University of BRISTOL

Introduction to Fuzzing

Dr. Sana Belguith

Sana.belguith@bristol.ac.uk

bristol.ac.uk

About Fuzzing

- No, it is not about Fuzzy logic
- Neither about fuzzy set membership

Security Software Testing

Memory Corruption Bugs

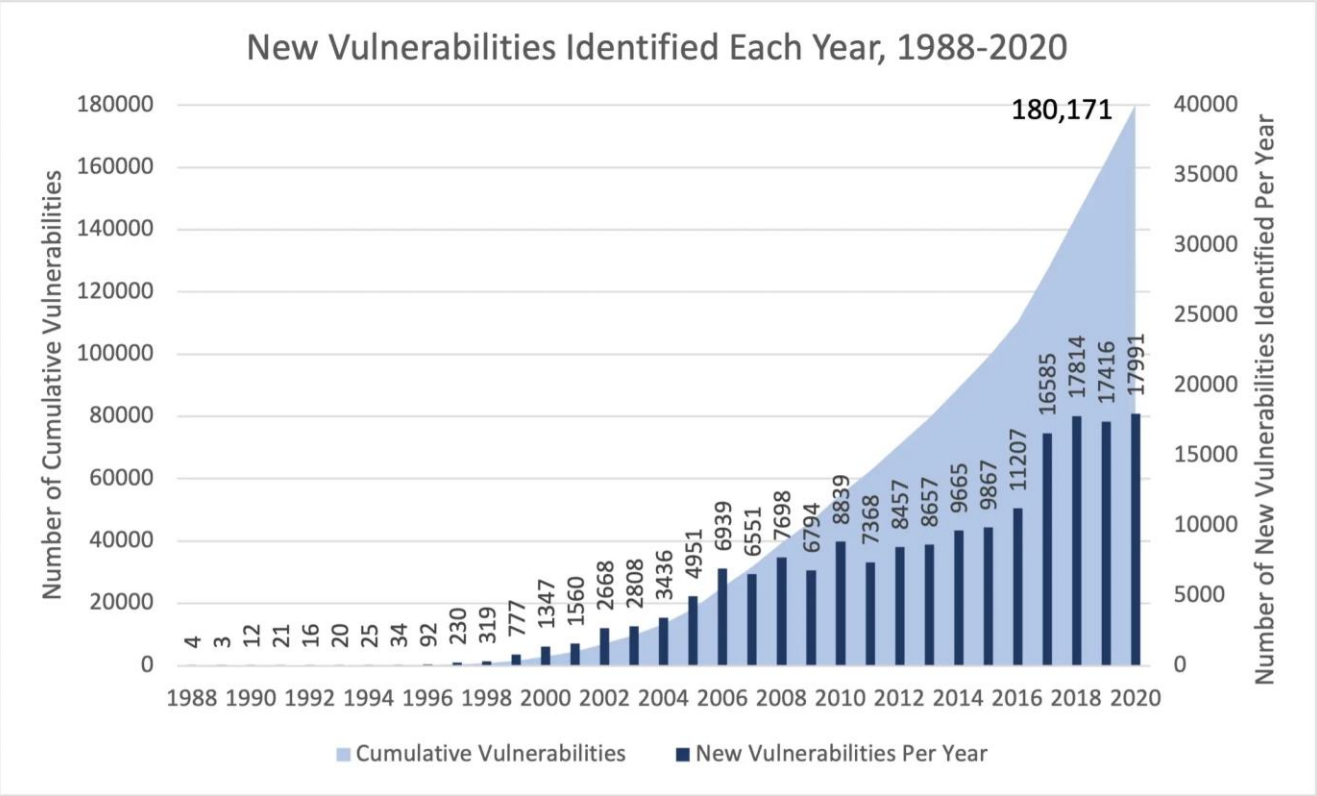
Exploitable

References:

1. book (Chapter 1, section 1.3): *Fuzzing for Software Security Testing and Quality Assurance.* Ari Takanen, Jared DeMott, Charlie Miller

2. Article *"Fuzzing: Hack, Art, and Science"* By P. Godefroid

Why do we care?



**https://securityintelligence.com/

The Plan

- Memory corruption vulnerabilities
- Fuzzing- finding vulnerabilities
- Types of Fuzzing
- Some existing solutions

Memory Corruption Vulnerabilities

- WYSINWYX: What You See Is Not What You eXecute by G. Balakrishnan et. al.
  - Higher level code -> low-level representation
  - Seemingly separate variables -> contiguous memory addresses

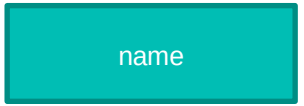
- Contiguous memory locations allow for boundary violations!

Example

```
#include <stdio.h>
int get_cookie(){
  return rand();}
int main(){
   int cookie;  char
   name[40];
   cookie = get_cookie();
   gets(name);
   if (cookie == 0x41424344)
      printf("You win %s\n!", name);
   else printf("better luck next time
   :(");  return 0;

}
```

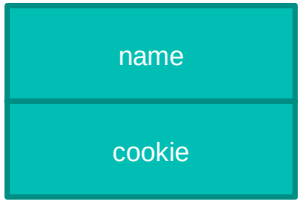
Example

```
#include <stdio.h>
int get_cookie(){
  return rand();}
int main(){
    int cookie;
    char name[40];
    cookie = get_cookie();
    gets(name);
    if (cookie == 0x41424344)
        printf("You win %s\n!", name);
    else printf("better luck next time :(");
    return 0;

}
```



Example

```
#include <stdio.h>
int get_cookie(){
  return rand();}
int main(){
    int cookie;
    char name[40];
    cookie = get_cookie();
    gets(name);
    if (cookie == 0x41424344)
        printf("You win %s\n!", name);
    else printf("better luck next time :(");
    return 0;

}
```



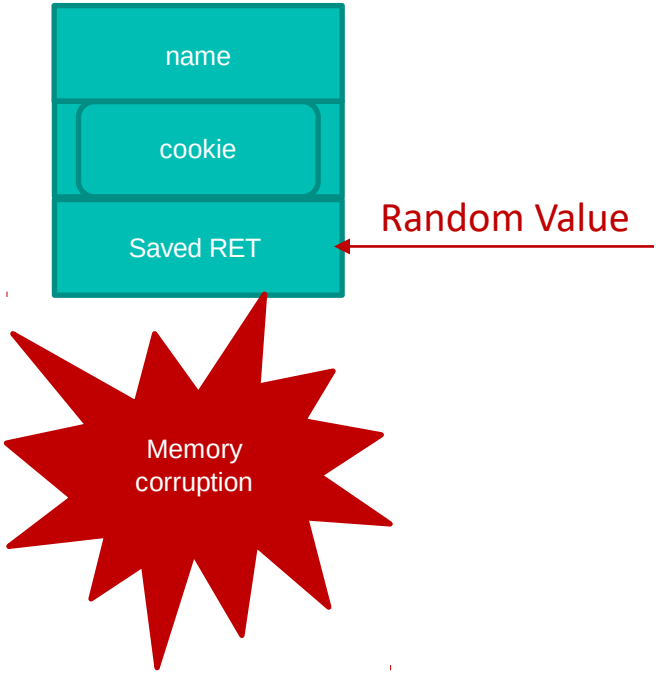
Example

```
#include <stdio.h>
int get_cookie(){
  return rand();}
int main(){
    int cookie;
    char name[40];
    cookie = get_cookie();
    gets(name);
    if (cookie == 0x41424344)
        printf("You win %s\n!", name);
    else printf("better luck next time :(");
    return 0;

}
```

Buffer Overflow!

Example

```
#include <stdio.h>
int get_cookie(){
  return rand();}
int main(){
    int cookie;
    char name[40];
    cookie = get_cookie();
    gets(name);
    if (cookie == 0x41424344)
        printf("You win %s\n!", name);
    else printf("better luck next time :(");
    return 0;

}
```



Side effects

- Over/underflow
- Sensitive data corruption
- Control data corruption (control hijacking)

Side effects

- Over/underflow
- Sensitive data corruption
- Control data corruption (control hijacking)

If exploit is done properly, Otherwise crash!

Fuzzing

- *It started on a dark and stormy night…. [Barton P. Miller, late1980s]*

https://pages.cs.wisc.edu/~bart/fuzz/Foreword1.html

Fuzzing

- Security testing technique
- Run program on many **abnormal/malformed** inputs, look for **unintended** behavior, e.g. crash.
  - An observable (measurable) side effect is essential
  - Should be scalable

- Underlying assumption: *if the unintended behavior is dependent on input, an attacker can craft such an input to exploit the bug.*

Types of Fuzzing

- Input based: mutational and Generative (grammar based)
- Application based: black-box and white-box
- Input Strategy: memory-less and evolutionary

Input Generation

- Mutation Based: mutate seed inputs to create new test inputs:
  Simple strategy is to randomly choose an offset and change the byte.

Pros: easy to implement and low overhead

Cons: highly structured inputs will become invalid quickly → low coverage.

Input Generation

- Generation (Grammar) Based:
  - learn/create the format/model of the input
  - generate new inputs based on the learned model

  e.g. well-known file formats (jpeg, xml, etc.)

Pros: Highly effective for complex structured input parsing applications → high coverage

Cons: expensive as models are not easy to learn or obtain.

JPEG file format

JFIF file structure		
Segment	**Code**	**Description**
SOI	`FF D8`	Start of Image
JFIF-APP0	`FF E0` *s1 s2* `4A 46 49 46 00 ...`	see below
JFXX-APP0	`FF E0` *s1 s2* `4A 46 58 58 00 ...`	optional, see below
... additional marker segments (for example SOF, DHT, COM)		
SOS	`FF DA`	Start of Scan
	compressed image data	
EOI	`FF D9`	End of Image

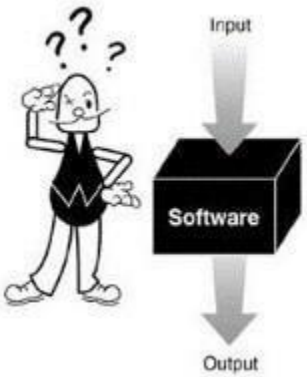
JPEG file format

JFIF APP0 marker segment		
Field	**Size (bytes)**	**Description**
APP0 marker	2	`FF E0`
Length	2	Length of segment excluding APP0 marker
Identifier	5	`4A 46 49 46 00` = "JFIF" in ASCII, terminated by a null byte
JFIF version	2	First byte for major version, second byte for minor version (`01 02` for 1.02)
Density units	1	Units for the following pixel density fields • `00` : No units; width:height pixel aspect ratio = Ydensity:Xdensity • `01` : Pixels per inch (2.54 cm) • `02` : Pixels per centimeter
Xdensity	2	Horizontal pixel density. Must not be zero
Ydensity	2	Vertical pixel density. Must not be zero
Xthumbnail	1	Horizontal pixel count of the following embedded RGB thumbnail. May be zero
Ythumbnail	1	Vertical pixel count of the following embedded RGB thumbnail. May be zero

JFIF file structure		
Segment	**Code**	**Description**
SOI	`FF D8`	Start of Image
JFIF-APP0	`FF E0` *s1 s2* `4A 46 49 46 00 ...`	see below
JFXX-APP0	`FF E0` *s1 s2* `4A 46 58 58 00 ...`	optional, see below
... additional marker segments (for example SOF, DHT, COM)		
SOS	`FF DA`	Start of Scan
	compressed image data	
EOI	`FF D9`	End of Image

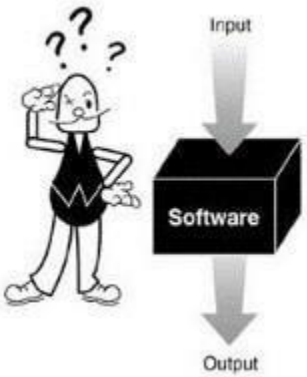
Application Monitoring

- Blackbox: Only interface is known.

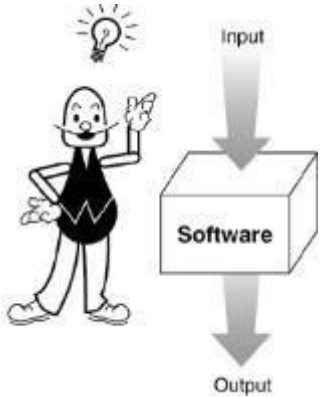


Application Monitoring

- Blackbox: Only interface is known.



- Whitebox: Application can be analysed/monitored.



Problem with Traditional Fuzzing

Blackbox + mutation: Aiming with luck!

```
... //JPEG parsing

read(fd, buf, size);

if (buf[1] == 0xD8 && buf[0] == 0xFF)
  // interesting code here

else
  pr_exit("Invalid file");
```

Problem with Traditional Fuzzing

- Apply more heuristics to:
  - Mutate better
  - Learn good inputs
- Apply more analysis (static/dynamic) to understand the application behavior.

➢ But remember the scalability factor!

Problem with ~~Traditional~~ Smart Fuzzing

smart fuzzing: Aiming with educated guess!

Evolutionary Fuzzing

- Rather than throwing inputs, *evolve them.*

- *Underlying assumption:*

  Inputs are parsed enough before going further deep in execution

Issues ?

Evolutionary Fuzzing

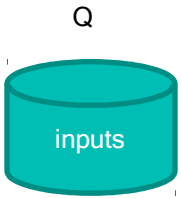
- What should be the feedback to evolve?
  - Code-coverage based fuzzing
    - Most of the contemporary fuzzers: AFL, AFLFast, Driller, VUzzer, ProbeFuzzer, CollAFL, Angora, QSYM, Nautilus, …
    - Uses code-coverage as the proxy metric for the effectiveness of a fuzzer
  - Directed fuzzing
    - Not much explored (BuzzFuzz, AFLGo, ParmeSan…)
    - There should be a way to find the destination and a sense of direction.

Evolving A Fuzzer

Lets start with something we are more familiar with- AFL

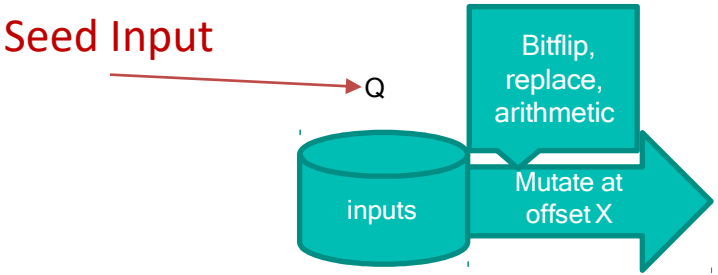
Open source from Google
We will use in next lab

Evolving A Fuzzer



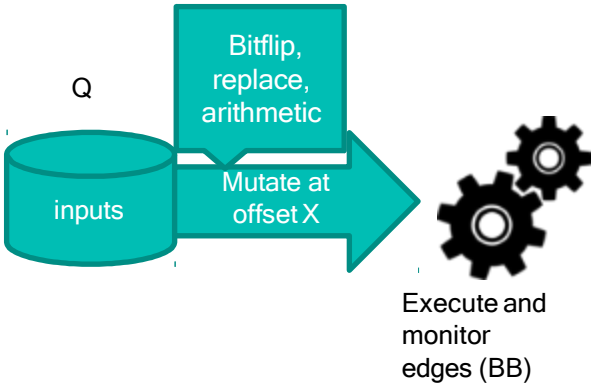
Evolving A Fuzzer

- Lets start with something we are more familiar with- AFL



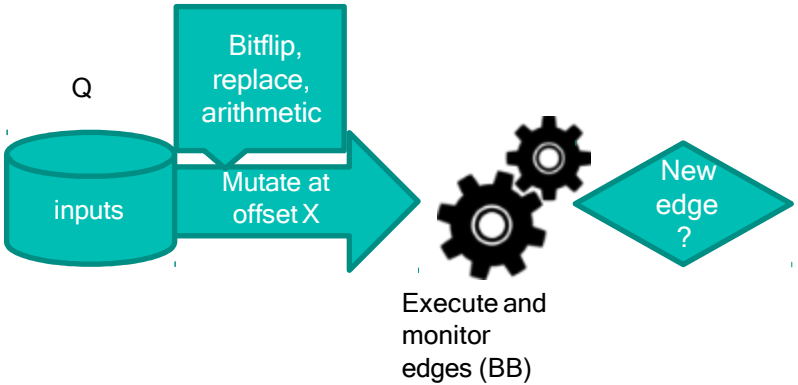
Evolving A Fuzzer

- Lets start with something we are more familiar with- AFL



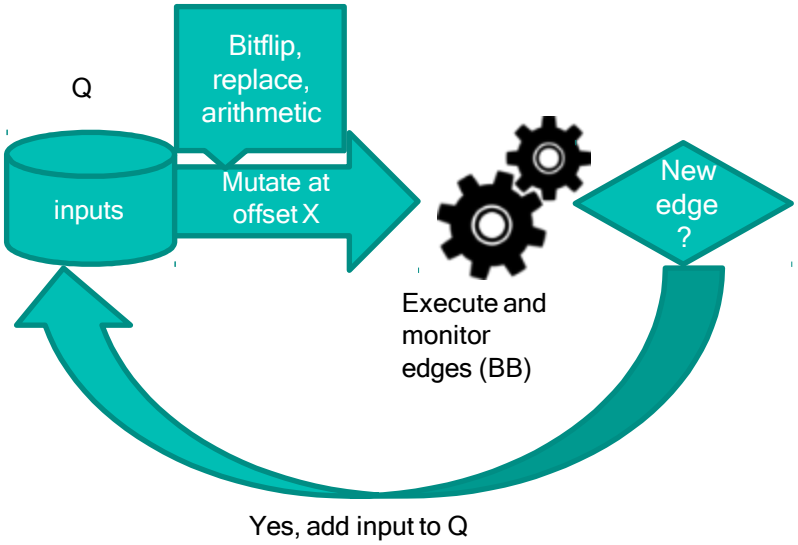
Evolving A Fuzzer

- Lets start with something we are more familiar with- AFL



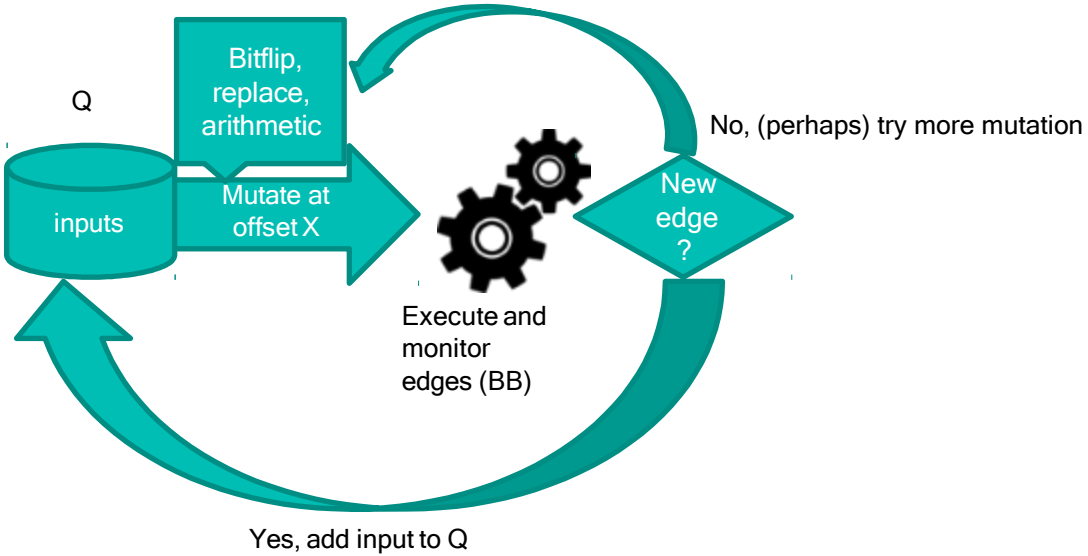
Evolving A Fuzzer

- Lets start with something we are more familiar with- AFL



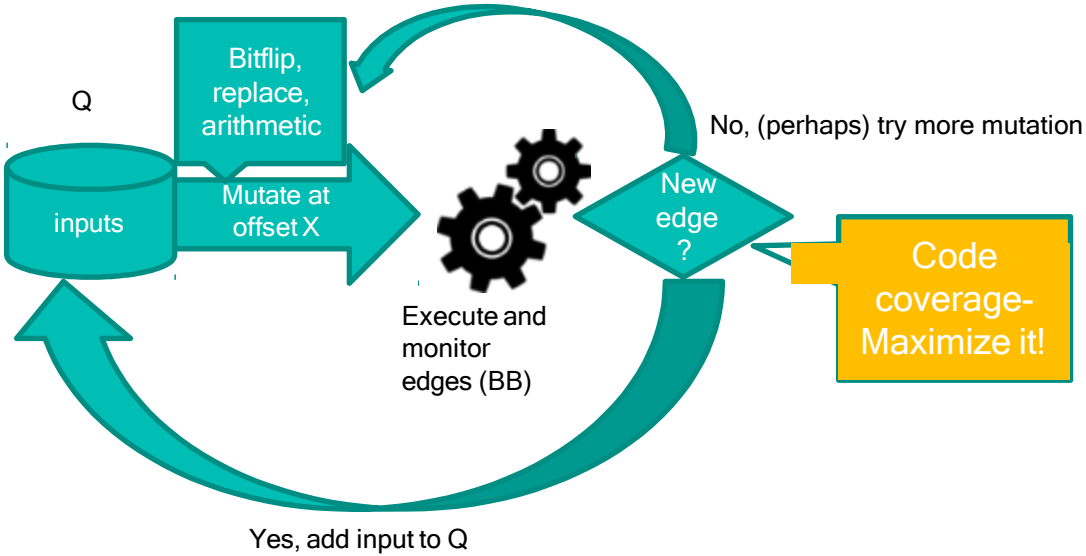
Evolving A Fuzzer

- Lets start with something we are more familiar with- AFL



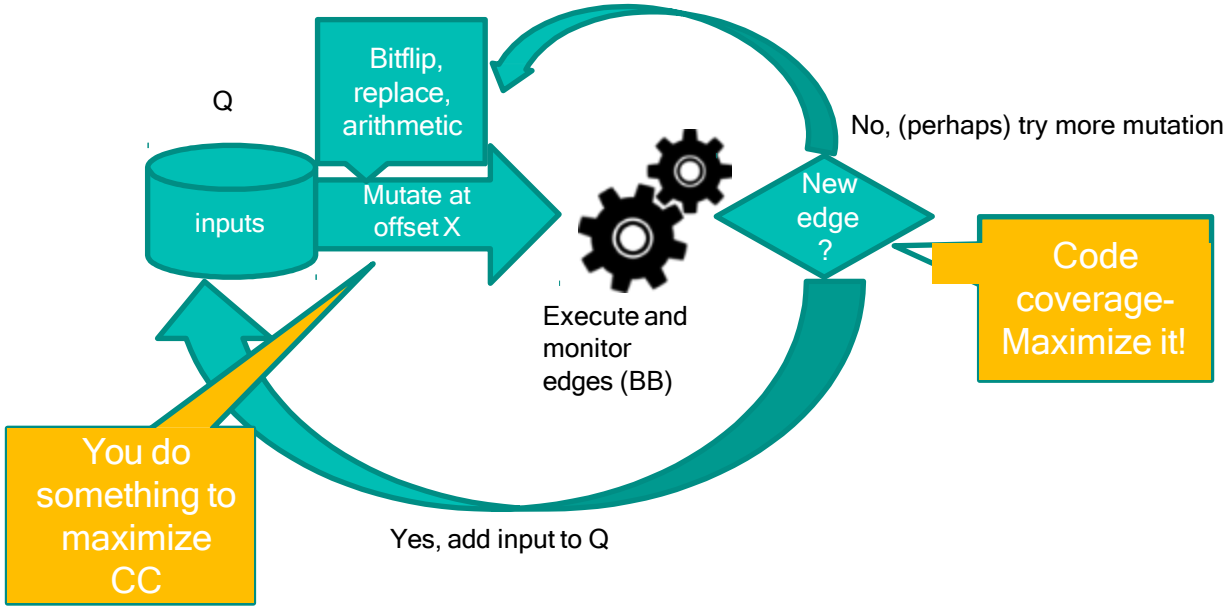
Evolving A Fuzzer

- Lets start with something we are more familiar with- AFL

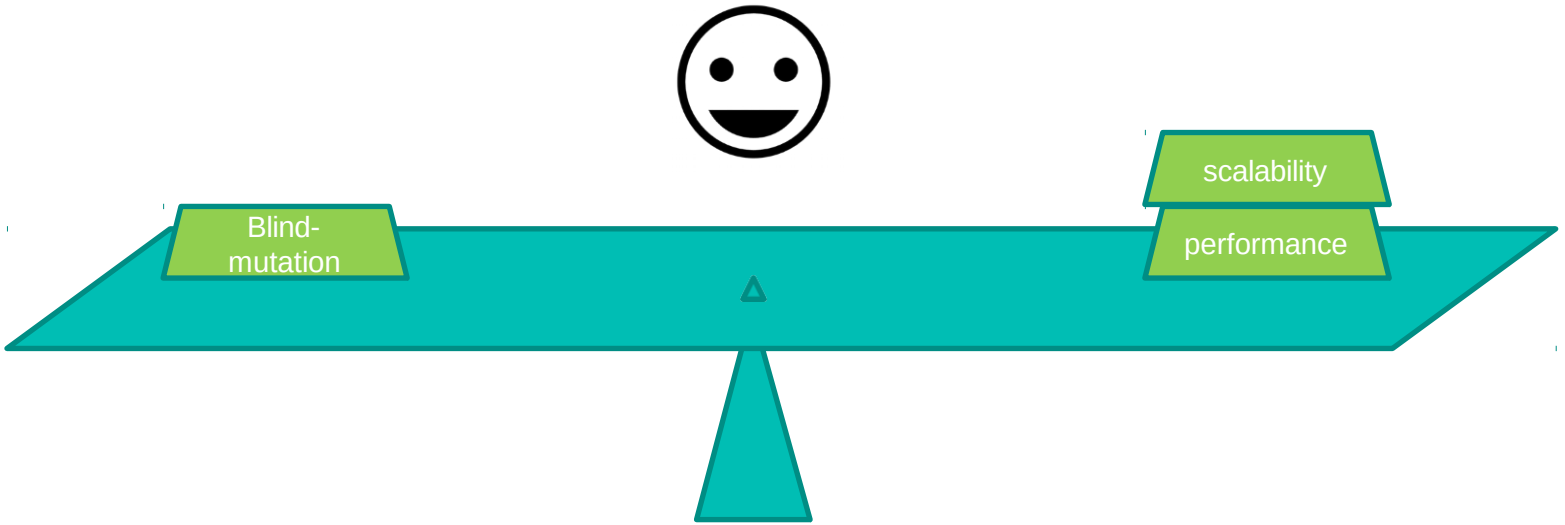


Evolving A Fuzzer

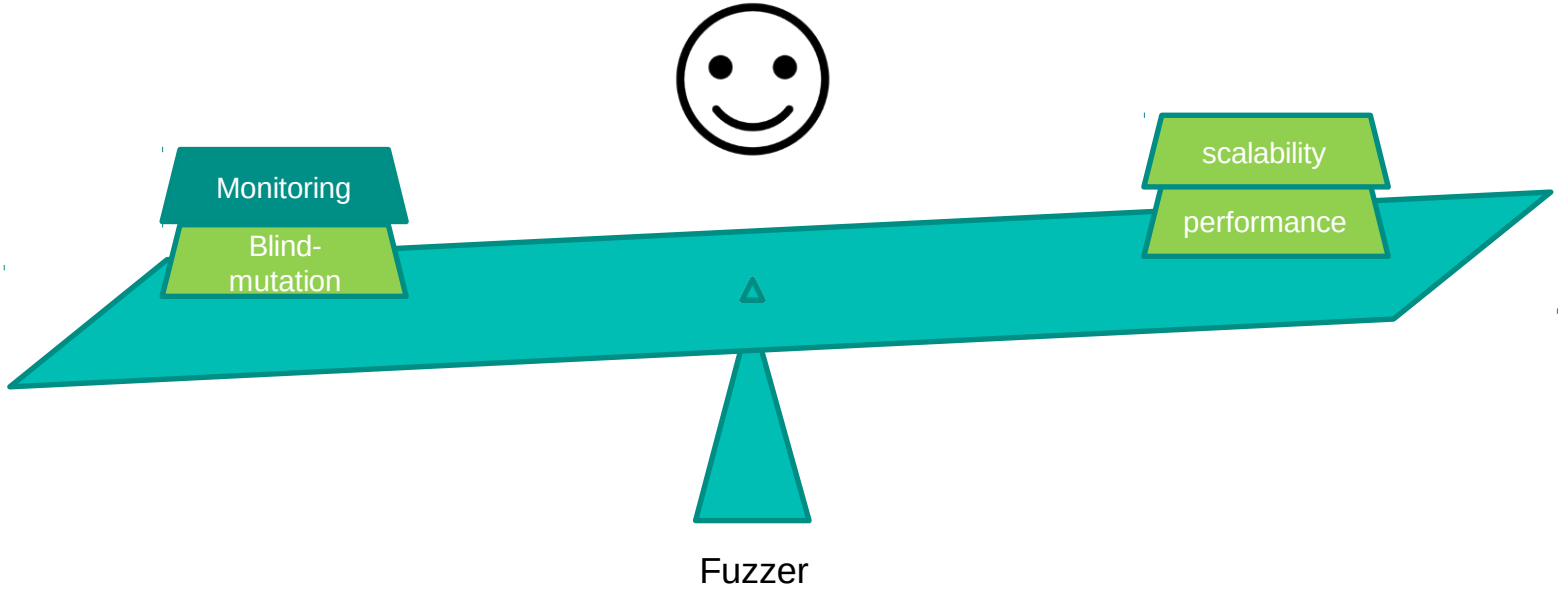
- Lets start with something we are more familiar with- AFL



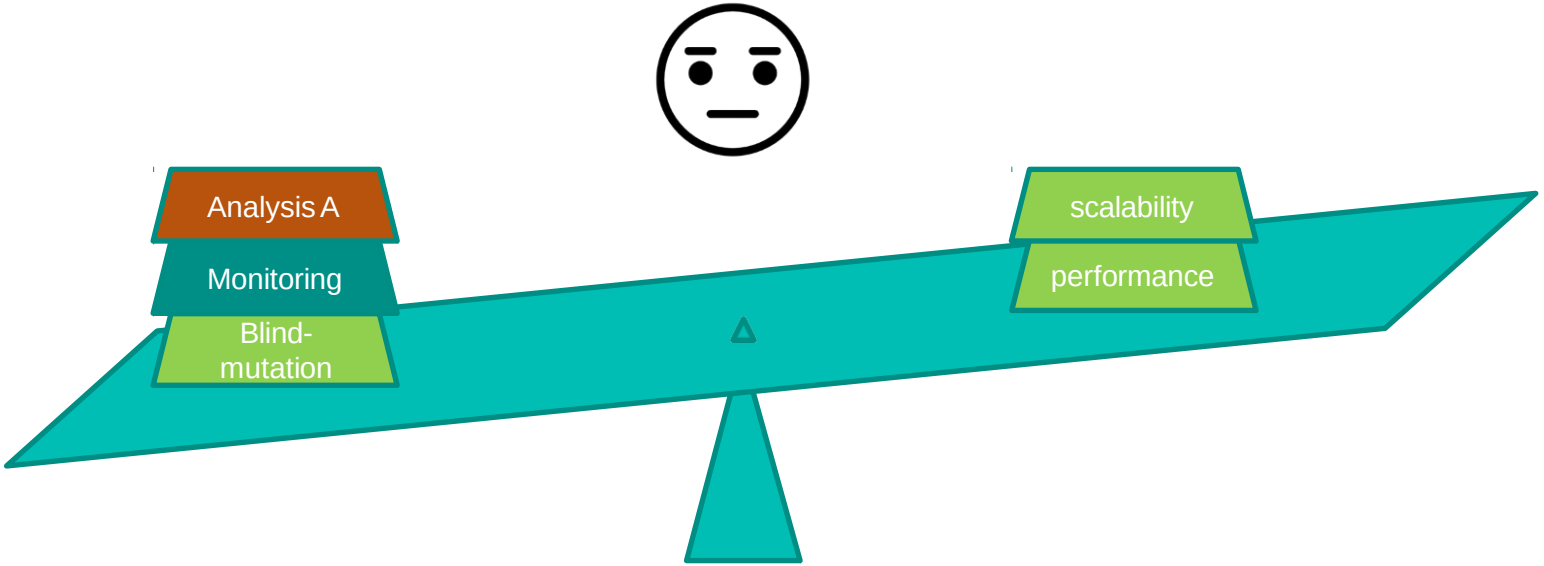
Fuzzing- A balancing Act



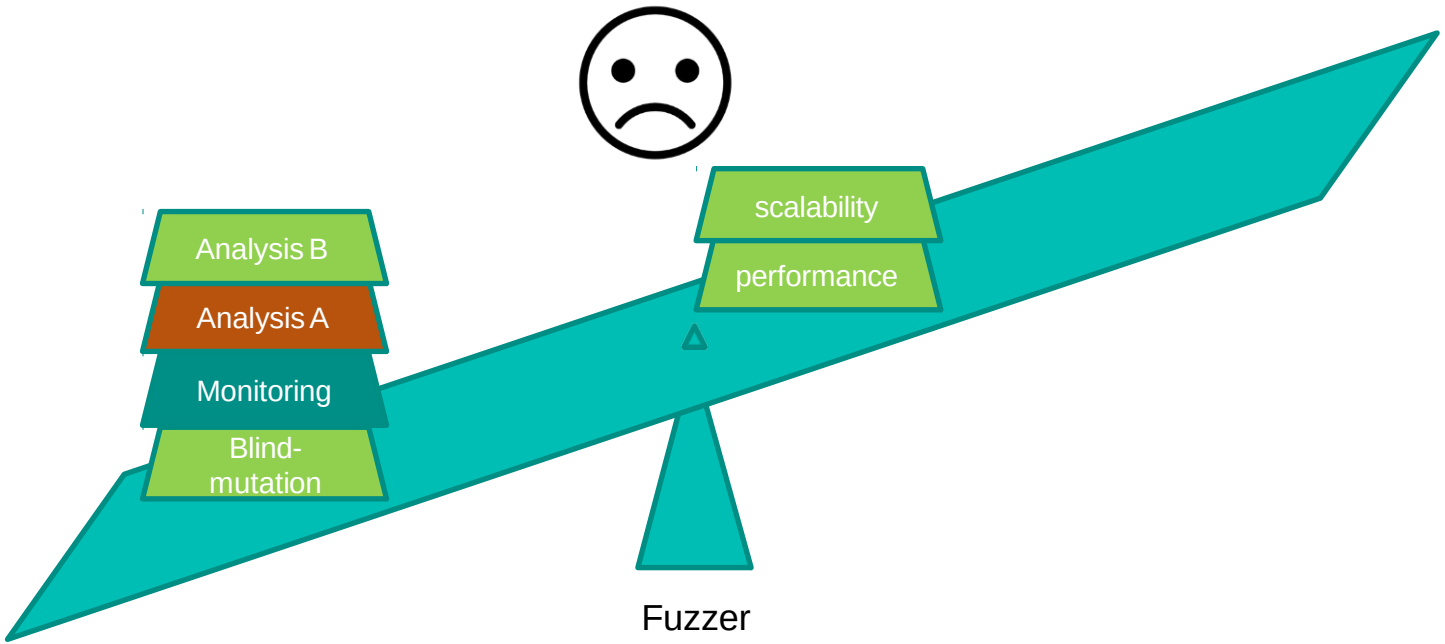
Fuzzing- A balancing Act



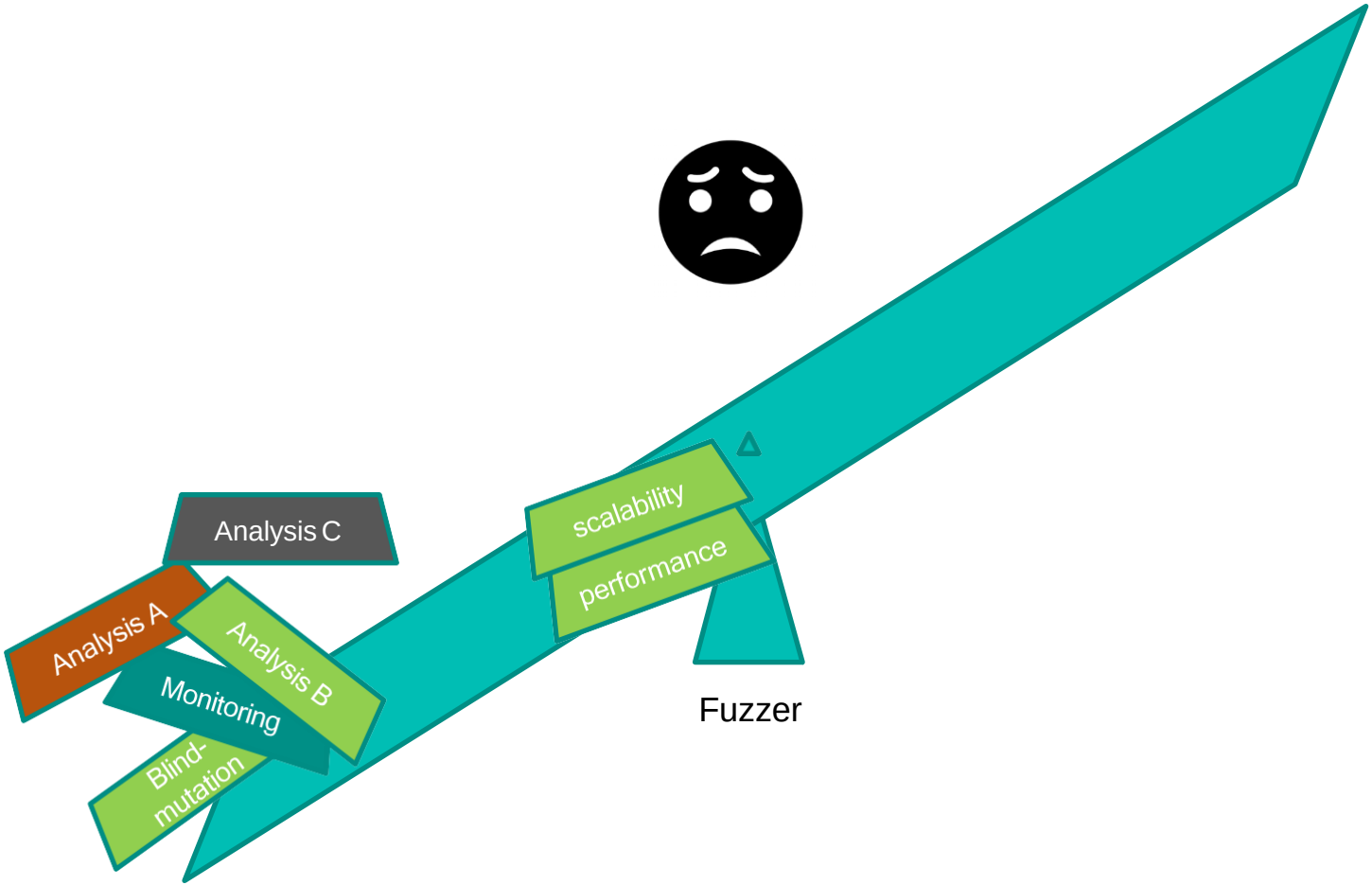
Fuzzing- A balancing Act



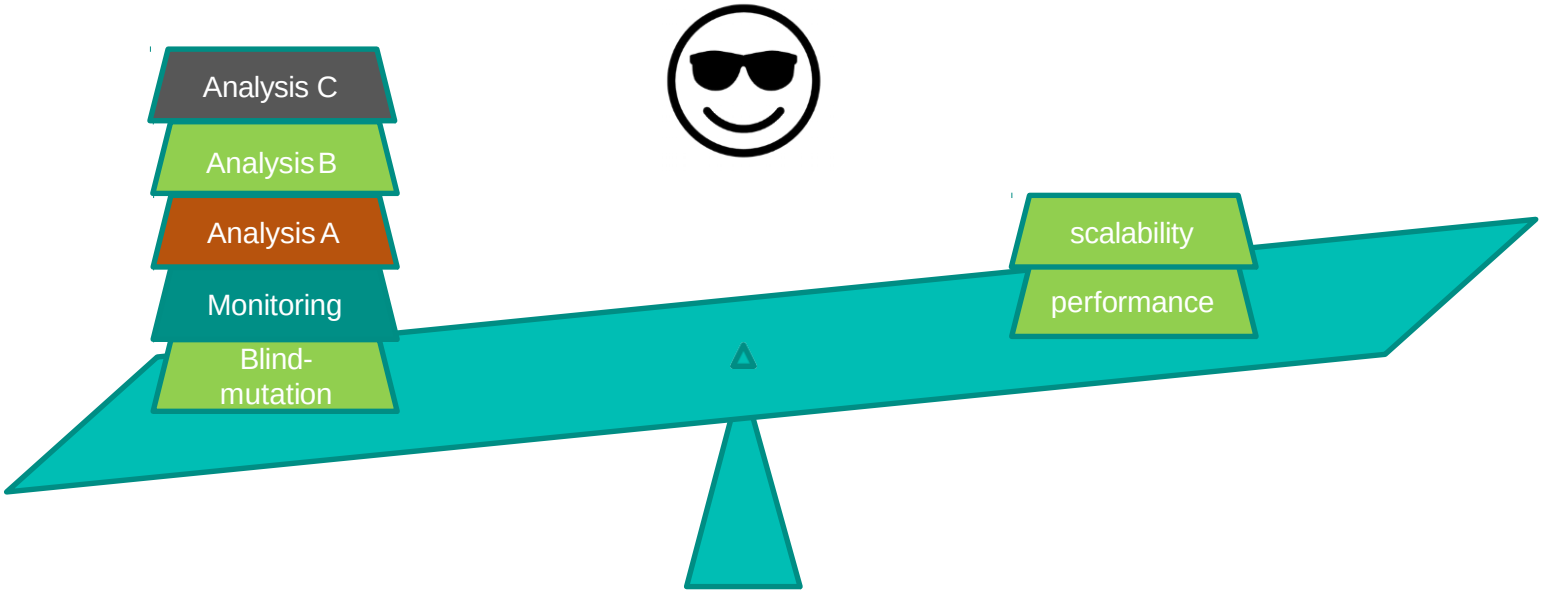
Fuzzing- A balancing Act



Fuzzing- A balancing Act



Fuzzing- A balancing Act



Happy Advanced Fuzzer

Conclusions

- Fuzzing – seems easy unless you try it!
- Scalability and performance cannot be negotiated much!
  - A good engineering, hardware assisted monitoring
- A good place to try program analysis techniques
  - Possibility to compromise correctness to make them scalable
- Software will remain integral part of the cyber world- make is secure!