Client-side Security in the modern web

Mauro Gentile
Security Consultant @
Minded Security
mauro.gentile@mindedsecurity.com

OWASP EU Tour 2013 - Rome
Rome, 27th June 2013

The OWASP Foundation
http://www.owasp.org

About me

- Mauro Gentile
  - MSc. in Computer Engineering
  - Application Security Consultant @ Minded Security
  - Creator of *snuck* – open source automatic XSS filter evasion tool
  - Security research and vulnerabilities @ http://www.sneaked.net

  - Twitter: @sneak_
  - Keywords: web application security, web browser security

Current state of the Web

- It's a matter of fact that the Web is evolving
  - Modern web applications actually resemble desktop apps
    - Small granularity of exchanged data through Ajax implies negligible response times and user satisfaction
  - We are moving towards more code client-side
    - Web browsers are the doors to the Web
    - Great user-experience
    - Perform tasks faster than ever
  - Web browsers offer sensational capabilities
    - HTML5
    - CORS
    - JavaScript

Clearly, expanding capabilities in any type of application generates progress, but possibly enlarges the possibilities to exploit new features with malicious intent.

Objectives and Motivation

- Create awareness about possible security issues
- Show how attack vectors are changing
- Discuss real world attack examples
- Describe interesting countermeasures:
  - Advantages
  - Drawbacks

Reflected and Stored XSS

- Smart and compact definition:
  - «External JavaScript running in our domain»
- Developer's perspective:
  - Dev. A: «My blacklist-based XSS filter is so robust that there is no chance for you to bypass it!»
  - Dev. B: «Come on, we are already filtering out <script>»
- Attacker's perspective:
  - «Well, HTML5 introduced new tags and attributes, why don't try with them?!»

HTML5-based XSS vectors

```
<input onfocus=write(1) autofocus>

<form id="test" /><button form="test"
formaction="javascript:alert(1)">X</button>

<video><source onerror="alert(1)">

<form><button formaction="javascript:alert(1)">X</button>
```

- Just the tip of the iceberg
  - Awesome research: *HTML5 Security Cheatsheet* – Heiderich [1]
- Robust protection:
  - White-list protection mechanisms

More client-side code => DOM XSS risk

- DOM XSS are becoming pervasive in modern web apps
  - JavaScript code analysis is complex
  - Vulnerabilities may come out through user interaction
  - No systematic approach for detecting these
    - Real-time data tainting: DOMinator

    https://dominator.mindedsecurity.com/

- Common example (*fat regexp*):

```
#<a
href=vbscript:MsgBox(document
.domain)'bla@bla.xxIE>funny
picture</a>
```

```
var h = location.hash.substring(1);
    if (h && h != "") {
            var re = new RegExp(".+@.+");
            if (h.match(re)) {
                    document.getElementById("email").innerHTML += h;
            }
    }
```


Protecting against DOM XSS

- Client-Side Encoding
  - jQuery-encoder or ESAPI4JS
    - Contextual Output Encoding on the client-side
    - Encode data from untrusted resources
    - Be careful: use the encoder for the right context
    - Be aware of potential attacker controlled *sources*
- «You're doing it wrong» example:

```
<div id="element"></div>
<script>
// data is controlled by the user
$('#element').html( '<a href="' + $.encoder.encodeForHTML(data) +
'">click me</a>' );
</script>
```

```
data = 'javascript:alert(1)';
```


Client-side XSS filters

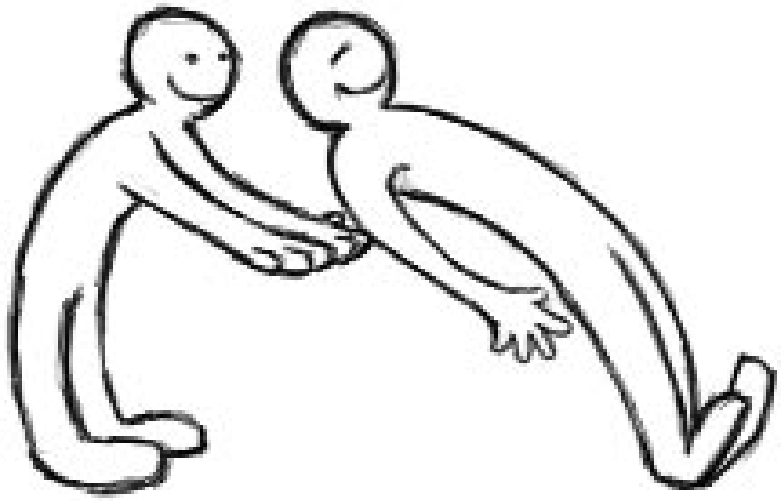
- Browsers built-in XSS protection
  - IE's XSS filter
  - XSSAuditor (Google Chrome)
  - NoScript (FF extension)
- Different approaches
  - Black-list for GET parameters
  - Input reflection inspection
- Trade-off: High coverage - False positives (usability)

Client-side XSS filters (cont'd)

- Further protection layer against reflected XSS
  - Since fixing all the vulnerabilities could be impossibile, why don't delegate the problem to the browser itself?
- Still feasible to find bypasses, especially in XSSAuditor
  - `?inp=<script/src=data:&inp=alert(1) />`
    - HTTP Parameter Pollution
    - ASP.NET
- Fairly good adoption, but surely not a panacea

Content Security Policy

- Innovative extension for protecting against XSS
  - Basic idea: explicitely define what your application needs to render and the *trusted origins* from which some content is served up
  - HTTP headers to enforce in the client a least-privilege environment



Content Security Policy (cont'd)

- Code / Data separation
  - Inline scripts are prohibited by default
  - Allow only scripts from whitelisted domains
  - Good granularity for selecting trusted domains for every type of content

- Very few sites are adopting this policy
  - Barriers to introduction
    - It may be really complex to rebuild the application in order to follow the code / data separation principle
  - It might be difficult for developers to understand the real benefits
    - We need to create awareness of the potential of such an introduction

Does CSP solve XSS?

- Misconfigured policies lead to XSS again

- Are the trusted domains really «trustable»?
  - If the attacker is able to inject the domain we are taking the contents from, then the CSP benefits are immediately frustrated

- Script-less Injections
  - Potentially the future of injection attacks
    - Stunning research:
      - *Postcards from the post-XSS world* – Zalewski [2]
      - *Scriptless Attacks - Stealing the Pie Without Touching the Sill* – Heiderich et al. [3]

Cross-Site Request Forgery

- Security issue under evolution
  - Not just missing random tokens involved
  - Evolution towards state-less CSRF countermeasures
    - The client-side itself may maintain a consistent state which is not known to the server
      - Issuing cross-domain requests to REST services is a concrete attack
  - CORS, Cross Origin Resource Sharing
    - Helping the attackers to forge invisible x-domain requests
  - HTTP header: *Origin*

Cross-Site Request Forgery (cont'd)

- CORS
  - Same-Origin policy relaxation
  - Domains can define trusted domain which can access their data
    - Similar to the idea behind crossdomain.xml
    - Resouce-by-resource granularity
  - The attacker may trigger x-domain requests through AJAX, although he cannot read the responses
    - CSRF exploits
    - *Invisible Arbitrary CSRF File Upload* – Kotowicz [4]

```
var xhr = new XMLHttpRequest();
xhr.open("POST",[URL], true);
xhr.withCredentials = true;
http.send();
```


«Novel» protections against CSRF

- HTTP Header Origin
  - The browser automatically inserts the issuing request domain
  - Checking whether the incoming Origin is what we expect is a good idea for protecting against CSRF
    - But, as usual, hackers could find a way to forge it x-domains through plugins…
- Double submit
  - Clever protection for state-less services
    - However, we could break it through MITM
      - Override the anti-CSRF cookie and put it in the request's body too

Cross-Domain communication

- CORS
  - Good and secure implementation among browsers
  - No bypass registered till now
- JSONP
  - Trick to «bypass» SOP
    - Potentially vulnerable to many issues


```
<script>
function func(data) {
    alert(data);
}
</script>
<script
src="//gimme.com/x.php?callback
=func"></script>
```

gimme.com/x.php

```
<?php
echo $_GET["callback"]."(0)";
?>
```


JSONP

- Potential issues
  - Cross-Site Scripting through Content-Sniffing
    - The attacker can inject the first bytes of the web page
    - "X-Content-Type-Options: nosniff" not set
  - DOM-Based XSS through HTTP Parameter Pollution
    - Very common in autocomplete AJAX input fields
  - Leveraging the only authentication for supplying private content is possibly an issue
    - Adopting ?callback=func&token=[personal_token] would prevent data stealing across domains
    - The attacker cannot know personal tokens a-priori

Sandboxed iframes

- HTML5 introduced a new feature to safely frame 3rd-party content
  - <iframe sandbox src=//evil.com></iframe>
  - Permission of the framed content can be restricted through flags:
    - allow-scripts
    - allow-forms
      - No CSRF starting from the «guest»
    - allow-same-origin
    - allow-top-navigation
      - It may break frame-busters, therefore adopt X-Frame-Options

UI Redressing

- «C'mon boy, we know everything about clickjacking…»
  - Really?
  - Cross-Domain Content Extraction mostly solved
  - Cross-Domain Injection still possible
  - Client-side XSS filters and <*iframe sandbox*> may vanish straightforward frame-busters
  - X-Frame-Options, correctly adopted by many popular sites
    - Robust protection, however the top, instead of the parent, is checked
    - Allowing widgets in our domain might make us vulnerable
  - Frame Hijacking: redefining the location of frames[x].frames[y]
    - Controlled iframes in a trusted domains look credible

UI Redressing (cont'd)

- Innovative exploitation scenarios
  - Clicks anticipation combined with history navigation
  - Drag and Drop operations with history nav.
    - Although quite complex, they frustrate XFO
  - Same-Origin content exfiltraction

- Future protections?
  - Prediction: HTTP headers for defining:
    - which domain is considered trusted to drag content into
    - what resource can be extracted and to which domain
  - Opera introduced the idea of exposing the event origin

What can we say?

- More and more attention on the client-side in the security of the modern Web:
  - *CSP*: The server teaches the browser which content can be rendered
  - *Origin*: The trust the server has for a certain domain is delegated to the browser
  - *UI Redressing*: the server indicates the browser how should behave wrt iframes
  - *CORS*: allowing the developers to avoid using a proxy server, and make them use the browser for x-domain requests

Summary

- The recently introduced technologies offer interesting capabilities; we reported some good and bad consequences
- HTML5 is designed with very good security principles in mind: new tools to solve the most problematic issues in the Web
- CSP is a good introduction to explicitely define what your application is required to render

Ending…

- What we've seen is just the upper layer of client-side security
  - Many other points should be taken into account

Amazing readings:

- *Browser Security Handbook*

  https://code.google.com/p/browsersec/wiki/Main
- *The Tangled Web: A Guide to Securing Modern Web Applications*

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

  Michal Zalewski
- *Web Application Obfuscation:* '-

  /WAFs..Evasion..Filters//alert(/Obfuscation/)-'

  Mario Heiderich, Eduardo Alberto Vela Nava, Gareth Heyes, David Lindsay

References (1/4)

[1] *HTML5 Security Cheatheet*
http://html5sec.org
manteined by Mario Heiderich

[2] *Postcards from the post-XSS world*
http://lcamtuf.coredump.cx/postxss/
Michal Zalewski

[3] *Scriptless Attacks - Stealing the Pie Without Touching the Sill*
http://www.nds.rub.de/media/emma/veroeffentlichungen/2012/08/16/scriptlessAttacks-ccs2012.pdf
Mario Heiderich, Marcus Niemietz, Felix Schuster, Thorsten Holz, Jörg Schwenk

[4] *Cross domain arbitrary file upload Redux*
http://blog.kotowicz.net/2011/05/cross-domain-arbitrary-file-upload.html
Krzysztof Kotowicz

Invisible arbitrary CSRF profile picture upload in Facebook
http://www.sneaked.net/invisible-arbitrary-csrf-profile-picture-upload-in-facebook
Mauro Gentile

References (2/4)

jquery-encoder
https://github.com/chrisisbeef/jquery-encoder
Chris Schmidt

Domxsswiki
https://code.google.com/p/domxsswiki/
Stefano Di Paola

Regular Expressions Considered Harmful in Client-Side XSS Filters
http://www.collinjackson.com/research/xssauditor.pdf
Daniel Bates, Adam Barth, Collin Jackson

Robust Defenses for Cross-Site Request Forgery
http://seclab.stanford.edu/websec/csrf/csrf.pdf
Adam Barth, Collin Jackson, John C. Mitchell

X-Frame-Options, or solving the wrong problem
http://lcamtuf.blogspot.it/2011/12/x-frame-options-or-solving-wrong.html
Michal Zalewski

References (3/4)

Web Application Security in front end
http://www.slideshare.net/eoftedal/web-application-security-in-front-end
Erlend Oftedal

Application Security for RIAs
http://www.slideshare.net/johnwilander/application-security-for-rias
John Wilander

HTML5 Top 10 Threats Stealth Attacks and Silent Exploits
http://media.blackhat.com/bh-eu-12/shah/bh-eu-12-Shah_HTML5_Top_10-WP.pdf
Shreeraj Shah

Html5 Security Realities
http://www.slideshare.net/BradHill2/w3-conf-hillhtml5securityrealities
Brad Hill

Something wicked this way comes
http://blog.kotowicz.net/2011/11/html5-something-wicked-this-way-comes.html
Krzysztof Kotowicz

References (4/4)

HTML5 Drag and Drop
http://dev.opera.com/articles/view/drag-and-drop/
Dev.Opera

Cookie Tossing in the Middle
http://websters prodigy.net/2013/05/24/cookie-tossing-in-the-middle/
Rich Lundeen

DOM Xss Identification and Exploitation
https://dominator.googlecode.com/files/DOMXss_Identification_and_exploitation.pdf
Stefano Di Paola

Cross Origin Madness or Your Frames Are Belong to Us
http://homakov.blogspot.ru/2013/02/cross-origin-madness-or-your-frames-are.html
Egor Homakov

Origin Policy Enforcement in Modern Browsers
https://www.frederik-braun.com/thesis/presentation_hackinparis2013.pdf
Frederik Braun

Questions?

Thanks!

Contacts

Twitter: @sneak_

Personal: gentile.mauro.mg@gmail.com

Blog: http://www.sneaked.net

Work: mauro.gentile@mindedsecurity.com

Site: http://www.mindedsecurity.com

Blog: http://blog.mindedsecurity.com

Twitter: http://www.twitter.com/mindedsecurity