A Bug Hunter's Journey from CTFs to Cash: Shifting to the Bug Bounty Hunter's Methodology

Ads Dawson (@0xmoose)

BugCrowd x UT Austin University Cybersecurity

18:00 - 20:00 pm ET, October 22, 2025, 2 hours - 1 hour talk, 45 minutes + 15 minutes Q&A & 1 hour VDP recon practical workshop

whoami

Ads Dawson (@GangGreenTemperTatum / @0xmoose)

Harnessing code to conjure creative chaos.. think evil; do good.

Staff AI Security Researcher @ Dreadnode

Bug bounty hunter & bench penetration tester

BugCrowd Hacker Advisory Board member

Caido hacker ambassador

BugCrowd ITMOAH 2024

OWASP chapter and project lead

My journey

School

Dropout

I started my journey in tech with a hands-on apprenticeship, choosing practical experience over a traditional academic path.

Network intern

Student and sleepless nights

My early career was dedicated to mastering network engineering at an advanced level with Cisco, where I built the foundational skills for a future in cybersecurity.

Web apps and OWASP

Hunger, curiosity and addiction

I transitioned into application security, leading chapters at OWASP, contributing to global standards like ASVS, and becoming an addicted bug bounty hunter.

AI + AppSec

Breaking and using cutting edge tech

Leading AI security research at Dreadnode, I now use bleeding-edge techniques to break next-gen AI and architect autonomous offensive security systems.

Hacker DNA

- Someone who breaks things just to see how they work
- Someone who fixes things only to break them again in a different way.
- Someone who refuses to accept *"that's just how it's done"* as an answer
- The ones who refuse to think in straight lines
- They see patterns others miss, anticipate failures before they happen
- Hack the planet

https://medium.com/@gonzo-hacks/the-hacker-mindset-humanitys-best-defense-a25d3e8b580e

Hacker mindset: Bounty, or VDP?

Bounties

Pro: You get paid cash rewards for valid findings, directly monetizing your hacking skills.

Con: The high competition often leads to duplicate findings, making it difficult to get your first payout.

VDPs

Pro: Less competitive environment to find your first real-world vulnerabilities and build a public track record through acknowledgments.

Con: You won't receive any monetary payment, as the reward, limited to experience and public recognition.

Red pill or blue pill

Welcome to the real world homie. Alright, so you've been grinding away at CTFs, you know how to pop a shell, and you're comfortable with the tools.
So, which one do you take?

Shifting from CTF to bounty: Threat Modelling

- Put yourself in the shoes of the company who owns this application, or any customer/stakeholder and ask yourself:
    - What brings risk to stakeholder of this application?
    - What is and isn't "allowed" from a users perspective?
    - Are there any warnings from the application that state the above
- Understand the attack surface, focus on business logic and think about chaining vulnerabilities with this in mind
- Become a power-user of the application, record your steps and observe those network requests being made
- Don't jump right into breaking it, understand these fundamentals first

https://projectdiscovery.io/blog/bug-bounty-etiquette-2-what-not-to-do

First: RTFM

Program and Developer Docs

1. What's in-scope and out of scope explicitly?
2. Read the developer docs to understand the inner and outer workings
3. Recon the tech stack and look for "focus areas" in the program
4. Are there any known issues?

BugBoss v3: https://www.youtube.com/watch?v=zIodGMqAuD8

The scope document is your contract. Ignoring it is the fastest way to get your reports closed as Not Applicable and waste weeks of your time

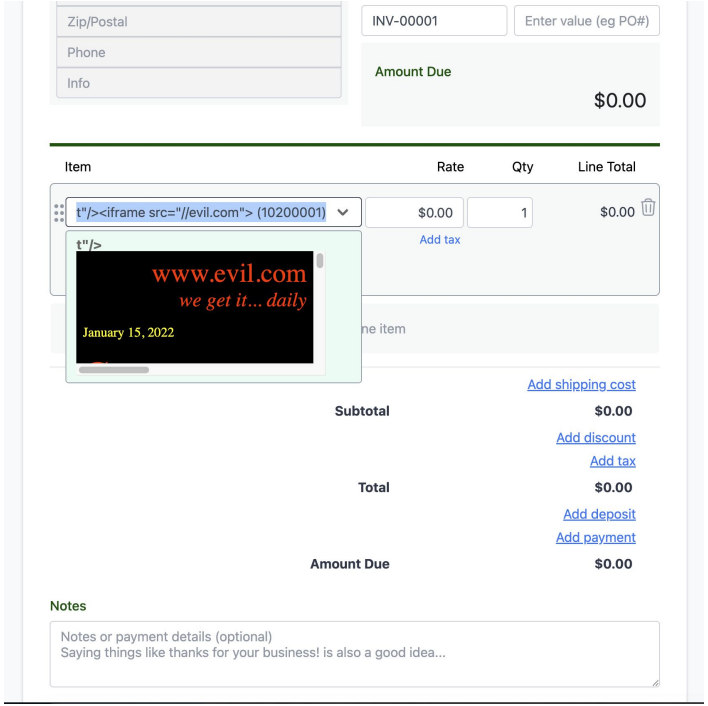
Developer docs are sometimes the secret weapon that most hunters ignore. The developer and API documentation is a treasure map written by the people who built the fort

Pick your battles

- "*Spray and pray*" vs. sniping: In CTFs, you're rewarded spraying at every possible vulnerability to grab flags
- In bug bounties, you get paid to be a sniper, patiently finding that one, unique, high-impact shot that no one else saw
- Low noise, high signal is key - the vulnerabilities that have a real business impact and make a company go, "Oh, that's bad."
- Don’t be a swole doge
- The "One-Trick" Cheems is a lie
- It's about depth, not just breadth of testing
- INVEST IN YOUR OWN PAYLOADS / *"I see this is being reflected in the DOM, could I use that payload from the other program?"*

Save dem' payloads.

```
`t"/><iframe src="//evil.com"> (10200001)`
```



Sell your submissions

- The anatomy of a good report first demonstrates immediate impact without over inflating
- Title is the first thing a triager sees. Make it count. It should be a concise summary of the vulnerability and its impact.
    - Bad Title: "XSS Bug Found"
    - Good Title: "Stored XSS in User Profile Page Leading to Admin Session Hijacking"
- Provide clear, numbered steps
- Use screenshots, GIFs, or even a short video for every key action.
- Include the full request and response payloads.
- Avoid AI slop
- PoC needs to be foolproof
    - Bad: "*An attacker can steal cookies.*"
    - Good: "*By stealing an administrator's session cookie, an attacker can gain full administrative access, allowing them to view, modify, or delete any user's data on the platform.*"

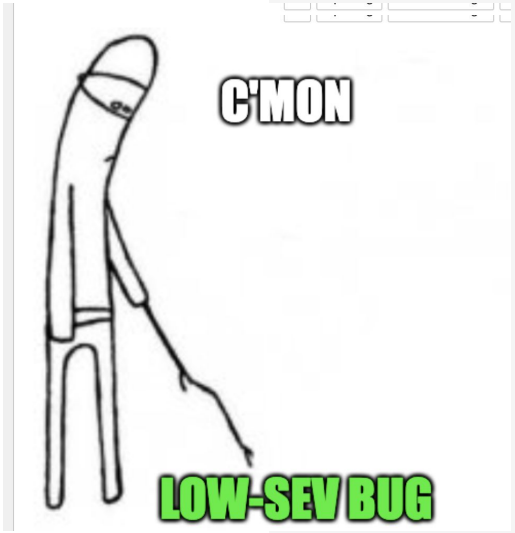
https://medium.com/@pm_/guide-crafting-a-neat-and-valuable-bug-bounty-report-0bf1bc933bdc

https://docs.hackerone.com/en/articles/8475116-quality-reports

Ads sample report: https://gist.github.com/GangGreenTemperTatum/4699f7503fe3eb2dcf7e35dd0dd63a22

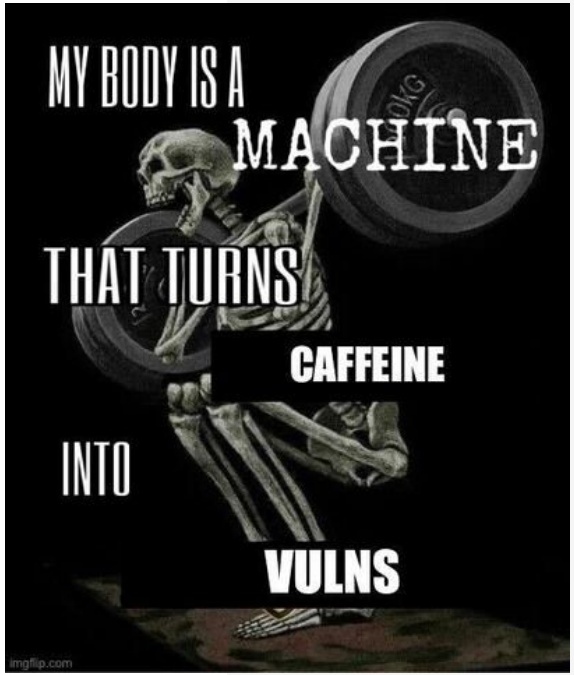
Case study: @ads P3->P1 ATO CHAIN YOUR BUGS

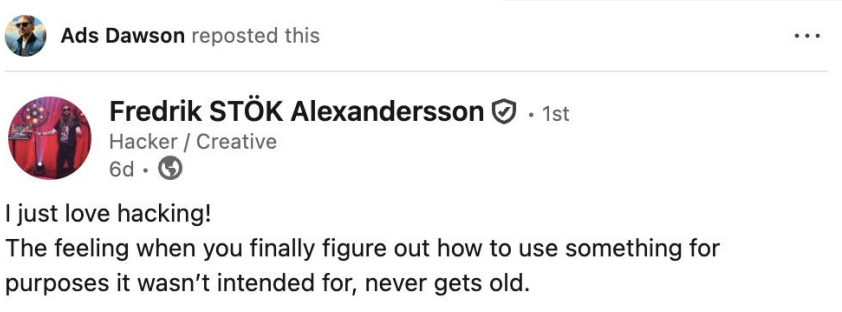
- Password reset current password bypass - so what?
- IDOR - can we perform that action on another user account?
- Need cookies - where can we inject data into the app to steal another users cookies?
- Chain these bugs == full account takeover of all users



LOVE what you do.

- Surround yourself with others in the same mindset
- Shiny rocks are awesome
- Share wins with friends, celebrate and root for your homies
- It’s OK if you aren’t this way and isn’t for everyone





Hacker psyche

- Welcome the challenge, real world programs aren't preloaded with flags
- Some devs are VERY good but don't test the simple stuff
- Embrace failure, don't be shy or scared
- Failure is an opportunity to learn
- Reach out to friends, don't suffer alone
- Speak up, say something
- Get that first bug under your belt
- If you don't understand a vulnerability class, research it and become a frickin expert at it
- There's no secret recipe to time spent on program vs vuln, find what works for you

Imposter syndrome is real

- Imposter syndrome thrives in environments where skill and achievement are central
- The more you learn about a complex field, the more you become aware of how much you don't know
- The highlight reel effect
- Track Your achievements and keep a running list of your successes, big or small
- Reframe Your Thoughts: Instead of thinking, "I got lucky," try thinking, "*my hard work paid off*"
- Speak up, share your feelings with a trusted friend, mentor, or colleague
- Embrace "Good Enough" and don’t strive for perfection
- Aim for progress and completion rather than an impossible

Never give up or leave any shiny rock unturned

- Why did my submission get declined/de-escalated?
- How can I further demonstrate impact?
- Does my report/submission read well, can I understand it?
- Can I chain this bug with something more meaningful?
- Store observations and bugs into RAM or take notes for later use

Community is king

When you hit a wall, someone in the community has a ladder

Bugs and payouts are temporary, but a strong network is the most valuable asset you'll ever build in this industry

1. When you hit a wall, someone in the community has a ladder
2. This industry is full of awesome people
   Those awesome people want to see you succeed
3. Follow researchers, join discussions, and absorb information
4. Found something? Share the win with others and educate

Collaboration: Sometimes the recipe for success

- Pair programming, but for pwnage
- You learn their perspective: You'll see how they approach a target, the unique tricks they use, and the logic they apply. It’s like getting hands-on training for free
- You share your own skills: Teaching someone else your techniques is one of the best ways to solidify your own knowledge
- You get fresh eyes on the target: When you've been staring at an endpoint for six hours, you go blind to the obvious. A hacking buddy can look at it for five minutes and spot the simple mistake you were making
- Sharing wins with friends is awesome

Stay hungry - never lose the student mindset

BugCrowd University

https://www.bugcrowd.com/resources/levelup/introduction-to-bugcrowd-university/

Public Submissions

Public writeups and submissions are a great way to understand how to shift your creative ways

Critical Thinking

https://www.criticalthinkingpodcast.io

PortSwigger Labs

https://portswigger.net/web-security

Researcher' blogs

https://jameskettle.com/, https://blog.orange.tw/ and more

YouTube

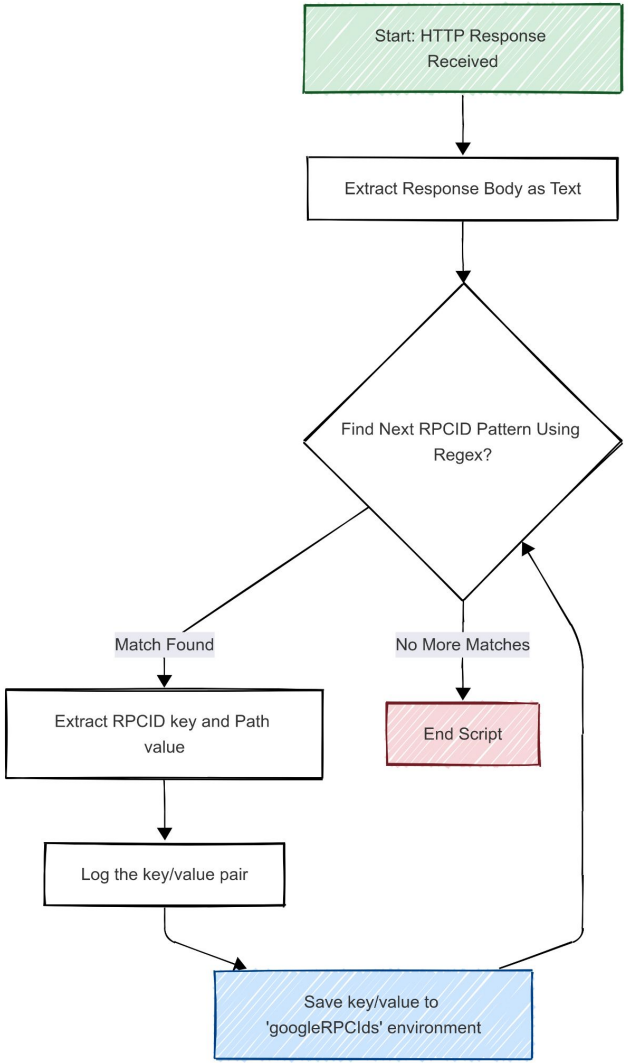
@JHaddix (Bug Bounty Hunter's Methodology), @NahamSec, @Medusa0xf, @AmrSecOfficial and more

Case Study

Hacker Mindset - Do your homework

Google Bard RPC ID’s - WTF? = Caido workflow

- Credit: @Rhynorater
- **Problem**: Google Bard’s API uses RPC cipher IDs for services
- **Why**: It’s painful for us as hackers to understand what RPC ID is what service - we don’t know what endpoints or services we are trying to exploit
- **Hacker mindset & solution**: The process begins when an HTTP response is received, and the script immediately starts looping through the response's text, using a Regular Expression to hunt for a specific code pattern.
- Each time a match is found, the script extracts the key (the rpcid) and the value (the descriptive path) and saves the pair into a shared environment, continuing this loop until no more matches exist in the file.



Final Thoughts

Hacker Mindset (watch video up until ~4:30mins)

My First Bug Bounty Experience (It Was a Mess!) - Credit: @CyberFlow

https://www.youtube.com/watch?v=lV9skVPlbnA

Practical hands-on workshop

- **Bio-break (five minutes)**
- Let’s put words into keystrokes
- Fire up your laptop
- Join the hunt (if you want to)
- LFG

Recon

Bio-break (five minutes)

Cengage VDP

In Scope Targets

"Any Cengage asset is considered as in scope for this program"

Wildcards are shiny rocks

https://bugcrowd.com/engagements/cengage-vdp

```
https://*.cengage.com,https://*.cengagebrain.com,https://*.cengagew
ork.com,https://*.ed2go.com,https://*.milady.com,https://*.gale.com
,https://*.infosec-skills.com,https://*.infosec-iq.com,https://*.in
fosecinstitute.com,http://starship-emulator.cengage.com/,https://a2
s.cengage.com/,https://websuite.visiblebody.com,https://www.visible
body.com,https://www.pmguidedreading.com,https://*.cengage.com.au,h
ttps://*.cengage.co.nz,https://gotit.net.au/dosage-calculations,htt
ps://*.nelsonnet.com.au,https://*.pmecollection.com.au,https://*.iq
central.cengage.com.au,https://pmolcards.com,https://nldcomprehensi
on.com,https://*.nelsonhub.cengagelearning.com.au,https://*.cengage
learning.com.au,https://*.cengageasia.com,https://www.asianglshop.c
om,https://www.cengageasiaestore.com,https://*.cengageresource.com
```

Recon - Tooling and teaching

"Information on a target system or network is gathered during reconnaissance in order to find any potential weaknesses that might be exploited."

Tools

- **Anything** from project discovery - PDTools
- Including wapplyzergo
- tomnomnom's tools - waybackurls, assetfinder, httprobe
- Wrapper tools - BBOT

Teachers

@JHaddix (Bug Bounty Hunter's Methodology), @NahamSec, @Medusa0xf, @AmrSecOfficial and more

Chain them together

- With recon, you get out what you put in
- Use multiple tools, chain them together and create structured i/o's

Recon steps (high-level)

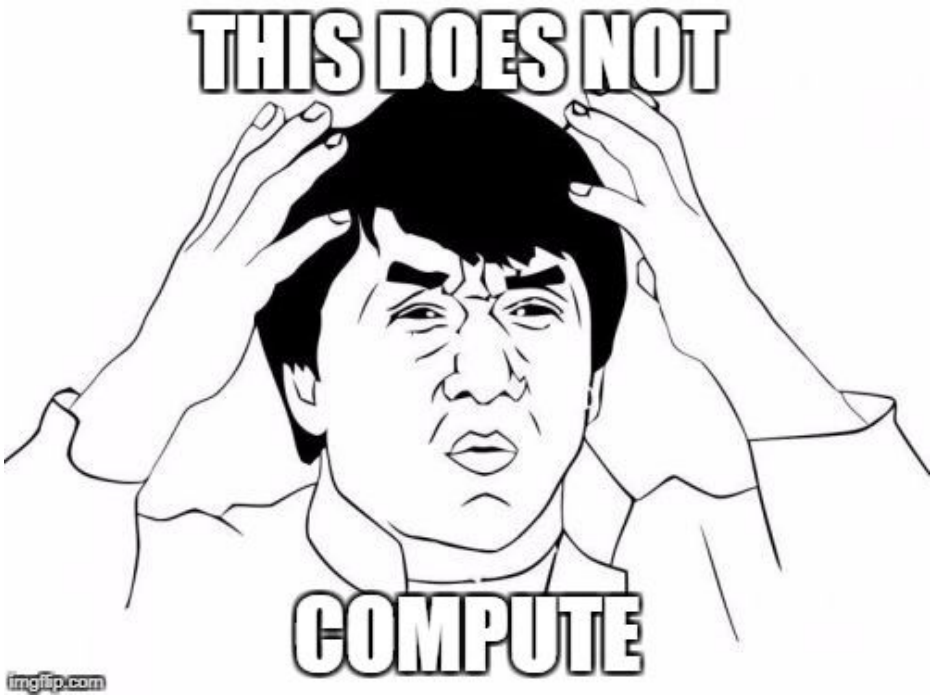
You get in what you get out. Recon is about finding leads.

Not including both scope & rules check or passive OSINT / discovery (IE public records, certs, archives).

1. **Passive discovery first** (no requests to target): crt.sh, certificate logs, public archives, GitHub, Subfinder/amass passive mode, Shodan/Censys. Builds a long list of assets without touching the target.
2. **Active enumeration (careful & scoped)**: DNS bruteforce, subdomain takeover checks, directory fuzzing (ffuf/gobuster), port scanning (nmap). Respect rate limits in program policy.
3. **Content & parameter discovery**: waybackurls, gau, LinkFinder, param spider, JS parsing to find hidden endpoints and parameters.
4. **Triage & fingerprinting**: whatweb/wappalyzer, aquatone screenshots, gf pattern matching.
5. **Vulnerability scanning (non-exploitive)**: Burp or ZAP passive/active scanners only if allowed by program rules; do NOT run aggressive scans against targets that forbid them.
6. **OSINT/secrets**: GitHub/GitLab search + truffleHog/Gitleaks for leaked keys (only public repos; use responsibly).

Hosting

So we’re gonna need a VPS



GangGreenTemperTatum/recon-install.sh

Active discovery & enumeration

Locating **assets**

```
subfinder config
$ subfinder -d

$ shuffledns
$ wget resolvers
assetnote/seclists wordlists

Permutations (environments)
$ alterx

Resolve them with dnsx
Port scan with naabu
```

https://github.com/trickest/resolvers

Goal: find live hosts, subdomains, ports and services (current surface).

Key actions: subdomain enumeration, DNS resolution, port/service scans (carefully).

Tools/commands: amass/subfinder, massdns/httpx, nmap (or masscan + nmap).

Watch for: wildcard DNS, rate limits, blocked IPs/WAF fingerprints.

Output: resolved host list (domain → IP), open ports, exposed services.

What next?

Not including both scope & rules check or passive OSINT / discovery (IE public records, certs, archives).

Content & resource mapping

Content discovery

Goal: map web endpoints, files, APIs, JS and parameters to test later.

Key actions: crawl sites, fetch JS, brute-force directories, extract endpoints/params.

Tools/commands: Burp/ZAP crawler, ffuf/gobuster, LinkFinder/grep for JS.

Look for: admin pages, hidden backups (.bak, .old), .git, sitemap/robots, API routes.

Output: endpoint inventory (URL, method, params), JS/API map.

```
Katana

Httpx

Find routes and potential javascript
injection points (content-type: html)
```

Realistic workflow

urlfinder

```
$ chaos-client -d cengage.com -silent | grep api | alterx
-silent | dnsx -silent | naabu -p 443,8443 -silent
```

```
$ urlfinder -d cengage.com
```


GangGreenTemperTatum/recon-install.sh

Static JS & API analysis

wget — download linked JS files
```
wget -r -l1 -H -nd -A '*.js' https://example.com
```
gau / waybackurls — collect historical & discovered URLs
```
echo example.com | gau > urls.txt
```
LinkFinder — extract URLs/endpoints from JS bundles
```
python3 LinkFinder.py -i bundle.js -o cli
```
ripgrep (rg) — fast code-pattern search in bundles
```
rg -n --hidden "https?://|/api/|Authorization|token" *.js
```
js-beautify — prettify/minified JS for inspection
```
js-beautify bundle.min.js -o bundle.beautified.js
```
prettier (npx) — reformat JS for easier reading
```
npx prettier --write bundle.min.js
```
ffuf — fuzz discovered API paths/parameters
```
ffuf -u https://api.example.com/FUZZ -w
/usr/share/wordlists/common.txt
```
dirsearch — enumerate API endpoints / directories
```
python3 dirsearch.py -u https://api.example.com/ -e json,js
```
gitleaks — scan repos/artifacts for leaked secrets
```
gitleaks detect --source=. --redact
```
truffleHog — entropy-based secret hunting in repos
```
trufflehog --entropy=True https://github.com/org/repo.git
```
jq — pretty-print and filter JSON API responses
```
curl -s 'https://api.example.com/v1/info' | jq .
```
Caido(proxy + Replay) — capture XHR, replay and modify requests
```
Intercept XHR, send interesting requests to Replay for testing.
```

Goal: extract endpoints, tokens, auth logic and interesting parameters from client-side code.

Key actions: download JS bundles, search for URLs, token names, feature flags, credentials.

Quick checks: XHR/Fetch patterns, Authorization, api/, /v1/, baseURL.

Tools/commands: LinkFinder, grep -E "https?://|api|token|auth" *.js.

Output: prioritized API endpoints, potential secret/key leaks, auth flow hints.

Fingerprinting & context

curl — quick response headers / cookie flags
```
curl -s -D - https://example.com -o /dev/null | rg -i "set-cookie|content-security-policy|access-control-allow-origin|x-powered-by"
```
httpie — readable header + body fetch
```
http --headers https://example.com
```
CORS test (curl with Origin) — check permissive CORS responses
```
curl -s -H "Origin: https://evil.com" -I https://api.example.com
```
securityheaders.com (web) — quick service to grade CSP/headers (manual)
```
Open https://securityheaders.com and enter example.com
```
WhatWeb — detect server, frameworks, plugins
```
whatweb https://example.com
```
Wappalyzer CLI — client/stack fingerprinting (npm)
```
npx wappalyzer https://example.com
```
Nmap (http-enum / script) — enumerate web server info & modules
```
nmap -sV --script http-enum -p 80,443 example.com
```
testssl.sh — TLS/SSL configuration & cipher checks
```
./testssl.sh --quiet example.com:443
```
WAFW00f — detect presence of web application firewall
```
wafw00f https://example.com
```
Nikto — quick server misconfig checks (headers, files)
```
nikto -h https://example.com
```
gospider / gau — collect subpages & endpoints for fingerprinting surface area
```
gau example.com > urls.txt
```

Goal: identify tech stack, libraries, headers and security controls that suggest vulnerabilities.

What to collect: response headers, cookies flags, CSP/CORS, server/X-Powered-By, library versions.

Tests: curl -I, test CORS with Origin header, check cookie Secure/HttpOnly.

Red flags: open CORS, insecure cookies, outdated libs, permissive CSP, missing rate limits.

Output: vulnerability signals (e.g., possible CORS/Cookie/Javascript-based attacks).

Recon attacks → common exploitable finds

- **Discover hidden admin/API endpoints** — *ffuf*
  ```
  ffuf -u https://example.com/FUZZ -w /usr/share/wordlists/dirb/common.txt -mc 200,401,403
  ```
- **Find historical/exposed artifacts (backups, git blobs, env files)** — *gau / waybackurls*
  ```
  echo example.com | gau | rg -i "\.(zip|tar|env|sql|git|bak)$" > artifacts.txt
  ```
- **Scan repos/artifacts for leaked secrets** — *gitleaks*
  ```
  gitleaks detect --source=. --report=gitleaks-report.json
  ```
- **Extract JS-exposed tokens / auth logic** — *LinkFinder*
  ```
  python3 LinkFinder.py -i bundle.js -o cli
  ```
- **Automated CORS & other web checks** — *nuclei (templates)*
  ```
  nuclei -u https://example.com -t nuclei-templates/cors/
  ```
- **Enumerate subdomains / discovery surface** — *subfinder / amass*
  ```
  subfinder -d example.com -o subdomains.txt
  ```
- **Detect dangling subdomains & takeover candidates** — *subjack / takeover*
  ```
  subjack -w subdomains.txt -t 50 -timeout 30 -o subjack-results.txt -ssl
  ```
- **Prioritize findings / run targeted exploit templates** — *nuclei (exploit templates)*
  ```
  nuclei -l urls.txt -t nuclei-templates/takeovers/ -o nuclei-takeovers.txt
  ```

Hidden admin/API endpoints → auth bypass / IDOR.

Exposed backups / git / env files → secrets & creds.

JS-exposed tokens/logic → token theft, bypass, SSRF hints.

Misconfigured CORS → cross-origin token exfiltration.

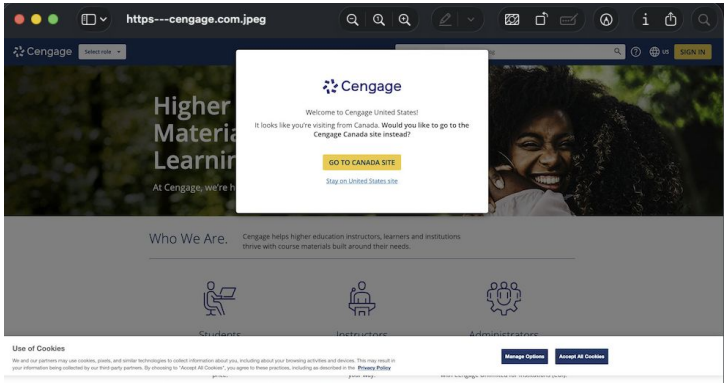
Dangling subdomains → subdomain takeover.

Backup/open cloud buckets → data leakage.

Note: each finding gives a targeted exploitation path.

Screenshots - Gowitness / Eyeballer

Use your favorite screenshotting tool like normal (EyeWitness or GoWitness) and then run them through Eyeballer to tell you what's likely to contain vulnerabilities, and what isn't.




```
Basic single URL:
gowitness scan single -u https://example.com
With multiple URLs from a file:
gowitness scan file -f urls.txt

Useful options:

# Full page screenshot (not just viewport)
gowitness scan single -u https://example.com --screenshot-fullpage
# Save as PNG instead of JPEG
gowitness scan single -u https://example.com --screenshot-format png
# Custom screenshot directory
gowitness scan single -u https://example.com -s /path/to/screenshots
# Save metadata to database for later reporting
gowitness scan single -u https://example.com --write-db
# Multiple URLs with database + full page
gowitness scan file -f urls.txt --write-db --screenshot-fullpage
By default:
- Screenshots are saved to ./screenshots/ directory
- Format is JPEG
- Only the viewport is captured (not full page)
- It waits 3 seconds after page load before capturing

You can then view the results with:
gowitness report serve
```

Cengage VDP

Recon - BBOT - Subdomains

https://bugcrowd.com/engagements/cengage-vdp

```
docker run --rm -it -v "$HOME/bbot-results:/output"
blacklanternsecurity/bbot:stable \
  -t
cengage.com,cengagebrain.com,cengagework.com,ed2go.com,milady.com,gale.com
,infosec-skills.com,infosec-iq.com,infosecinstitute.com,starship-
emulator.cengage.com,a2s.cengage.com,websuite.visiblebody.com,visiblebody.
com,pmguidedreading.com,cengage.com.au,cengage.co.nz,gotit.net.au,nelsonne
t.com.au,pmecollection.com.au,iqcentral.cengage.com.au,pmolcards.com,nldco
mprehension.com,nelsonhub.cengagelearning.com.au,cengagelearning.com.au,ce
ngageasia.com,asianglshop.com,cengageasiaestore.com,cengageresource.com \
  -p subdomain-enum \
  --output-dir /output/ \
  --force \
  --allow-deadly
```

CENGAGE

Cengage VDP

Recon - BBOT (cheat sheet) - Kitchen Sink

https://bugcrowd.com/engagements/cengage-vdp

```
docker run --rm -it -v "$HOME/bbot-results:/output"
blacklanternsecurity/bbot:stable -t /output/cengage-
bbot-subdomain-enum/subdomains.txt \
    -p kitchen-sink \
    --output-dir /output/cengage-bbot-kitchen-sink \
    --force \
    --allow-deadly \
    --name cengage-bbot-kitchen-sink \
```

Thank you! Stay in touch

Q&A?

- **LinkTree**: https://linktr.ee/adsdawson
- **LinkedIn**: https://www.linkedin.com/in/adamdawson0/
- **GitHub**: https://github.com/GangGreenTemperTatum
- **BugCrowd Slack (BCBuzz)**: @ads
- **BugCrowd Author Site**: bugcrowd.com/blog/author/ads-dawson/
- **Slides available**: https://ganggreentempertatum.github.io/