BOA

Buffer Overrun Analyzer

Motivation

The C programming language is riddled with security vulnerablitilies, the most prominent of which is the risk of buffer overruns. Programs written in C are widely used today, many of them with legacy code.

Buffer Overrun

There is an overrun in the following code, can you spot it?

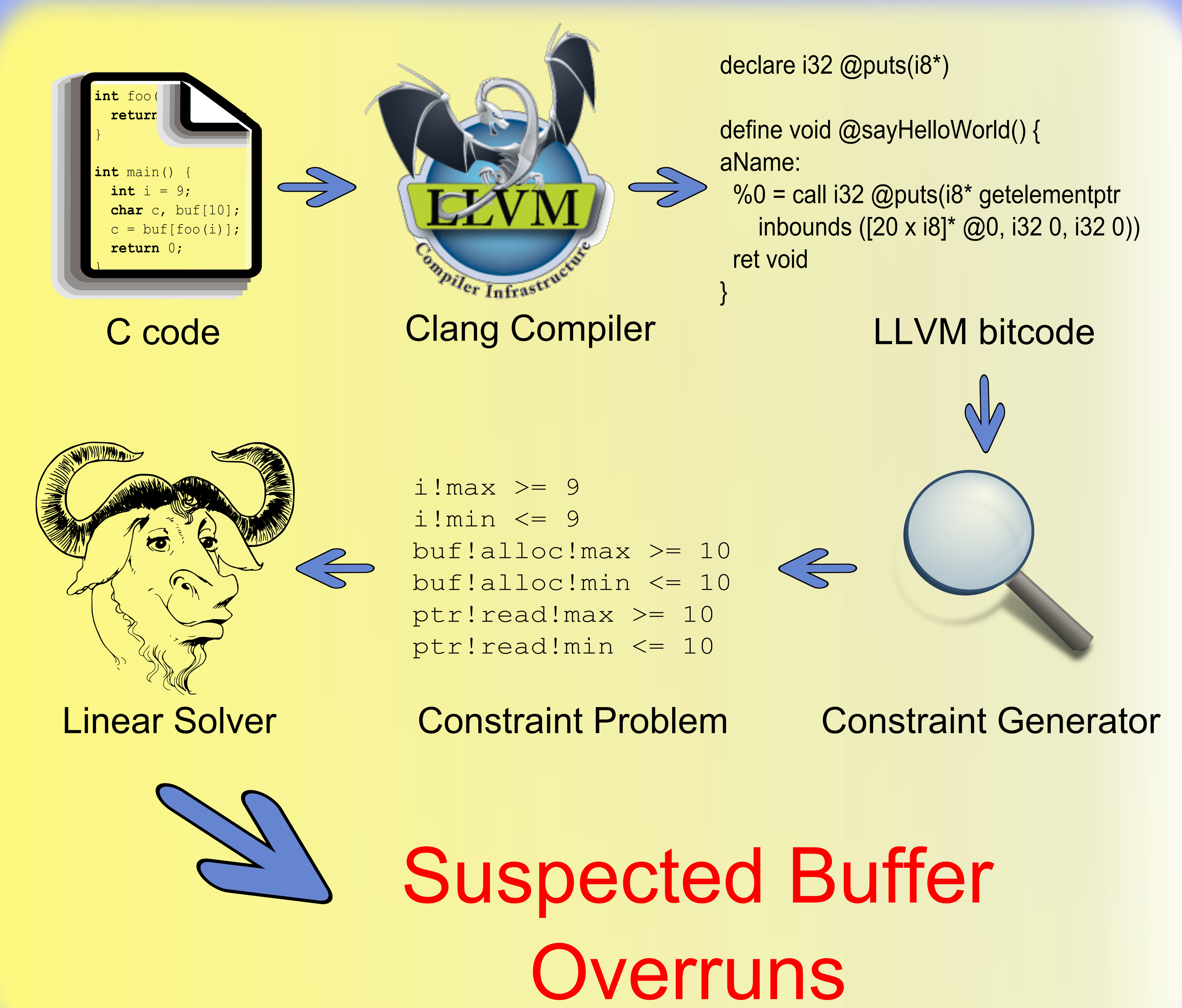
```c
int main(int argc, char *argv[]) {
  char header[2048], buf[1024];
  char *cc, *cc2, *ptr;
  int i;
  FILE *fp;
  ...
  ptr = fgets(header, 2048, fp);
  cc = copy_buffer(header);
  for (i = 0; i < 10; i++) {
    ptr = fgets(buf, 1024, fp);
    cc = copy_buffer(buf);
  }
  ...
}

char* copy_buffer(char *src) {
  char *copy;
  copy = malloc(strlen(src));
  strcpy(copy, src);
  return copy;
}
```

Approach

Translate source code statements into linear constraints on all buffers and primitives. The solution of the resulting linear problem supplies bounds on values of these variables. Analysis of these bounds indicates possible out-of-bounds access to any array.

Goal

Given a C program, statically identify for each buffer whether it may be accessed beyond its allocation.
Boa operates on C code as is, without requiring the programmer to provide any meta information. The provided analysis is sound - report 100% of the possible buffer overruns in the code, while minimizing the amount of false positives.



Blame System

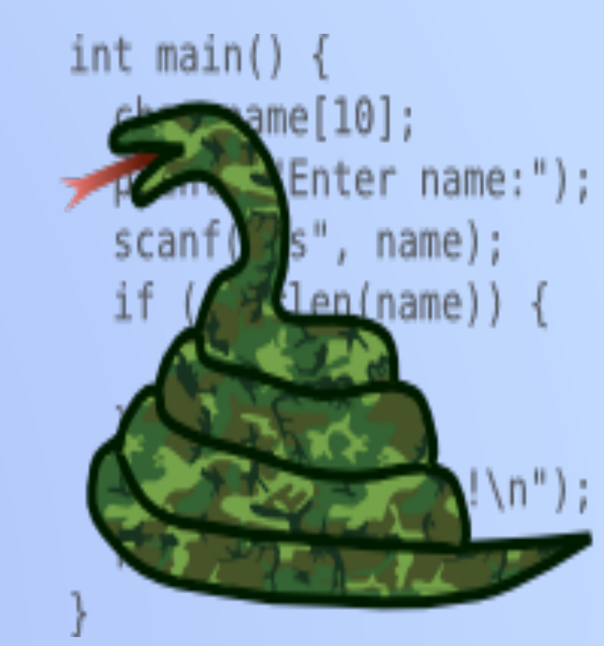
A list of suspected buffers is not informative enough, lacking root cause information and making it difficult to discern between actual overruns and false alarms.
BOA introduce the blame system - for each overrun it supplies a concise set of C statements which caused boa to classify it as an overrun, sorted by probable relevance.

Constraint Generator

Linear constraints are generated out of LLVM bitcode instructions. The constraint generator operates in a single pass and performs flow and mostly context insensitive analysis.

Constraint Types

- Integer Arithmetic
- Static Buffer Allocations
- Dynamic Buffer Allocations
- Buffer Aliasing
- Direct Array Access
- Nested Variables (structs)
- Functions
  - With Definition
  - Known Library Functions
  - Unknown functions



Example

```c
int foo(int input) {
  return input + 1;
}

int main() {
  int i = 9;
  char c, buf[10];
  c = buf[foo(i)];
  return 0;
}
```

```
foo!max >= foo!input!max + 1
foo!min <= foo!input!min + 1

i!max >= 9
i!min <= 9

buf!alloc!max >= 10
buf!alloc!min <= 10

foo!input!max >= i!max
foo!input!min <= i!min

buf!used!max >= foo!max
buf!used!min <= foo!min
```

The Linear Problem

The constraints form a linear problem, any solution provides bounds to the values of the integers and size of the buffers. In order to tighten the bounds, the linear solver is directed to maximize -

$$\Sigma_x(x!min - x!max)$$

For each buffer, if buf!alloc!min <= buf!used!max there is a possible overrun.

Edo Cohen, Tzafrir Rehan, Gai Shaked
Supervised by: Dr. Eran Yahav and Mr. Michael Kuperstein