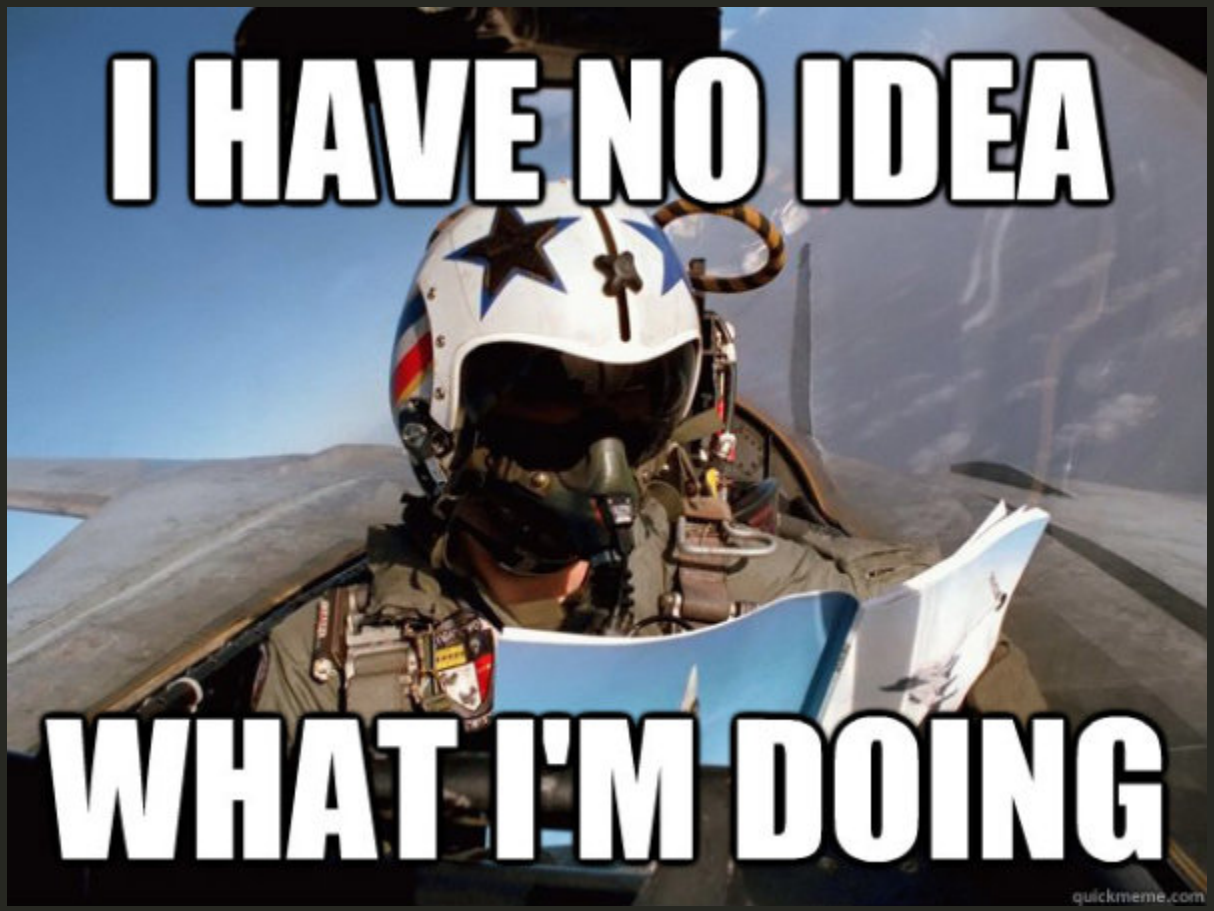
Browser Fuzzzing Tradecraft

Disclaimer

This research and opinions in this presentation are my own and are in no way affiliated with my employer or any other entities referenced in this presentation.

whoami



Agenda

1. What to expect
2. Intro to Fuzzing
3. Logistics of Fuzzing
4. Pitfalls when Fuzzing Browsers
5. Hands-on Time

What to Expect

Goals:

- Just get up and running

Not goals:

- Leaving this session with 0day
- Writing your own fuzzer
- Anything regarding exploit dev

Lessons:

- Actual work will be about trade-offs (more sand vs better spec)
- If you are seriously committed to fuzzing, regardless of scale:
  - *Always* be fuzzing
  - *Always* be improving your fuzzer
- In practice picking a fruitful target requires honed intuition

Intro to Fuzzing

What is fuzzing?

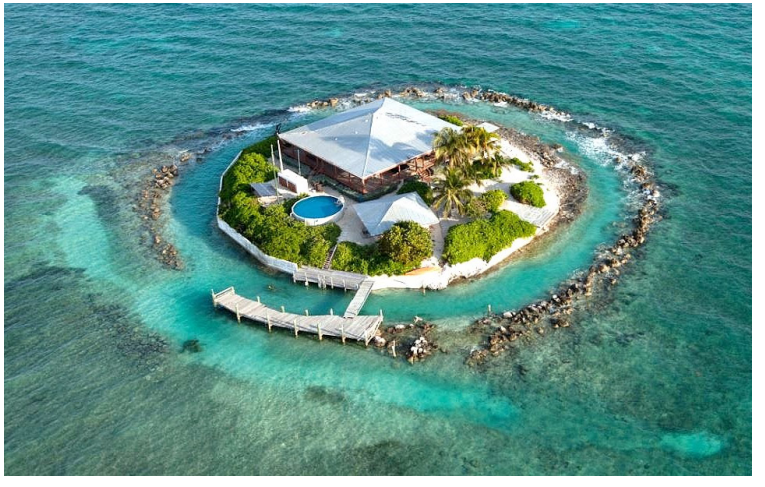
- The technique dates back to the 1950s, the modern term comes from Barton Miller at University of Wisconsin Madison
- Fuzzing is the act of programmatically producing input and passing it to a target
- Originally used as a means to QA software, current form is predominately about illiciting security bugs

Mutation vs. Evolutionary vs. Grammar fuzzing

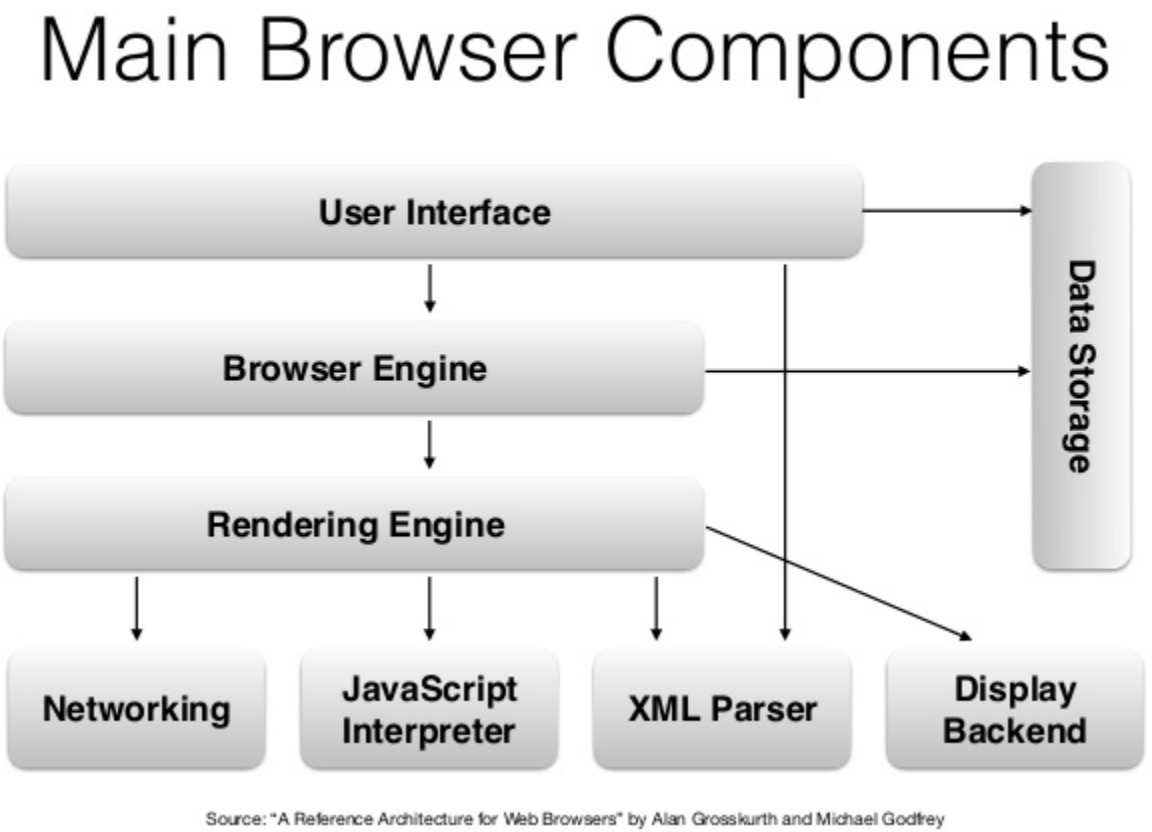
- Mutation (radamsa)
  - Have your target binary and a corpus of valid inputs
  - Pick a file, pick a location within file, pick a mutation and feed to target
  - Mutations: flip bits, flip bytes, char insertion, etc...
- Evolutionary (AFL)
  - Instrument the target
  - Watch the paths taken over mutated input
  - Note what changes result in new execution paths through binary
- Grammar
  - Turns out mutation/evolutionary *really* isn't a viable approach for browsers
  - Producing valid JS/HTML/etc through mutation = No
  - Instead of flipping bytes on valid input, define the valid grammar s.t. *all* outputs are syntactically valid

Why fuzz?

- For the fun of bug finding
- For profit:
  - Find bugs
  - Develop exploits
  - Sell them for top dollar
  - Buy private island



Why fuzz browsers?



This is super dated btw.

How is fuzzing browser different?

- Attack surface
  - Rendering engine
  - Javascript engine
  - Arbiter (child process management)
  - 3rd party extensions
  - 3rd party dependencies
- A different kind of beast
  - Operation Aurora really underscored that clients are soft targets
  - Watering hole gets loads of targets and bypass the firewall
  - Super valuable for LEO
- More recent hardening efforts
  - Edge/IE now runs as a super low integrity processes
  - Chrome has a solid process sandbox
  - Integration of CFG and EMET makes exploitation harder

Logistics

Sand arrangements

- Don't fuzz on your main gear
  - Prototyping is fine, but real runs will be I/O intensive
- Cloud vs. Bare-metal
  - Mostly matter of personal preference and budget
  - If you fuzz intelligently, then cloud based is probably only marginally more expensive than bare-metal and frees you from grunt work
  - I veer toward paranoia and recommend bare-metal
- Saving your gear/wallet
  - Most fuzzing does lots of disk I/O
  - As your drive gets used more and more it becomes more prone to failure (see Backblaze reports)
  - Save your drives by loading your box up with RAM
  - Allocate 75% of your RAM as a ramdisk (filesystem backed by ram)
  - This is where your mutated samples will live
  - On crash: fuzzer moves appropriate sample to crashers dir on non-volatile storage

Orchestration

- Scaling
  - One manager: manages nodes, crashes, resource allocation, crash binning/prioritization
  - Many nodes: individual node fuzzes a single browser/version
- Examples
  - Grinder
  - ClusterFuzz

You're up and running...

...spent 1000s of hours developing, refining, and operating your badass custom fuzzer...

...eventually find crashing inputs...

(private island, here we come!)

...as the browser crashes, a crash report is auto-generated and phoned home to MSFT/GOOG/MOZ with your bug

...F*CK!

(goodbye, island :'( )


STOP
SNITCHING

Other Gotchas

1. Configure your VM adequately
    - I use 2-4 CPUs/Cores + >= 4 GB of RAM
    - Don't let the guest VM run away (and potentially tank the host)
    - But under-resourced will lead to system thrash and inconsistent bugs
2. Control the *whole* process
    - Set the browser to start on a blank page or set 'about:blank' as the home page
    - Start the process with a clean heap and no unnecessary modules
3. Serve your crashers over HTTP
    - File -> Open may screw with things
    - Likely only valuable find if it crash occurs when server over net
4. Make sure you see everything
    - Edge: Enable Heap Pages (`gflags /i $PROCESS_IMAGE +hpa`) to detect trashed heap accesses sooner
    - Chrome/Firefox/Safari: Build with ASAN

Other Gotchas

1. Edge-only: Disable tab recovery
    - regedit.exe
    - Direct to: HKEY_CURRENT_USER\Software\policies\Microsoft
    - Make new key titled 'MicrosoftEdge'
    - Create a subkey named 'Recovery'
    - Create a new DWORD in 'Recovery' titled 'AutoRecover' with value 0x2
    - Direct to: HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft
    - Make new key titled 'MicrosoftEdge'
    - Create a subkey named 'Recovery'
    - Create a new DWORD in 'Recovery' titled 'AutoRecover' with value 0x2

Reference

Going Hands-on

Going Hands-on: Fuzzing Edge with Wadi

1. Get your VM: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
2. Install Python2.7 on the VM
3. Download WinDBG (+tools): https://developer.microsoft.com/en-us/windows/hardware/download-windbg
4. Grab Wadi: https://github.com/sensepost/wadi
5. Install pre-reqs (per README.md)
6. Start fuzzing: `python wadi.py IE.js 8000`

Going Hands-on: Fuzzing Firefox with domato (Advanced)

1. Download Firefox ASAN build: https://developer.mozilla.org/en-US/docs/Mozilla/Testing/Firefox_and_Address_Sanitizer
2. ASAN Flags (for reference): https://github.com/google/sanitizers/wiki/SanitizerCommonFlags
3. Download domato: https://github.com/google/domato
4. Exercise: Write a python script that:
    - Calls domato to generate N sample files
    - Starts the Python SimpleHTTPServer hosting the samples
    - For each sample file, trace Firefox requesting the sample:
        - On a crash, save the sample file & the state of the Firefox process when it crashed
        - After some timeout (presumed to not crash), kill the process and move onto the next sample

python-ptrace: https://github.com/haypo/python-ptrace will help with crash detection

Thanks! :)