

Enhancing Symbolic Execution of Programs for Control Flow Path Feasibility Analysis

KLEE 2026 workshop

Hovhannes Movsisyan (hovhannes.movsisyan@rau.am)

Hripsime Hovhannisyan

Tigran Avagyan

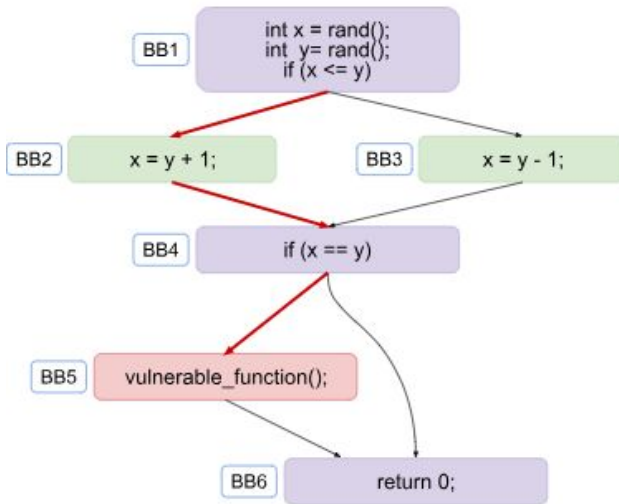
Hayk Aslanyan

Center of Advanced Software Technologies

Why verify a reported path?

Static analysis can report a vulnerability along an infeasible execution path.

```
1.  int main() {  
2.      int x = rand();  
3.      int y = rand();  
4.      if (x <= y)  
5.          x = y + 1;  
6.      else if (x > y)  
7.          x = y - 1;  
8.      if (x == y)  
9.          vulnerable_function();  
10.     return 0;  
11. }
```



Reported a vulnerability along *BB1* → *BB2* → *BB4* → *BB5*
Contradictory statements $x \leq y$; $x = y + 1$; and $x == y$;

Problem

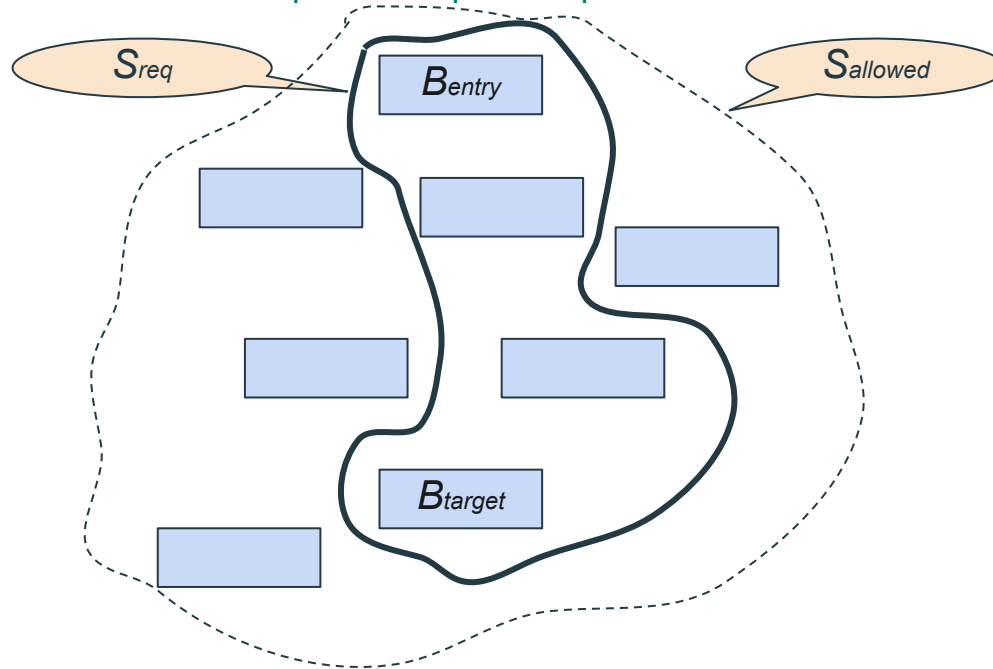
Check the feasibility of the given interprocedural control flow path using symbolic execution.

Agenda

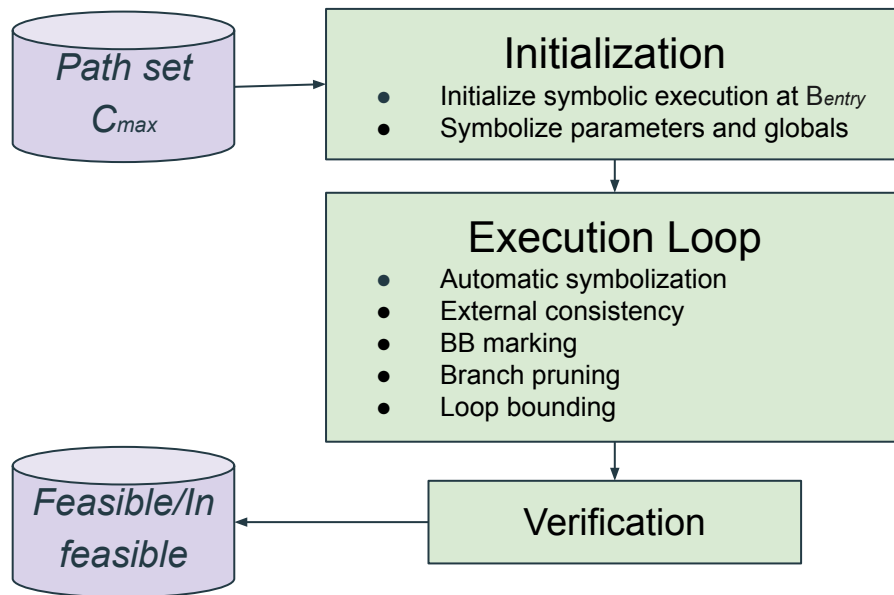
- Path feasibility algorithm
- Requirements for automated analysis
 - Automatic symbolization approaches
 - Memory model supporting symbolic pointers
 - Loop bounding strategy
- Experimental evaluation

Path feasibility algorithm: input path set

Deciding whether at least one path in a specified path set is feasible



Path feasibility algorithm



Automatic symbolization

Path feasibility analysis must be fully automated

- Entry-point function parameters
- Uninitialized local variables and memory locations
- External global variables
- External function return values
- External function pointer arguments
- **Symbolic pointers**

Memory model supporting symbolic pointers

- No predefined memory buffer under a symbolic pointer
- A single memory object is created during the first dereference
- Repeated accesses to the same address reuse its object; new accesses may create objects or fork over aliases.

```
1.  int foo(int *arr){  
2.      if (arr[5] < 10)  
3.          return 1;  
4.      else if (arr[5] > 10)  
5.          return 0;  
6.      else  
7.          return -1;  
8.  }
```

- Line 1: `arr` is a symbolic pointer
- Line 2: creation of `arr[5]`
- Line 4: uses the object `arr[5]` created before
- ...

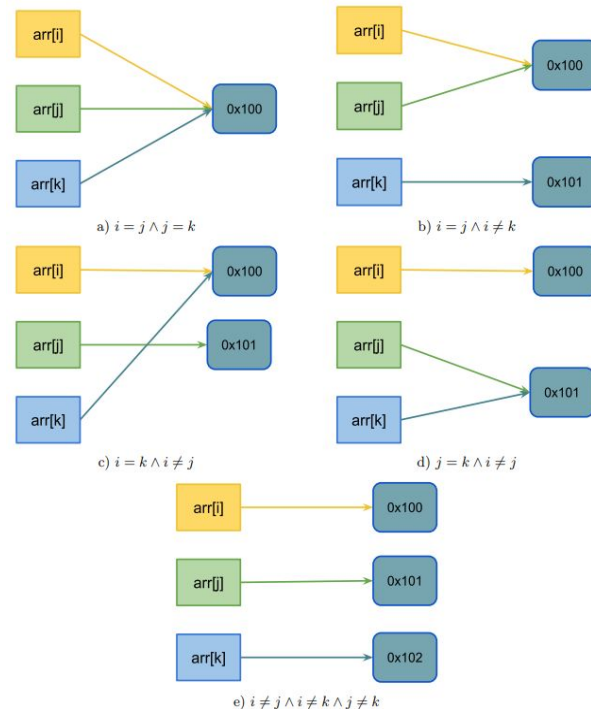
Modelling aliasing scenarios

```

1. void foo(int i, int j, int k, char *arr){
2.     char a = arr[i];
3.     char b = arr[j];
4.     char c = arr[k];
5. }

```

- Line 2: create a new memory object for `arr[i]`
- Line 3: `i != j`: create a new memory object for `arr[j]`
- Line 3: `i == j`: reuse `arr[i]` for `arr[j]`
- ...



Loop bounding strategy

Terminate long/infinite loops and continue symbolic execution

C_{max} - the maximum number of iterations.

- Track loop information
 - Header, exit edges, and iteration count
- Redirection of the symbolic state
 - If a loop header executes more than C_{max} times, the algorithm redirects the symbolic state using only exit edges whose source basic block is executed by the current symbolic state at least once.
- Memory resymbolization
 - Symbolize the memory locations written during the loop execution

Approximation: improves progress, but can admit spurious paths or miss feasible ones.

Implementation and evaluation

Implemented in KLEE

Experiment	Input	Question
GNU Coreutils	Recorded executable paths	Can the method verify real world feasible paths?
MLH + Juliet	Memory-leak reports	Can the method remove false positives and keep true positives?
Infer / CSA + Juliet	Use-after-free / double-free reports	How does verification compare with KLEEF?

GNU Coreutils

All Feasible Paths	234	
Verified	223	95.3%
Failed Cases	11	4.7%
Standard Execution	174	74.4%
Bounded Execution	49	20.9%
DFS Search Strategy	221	94.4%
BFS Search Strategy	2	0.8%

All samples are feasible traces. Precision is not measured.

MLH + Path Feasibility on Juliet CWE-401 (memory leaks)

Method	TP	TN	FP	FN	Precision(%)	Recall(%)	F1-score(%)
MLH	868	3,940	666	0	56.6	100	72.3
MLH + our method	868	4,606	0	0	100	100	100

666 false positives removed

56.6% → 100%

precision

100%

recall preserved

72.3% → 100%

F1-score

MLH + Path Feasibility on real-world projects

Accepted issue reports across five open-source projects

Project	Report numbers	Repository / project
OpenSSL	20870	github.com/openssl/openssl
FFmpeg	10342	lists.ffmpeg.org
radare2	21703, 21704	github.com/radareorg/radare2
BIND 9	4282	gitlab.isc.org/isc-projects/bind9
clib	292, 293, 295	github.com/clibs/clib

All listed reports were accepted by the maintainers.

KLEEF comparison: Infer reports on CWE-416 (use-after-free)

Method	TP	FP	Precision(%)	Recall(%)	F1-score(%)
Infer	96	0	100	100	100
Infer + KLEEF	0	0	0	0	0
Infer + our method	96	0	100	100	100

KLEEF classified 36 true-positive reports as infeasible and crashed on 60.

KLEEF comparison: CSA reports on CWE-415 (double-free) and CWE-416 (use-after-free)

Method	TP	FP	Precision(%)	Recall(%)	F1-score(%)
CSA	780	475	62.1	100	76.6
CSA + KLEEF	30	0	100	3.8	7.4
CSA + our method	780	473	62.2	100	76.7

A feasible path can still carry a false defect report.

This explains the 473 false positives retained by path verification.

Conclusion and Future Work

Conclusion

- path feasibility algorithm using directed symbolic execution
- automatic symbolization
- memory model
- bounded loop strategy
- experimental evaluation

Future work

- parallel analysis
 - split large complex paths into smaller segments and analyze them simultaneously
- search strategy
 - develop and implement a specialized strategy to prioritize and reach the endpoint more efficiently

Thank You



hovhannes.movsisyan@rau.am



[Our paper](#)



[Center of Advanced Software
Technologies](#)

Additional slides

Path feasibility algorithm: input, output

Deciding whether at least one path in a specified path set is feasible

Input:

1. First basic block of all paths' entry-point function (B_{entry}).
2. Last basic block of all paths (B_{target}).
3. The set of basic blocks representing the allowed execution scope ($S_{allowed}$).
4. A set of basic blocks representing the specific milestones that must be reached to consider the verification successful:

$$S_{req} = \{B_{entry}, B_1, B_2, \dots, B_{target}\} \subseteq S_{allowed}$$

5. Loop iteration bound C_{max} .

Output:

A boolean value indicating whether there is a feasible path that starts from B_{entry} , remains entirely within $S_{allowed}$, covers the set S_{req} , and reaches B_{target} .

Path feasibility algorithm: Description

- Preprocessing:
 - Extend $S_{allowed}$ until the first function's entry basic block. Replace B_{entry} with the entry basic block of the function.
- Initialization:
 - Initialize symbolic execution at B_{entry} .
 - Symbolize entry-point function parameters and external variables.
- Execution Loop:
 - Automatic Symbolization: Symbolize uninitialized memory locations, and environment-dependent values.
 - Basic Block Marking: Mark basic blocks executed from S_{req} .
 - Pruning: Prune symbolic execution if $B_{successor} \notin S_{allowed}$.
 - External Consistency: Re-symbolization of errno, return values, and memory locations referenced by pointer arguments of external functions.
 - Loop Iteration Limit: Terminate a loop if its header executes more than C_{max} times.
- Verification:
 - Feasible: B_{target} is reached and all basic blocks in S_{req} are marked
 - Infeasible: Symbolic execution terminates without reaching B_{target} , or without executing all elements in S_{req}

Single path analysis

Path: Let P denote a sequence of basic blocks in the program's ICFG:

$$P = \langle B_{\text{entry}}, B_1, B_2, \dots, B_{\text{target}} \rangle.$$

Where:

- B_{entry} is the first basic block of the path's entry-point function.
- B_{target} is the final basic block in the sequence.

In this case, we apply the algorithm where $S_{\text{allowed}} = S_{\text{req}} = \{B \mid B \in P\}$.

GNU Coreutils: cost varies with path complexity

Representative paths from GNU Coreutils; time includes symbolic execution and query solving.

Blocks	Instructions	Time (s)	Queries	Search	Bound
1,191	10,328,647	1,270	68,673	DFS	3
1,155	10,124,940	1,329	68,713	DFS	3
1,054	10,680,091	1,852	68,709	DFS	3
855	12,532,657	1,739	92,857	BFS	3
850	13,663,509	1,732	251,199	BFS	3
477	13,286	23	2,116	DFS	∞
308	12,845	25	2,267	DFS	∞

Search strategies: DFS verified 221 paths; 2 paths required BFS.