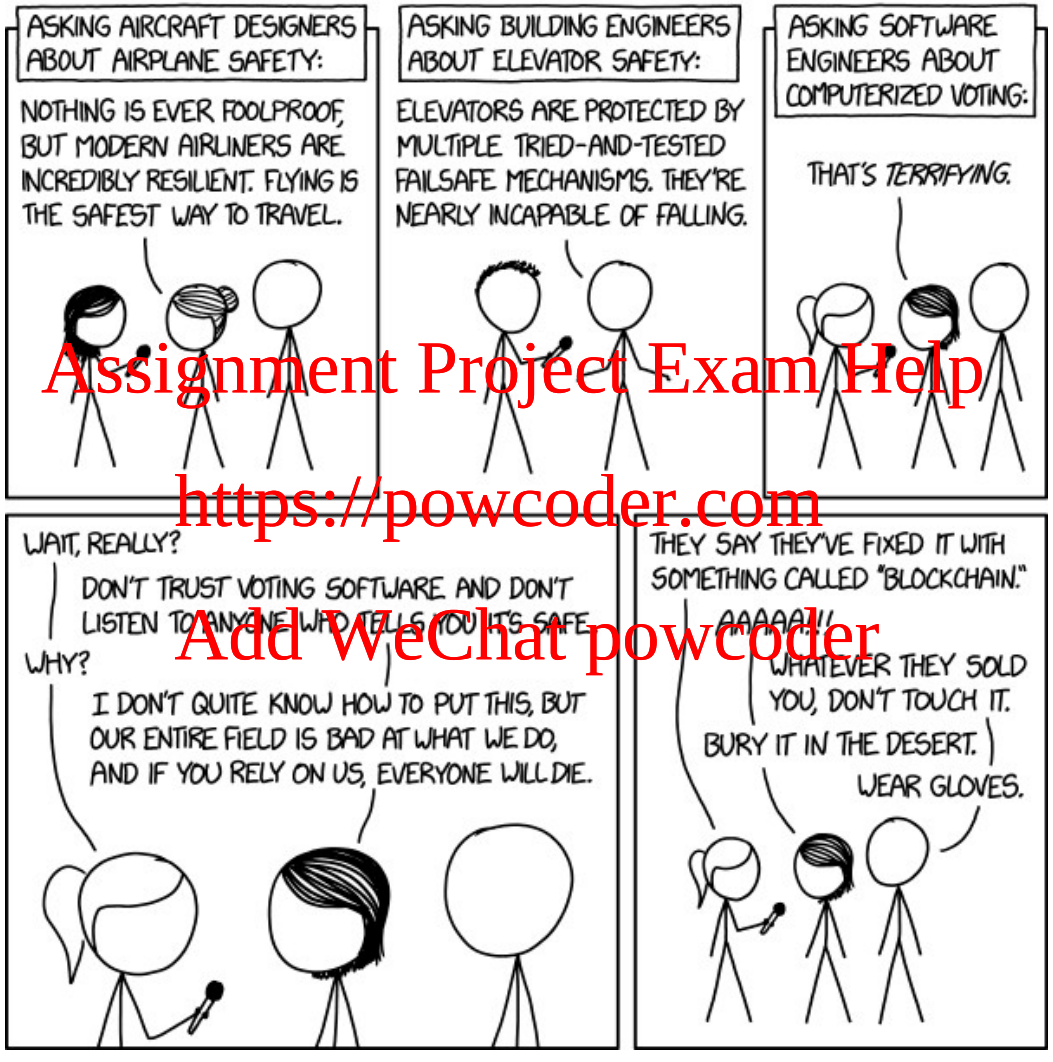

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Fuzzing

History

- Miller et al. "An Empirical Study of the Reliability of UNIX Utilities", CACM 33(12), 1990

Utility	VAX (v)	Sun (s)	HP (h)	i386 (x)	AIX 1.1 (a)	Sequent (d)
adb	•○	•	•	○	–	–
as	•			•	•	•
awk						
vi	•	•	–			
wc						
yacc						
# tested	85	83	75	55	49	73
# crashed/hung	25	21	25	16	12	19
%	29.4%	25.3%	33.3%	29.1%	24.5%	26.0%

Why do we care?

- Pwnie award (2009)

Lamest Vendor Response

Awarded to the vendor who mishandled a security vulnerability most spectacularly.

- Linux
  Continually assuming that all kernel memory corruption bugs are only Denial-of-Service

  The Linux kernel development team was nominated several times over for their ongoing lack of handling of bugs of "unknown impact" and generally assuming that all kernel memory corruption issues are only Denial-of-Service issues. Here's a hint: Just because you can only get a DoS from a bug, doesn't mean that skilled hackers can't get a root shell out of it.

Fuzzing

- Generate random input
  - Command line args, files, network traffic, function arguments.
- Run program
- Check for errors

- Low probability of hitting a bug
  - Can we do better?

Mutational Fuzzing

- A.K.A. dumb fuzzing or black-box fuzzing
- Randomly change ***seed*** input
- Example
  - Standard HTTP GET request
    - `GET /index.html HTTP/1.1`
  - Mutated requests
    - `AAAAAA...AAAA/index.html HTTP/1.1`
    - `GET ///////index.html HTTP/1.1`
    - `GET %n%n%n%n%n%n.html HTTP/1.1`
    - `GET /AAAAAAAAAAAAA.html HTTP/1.1`
    - `GET /index.html HTTTTTTTTTTTTTP/1.1`
    - `GET /index.html HTTP/1.1.1.1.1.1.1.1`

Mutational Fuzzing

- Little or no knowledge of target input format required
  - Do need some valid input as seeds
- Easy to set-up
- Mutation heuristics:
  - Purely random
  - Flip a bit (or a group of bits)
  - Change byte values
  - Known integers
  - Input splicing
  - Insert or delete bytes

Fuzzing a PDF viewer

Credit: Tal Garfinkel – Stanford/VMware

- Google for .pdf (about 4 billion results)
- Crawl pages to build a corpus
- Use fuzzing tool (or script to)
  - Grab a file
  - Mutate that file
  - Feed it to the program
  - Record if it crashed (and input that crashed it)

Mutational Fuzzing – cons

- Quality depends on choice of seed files
  - One vacation photo is good. 200 is a waste of time.
- Problem with input sanity
  - CRC
  - Compressed files
  - Language syntax

Generational fuzzing

- A.K.A. grammar-based fuzzing, white-box fuzzing, smart fuzzing
- Creates random input files based on known structure
- Complex to create, use, etc.
- Require knowledge of the protocol

Targeting known bugs

- Insert long strings
- Large/small/boundary numbers
- Unicode
- Format strings

How much is enough?

- When to stop? Fuzzing can continue generating random inputs indefinitely
- Example:
  - 300KB input image
  - One specific byte change causes a crash
  - Success probability: 0.000000133%
  - At 2 seconds per test we expect a crash after a year

Code Coverage

- A metric of how well a code is tested
  - Variety of profiling tools (e.g. `gcov`)

- Three types:

  - Line coverage – which lines of code have been executed
  - Branch coverage – which branches have been taken

  - Path coverage – which paths were taken

Code coverage example

```
if (a>2)
    a = 2;
if (b > 2)
    b = 2;
```


- How many test cases are required for full coverage?
- Path coverage: 1
- Branch coverage: 2
- Path coverage: 3

Issues with code coverage

- Does not guarantee no bugs


```
mySafeCpy(char *dst, char*
src){
  if(dst && src)
    strcpy(dst, src);
}
```


- Some parts of the program are unlikely to be covered
  - Error checking, dead code, etc.

More issues with code coverage

- Path explosion
  - n branches require $2n$ test cases for branch coverage, 2^n for code coverage
  - Loops can imply an unlimited number of paths
- Infeasible paths


```
if (a > 2)
    a = 2;
if (a < 0)
    a = 0;
```

Evolutionary fuzzing

- A.K.A. gray-box fuzzing
- Generates new inputs based on the response from the program
  - Use code coverage metrics
  - Prioritise based on access to dangerous functions

Fuzzing problems

- Crashes vs. vulnerabilities
  - A single vulnerability can cause multiple crashes
  - Can hit the same boring bug on many inputs
  - What about bugs that do not cause a crash?
- Cost
  - Many tests
  - Potentially resource intensive

Chromium ClusterFuzz

- Hundreds of virtual machines
- Thousands of simultaneous instances
- 50,000,000 tests per day (2012 data)
- 10 times bigger in 2016:
  - 14,366,459,772 test inputs over 30 days
  - 112 bugs found