OWASP OWTF
SummerStorm

Abraham Aranguren

OWASP OWTF Project Leader
@7a_ @owtfp
abraham.aranguren@owasp.org

Agenda

- GSoC Overview
- What is OWASP OWTF?
- Status update on OWTF GSoC projects
  - OWTF Reporting
  - OWTF Multiprocessing
  - OWTF MiTM Proxy
  - OWTF Testing Framework
- OWASP Testing Guide with OWTF
- Conclusion

Agenda

- GSoC Overview
- What is OWASP OWTF?
- Status update on OWTF GSoC projects
  - OWTF Reporting
  - OWTF Multiprocessing
  - OWTF MiTM Proxy
  - OWTF Testing Framework
- OWASP Testing Guide with OWTF
- Conclusion

Google Summer of Code (GSoC) Overview

GSoC Stats + Outcome

- OWASP got 11 slots from Google
- OWASP received 84 proposals
- 73 students (87%) could not be selected.
- Final slot breakdown:
  - 4 - OWASP ZAP
  - 4 - OWASP OWTF
  - 1 - OWASP Hackademic
  - 1 - OWASP ModSecurity
  - 1 - OWASP PHP Security Project

http://blog.7-a.org/2013/06/owasp-owtf-gsoc-selection-stats-and-poll.html

OWTF GSoC Overview

- 14 students showed interest (email)
- 11 (79%) students submitted a proposal
- 14 proposals were submitted (16% of 84)
- 5 OWTF proposals ended in the top 11
- 1 student was lost in de-duplication process (accepted by another org)
- 4 OWTF proposals were finally selected (36% of 11)

http://blog.7-a.org/2013/06/owasp-owtf-gsoc-selection-stats-and-poll.html

Why submit for OWTF?

OWTF GSoC student poll summary:

- “It’s python”
- “I like this project”
- “It’s a project I can do with my skills”
- “OWTF is the best project to learn about other tools/security”
- “Other mentors/org didn’t reply” (!)
- “Quick feedback/encouragement/advice”

http://blog.7-a.org/2013/06/owasp-owtf-gsoc-selection-stats-and-poll.html

Selected OWTF Proposals

- Reporting: Assem Chelli
- Multiprocessing: Ankush Jindal
- MiTM Proxy: Bharadwaj Machiraju
- Testing Framework: Alessandro Fanio González

http://blog.7-a.org/2013/06/owasp-owtf-gsoc-selection-stats-and-poll.html

Dedicated OWTF mentors

Without them 3 OWTF students would have been lost (GSoC 1 dedicated mentor x student rule):

Andrés Morales, Andrés Riancho, Azeddine Islam Mennouchi, Gareth Heyes, Hani Benhabiles, Javier Marcos de Prado, Johanna Curiel, Krzysztof Kotowicz, Martin Johns

THANK YOU for stepping up!

Questions?

What is OWASP OWTF?

aka The Offensive (Web) Testing Framework

OWTF = Test/Exploit ASAP

OWTF’s Chess-like approach

OWTF Plugin Groups (-g)

- web: Try to cover the OWASP Testing Guide

owtf.py http://demo.testfire.net (-g web: optional) web only
owtf.py –l web List web plugins

- net: Somewhat like nmap scripts

owtf.py demo.testfire.net (-g net: optional) portscan + probe
NOTE: if a web service is found, web plugins will also run
owtf.py –l net List net plugins

- aux: Somewhat like msfcli in metasploit

owtf.py -f -o Targeted_Phishing SMTP_HOST=mail.pwnlabs.es
SMTP_PORT=25 SMTP_LOGIN=victim SMTP_PASS=victim
EMAIL_FROM=sevena@pwnlabs.es EMAIL_PRIORITY=no
EMAIL_SUBJECT='Test subject' EMAIL_BODY='test_body.txt'
EMAIL_TARGET='victim@pwnlabs.es' Phishing via SET
owtf.pl –l aux List aux plugins

Web Plugin Types (-t)

At least 50% (32 out of 64) of the tests in the OWASP Testing guide can be legally* performed to some degree without permission

* Except in Spain, where visiting a page can be illegal ☺
* This is only my interpretation and not that of my employer + might not apply to your country!

OWTF Report = Chess-like Analysis

You need to understand this to use the OWTF report efficiently ☺

From Alexander Kotov - "Think like a Grandmaster":

1) Draw a list of candidate moves (3-4) 1st Sweep (!deep)

1) Draw up a list of candidate paths of attack = rank what matters

2) Analyse each variation only once (!) 2nd Sweep (deep)

2) Analyse [tool output + other info] once and only once

3) After step 1 and 2 make a move

3) After 1) and 2) exploit the best path of attack

Ever analysed X in depth to only see "super-Y" later?

Demo 1: Admin interface

Watch it: http://www.youtube.com/watch?v=z0n5dYa0WR4

Pre-Engagement: No permission to test preparation

1) Run passive plugins legit + no traffic to target
Sitefinity CMS found

2) Identify best path of attack:

- Sitefinity default admin password
- Public sitefinity shell upload exploits

Engagement: Permission to test exploitation

Try best path of attack first

Demo 1: Outcome

1 minute after getting permission

Demo 1: Outcome

5 minutes after getting permission

Demo 2: Crossdomain

Watch it: http://www.youtube.com/watch?v=ni3Htb4Ya-U

Attack preparation (pre-engagement safe) preparation

1) Run semi-passive plugins legit

Missconfigured crossdomain, fingerprint wordpress version

2) Identify best path of attack:

crossdomain + phishing + wordpress plugin upload + meterpreter

3) Replicate customer environment in lab

4) Prep attack: Adapt public payloads to target

5) Test in lab

Launching the attack exploitation

1) Tested attack works flawlessly on the first shot

2) Pivot

3) Show impact

OWTF Financials: Ideas plz ☺

Funding granted so far (THANK YOU Brucon + Google!):

- €5,000 – Brucon 5x5

http://blog.brucon.org/2013/02/the-5by5-race-is-on.html

- $2,000 – GSoC ($500 x student)

What should we do with that money?

Questions?

Status update
on
OWTF GSoC Projects

OWTF Reporting by Assem Chelli

Gareth Heyes (@garethheyes)
Azeddine Islam Mennouchi, Hani Benhabiles, Johanna Curiel, Abraham Aranguren

Reporting Agenda

- Old report limitations
- Reporting goals
- Pre-implementation research
- Prototype voting/feedback
- Upcoming features

Old Report != Sexy

- Online sample: http://goo.gl/iZshVJ

Old report limitations

- Complicated + hard to understand
- Poor loading time of “big” reports (i.e. 30+ websites)
- Not cross-browser compatible (Firefox only)
- Inability to suit various screen sizes
- Not visually appealing :(
- Direct HTML generation from python code

Reporting Goals

- UI simplification + intuitiveness
- Better load time + responsiveness
- Cross-browser compatibility
- Improved screen size support (i.e. mobile users, etc)
- Improve visual appeal with community backing
- Build a skin system Users can choose/create skins
- Move HTML into template files:

!python = designer-friendly = more people can help us

- Optimise click flow + mouse movement

Pre-implementation research

Twitter bootstrap gives us:

- Browser compatibility
- Pre-configured layouts
- Pre-defined styles
- Icon sets
- jQuery plugin integration
- Responsiveness + Simplicity

Pre-implementation research

Jinja2 gives us:

- A python templating engine
- Python-like expressions
- Templates evaluated in a sandbox

Prototype Voting/Feedback

Demo 3:
Online Survey Results

https://docs.google.com/file/d/0B5P-99g5h0-6Znd5ajZKbVJqbU0

Want to vote? ☺ Shortcut: http://7-a.org + search “voting”

Survey: https://docs.google.com/forms/d/1w613Y-rwPMw454k2oAd2MuOIe8zDg6YNejaMLg29CUQ/viewform

Demo 4: Voted Prototype

Play with it! http://assem-ch.github.io/owtf-report-prototypes/Prototypes/BS_default_white_default/index.html

Upcoming Features (WIP)

- Implement skin system
- Implement chosen prototype
- Extraction of CSS/HTML into templates
- Sub-report loading via AJAX
- Default plugin vulnerability rankings

Questions?

OWTF Multiprocessing by Ankush Jindal

Andrés Riancho (@w3af)
Abraham Aranguren

Multiprocessing Agenda

- Multiprocessing goals
- Pre-implementation research
- Development challenges
- Net plugins demo
- Upcoming features

Multiprocessing Goals

- Reduce scanning time
- Port of OSCP scripts into OWTF net plugins
- Scan multiple targets in parallel
- Rational usage of disk/RAM/CPU
- Stability + Reliability = !crash
- Identify + parallelise bottleneck components:

Plugin execution, Reporting

Pre-Implementation Research

- Tested candidate libraries:

Library	Multiprocessing	Threading	gevent (distributed)
Shared Memory	No	Yes	Yes

- Results:

1. Shared memory led to incorrect results in legacy code
2. Multiprocessing performed better or approx. the same
3. Threading = GIL FUD on multiple-core machines ☺

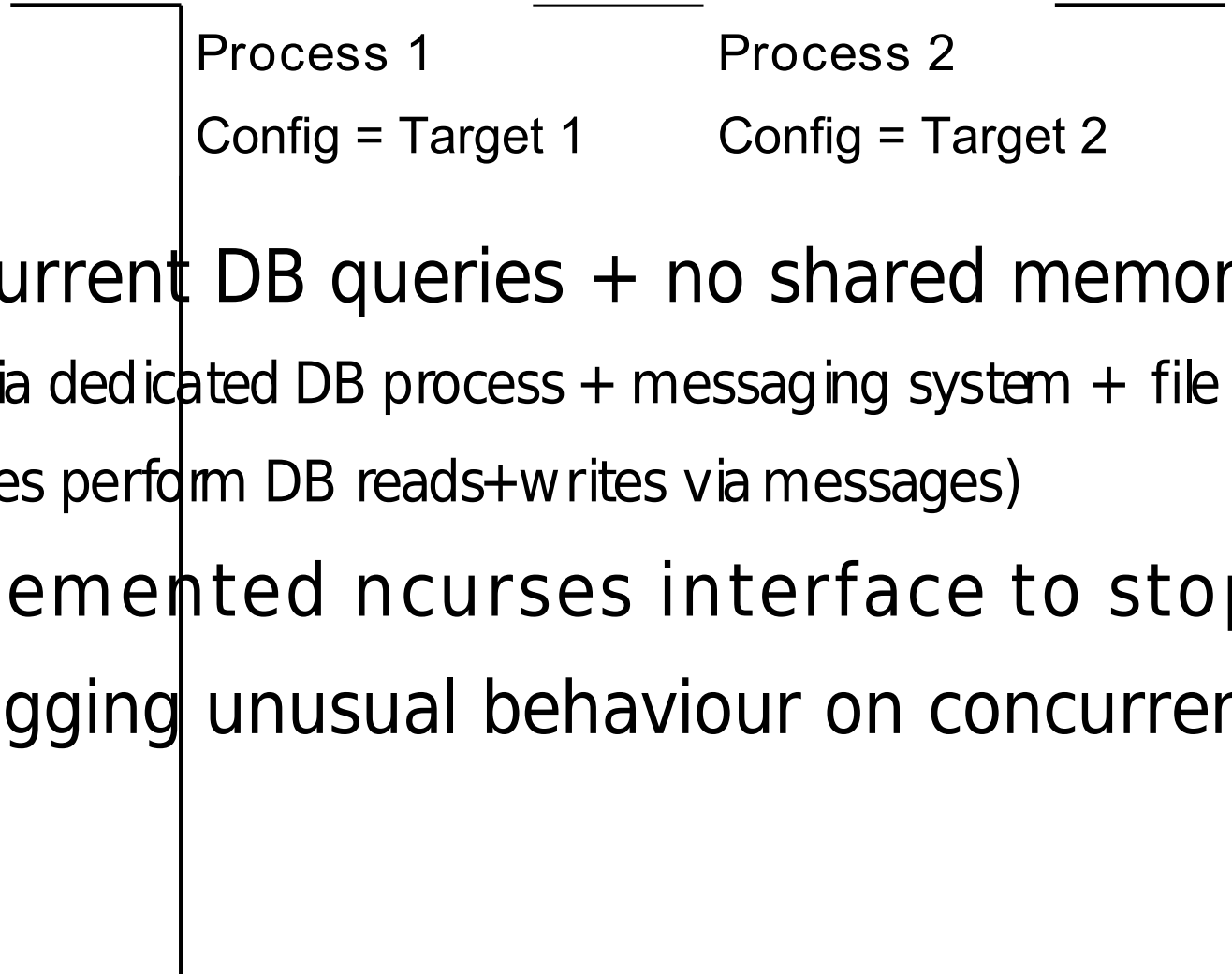
- Conclusion:

Multiprocessing for plugins, Threading for smaller tasks

Challenges during development

- OWTF resets config on the fly via “SwitchToTarget”

Solved via memory separation in multiprocessing



- Concurrent DB queries + no shared memory + File DB:

Solved via dedicated DB process + messaging system + file locks for integrity

(Processes perform DB reads+writes via messages)

- Implemented ncurses interface to stop OWTF
- Debugging unusual behaviour on concurrent processes ☺

Demo 5: Net Plugins

Watch it: http://youtu.be/_I_Wv6VuyQk

Port of the OSCP scripts into OWTF:

- Ping sweep + DNS zone transfers + port scanning
- Port scanning via nmap using “waves” (--portwaves)

owtf.py --portwaves=10,100,1000 target.com

First scan “top 10” ports, then “remaining until top 100”, ..

- Firing relevant net plugins depending on ports open

Net plugins implement:

- Vulnerability probing of network services (i.e. ftp, smtp,..)

Upcoming Features

- Plugin profiling for better resource usage:

Monitor resources to determine “launchable” plugins depending on [load + expected resource consumption]

- Reporter process:

To run in parallel + reduce report re-assembly iterations (i.e. instead of re-assemble once x plugin execution)

- Identify + parallelise other bottleneck components

Questions?

OWTF MiTM Proxy
by
Bharadwaj Machiraju

Krzysztof Kotowicz (@kkotowicz)
Javier Marcos de Prado, Martin Johns,
Abraham Aranguren

MiTM Proxy Agenda

- MiTM Proxy Goals
- Pre-implementation research
- Development challenges
- Examples of working functionality ☺
- Performance benchmarks
- Upcoming features

MiTM Proxy Goals

- Extended grep plugin coverage:

  1) Data from manual browsing

  2) Data from proxified tools

- Tool proxification (if launched from OWTF)
- SSL MiTM
- Proxy cache: Avoid redundant requests
- Request Throttling based on target responsiveness

(i.e. avoid unintended DoS)

- Intelligent request retries

(i.e. ensure HTTP response retrieval where possible)

Pre-Implementation Research

- Goal:

Select best python proxy framework best starting point

- Test Cases:

Speed, HTTP Verb support, HTTP/1.1, HTTPS support, etc.

- Frameworks:

Twisted, Mitmproxy, Tornado, Honeyproxy

- Verdict: Tornado

Best [performance + feature-set + reusability]

Pre-Implementation Research

MiTM Proxy

Pre-Implementation

Research Doc

https://docs.google.com/file/d/0B5P-99g5h0-6NjJDaF9BUGpVY28

Development Challenges

- Tornado: Is a python web framework (!proxy)

Pros	Cons
Scalability: Tens of thousands of connections	Not built to make proxy servers
Server + Client = Proxy	Client is more limited than server. Solution: Use tornado's async curl client

- SSL MiTM: on-the-fly certificate generation, etc.
- Proxy cache: Race condition handling
- Tool Proxification: Not all tools could be proxified

BUT Tool Proxification for tools with proxy CLI options IS working ☺

Proxy SSL MiTM is working ☺

Proxy Cache is working ☺

Race-condition handling is working ☺

Performace Benchmarks

Upcoming features

- Improved grep plugins: Run on all transactions
- Request Throttling based on target responsiveness

(i.e. avoid unintended DoS)

- Intelligent request retries

(i.e. ensure HTTP response retrieval where possible)

- Cookie based authentication

At proxy level = Ability to scan authenticated portions of a website.

- Plug-n-Hack support: Upcoming Mozilla standard DONE

Questions?

OWTF Testing Framework
by
Alessandro Fanio González

Andrés Morales Zamudio (@andresmz)
Abraham Aranguren

Testing Framework Agenda

- Importance of testing
- Testing framework goals
- Pre-implementation research
- Development challenges
- Initial focus: Unit testing
- New focus: Functional testing
- Upcoming features

Importance of testing

- Improve code quality
- Ensure everything works as expected
- Prevent unintentional bugs:

While developing new features or fixing other bugs

- Provide stability to the project

Testing Framework Goals

- Writing OWTF tests = As easy as possible
- Ensure OWTF integrity after code changes:
  1. Automated tests to verify OWTF modules behave as expected (unit tests)
  2. Automated tests to verify OWTF security test output is as expected (functional tests)

Pre-implementation research

Goals: Determine best starting point

1. Select best testing/mocking library for unit tests
2. Select best mock web server for functional tests

Tests:

1. Feature-set comparison among many mocking libraries
2. Reuse of Bharadwaj's research (for mock web server)

Results:

1. Best mock library for OWTF = Flexmock
2. Best mock web server for OWTF = Tornado

Development Challenges

- Understand internal OWTF components
- Extend the testing library to complete features
- Make the testing framework easy to use:

Generate classes and methods dynamically, using metaclasses and introspection

- Fix broken tests due to fast-moving codebase

Due to initial unit testing focus

Initial focus: Unit testing

Important metric for unit testing = code coverage

Test coverage:

Number of executed lines of code after running all tests

When we run the entire test suite:

1. An HTML code coverage report is generated

2. Lines executed x file can be viewed in the report

Current OWTF code coverage = 58%

New focus: Functional testing

Unit test approach	Functional test approach
Pro: Fast	Con: Slower
Pro: Isolated	Con: Not isolated
Pro: Code coverage metrics (i.e. are we at 100% or not?)	Con: No code coverage metrics
Con: Harder to write (i.e. you kinda have to love/know TDD ☺)	Pro: Easier to write (i.e. closer to command-line usage)
Con: Code dependent (i.e. refactoring = broken test)	Pro: Code independent (i.e. refactoring != broken test)
Con: Difficult to create tests for security edge cases (i.e. unusual web server behaviour)	Pro: Easier to create tests for security edge cases (i.e. unusual web server behaviour)
Con: Can't find bugs due to third-party tools/incompatibilities	Pro: Will find bugs due to third-party tools/incompatibilities

Demo 6: A testing example

Watch it: http://youtu.be/ypLwjzORKfQ

Functional testing:

- Set the web server to return a custom robots.txt file, and start the server
- Write tests (almost) as if you were using OWTF from the command line: run the Spiders_Robots_and_Crawlers plugin
- Assert that the URLs contained in robots.txt are in the OWTF output

Unit testing:

- Show code coverage report from initial project focus

Upcoming features

Functional tests for:

1. web plugins: OWASP Testing Guide coverage
2. net and aux plugins: PTES coverage

- Automated Continuous Integration:

Run tests automatically after each commit

Questions?

OWASP Testing Guide with OWASP OWTF

Context consideration:

Case 1 robots.txt Not Found

…should Google index a site like this?

Or should robots.txt exist and be like this?

```
User-agent: *
Disallow: /
```

Case 1 robots.txt Not Found - Semi passive

- Direct request for robots.txt
- Without visiting entries

Case 2 robots.txt Found – Passive

- Indirect Stats, Downloaded txt file for review, “Open All in Tabs”

OWTF HTML Filter challenge: Embedding of untrusted third party HTML
Defence layers:
1) HTML Filter: Open source challenge
Filter 6 unchallenged since 04/02/2012, Can you hack it? ☺
http://blog.7-a.org/2012/01/embedding-untrusted-html-xss-challenge.html
2) HTML 5 sanboxed iframe
3) Storage in another directory =cannot access OWTF Review in localStorage

Start reporting!: Take your notes with fancy formatting

Step 1 – Click the “Edit” link

Step 2 – Start documenting findings +Ensure preview is ok

Start reporting!: Paste PoC screenshots

The magic bar ;) – Useful to generate the human report later

Passive Plugin

Step 1- Browse output files to review the full raw tool output:

Step 2 – Review tools run by the passive Search engine discovery plugin:

Was your favourite tool not run?
Tell OWTF to run your tools on: owtf_dir/profiles/resources/default.cfg (backup first!)

Tool output can also be reviewed via clicking through the OWTF report directly:

The Harvester:

- Emails
- Employee Names
- Subdomains
- Hostnames

http://www.edge-security.com/theHarvester.php

Metadata analysis:

- TODO: Integration with FOCA when CLI callable via wine (/cc @chemaalonso ☺)
- Implemented: Integration with Metagoofil

http://www.edge-security.com/metagoofil.php

Inbound proxy not stable yet but all this happens automatically:
robots.txt entries added to “Potential URLs”
URLs found by tools are scraped +added to “Potential URLs”
During Active testing (later):
“Potential URLs” visited +added to “Verified URLs” +Transaction log

All HTTP transactions logged by target in transaction log

Step 1 – Click on “Transaction Log”

Step 2 – Review transaction entries

Step 3 – Review raw transaction information (if desired)

Step 1 - Make all direct OWTF requests go through Outbound Proxy: Passes all entry points to the tactical fuzzer for analysis later

Step 2 - Entry points can then also be analysed via tactical fuzzer:

Goal: What is that server running?

Manually verify request for fingerprint:

Whatweb integration with non-aggresive parameter (semi passive detection):

https://github.com/urbanadventurer/WhatWeb

Fingerprint header analysis: Match stats

Convenient vulnerability search box (1 box per header found ☺):
Search All Open all site searches in tabs

NVD - http://web.nvd.nist.gov - CVSS Score =High


OSVDB - http://osvdb.org - CVSS Score =High

http://www.securityfocus.com - Better on Google

Passive Fingerprint analysis

http://toolbar.netcraft.com - Passive banner grab,etc.

- CMS
- Widgets
- Libraries
- etc

http://builtwith.com

Search in the headers without touching the site:

http://www.shodanhq.com/

Passive suggestions
- Prepare your test in a terminal window to hit “Enter” on “permission minute 1”

What else can be done with a fingerprint?

Environment replication
Download it .. Sometimes from project page ☺

Also check http://www.oldapps.com/, Google, etc.

Static Analyis, Fuzz, Try exploits, ..

RIPS for PHP: http://rips-scanner.sourceforge.net/

Yasca for most other (also PHP):

http://www.scovetta.com/yasca.html

Questions?

http://www.robtex.com - Passive DNS Discovery

http://centralops.net

http://centralops.net

Has Google found error messages for you?

Check errors via Google Cache

The link is generated with OWTF with that box ticked: Important!

https://www.ssllabs.com/ssldb/analyze.html

Pretty graphs to copy-paste to your OWTF report ☺

https://www.ssllabs.com/ssldb/analyze.html

Do not forget about Strict-Transport-Security!
sslstrip chances decrease dramatically:
Only 1st time user visits the site!

Not found example:

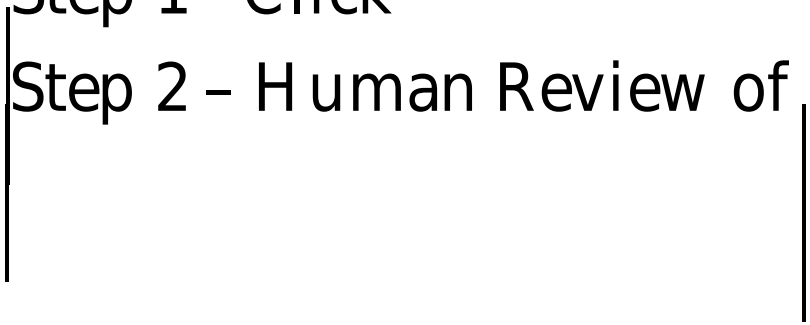
Found example:

HTML content analysis: HTML Comments

Efficient HTML content matches analysis

Step 1 - Click

Step 2 – Human Review of Unique matches



Efficient HTML content matches analysis

Step 1 - Click

Step 2 –Review Unique matches (click on links for sample match info)

Want to see all? then click

HTML content analysis: CSS and JavaScript Comments (/**/)

HTML content analysis: Single line JavaScript Comments (//)

HTML content analysis: PHP source code

HTML content analysis: ASP source code

Questions?

If you find an admin interface don’t forget to ..
Google for default passwords:

Disclaimer: Permission is required for this

http://centralops.net

Is the login page on “http” instead of “https”?

Pro Tip: When browsing the site manually ..
... look carefully at pop-ups like this:

Consider (i.e. prep the attack):

Firesheep: http://codebutler.github.com/firesheep/
SSLStrip: https://github.com/moxie0/sslstrip

Mario was going to report a bug to Mozilla and found another!

A buse user/member public search functions:

- Search for “” (nothing) or “a”, then “b”, ..
- Download all the data using 1) +pagination (if any)
- Merge the results into a CSV-like format
- Import +save as a spreadsheet
- Show the spreadsheet to your customer

Analyse the username(s) they gave you to test:

- Username based on numbers?

USER12345

- Username based on public info? (i.e. names, surnames, ..)

name.surname

- Default CMS user/pass?

Part 1 – Remember Password: Autocomplete

Good	Bad
Via 1) <form … autocomplete="off"> Or Via 2) <input … autocomplete="off">	<form action="/user/login" method="post"> <input type="password" name="pass" />

Manual verification for password autocomplete (i.e. for the customer)
Easy "your grandma can do it" test:

1. Login
2. Logout
3. Click the browser Back button twice*
4. Can you login again –without typing the login or password- by re-sending the login form?

Can the user re-submit the login form via the back button?
*Until the login form submission

Other sensitive fields: Pentester manual verification

- Credit card fields
- Password hint fields
- Other

Part 2 - Password Reset forms

Manually look at the questions / fields in the password reset form

- Does it let you specify your email address?
- Is it based on public info? (name, surname, etc)
- Does it send an email to a potentially dead email address you can register? (i.e. hotmail.com)

Goal: Is Caching of sensitive info allowed?

Manual verification steps: “your grandma can do it” ☺ (need login):

1. Login
2. Logout
3. Click the browser Back button
4. Do you see logged in content or a this page has expired error / the login page?

Manual analysis tools:

- Commands: curl –i http://target.com
- Proxy: Burp, ZAP, WebScarab, etc
- Browser Plugins:

https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/

https://addons.mozilla.org/en-US/firefox/addon/firebug/

HTTP/1.1 headers

Good

Cache-Control: no-cache

Bad

Cache-control: private

HTTP/1.0 headers

Good

Pragma: no-cache
Expires: <past date or illegal (e.g. 0)>

Bad

Pragma: private
Expires: <way too far in the future>

The world

Good

https://accounts.google.com

Cache-control: no-cache, no-store
Pragma: no-cache
Expires: Mon, 01-Jan-1990 00:00:00 GMT

Bad

No caching headers =caching allowed

HTTP/1.1 200 OK
Date: Tue, 09 Aug 2011 13:38:43 GMT
Server: ....
X-Powered-By: ....
Connection: close
Content-Type: text/html; charset=UTF-8

Repeat for Meta tags

Good	Bad
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">	<META HTTP-EQUIV="Cache-Control" CONTENT="private">

Step 1 – Find CAPTCHAs: Passive search

Offline Manual analysis:

- Download image and try to break it
- Are CAPTCHAs reused?
- Is a hash or token passed? (Good algorithm? Predictable?)
- Look for vulns on CAPTCHA version

CAPTCHA breaking tools

PWNtcha - captcha decoder - http://caca.zoy.org/wiki/PWNtcha

Captcha Breaker - http://churchturing.org/captcha-dist/

Manually Examine cookies for weaknesses offline

Base64 Encoding (!=Encryption ☺)	Decoded value
MTkyLjE2OC4xMDAuMTpvd2FzcHVzZXI6cGFzc3dvcmQ6MTU6NTg=	owaspuser:192.168.100.1:a7656fafe94dae72b1e1487670148412

Questions?

http://hackvertor.co.uk/public

Lots of decode options, including:

- auto_decode
- auto_decode_repeat
- d_base64
- etc.

http://hackvertor.co.uk/public

F5 BIG-IP Cookie decoder:

http://blog.taddong.com/2011/12/cookie-decoder-f5-big-ip.html

- Secure: not set=session cookie leaked=pwned
- HttpOnly: not set =cookies stealable via JS
- Domain: set properly
- Expires: set reasonably
- Path: set to the right /sub-application
- 1 session cookie that works is enough ..

Manually check when verifying credentials during pre-engagement:
Login and analyse the Session ID cookie (i.e. PHPSESSID)

Good	Bad (normal +by default)
Before: 10a966616e8ed63f7a9b741f80e65e3c After: Nao2mxgho6p9jisslen9v3t6o5f943h	Before: 10a966616e8ed63f7a9b741f80e65e3c After: 10a966616e8ed63f7a9b741f80e65e3c

IMPORTANT: You can also set the session ID via JavaScript (i.e. XSS)

Session ID:

- In URL
- In POST
- In HTML

Example from the field:
http://target.com/xxx/xyz.function?session_num=7785

Look at unauthenticated cross-site requests:

http://other-site.com/user=3&report=4
Referer: site.com

Change ids in application: (ids you have permission for!)
http://site.com/view_doc=4

Headers Enabling/Disabling Client-Side XSS filters:

- X-XSS-Protection (IE-Only)
- X-Content-Security-Policy (FF >=4.0 +Chrome >=13)

Review JavaScript code on the page:

```
<script>
document.write("Site is at: " +document.location.href +".");
</script>
```

Sometimes active testing possible in your browser
(no trip to server =not an attack =not logged):
http://target.com/...#vulnerable_param=xss

http://blog.mindedsecurity.com/2010/09/twitter-domxss-wrong-fix-and-something.html

Did Google find SQLi for you?

<!--#exec cmd="/bin/ls /" -->
<!--#INCLUDE VIRTUAL="/web.config"-->

1. Browse Site
2. Time requests
3. Get top X slowest requests
4. Slowest = Best DoS target

Google searches: inurl:wsdl site:example.com

Public services search:
http://seekda.com/
http://www.wsindex.org/
http://www.soapclient.com/

WSDL analysis

Sensitive methods in WSDL?

i.e. Download DB, Test DB, Get CC, etc.

http://www.example.com/ws/FindIP.asmx?WSDL

```
<wsdl:operation name="getCreditCard" parameterOrder="id">
    <wsdl:input message="impl:getCreditCardRequest" name="getCreditCardRequest"/>
    <wsdl:output message="impl:getCreditCardResponse"
name="getCreditCardResponse"/>
</wsdl:operation>
```

Same Origin Policy (SOP) 101

1. Domain A's page can send a request to Domain B's page from Browser
2. BUT Domain A's page cannot read Domain B's page from Browser

http://www.ibm.com/developerworks/rational/library/09/rationalapplicationdeveloperportaltoolkit3/

• Request == Predictable Pwned "..can send a request to Domain B" (SOP)

CSRF Protection 101:

• Require long random token (99% hidden anti-CSRF token) Not predictable

• Attacker cannot read the token from Domain B (SOP) Domain B ignores request

Potentially Good	Bad
Anti-CSRF token present: Verify with permission	No anti-CSRF token

Similar to CSRF:
Is there an anti-replay token in the request?

Potentially Good	Bad
Anti-CSRF token present: Verify with permission	No anti-CSRF token

1) Passive search for Flash/Silverlight files +policies:

Flash file search:

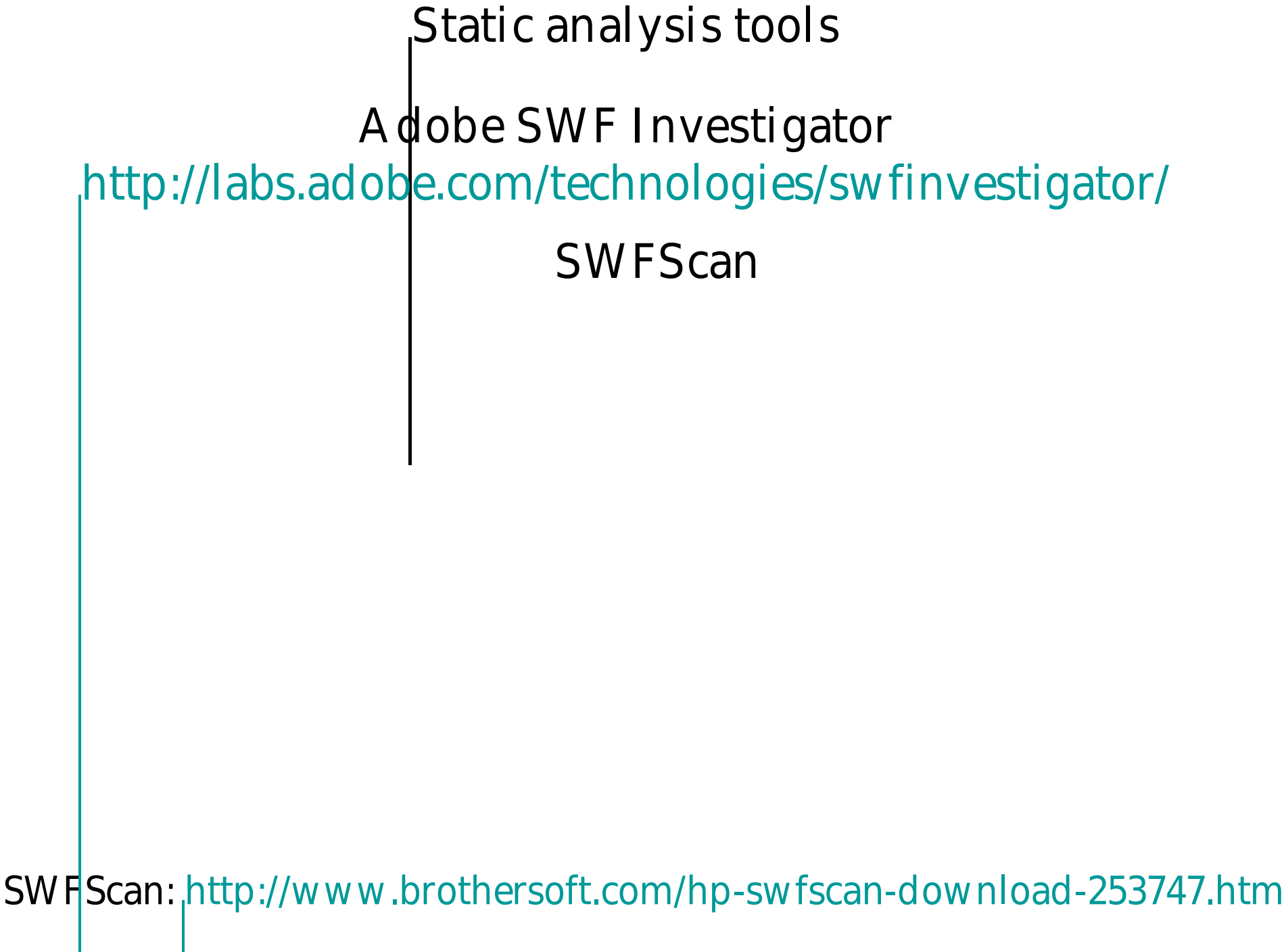
Silverlight file search:

Static analysis: Download + decompile Flash files

```
$ flare hello.swf
```

Flare: http://www.nowrap.de/flare.html

Flasm (timelines, etc): http://www.nowrap.de/flasm.html


Static analysis tools
Adobe SWF Investigator
http://labs.adobe.com/technologies/swfinvestigator/
SWFScan
SWFScan: http://www.brothersoft.com/hp-swfscan-download-253747.html

Active testing ☺

1) Trip to server =need permission

http://target.com/test.swf?xss=foo&xss2=bar

2) But … your browser is yours:

No trip to server =no permission needed

http://target.com/test.swf#?xss=foo&xss2=bar

Good news: Unlike DOM XSS, the #trick will always work for Flash Files

Some technologies allow settings that relax SOP:

- Adobe Flash (via policy file)
- Microsoft Silverlight (via policy file)
- HTML 5 Cross Origin Resource Sharing (via HTTP headers)

Cheating: Reading the policy file or HTTP headers !=attack

http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security.html

Policy file retrieval for analysis

CSRF by design read tokens =attacker WIN

Flash / Silverlight - crossdomain.xml

```
<cross-domain-policy>
<allow-access-from domain="*"/>
</cross-domain-policy>
```

Bad defence example: restrict pushing headers accepted by Flash:
All headers from any domain accepted

```
<allow-http-request-headers-from domain="*" headers="*" />
```

Flash: http://kb2.adobe.com/cps/403/kb403185.html

CSRF by design read tokens =attacker WIN

Silverlight - clientaccesspolicy.xml

```
<?xml version="1.0" encoding="utf-8"?><access-policy><cross-domain-access><policy>
    <allow-from http-request-headers="SOAPAction">
        <domain uri="*"/>
    </allow-from>
    <grant-to><resource path="/" include-subpaths="true"/></grant-to>
  </policy></cross-domain-access></access-policy>
```

Silverlight: http://msdn.microsoft.com/en-us/library/cc197955%28v=vs.95%29.aspx

Need help?

Workshop exercise

1) Install swftools:

```
wget http://www.swftools.org/swftools-0.9.2.tar.gz
tar xvfz swftools-0.9.2.tar.gz
cd swftools-0.9.2
sh ./configure
make
make install
whereis swfdump      Check that we have swfdump installed now
swfdump: /usr/local/bin/swfdump
```

Workshop exercise (continued)

2) Analyse vulnerable file:

wget http://demo.testfire.net/vulnerable.swf Download vulnerable file

swfdump -a vulnerable.swf >vulnerable.txt Disassemble flash file

grep -B1 GetVariable vulnerable.txt| tr " " "\ n"| grep '("'| sort –u Get FlashVars

("empty_mc")

("externalInterfaceVar")

("flash")

("font")

("fontTxtFieldExists")

("fontVar")

("getUrlBlankVar")

("getUrlJSParam")

("getUrlParentVar") Used in this example

…

Workshop exercise (continued)

3) Verify using the "#" trick (payload not sent to target):

http://demo.testfire.net/vulnerable.swf#getUrlParentVar=javascript:alert('pwned!')

Click on "Get URL (parent)" for example above

And you get:
XSS ☺

UI Redressing protections:

- X-Frame-Options (best)
- X-Content-Security-Policy (FF >=4.0 +Chrome >=13)
- JavaScript Frame busting (bypassable sometimes)

Good

Bad

X-Frame-Options: Deny

Andrew Horton's "Clickjacking for Shells":
http://www.morningstarsecurity.com/research/clickjacking-wordpress

Krzysztof Kotowicz's "Something Wicked this way comes":
http://www.slideshare.net/kkotowicz/html5-something-wicked-this-way-comes-hackpra
https://connect.ruhr-uni-bochum.de/p3g2butmrt4/

Marcus Niemietz's "UI Redressing and Clickjacking":
http://www.slideshare.net/DefconRussia/marcus-niemietz-ui-redressing-and-clickjacking-about-click-fraud-and-data-theft

Adi Mutu (@an_animal), Alessandro Fanio González, Anant Shrivastava, Andrés Morales, Andrés Riancho (@w3af), Ankush Jindal, Assem Chelli, Azeddine Islam Mennouchi, Bharadwaj Machiraju, Chris John Riley, Gareth Heyes (@garethheyes), Hani Benhabiles, Javier Marcos de Prado, Johanna Curiel, Krzysztof Kotowicz (@kkotowicz), Marc Wickenden (@marcwickenden), Marcus Niemietz (@mniemietz), Mario Heiderich (@0x6D6172696F), Martin Johns, Michael Kohl (@citizen428), Nicolas Grégoire (@Agarri_FR), Sandro Gauci (@sandrogauci), OWASP Testing Guide contributors

All those OWTF students that tried to participate in the GSoC even if they couldn't make it this time

Finux Tech Weekly – Episode 17 – mins 31-49
http://www.finux.co.uk/episodes/mp3/FTW-EP17.mp3
Finux Tech Weekly – Episode 12 – mins 33-38
http://www.finux.co.uk/episodes/mp3/FTW-EP12.mp3
http://www.finux.co.uk/episodes/ogg/FTW-EP12.ogg
Exotic Liability – Episode 83 – mins 49-53
http://exoticliability.libsyn.com/exotic-liability-83-oh-yeah

Q & A

Contact/Links:

http://owtf.org
@7a_ @owtfp
abraham.aranguren@owasp.org