1) CSRF : (WSTG-SESS-05)
- forces an to do an action unknowingly or unwantedly.

↳ XSS → dynamic
↳ CSRF → static

2) XSS vs CSRF:

XSS = uses script CSRF = uses static content

XSS ≠ CSS, as CSS = Cascading Style Sheets.

3) CSRF Visualization: I Know.

→ Transaction that does not req: dynamic content
→ submitted without CSRF tokens / mechanisms
→ weak tokens / predictable tokens.

4) ZAP Anti CSRF Test form:

→ POC to find CSRF flaws using POST Reqs.
→ "Generate Anti CSRF forms"
→ Similar to BURP → Generate CSRF POC (PRO only)

→ Day 5: Part 2: CSRF Exercise

→ Day 5: Part 3: Logic Attacks

- Logic Flaws = Business logic bypass
- Not prominent in industry. (not prominently discussed)
- DIFFICULT TO RESOLVE:

→ Every app has many logic flows and many ways to bypass the logic context

→ Depends on application CONTEXT

2) Flow of Logic Flaws:

1 → 2 → 3 → 4

1 → 2 → 3 ↴ ↱ 4

3.5

3) Logic Flaws : Manual Discovery:

- Automated Tools → logic flaw → unnoticed

∴ pronounced in auto scanners

- Manual Discovery = prominent + most effective.
- OWASP TG = infinite combos = How to Proceed ?

WSTG - BVSL - 01

WSTF - BVSL - 02

⋮

09

Day 5 : Part 4 : Python for Web App Pentest:

- Python: → Interpreted (GUIDO VAN ROSSUM)
  → 1980's (Monty Python's Flying Circus)
  → Public Release → Feb 1991
  → derived from ABC Language
  (Glue of Scripting stuff) (RAD)
- why Python? → RAD (Rapid App development)
  → easy
  → Right tool for the job
  → Many libs, many integrations

But, always be TOOL AGNOSTIC

- Python Availability (python -v) (usually default on Linux / MAC)
- Python 2 vs Python 3
  (legacy) (future and present of language)
- Python = Simple to use (STUS)

② why to use python when so many tools?
→ Customize to you
→ use anytime, anywhere as personally needed.

③ Learn Python Basics

④ Python Data Types: → String
→ Boolean
→ int
→ Float

- if / elif / else

⑤ Python Loops:

⑥ Python Lists + Dictionaries:

⑦ Python Web Libs: ↦ urllib, urllib 2, 3
↳ httplib, 2, Requests

Eg: of HTTPLib in python coding.

⑧ POST via Requests
Requests + SSL/TLS (r = requests.get('https://xyz.com'))

⑨ Printing (a lot of) Together
(splitlines() removes the newline at end of each line) (./timing.py | sort -n | tail -10)

Day 5: Part 5: Python [EXERCISE]

Day 5: Part 6: wPscan + Exploit DB

① wPscan(1) → word press scanner (automated)
most tools = General or Manual (not specific)
wPscan = tool SPECIFIC for wordpress
↳ enumerating WP flaws / vulns
↳ WP is very ubiquitous (public / internal) sites
Check videos for wPscan tool and cmds.
↳ Actively updated; VVLN scanner for Wordpress

Day5: Part 7: WPScan + Exploit DB [EXERCISE]

Day5: Part 8: BURP Scanner [PRO]

- Burp scanner (Automated scanner, only in PRO)
- Caveat Emptor
- Passive: what is present already. No extra req
- Active: makes new req and gathers responses
- Live scanning = Passive + Active (Active scan while traversing)

→ Always stay in scope.

→ Active scans → can now run multiple scans in parallel (not one MONOLITHIC scan)
- Allows ↑ efficiency (as its better than a single large scan, huge)

→ Advanced scope control (What(not) to SCAN)

→ Scanning particular Nodes.
- Launch separate
- Add to task and scan (if found a new branch)

→ Burp scanner confidence levels
- Certain
- Firm
- Tentative
- Info

→ Burp (Retesting + Remediation)
- → open scan
- → check the item to be retested
- → Right click → Audit item (to retest)

Day 5 : Part 9 : Metasploit (Tool)

- Most popular → exploitation framework
- Made in RUBY (largest RUBY ever)
- Both O/Source + commercial devs available.

② MS+ Web Testing (N/W well known, but also has WEB functions).

③ Auxilliary → perform functions like scanning, spidering, crawling and queuing basic web server. (aux/scanner/http)

>150 entries in A/scanner/http

④ MS Crawlers : → A/c/msfcrawler
→ A/s/http/crawler } 2 scanners

Crawl/Scan is great but cant replace Burp/ZAP

⑤ DB_Import : (used to import other tools)

→ Most efficient way to prime MS = MS+db_import.

→ Tools output files —db_imp→ compatible DB for MS (structure same).

→ Hence external tools ⇌ MS, due to DB_imp

⑥ MS Integration :

① DB_Import : Any tool with Msploit

⑦ Beef + MSploit:

Hooked Browser + Metasploit =>
- better, faster exploits.
- expands capabilities for XSS.

⑧ Beef + MSploit: → Lim privs → ↓ persistence

→ Configure stuff. (beef/config.yaml).

→ on msploit check for Msploit enabled = true.

→ Msploit username + password.

i) Update config file

ii) Start MS RPC using msfconsole =>

load msgrpc ServerHost=127.0.0.1 Pass=pass

iii) Update config.yaml in ext/msploit with details of the RPC started

iv) Start BeEF

v) Inject some Msploit.

⑨ SQLMap ↔ MSploit ⟷

sqlmap → msploit or (--os-pwn, --priv-esc, --msf-path)

msploit → sqlmap

⑩ MSploit + known Vulns.

- Main use → known vulns
- custom app testing = aux modules.

Examples: CMS, DB's, SQLi, Shellshock, Heartbleed, Drupalgeddon

(11) Drupal: (like WP) (serve content to end users)

WP and Drupal: Commonly deployed CMS.

Joomla (3rd)

WP + Plugins
Drupal + Modules
} extend functionality

(12) Drupalgeddon:

Unauth SQLi on all Drupal 7 systems (Oct 15, 2014) (11 pm UTC) (assume all systems are hacked)

- → Unfiltered data access
- → RCE
- → Local Privesc etc.

Widespread auto exploits (in hours)

Drupalgeddon Details: (Drupal used PrepStmt() for SQLi)

Input → expandArguments() → PDO → Emulate Prep
(array → objects compatible)

Stmts → Emulated PS allows multiple queries as 1 statement.

∴ any Input → expandArguments() = EXPLOIT!!

MSploit: multi/http/drupal_drupageddon

eg: use exploit/multi/http/drupal_drupageddon.

Day5: Part 10: MSploit: Exercise

Day 5: Part 11: Drupalgeddon 2: exercise

Day 5: Part 12: When Tools fail

- what to do when tools fail
  - Diff in server configs
  - Tool Quality
  - some options can/cant
  - Pentest ⇒ Non Determine
- Taking it to the Next Level: (Abandon exploitation? NO)
  - Learn all about MediaWiki
    - MSploit
      - Djvu (dint work; default)
      - PDFgmages (worked; not default, specify the file name to MSploit)

(exploit/multi/http/mediawiki_thumb)

Day 5: Part 13: Exercise when Tools Fail

Day 5: Part 14: Business of Pentesting:

Prepping

① Skills, Toolkit → Pentest specifics
(Personal Prep) → Comm^n Plans

Skills → Build skills; Stay informed; Join chapters. (community priv)
Practices,

Toolkit → ◦ Tools; Backup Test Results
◦ Custom scripts
◦ Vuln Scanners

Pre-Engagement Specifics: (Skills, Toolkit, Backup)

◦ Permission ◦ Scope ◦ Rules of Engagement.
◦ Source details ◦ Source code (WBox) ◦ Filtering
◦ Test Accts ◦ Backups ◦ Architecture Diagrams.
◦ green zones

③ Comm^n Plans: → Include who all in Emails
→ Doc frequencies, date/time of standups meetings
↳ ETA of final report.
(Set a status expectation)

(4) Additional comm^n Plans:

- Begin scheduling final briefing. Ask when stakeholders can be present for meet.
- What are acceptable comm^n for sensitive / confid data as such. (email, phone, fileshare etc).

Day 5: Part 15: Post Assessment

- Assessment Finished → Results communicated
- RESULTS → Reports → any format docs
  - XL sheets → Portals (Archer)

(1) Reports: I know. (well written, informative, Reproduceable)

- Exec Summary
- Test Params
- Findings.

} can use for more appendix to incl. addl. info.

① EXEC Summary : High level summary for execs to understand.

② Test Params : Project obj, Scope, Schedule, Targets, Test name, URL, Limitations, Finding summary, Remediation summary.

③ Findings. : Tech details. What we do. I know.

④ Report Appendix:

- ↳ Finding affected many params / URL.
- ↳ enumerated uname / passwords
- ↳ Document Authorization Letters
- ↳ CMD output thats V Long.
- ↳ Data exfil during exploit.

⑤ Report Automation : (Templates integrated into WORD)

- Fast
- Efficient
- consistent
- ↑ Quality
- ↓ time reporting
- ↑ time hacking.

(6) DeBrief Presentation:

- Debrief PPT = Pentesters can talk to app team and stakeholders (interactive discussion)
- As multiple groups = multiple intellects (multiple interest) = multiple presentations by pentes