frontend dev lack the security background and cause obvious mistakes
- Theory two:
  - Compared to other execution environments, JavaScript in web documents is overly complex
    - Fragmented over the document
    - Comes from multiple sources
    - Relies on dynamic code generation
    - Non-linear control-flow through event driven concurrency model

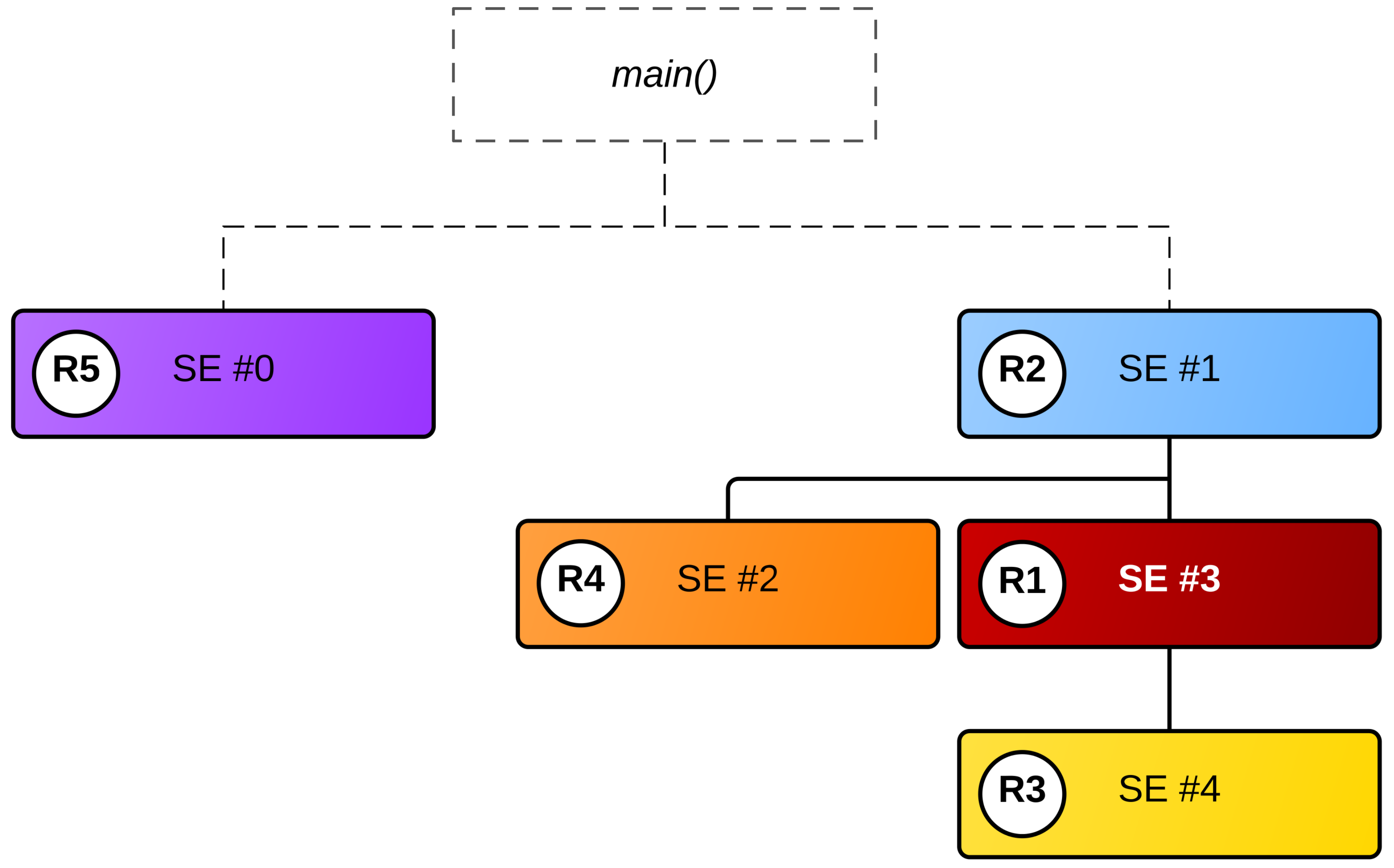
Study 2: Going deeper into CXSS [CCS15]

- Methodology: Advanced taint browser
  - Firefox-based implementation
- New capabilities
  - Recording of all string operations on the tainted data
  - Full function-tracing
    - For the full flow from source to sink
    - Including call relation ships
  - Tracking of involved code contexts
    - Each <script> tag spans it's own code context

Measurable properties of JS complexity [CCS15]

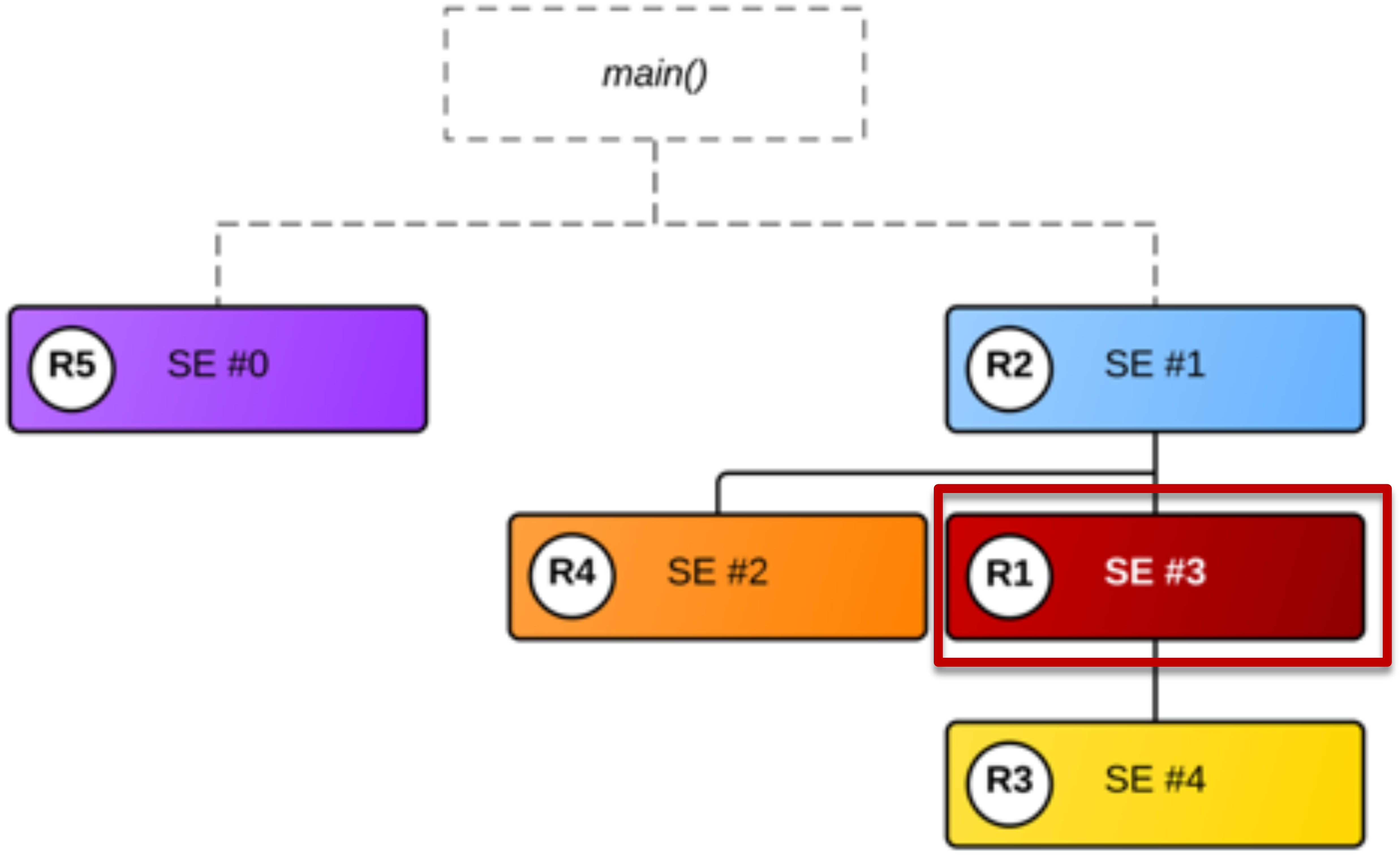
- How can we measure "complexity"?
  - i.e., how difficult would it be for a human to spot the vulnerability?

- Metrics:
  - M1: Number of Operations on the Tainted Data
  - M2: Number of Involved Functions
  - M3: Number of Involved Contexts
  - M4: Code Locality of Source and Sink
  - M5: Callstack Relation between Source and Sink



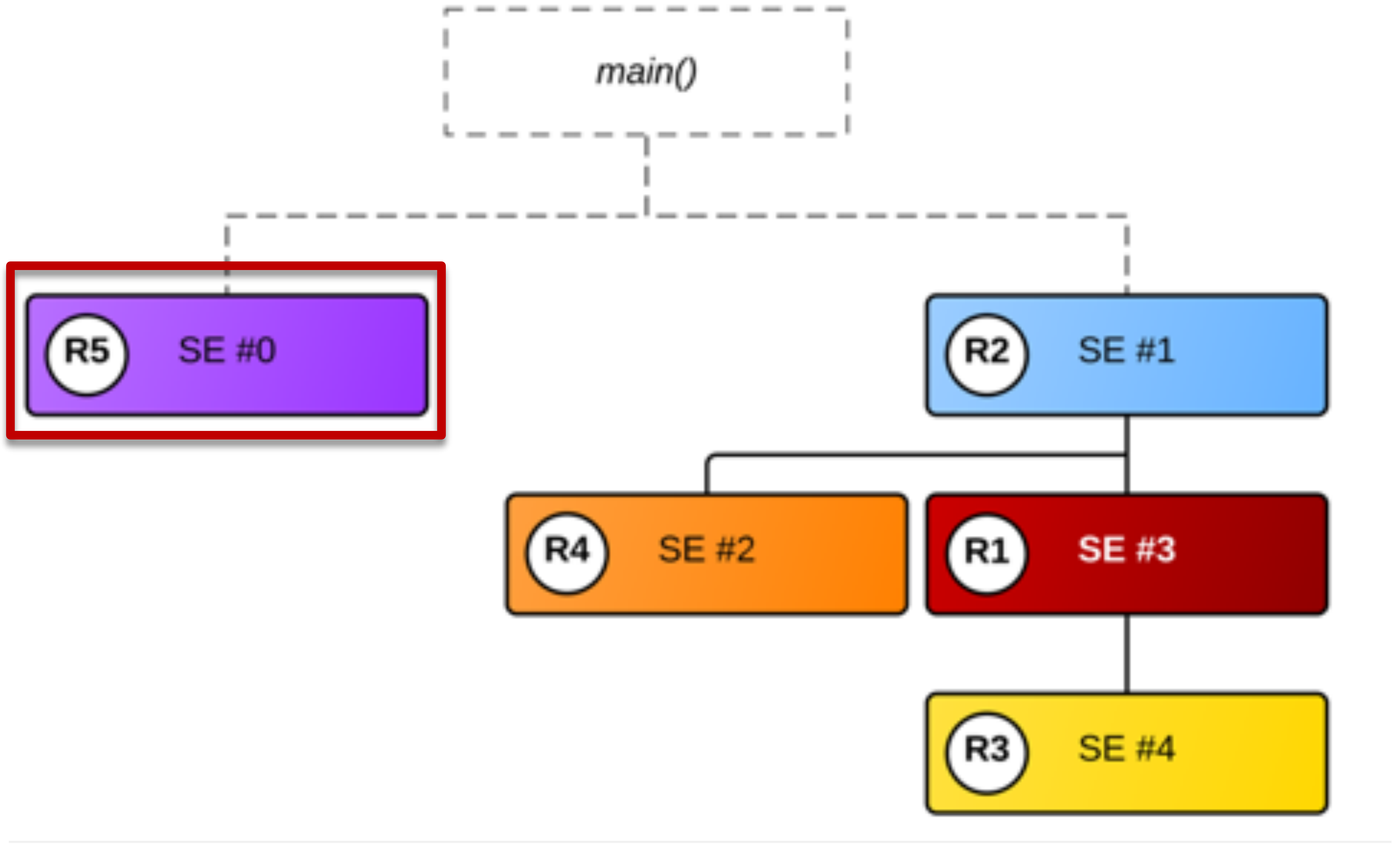
M5: Relation 1 [CCS15]

```
<script>
var source = location.href;
...
document.write(source);
</script>
```



M5: Relation 5 [CCS15]

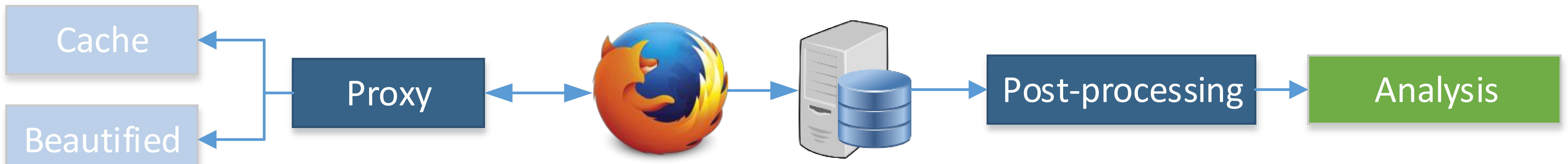
```
<script>
var global = location.href;
...
</script>
...

<script>
eval(global);
</script>
```



Normalizing the Data Set [CCS15]

- Data set: 1,273 real-world vulnerabilities
  - many of them minified
  - Causes issues with metrics
  - many of them not stable (e.g. banner rotation)

- Need to be normalized for a sound analysis
  - Local cache-based vulnerability persistence architecture
  - Allow repeatable experiments



Results [CCS15]

- M1: Number of Operations on the Tainted Data
- M2: Number of Involved Functions
- M3: Number of Involved Contexts
- M4: Code Locality of Source and Sink
- M5: Callstack Relation between Source and Sink

	LC	**MC**	**HC**
M_1	≤ 9	≤ 22	>22
M_2	≤ 4	≤ 10	>10
M_3	≤ 2	3	>3
M_4	≤ 75	≤ 394	>394
M_5	R_1, R_2	R_3, R_4	R_5

	LC	**MC**	**HC**
C_{M1}	1,079	134	60
C_{M2}	1,161	85	27
C_{M3}	1,035	178	60
C_{M4}	*920*	*179*	*51*
C_{M5}	1,094	120	59
Combined	**813** 63.9%	**261** 20.5%	**199** 15.6%

Interpretation [CCS15]

Interpretation [CCS15]

- Are developers overwhelmed by the complexity of flows?
    - ~15% complex flows

Interpretation [CCS15]

- Are developers overwhelmed by the complexity of flows?
  - ~15% complex flows
- Are developers not aware of the pitfalls?
  - Found evidence
    - Explicit decoding
    - Improper API usage
    - Single line flaws

Interpretation [CCS15]

- Are developers overwhelmed by the complexity of flows?
  - ~15% complex flows
- Are developers not aware of the pitfalls?
  - Found evidence
    - Explicit decoding
    - Improper API usage
    - Single line flaws
- Are there special circumstances in the Web model that cause such flaws?
  - Third-party flaws cause vulnerability in including application
  - Unstructured JS embedding in web documents leads to non-linear data/control flows

Case example: Convoluted CXSS [CCS15]

- First party inline script created HTML Meta tags from source data

```
if (parts.length > 1) {
  var kw = decodeURIComponent(parts.pop());
  var meta = document.createElement('meta');
  meta.setAttribute('name', 'keywords');
  meta.setAttribute('content', kw);
  document.head.appendChild(meta);
}
```

- A 3rd party, external script reads the meta tags and uses them for DOM manipulation

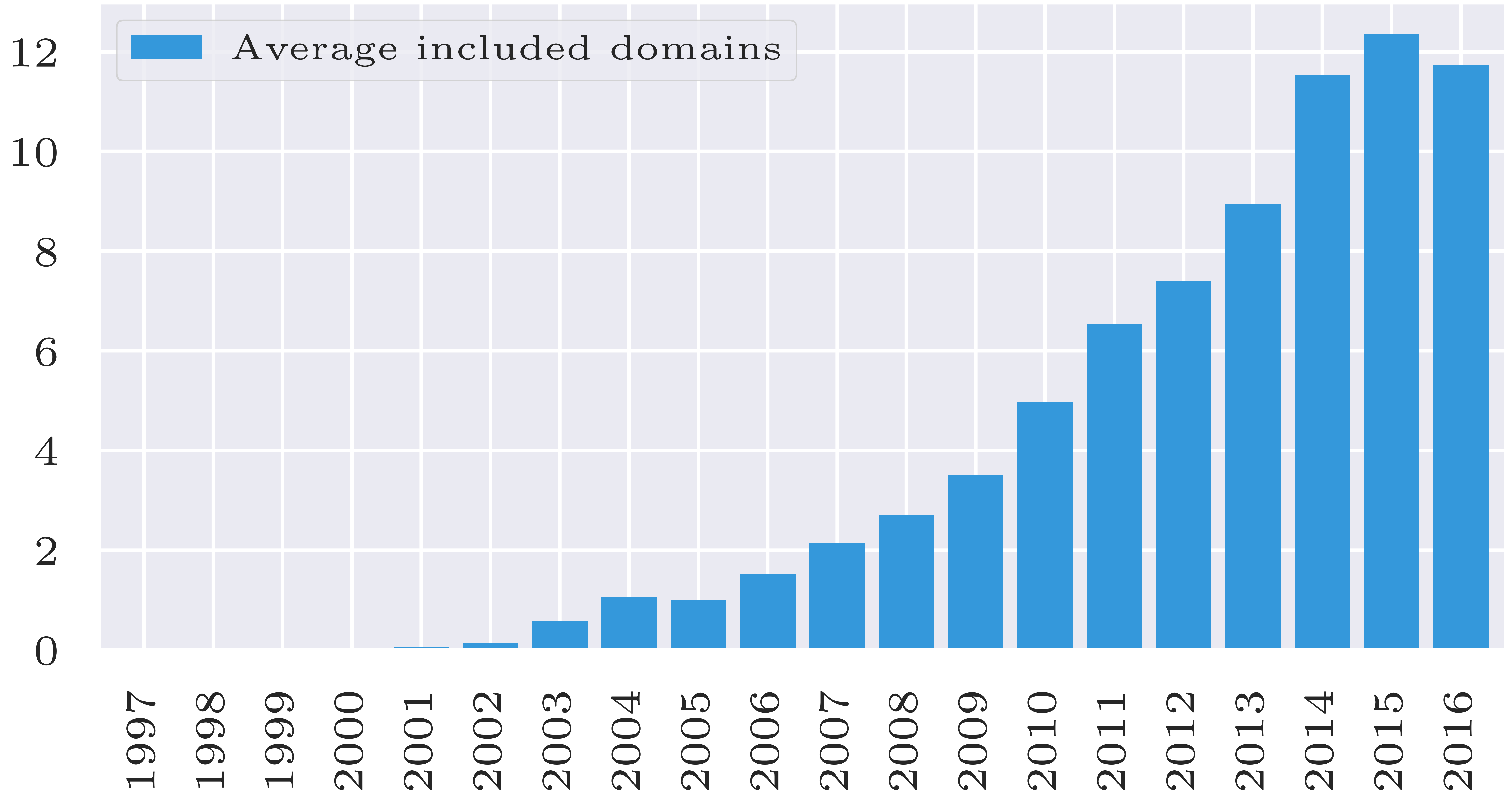
```
function getKwds() {
  var th_metadata = document.getElementsByTagName("meta");
  ...
}
var kwds = getKwds();
document.write('<iframe src="...&loc=' + kwds + '"></iframe>');
```

Mutation-based XSS

- Research result from the time of our first study
  - New potential source for CXSS: The DOM (!)
- Observation: Certain browser APIs mutate values
- Idea: use error-tolerant parsing to bypass filtering techniques
  - `element.innerHTML = '<img src="test.jpg" alt="``onload=xss()" />'`
  - `element.innerHTML   '<img alt=``onload=xss() src="test.jpg" >'`
- On first parse, nothing breaks
  - If innerHTML output is used again, we have an XSS
- Several examples shown by Heiderich et al. at CCS'13

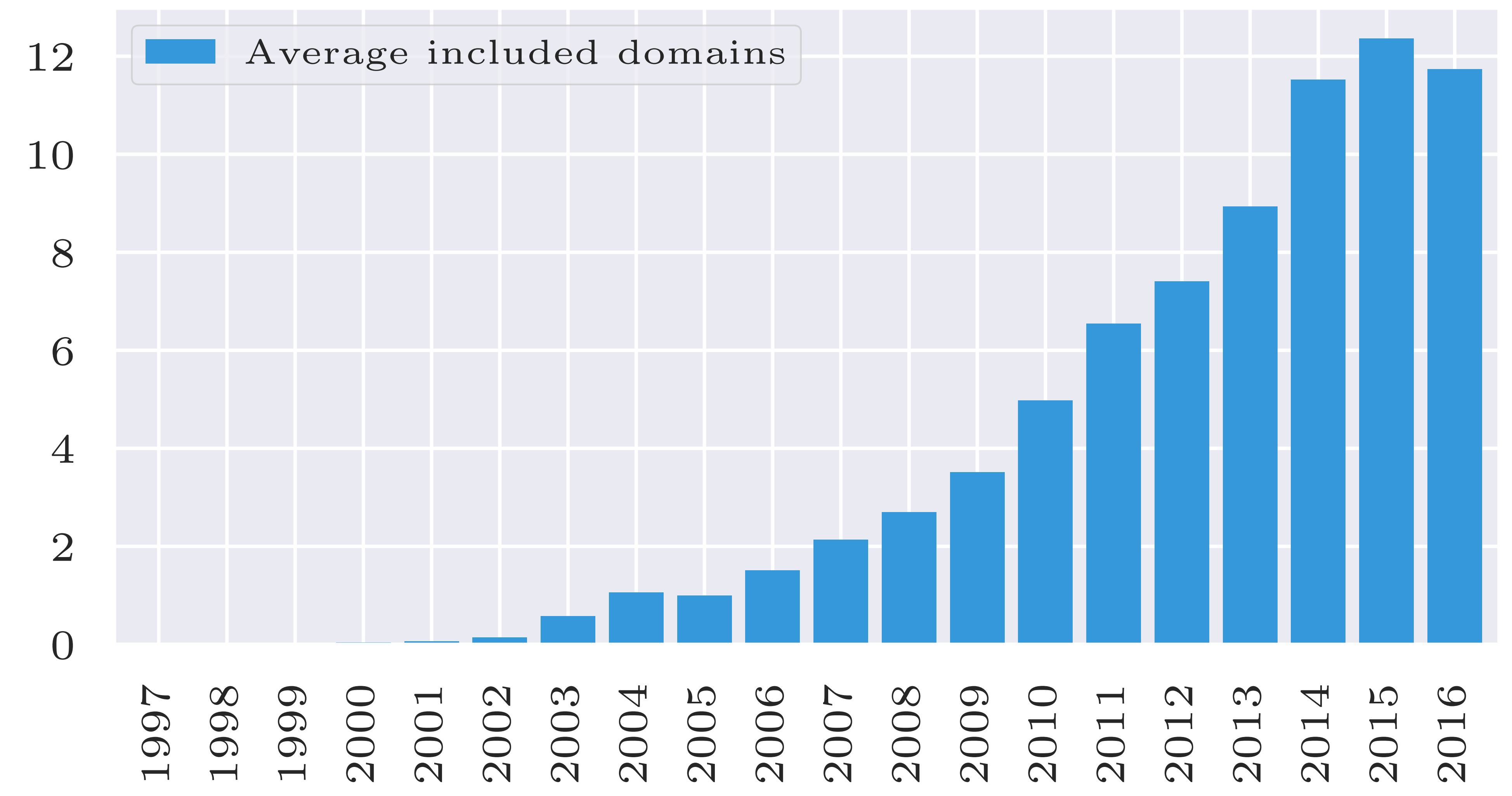
Study 3: The add that got me hacked

- Recall this figure?



3rd party involvement

- Investigated 1,273 real-world exploits
  - 835 caused by first-party code only
  - 273 caused by third-party code only
  - 165 as combination of first- and third-party code
- Additional problem
  - Script delegation
  - 3rd parties including further scripts
    - We have seen inclusion chains with lengths up to 8



Bonus Study: CXSS over the years [Usenix17]

Bonus Study: CXSS over the years [Usenix17]

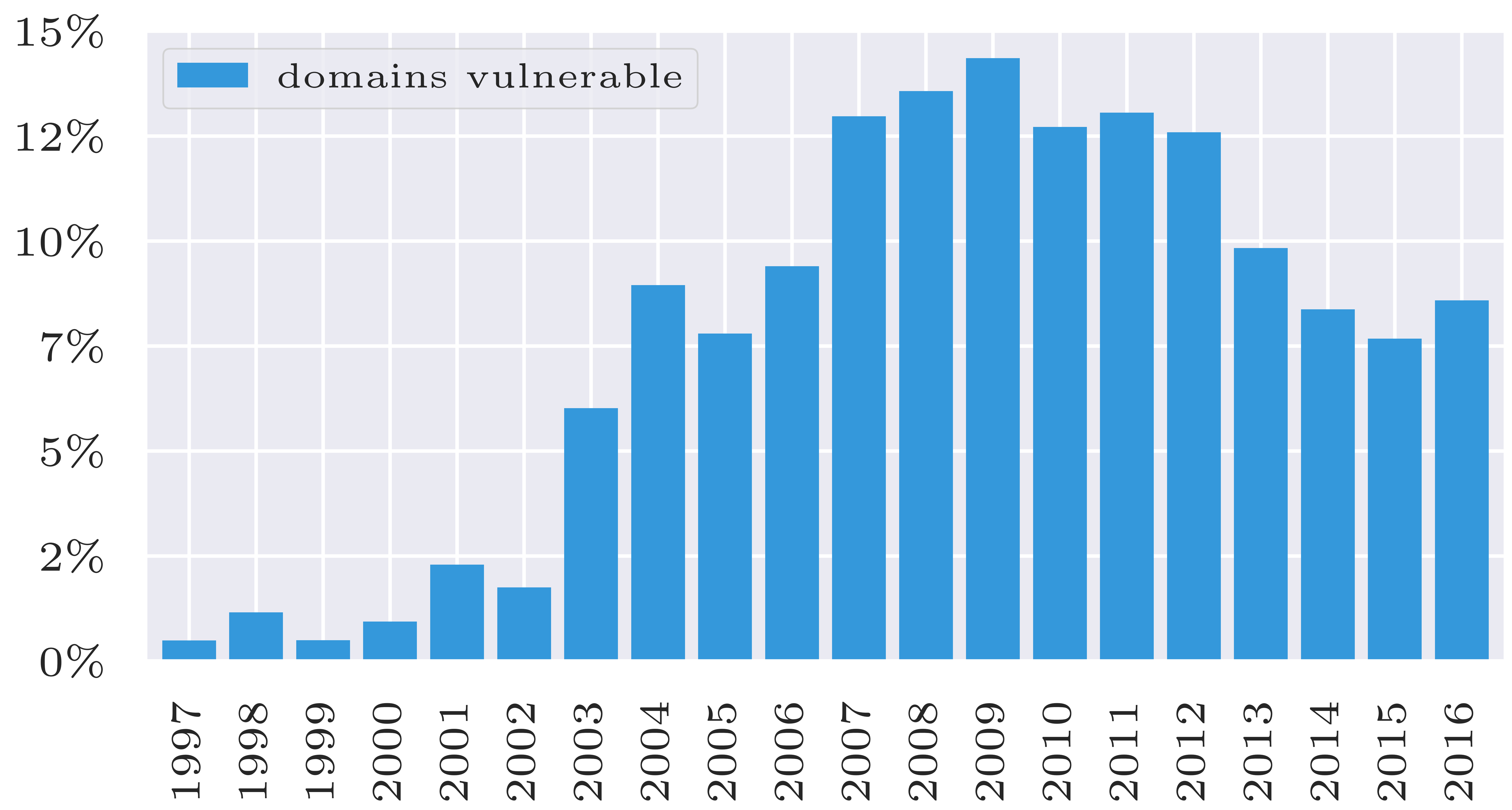
- Question:
  - Is this a new phenomena?
  - Was there a paradigm shift in application architecture that caused the rise of CXSS?

Bonus Study: CXSS over the years [Usenix17]

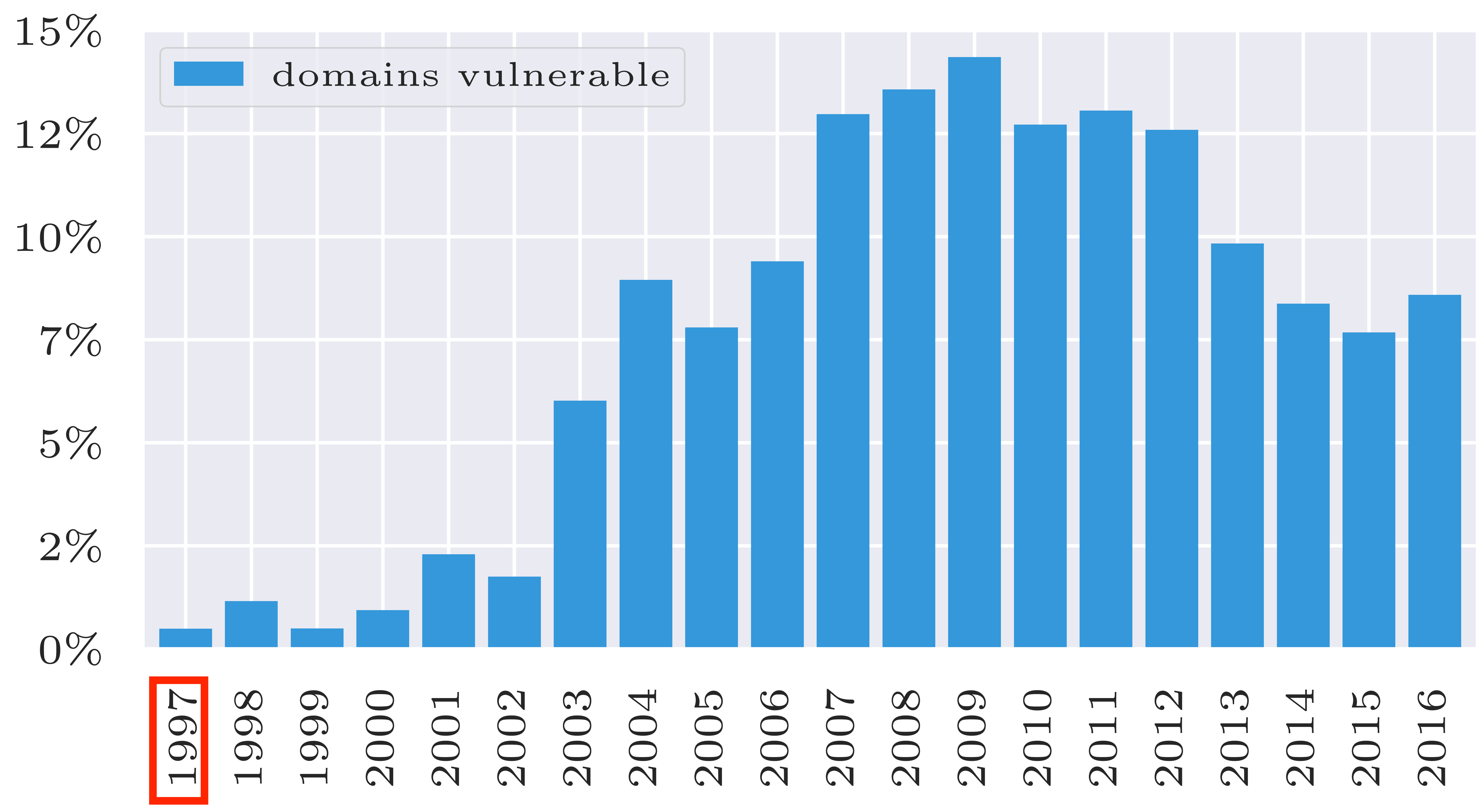
- Question:
  - Is this a new phenomena?
  - Was there a paradigm shift in application architecture that caused the rise of CXSS?
- Enter archive.org
  - Full record of client-side code
  - Especially relevant for us: Full JS code
  - Hence, we can security test the past :)

IAS INSTITUTE FOR APPLICATION SECURITY

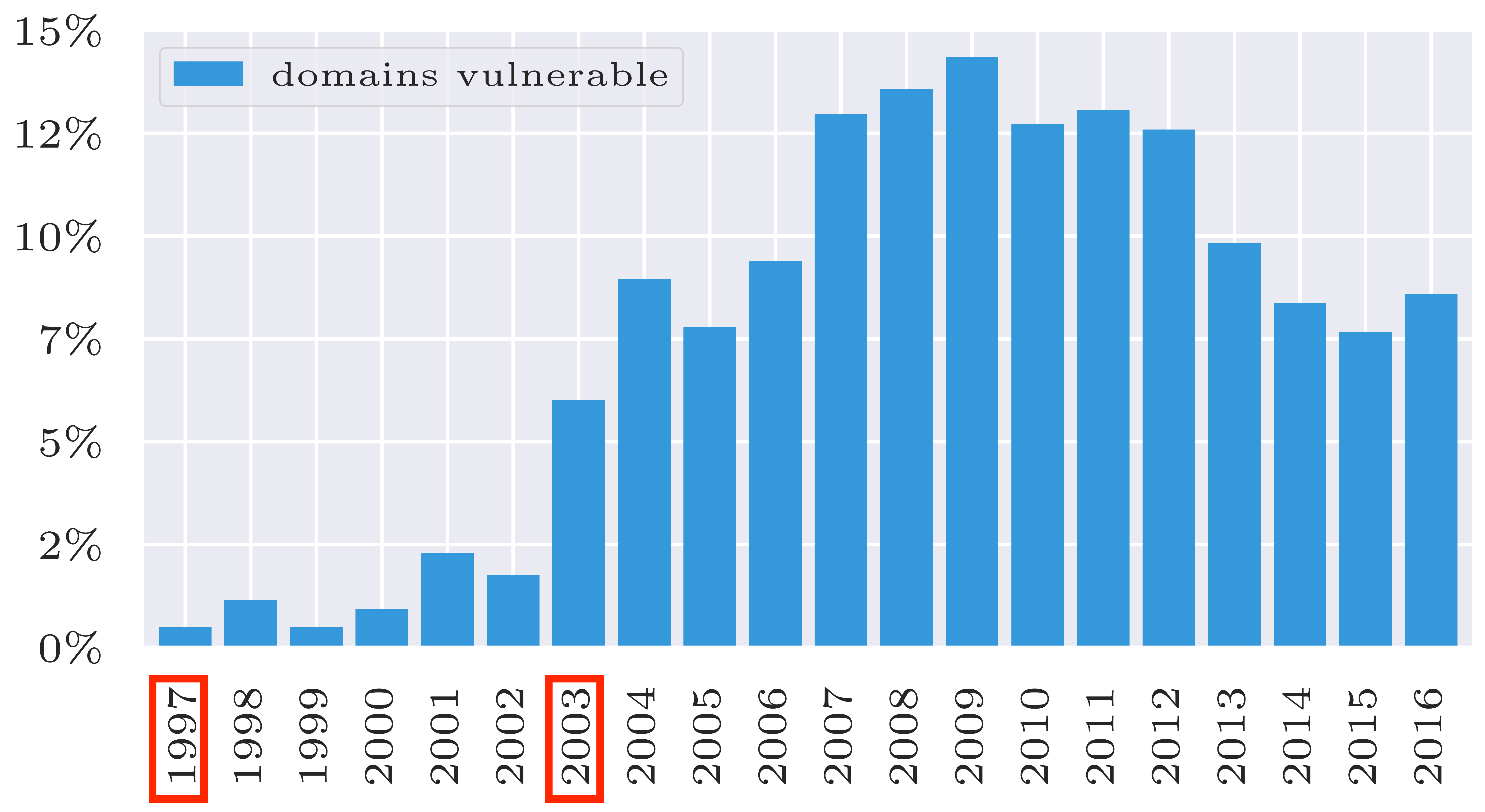
Client-Side Cross-Site Scripting over the ages



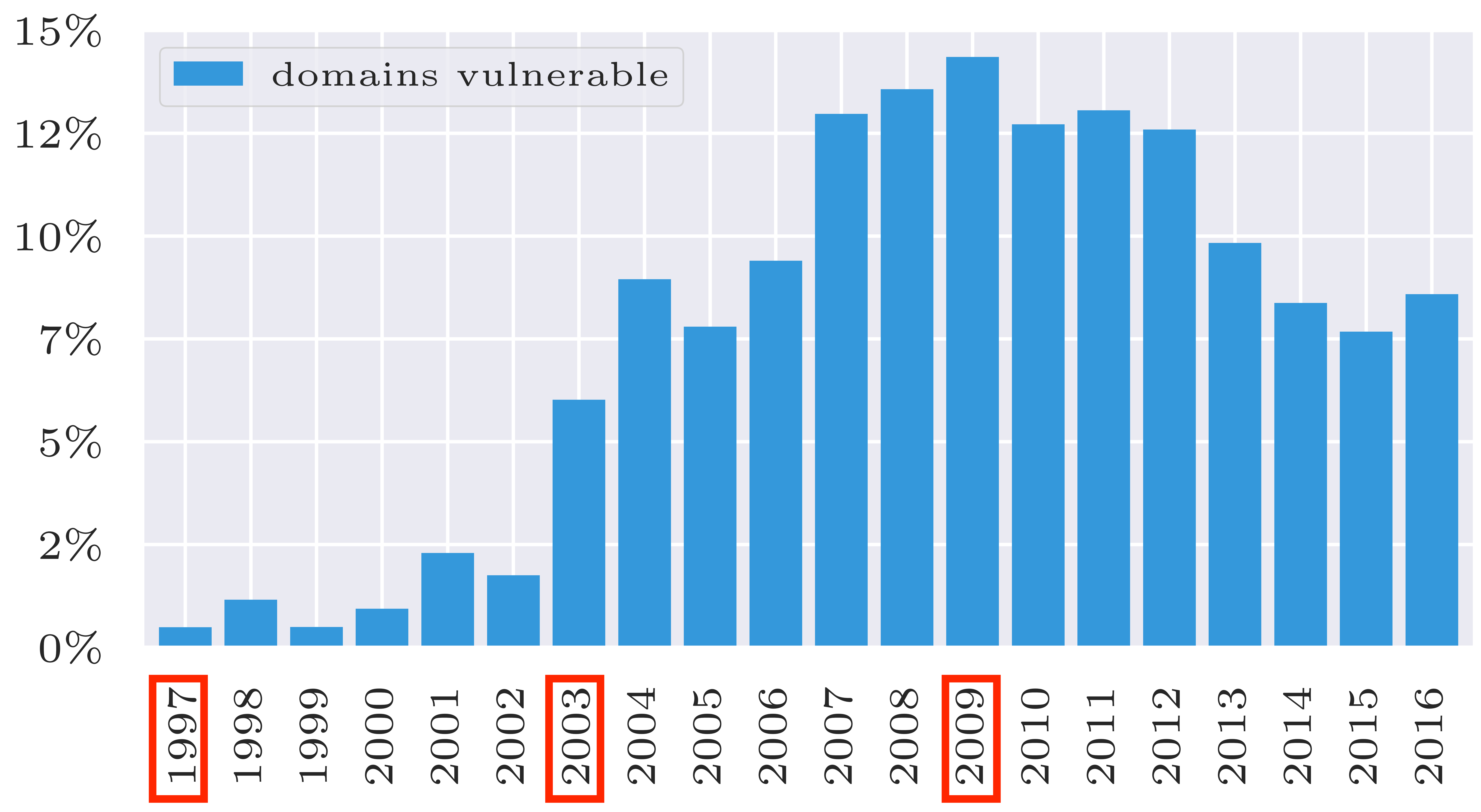
Client-Side Cross-Site Scripting over the ages



Client-Side Cross-Site Scripting over the ages



Client-Side Cross-Site Scripting over the ages



Persistent Client-side XSS

Persistent CXSS

Persistent CXSS

- Server-side XSS is partitioned in two classes
  - Reflected XSS
  - Stored/persistent XSS

Persistent CXSS

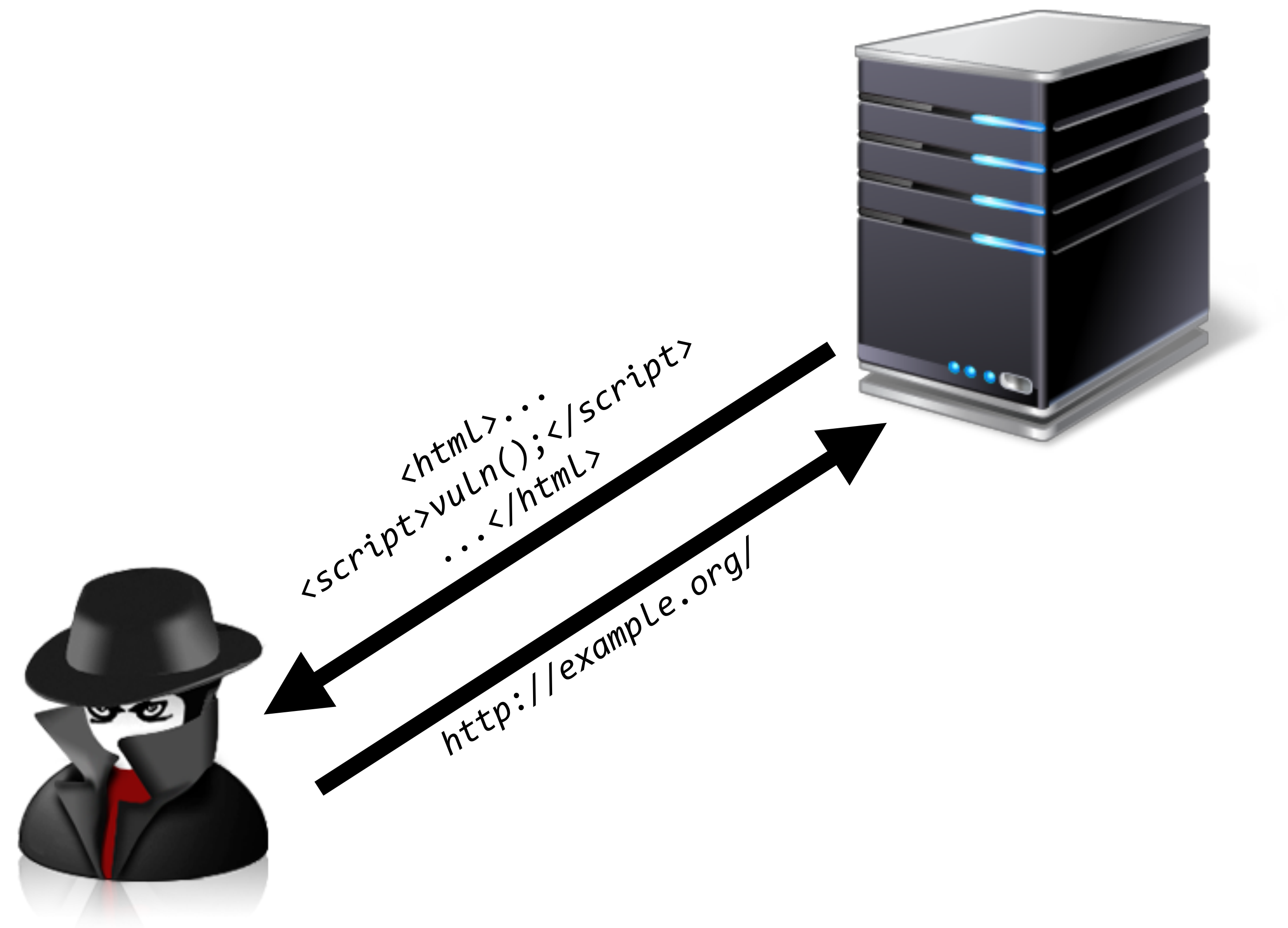
- Server-side XSS is partitioned in two classes
  - Reflected XSS
  - Stored/persistent XSS
- These dimensions also apply to the client-side
  - Up to this point, we only discussed *reflected CXSS*

Persistent CXSS

- Server-side XSS is partitioned in two classes
  - Reflected XSS
  - Stored/persistent XSS
- These dimensions also apply to the client-side
  - Up to this point, we only discussed *reflected CXSS*
- Introducing *Persistent CXSS*
  - The browser has mechanisms to persist data on the client-side
  - This data can be read by JavaScript
  - Thus, flows from the browser's storage into the DOM could lead to code injection

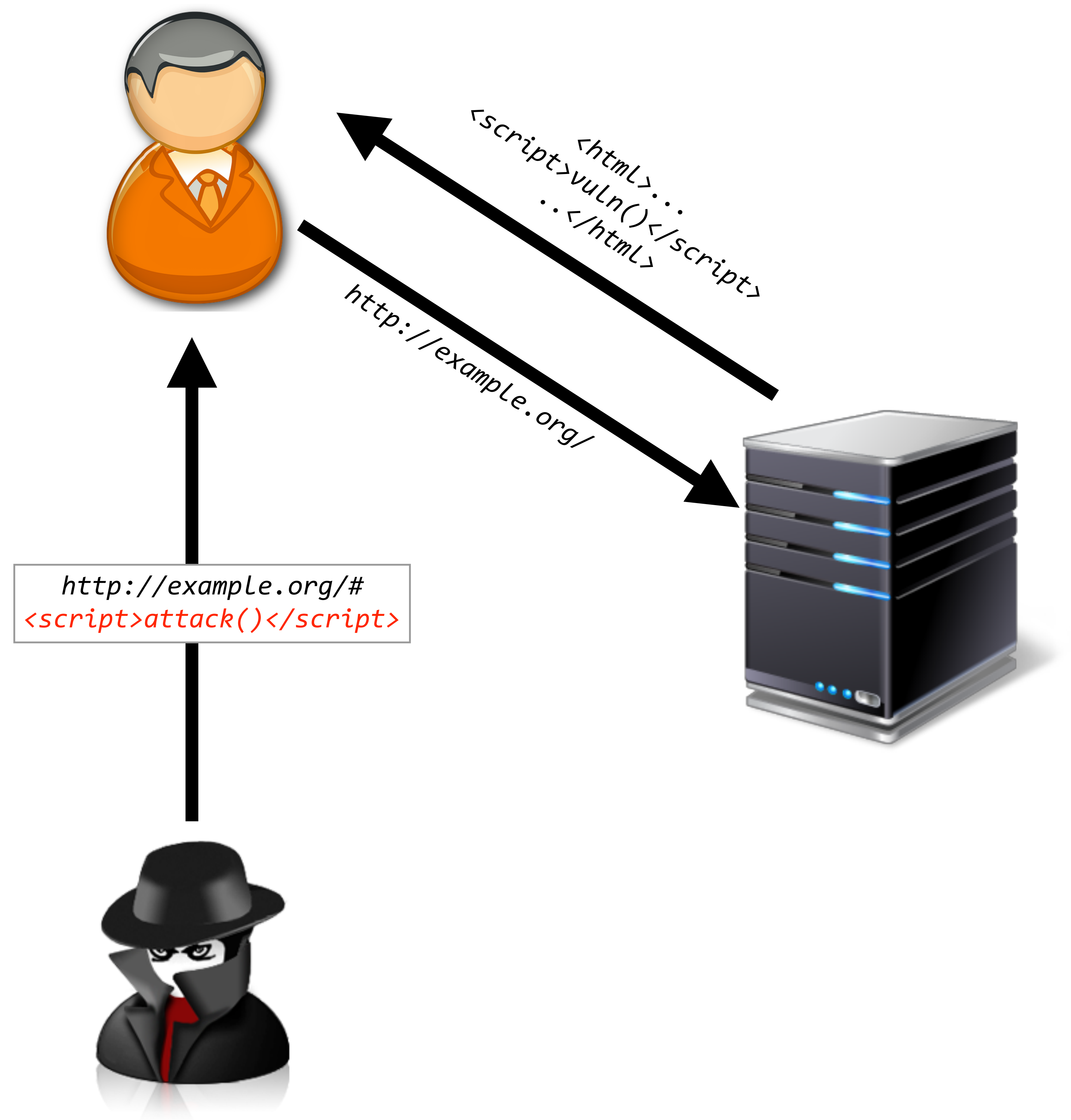
Persistent Client-Side Cross-Site Scripting

1. Attacker analyzes client-side JavaScript code for vulnerabilities
   - searches for unfiltered usage of attacker-controllable data (e.g., URL), **flowing to persistent storage**
   - searches for execution of persistent storage
     - example: cookie stores first visited URL, is used in eval statement later



Persistent Client-Side Cross-Site Scripting

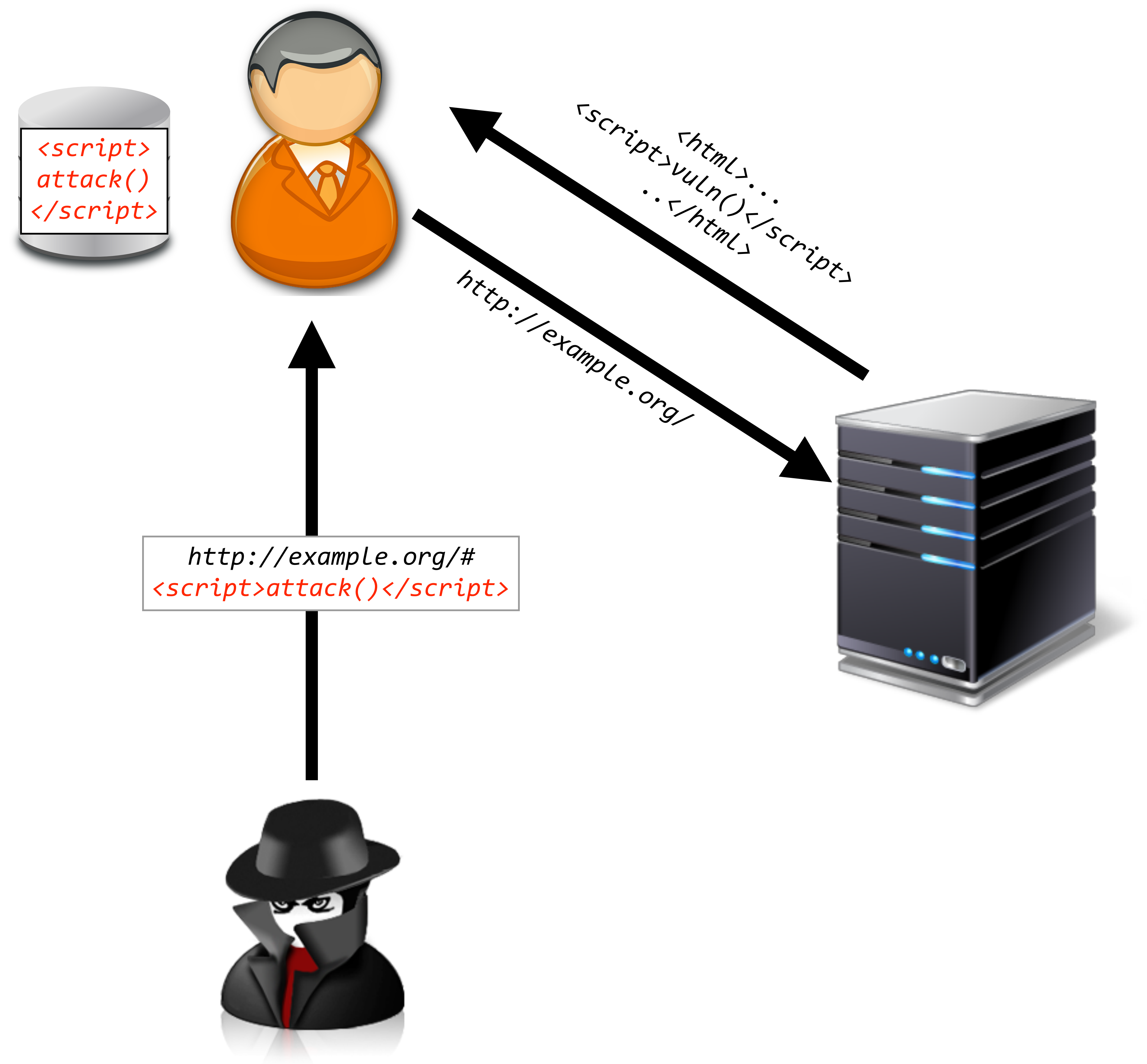
1. Attacker analyzes client-side JavaScript code for vulnerabilities
   - searches for unfiltered usage of attacker-controllable data (e.g., URL), **flowing to persistent storage**
   - searches for execution of persistent storage
     - example: cookie stores first visited URL, is used in eval statement later
2. Attacker tricks victim into visiting URL with payload, e.g., in fragment
   - data-persisting JavaScript is delivered to client
   - exploit payload is stored in persistent storage



<html>...<script>vuln()</script>...</html>
http://example.org/
http://example.org/#<script>attack()</script>

Persistent Client-Side Cross-Site Scripting

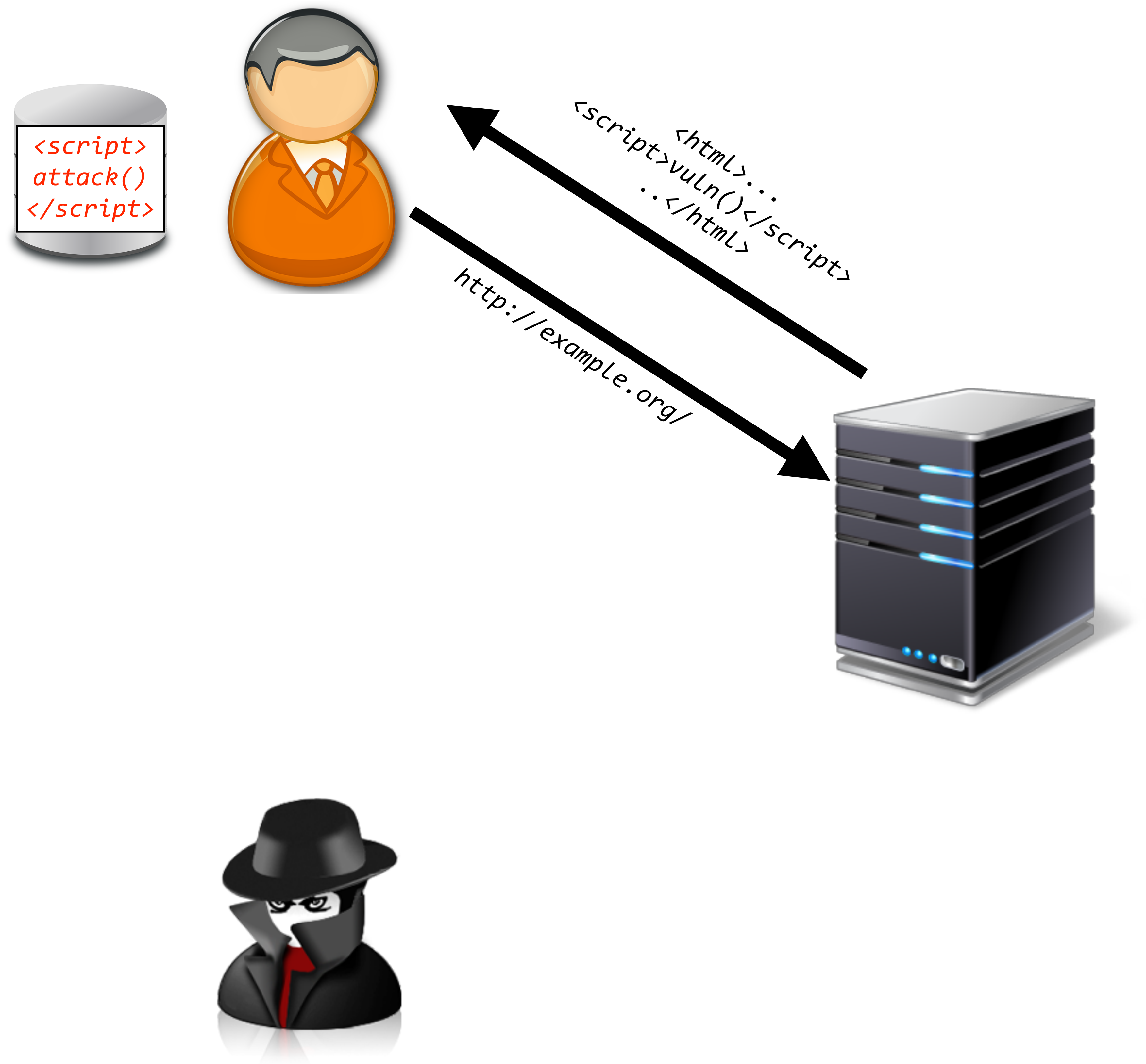
1. Attacker analyzes client-side JavaScript code for vulnerabilities
   - searches for unfiltered usage of attacker-controllable data (e.g., URL), **flowing to persistent storage**
   - searches for execution of persistent storage
     - example: cookie stores first visited URL, is used in eval statement later
2. Attacker tricks victim into visiting URL with payload, e.g., in fragment
   - data-persisting JavaScript is delivered to client
   - exploit payload is stored in persistent storage



<script>attack()</script>
http://example.org/
<html>...<script>vuln()</script>...</html>
http://example.org/#<script>attack()</script>

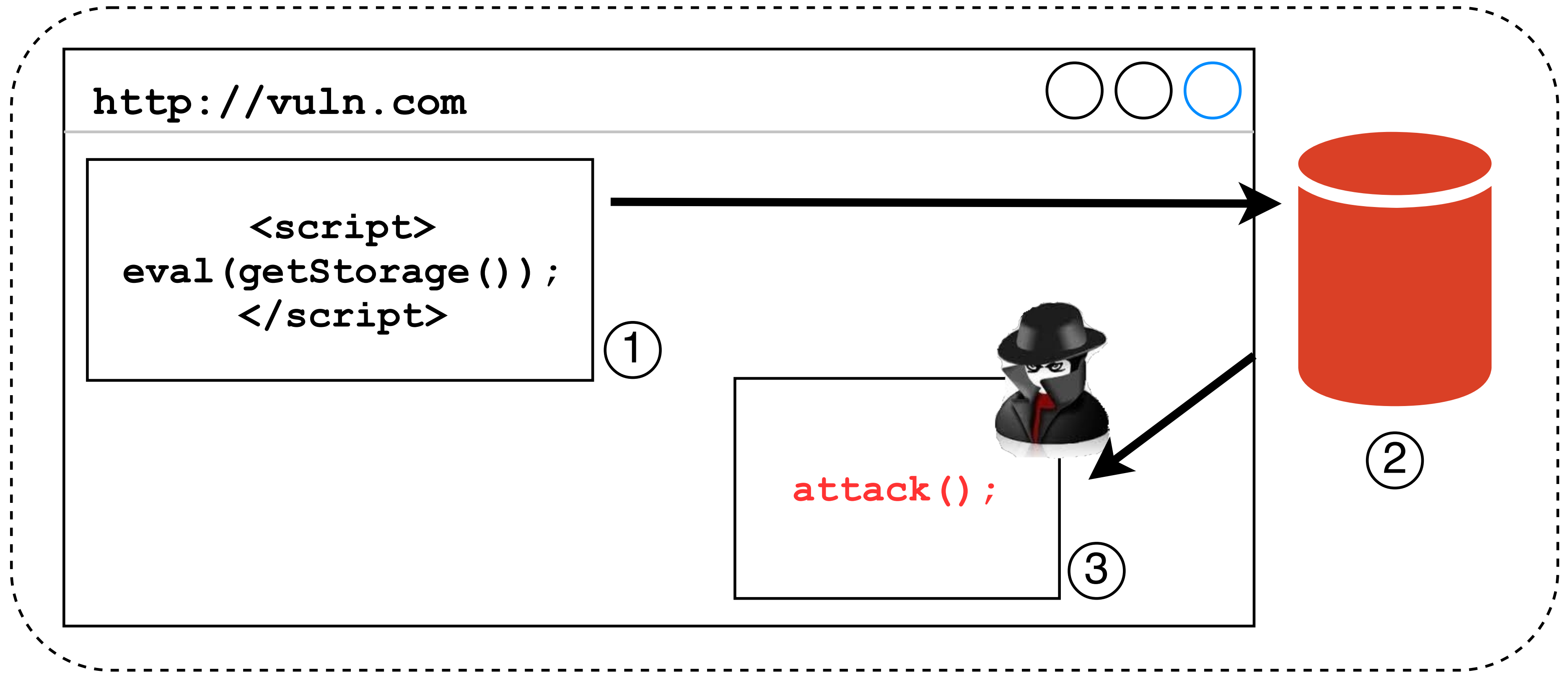
Persistent Client-Side Cross-Site Scripting

1. Attacker analyzes client-side JavaScript code for vulnerabilities
2. Attacker tricks victim into visiting URL with payload, e.g., in fragment
3. On every page visit, payload is extracted from persistent storage
   - flow from storage to execution sink
   - malicious payload is executed



Persistent CXSS - Sources

- The set of relevant sources deviates from the known pattern
  - (The sinks remain the same)
- document.cookie
  - JavaScript API to read and write cookies that apply to the web document
- document.localStorage
  - Key/value store for JavaScript
- IndexedDB
  - structured storage API with indexing support



Persistent CXSS in the wild [NDSS19]

- Dedicated empirical study
  - Alexa Top 5000
  - Crawl two levels deep
  - In total 12,489,576 web documents analysed

- In 8% of the examined origins we found at least one exploitable flow

	Cookie			Local Storage		
Sink	Total	Plain	Expl.	Total	Plain	Expl.
HTML	496	319	132	234	226	105
JavaScript	547	470	72	392	385	108
Script Src	1,385	533	17	626	297	11
Total	1,645	906	213	941	654	222

Key differences between reflected and persistent CXSS

- Persistent CXSS requires two flows
  - One into the storage
  - One form storage into the DOM
- But:
  - The two flows don’t have to be connected
  - They don’t even have to occur in the same web document
- In fact, persistent CXSS can be abused to transform a reflected XSS into the persistent variant
- Furthermore, Cookie-flows expand the attacker model considerably
  - Cookie tossing, network attackers, …

Resolving persistent CXSS

- Unlike reflected CXSS, persistent CXSS is a result of indented functionality
  - Local caching/maintaining of information
- A problem arises, if the cached information is required to contain *code* portions
  - Cached JavaScript libraries, cached HTML snippet, cached configuration, containing, e.g., URLs
- In such cases encoding breaks the functionality

```
var hostname = localStorage.getItem("hostname");
var script = document.createElement("script");
script.src = hostname + "foo.js";
document.body.appendChild(script);
```

- Cleansing the storage of affected users is difficult

```
ofType = what
return first === 1 && last === 0 ?

// Shortcut for :nth-*(n)
function( elem ) {
    return !!elem.parentNode;
} :

function( elem, context, xml ) {
    var cache, outerCache, node, diff, nodeIndex, start,
        dir = simple !== forward ? "nextSibling" : "
            previousSibling",
        parent = elem.parentNode,
        name = ofType && elem.nodeName.toLowerCase(),
        useCache = !xml && !ofType;
```

Combating CXSS

Combating CXSS: Secure programming (I)

- Problems originate from use of insecure APIs
  - eval, document.write, innerHTML
  - and the use of user-provided input in them

- Depending on the context, functionally equivalent APIs exist
  - *document.createElement, element.innerText*
  - *JSON.parse*

```
function writeURLInsecure() {
  document.write("<p>The current URL is: "
    + location.href + "</p>");
}
```

```
function writeURLSecure() {
  var p = document.createElement("p");
  p.innerText = "The current URL is: " + location.href;
  document.write(p.outerHTML);
}
```

Combating CXSS: Secure programming (II)

```
function loadAdvertisementInsecure() {
  document.write("<script src='http://ad.com/?referrer=" + location.href + "'></script>");
}
```

```
function loadAdvertisementSecure() {
  var script = document.createElement("script");
  script.src = 'http://ad.com/?referrer=' + location.href;
  document.body.appendChild(script);
}
```

- `element.src` ensures that attacker-controllable data can only be in src attribute

Combating CXSS: Secure programming (III)

```
function parseJSONInsecure(json) {
  var object = eval(json);
}
```

```
function parseJSONSecure(json) {
  var object = JSON.parse(json);
}
```

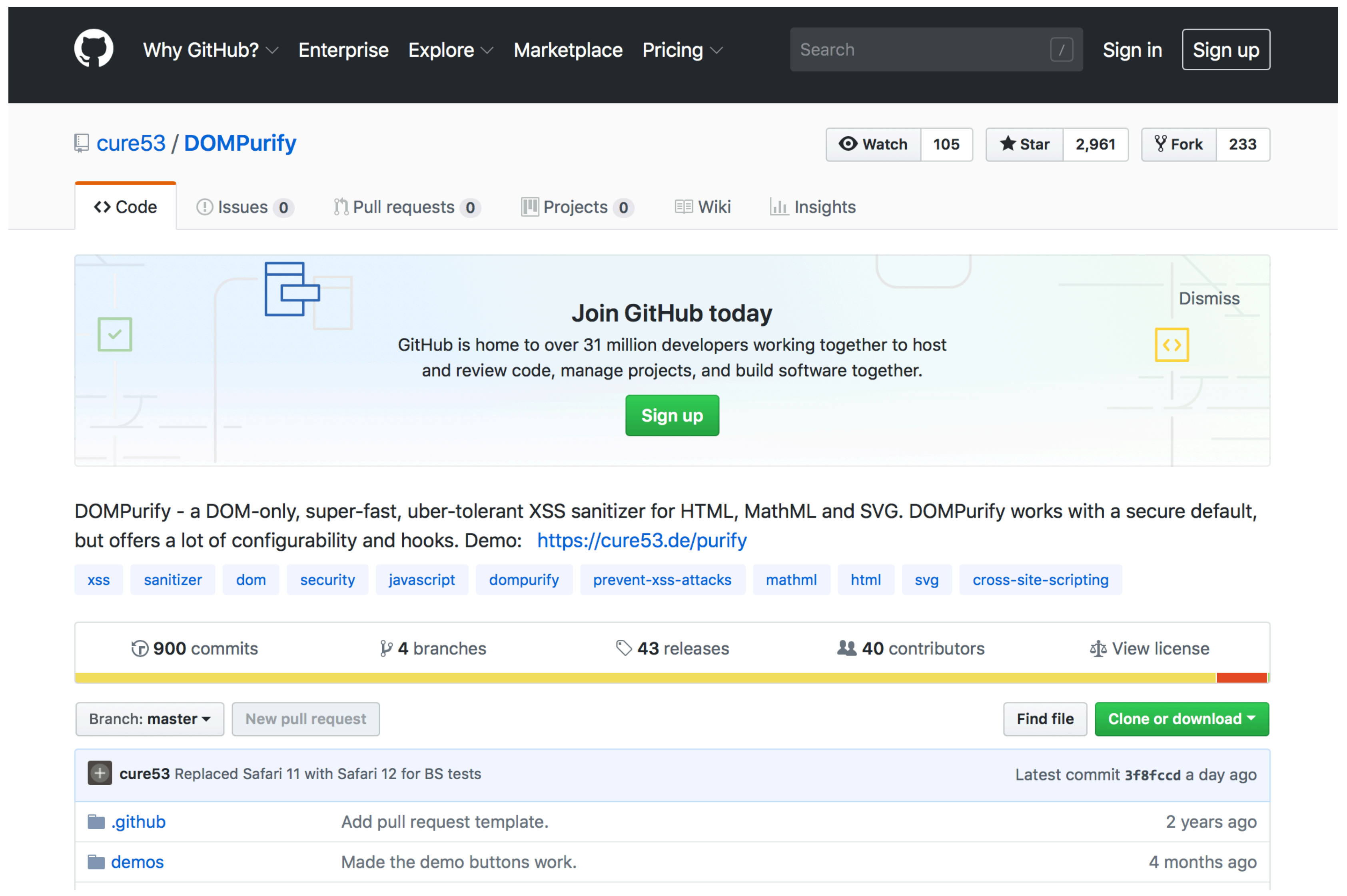
```
function registerGlobalInsecure(key, value) {
  eval(key + " = '" + value + "'");
}
```

```
function registerGlobalSecure(key, value) {
  window[key] = value;
}
```

- Depending on the desired use, either
  - use `JSON.parse`
  - use `object[key] = value` notion

Combating CXSS: Client-side sanitization

- On the server-side the main defence against XSS is output encoding
- The same works on the client-side
- Unfortunately, the browser does not offer native encoding/sanitizing functionality
  - `URIencode()` and `escape()` do not catch all cases
  - R.I.P. IE's `toStaticHTML()`
- Building your own encoder is not without pitfalls
  - Currently best option: DOMPurify from Cure53
  - https://github.com/cure53/DOMPurify



Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

DOMPurify - a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG. DOMPurify works with a secure default, but offers a lot of configurability and hooks. Demo: https://cure53.de/purify

xss sanitizer dom security javascript dompurify prevent-xss-attacks mathml html svg cross-site-scripting

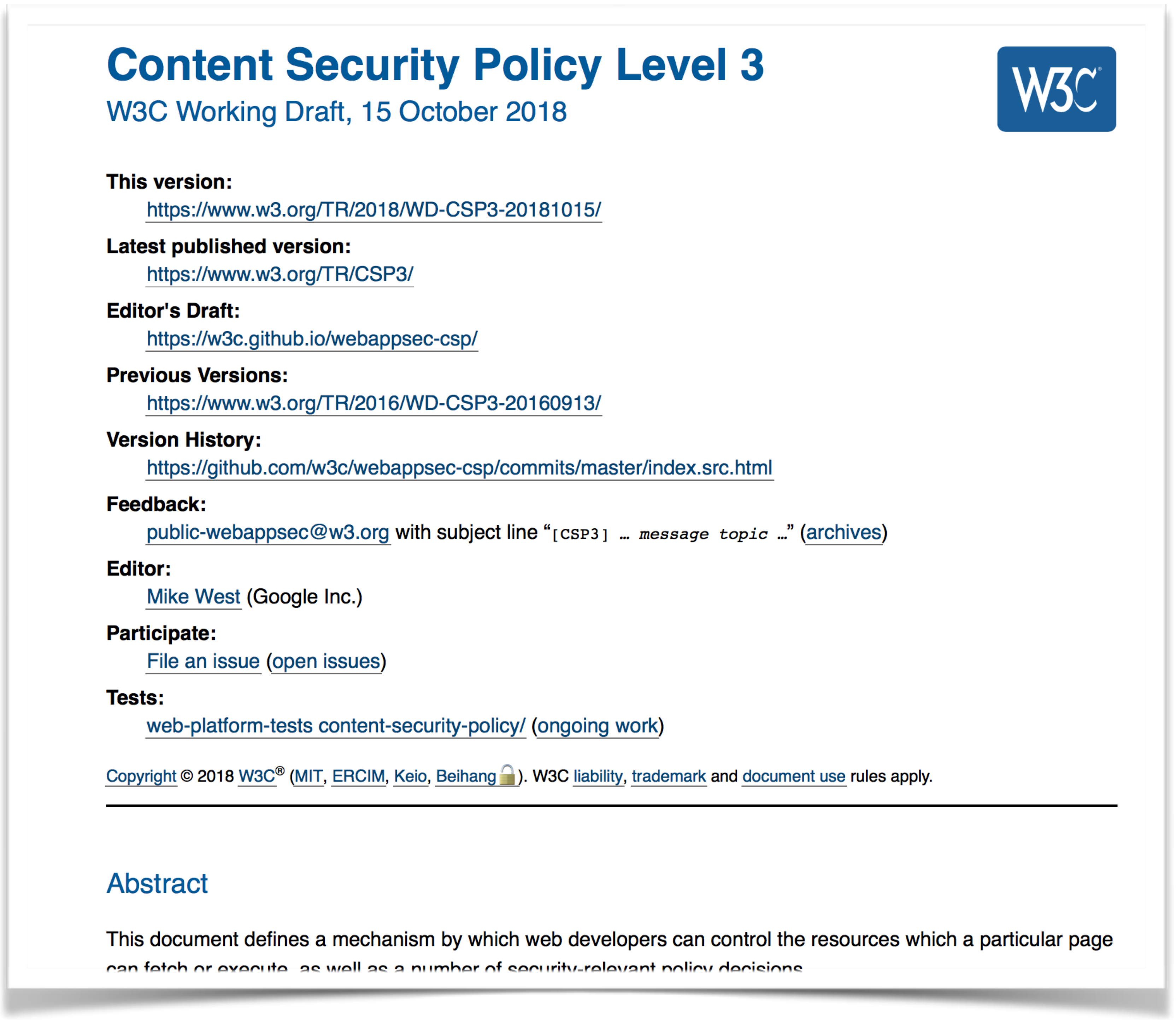
900 commits | 4 branches | 43 releases | 40 contributors | View license

cure53 Replaced Safari 11 with Safari 12 for BS tests — Latest commit 3f8fccd a day ago

.github	Add pull request template.	2 years ago
demos	Made the demo buttons work.	4 months ago

Combating CXSS: Content Security Policy

- Content Security Policy is a browser-based standard to stop XSS exploits
  - Mitigates exploits but does not stop injection
  - Declarative policy, set by the server, enforced by the browser
- A strong CSP is highly effective against CXSS
  - But don't allow: unsafe-line, unsafe-eval and strict-dynamic
- More details tomorrow at 11:00 in my other talk



Content Security Policy Level 3

W3C Working Draft, 15 October 2018

This version:
https://www.w3.org/TR/2018/WD-CSP3-20181015/

Latest published version:
https://www.w3.org/TR/CSP3/

Editor's Draft:
https://w3c.github.io/webappsec-csp/

Previous Versions:
https://www.w3.org/TR/2016/WD-CSP3-20160913/

Version History:
https://github.com/w3c/webappsec-csp/commits/master/index.src.html

Feedback:
public-webappsec@w3.org with subject line "`[CSP3] … message topic …`" (archives)

Editor:
Mike West (Google Inc.)

Participate:
File an issue (open issues)

Tests:
web-platform-tests content-security-policy/ (ongoing work)

Copyright © 2018 W3C® (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and document use rules apply.

Abstract

This document defines a mechanism by which web developers can control the resources which a particular page can fetch or execute, as well as a number of security-relevant policy decisions.

Combating CXSS: Trusted Types

- New proposal from Google
  - Typed DOM APIs that do not accept strings
  - Instead “templates” have to be created
  - Secure creation of these templates allow reliable code audit

```
const templatePolicy = TrustedTypes.createPolicy('template', {
  createHTML: (templateId) => {
    const tpl = templateId;
    if (/^[0-9a-z-]$/.test(tpl)) {
      return `<link rel="stylesheet" href="./templates/${tpl}/style.css">`;
    }
    throw new TypeError();
  }
});

const html = templatePolicy.createHTML(location.hash.match(/tplid=([^;&]*)/)[1]);
// html instanceof TrustedHTML
document.head.innerHTML += html;
```

Summary

Summary

- Client-side XSS is surprisingly common
  - In fact, Google considers it to be the most relevant class of XSS problems
- The JavaScript execution model makes finding non-trivial flows difficult
- Persistent CXSS adds another dimension to the problem
  - Backdoored intended functionality, extended attacker model
- Methodology for secure development and mitigation of CXSS exist
  - But have to applied correctly

Q&A