Automated Memory Error Repair Based on Hybrid Program Analysis

ZECHANG QIAN
(Supervisor: Katsuhiko Gondow)

Abstract

Automated program repair is a technology which aims to fix program errors and vulnerabilities automatically. In the field of memory error repair, with the development of bug detection tools, we can easily detect memory errors in programs. However, fixing those errors is time-consuming and error-prone. Because the program ' s heap-related behavior plays a critical role in memory error repair, the existing techniques are mainly based on static analysis, where the static bug-finder is used to detect program memory errors and then repair tools collect the essential information via static analysis. But since static bug-finder may give wrong alarms which will affect the performance of the repair tools, and static analysis often requires high overhead.

*We present **HAMER**, a hybrid automatic memory error repair tool that aims to address those shortcomings by using hybrid analysis. HAMER first uses fuzzer to check the alarms given by the static bug-finder and extracts the real errors from those alarms. Then it tries to fix those errors by using hybrid analysis.*

I. Introduction

Memory errors, such as memory leaks, can have catastrophic effects, thus detecting and fixing them has always been a critical task for developers. Memory error detection performance is improving with the development of memory error detection technologies, however resolving these problems takes a lot of time and work for developers, and erroneous patches might lead to more significant effects.

Existing memory error repair techniques[2, 3] are mainly based on static analysis. This is because resolving issues like memory leaks necessitates an understanding of heap-related behavior, such as error source and sink. On the other hand, static analysis techniques might have a high time and space overhead, and they aren't always particularly good at dealing with problems such as indirect calls and alias. While dynamic analysis can help with these issues, it does not provide enough information to generate patches.

In this paper, we present **HAMER**, a hybrid analysis-based memory error repair technique. We use a static bug-finder to detect the program first, then use a fuzzer to detect the alarms and extract the real bugs. After that, we collect program variables that can be utilized to synthesize patches by variable dependency analysis. We also gather the test cases generated by the fuzzer which trigger or do not trigger the errors and then use a component-based program synthesis technique to try to generate patches from these variables and test cases. Finally, We utilize a fuzzer to detect the current fixed program, and if the repair is erroneous, we collect the test case that triggers the errors and repeat our repair method until the error is resolved or timeout. This strategy ensures that the patches generated by HAMER can repair current errors while without introducing new ones.

Contributions. This paper makes the following contributions:

- We present a new technique for repairing memory errors based on hybrid analysis.
- We present HAMER[1], a memory error re-

[1] https://github.com/QIANZECHANG/MyResearch

pair tool that implements the proposed approach.

II. Overview

We illustrate key features of HAMER and how it works.

i. Motivating Example

```
typedef struct N{
  int v;
}node;

node *new_node1(int a){
  node *n=(node*)malloc(sizeof(node));
  n->v=a;
  return n;
}

node *new_node2(int a){
  node *n=(node*)malloc(sizeof(node));
  n->v=a*a;
  return n;
}

int func(int a){
  node* (*p[])()={new_node1,new_node2};
  node *x;
  node *y=(node*)malloc(sizeof(node));//o2
  x=(*p[0])(a);//o0
  if(a<5){
    x=(*p[1])(a);//o1
  }
  x->v=10;
  return 0;
}
```

Figure 1: *o0, o1, o2 occur memory leak*

```
int func(int a){
  node* tmp_o0;
  node* tmp_o2;
  int tmp_a = a;
  node* tmp_o1;
  node* (*p[])()={new_node1,new_node2};
  node *x;
  node *y=(node*)malloc(sizeof(node));
  tmp_o2 = y;
  x=(*p[0])(a);
  tmp_o0 = x;
  if(a<5){
    x=(*p[1])(a);
    tmp_o1 = x;
  }
  x->v=10;
  if(tmp_a<=4)free(tmp_o1);
  free(tmp_o2);
  free(tmp_o0);
  return 0;
}
```

Figure 2: *HAMER-generated patch*

III. Approach

IV. Evaluation

V. Related Work

VI. Conclusion

References

[1] Xiang Gao, Bo Wang, Gregory J. Duck, Ruyi Ji, Yingfei Xiong, and Abhik Roychoudhury. Beyond tests: Program vulnerability repair via crash constraint extraction. *ACM Trans. Softw. Eng. Methodol.*, 30(2), feb 2021.

[2] Seongjoon Hong, Junhee Lee, Jeongsoo Lee, and Hakjoo Oh. Saver: Scalable, precise, and safe memory-error repair. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 271–283, 2020.

[3] Junhee Lee, Seongjoon Hong, and Hakjoo Oh. Memfix: Static analysis-based repair of memory deallocation errors for c. ESEC/FSE 2018, 2018.

[4] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. Angelix: Scalable multi-line program patch synthesis via symbolic analysis. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 691–701, 2016.

[5] Ridwan Shariffdeen, Yannic Noller, Lars Grunske, and Abhik Roychoudhury. Concolic program repair. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 390–405, 2021.