

Introduction to Fuzzing

Sanjay Rawat

sanjay.rawat@Bristol.ac.uk

About the technique

- It is about Fuzzing

References:

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

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

About the technique

- It is about Fuzzing



References:

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

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

About the technique



- It is about Fuzzing
- No, it is not about Fuzzy logic

References:

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

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

About the technique



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

References:

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

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

About the technique



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

Software Testing

References:

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

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

About the technique



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

Security Software Testing

References:

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

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

About the technique



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

Security Software Testing Memory-corruption bugs

References:

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

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

About the technique



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

Security Software Testing
Memory-corruption bugs
Exploitable!

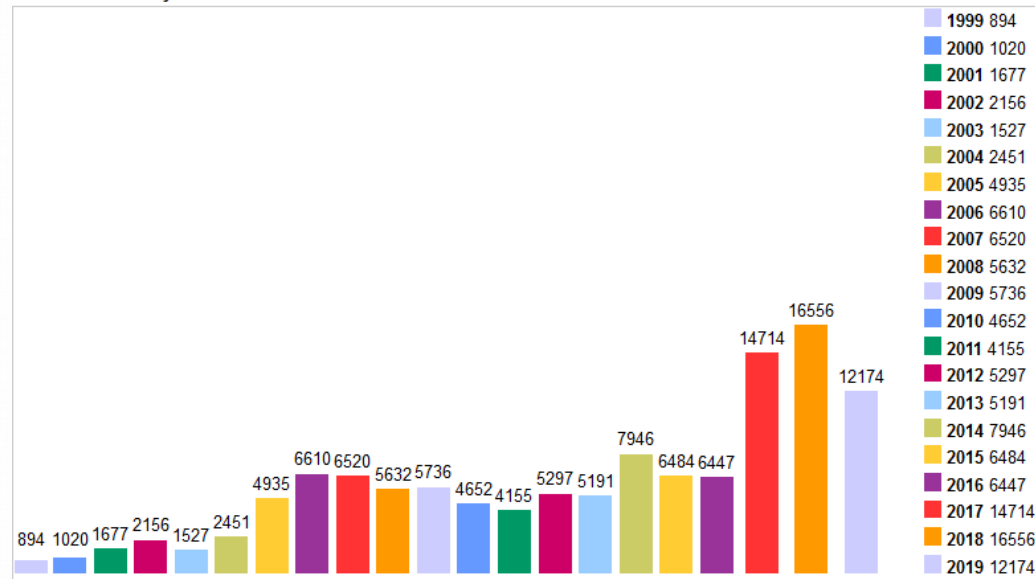
References:

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

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

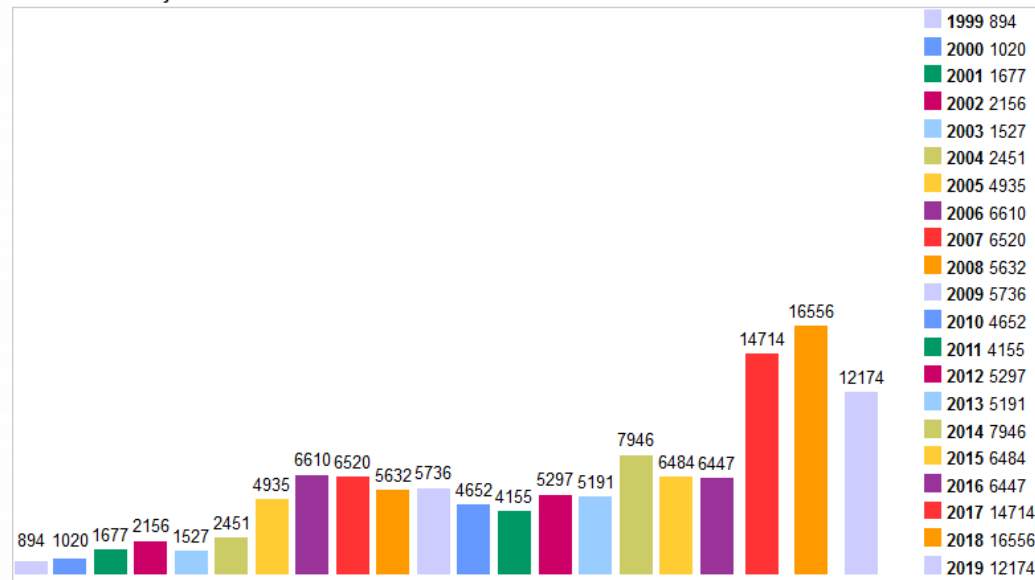
Why do we care?

Vulnerabilities By Year



Why do we care?

Vulnerabilities By Year



**<http://www.cvedetails.com>

Organization

- 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

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;
}
```



name

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;
}
```

name
cookie

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;
}
```

name
cookie
Saved RET

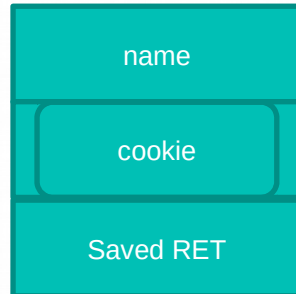
example

```
#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;
}
```



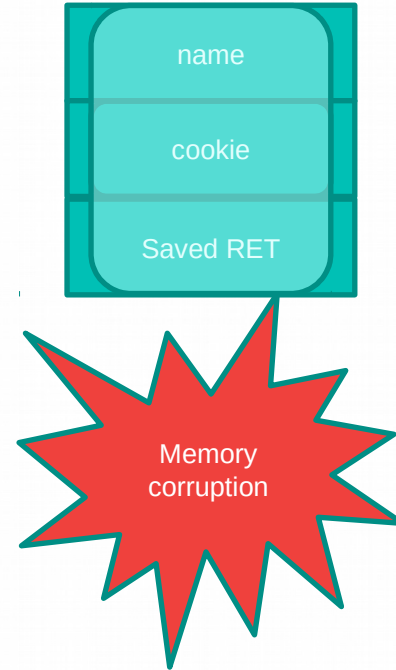
example

```
#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;
}
```



Side effects

Side effects

- Over/underflow

Side effects

- Over/underflow
- Sensitive data corruption

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 done properly-exploit

Side effects

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

If done properly-exploit

Otherwise crash!

Fuzzing

Fuzzing

- *It started on a dark and stormy night....* [Barton P. Miller, late 1980s]



Fuzzing

Fuzzing

- Run program on many **abnormal/malformed** inputs, look for **unintended** behavior, e.g. crash.

Fuzzing

- Run program on many **abnormal/malformed** inputs, look for **unintended** behavior, e.g. crash.



- An observable (measurable) side effect is essential

Fuzzing

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



Fuzzing

- 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

Input Generation

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

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.

Cont..

- Generation (Grammar) Based: Learn/create the format/model of the input and based on the learned model, generate new inputs.
 - 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 <ul style="list-style-type: none">00 : No units; width:height pixel aspect ratio = Ydensity:Xdensity01 : 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

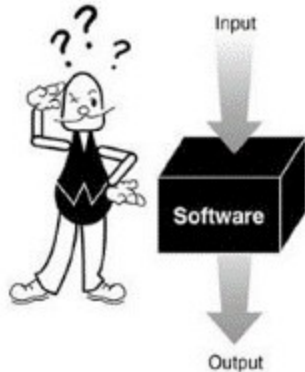
JFIF file structure

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

Application Monitoring

- Blackbox: Only interface is known.



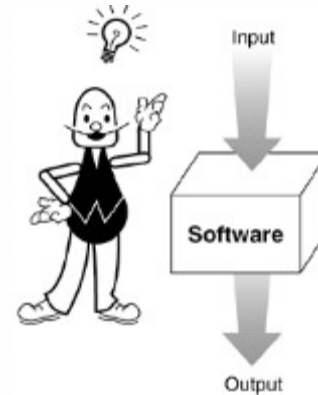
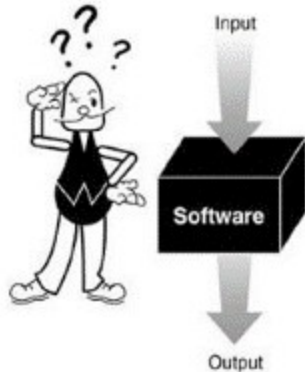
Application Monitoring

- Blackbox: Only interface is known.



Application Monitoring

- Blackbox: Only interface is known.
- Whitebox: Application can be analysed/monitored.
Static & Dynamic analysis



Problem with Traditional Fuzzing

Problem with Traditional Fuzzing

Blackbox + mutation: Aiming with luck!

Problem with Traditional Fuzzing

Blackbox + mutation: Aiming with luck!



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

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

Problem with Traditional Fuzzing

- Apply more heuristics to:
 - Mutate better
 - Learn good inputs

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

Problem with ~~Traditional~~ Smart Fuzzing

smart fuzzing: Aiming with educated guess!

Problem with ~~Traditional~~ Smart Fuzzing

smart fuzzing: Aiming with educated guess!



Evolutionary Fuzzing

Evolutionary Fuzzing

- Recall: memory-less and Evolutionary fuzzing

Evolutionary Fuzzing

- Recall: memory-less and Evolutionary fuzzing
- Rather than throwing inputs, *evolve them*.

Evolutionary Fuzzing

- Recall: memory-less and Evolutionary fuzzing
- Rather than throwing inputs, *evolve them*.
- *Underlying assumption:*
 - *Inputs are parsed enough before going further deep in execution*

Evolutionary Fuzzing

- Recall: memory-less and Evolutionary fuzzing
- Rather than throwing inputs, *evolve them*.
- *Underlying assumption:*
 - *Inputs are parsed enough before going further deep in execution*

But, there are issues

Evolutionary Fuzzing

- What should be the feedback to evolve?
 - Code-coverage based fuzzing
 - Most of the contemporary fuzzers are here (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, AGLGo, ...)
 - There should be a way to find the destination and a sense of direction.

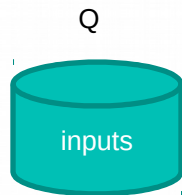
Evolving A Fuzzer

Evolving A Fuzzer

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

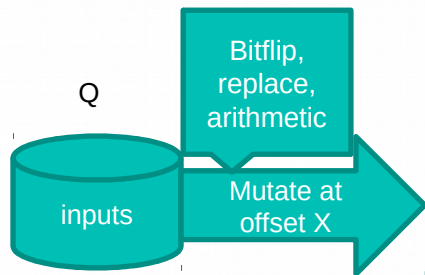
Evolving A Fuzzer

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



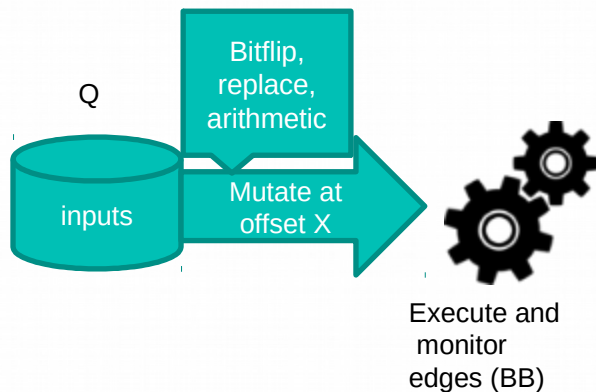
Evolving A Fuzzer

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



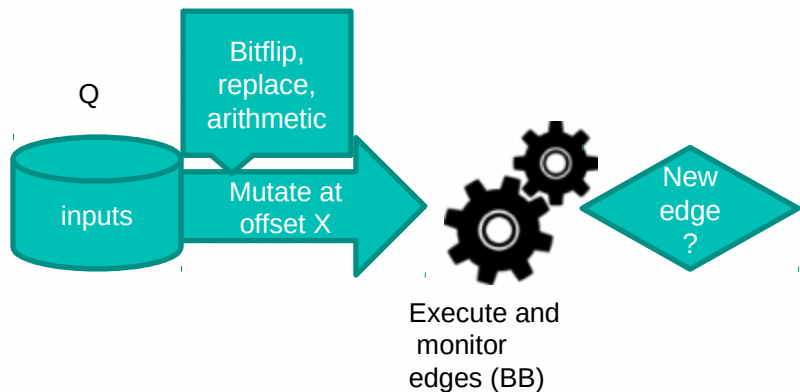
Evolving A Fuzzer

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



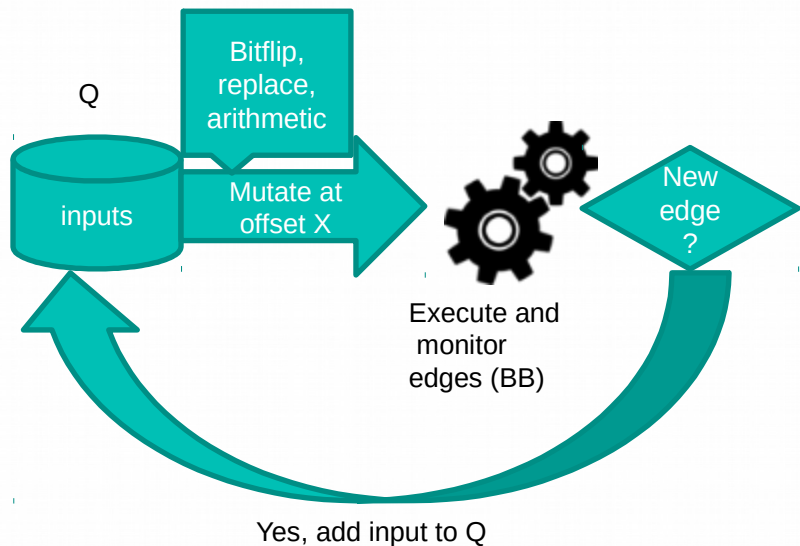
Evolving A Fuzzer

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



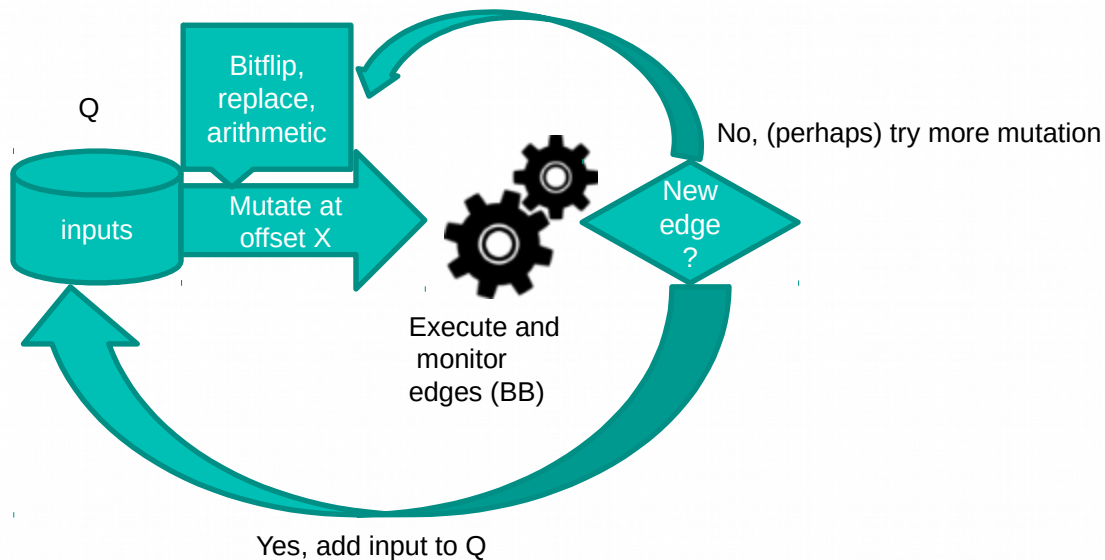
Evolving A Fuzzer

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



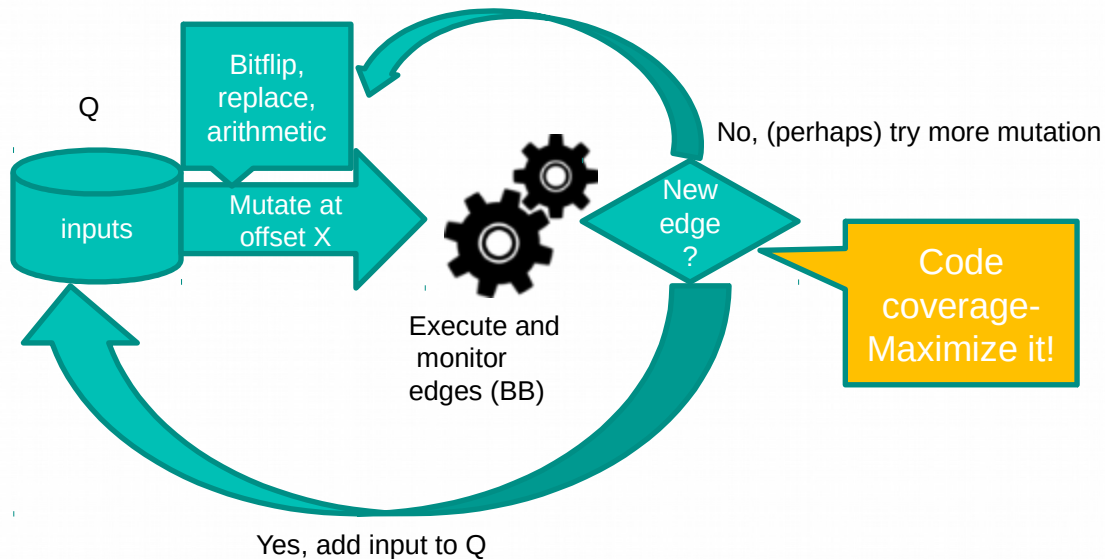
Evolving A Fuzzer

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



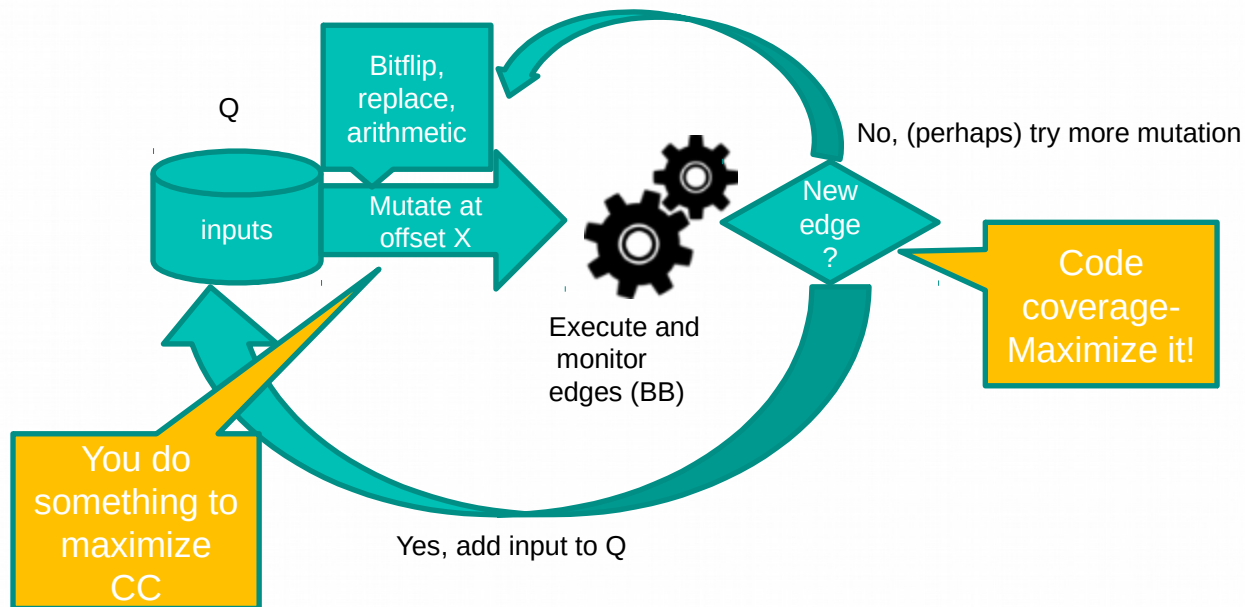
Evolving A Fuzzer

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



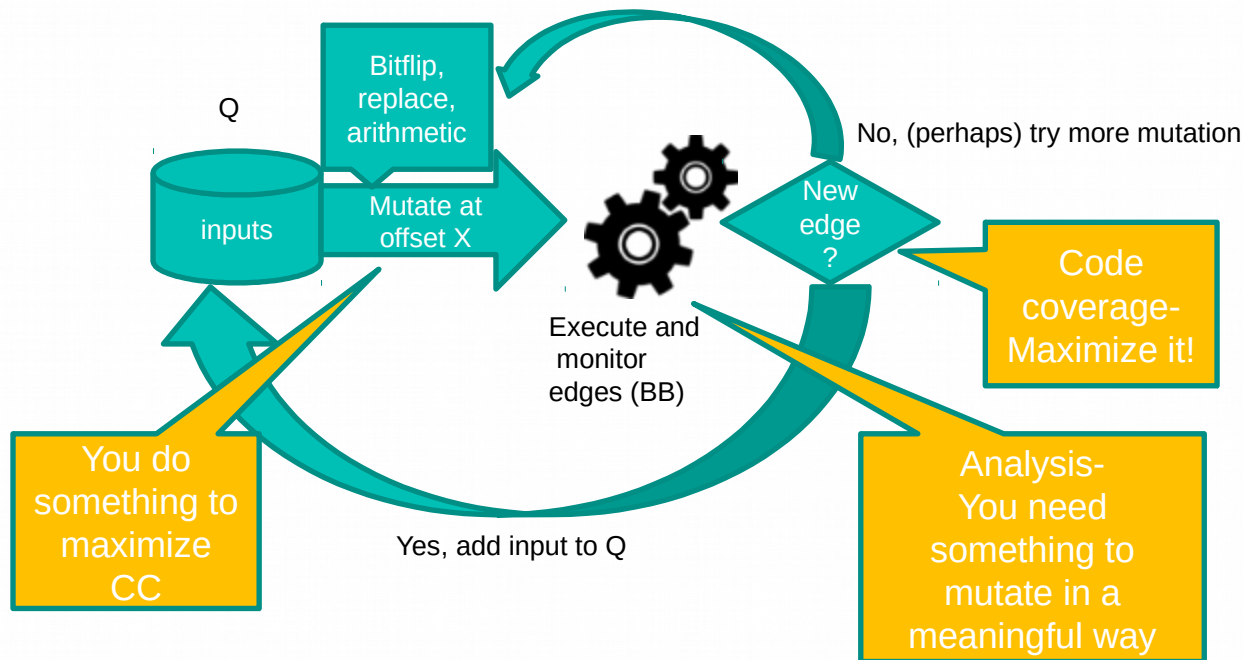
Evolving A Fuzzer

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

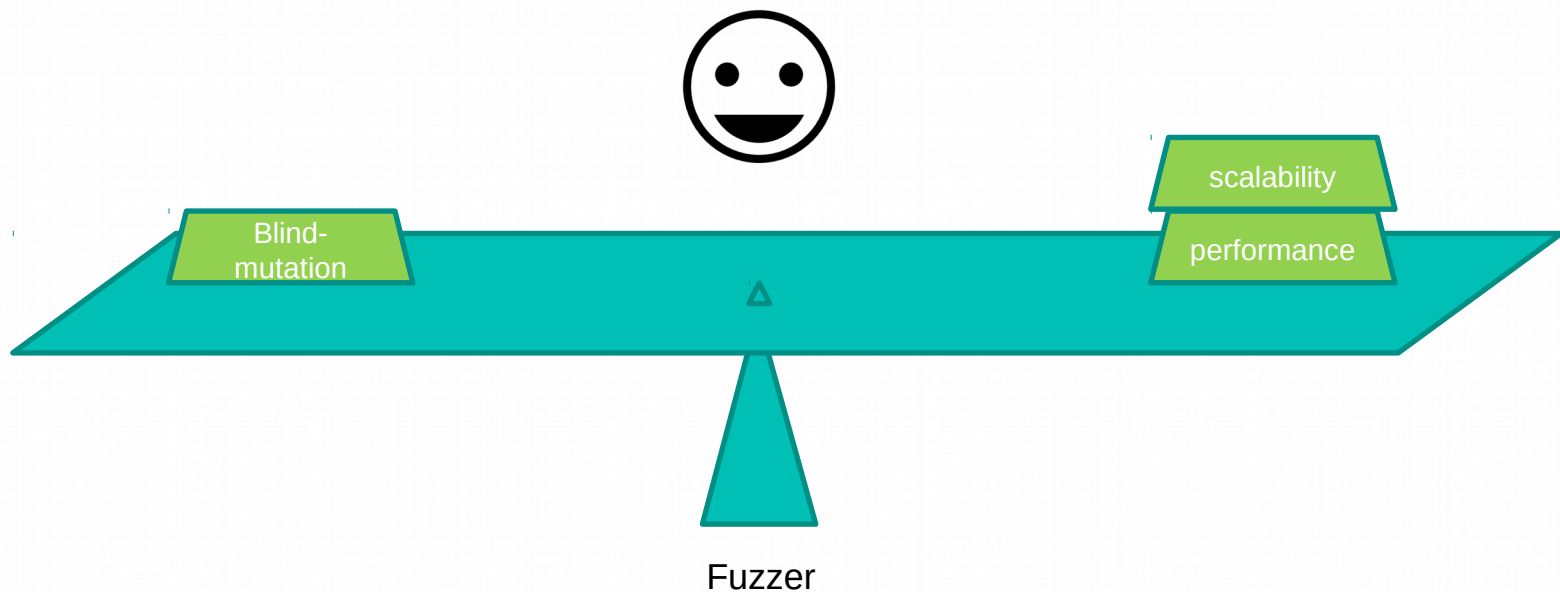


Evolving A Fuzzer

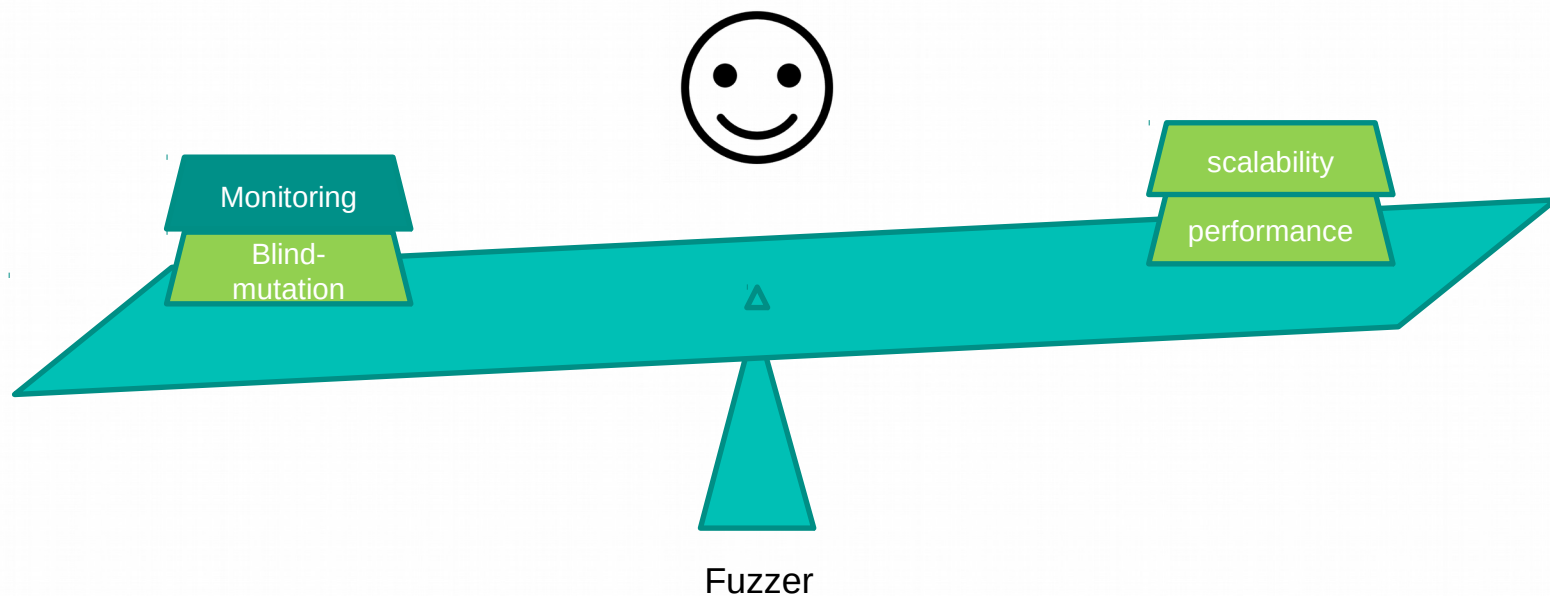
- Lets start with something we are more familiar with- AFL



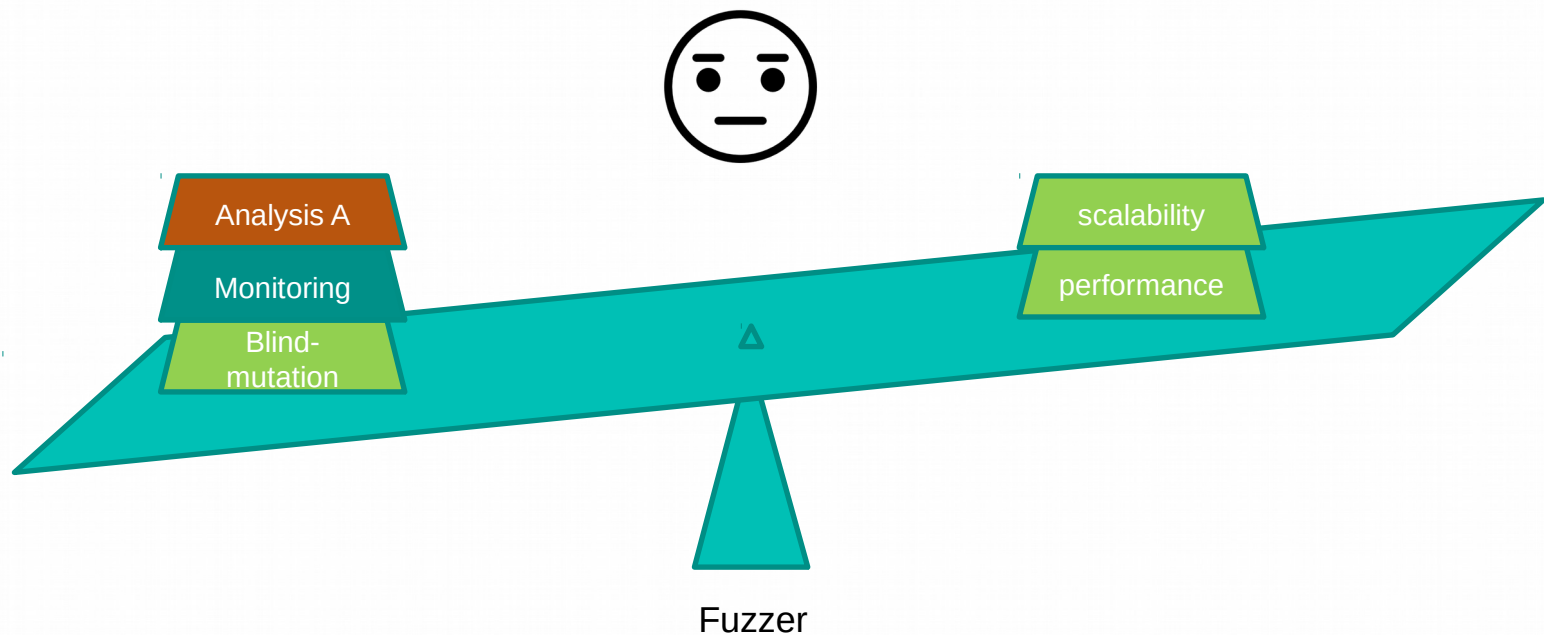
Fuzzing- A balancing Act



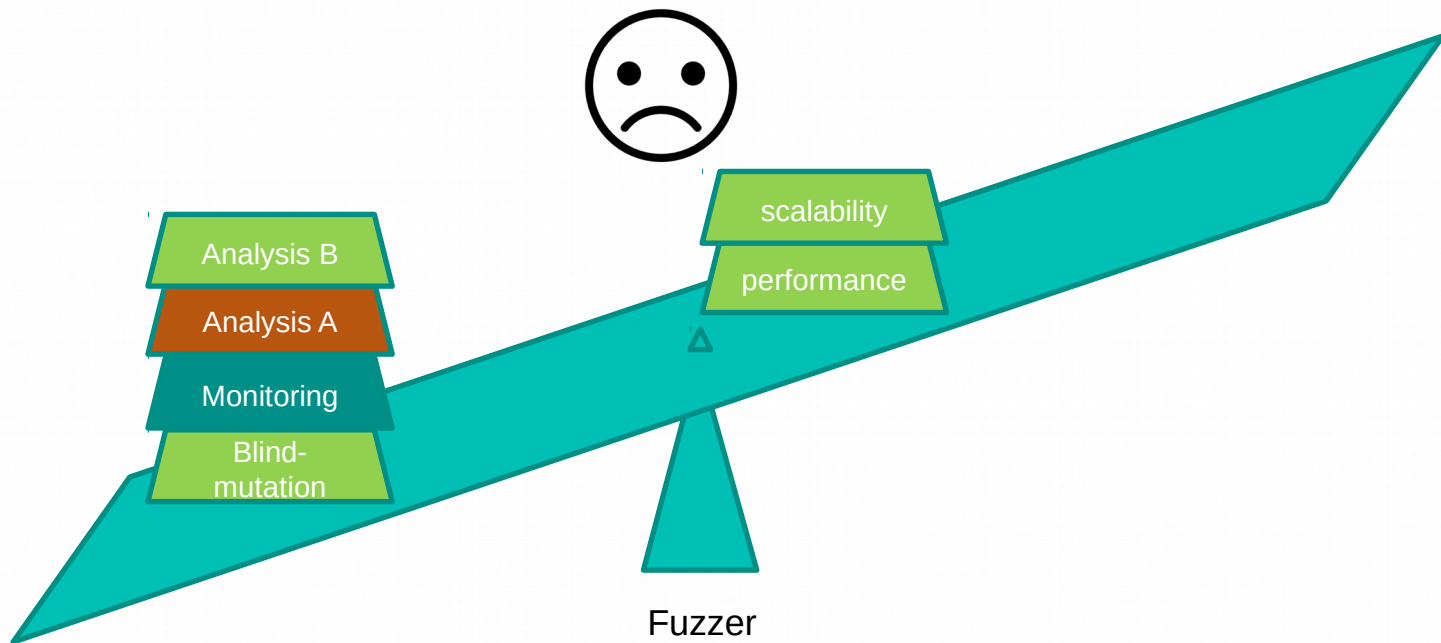
Fuzzing- A balancing Act



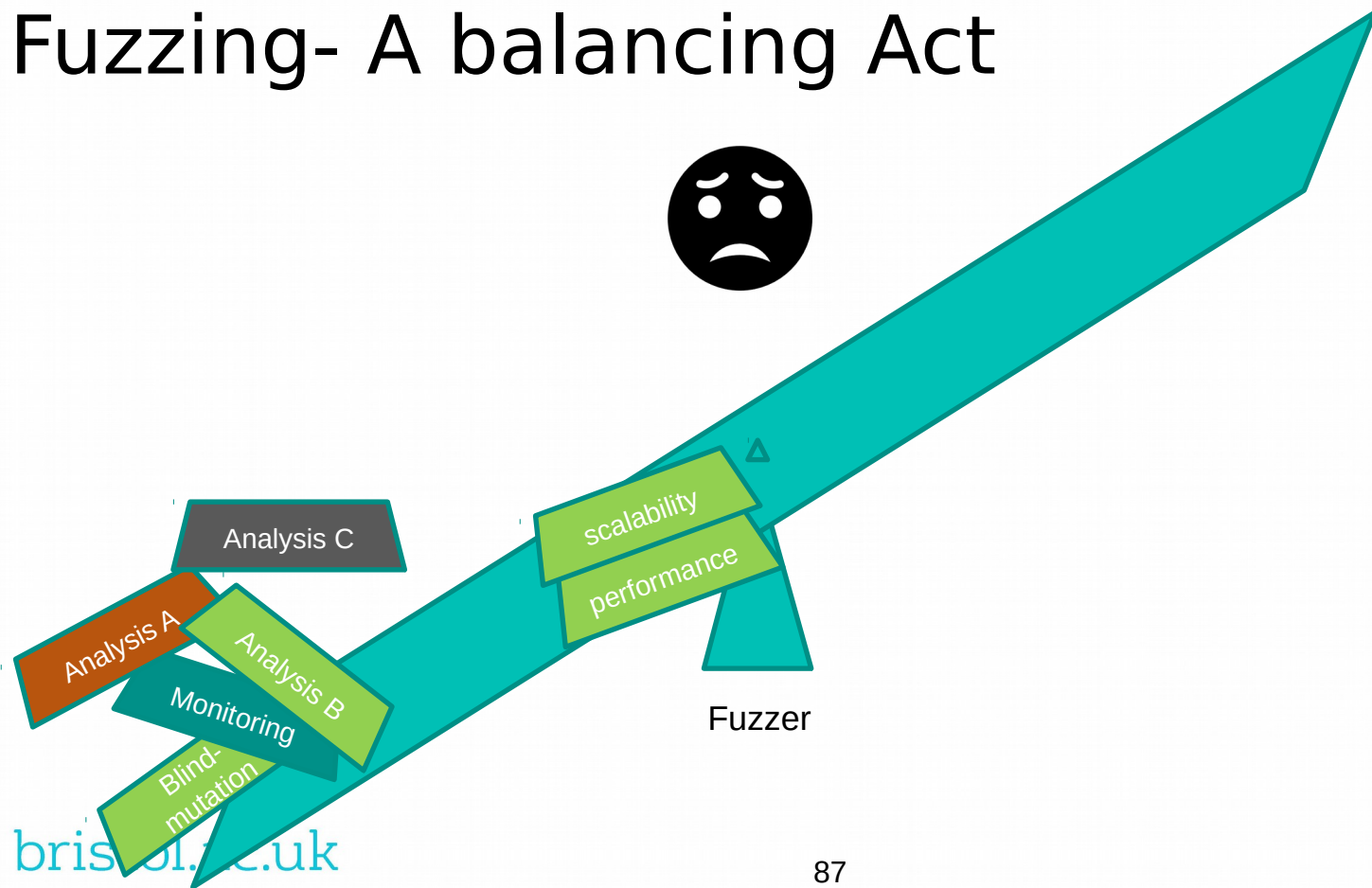
Fuzzing- A balancing Act



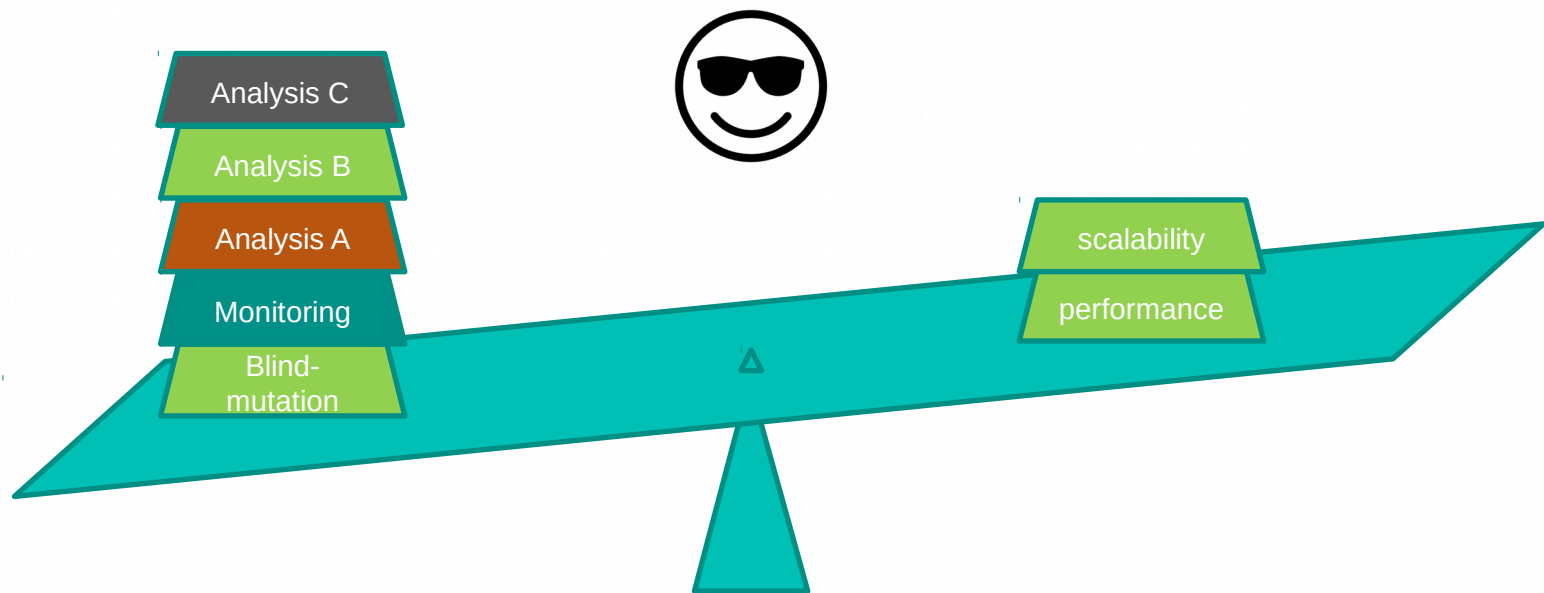
Fuzzing- A balancing Act



Fuzzing- A balancing Act

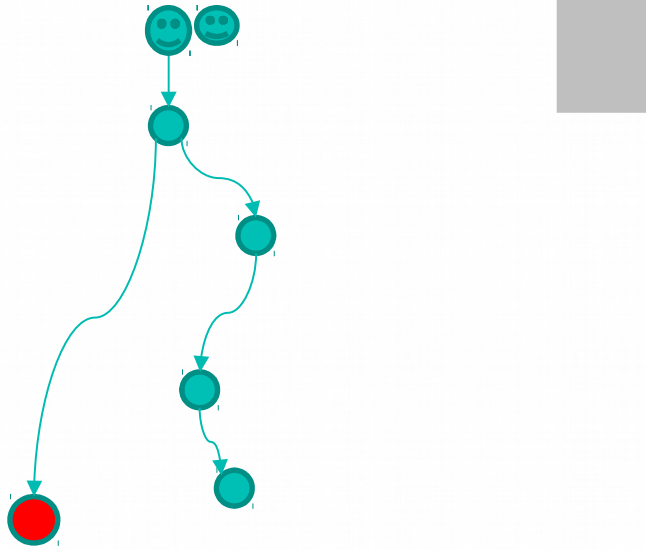


Fuzzing- A balancing Act

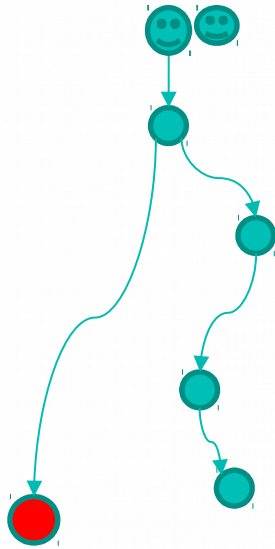


Happy Advanced Fuzzer

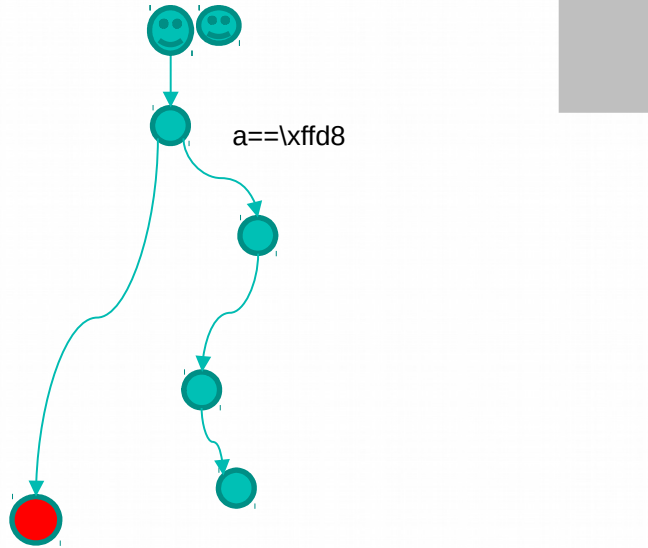
Problem Exemplified....



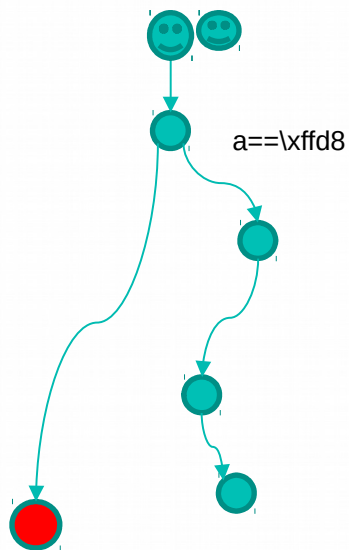
Problem Exemplified....



Problem Exemplified....

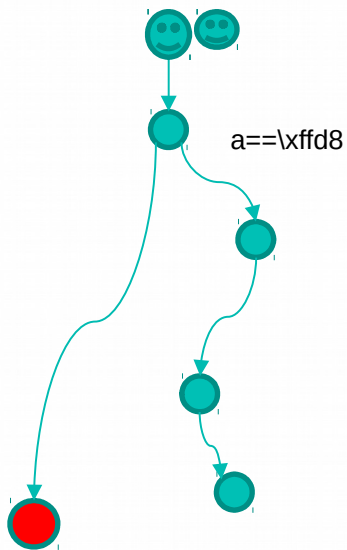


Problem Exemplified....

[illegible]

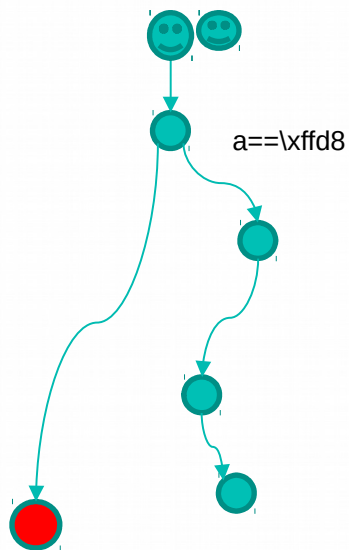
Problem Exemplified....

Where is 'a'?

[illegible]

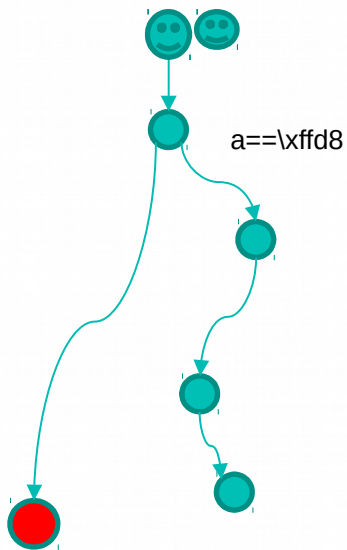
Problem Exemplified....

Where is 'a'?

[illegible]

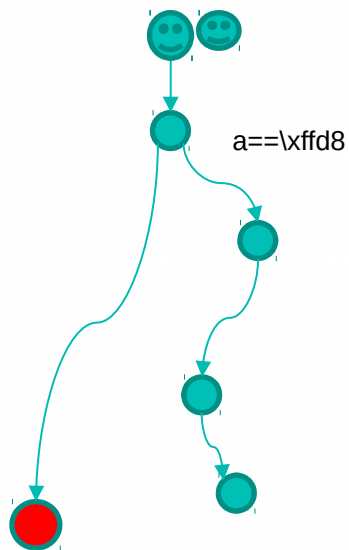
Problem Exemplified....

Where is 'a'?

[illegible]

Problem Exemplified....

Where is 'a'?

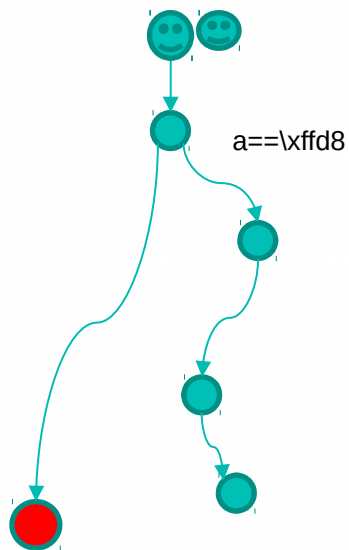


What values?

[illegible]

Problem Exemplified....

Where is 'a'?

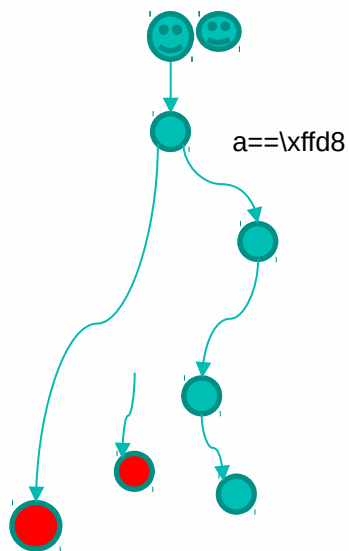


What values?

[illegible]

Problem Exemplified....

Where is 'a'?

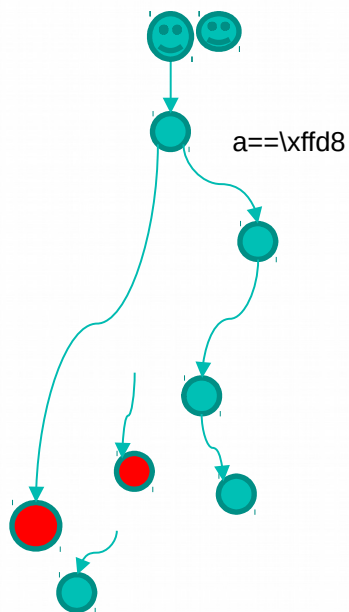


What values?

[illegible]

Problem Exemplified....

Where is 'a'?

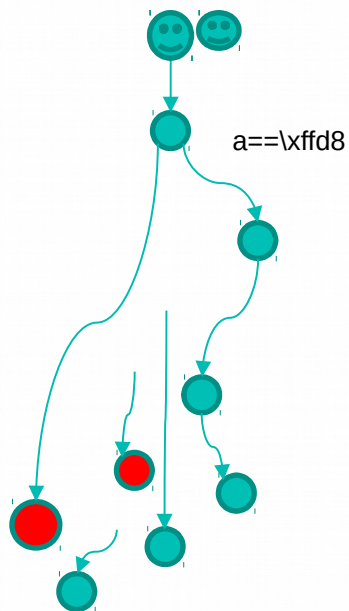


What values?

[illegible]

Problem Exemplified....

Where is 'a'?

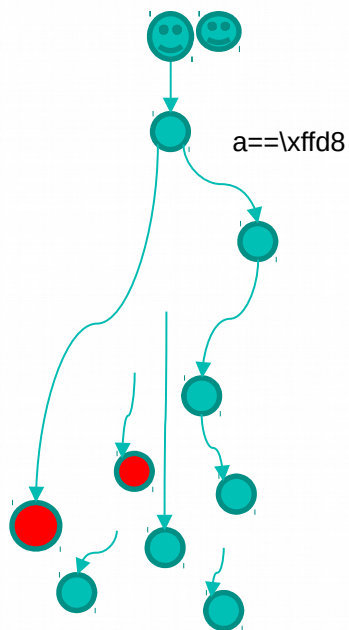


What values?

[illegible]

Problem Exemplified....

Where is 'a'?

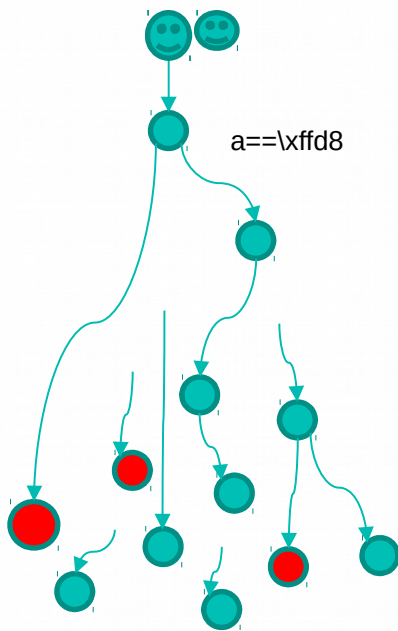


What values?

[illegible]

Problem Exemplified....

Where is 'a'?

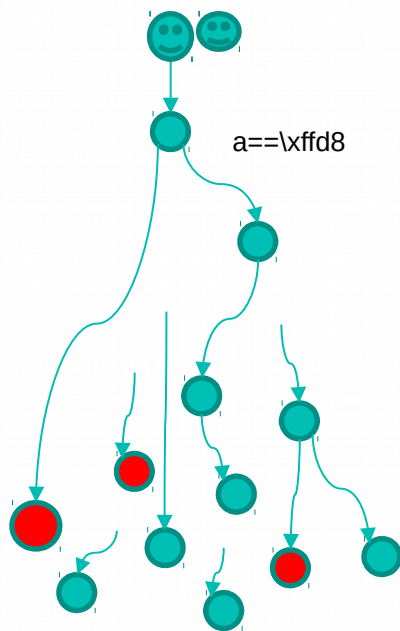


What values?

[illegible]

Problem Exemplified....

Where is 'a'?



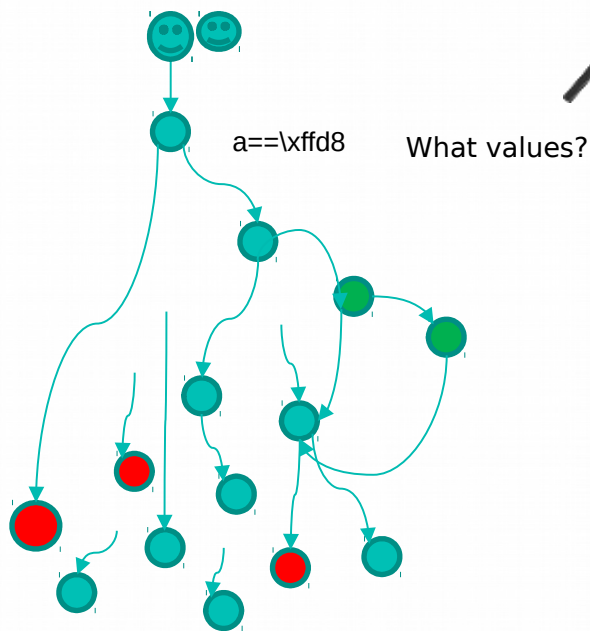
What values?

[illegible]

Easy paths (superficial paths), error code

Problem Exemplified....

Where is 'a'?

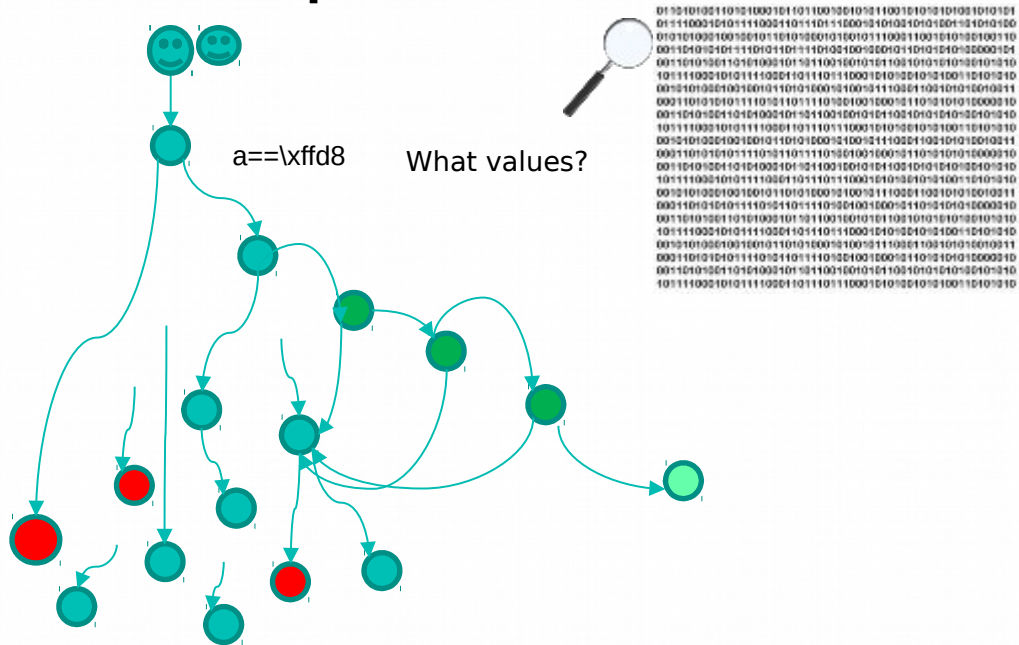


```
011010100110101000101101100101010110010101010101010101
01111000101011110001101110111000101001010101010101010101
01010100010010101101010001010010111000110010101010100110
001101010101110101101110100100100010101010101010000101
0011010101010101000101101100100101010101010101010101010
10111100010101111000101110111000101010101010101010101010
001010100010010101010001010010111000100101010101010101011
000110101010111010111010100100100010110101010101000010
001101010101010100010110100100101102101010101010101010
1011110001010111000101110111000101010101010101010101010
00101010010010010101010100010010111000101010101010101010
00101010010010010101010100010010111000101010101010101011
000110101010111010111010100100010110101010101000010
0011010101010100010101001001010101010101010101010101010
1011110001010111000101110111000101010101010101010101010
0010101001001001010101000101001011100010010101010101011
001010100100100101010101000101001011100010010101010101011
000110101010111010111010100100010110101010101000010
0011010101010101000101101100100101010101010101010101010
1011110001010111000101110111000101010101010101010101010
0010101001001001010101000101001011100010010101010101011
000110101010111010111010100100010110101010101000010
001101010101010001011001001010101010101010101010101010
1011110001010111000101110111000101010101010101010101010
```

Easy paths (superficial paths), error code

Problem Exemplified....

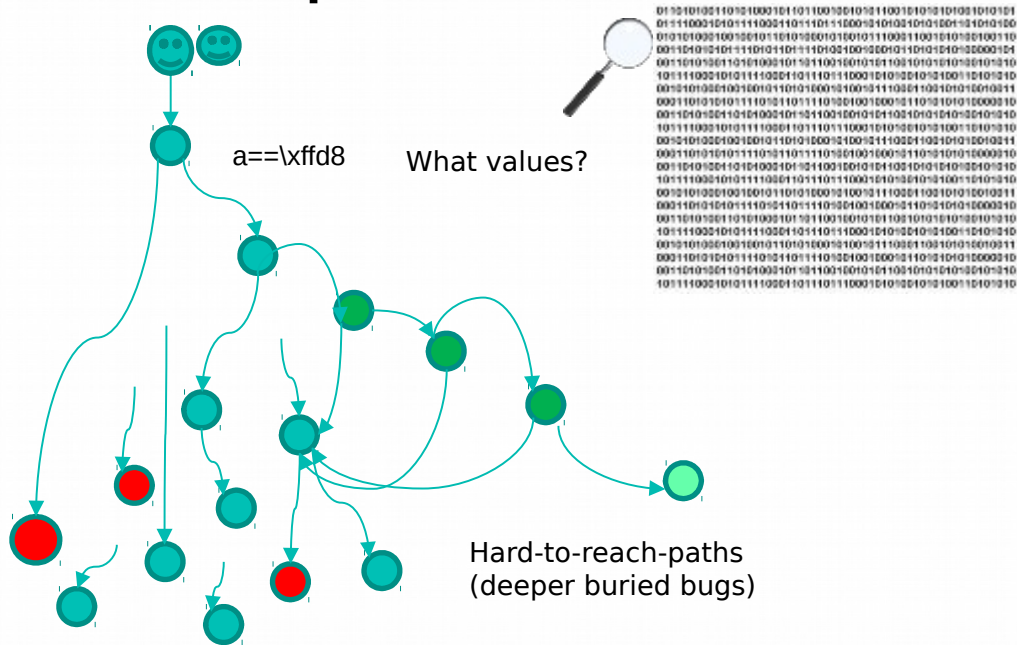
Where is 'a'?



Easy paths (superficial paths), error code

Problem Exemplified....

Where is 'a'?



Easy paths (superficial paths), error code

Issues identified...

- For smart code-coverage based fuzzer, it is important to have some knowledge about:

Issues identified...

- For smart code-coverage based fuzzer, it is important to have some knowledge about:
 - Where (which offsets in input) to apply mutation

Issues identified...

- For smart code-coverage based fuzzer, it is important to have some knowledge about:
 - Where (which offsets in input) to apply mutation
 - What values to replace with.

Issues identified...

- For smart code-coverage based fuzzer, it is important to have some knowledge about:
 - Where (which offsets in input) to apply mutation
 - What values to replace with.
 - How to avoid traps (paths leading to error handling code)

Fuzzing+Symbex

Fuzzing+Symbex

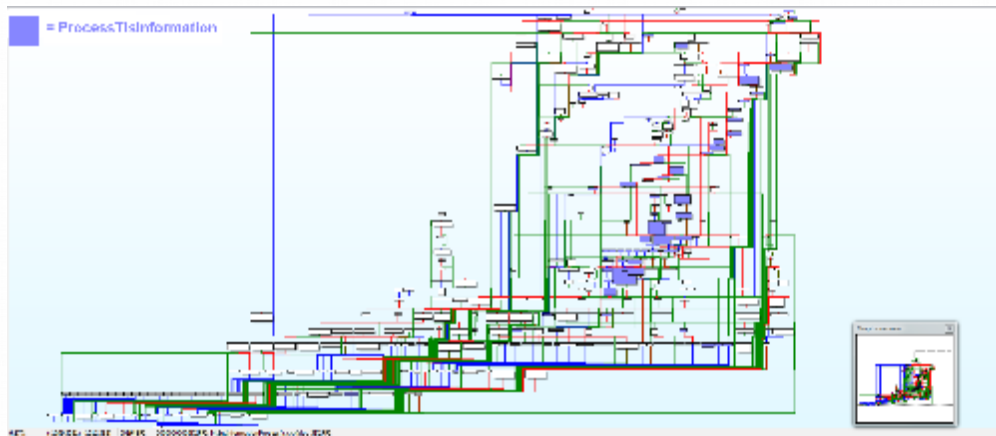
- Symbolic/concolic execution can answer such questions.
 - *Driller: Augmenting Fuzzing Through Selective Symbolic Execution*, NDSS'16

Fuzzing+Symbex

- Symbolic/concolic execution can answer such questions.
 - *Driller: Augmenting Fuzzing Through Selective Symbolic Execution*, NDSS'16
- But... Scalability?

Fuzzing+Symbex

- Symbolic/concolic execution can answer such questions.
 - *Driller: Augmenting Fuzzing Through Selective Symbolic Execution*, NDSS'16
- But... Scalability?



Observations on Fuzzing+Symbex

- Lava: Large-scale automated vulnerability addition,” in Proc. IEEE S&P '16. IEEE Press, 2016.

Observations on Fuzzing+Symbex

- Lava: Large-scale automated vulnerability addition,” in Proc. IEEE S&P '16. IEEE Press, 2016.
 - quickly and automatically injecting large numbers of realistic bugs into program source code.

Observations on Fuzzing+Symbex

- Lava: Large-scale automated vulnerability addition,” in Proc. IEEE S&P '16. IEEE Press, 2016.
 - quickly and automatically injecting large numbers of realistic bugs into program source code.
 - injected bug is designed to be triggered only if a particular set of multi-bytes in the input is set to a magic value

Observations on Fuzzing+Symbex

- Lava: Large-scale automated vulnerability addition,” in Proc. IEEE S&P '16. IEEE Press, 2016.
 - quickly and automatically injecting large numbers of realistic bugs into program source code.
 - injected bug is designed to be triggered only if a particular set of multi-bytes in the input is set to a magic value
 - Results are not very encouraging!

Concrete results (From LAVA paper)

Program	Total Bugs	Unique Bugs Found		
		FUZZER	SES	Combined
uniq	28	7	0	7
base64	44	7	9	14
md5sum	57	2	0	2
who	2136	0	18	18
Total	2265	16	27	41

QSYM- Enhancing Fuzzing by Enhancing Symbex

QSYM- Enhancing Fuzzing by Enhancing Symbex



QSYM- Enhancing Fuzzing by Enhancing Symbex

- Presented in Usenix Sec'18




QSYM- Enhancing Fuzzing by Enhancing Symbex

- Presented in Usenix Sec'18
- Focuses on scaling symbex
 - Native execution, contrary to IR based execution in existing symbex tools
 - Instruction-level symbolic execution
 - Only the relevant instructions are executed symbolically (taintflow analysis)
 - Solving only relevant constraints related to the target branch
 - Optimistic Solving
 - ...



QSYM- Enhancing Fuzzing by Enhancing Symbex


- Presented in Usenix Sec'18
- Focuses on scaling symbex
 - Native execution, contrary to IR based execution in existing symbex tools
 - Instruction-level symbolic execution
 - Only the relevant instructions are executed symbolically (taintflow analysis)
 - Solving only relevant constraints related to the target branch
 - Optimistic Solving
 - ...
-  Maintaining scalability with good heuristics + program analysis to improve coverage

VUzzer- going further with more analysis

VUzzer- going further with more analysis

- Presented at NDSS'17
- Uses taintflow analysis + several heuristics
- Main idea:

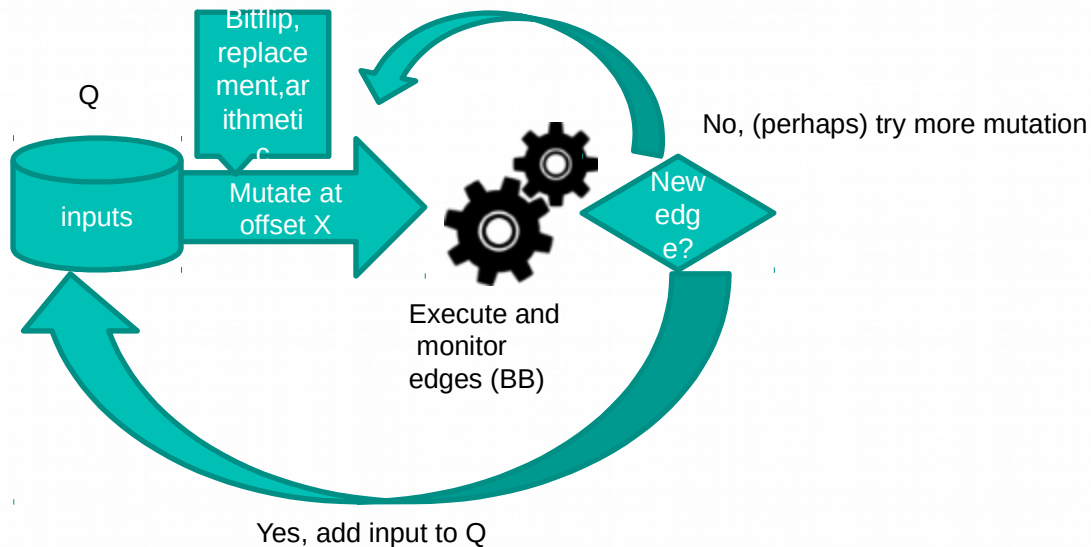
VUzzer- going further with more analysis

- Presented at NDSS'17
 - Uses taintflow analysis + several heuristics
 - Main idea:
 - Leverage application's control- and data-flow features to infer input properties:
applications is designed to work with that input!
 - *Dynamic taintflow analysis*
 - Prioritize and deprioritize paths: *Certain paths are difficult to execute as they are guarded by constraints (nested conditions)!*
 - *Static analysis and error handling code*
-  ▪ Combines static and dynamic analysis + heuristics to improve coverage

Evolving Taintflow based Solution

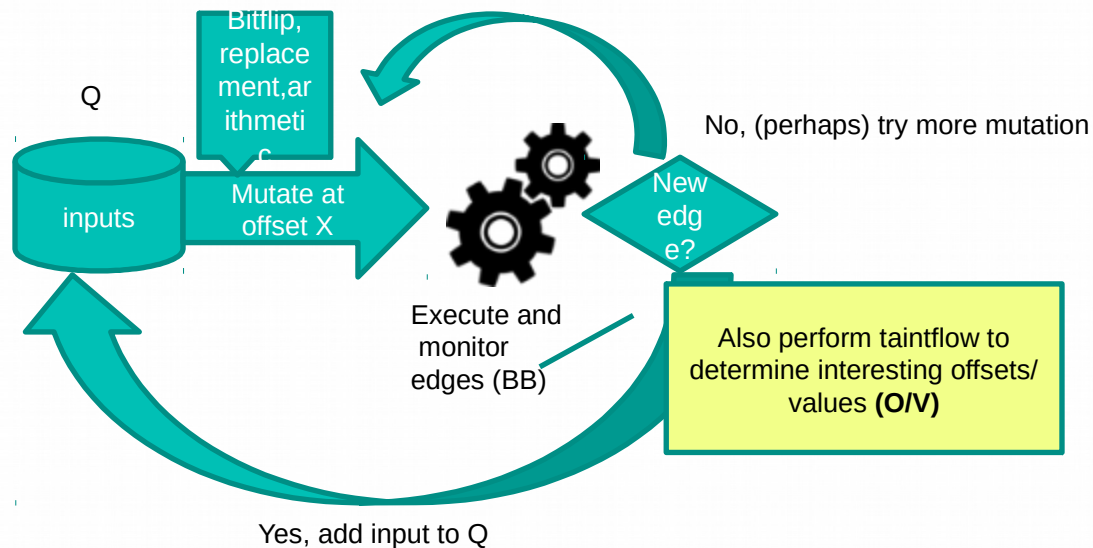
Evolving Taintflow based Solution

- Moving to Vuzzer...



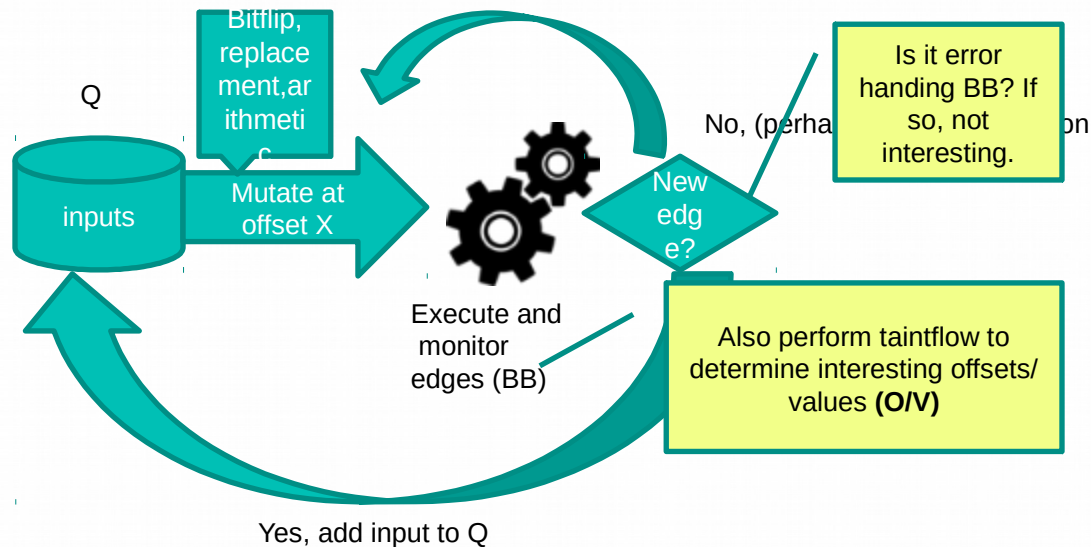
Evolving Taintflow based Solution

- Moving to Vuzzer...



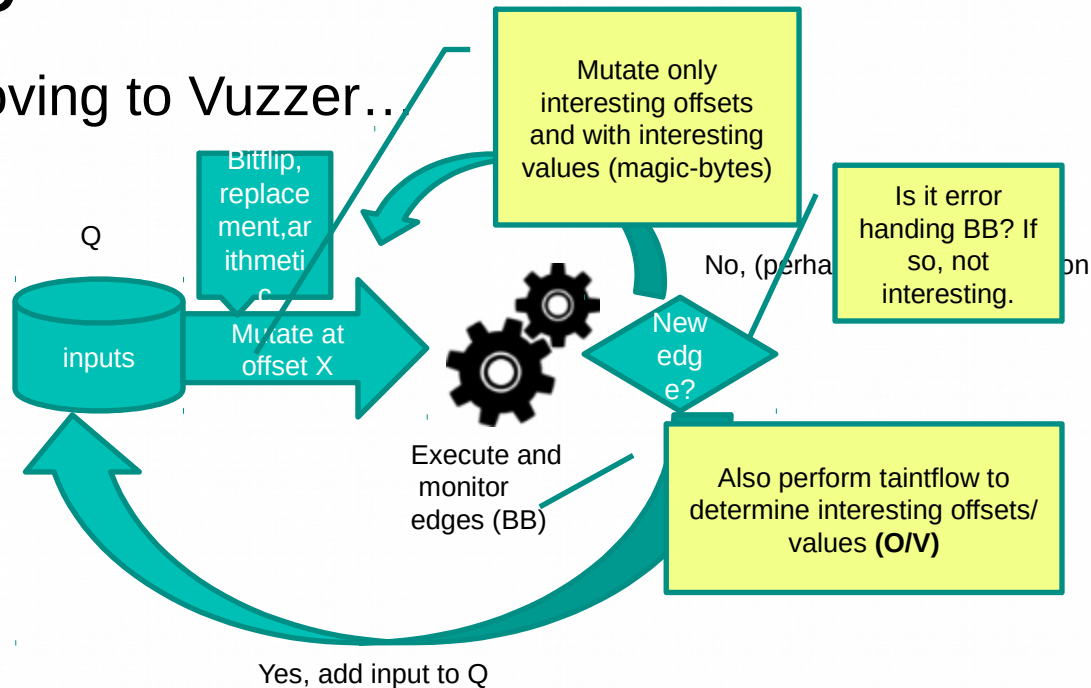
Evolving Taintflow based Solution

- Moving to Vuzzer...



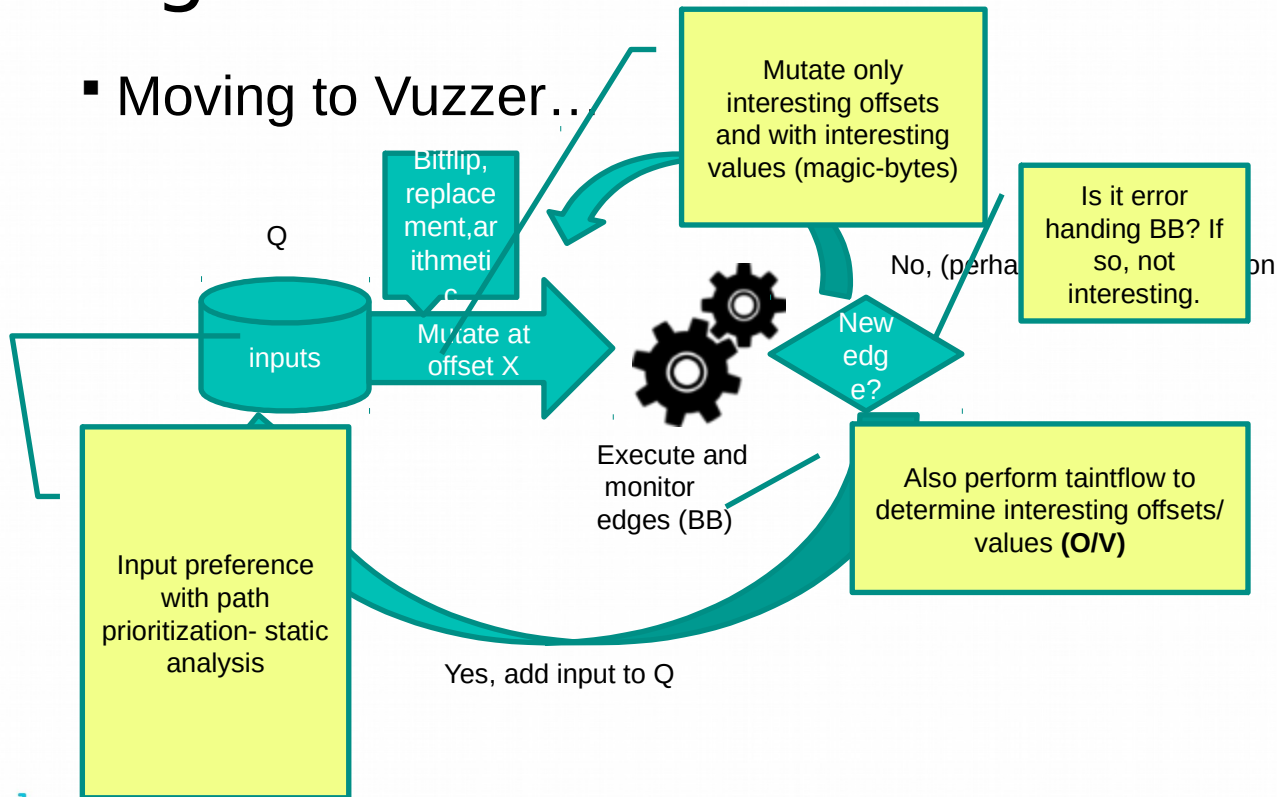
Evolving Taintflow based Solution

- Moving to Vuzzer...



Evolving Taintflow based Solution

▪ Moving to Vuzzer...



There are problems... still!

There are problems... still!

- Unaware of whether offset is processed by application
 - Waste of mutation time

There are problems... still!

- Unaware of whether offset is processed by application
 - Waste of mutation time
- What and Where to mutate
 - Different bugs have different triggering conditions
 - Buffer overflow involves buffer (strings, arrays etc.)
 - Integer overflow involves integer data type.

There are problems... still!

- Unaware of whether offset is processed by application
 - Waste of mutation time
- What and Where to mutate
 - Different bugs have different triggering conditions
 - Buffer overflow involves buffer (strings, arrays etc.)
 - Integer overflow involves integer data type.

Traditional Byte by byte mutation may not be a very effective strategy!

There are problems... still!

- Unaware of whether offset is processed by application
 - Waste of mutation time
- What and Where to mutate
 - Different bugs have different triggering conditions
 - Buffer overflow involves buffer (strings, arrays etc.)
 - Integer overflow involves integer data type.

Traditional Byte by byte mutation may not be a very effective strategy!

TIFF (presented at ACSAC 2018)

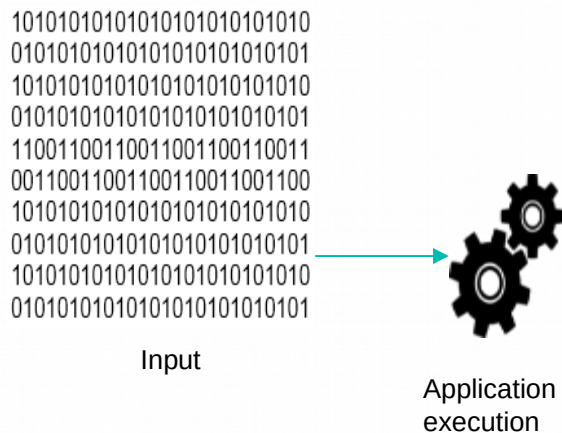
Type inference: Main Insight

Type inference: Main Insight

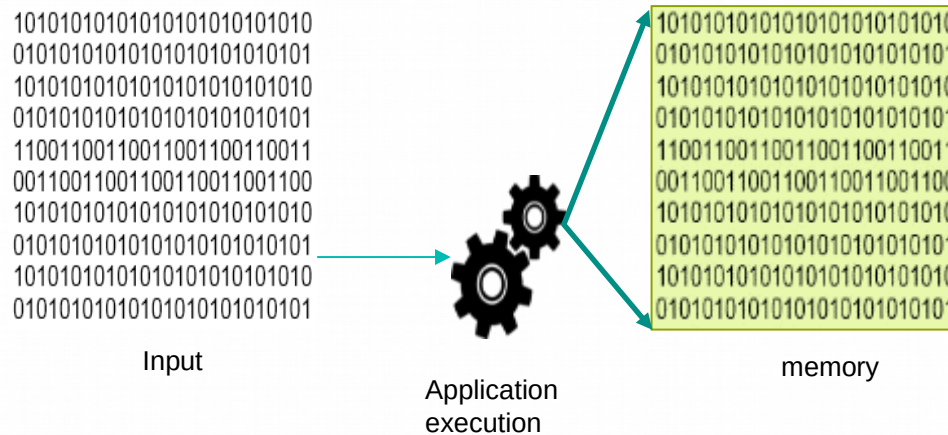
```
1010101010101010101010101010
0101010101010101010101010101
1010101010101010101010101010
0101010101010101010101010101
11001100110011001100110011
00110011001100110011001100
1010101010101010101010101010
0101010101010101010101010101
1010101010101010101010101010
0101010101010101010101010101
```

Input

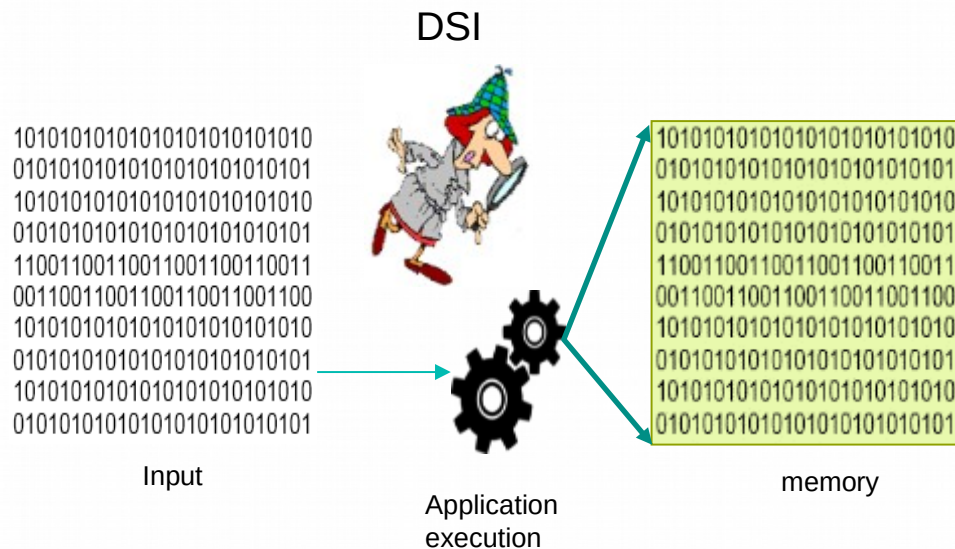
Type inference: Main Insight



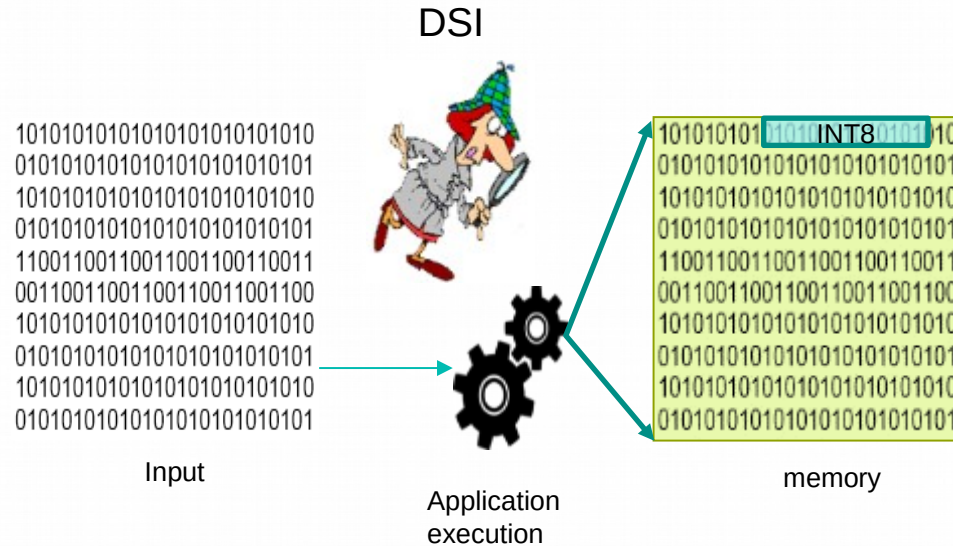
Type inference: Main Insight



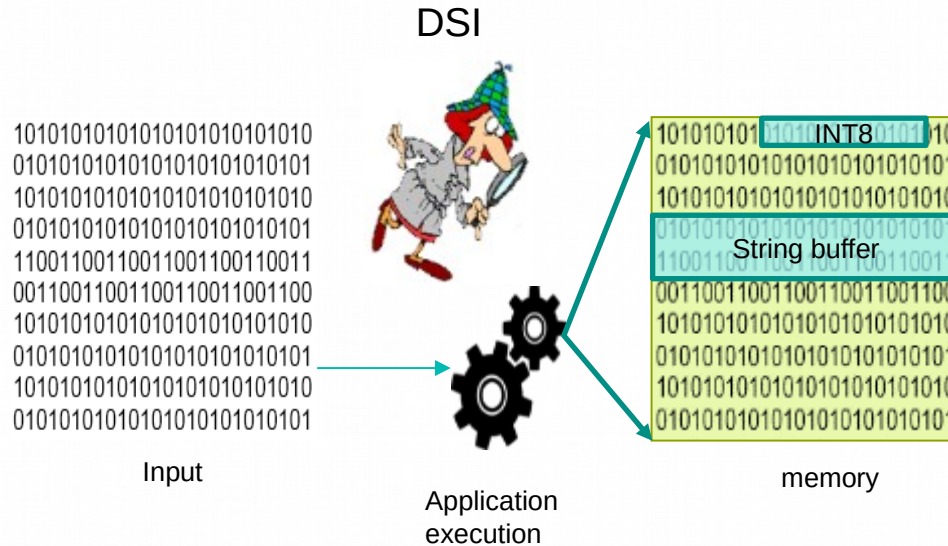
Type inference: Main Insight



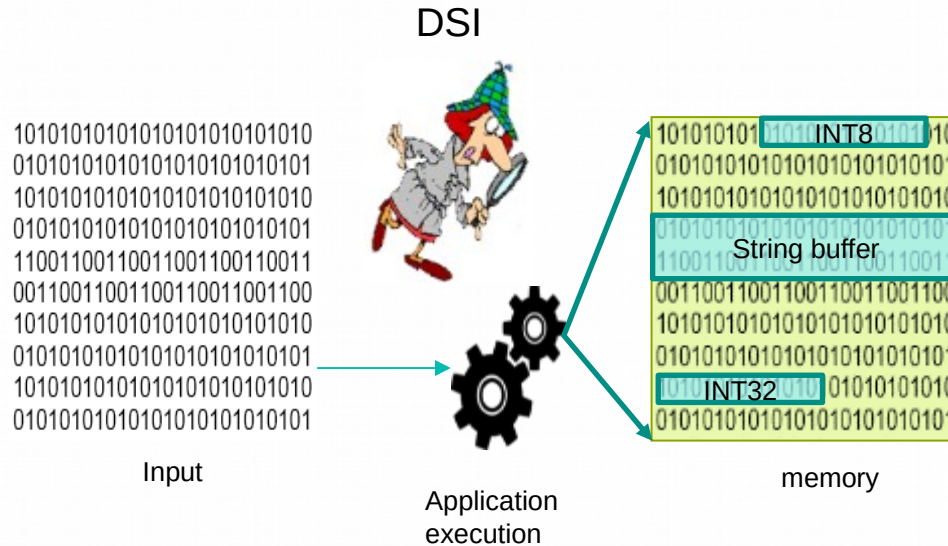
Type inference: Main Insight



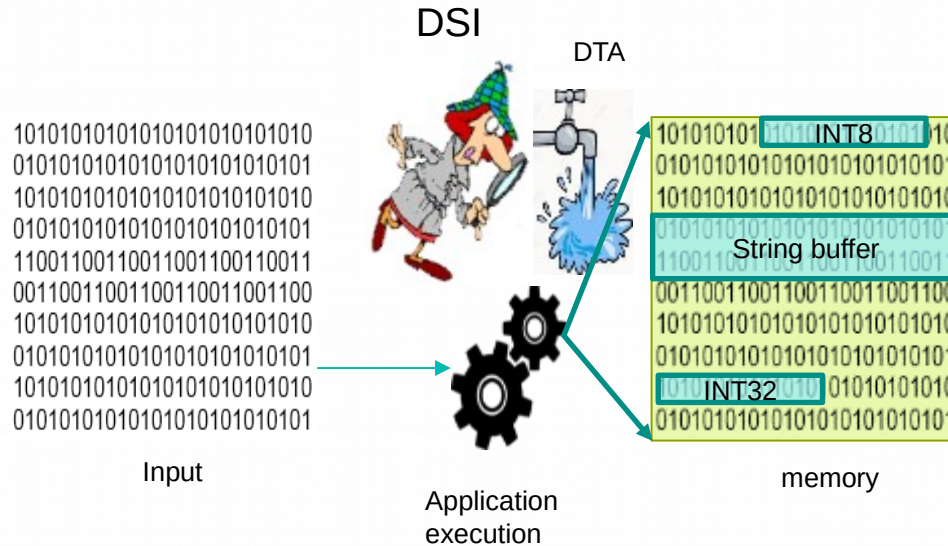
Type inference: Main Insight



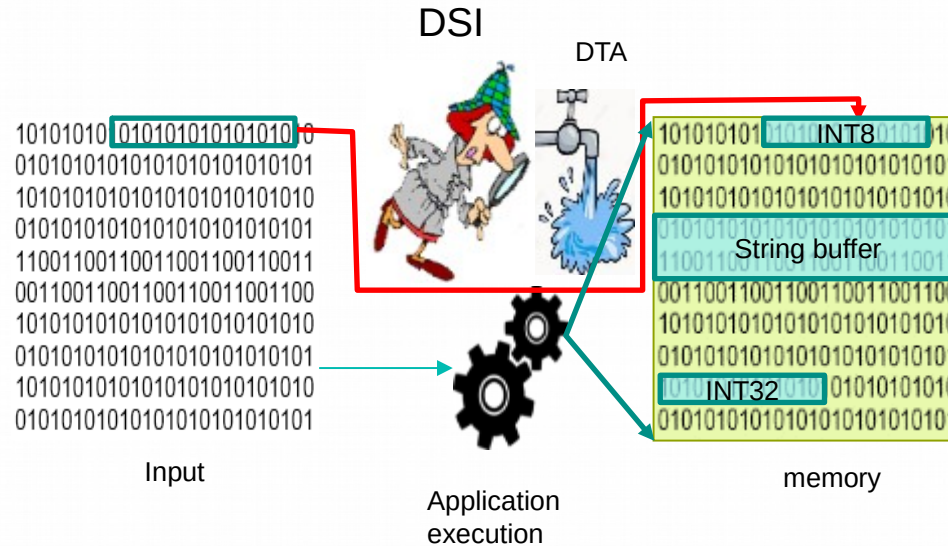
Type inference: Main Insight



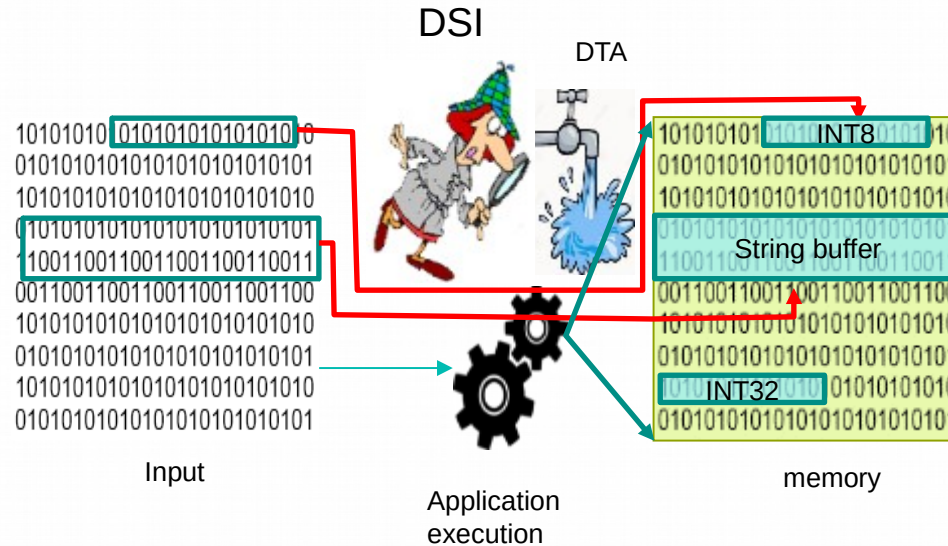
Type inference: Main Insight



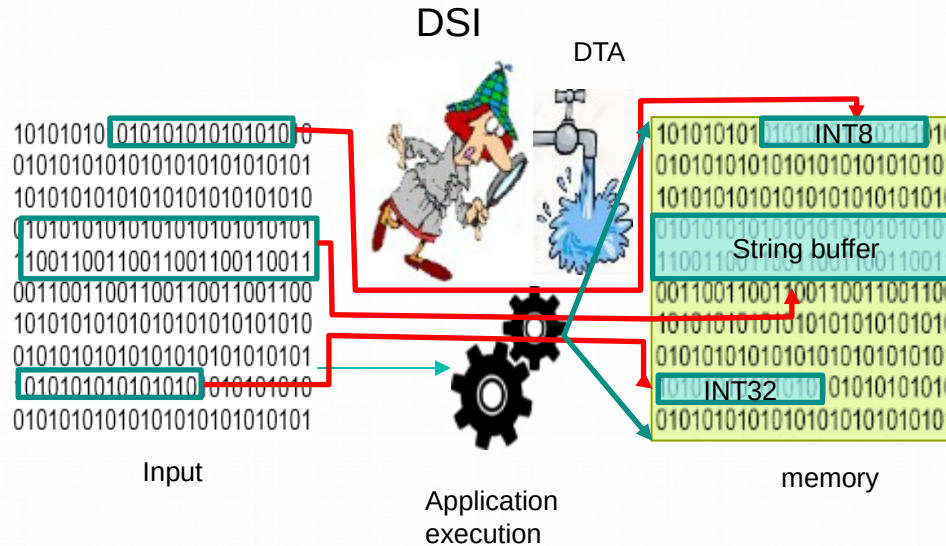
Type inference: Main Insight



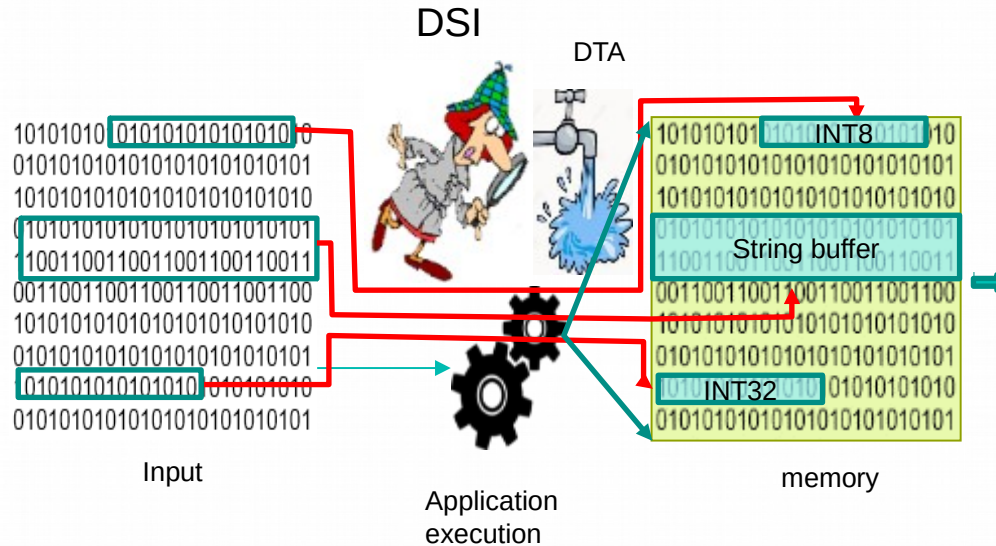
Type inference: Main Insight



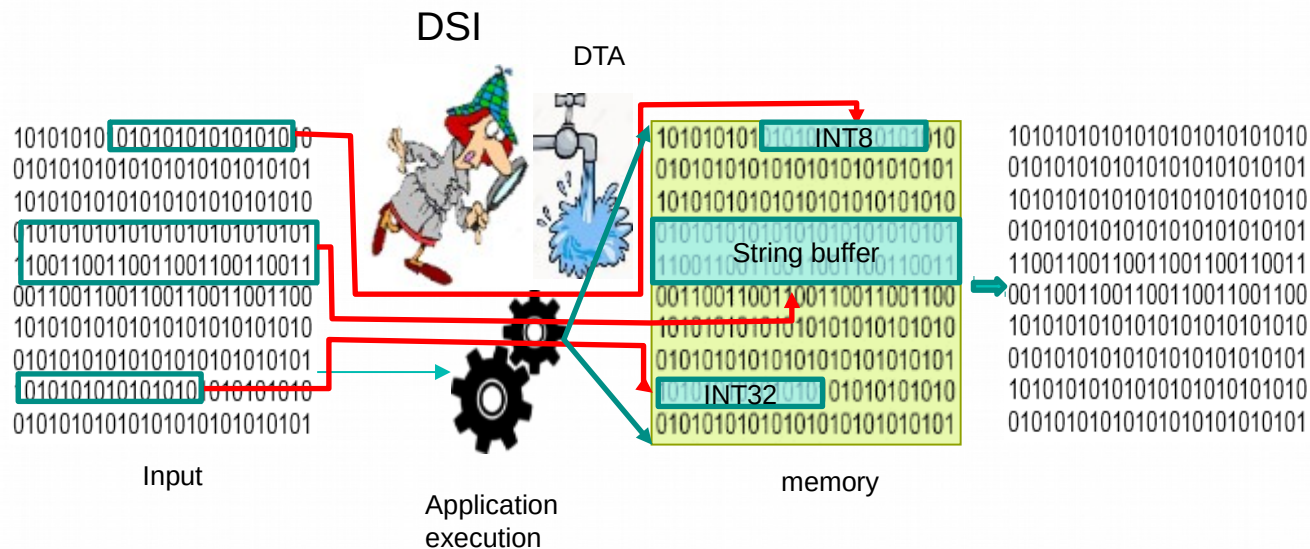
Type inference: Main Insight



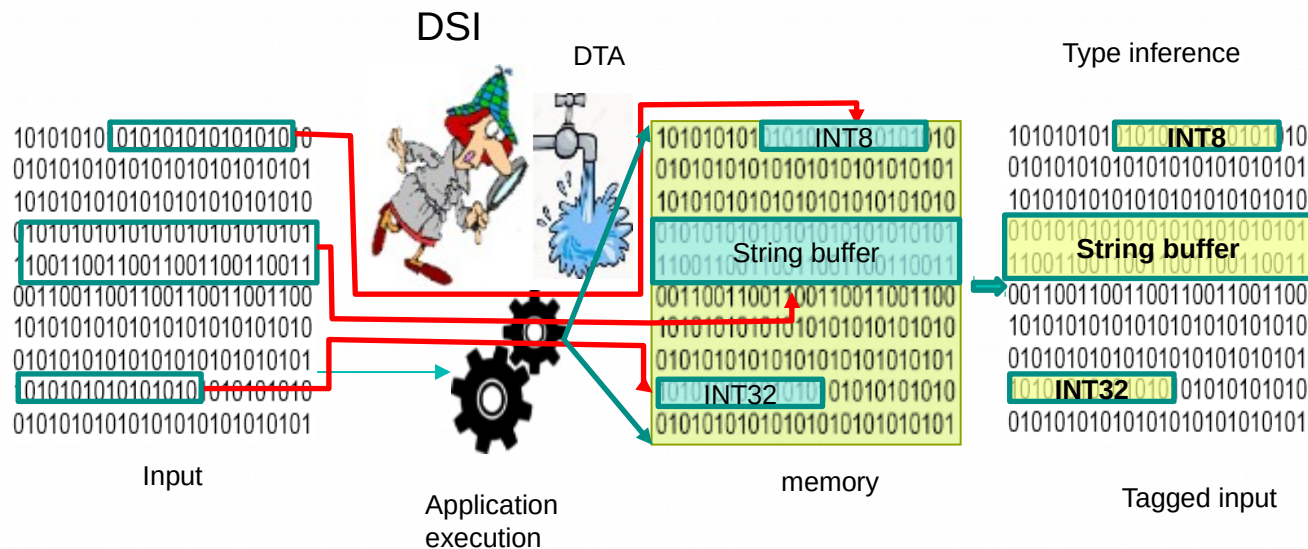
Type inference: Main Insight



Type inference: Main Insight



Type inference: Main Insight



Input format learning (Grammar based fuzzing)

Input format learning (Grammar based fuzzing)



Input format learning (Grammar based fuzzing)

- TIFF brought forward the idea of bringing grammar and mutation based fuzzing closer!



Input format learning (Grammar based fuzzing)

- TIFF brought forward the idea of bringing grammar and mutation based fuzzing closer!
- Angora (S&P'18)- on source code (LLVM based)



Input format learning (Grammar based fuzzing)

- TIFF brought forward the idea of bringing grammar and mutation based fuzzing closer!
- Angora (S&P'18)- on source code (LLVM based)
- ProFuzzer (S&P'19)- learning input structure with execution patterns



Input format learning (Grammar based fuzzing)

- TIFF brought forward the idea of bringing grammar and mutation based fuzzing closer!
- Angora (S&P'18)- on source code (LLVM based)
- ProFuzzer (S&P'19)- learning input structure with execution patterns
- GRIMOIRE: Synthesizing Structure while Fuzzing (Usenix Sec'19)- grammar inference.



Input format learning (Grammar based fuzzing)

- TIFF brought forward the idea of bringing grammar and mutation based fuzzing closer!
- Angora (S&P'18)- on source code (LLVM based)
- ProFuzzer (S&P'19)- learning input structure with execution patterns
- GRIMOIRE: Synthesizing Structure while Fuzzing (Usenix Sec'19)- grammar inference.
- and many more.. See: <https://github.com/fengjixuchui/FuzzingPaper>



Input format learning (Grammar based fuzzing)

- TIFF brought forward the idea of bringing grammar and mutation based fuzzing closer!
- Angora (S&P'18)- on source code (LLVM based)
- ProFuzzer (S&P'19)- learning input structure with execution patterns
- GRIMOIRE: Synthesizing Structure while Fuzzing (Usenix Sec'19)- grammar inference.
- and many more.. See: <https://github.com/fengjixuchui/FuzzingPaper>



▪ Focus shifted to How to mutate sensibly?

Evaluating Fuzzers- A tough question!

- What experimental setup is needed to produce trustworthy results?
[Evaluating Fuzz Testing, Klees *et. al.* CCS'18]
- There is a randomness in mutation operation- thus results may differ run to run. Multiple runs.
- Dataset- quite arbitrary (LAVA-M, Google fuzz, a set of real-world applications, binutils,..)
 - VUzzer, perhaps for the 1st time, used *three different datasets* in the evaluation (DARPA CGC, LAVA-M, real-world apps)
- Seed selection- which inputs to start with?

Evaluating Fuzzers- A tough question!

- How to measure efficiency?
 - Code-coverage, but what about directed fuzzers?
 - Also for binary only fuzzers, measuring code coverage is not that straight forward-static binary instrumentation
 - Also, for source code based fuzzers, what about library code?
 - Uniqueness of crashes
 - How to differentiate several crashes? Often coredump does not have enough information!
 - Root-cause analysis (not much is there! *Failure Sketching*, G. Candea EPFL)

Good Engineering

- (the scope of) Optimization is everywhere in a fuzzer.
- Light-weight fuzzers (e.g. AFL)
 - Branch bitmap (64K to be fit into the cache)
 - Fork()
 - Input trimming
- Every program analysis introduces a performance hit
 - F1 (World's fastest grammar based fuzzer- it is F0 by Brandon Falk)
- VUzzer uses memory file system (tmpfs).
- Vectorized Emulation: Putting it all together (Brandon Falk)

Conclusions

Conclusions

- Fuzzing – seems easy unless you try it!

Conclusions

- Fuzzing – seems easy unless you try it!
- Scalability and performance cannot be negotiated much!
 - A good engineering, hardware assisted monitoring

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

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!