Cross-domain leakiness

Divulging sensitive information & attacking SSL sessions

Chris Evans - Google
Billy Rios - Microsoft

Who are we?

- **Chris Evans**
  - Troublemaker, Engineer, Tech Lead, Google Security Team
  - http://scary.beasts.org/
  - http://scarybeastsecurity.blogspot.com/
- **Billy Rios**
  - Model, Security Engineer, Microsoft
  - http://xs-sniper.com/

Shoulders of giants

Credits

- Filipe Almeida, Google
- Michal Zalewski, Google
- Drew Hintz, Google
- Kanatoko

Cross-domain leakiness

Overview

- Introduction
- Background information
- Cross-domain bugs
- Cross-domain issues by design
- Attacking browser SSL sessions
- Demo

Cross-domain leakiness

Introduction

- Is your web app cross-domain safe?
- What about your users' browsers?

Background information

Man-in-the-middle (MITM)

- Someone who can intercept your network traffic
  - Particularly on wireless networks
- No MITM
  - Attacks you via enticing to evil URLs
- Passive MITM
  - Only reads your traffic going past
  - e.g.: government black box @ ISP
- Active MITM
  - Can also forge responses / requests
  - e.g.: free wireless

Background information

Single session browsing

- A model of web usage for paranoid users
- Based on distrust
  - Of same origin policy
  - Of browser's security
  - Of separation of data
  - Of web app

Background information

Cross-site script inclusion

- Sometimes called XSSI
- Sometimes bucketed under XSRF / CSRF
- Loads remote authenticated resource
  - <script src="http://remote.com/sensitive"/>

Background information

Cookie security model

- Original cookie model:
  - Send if domain, path match
- Same origin policy added for DOM access:
  - Grant access if protocol, domain, port match
- Cookie model leaked https cookies over http
  - "Secure" cookie attribute added

Cross-domain bugs

Firefox #1: image theft

- Can steal authenticated images
- Fixed in latest 2.0 / all 3.0
- Introduced with canvas getImageData()
  - Or toDataUrl()
- Previously no way to read image pixels
- Domain is checked but 302 redirect trick works
- WebKit nightly also had same bug
- 302 redirect trick also worked in past
- Simple demo

Cross-domain bugs

Firefox #2: hex string theft

- Can steal authenticated hex strings
- Fixed in latest 3.0 / 2.0.
- Again uses 302 trick.
- Reads Javascript error messages across domain.
- Simple demo

Cross-domain bugs

Safari #1: File theft

- Still not yet fixed in a production release
  - So withholding details
  - Bug #1 / #2 reported May / June 2008
- Failure to do origin check in obscure cross-domain area
- Simple demo

Cross-domain bugs

Safari #2: XML theft

- Also not yet fixed in production release
- XML often used as protocol for:
  - AJAX apps
  - Web services
  - Feeds (including authenticated ones)
- Also don't forget XHTML
- Simple demo

Cross-domain bugs

Safari #3: File theft

- Fixed with CVE-2008-3638
- Failure to enforce remote to local boundaries
- Involves Java Applets
- http://xs-sniper.com/blog/2008/11/19/stealing-files-with-safari/
- getAppletContext().showDocument(url);

Cross-domain bugs

Safari #4: File theft

- Fixed with CVE-2009-0137
- Failure to enforce remote to local boundaries
- Inadequate sanitization/filtering/encoding of Attacker controlled XML

Cross-domain bugs

Safari #4: File theft

- Attack XML
  - **<content:encoded><![CDATA[**
    <body src=”image.JPG” onload=”javascript:alert ('xss’);””<onload=””
    ]]>
    </content:encoded>
- Resulting HTML
  - ** **
    <body src=”image.JPG” onload=”javascript:alert (‘xss’);”>
    **<onload></onload></body> <!– end articlebody – > **

Cross-domain bugs

WebKit: pixel theft

- Nice combination of features came together in WebKit nightly
- Illustrates danger of unexpected interactions
- Collaborating features:
  - SVG support
  - SVG as <img> target
  - getImageData()
  - <image> within SVG
- Could be worse with <html:iframe>

Cross-domain bugs

You want more?

- New browser features always adding new cross-domain areas and interactions
  - Sometimes obscure
  - CSS3
- Need a comprehensive list of where browsers will handle data cross-domain
- Spreadsheet to document areas and testing:
  - https://spreadsheets.google.com/ccc?key=pEFQCm3fodP3jM-IyIwjwSw

What's in a Name?

Case Study: Flash DNS Rebinding

- Quick reminder of DNS Rebinding / Pinning Issues
- The Attacker controls DNS for Foo.com
- Make a request for Foo.com, foo.com points to 111.111.111.111
- The Attacker changes the DNS entry for Foo.com to 10.1.1.1
- Attacker uses previously loaded content to steal information from 10.1.1.1

What's in a Name?

Case Study: Flash DNS Rebinding

- Fixed with CVE-2008-1655
- Distinguishing between two forms of the same name
- Jumperz (Kanatoko) is the MAN!
  - Based off of the original, but with a small twist

What's in a Name?

Case Study: Flash DNS Rebinding

- Dealing with domain names ending with a "."
- Now, are XS-Sniper.com and XS-Sniper.com. the same?
- Sockets in Flash made things interesting ☺
- This is fixed… why talk about this now?
  - SSL?
  - Same Origin Policy?

Design issue

CSS property theft

- Browsers load CSS cross-domain
  - <link rel="stylesheet" href="http://remote/blah" />
- Browsers auto-detect and extract inline <style> from HTML
- Can read property values from JavaScript
- Great cross-app way of determining login status
- Not much of sensitivity stored in properties yet
- Login detection demo

Design issue

XSSI - remote script inclusion

- This problem is nasty because it gets worse over time!
  - More and more textual constructs are becoming valid JavaScript
- Valid JavaScript from remote domain will execute just fine
- Can't read source but can observe side effects from executing source

Cross-site script inclusion

XSSI: stealable constructs

- Function callback
  - e.g. "callback_func(1, 'data');"
- Setting variables
  - e.g."var result = 100;"
- Function definition
  - e.g. "function blah() { var a = 1; }"
- JavaScript array data (FireFox 2)
  - e.g. "[1, 2, 3, 4]"

XSSI

Future directions

- XML theft
  - Remembering that XHTML and some HTML parses as XML
  - XML is valid JavaScript with E4X support
    - FireFox2, FireFox3
- Theft via JavaScript injection into XML
- Random FireFox bug: XML injection into XML
- Demo: E4X based theft

XSSI

Future directions

- Core language overloads
  - Overload XML constructors
  - Overload Number constructor
  - Overload Error types
  - Overload primitive objects

XSSI

Remediation

- Apply XSRF protection
  - Not feasible to do for all authenticated GETs
- Break JavaScript execution or syntax
  - while(1); is seen a lot
    - What if the "1" fired Number()
  - Prefer to break syntax hard or use something unlikely to be overloadable
  - e.g. "for (;;);" or ")]}"
- Include XML prolog and DTD always

Active MITM against SSL sites

- Mixing Content… it shouldn't be DONE!
- HTTP script cannot access SECURE cookies
- HTTPS pages can load HTTP javascript
- Find insecure script references (CSS works too)
- FORCE the loading of insecure script references over HTTPS

Active MITM against SSL sites

HTTP page with:
<html><body>.....
<script src="http://domain/somejavascript.js"></script>
.....</body></html>

Then force that page to be loaded as HTTPS

Active MITM against SSL sites

- The attacker forces the victim's browser to render the HTTPS site with mixed content
- The HTTP Javascript request/response is tamperable by the attacker
- Now an attacker can MITM an SSL protected site without ANY WARNINGS on many browsers!
- Real Life Examples…
  - https://www.apple.com/
  - https://wordpress.com/

Design issue

Cookie forcing

- Name picked based on Google Search
  - "Cookie forcing" -> 123 hits
- OMG! Name does not have "jacking" in it
  - Call it "cookie force-jacking" if that would make you happy :)

Cookie forcing

What is it?

- The cookie model is still a bit dangerous, even with Secure flag
- "Read" was fixed but not "Write"
- Therefore, http://bank.com/ can overwrite Secure cookies used by https://bank.com/
- This part is well known and not new
  - New part is attack details / delivery mechanism

Cookie forcing

What can evil cookies do?

- Apps often trust their cookies completely
  - I put it there so I can trust it
  - Using https so not expecting integrity violation
- Evil cookie forcing is sidejacking to the max - breaking an https app despite it following (current) best practices

Cookie forcing

XSS via cookie planting

- App escaped the cookie on write...
- ... so no need to be careful on read, right?
- XSS via HTML generation or DOM access
- XSS via JSON eval()

Cookie forcing

XSRF via cookie planting

- One common XSRF protection is to compare nonce with URL param
- For apps at scale, store nonce in cookie
- Now, attacker controls nonce cookie and URL param!

Cookie forcing

App logic cookies

- Some cookies affect app logic
- Irritations
  - Display language
  - Persistently break app
- More seriously
  - Sensitive settings
  - Debug modes

Cookie forcing

Login / logout XSRF

- Not much to stop you dropping in attacker's account's auth cookie
  - (Well, see mitigations to follow)
  - Links nicely to Billy's "Biting the Hand that Feeds you" presentation
- Same applies to nuking existing auth cookie
- Attack can be "silent"
- Theft of data on POST thwarted by XSRF protection
  - How is XSRF token mismatch handled?

Cookie forcing

Mitigations

- Assume cookie data is evil
  - Parse and escape it, don't eval() it
- Sign your sensitive cookies
  - Do not forget to tie to user
  - Best to tie to current session
- Login / logout XSRF is hard
  - You can randomize your cookie names
  - But then must handle attacker's dupes
    - Which is the real one?!

Attacking the paranoid

Accessing your bank?

- Recall the 1 window, 1 tab model
- Hit https bookmark only
- Then you got pwned
  - What happened?

Attacking the paranoid

What's your browser up to?

- Updating anti-phishing / malware lists
- Loading https cert details... over http
- Update pings
- RSS feed updates

Attacking the paranoid

Hmm... plain http

- Attacker can respond with 302 redirect
  - To arbitrary domain
- Browser sends out new request
- Attacker can respond with some cookies
- Set or clear arbitrary cookies on any domain
  - Including those marked Secure
  - Or could shadow existing cookies
- Beauty of attack is silence
  - Background request redirects not noisy on URL bar

Attacking the paranoid

Other opportunties

- Poison cache to affect future sessions
  - http only
- Poison cookies to affect future sessions
- Very stealthy exploit of any XSRF bug
- Scary: self-XSS attacks with "divided login" attack

Attacking the paranoid

Mitigations

- Do not rely on browser https to provide integrity on untrusted networks
- Use VPN to a more trusted network
- Disable browser's features leading to plain http requests (but did you miss one?)
- **Set your proxy to localhost:1 for http protocol**

Demo

For cookie forcing


Gratuitous marmot picture

Q & A