Carnegie Mellon University

Advanced Automated Testing

17-313 Fall 2022

Administrivia

- HW5 due Friday (Nov 18)
  - Document N analysis tools across N team members
  - LIME analysis
- HW6 details will be released end of week
  - Checkpoint will be in recitation Nov 30 / Dec 2

S3D

Learning Goals

- Describe random test-input generation strategies such as fuzz testing
- Characterize challenges of performance testing and suggest strategies
- Reason about failures in microservice applications how chaos engineering can be applied to test resiliency of cloud-based applications
- Describe A/B testing for usability

S3D

Puzzle: Find x such `p1(x)` returns `True`

```
def p1(x):
  if x * x - 10 == 15:
    return True
  return False
```


S3D

Puzzle: Find x such `p2(x)` returns `True`

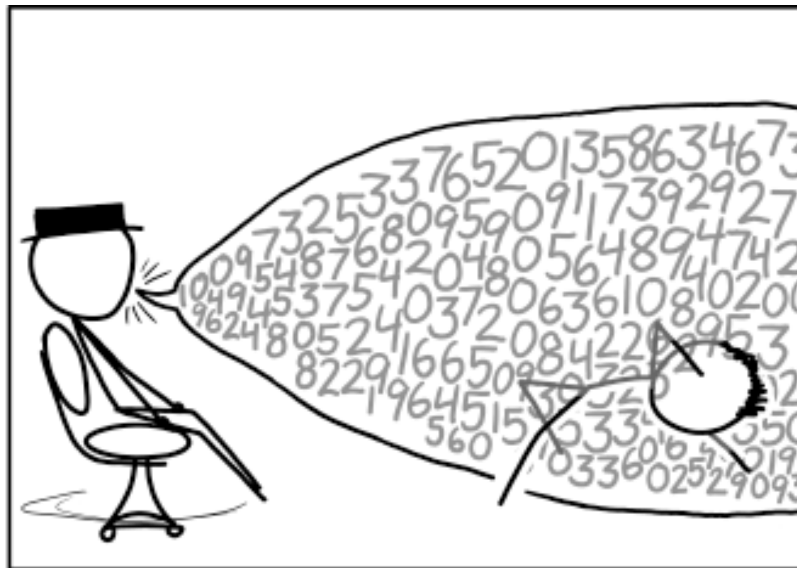
```
def p2(x):
  if x > 0 and x < 1000:
    if ((x - 32) * 5/9 == 100):
      return True
  return False
```


S3D

Puzzle: Find x such `p3(x)` returns `True`

```
def p3(x):
  if x > 3 and x < 100:
    z = x - 2
    c = 0
    while z >= 2:
      if z ** (x - 1) % x == 1:
        c = c + 1
      z = z - 1
    if c == x - 3:
      return True
  return False
```


S3D



Original: https://xkcd.com/1210 CC-BY-NC 2.5

Fuzz Testing

Security and Robustness

Barton P. Miller, Lars Fredriksen and Bryan So

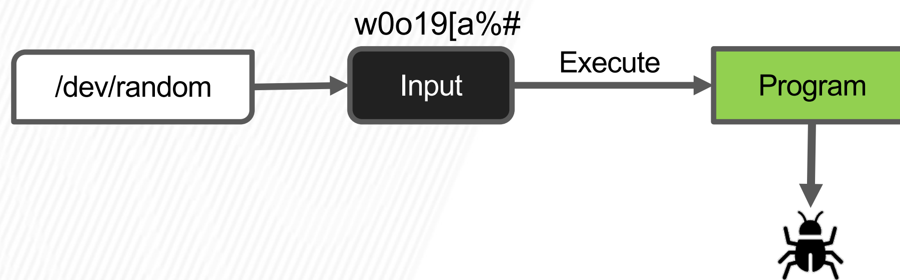
Study of the Reliability of UNIX Utilities

COMMUNICATIONS OF THE ACM/December 1990/Vol.33, No.12 33

Communications of the ACM (1990)

"On a dark and stormy night one of the authors was logged on to his workstation on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash."

Fuzz Testing



A 1990 study found crashes in:
adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi

S3D

Common Fuzzer-Found Bugs in C/C++

<u>Causes</u>: incorrect arg validation, incorrect type casting, executing untrusted code, etc.

<u>Effects</u>: buffer-overflows, memory leak, division-by-zero, use-after-free, assertion violation, etc. (“crash”)

<u>Impact</u>: security, reliability, performance, correctness

How do you make programs “crash” when a bug is encountered?

Automatic Oracles: Sanitizers

- Address Sanitizer (ASAN)   ***
- LeakSanitizer (comes with ASAN)
- Thread Sanitizer (TSAN)
- Undefined-behavior Sanitizer (UBSAN)

https://github.com/google/sanitizers

S3D

AddressSanitizer

Compile with `clang –fsanitize=address`

```
int get_element(int* a, int i) {
   return a[i];
}
```

Is it null?

```
int get_element(int* a, int i) {
   if (a == NULL) abort();
   return a[i];
}
```

Is the access out of bounds?

```
int get_element(int* a, int i) {
   if (a == NULL) abort();
   region = get_allocation(a);
   if (in_heap(region)) {
     low, high = get_bounds(region);
     if ((a + i) < low || (a +i) > high) {
       abort();
     }
   }
   return a[i];
}
```

Is this a reference to a stack-allocated variable after return?

```
int get_element(int* a, int i) {
   if (a == NULL) abort();
   region = get_allocation(a);
   if (in_stack(region)) {
     if (popped(region)) abort();
     …
   }
   if (in_heap(region)) { ... }
   return a[i];
}
```

S3D

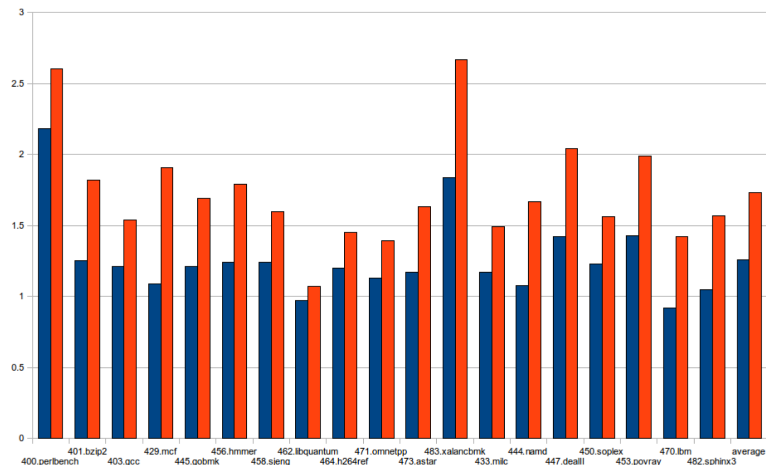
AddressSanitizer

https://github.com/google/sanitizers/wiki/AddressSanitizer

Asan is a memory error detector for C/C++. It finds:

- Use after free (dangling pointer dereference)
- Heap buffer overflow
- Stack buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs
- Memory leaks

Slowdown about 2x on SPEC CPU 2006

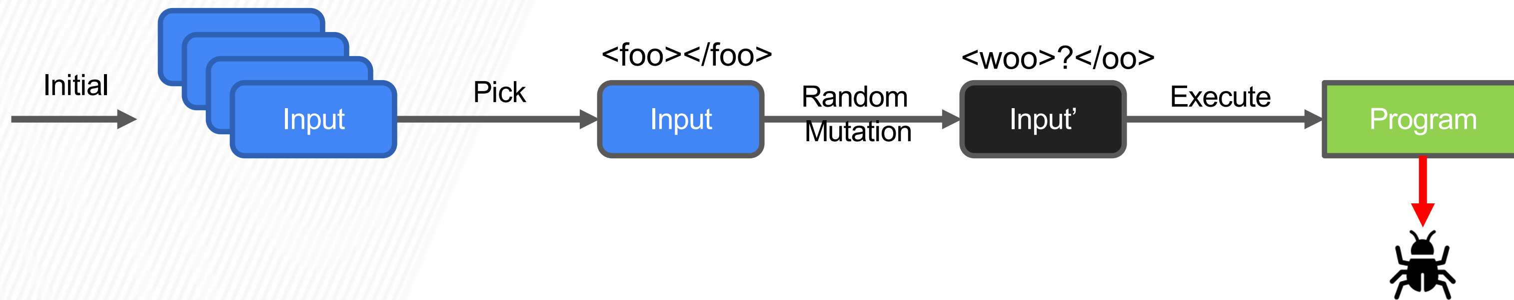


Strengths and Limitations

- **Exercise**: Write down two strengths and two weaknesses of fuzzing.
  Bonus: Write down one or more assumptions that fuzzing depends on.

S3D

Mutation-Based Fuzzing (e.g. Radamsa)



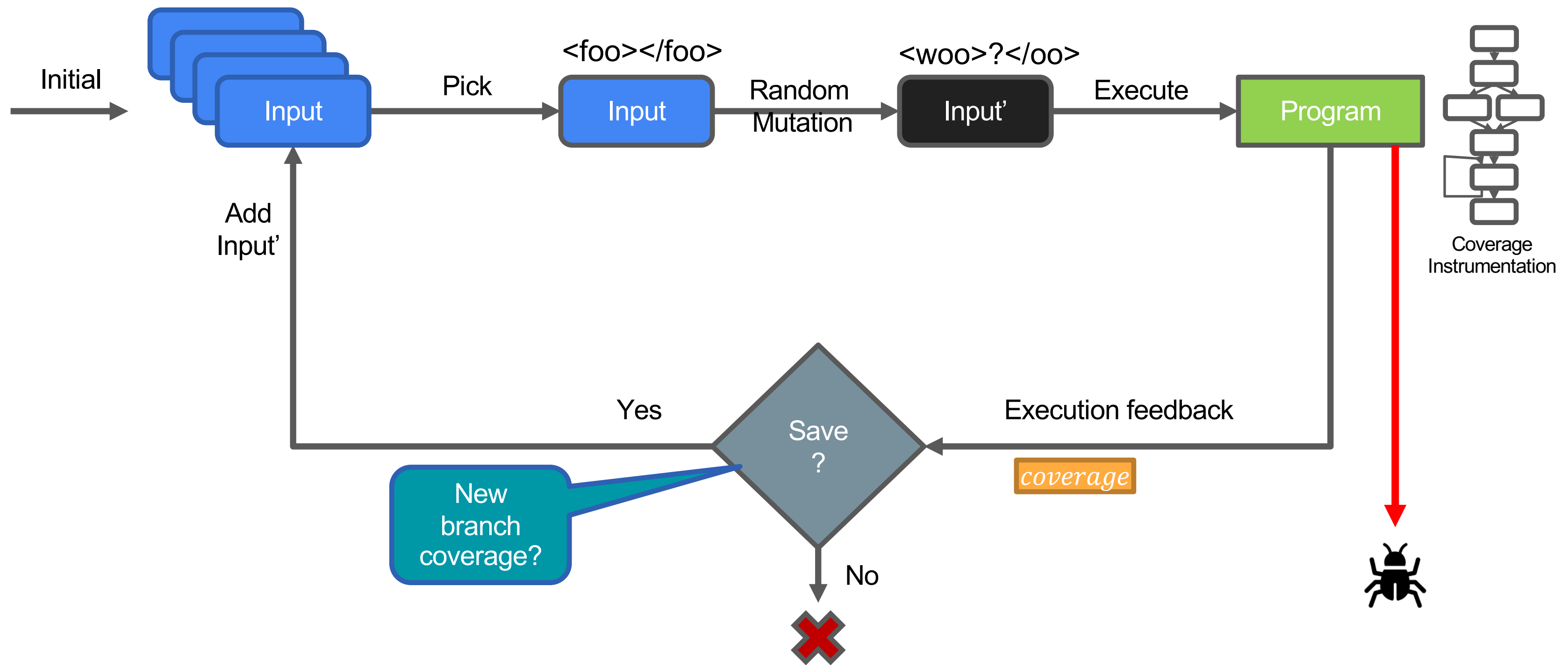
S3D

Mutation Heuristics

- Binary input
  - Bit flips, byte flips
  - Change random bytes
  - Insert random byte chunks
  - Delete random byte chunks
  - Set randomly chosen byte chunks to *interesting* values e.g. INT_MAX, INT_MIN, 0, 1, -1, …
- Text input
  - Insert random symbols relevant to format (e.g. “<“ and “>” for xml)
  - Insert keywords from a dictionary (e.g. “<project>” for Maven POM.xml)
- GUI input
  - Change targets of clicks
  - Change type of clicks
  - Select different buttons
  - Change text to be entered in forms
  - … Much harder to design

S3D

Coverage-Guided Fuzzing (e.g. AFL)



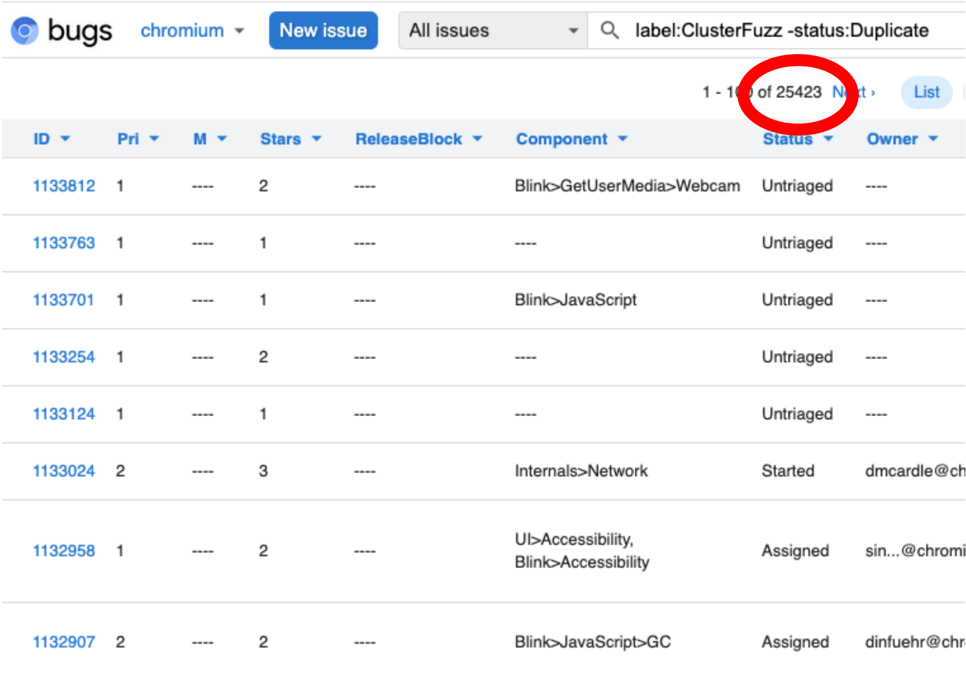
Coverage-Guided Fuzzing with AFL

The bug-o-rama trophy case

http://lcamtuf.coredump.cx/afl/

IJG jpeg 1	libjpeg-turbo 1 2	libpng 1
libtiff 1 2 3 4 5	mozjpeg 1	PHP 1 2 3 4 5 6 7 8
Mozilla Firefox 1 2 3 4	Internet Explorer 1 2 3 4	Apple Safari 1
Adobe Flash / PCRE 1 2 3 4 5 6 7	sqlite 1 2 3 4...	OpenSSL 1 2 3 4 5 6 7
LibreOffice 1 2 3 4	poppler 1 2...	freetype 1 2
GnuTLS 1	GnuPG 1 2 3 4	OpenSSH 1 2 3 4 5
PuTTY 1 2	ntpd 1 2	nginx 1 2 3
bash (post-Shellshock) 1 2	tcpdump 1 2 3 4 5 6 7 8 9	JavaScriptCore 1 2 3 4
pdfium 1 2	ffmpeg 1 2 3 4 5	libmatroska 1
libarchive 1 2 3 4 5 6 ...	wireshark 1 2 3	ImageMagick 1 2 3 4 5 6 7 8 9 ...
BIND 1 2 3 ...	QEMU 1 2	lcms 1

ClusterFuzz @ Chromium



Testing Performance

S3D

Performance Testing

- Goal: Identify *performance bugs*. What are these?
  - Unexpected bad performance on some subset of inputs
  - Performance degradation over time
  - Difference in performance across versions or platforms
- Not as easy as functional testing. What's the oracle?
  - Fast = good, slow = bad // but what's the threshold?
  - How to get reliable measurements?
  - How to debug where the issue lies?

S3D

Performance Regression Testing

- Measure execution time of critical components
- Log execution times and compare over time

Job 12e96643840000

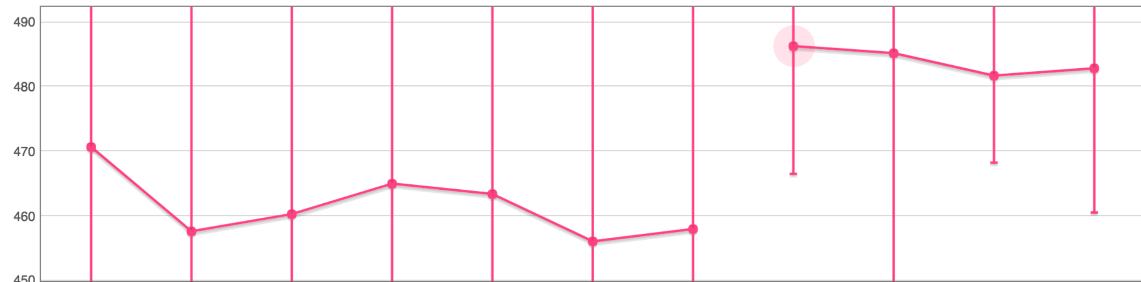
Issue 808613 · Analyze benchmark results · 2.0 hours · 2/14/2018, 9:48:34 AM

Differences found after commits

Re-record loading.desktop story set by ksakamoto@chromium.org

Job arguments

benchmark	loading.desktop
chart	cpuTimeToFirstMeaningfulPaint
configuration	chromium-rel-mac11-pro
statistic	avg
story	Pantip
target	telemetry_perf_tests
tir_label	warm
trace	Pantip



Re-record loading.desktop story set by ksakamoto@chromium.org

Build

builder	Mac Builder
isolate_hash	630b5fe7ae1b260e78db8823309 9249b5640517b

Test

task_id	3baea4beaa7f1710
bot_id	build197-b4
isolate_hash	146eb87de6d2594cc3a9ee9f351 8f69fc3d0c2c3

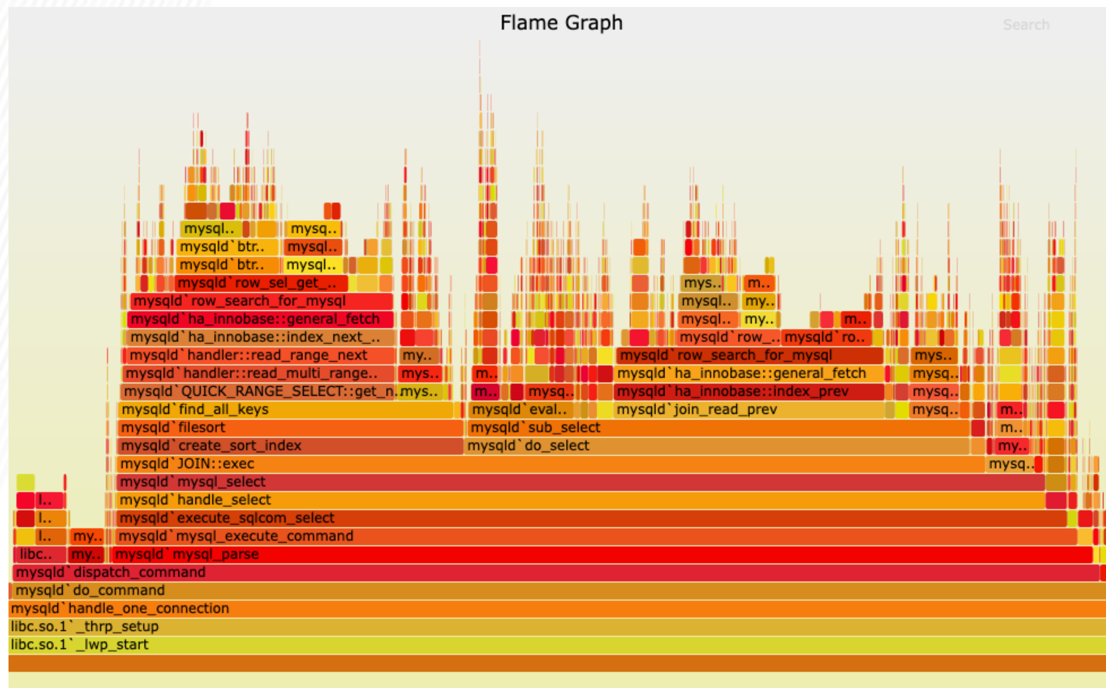
Values

trace	Pantip_2018-02-14_11-40-07_93865.html
trace	Pantip_2018-02-14_11-40-42_21734.html

S3D

Profiling

- Finding bottlenecks in execution time and memory
- Flame graphs are a popular visualization of resource consumption by call stack.



Domain-Specific Perf Testing (e.g. JMeter)

Apache JMeter Dashboard by UbikLoadPack

Apr 10, 2018 15:56:21 to Apr 10, 2018 16:03:40

data_source: jmeter_influx | application: JMeter_demo | transaction: JR_OK | Start/stop marker

Summary

Total Requests	Failed Requests	Received Bytes	Sent Bytes	Error Rate %
2107 Requests	1684 Failed	1 MiB	1 MiB	79.92%

Total Throughput

8.0, 7.0, 6.0, 5.0, 4.0

15:58, 16:00, 16:02

	min	max	avg
Req / s	4.40	7.60	6.94

Total Errors

45, 40, 35, 30, 25, 20, 15

15:58, 16:00, 16:02

	total
Num of Errors	2 K

Active Threads

6, 5, 4, 3, 2, 1, 0

15:58, 16:00, 16:02

2018-04-10 16:03:40

Threads:

	current
Threads	

Transactions Response Times (95th pct)

http://jmeter.apache.org

Performance-driven Design

- Modeling and simulation
  - e.g. queuing theory
- Specify load distributions and derive or test configurations

Stress testing

- Robustness testing technique: test beyond the limits of normal operation.
- Can apply at any level of system granularity.
- Stress tests commonly put a greater emphasis on robustness, availability, and error handling under a heavy load, than on what would be considered “correct” behavior under normal circumstances.

S3D

Soak testing

- **Problem:** A system may behave exactly as expected under artificially limited execution conditions.
  - E.g., Memory leaks may take longer to lead to failure (also motivates static/dynamic analysis, but we'll talk about that later).
- **Soak testing:** testing a system with a significant load over a significant period of time (*positive*).
- Used to check reaction of a subject under test under a possible simulated environment for a given duration and for a given threshold.

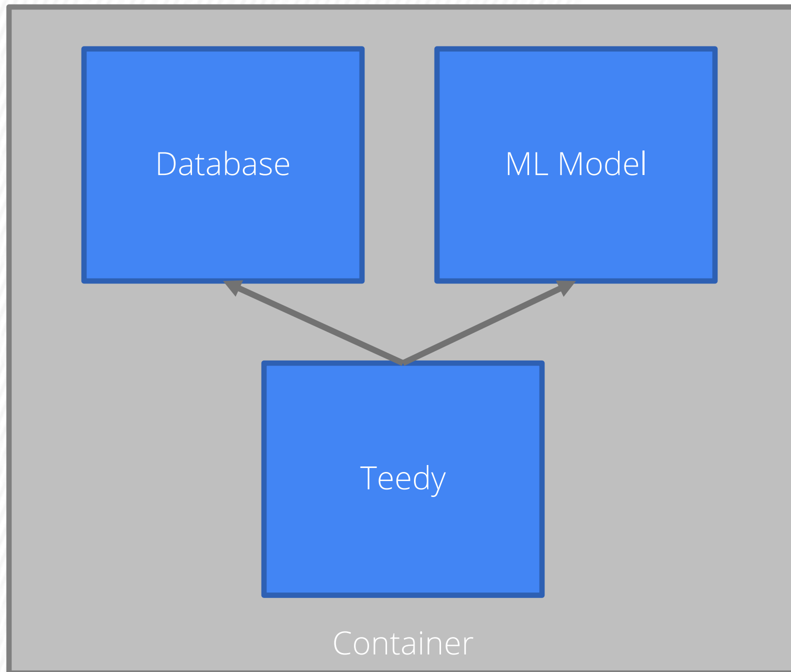
S3D

Microservice Failures and Chaos Engineering

(Slides credit Christopher Meiklejohn)

S3D

Monolithic Application



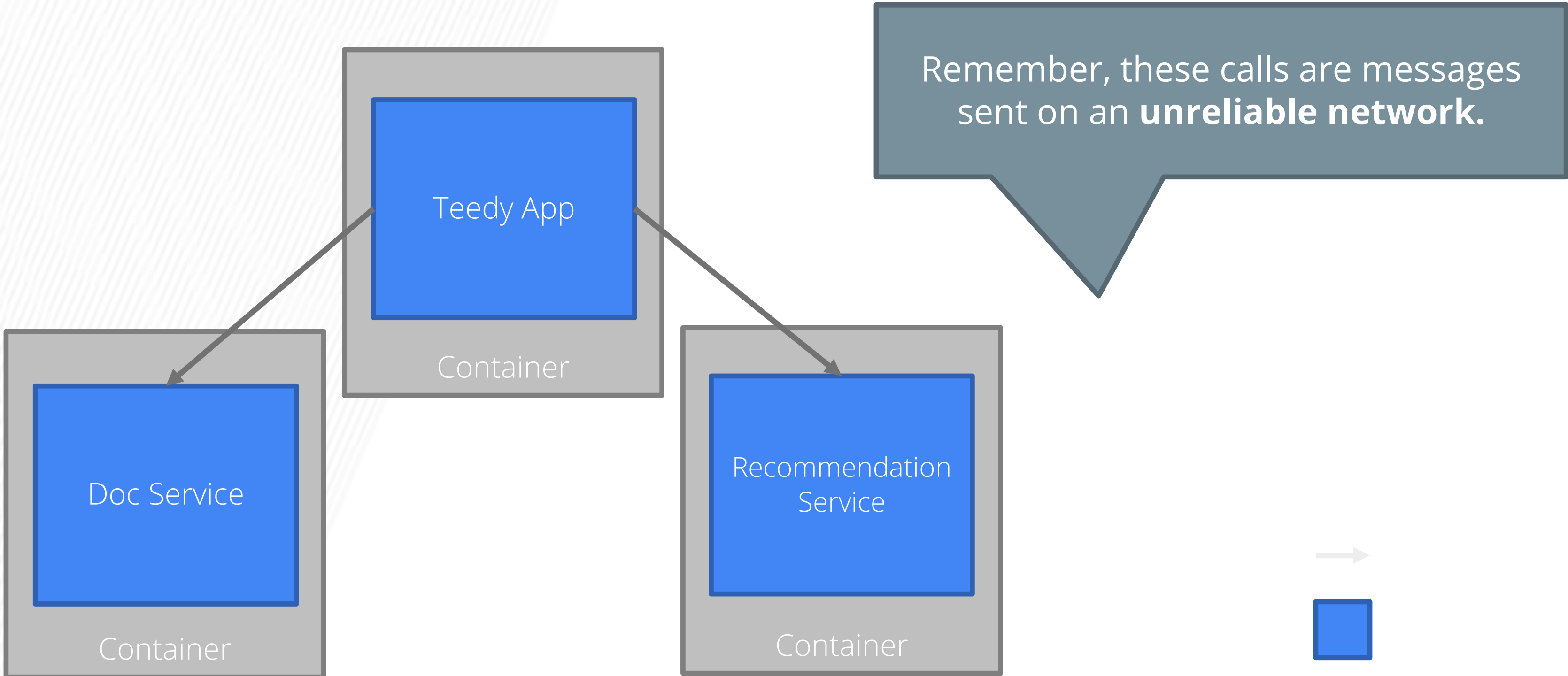
What kind of failures can happen here?

How likely is that error to happen?

How do I fix it?

S3D

Microservice Application



S3D

Failures in Microservice Architectures

1. Network may **be partitioned**
2. Server instance **may be down**
3. Communication between services may **be delayed**
4. Server **could be overloaded** and responses delayed
5. Server **could run out of** memory or CPU

All of these issues
can be indistinguishable
from one another!

Making the calls across the network to multiple machines makes the probability that the system is operating under failure **much higher.**

These are the problems of
latency and **partial failure.**

S3D

Where Do We Start?

How do we even **begin to test these scenarios?**

Is there any **software** that can be used to test these types of failures?

Let's look at a **few ways** companies do this.

S3D

Game Days

Purposely **injecting failures** into critical systems in order to:

- Identify **flaws** and “latent defects”
- Identify **subtle dependencies** (which may or may not lead to a flaw/defect)
- Prepare a **response** for a disastrous event

Comes from “resilience engineering” typical in high-risk industries

Practiced by Amazon, Google, Microsoft, Etsy, Facebook, Flickr, etc.

S3D

Game Days

Large-scale applications are built on and with **“unreliable”** components

Failure is inevitable (fraction of percent; at Google scale, ~multiple times)

Goals:

- **Preemptively trigger** the failure, observe, and fix the error
- Script testing of **previous failures** and ensure system remains resilient
- Build the necessary relationships between teams **before** disaster strikes

S3D

Example: Amazon GameDay

Full data center destruction (Amazon EC2 region)

- No advanced notice of **which** data center will be taken offline
- No notice of **when** the data center will be taken offline
- Only advance notice (months) that a GameDay **will be happening**
- **Real failures in the production environment**

Discovered **latent defect** where the monitoring infrastructure responsible for detecting errors and paging employees **was located in the zone of the failure!**

Not all failures can be actually performed and must be **simulated!**

Other examples: Google

Terminate network in Sao Paulo for testing:

- Hidden dependency takes down links in Mexico which would have remained undiscovered without testing

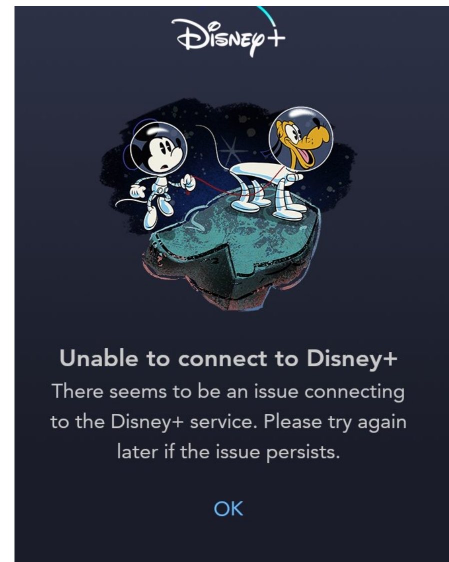
Turn off data center to find that machines won’t come back:

- Ran out of DHCP leases (for IP address allocation) when a large number of machines come back online unexpectedly.

S3D

Real Issues: Disney+ Launch

- Lots of issues reported on launch day.
- Disney had planned for a spike in traffic.
  - Tested massive concurrent video streaming capability.
- BUT: the stress was in paths other than streaming
  - User account creation
  - Logins and auth
  - Browsing old titles



S3D

Netflix is another heavy cloud user...

Significant deployment in Amazon Web Services in order to remain **elastic** in times of high and low load (first public, 100% w/o content delivery.)

Pushes code into production and modifies runtime configuration hundreds of times a day

Key metric: **availability**

SPS is the primary indicator of the system's overall health.

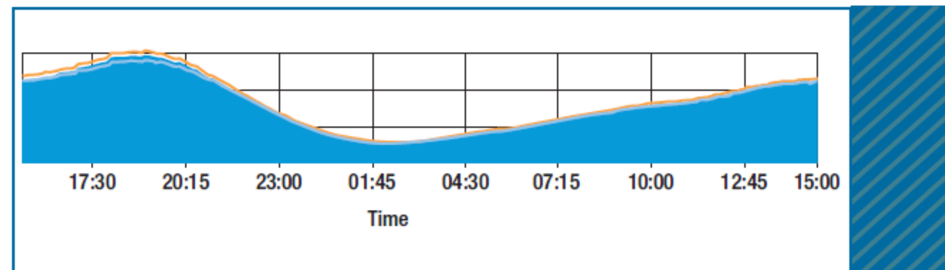


FIGURE 2. A graph of SPS ([stream] starts per second) over a 24-hour period. This metric varies slowly and predictably throughout a day. The orange line shows the trend for the prior week. The *y*-axis isn't labeled because the data is proprietary.

S3D

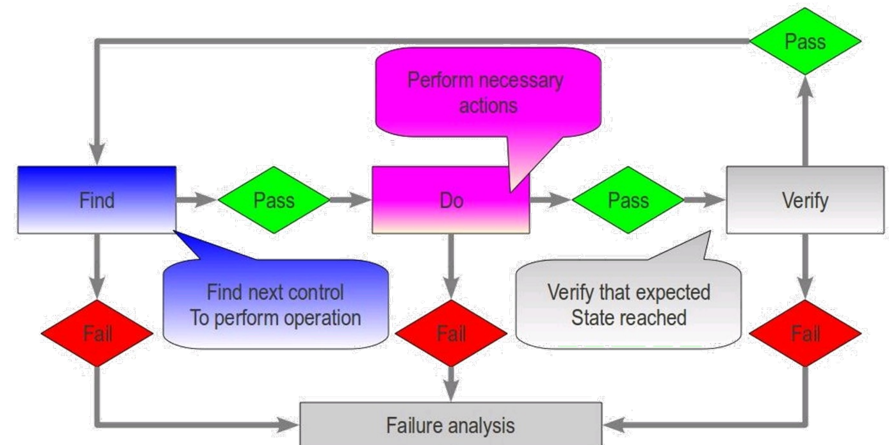
Chaos monkey/Simian army

- A Netflix infrastructure testing system.
- “Malicious” programs randomly trample on components, network, datacenters, AWS instances…
  - Force failure of components to make sure that the system architecture is resilient to unplanned/random outages.
- Netflix has open-sourced their chaos monkey code.

Testing Usability

Automating GUI/Web Testing

- This is hard
- Capture and Replay Strategy
  - mouse actions
  - system events
- Test Scripts: (click on button labeled "Start" expect value X in field Y)
- Lots of tools and frameworks
  - e.g. Selenium for browsers
- (Avoid load on GUI testing by separating model from GUI)
- Beyond functional correctness?



S3D

Manual Testing?

- Live System?
- Extra Testing System?
- Check output / assertions?
- Effort, Costs?
- Reproducible?

GENERIC TEST CASE: USER SENDS MMS WITH PICTURE ATTACHED.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Go to Messages Menu	Message Menu appears
3	Select "Create new Message"	Message Editor screen opens
4	Add Recipient	Recipient is added
5	Select "Insert Picture"	Insert Picture Menu opens
6	Select Picture	Picture is Selected
7	Select "Send Message"	Message is correctly sent

S3D

Usability: A/B testing

- Controlled randomized experiment with two variants, A and B, which are the control and treatment.
- One group of users given A (current system); another random group presented with B; outcomes compared.
- Often used in web or GUI-based applications, especially to test advertising or GUI element placement or design decisions.

S3D

Example

- A company sends an advertising email to its customer database, varying the photograph used in the ad...

Example: group A (99% of users)

Act now! Sale ends soon!

S3D

Example: group B (1%)

Act now! Sale ends soon!

S3D

A/B Testing

- Requires good metrics and statistical tools to identify significant differences.
- E.g. clicks, purchases, video plays
- Must control for confounding factors

S3D