The Future of Web Application Security

W3Conf, November 15 & 16, 2011

Brad Hill
@hillbrad
bhill@paypal-inc.com

Scott Stender
@scottstender
scott@isecpartners.com
The History of Web App Security
Attacker as Eavesdropper and Impersonator
Attacker as Client
OWASP Top 10 2010

- A1: Injection
- A2: Cross-Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards
Attacker as Server

Bank.com

Evil.com
Confusing the Client: CSRF

Cookie: mycreds=sekr1t
GET /transfer?amt=$1M&to=Mallory

<iframe src=https://bank.com/transfer?amt=$1M&to=Mallory>

GET /dancingpigs.html
Client Code Injection: Reflected XSS

GET /dancingpigs.html

<iframe src="https://bank.com/?amt=<script>evil.js</script>"

<script>evil.js</script>
Client Code Injection: Stored XSS

GET /comment

POST /comment?txt=<script>evil.js</script>
OWASP Top 10 2010

• A1: Injection
• A2: Cross-Site Scripting (XSS)
• A3: Broken Authentication and Session Management
• A4: Insecure Direct Object References
• A5: Cross-Site Request Forgery (CSRF)
• A6: Security Misconfiguration
• A7: Insecure Cryptographic Storage
• A8: Failure to Restrict URL Access
• A9: Insufficient Transport Layer Protection
• A10: Unvalidated Redirects and Forwards
Conventional Wisdom of Web Security:

Never trust the client.

Defend the server, at the server.
HTTPS

WEB SECURITY 1.0
Enter Web 2.0
AJAX Empowers Clients
Mobile Code, not just Mobile UI
Client Code Injection: DOM XSS

```
GET /dancingpigs.html
```

```
<iframe src=https://bank.com/app.js#alert(xss)>
```

```
GET /app.js
```

```
eval(location.href.split("#")[1]);
```

```
GET /dancingpigs.html
```

```
#alert(xss)
```

```
DOM
```

```
#alert(xss)
```

```
GET /app.js
```

```
<iframe src=https://bank.com/app.js#alert(xss)>
```
Mindedsecurity.com tested a sample of the Alexa Top 1M, and found 56% of sites they examined had exploitable DOM XSS vulnerabilities.

Source of real vulnerabilities against Twitter, Omniture, more...

DOM XSS Sources:
- `document.URL`
- `document.URLUnencoded`
- `document.documentURI`
- `document.location`
- `document.referrer`
- `window.name`
- `window.location`
- `window.dialogArguments`
- `postMessage (HTML5)`
Web APIs beyond the Browser
What is the sound of Cross-Site Scripting without a Site?

Credit: kyleosborn.org
Data from Everywhere: Meet the Mashups
One App, Many Servers

- Accessing map data is great – if you are Google

- What about every other use of maps?

- Can’t we re-use that service?
NO!

Bank.com

Evil.com
First Generation Mashups: Hacking the Same-Origin Policy

• Clever uses of `<script> <img> <iframe>` aggregate content and work around the SOP

• JSONP is a fantastic example of a mature technology built on clever use of `<script>`

Or…
Making Mashups Work

• JSONP is a great way of inviting arbitrary code execution in your DOM

• JSONP places all of your security in the hands of the script source

• Sourcing script gives that server checkin privileges on your code
But they are trustworthy...

- Even trustworthy companies are compromised
- Even trustworthy companies enjoy your customers and your data
- Are your users, your customers, expecting that security model?
HTML5+

Mashing Up with permission
CORS and XHR2

• Cross-Origin Resource Sharing: Server-supplied headers permitting the client to relax the Same-Origin Policy
  – With or without credentials

• Richer response types with XMLHttpRequest Level 2
  – ArrayBuffer, Blob, File, FormData
  – Also allows anonymous requests
WebSockets

• Bi-directional communication between server and Web client

• May be cross-origin, client sends verified origin
CORS, XHR2, WebSockets

- No policy file at well-known location as, e.g.:
  - crossdomain.xml for Flash
  - clientaccesspolicy.xml for Silverlight
- Access control is dynamically determined, per-resource

Harder to identify and audit cross-domain permissions on a server than with plugin-based approaches.
postMessage

- Cross-Document and Channel Messaging establish communication between instances of client-side code in the same browser, including across origins
- Includes “origin” targets, but ports can be passed to other origins or an origin may itself be a mashup
Web RTC

• `postMessage` communicates between client-side apps in the *same* browser...

• Web RTC (in development) communicates real-time multimedia between client-side apps running in *different* browsers
HTTP, XHR

CORS, XHR2, WebSockets

postMessage

WebRTC

WebStorage
#alert(xss)
Assets and attack surface are moving to the client.

We are still only defending at the server, but it does not see all data flows.
Servers are still broken…

But the future of Web App Security is in the client.
So what now?
Testing/Verification

• Comprehensive analysis of standard web apps already stressed our abilities to test

• Now we have to manage the entire set of potential state in a DOM

• Test tools, even developer tools, are not where they should be
The State of Mobile Code Engineering:

I DON'T ALWAYS TEST MY CODE

BUT WHEN I DO I DO IT IN PRODUCTION

Stay vulnerable, my friends.
What do we do? What can we do?

• Today: Features you want come with built-in “Game Over” security problems
  – JSONP and `<script src=>` patterns
  – plaintext HTTP

• We need Mobile Code designed to be securable:
  – Compartmentalized
  – Decoupled
  – Explicit
  – Testable: Think TDD
Today:

• Use HTML5+ Mashup APIs
• Authenticate Origins: HTTPS and HSTS everywhere
• Secure Design: Compartmentalize and Sandbox Origins
• Client-Side Testing
Design: Compartmentalization

• Use httpOnly cookies
  – Isolate session credentials’ value from the DOM

• Minimize foreign origins
  – Remember: `<script src='X'>` is equivalent to giving ‘X’ unsupervised check-in privileges to your codebase

• Create unique origins to isolate apps and sandbox dangerous / active content
  – example.com, example-sandbox.com, example-marketing.com
  – Remember that scripts can reset their origin to something less-specific!
Seamless IFRAME Sandbox

Blog blog blog blog blog blog blog blog...

USER COMMENTS:
<iframe seamless sandbox>

origin=null

Mallory: <script>Whoops, a bypass!</script>
IFRAME Sandbox + JSONP

```html
<iframe sandbox="allow-scripts">
  script src=evil.com/jsonp?callback=pwned
</iframe>
```
Implementation: Validate!

• Context-sensitive validation of untrusted data
  – HTML, Attribute, CSS, JS, JSON, URL

• Do this client-side, even if you also do it server-side
  – Better view of the data through the local parser

• [http://plugins.jquery.com/project/jqencoder](http://plugins.jquery.com/project/jqencoder)
Some great starts on tooling:

- **DOMinator**
  - Firefox runtime tool for analyzing and identification of DOM-based XSS
  - Stefano Di Paola and MindedSecurity
  - [http://blog.mindedsecurity.com/2011/05/dominator-project.html](http://blog.mindedsecurity.com/2011/05/dominator-project.html)

- **DOM Snitch**
  - Chrome runtime tool: passive analysis of dangerous client-side practices in the DOM
  - Radoslav Vasilev @ Google

- Other DOM XSS tools from IBM, Blueinfy

- **WebDriver** and **Selenium** for testing in-the-client
Near Future

• Make and Demand better tooling.
• Design for and use a Content Security Policy
• Tell your browser vendors and API providers that security matters to you!
Least-Privilege: Content Security Policy

• New work pioneered by Mozilla and Google, now on W3C recommendation track.

• Header or META tag to allow/force a page to drop privileges and capabilities
  – No inline script, no code from strings (eval), no data: URIs (mostly)
  – Code must come from libraries at origins specified in script-src whitelist
  – Origin whitelisting for other features:
    • Image, media, frame, font, object/plugin
Challenges Remaining
Clickjacking / UI Redressing

• A serious threat to in-context mashups.

• “Pay” can be reversed. You can’t claw back your social graph after being Friend-jacked.

• X-Frame-Options allows refusal to render when framed
  – Breaks the Web to secure it
  – Not helpful when you want to be framed
  – Work underway to fix this in W3C WebAppSec WG
The DOM is still a mess

• Hundreds of browser-specific quirks
  – Hinders tool market, understanding of attack surface, cross-platform testing

• Amazing and innovative efforts at client-side security models by Mario Heiderich, Gareth Heyes, etc.
  – Client-side, foolproof anti-XSS, sandboxing, even RBAC in the DOM
  – Constantly foiled by the mess of the DOM
Need for a Strict DOM?

The Standard
The Whole Standard
And Nothing But the Standard
Thank you!
We are:
bhill@paypal-inc.com, @hillbrad
scott@isecpartners.com, @scottstender

Others to follow:
@WisecWisec, @garethheyes, @0x6D6172696F, @johnwilander,
@jeremiahg, @kkotowicz, @randomdross, @lcamtuf

http://www.schemehostport.com/  (Adam Barth)
The Tangled Web (new book by lcamtuf)