Security in knowledge

Relax Everybody: HTML5 Is Securer Than You Think

Martin Johns (@datenkeller)

SAP AG

RSA CONFERENCE EUROPE 2013

Session ID: ADS-W08

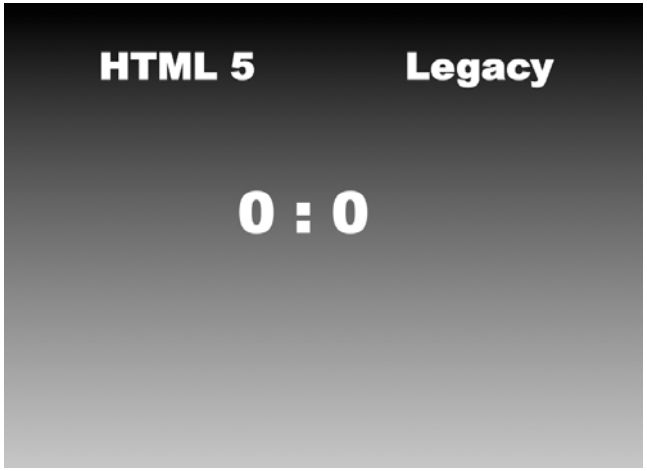
Session Classification: Advanced

Motivation

- For some reason, there is a preconception that HTML5 is terribly insecure…
- This is unfortunate, as (probably for the first time) new browser features come with a well designed security model
- In this talk, I will compare selected HTML5 technologies with their legacy counterparts

Motivation

- For some reason, there is a preconception that HTML5 is terribly insecure…
- This is unfortunate, as (probably for the first time) new browser features come with a well designed security model
- In this talk, I will compare selected HTML5 technologies with their legacy counterparts
- … and I will keep score



Outline

- Technical background
- Client-side cross-domain communication
- In-browser communication
- Client-side persistence
- ClickJacking protection
- Bonus track: The browser's new security capabilities

Technical Background

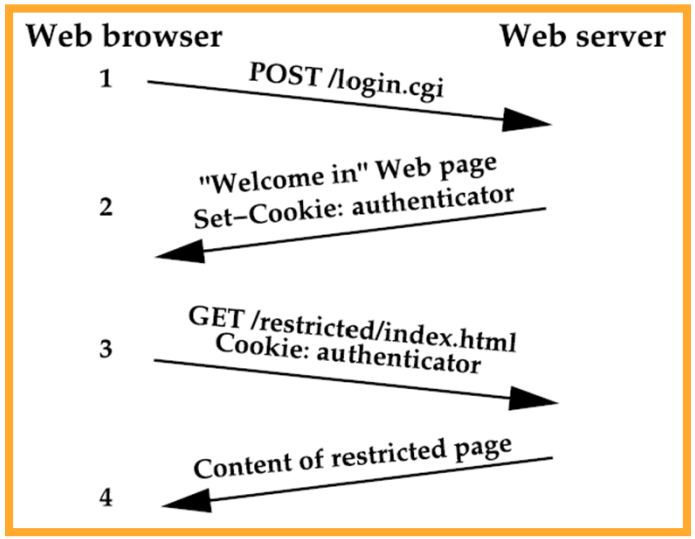
Security in knowledge

#RSAC

RSA CONFERENCE EUROPE 2013

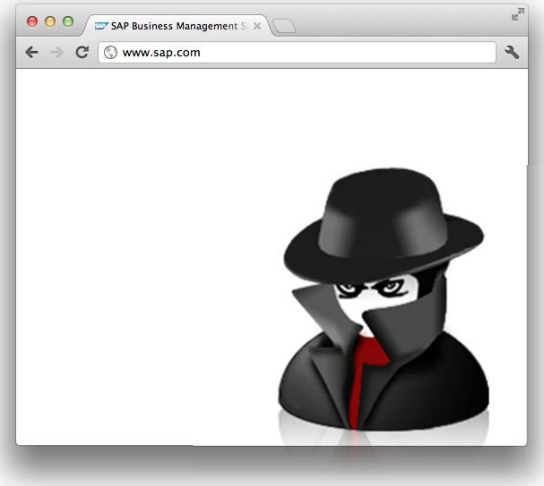
Web authentication tracking

- The browser maintains the authenticated state automatically
- After the initial authentication, everything is transparent
- Techniques:
  - Authenticated session cookies
    - Including all currently used social login techniques
  - HTTP authentication
  - Client-side SSL certificates



Introducing: The Attackers

- The Web Attacker
  - The predominant attacker model of this talk
  - Is able to display Web documents in the victim's browser
    - E.g., through the means of a nicely done Web page with cat content
    - The code of this page runs in your (!) authentication context
- The Network Attacker
  - Resides on the network link between the browser and the server
  - Can alter/observe unprotected traffic
  - Protection: SSL

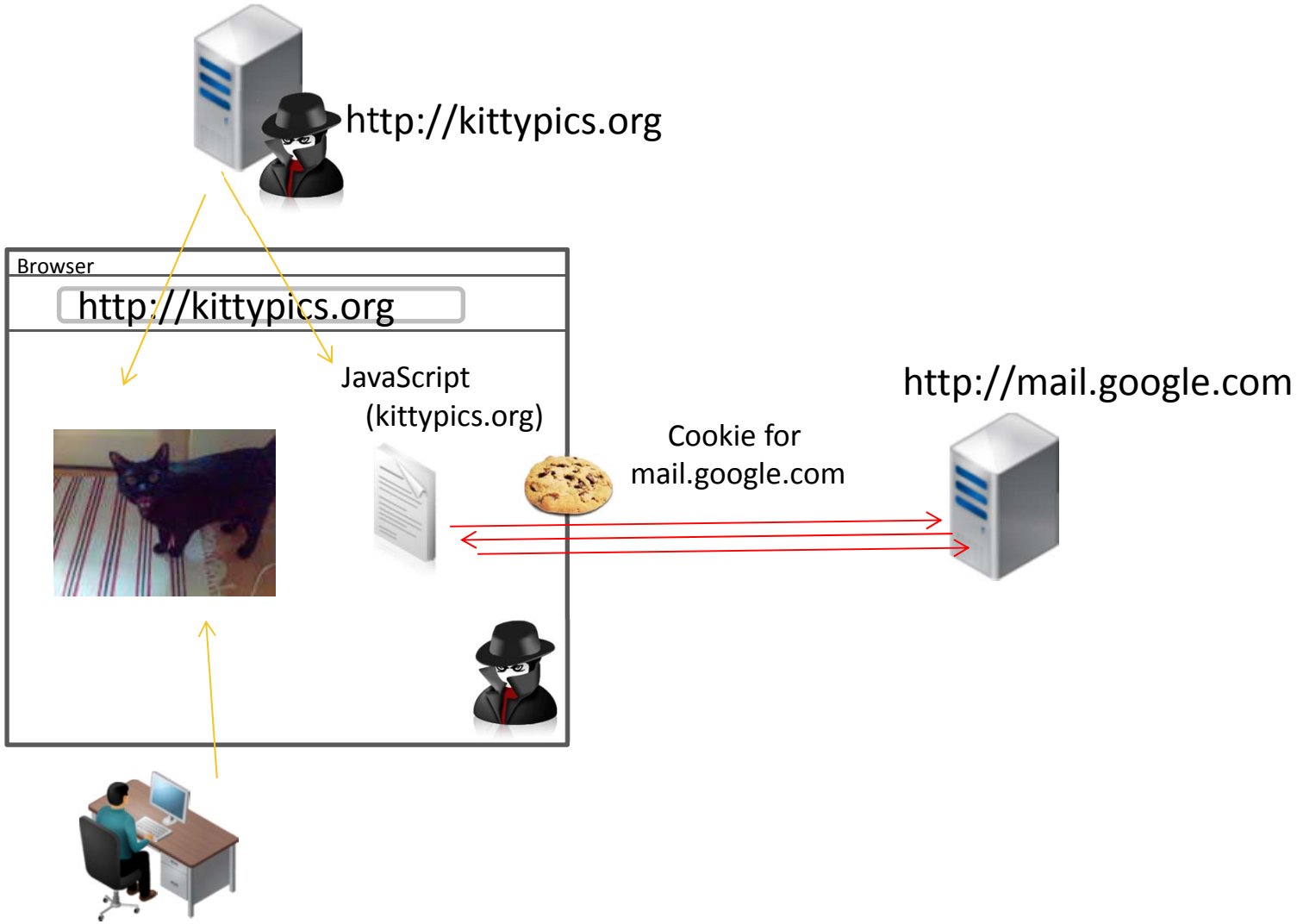


The Same-Origin Policy (SOP)

- The **only** client-side security measure
  - Defines basic access rights in HTTP
  - Two elements have the "same origin" if the
    - protocol, port, and host
  - are the same for both elements
- Confines active code to Web documents of the same "owner"

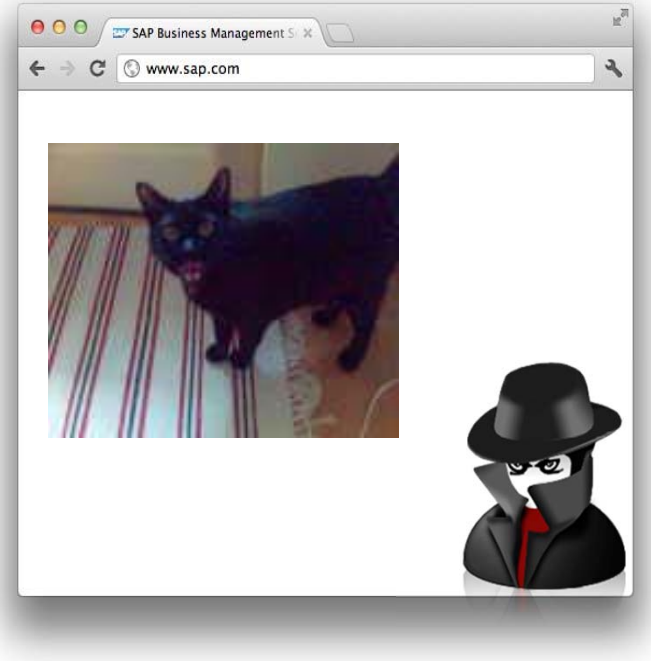
Originating document	Accessed document	Non–IE browser	Internet Explorer
http://example.com/**a/**	http://example.com/**b/**	Access okay	Access okay
http://example.com/	http://**www.**example.com/	Host mismatch	Host mismatch
http://example.com/	**https**://example.com/	Protocol mismatch	Protocol mismatch
http://example.com**:81**/	http://example.com/	Port mismatch	Access okay

A World Without the SOP



A World Without the SOP

- In a world without the SOP, the Web attacker can
  - Read/write the contents of any (crossdomain) Iframe
  - Send state full, authenticated HTTP requests to any server
  - Read/write the locally stored information of any site



HTML5 and the SOP

- Interestingly enough, many HTML5 APIs allow **softening** the SOP
- Q: So HTML5 **is** a bad thing, isn't it?
- Short answer: No!
- Long answer: No, because the old way was worse
  - The HTML5 APIs satisfy a functional need, that predated them…

Client-side cross-domain communication

Security in knowledge

#RSAC

RSA CONFERENCE EUROPE 2013

The Problem

- **Developer:** I would like to offer cross-domain data providing service
  - The user's authentication context with the data provider is in the browser
  - Hence, the data is personalized, without the user's need to share his credentials
- **SOP:** No, no, no! You are not allowed to do so!
- **Developer:** Well, I will do it anyways...

Legacy Technique 1: JSONP

- HTML tags are not subject to the SOP
  - This includes the script-tag
    <script src="http://x-domain.host">
- JSONP
  - Offer an HTTP-endpoint which expects the name of a JavaScript callback function in one of its URL parameters
  - Generate a script file, which calls this callback function with the requested data as argument

JSONP: Example

Callback function definition

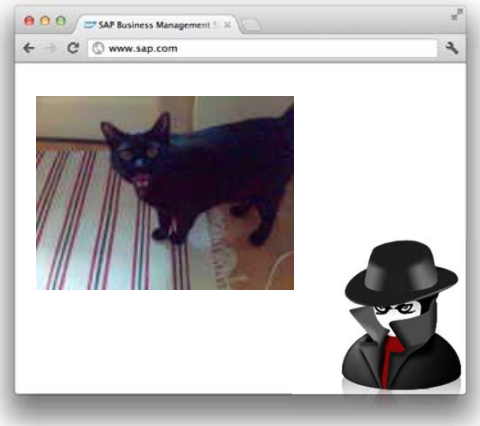
```
<script>
function processJSON (json) {
  // Do something with the JSON response
};
</script>

<script src='http://api.flickr.com/services/feeds/photos_public.gne?
jsoncallback=processJSON&tags=monkey&tagmode=any&format=json'></script>
```

Callback function passing

JSONP (in)Security

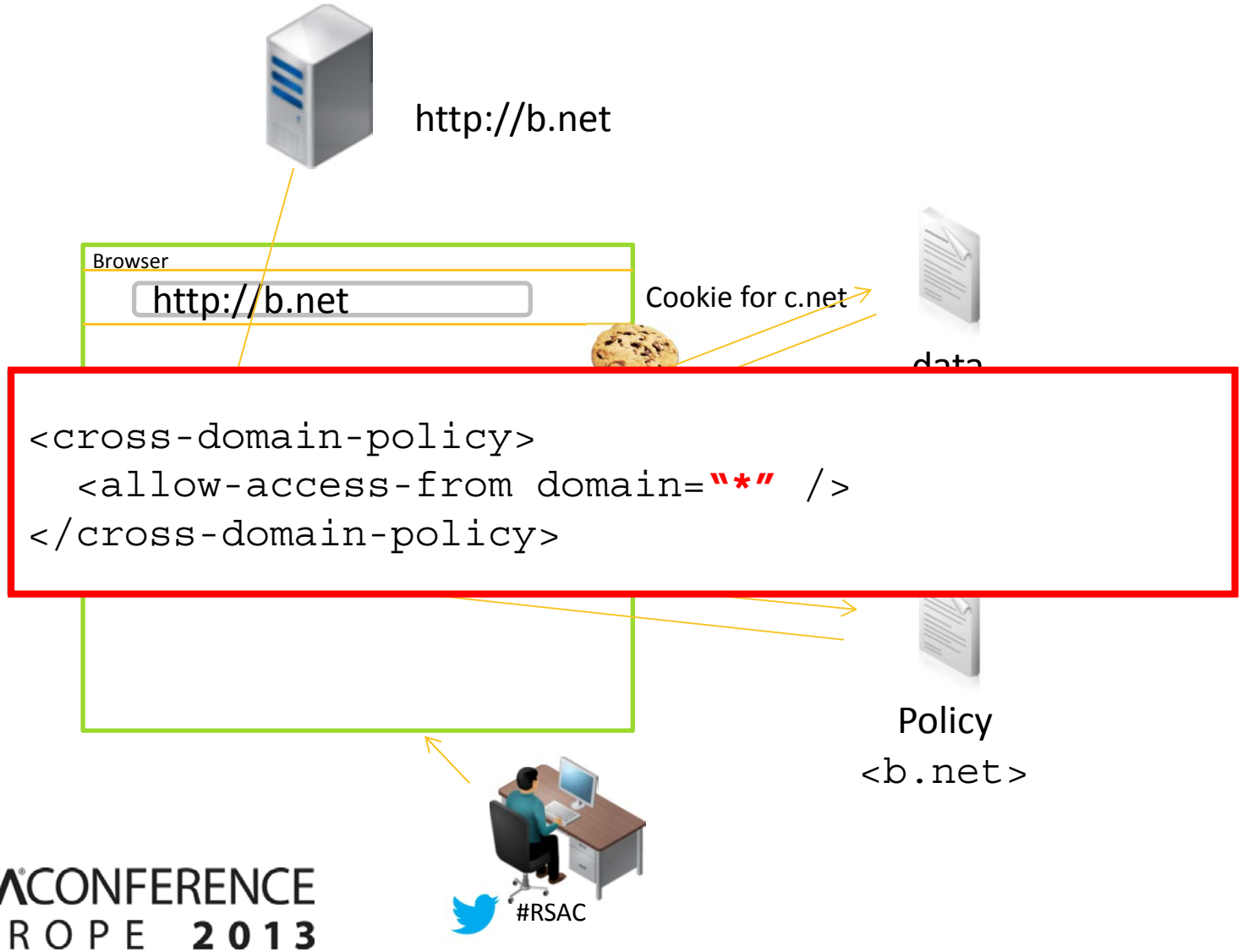
- JSONP is an valid option for **public** data
  - However, for private data not so much…
- The Web attacker can insert a script tag pointing to the JSONP interface in his site
- Through providing his own callback function, he receives the private data



Legacy Technique 2: Crossdomain.xml

- The call for crossdomain requests was first answered by Flash
- Through providing a policy file (crossdomain.xml) a site can widen its trust boundaries selectively
  - Whitelist approach
  - Sites listed in the policy are allowed to send/receive crossdomain HTTP requests

crossdomain.xml

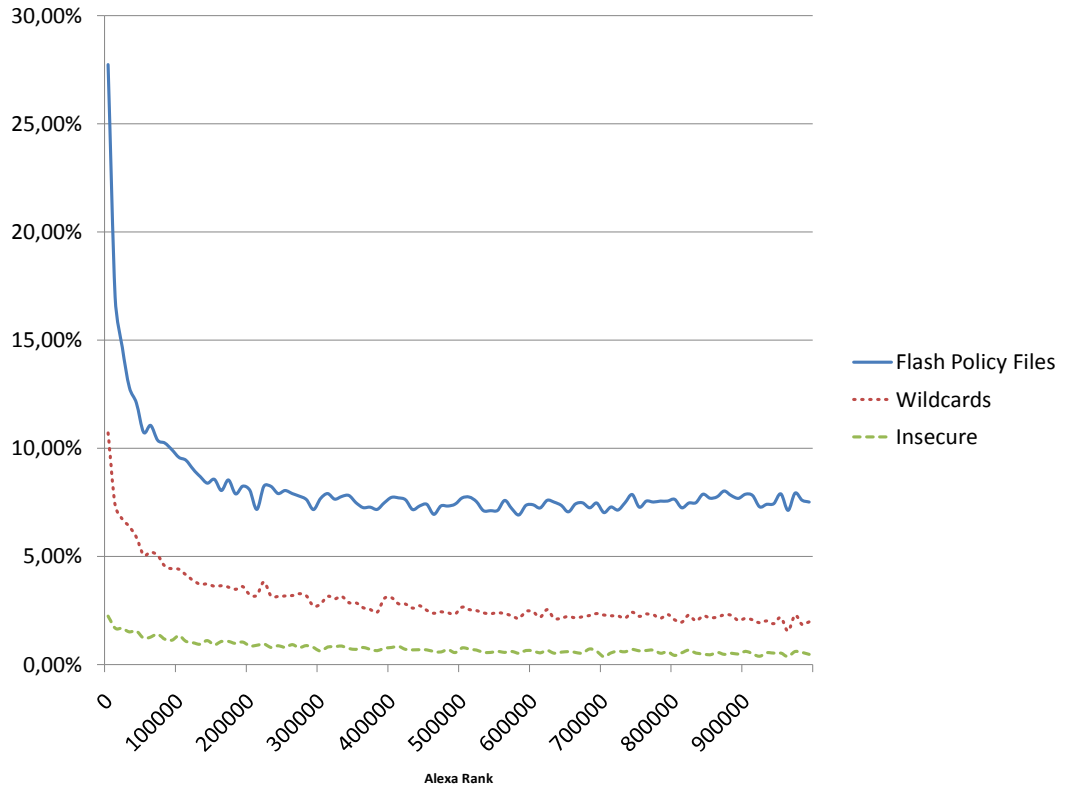


crossdomain.xml (in)Security

- Web sites with a general wildcard in their policy allow **all** domains crossdomain access
- This equals a waiving of the same-origin policy
- …how common is this?

Survey

- We examined the crossdomain.xml files of the Alexa top 1.000.000 sites
- Wildcard policy: 31,011 files (roughly every third policy file, 2,8% of all analyzed sites)



The HTML5 Way: CORS

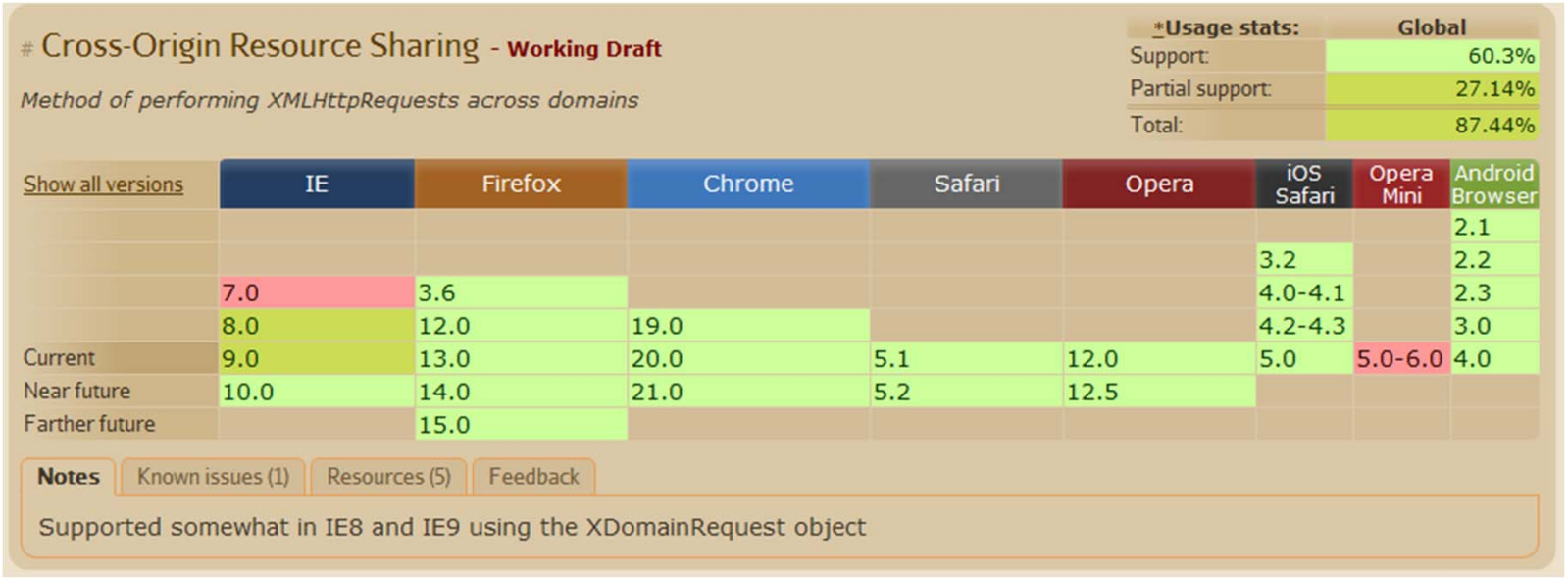
- Cross-Origin Resource Sharing
- Native extension of the browser's XMLHttpRequest object
- Allows sending of cross-domain HTTP Requests
- The HTTP Response is checked for an `Allow-From` header
  - Authorizes the request through carrying the names of the whitelisted domains
- Only if this header is present and the requester's domain is present in its value, the response is passed to the JavaScript

CORS Security

- CORS allows the sending of cross-domain requests
  - But only requests that could also be generated with HTML tags ("simple requests")
  - "Complex requests" require a preflight handshake with the server
- CORS allows wildcards ("*") in the Allow-From header
  - However, requests to resources with wildcards are not allowed to carry authentication information (e.g., cookies)
- CORS allows fine grained control
  - Whitelisting on a resource level
  - Dynamic setting of the header based on request origin and execution context

CORS verdict

- CORS is secure by default
  - No response header – request fails
- Using CORS insecurely is very very hard
- CORS is widely supported



Cross-Origin Resource Sharing - Working Draft

Method of performing XMLHttpRequests across domains

*Usage stats:	Global
Support:	60.3%
Partial support:	27.14%
Total:	87.44%

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser
								2.1
						3.2		2.2
	7.0	3.6				4.0-4.1		2.3
	8.0	12.0	19.0			4.2-4.3		3.0
Current	9.0	13.0	20.0	5.1	12.0	5.0	5.0-6.0	4.0
Near future	10.0	14.0	21.0	5.2	12.5			
Farther future		15.0						

Notes | Known issues (1) | Resources (5) | Feedback

Supported somewhat in IE8 and IE9 using the XDomainRequest object

HTML 5 Legacy

0 : 0

HTML 5 Legacy

1 : 0

In-Browser Communication

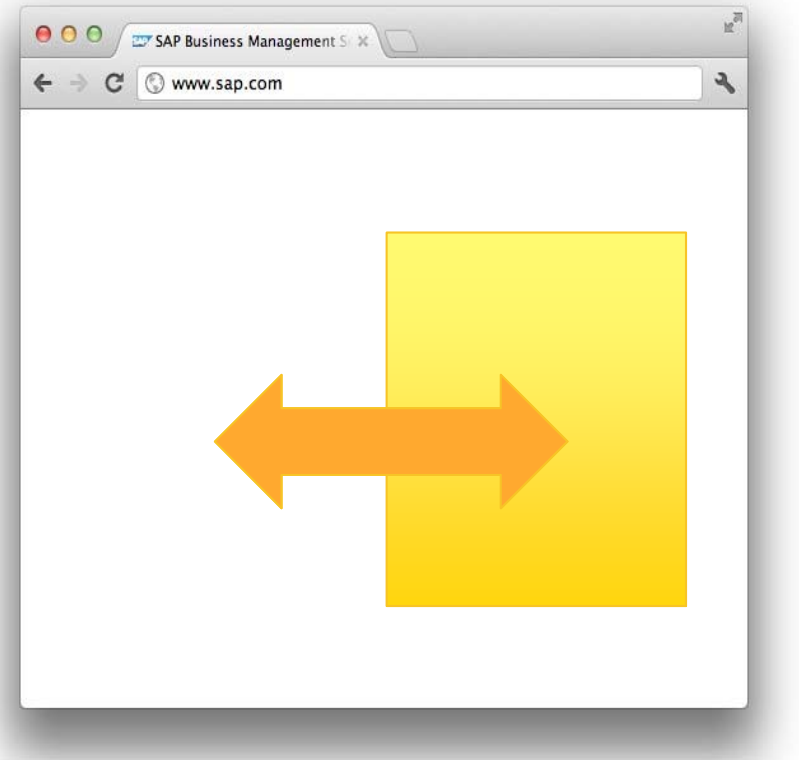
Security in knowledge

#RSAC

RSA CONFERENCE EUROPE 2013

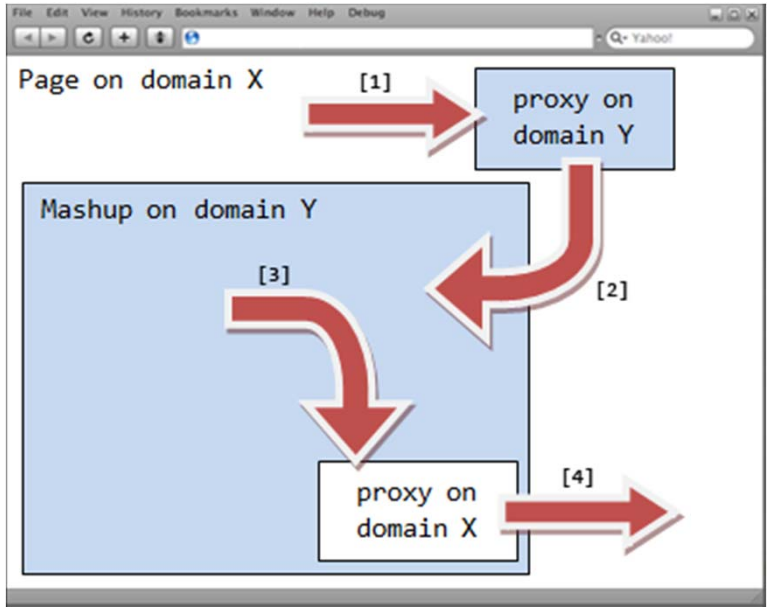
The Problem

- **Developer:** I would like to communicate with this crossdomain iframe
- **SOP:** No, no, no! You are not allowed to do so!
- **Developer:** Well, I will do it anyways...



Legacy 1: hash-identifier passing

- Hash (or fragment) identifier
  - The hash (#) in a URL points to a local anchor
  - Reload a document with a changed hash does not cause a actual reload
- Communication technique
  - The father frame sets the iframe source, passing the message in the hash
  - The iframe sets the parent's location, passing the reply in the hash



Legacy 2: window.name

- window.name
  - window.name is a (somewhat strange) DOM property
  - Its value can be set crossdomain
  - And after the value has been set, it survives navigation
- Hence, it can be used for in-browser communication
  - For instance, the Dojo framework supports it as one of their data transports

Hash and name (in)Security

- Authenticity
  - Both techniques have in common, that they have no assurance about sender authenticity
- Confidentiality
  - window.name maintains its value upon navigation
  - If the adversary is able to navigate a frame or window that carries sensitive information in window.name, data leaks can occur

Legacy 3: Domain relaxation

- Situation: Two documents hosted on separate sub-domains want to exchange data
- In this case, the browser allows relaxing the SOP via setting the `document.domain` property
  - The property can only be set to a valid suffix including the father domain
  - Example: purchase.example.org -> example.org
- If both documents relax their domain, they have **full** JavaScript access to their respective DOMs

Domain Relaxation: (in)Security

- Domain relaxation weakens the SOP's security guarantees
  - We wanted: Data exchange
  - We granted: Full access
- Furthermore, only coarse grained control
  - The document is now open to **all** subdomains, not only the desired communication partner
  - An XSS in one of the subdomains suffices to compromise the document

The HTML5 Way: PostMessage

- PostMessage is an API for cross-domain signaling in the browser
- Usage
  - Sender: *target*.postMessage(message, targetOrigin)
  - Receiver: sets up event handler for the "message" event

```
window.addEventListener("message", handlePostMessage);

function handlePostMessage(event){
    if(event.origin === 'http://example.net'){
        // do something
   }
}
```

PostMessage Security

- The PostMessage API has strong security guarantees
- Confidentiality
  - Message is only delivered to the target origin
- Authenticity
  - The message carries unspoofable information about the sender origin
- Integrity
  - The message cannot be intercepted or altered by third parties

HTML 5 Legacy

1 : 0

HTML 5 Legacy

2 : 0

Local Persistent State

Security in knowledge

#RSAC

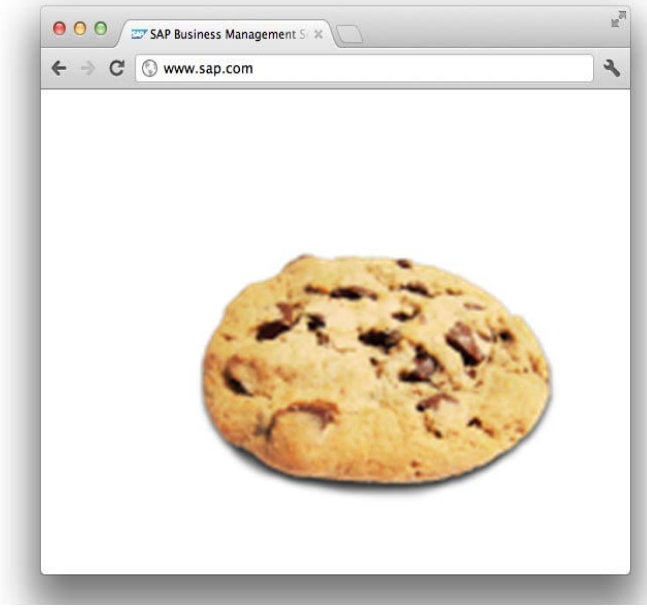
RSA CONFERENCE EUROPE 2013

The Problem

- **Developer:** I would like to permanently store data on the user's browser
- **SOP:** No, no, no! You are not allowed to do so!
  - Local state is in general accessible only under URL-schemes that differ from http(s)
- **Developer:** Well, I will do it anyways...

Legacy Technique: Cookie hacks

- Cookies can be set using the `document.cookie` property
  - This way the data stays in the browser even when the window/tab is closed
  - On a later visit, the data can be retrieved in the same fashion
  - Hence, local persistent state…



Cookies: Network overhead

- The purpose of the cookie is to maintain state that *is communicated to the server*
- Hence, all matching cookies are sent to the server with every request
  - This is hardly saving bandwidth…

Cookies (in)Security

- Cookies adhere to a significantly more lax SOP:
  - Protocol (http/https) and port are ignored
  - Cookies of father domains are send with requests to subdomains
- Attacks (Web attacker)
  - XSS on a subdomain: Read the state of all father domains
  - XSS on a service hosted on the same server (e.g., under port 8080): Read state of co-located applications
- Attacks (Network attacker)
  - Create http request: Read local state of application
    - Even the state of applications using https

The HTML5 Way: LocalStorage

- JavaScript API to store data in the browser
- Access only for same-origin scripts
- **Strict** enforcement of the same-origin policy

```
<script>
  //Set Item
  localStorage.setItem("foo","bar");
  ...
  //Get Item
  var testVar = localStorage.getItem("foo");
  ...
  //Remove Item
  localStorage.removeItem("foo");
</script>
```

HTML 5 Legacy

2 : 0

HTML 5 Legacy

3 : 0

ClickJacking Protection

Security in knowledge

#RSAC

RSA CONFERENCE EUROPE 2013

ClickJacking Legacy: Framebusters

- ClickJacking (aka UI Redressing)
  - Framing crossdomain content
  - Hiding the frame with CSS
  - Tricking the victim to click security sensitive UI
  - …
  - Profit
- Legacy protection: JavaScript framebusters

```
<script>
  if (parent!= self)
    parent.location = self.location;
</script>
```

Framebuster (in)Security

- Several ways exist to circumvent this protection:
  - Prevent JavaScript execution
    - Misusing modern XSS filters
    - Using sandboxed iframes
  - Prevent redirect
    - 204 flushing
    - Double framing
    - By asking the user nicely (onbeforeunload event)
- It is possible to build secure frame busters. However, the knowledge about it is not widely spread

The HTML5 way: X-Frame-Options

- Approach introduced by Microsoft to counter Clickjacking attacks
- Idea is similar to frame busting: Avoid unauthorized framing of a page
- Implementation:
  - Non-JavaScript solution
  - Based on an HTTP Response header
  - Browser enforces the Web server's desired behavior

HTML 5 Legacy

3 : 0

HTML 5 Legacy

4 : 0

Bonus track: Fighting XSS

Security in knowledge

#RSAC

RSA CONFERENCE EUROPE 2013

Cross-site Scripting (XSS)

- We (the security community) know about the general XSS problems since more than 10 years
  - The first advisory was in the year 2000
- Since the growing dominance of Web Applications we also understood the severity of the problem
  - Still, it appears as if we cannot handle the problem
- In 2011 more than 50% of all examined Web sites had at least one XSS problem (data collected by White Hat Security)
- This year we ran a study on DOM-based XSS
  - We fully automatically found DOM-based XSS problems in 10% of the Alexa 5000

XSS countermeasures

- Modern browser bring several means to contain XSS
  - Sandboxed iFrames
  - Content Security Policy (CSP)
  - Client-side XSS filter

Sandboxed iFrames

- In a sandboxed Iframe, JS execution is prevented
  - Render untrusted data in sandboxed Iframes to stop XSS-based JS
- Even better: Using the srcdoc attribute
  - srcdoc contains the to be rendered markup directly
- Problem:
  - Layout loses rendering flexibility

Content Security Policy (CSP)

- Server specifies legitimate script sources
  - Whitelisting of hosts
- Forbids
  - JavaScript within the HTML
  - String to code conversion (e.g., eval())
- With these rules the vast majority of cross-site scripting is mitigated

Client-side XSS filter

- Most modern browser provide client-side XSS filter
  - Internet Explorer, Google Chrome, Apple Safari
  - For Firefox the add-on NoScript is required
- Combat “reflected XSS”
- String comparison between URL and script content
  - Catches the most simple XSS attacks

HTML 5 Legacy

4 : 0

HTML 5 Legacy

6½ : 0

Conclusion

- Modern browser APIs realize needed client-side techniques
- These APIs have be designed with solid security considerations
  - Strict adherence to the same-origin policy
- They are not only superior on a functional level but in general they are actually **more secure** than their legacy counterparts

Security in knowledge

Thank you!

RSA CONFERENCE EUROPE 2013

#RSAC