Markup Injections

Volksmund: XSS

Mario Heiderich

Überblick

- Reine Markup Injections
- Clickjacking und UI Redressing
- Cross Site Scripting
  - JavaScript Injections
  - CSS Injections
  - Unsichtbarer Payload
- XML Injections
- DOM Injections

Warum der Titel?

- XSS als Begriff zu spezifisch
  - Iframes und SOP
- Auch inaktives Markup kann böse sein
- Browser-spezifische Verhaltensmuster
- Stylesheets
- Unsichtbare Payloads
- Angriffe auf das DOM
- Die E4X Überraschung

Zuerst die Basics

- Gibt es Markup das inaktiv aber dennoch schädlich ist?
- Beispiele bitte :)

Zum einen...

- ... kennen wir ja bereits Event-Handler
- `<a onclick=”stealCookies()”>click me</a>`
- `<a href=”javacript:alert(1)”>click me</a>`
- `<b onmouseout=”stealPassword()”>hover me</b>`
- `<iframe onload=”alert(1)”></iframe>`
- `<form onsubmit=”submitToSoElse()”></form>`
- `...`

User-Interaction?

- Manche Event-Handler benötigen *keinerlei* User Interaction zum Feuern
- Z.B. `body`, `iframe`, `script`, `link`, `img`, `object`, `embed`, `applet` und einige weitere
- Mit anderen kann User Interaction... erzwungen werden
- Absolut positionierte `div`s mit 100% width und height...
- Man muss einfach hovern...

Fehlerbehandlung

- `img` Tags, `script` Tags und alle anderen die Ressourcen nachladen können
- Wirre Interpretation seitens der Browser
  - `<img src onerror=alert(1)>`
  - `<img src= onerror=alert(2)>`
  - `<img src=”” onerror=alert(3)>`
  - `<img src=x onerror=alert(4)//`
  - Und natürlich `<image src=x onerror=alert(1)>`

Exotische Varianten

- In Firefox akzeptieren sogar `noembed` und `noframes` Event Handler
- Oder einfach Phantasie-Tags
  - `<omg!? onclick=alert(1)>click</omg!?>`
  - `<marquee onstart=alert(1)//`
  - `<input type=image src=x onerror=alert(1)//`

Richtig böse Tags

- Was sind denn nun die richtigen Fieslinge?
- Die die am *unschuldigsten* erscheinen
- Wie `label` oder `ul`, `ol` und andere
- Bad-Asses wie `base`
- `meta`, `link` und `xml`
- Legacy-Tags wie `isindex`, `layer` und `ilayer`

Und was ist mit ...

- ...Attributen?
- Gibt es gefährliche Attribute neben den Event Handern?
- Yep – `style` Attribute
- Und `id` Attribute
- "*Objects in the global scope are closer than they appear*"

Style Attribute - okay...

- Erlauben absolute Positionierung
- `expression()` im IE6-7 und IE8 im Compat Mode
- `-moz-binding, background-image`
- Elemente beliebig strecken und platzieren
- Unsichtbare Elemente sichtbar machen
- Elemente zwischen verketteten Events ein- und ausblenden
- CSS History Hacks

IDs und Labels

- Ein Click auf ein Label mit passendem Form-Element...
- ... triggert einen echten Click Event
- Praktisch und sinnvoll für alle Form Elemente
- Abgesehen von Submit Buttons
- Exploitbar?

IDs und das DOM

- Ohne Doctype haben ID Attribute direkten Einfluss auf das DOM
- Pro Element mit ID wird eine ungewöhnliche DOM Property neu erstellt
- Objekte im globalen Scope
- Interessant wird das Ganze im IE6 – IE8 RC1

DOM Redressing

- Einige Beispiele:
  - `<form id=”document”>`
    `<input cookie=”foo” />`
    `</form>`
  - `<select id=”document”>`
    `<option id=”body”>foo</option>`
    `</select>`

- Native Properties können per Markup überschrieben werden

Background

- Auch das alte Background-Attribut ist gefährlich
- Opera akzeptiert javascript: URIs
- Und führt den Code aus
- `<table background=javascript:alert(1)>`

CSS Injections

- Clickjacking
- “Redressing the UI”, Repositionierung bestehender und Überlappung mit neuen Elementen
- Z-Index
- Stylesheets und deren “Events”
- Stylesheets können Attributwerte auslesen
- So auch Passwortfelder
  - `input[value^=a]` und `input[$=z]` als ersten Schritt

JavaScript via CSS ausführen

- Jeder Browser hat seine eigene Methode – fast jeder
- Gecko nutzt `-moz-binding`
- Safari und Chrome kennen `binding()` – wird aber noch nicht unterstützt
- Opera bringt frischen CSS XSS Support via SVG Fonts
- Internet Explorer nutzt `expression()`

-moz-binding

- Mittlerweile fast nicht mehr für Angriffe zu missbrauchen
- Prima in FF2 (cross domain)
- Fast ebenso prima in FF3 (same domain und dataURIs)
- Fast nicht mehr zu nutzen in FF3+ (same domain)
- Aber immer noch zu verwenden in Verbindung mit Link Tags

...heisst?

- LINK Tags im Header
- @import Direktiven
- Auch gerne mit UTF-7
- Oder anderen skurrilen Charsets – UTF-32BE etc.

Moment!

- Was waren noch gleich Data URIs?
  - Im wesentlichen URIs die nicht verweisen sondern beinhalten
  - Text oder binäres
- Aufbau einer Data URI
  - `data:` Protokoll Präfix
  - Mime Type – wie `text/html` oder `application/pdf`
  - Charset – wie `UTF-8`
  - `data:text/html;charset=utf-8,<h1>Hi!</h1>`

SVGXSS in Opera 10

- Vektorschriften im SVG Format via CSS nutzen
- So funktionierts:

```
<?xml version="1.0" standalone="no"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
@font-face {
    font-family: y;
    src: url("test2.svg#x") format("svg");
}
body {
    font: 100px "y";
}
</style>
</head>
<body></body>
</html>
```

SVGXSS Teil 2

- Die bösartige SVG “Schrift”

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dt
d" >
<svg onload="alert(1)" xmlns="http://www.w3.org/
2000/svg">
<defs>
<font id="x" horiz-adv-x="450" >
  <font-face
    font-family="y"
  />
</font>
</defs></svg>
```

expression()

- Kein Sicherheitslücke – eher ein Feature
- JavaScript kann uneingeschränkt im CSS eingebunden sein
- Hat vollen Zugriff auf das DOM der geladenen Seite
- Kann mit Kommentaren und Nullbytes verschleiert werden
  - `* { xss:expression(alert1());}`

Zurück zu den Wurzeln

- Unsichtbarer Payload und DOMXSS
- You tell me!

Payload verstecken

- Keine Sichtbarkeit für Server und IDS
- Möglichkeit eins:
  - `location.hash`
  - `substr(1)`
  - Evaluieren des Resultats
- Möglichkeit zwei:
  - `window.name`
  - Viele viele Megabytes
  - Und speichert unabhängig von Protokoll und Domain

IDS und WAFs

- Fast keine Möglichkeiten zur Erkennung
- Sehr kurze Trigger-Vektoren
- Die wiederum leicht zu verschleiern
- WAFs im Client?
- Mehr Fokus auf die Trigger
- Kürzester Vektor? `eval(name)`
- Kürzester Vektor ohne Klammern?

DOMXSS

- Unsicheres Markup
- DOM Properties die per remote beeinflussbar sind
- `location` Objekt
- `document` Objekt
- Und viele weitere Properties

AJAX

- Mit XHR kann eine Menge erreicht werden
  - Information Disclosure
  - Passwörter auslesen
  - Anti-CSRF Tokens auslesen
  - HTTP Requests fast beliebiger Methode
- Komplette Prozesse im Hintergrund
- DOM Worker helfen in zukünftigen Browsern

AJAX Requests

- API Calls missbrauchen
  - `my.getUserInfo(12345,'phone');`
- Angriffe auf den Server via `setRequestHeader()`
- Auslesen von Daten aus dem Back-End
- Server DoS mit Loops
- Client DoS gegen Browser oder Extensions
- Sehr interessant im Falle NoScript
- Dazu mehr zu späterem Zeitpunkt

AJAX Responses

- JSON Hijacking
  - Auch dazu später mehr
- Vor-Filter und Logging
- Überschreiben des XHR Objekts

Pause