Shape Analysis Using Separation Logic

Author

Tomáš Brablec

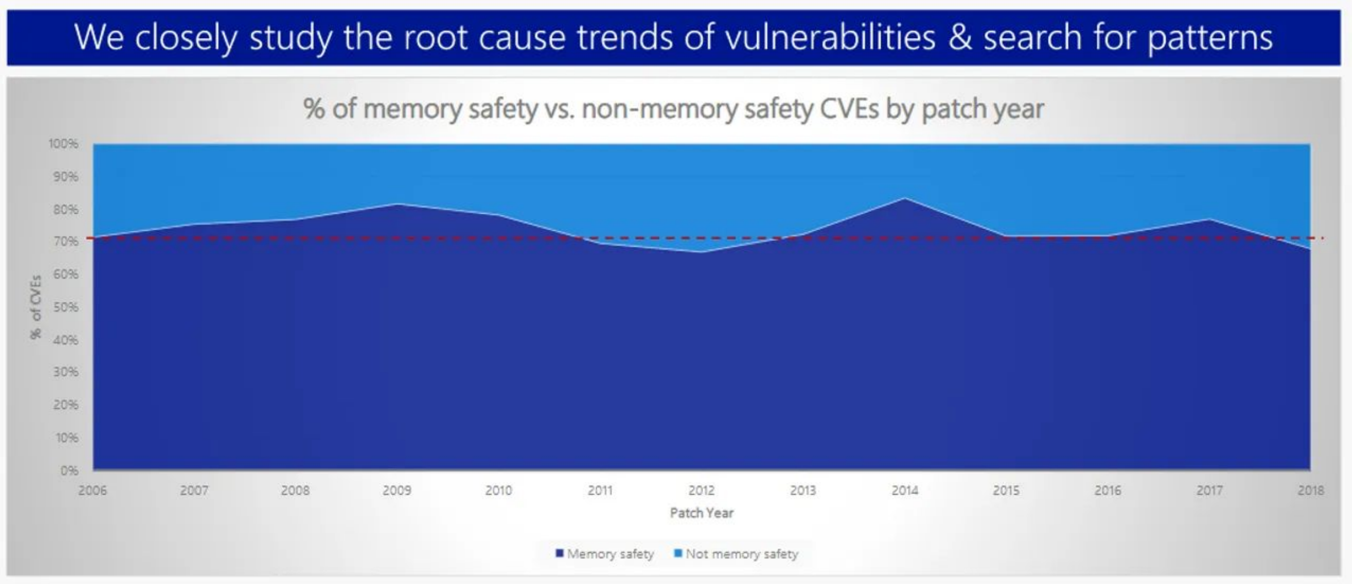
Supervisors

Tomáš Dacík, Tomáš Vojnar

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Motivation

- Manual memory management creates a whole class of bugs
- Memory errors are a common source of security vulnerabilities
- Dynamic analysis does not prove correctness
- There is space for static analysis and possibly formal verification



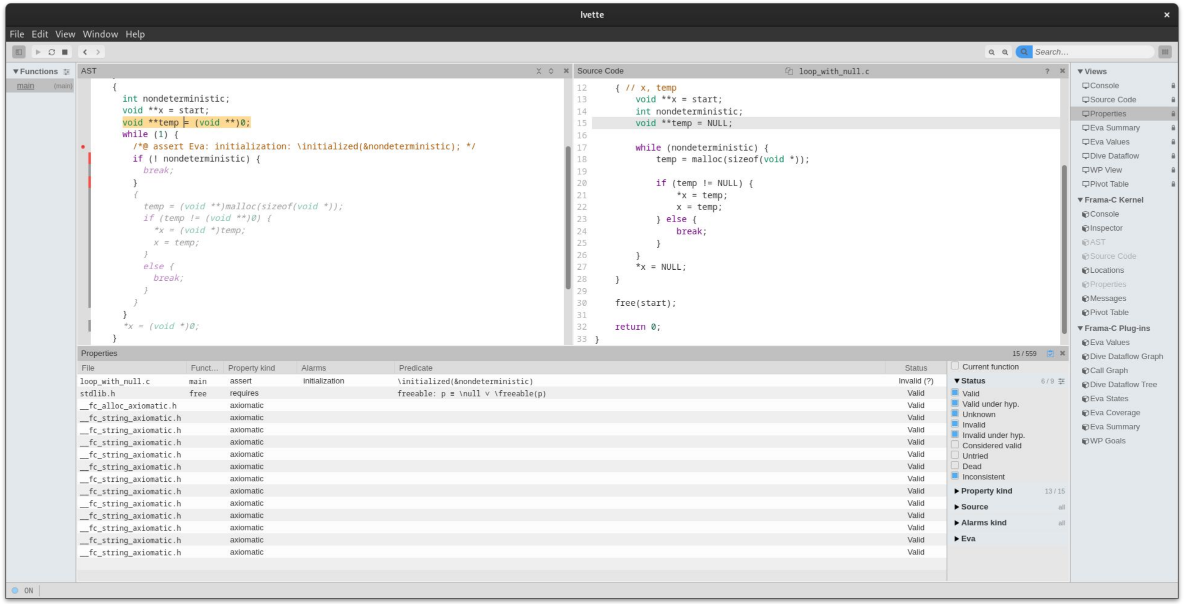
Microsoft: Around 70% of CVEs are memory related

Introduction

- The goal is to develop a prototype of a static shape analyser
- Use separation logic to represent program states
- Focus on programs that work with linked lists
- Utilize the Frama-C framework to simplify implementation
- Use the Astral solver to efficiently work with separation logic

Frama-C

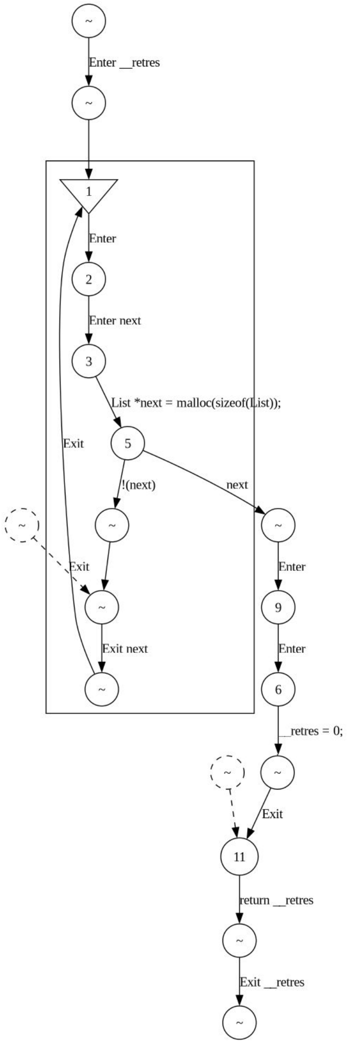
- Modular framework for analysis of C code
- The program is represented in the C Intermediate Language (CIL)
- CIL is based on a modified AST with extra information
- a GUI frontend called Ivette is under development



Software Analyzers

Dataflow Analysis

- Runs on the Control Flow Graph (CFG) of the input code
- Each node of the CFG is assigned an analysis state
- States of all nodes are updated until a fixpoint is found for all of them
- The plugin defines a transfer function that computes new states for nodes based on the states of predecessor nodes
- Updates are propagated along the edges of the CFG



example of a CFG

Separation Logic

- Developed to solve the problem of globality in analyses
- SL formulae describe the state of the heap
- Enables abstraction over linked lists of arbitrary length
- Dedicated solver for SL – Astral

$$
\begin{aligned}
\varphi &::= x = y \mid x \neq y && \text{(pure atoms)}\\
&\quad \mid x \mapsto y \mid \mathrm{ls}(x, y) && \text{(spatial atoms)}\\
\varphi &::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \wedge_{\neg} \varphi \mid \neg\varphi && \text{(boolean connectives)}\\
\varphi &::= \varphi * \varphi \mid \varphi \mathrel{-\!\!\circledast} \varphi && \text{(spatial connectives)}
\end{aligned}
$$

syntax of SL formulae

Astral

Shape Analysis

- Derives the shapes of data structures on the heap
- Previous work defines abstraction rules for a minimal language
- My objectives:
  - adapt this method to work on a subset of C
  - use Astral for solving SL formulae
- The use of Astral provides greater flexibility than the original method

Implementation

- Currently limited to programs without function calls
- Preprocessing is done on the AST to simplify statements
- Constants are removed, field accesses are converted to dereferences
- During the analysis, the formulae are simplified between statements
- Results are reported using Ivette

Original	Preprocessed
`int main() {`	`int main() {`
	`void *_nil = 0;`
`List *list = (List *)malloc(sizeof(List));`	`List *list = malloc(sizeof(List));`
`if (list == NULL) return 1;`	`if (list == _nil) return 1;`
`list->next = NULL;`	`*list = _nil;`
`list->data = 42;`	

preprocessing example

- The visit of every relevant statement modifies the state formulae

- `malloc` splits the state into the cases of successful and failed allocation

- Dereference of a list node splits the state according to the length of the list

Analyzed statement	State formulae
	$\varphi := \mathrm{emp}$
`a = malloc(8);`	
	$\varphi_1 := (a \mapsto l_0)$ $\varphi_2 := (a = \mathrm{nil})$
	$\varphi := \mathrm{ls}(x, \mathrm{nil})$
`y = x->next;`	
	$\varphi_1 := (x \mapsto y) * \mathrm{ls}(y, \mathrm{nil})$ $\varphi_2 := (x \mapsto y) * (y = \mathrm{nil})$

Abstraction and simplification

- The abstraction ensures that the analysis will terminate
- Between statements, the formulae are also simplified and deduplicated
- This is done to speed up the analysis

Type of simplification	**Input formula**	**Output formula**
abstraction to list segments	$(x \mapsto l_0) * (l_0 \mapsto y)$	$\mathrm{ls}(x, y)$
removal of variables at the end of their scope	$(x = y) * (y \mapsto z)$	$(y \mapsto z)$
removal of irrelevant inequalities	$(x \mapsto y) * (y \neq l_0)$	$(x \mapsto y)$
removal of unsatisfiable formulae	$(x \mapsto y) * (x = y)$	

examples of simplifications

Example of analysis results

- The analyzed code constructs a linked list, traverses all nodes, and then deallocates it

- The following program works with a singly linked list

```
typedef struct List {
        struct List *next;
        int data;
} List;
```

- The program starts by allocating the first node

```
int main() {
    List *start = malloc(sizeof(List));
    if (start == NULL)
        return 1;
```

- At this point, the state is represented by a formula

$$(\_\mathrm{nil} = \mathrm{nil}) * (\mathrm{start} \mapsto \mathrm{alloc}_0) * (\mathrm{start} \neq \_\mathrm{nil})$$

Example of analysis results

- Then the whole list is allocated, up to a nondeterministic length

```
{ // construct a linked list of unknown size
    List *list = start;
    list->next = NULL;
    int nondeterministic;
    while (nondeterministic) {
        List *next = malloc(sizeof(List));
        if (next == NULL)
            return 1;
        list->next = next;
        list = list->next;
    }
    list->next = NULL;
}
```

- After this, the state is

$$\_\mathrm{nil} = \mathrm{nil} * \mathrm{ls}(\mathrm{start}, \_\mathrm{nil}) * \mathrm{start} \neq \_\mathrm{nil}$$

Example of analysis results

- After traversing the list, the state stays the same since the shape of the data structure did not change

```
{ // walk to the end of the list
    List *list = start;
    while (list != NULL) {
        List *next = NULL;
        next = list->next;
        list->data = 42;
        list = next;
    }
}
```

Example of analysis results

- At last, all nodes are deallocated

```
{ // free the list
    List *list = start;
    while (list != NULL) {
        List *next = NULL;
        next = list->next;
        free(list);
        list = next;
    }
}
```

- The state is changed to reflect that

$$(\_nil = nil) * (start \neq \_nil)$$

Error detection

- The analysis is not only able to detect the shape of data in a correct program, it can also detect memory errors
- If we try to access the list after deallocation, the bug is detected

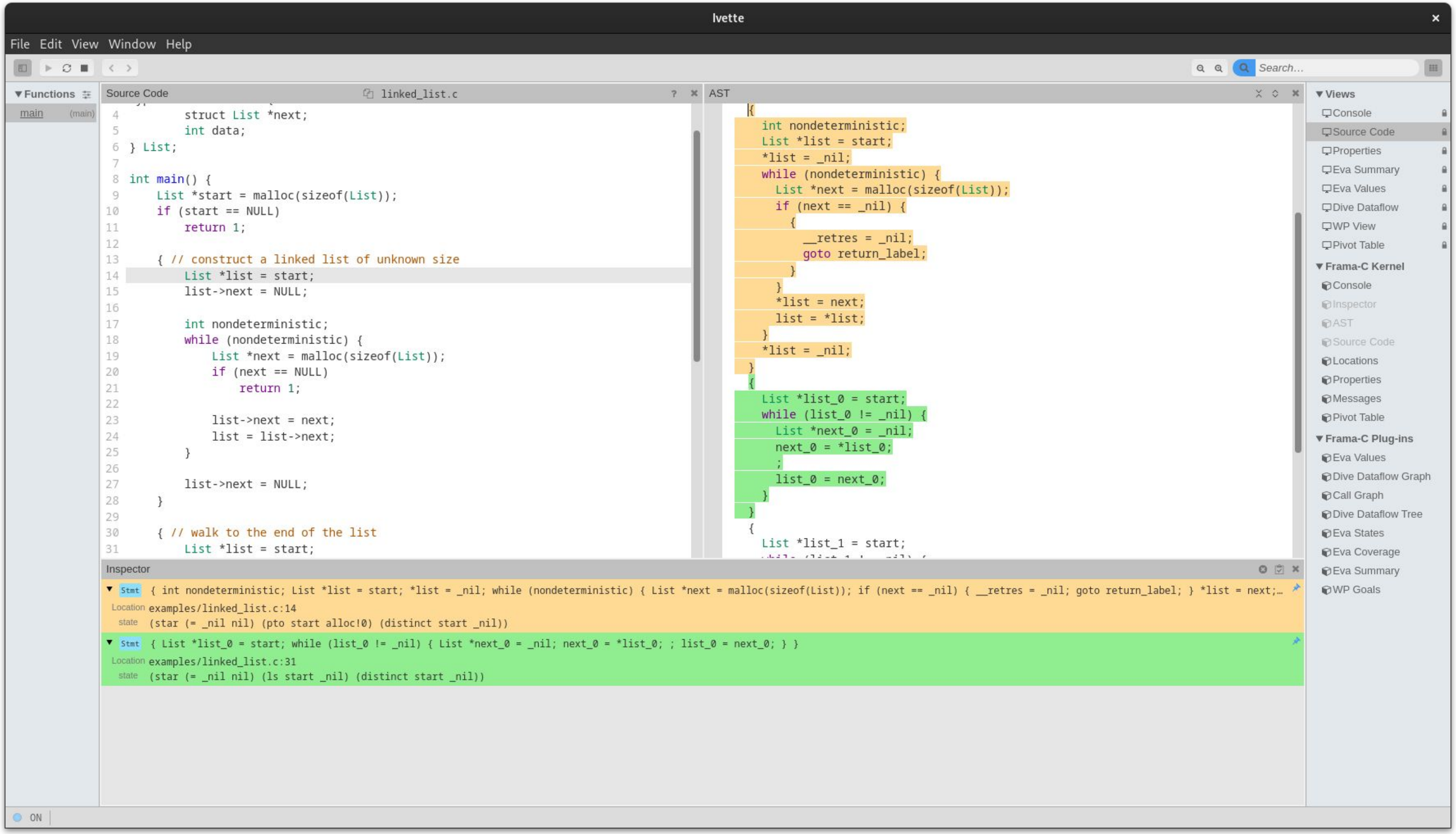
```
start->next = NULL; // use after free
```

- After adding this line below the deallocation, an error is raised

```
[SLplugin] examples/linked_list.c:54: Failure:
    detected a dereference of an unallocated variable
```

Results in Ivette

- These results can be viewed using the Ivette GUI extension



Future work

- Support the analysis of function calls, possibly with caching of results
- Preprocess complex statements to enable the analysis of more programs
- Support global and static variables
- Most importantly, extend the analysis to more complex data structures (doubly linked lists, nested lists)
- Benchmark the analysis and compare it to other solutions

Summary

- Studied topics: Frama-C framework, dataflow analysis, separation logic, Astral solver, shape analysis
- Implemented a prototype of shape analysis using separation logic, able to analyze simple programs that work with linked lists