AI for Security (AI4Sec) Foundations

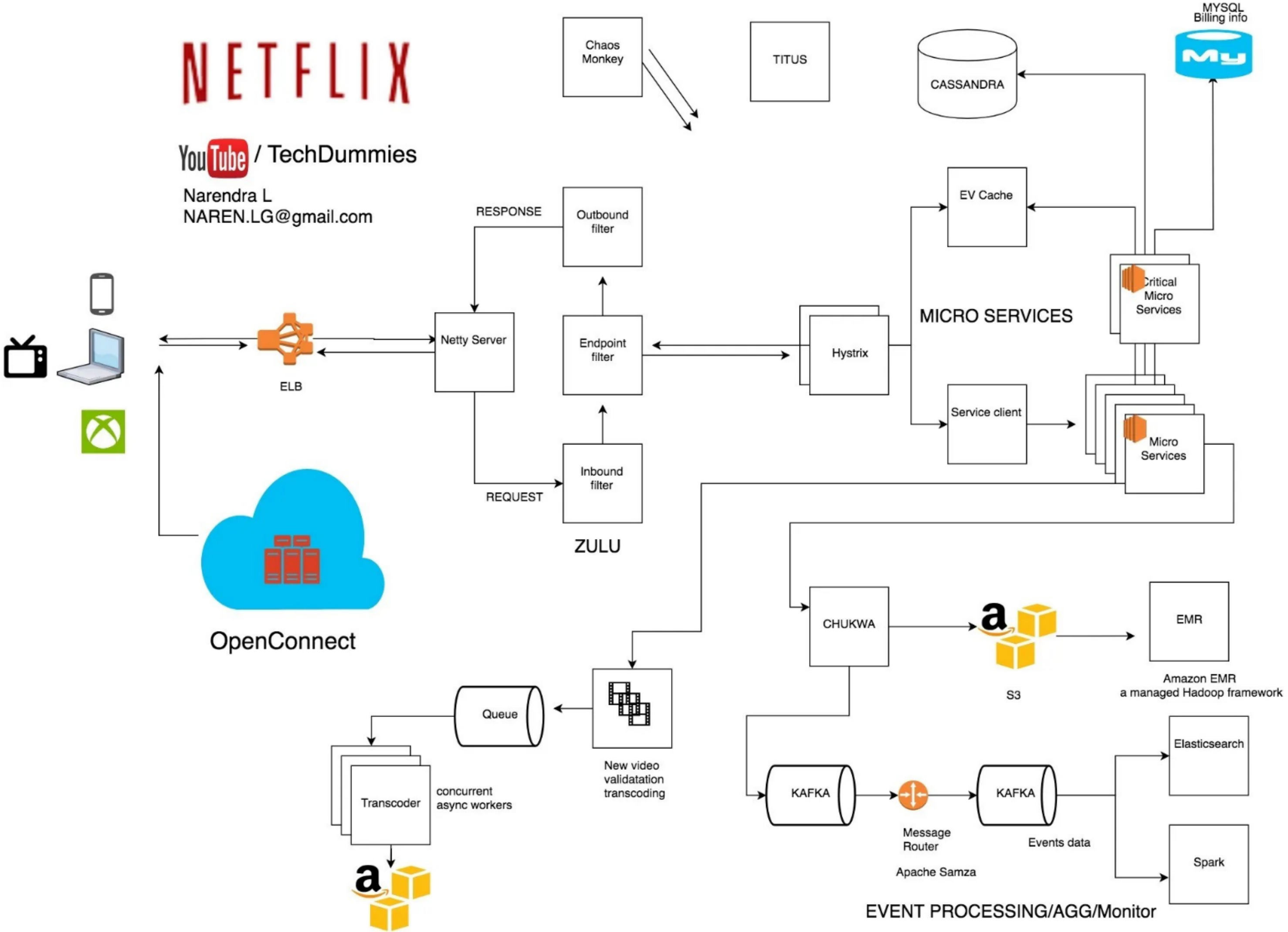
Security + AI

COMP-5870/6870

Engineered System

"a combination of components that work in synergy to collectively perform a useful function. The engineered system could, for example, wholly or in part constitute a new technology for a new product line a new manufacturing process, a technology to improve the delivery of a service, or an infrastructure system."

Netflix Infrastructure

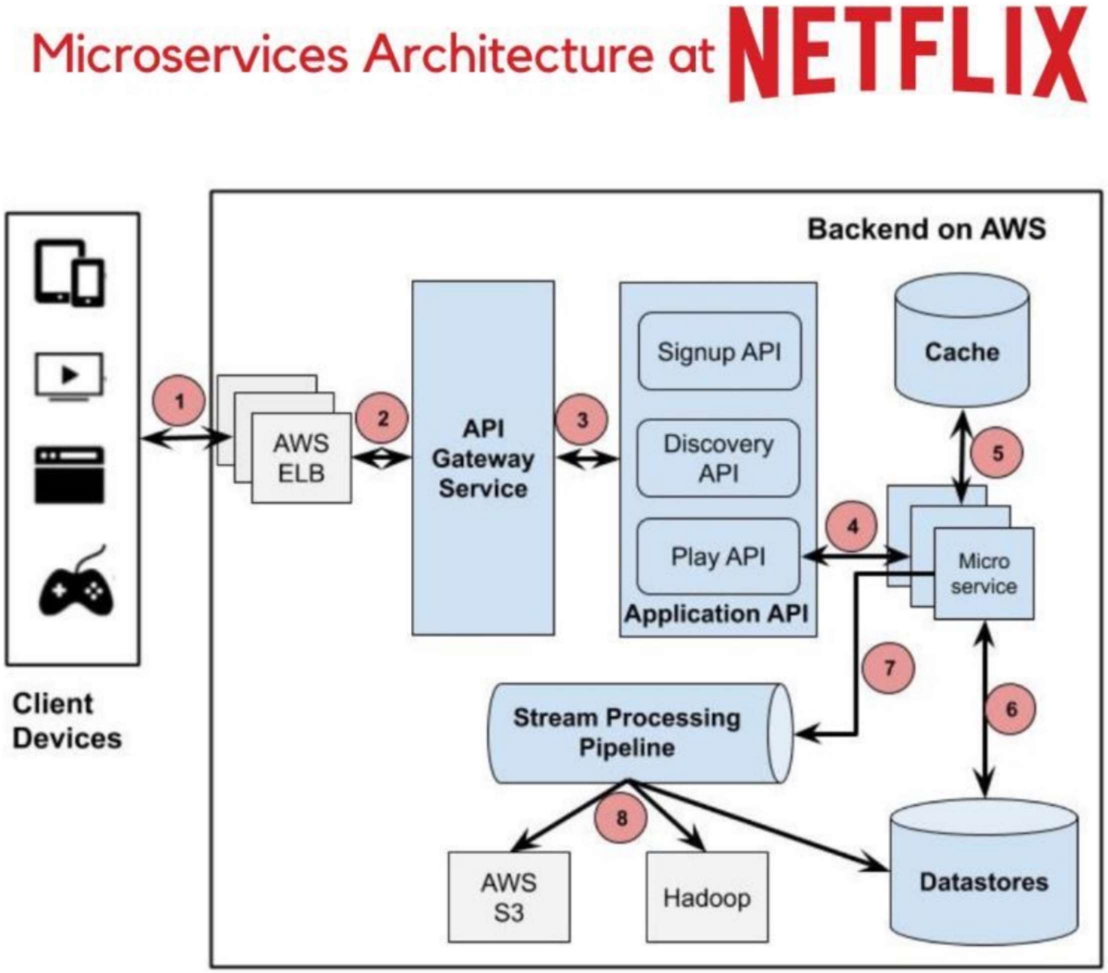


NETFLIX

YouTube / TechDummies

Narendra L
NAREN.LG@gmail.com

Netflix Infrastructure

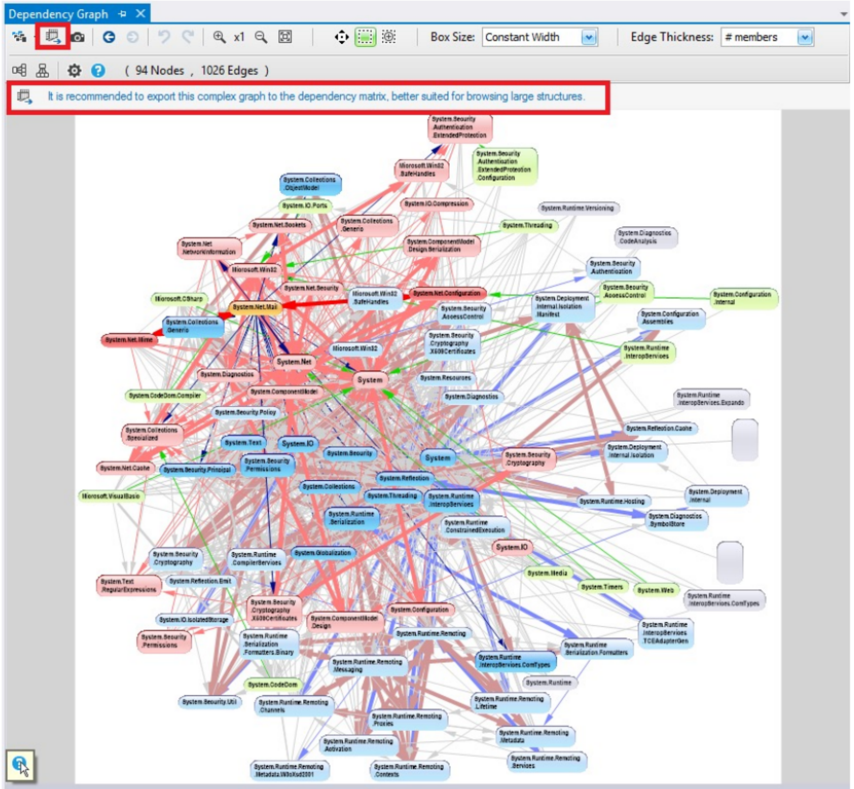


Credits: Cao Duc Nguyen

Human-Centric Approach

Human-Centric Approach

- Graph dependencies and relationships



Dependency Graph

Box Size: Constant Width

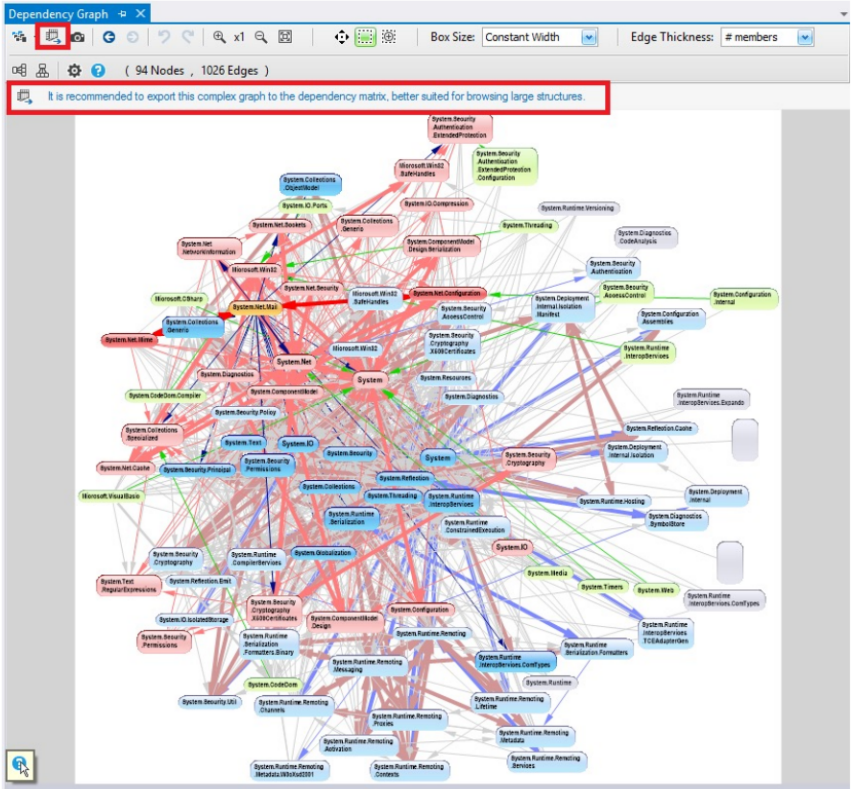
Edge Thickness: # members

(94 Nodes , 1026 Edges)

It is recommended to export this complex graph to the dependency matrix, better suited for browsing large structures.

Human-Centric Approach

- Graph dependencies and relationships
- Experts review and predict impact of various outages



Dependency Graph

Box Size: Constant Width

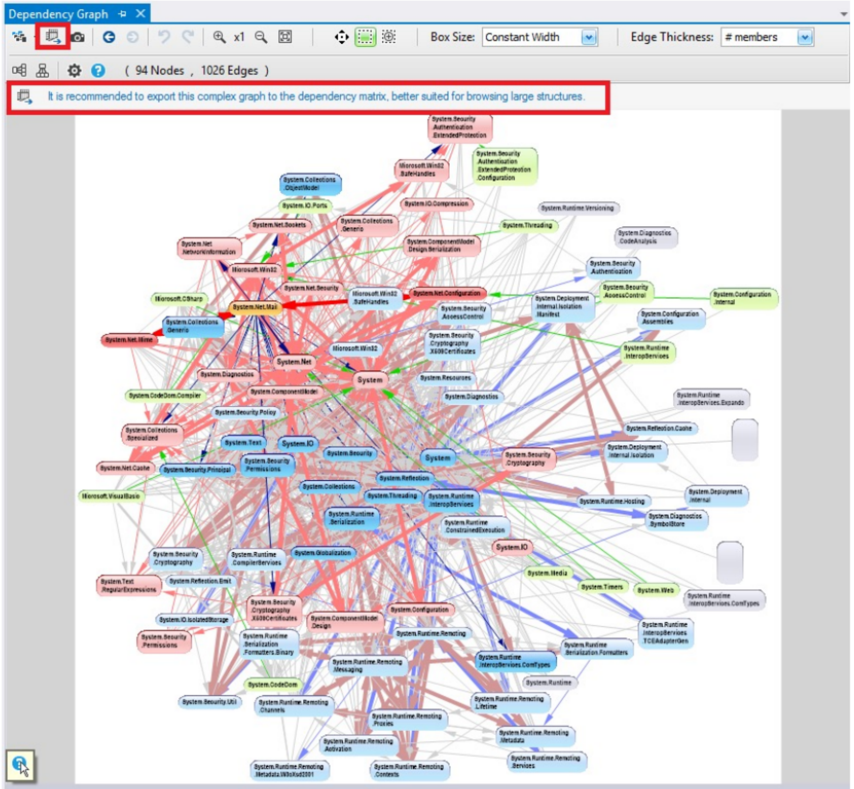
Edge Thickness: # members

(94 Nodes , 1026 Edges)

It is recommended to export this complex graph to the dependency matrix, better suited for browsing large structures.

Human-Centric Approach

- Graph dependencies and relationships
- Experts review and predict impact of various outages
- Evaluate via thought experiments and some testing



Box Size: Constant Width

Edge Thickness: # members

(94 Nodes , 1026 Edges)

It is recommended to export this complex graph to the dependency matrix, better suited for browsing large structures.

Automation-Centric Approach

Automation-Centric Approach

- Write 10 lines of code
  - Select from list
  - Run command

Automation-Centric Approach

- Write 10 lines of code
  - Select from list
  - Run command
- Give list of every service, process, job, etc and hard-crash selected elements

Automation-Centric Approach

- Write 10 lines of code
  - Select from list
  - Run command
- Give list of every service, process, job, etc and hard-crash selected elements
- Run Mon-Fri @ 7am

38
CHAOS
monkey

Automation-Centric Approach

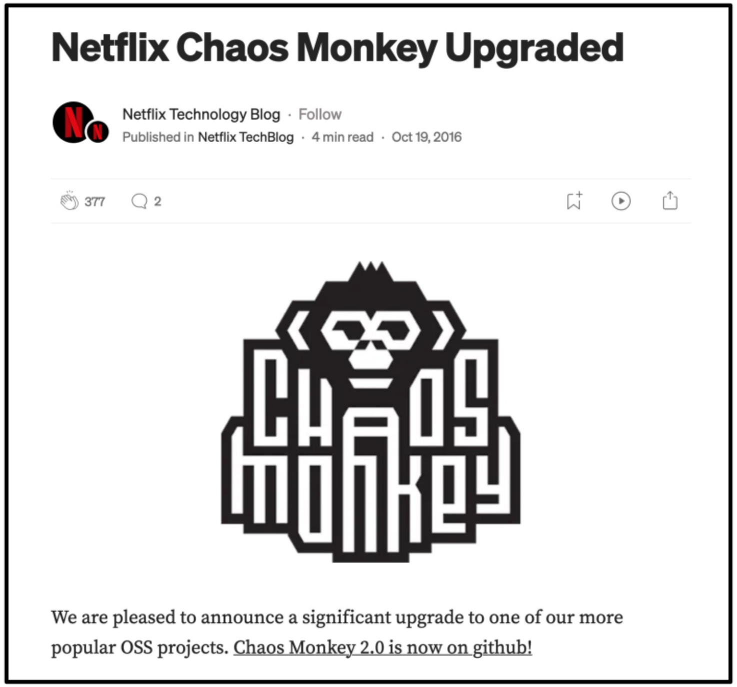
- Write 10 lines of code
  - Select from list
  - Run command

- Give list of every service, process, job, etc and hard-crash selected elements

- Run Mon-Fri @ 7am

Chaos Engineering

The goal of **chaos engineering** is to improve resilience by intentionally but unknowingly triggering system failures so they can be remediated.



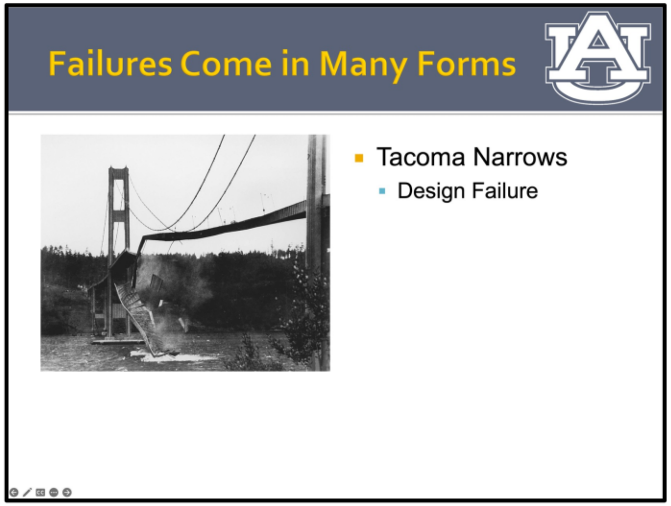
Netflix Chaos Monkey Upgraded

Netflix Technology Blog · Follow
Published in Netflix TechBlog · 4 min read · Oct 19, 2016

377 2

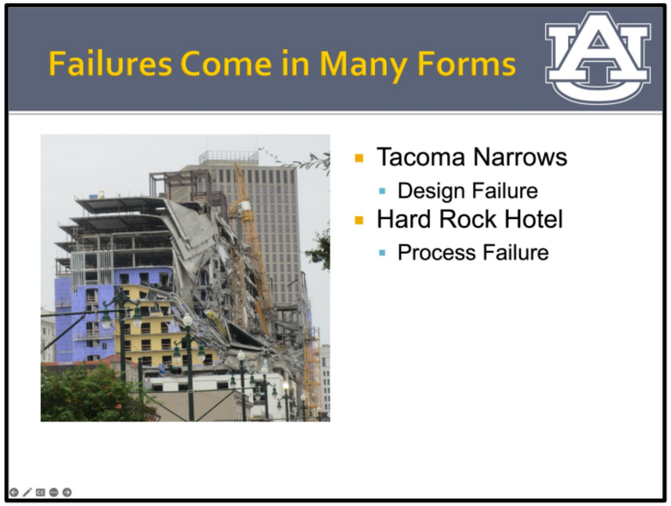
We are pleased to announce a significant upgrade to one of our more popular OSS projects. Chaos Monkey 2.0 is now on github!

Predicting/Avoiding Failure



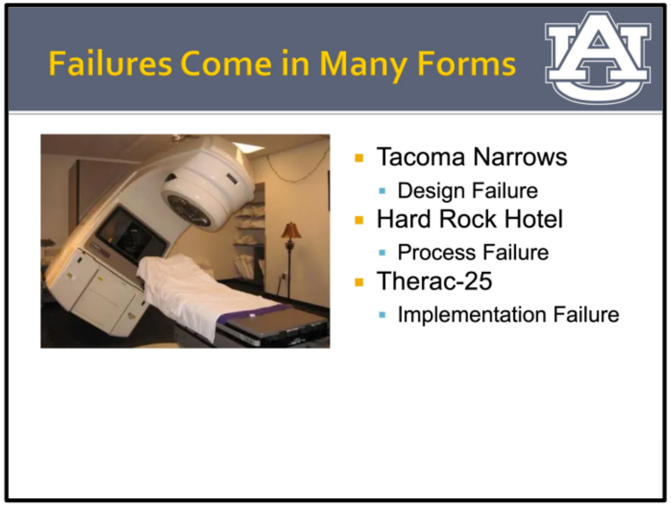
Failures Come in Many Forms

- Tacoma Narrows
  - Design Failure



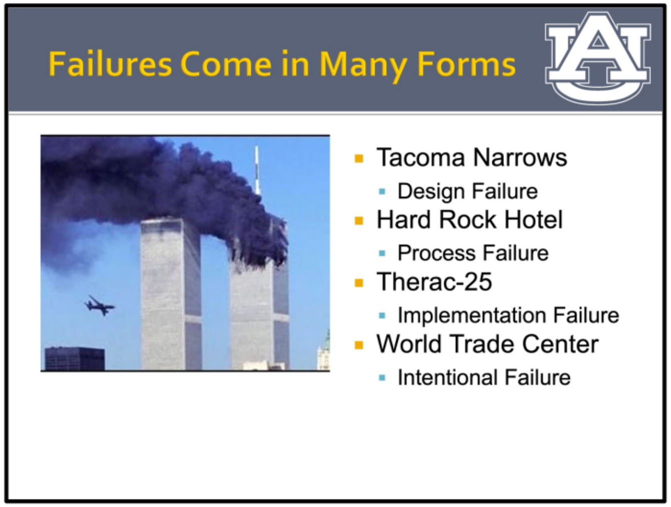
Failures Come in Many Forms

- Tacoma Narrows
  - Design Failure
- Hard Rock Hotel
  - Process Failure



Failures Come in Many Forms

- Tacoma Narrows
  - Design Failure
- Hard Rock Hotel
  - Process Failure
- Therac-25
  - Implementation Failure



Failures Come in Many Forms

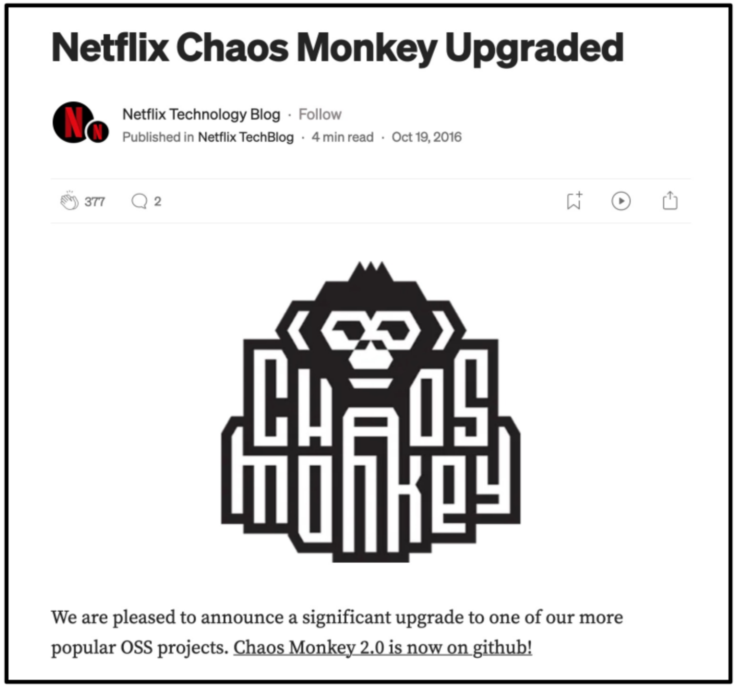
- Tacoma Narrows
  - Design Failure
- Hard Rock Hotel
  - Process Failure
- Therac-25
  - Implementation Failure
- World Trade Center
  - Intentional Failure

Chaos Engineering

The goal of **chaos engineering** is to improve resilience by intentionally but unknowingly triggering system failures so they can be remediated.

In order of preference:

1. Not have outages
2. Outages that can be immediately reverted
3. Have outages when devs/SREs are at work
4. Have outages

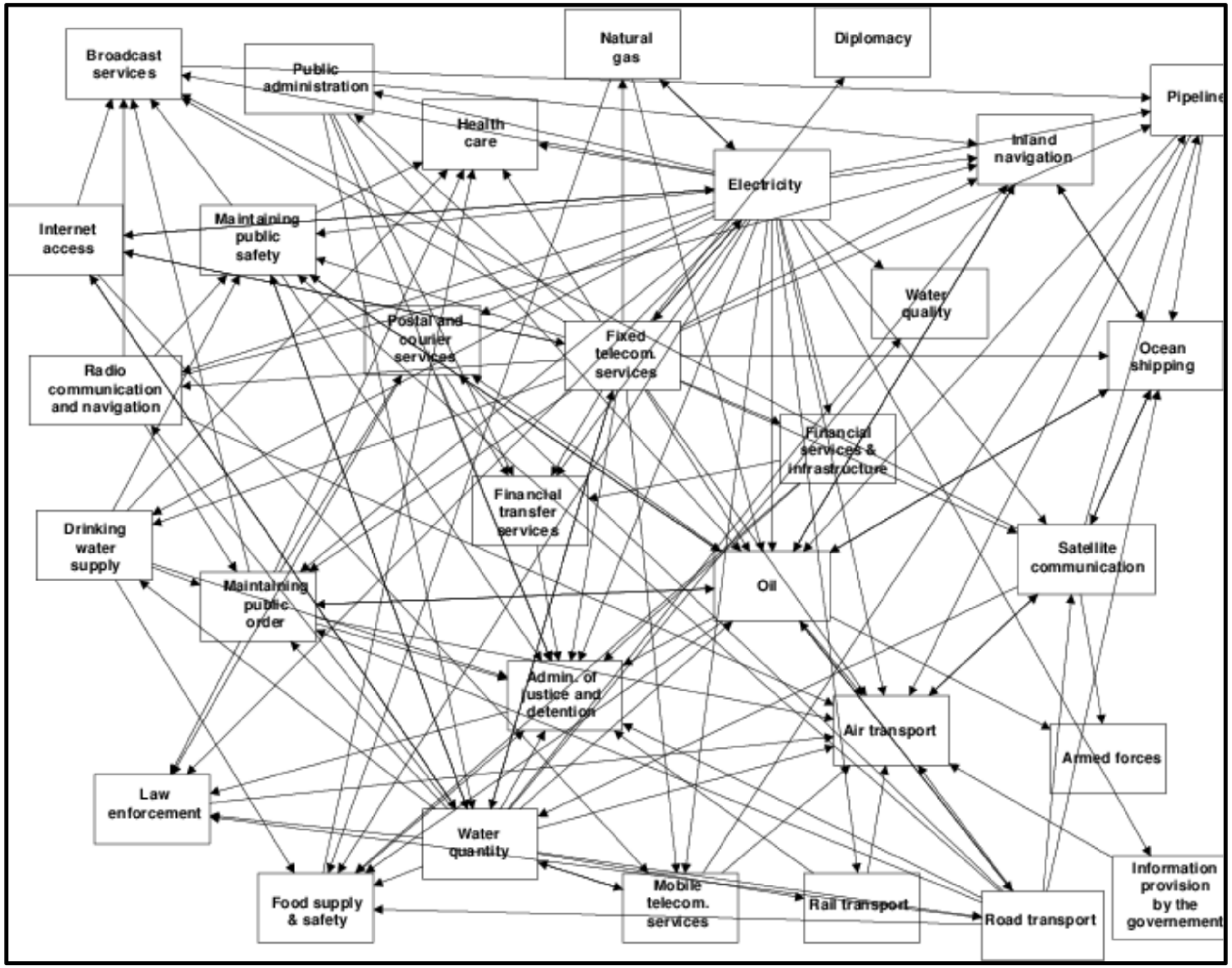


Netflix Chaos Monkey Upgraded

Netflix Technology Blog · Follow
Published in Netflix TechBlog · 4 min read · Oct 19, 2016

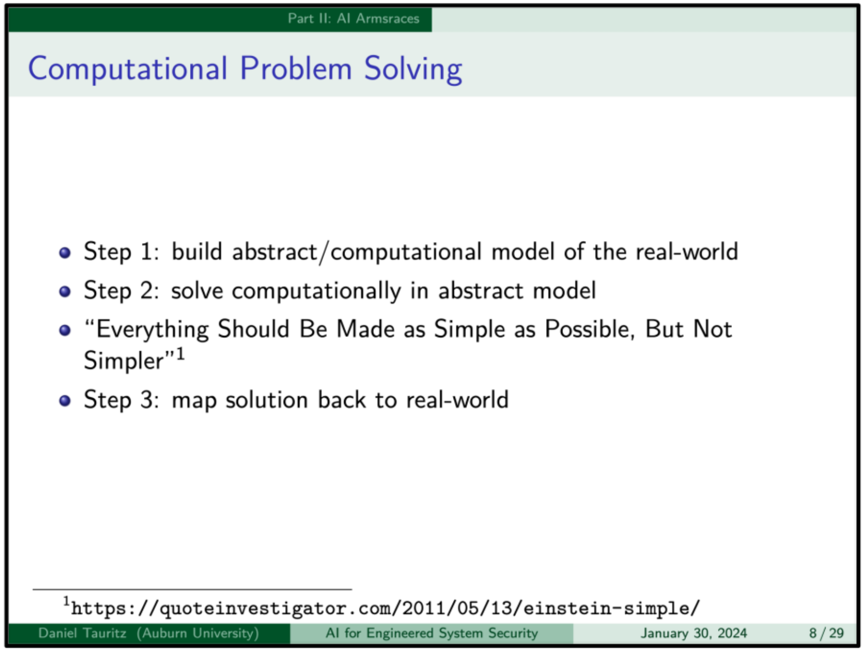
We are pleased to announce a significant upgrade to one of our more popular OSS projects. Chaos Monkey 2.0 is now on github!

Can't YOLO All Scenario



AI is Useful in Some Scenarios

- Infeasibly large search space



Part II: AI Armsraces

Computational Problem Solving

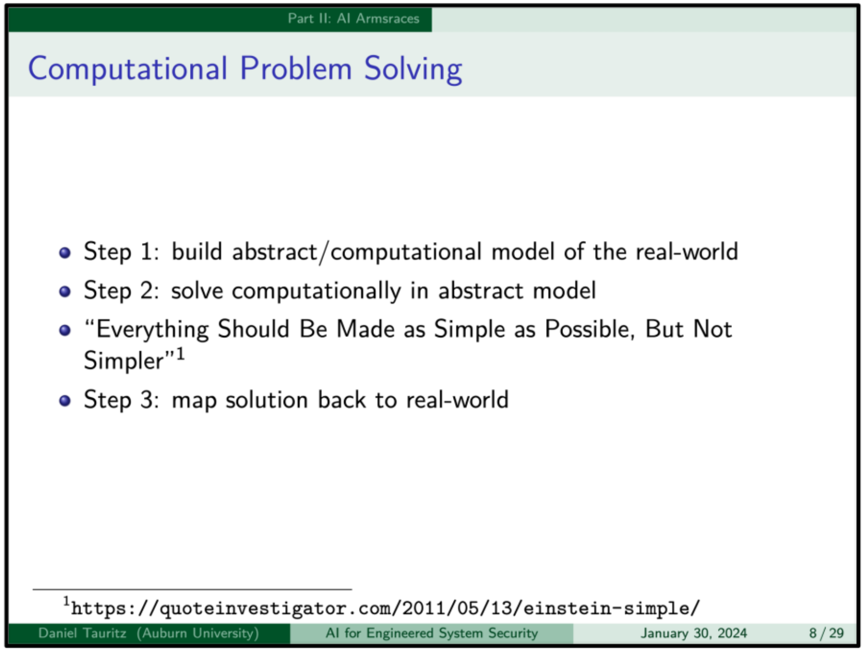
- Step 1: build abstract/computational model of the real-world
- Step 2: solve computationally in abstract model
- "Everything Should Be Made as Simple as Possible, But Not Simpler"[1]
- Step 3: map solution back to real-world

[1] `https://quoteinvestigator.com/2011/05/13/einstein-simple/`

Daniel Tauritz (Auburn University) AI for Engineered System Security January 30, 2024 8 / 29

AI is Useful in Some Scenarios

- Infeasibly large search space
- Computational experimentation
  - Emulates/Simulate



Part II: AI Armsraces

Computational Problem Solving

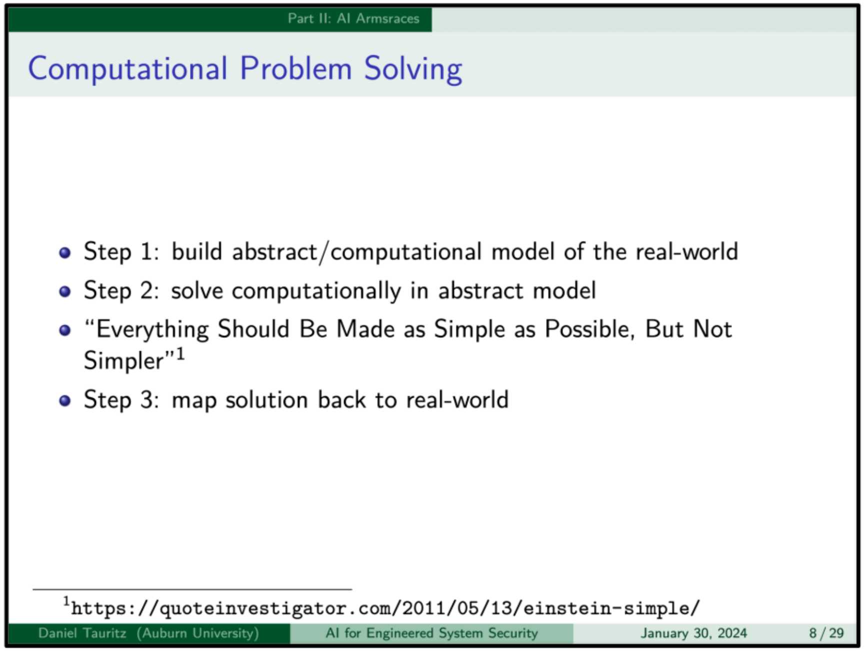
- Step 1: build abstract/computational model of the real-world
- Step 2: solve computationally in abstract model
- "Everything Should Be Made as Simple as Possible, But Not Simpler"[1]
- Step 3: map solution back to real-world

[1] https://quoteinvestigator.com/2011/05/13/einstein-simple/

Daniel Tauritz (Auburn University) AI for Engineered System Security January 30, 2024 8 / 29

AI is Useful in Some Scenarios

- Infeasibly large search space
- Computational experimentation
  - Emulates/Simulate
- Gradient available to quantify progress
  - Implicit or explicit



Part II: AI Armsraces

Computational Problem Solving

- Step 1: build abstract/computational model of the real-world
- Step 2: solve computationally in abstract model
- "Everything Should Be Made as Simple as Possible, But Not Simpler"[1]
- Step 3: map solution back to real-world

[1] https://quoteinvestigator.com/2011/05/13/einstein-simple/

Daniel Tauritz (Auburn University) AI for Engineered System Security January 30, 2024 8 / 29

Deserializing Data

Parsing untrusted data is a very common source of buffer overflows due to complexity.

Fuzz Testing

Fuzzing is an approach to software testing in which the invariant being evaluated is not dependent on the value of inputs.

Fuzz Testing

Fuzzing is an approach to software testing in which the invariant being evaluated is not dependent on the value of inputs.

Conventional Testing

- Unit Test Invariant:
  - 2 + 4 =?= 6
- Integration Invariant:
  - 2 + 4 and translated to German =?= “sechs”

Fuzz Testing

Fuzzing is an approach to software testing in which the invariant being evaluated is not dependent on the value of inputs.

Conventional Testing

- Unit Test Invariant:
  - 2 + 4 =?= 6
- Integration Invariant:
  - 2 + 4 and translated to German =?= “sechs”

Fuzz Testing

- Unit Test Invariant:
  - int + int =?= int
- Integration Invariant:
  - int + int and translated to non-English =/= word in English dictionary

Buffer Overflows

Buffer overflows are class of memory corruption bugs where a program attempts to put **too-much data** into **too-small** of a **memory** allocation.

```
void print_name(char** argv) {
    char buf[4];
    strcpy(buf, argv[0]);
    printf(“Running: %s”, buf);
}
```

Buffer Overflow Example

```
void handle_input(char *str) {
   char buffer[4];
   strcpy(buffer, str);
}

int main() {
  char str = get_user_input();
  handle_input(str);
}
```

Buffer Overflow Example

```
void handle_input(char *str) {
   char buffer[4];
   strcpy(buffer, str);
}

int main() {
  char str = get_user_input();
  handle_input(str);
}
```

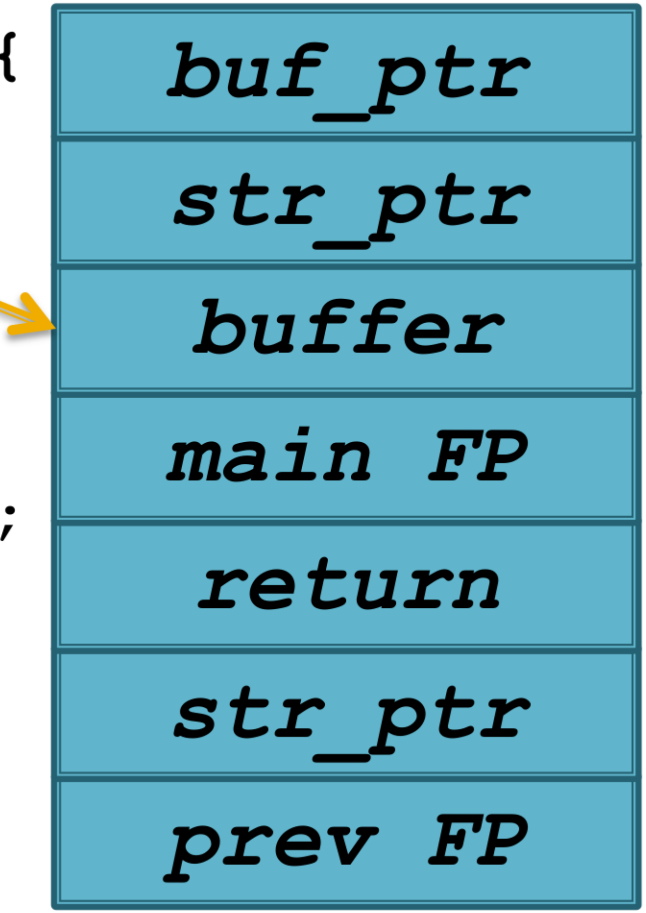
4 Bytes

Attacker Controlled Bytes

Buffer Overflow Example

```
void handle_input(char *str) {
    char buffer[4];
    strcpy(buffer, str);
}

int main() {
   char str = get_user_input();
   handle_input(str);
}
```



buf_ptr

str_ptr

buffer

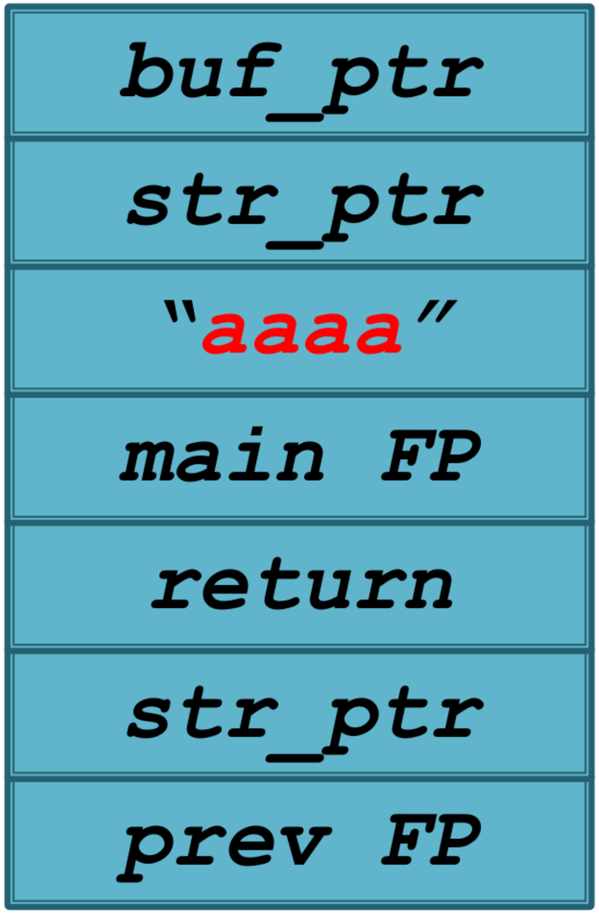
main FP

return

str_ptr

prev FP

Buffer Overflow Example



buf_ptr

str_ptr

“aaaa”

main FP

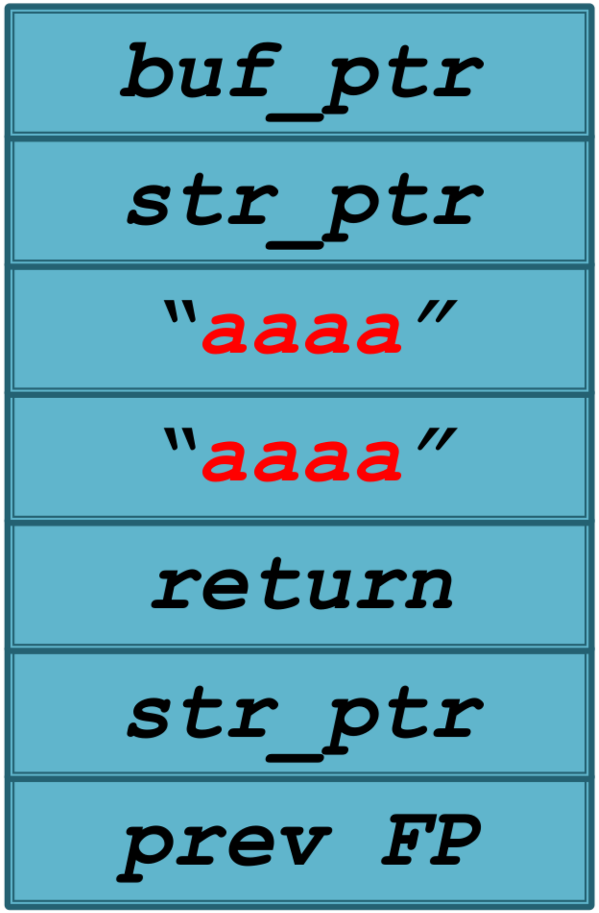
return

str_ptr

prev FP

```
python –c “print ‘a’* 1024” | app
```

Buffer Overflow Example


buf_ptr

str_ptr

“aaaa”

“aaaa”

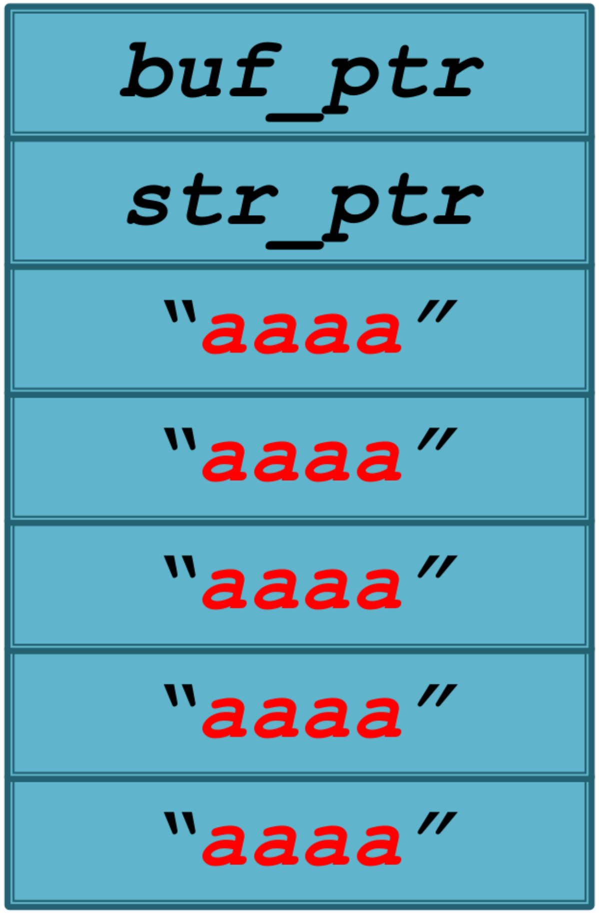
return

str_ptr

prev FP

```
python –c “print ‘a’* 1024” | app
```

Buffer Overflow Example



buf_ptr

str_ptr

"aaaa"

"aaaa"

"aaaa"

"aaaa"

"aaaa"

```
python –c "print 'a'* 1024" | app
```

Deserializing Data

Parsing untrusted data is a very common source of buffer overflows due to complexity.

Deserializing Data

Parsing untrusted data is a very common source of buffer overflows due to complexity.

- Realistic error triggers for arbitrary format:
  - 23rd byte = “P”, 61st is “H” or “B”, 81st =/= “Z”
  - 853 bytes total, 11th = 75th equal, 12th =/= 76th
  - 48th byte is the same as previous input’s 16th
  - (more that we haven’t found)

PEAS Description

- Fully Observable vs. Partially Observable
- Deterministic vs. Stochastic vs. Strategic
- Episodic vs. Sequential
- Static vs. Dynamic vs. Semi-Dynamic
- Discrete vs. Continuous
- Single-Agent vs. Multi-Agent
- Competitive vs. Cooperative
- Known vs. Unknown

How can we apply AI?

- What is the search space and is it infeasibly large?

- How can we conduct computational experiments to test processing?

- What can we use as our gradient to know whether “better” or “worse”?

Search Space

What is the search space and is it infeasibly large?

Search Space

What is the search space and is it infeasibly large?

all inputs that could be encountered

- Valid format data
- Valid format with 1 bit flipped
- 8,593 “a” bytes
- `cat /dev/random | app`

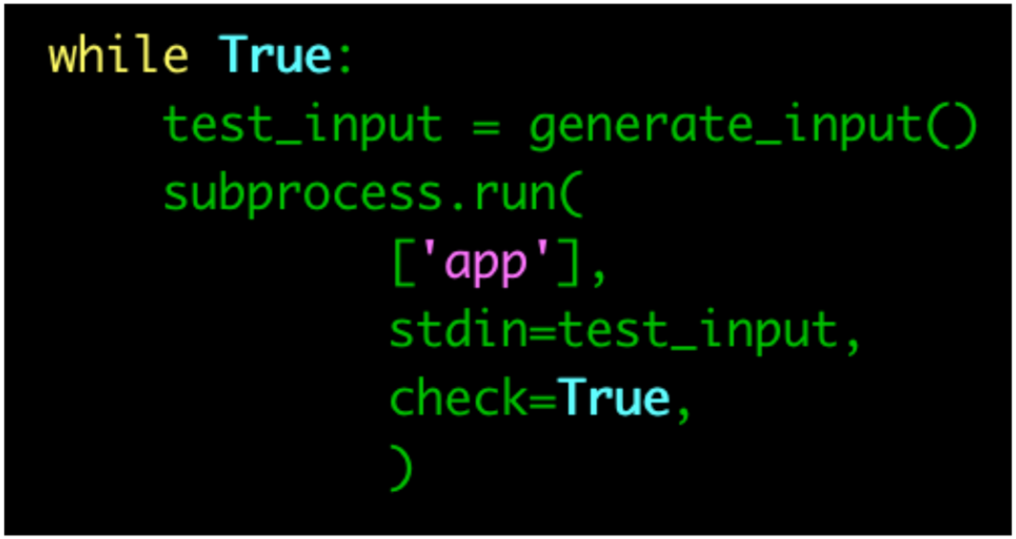
Computational Experiments

How can we conduct computational experiments to test processing?

Computational Experiments

How can we conduct computational experiments to test processing?

generate-and-test via script



```
while True:
    test_input = generate_input()
    subprocess.run(
            ['app'],
            stdin=test_input,
            check=True,
            )
```

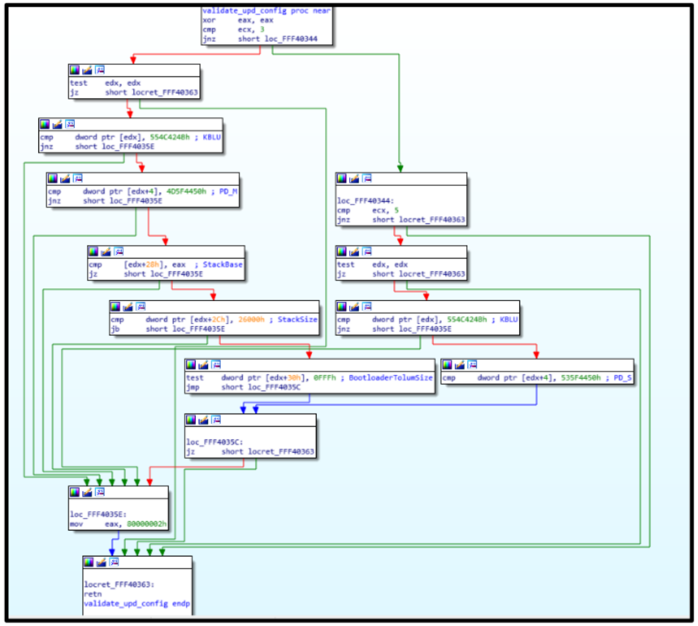
Gradient

What can we use as our gradient to know whether “better” or “worse”?

Gradient

What can we use as our gradient to know whether "better" or "worse"?

code coverage metrics



```
validate_upd_config proc near
xor     eax, eax
cmp     ecx, 3
jnz     short loc_FFF40344
```

```
test    edx, edx
jz      short locret_FFF40363
```

```
cmp     dword ptr [edx], 554C4248h ; KBLU
jnz     short loc_FFF4035E
```

```
cmp     dword ptr [edx+4], 4D5F4450h ; PD_M
jnz     short loc_FFF4035E
```

```
cmp     [edx+28h], eax  ; StackBase
jz      short loc_FFF4035E
```

```
cmp     dword ptr [edx+2Ch], 26000h ; StackSize
jb      short loc_FFF4035E
```

```
loc_FFF40344:
cmp     ecx, 5
jnz     short locret_FFF40363
```

```
test    edx, edx
jz      short locret_FFF40363
```

```
cmp     dword ptr [edx], 554C4248h ; KBLU
jnz     short loc_FFF4035E
```

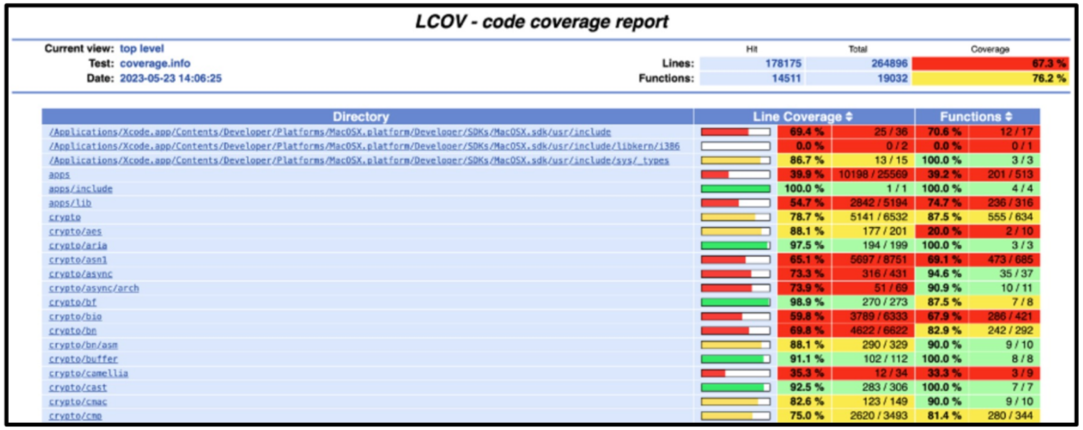
```
test    dword ptr [edx+30h], 0FFFh ; BootloaderTolumSize
jmp     short loc_FFF4035C
```

```
cmp     dword ptr [edx+4], 535F4450h ; PD_S
```

```
loc_FFF4035C:
jz      short locret_FFF40363
```

```
loc_FFF4035E:
mov     eax, 80000002h
```

```
locret_FFF40363:
retn
validate_upd_config endp
```



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	coverage.info	Lines:	178175	264896	67.3 %
Date:	2023-05-23 14:06:25	Functions:	14511	19032	76.2 %

Directory	Line Coverage ⇕		Functions ⇕	
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include	69.4 %	25 / 36	70.6 %	12 / 17
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/libkern/i386	0.0 %	0 / 2	0.0 %	0 / 1
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/sys/_types	86.7 %	13 / 15	100.0 %	3 / 3
apps	39.9 %	10198 / 25569	39.2 %	201 / 513
apps/include	100.0 %	1 / 1	100.0 %	4 / 4
apps/lib	54.7 %	2842 / 5194	74.7 %	236 / 316
crypto	78.7 %	5141 / 6532	87.5 %	555 / 634
crypto/aes	88.1 %	177 / 201	20.0 %	2 / 10
crypto/aria	97.5 %	194 / 199	100.0 %	3 / 3
crypto/asn1	65.1 %	5697 / 8751	69.1 %	473 / 685
crypto/async	73.3 %	316 / 431	94.6 %	35 / 37
crypto/async/arch	73.9 %	51 / 69	90.9 %	10 / 11
crypto/bf	98.9 %	270 / 273	87.5 %	7 / 8
crypto/bio	59.8 %	3789 / 6333	67.9 %	286 / 421
crypto/bn	69.8 %	4622 / 6622	82.9 %	242 / 292
crypto/bn/asm	88.1 %	290 / 329	90.0 %	9 / 10
crypto/buffer	91.1 %	102 / 112	100.0 %	8 / 8
crypto/camellia	35.3 %	12 / 34	33.3 %	3 / 9
crypto/cast	92.5 %	283 / 306	100.0 %	7 / 7
crypto/cmac	82.6 %	123 / 149	90.0 %	9 / 10
crypto/cmp	75.0 %	2620 / 3493	81.4 %	280 / 344