Fuzzgrind: an automatic fuzzing tool

Gabriel Campana
Sogeti / ESEC
gabriel.campana(at)sogeti.com

Plan

Roadmap

Roadmap

State of the art

Fuzzing

Technique to search for software implementation errors by injecting invalid data.

- Test generation:
  - random,
  - input mutation,
  - model-based.
- Several fuzzing software programs.
- Endless process: study of specifications, reverse engineering of protocols, new development for each target, etc.
- Innovative theories: `Autodafe`, `Flayer`, `Sage`, etc.

Roadmap

Goal

- Let fuzzing be **completely** automatic.
- Give a target program and an input file,
- New inputs generated automatically,
- Wait for crashes.

Roadmap

Concept

Symbolic execution

Use of algebraic expressions to represent the variable values throughout the execution of the program.

1. Symbolically execute the target program on a given input,
2. Analyze execution path and extract path conditions depending on the input,
3. Negate each path condition,
4. Solve constraints and generate new test inputs.
5. This algorithm is repeated until all executions path are (ideally) covered.

$\Longrightarrow$ Increase code coverage to discover new bugs.

Symbolic execution: example

input = "\x06\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
a=6	i0	

Symbolic execution: example

input = "\x06\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0	
a=6, b=15	i1	

Symbolic execution: example

input = "\x06\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0	
a=6, b=15	i1	

Symbolic execution: example

input = "\x06\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0	
a=6, b=15	i1	

Symbolic execution: example

input = "\x06\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0 i1	
a=6, b=15, c=9	c=i0+3	

Symbolic execution: example

input = "\x06\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0 i1	
	c=i0+3	
a=6, b=15, c=9		i0+3 <= 42

Symbolic execution: example

```
input = "\x28\x00\x00\x00\x0f\x00\x00\x00"
```

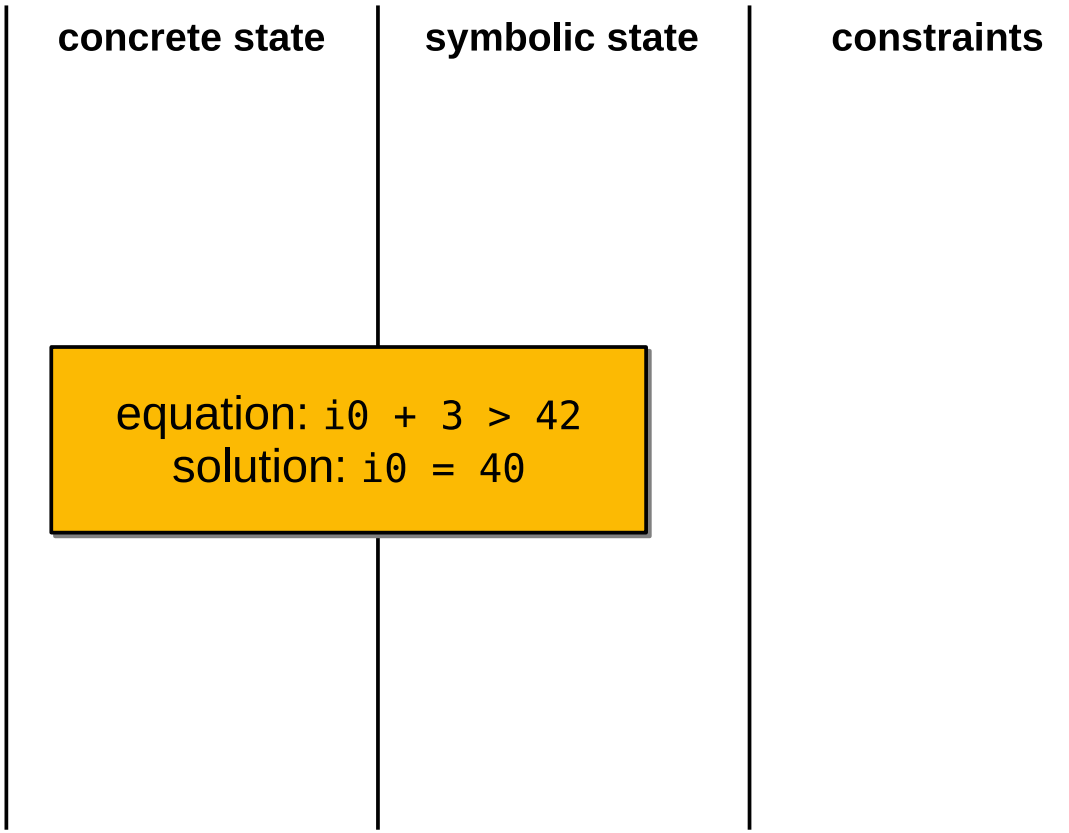
```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```



Symbolic execution: example

input = "\x28\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
a=40	i0	

Symbolic execution: example

```
input = "\x28\x00\x00\x00\x0f\x00\x00\x00"
```

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0	
a=40, b=15	i1	

Symbolic execution: example

```
input = "\x28\x00\x00\x00\x0f\x00\x00\x00"
```

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0	
a=40, b=15	i1	

Symbolic execution: example

input = "\x28\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0	
a=40, b=15	i1	

Symbolic execution: example

input = "\x28\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0 i1	
a=40, b=15, c=43	c=i0+3	

Symbolic execution: example

input = "\x28\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0 i1	
	c=i0+3	
a=40, b=15, c=43		i0+3 > 42

Symbolic execution: example

input = "\x28\x00\x00\x00\x0f\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0 i1	
	c=i0+3	
a=40, b=15, c=43		i0+3 > 42 i0 – i1 != 7

Symbolic execution: example

```
input = "\x28\x00\x00\x00\x21\x00\x00\x00"
```

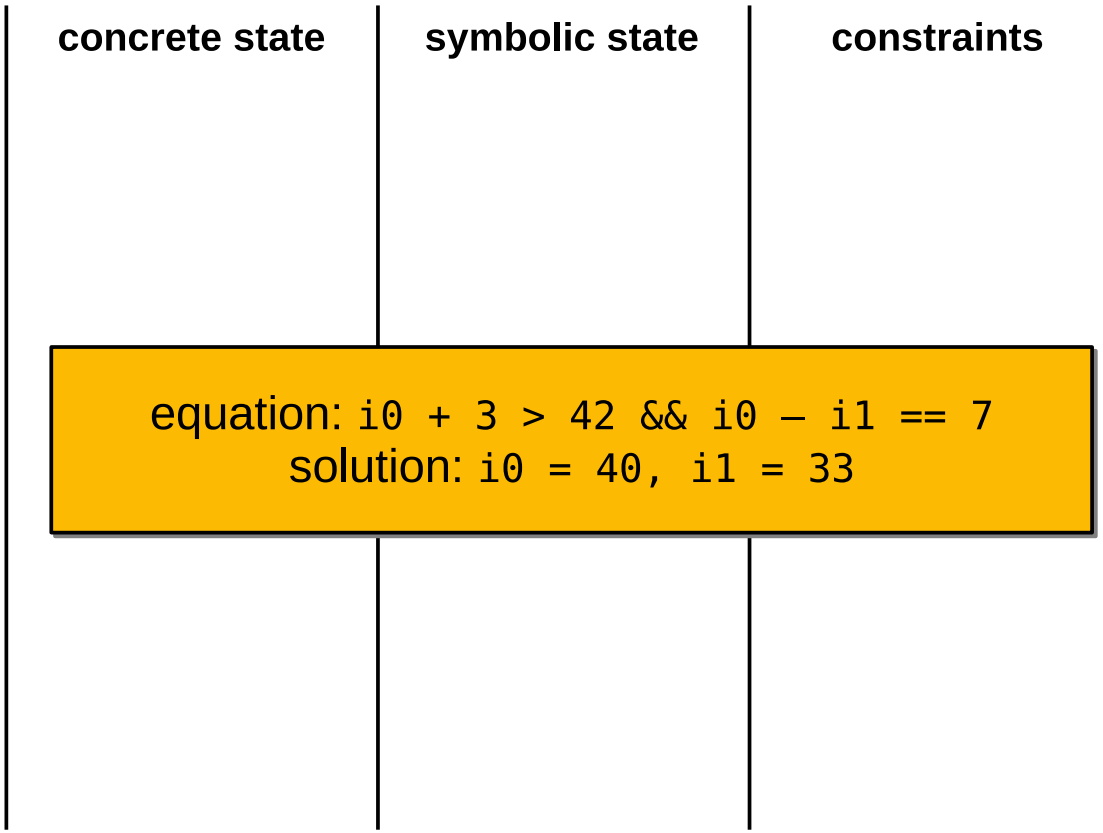
```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```



Symbolic execution: example

input = "\x28\x00\x00\x00\x21\x00\x00\x00"

```
void main() {
  int a, b;
  FILE *fp = fopen("input", "r");

  fread(&a, sizeof(int), 1, fp);
  fread(&b, sizeof(int), 1, fp);

  f(a, b);
}

void f(int a, int b) {
  int c = a + 3;

  if (c > 42) {
    if (a - b == 7)
      error();
  }
}
```

concrete state	symbolic state	constraints
	i0	
	i1	
	c=i0+3	
		i0+3 > 42
		i0 − i1 == 7
a=40, b=33, c=43		

Roadmap

Roadmap

Valgrind

- Framework for Dynamic Binary Instrumentation.
- Multiple architectures supported.
- Generic framework for creating program analysis tools (e.g.: *Memcheck*).
- Machine-code interpreter: just-in-time instruction recompilation.
- Nothing from the original program ever gets run directly.

Overall view

1. Disassembly and translation from machine code (x86) into Intermediate Representation (VEX),
2. IR optimization,
3. IR instrumentation by the plugin,
4. Conversion of the instrumented IR into machine code (x86) and register allocation,
5. Instrumented code execution.

IR and instrumentation

```
0x4000A99:  movl %eax,%ecx

   #-----  IMark(0x4000A99, 2)  ------
   PUT(4) = GET:I32(0)                    # copy of EAX into ECX
```

```
0x4000A9B:  leal  0x2C(%ebx), %esi

   #-----  IMark(0x4000A9B, 6)  ------
   PUT(60) = 0x4000A9B:I32                # EIP update
   t0 = Add32(GET:I32(12),0x2C:I32)       # addition of EBX content with 0x2C
   PUT(24) = t0                           # result copy into ESI
```

System calls

1. copy virtual registers into real registers (apart `EIP`),
2. do the system call,
3. copy real registers into virtual registers (apart `EIP`),
4. restore stack pointer.

Roadmap

STP – A Fast Prover

- Constraint solver, generated by static and dynamic analysis programs.
- Especially fast.
- Used in *Automatic Patch-Based Exploit Generation* paper.
- Take in input a request set of one or multiples constraints.
- Constraints composed of a set of functions and predicates.

- Output tells if request is satisfiable or not.
- Counter-example display.

Example

```
# cat file.c
  ...
  char x, y;
  if (x * y == 16)
  ...
```

```
# cat file.stp
x : BITVECTOR(8);
y : BITVECTOR(8);
QUERY(NOT(BVMULT(8, x, y) = 0h10));
```

```
# stp -p file.stp
Invalid.
ASSERT( y = 0hex05    );
ASSERT( x = 0hexD0    );
```

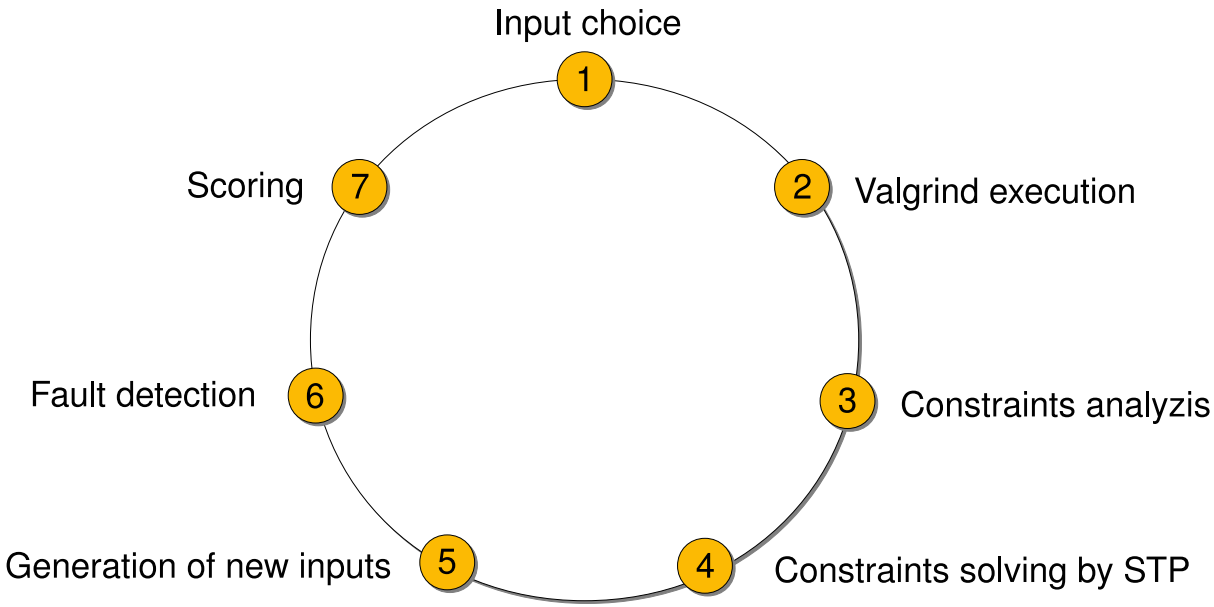
Softwares used

- Valgrind: path conditions search.
- STP: constraints solving.
- Python scripts to link all of this.

Roadmap

1. Introduction
2. Valgrind and STP
3. Implementation
   - Valgrind plugin
   - Constraint analysis
   - Constraints solving
   - Fault detection
   - Scoring
4. Conclusion

Overall view



Roadmap

2) Valgrind plugin: search for path conditions

1. Initial tainting of data from the tainted source,
2. Propagation and display of constraints associated with tainted data.
3. Untaint data.

2) Valgrind plugin: initial tainting

- Based on Flayer's implementation.
- Supported inputs: files and standard input.
- System calls monitoring.
- `open`: file descriptor monitoring,
- `close`: ends file descriptor monitoring,
- `read`, `mmap`: tainted data addresses.

Tainted data

Constraint depending on the number i from the byte read/mmaped.

2) Valgrind plugin: propagation and display

- Function of the tool in charge of the instrumentation of the IR.
- Tainted data: memory address and registers.
- Instruction doesn't depend on tainted data $\Longrightarrow$ ignored/result untainted,
- Operation on tainted data $\Longrightarrow$
  - Tainting of temporary that saves the result,
  - Temporary associated to the constraint that is associated to the operation.

```
t0 = Add32(GET:I32(12), 0x2C:I32)
```

- Condition depends on tainted data $\Longrightarrow$ associated constraint is displayed.

```
0x08048e0d: CmpEQ32(8Uto32(LDle:I8(input(0))),0x0:I32) => 0
```

Roadmap

3) Constraint analysis

- Translation of Valgrind's IR into STP language.
- Constraints optimization and negation.

```
0x08048e0d: CmpEQ32(8Uto32(LDle:I8(input(0))),0x0:I32) => 0
```

```
x0 : BITVECTOR(8);
QUERY(
  NOT((
    IF (((0h000000@x0) = 0h00000000)) THEN
      (0b1)
    ELSE
      (0b0)
    ENDIF)
  = 0b1));
```

Roadmap

4) Constraints solving, 5) New inputs

- Resolution of constraints using STP.

```
./stp/stp -p /tmp/example.stp
Invalid.
ASSERT( x0  = 0hex00  );
```

- If query is invalid, assign this new values,
- Generation of new test files.

Roadmap

6) Detecting faults

- Ptrace,
- Signals: `SIGSEGV`, `SIGKILL`, `SIGABRT`.
- Crackmes, tests: search of patterns in output.

Roadmap

7) Scoring

- Valgrind's `Lackey` plugin,
- Number of executed basic blocks.

Demo

Roadmap

Results

- Fuzzing **completely** automatic.
- New vulnerabilites in `readelf` and `swfextract`.
- Vulnerabilities found in `libtiff` 3.8.2 in a few minutes (discovered by Tavis Ormandy, led to the execution of unsigned code on the PSP and the iPhone).
- Resolution of simple crackmes.

Caveats

- Path explosion,
- Loop iterations,
- Cryptographic functions,
- Data tainting can be losed:

```
char *digits = "0123456789abcdefghijklmnopqrstuvwxyz";
if (digits[x] == '1')

CmpEQ32(LDle(Add32(x, digits)), 0x31)
```

Improvement

- Monitoring of network inputs.
- Scoring tool.
- Parallelization.
- Constraints caching.
- Fuzzgrind is licenced under GPL: contribute!
- `http://www.security-labs.org/fuzzgrind`

Thank your for your time.

Q&A ?