Software Target-Focused Flow Analysis

Harsha Deshmukh & Daniel Sokoler, Purdue University

Research Question

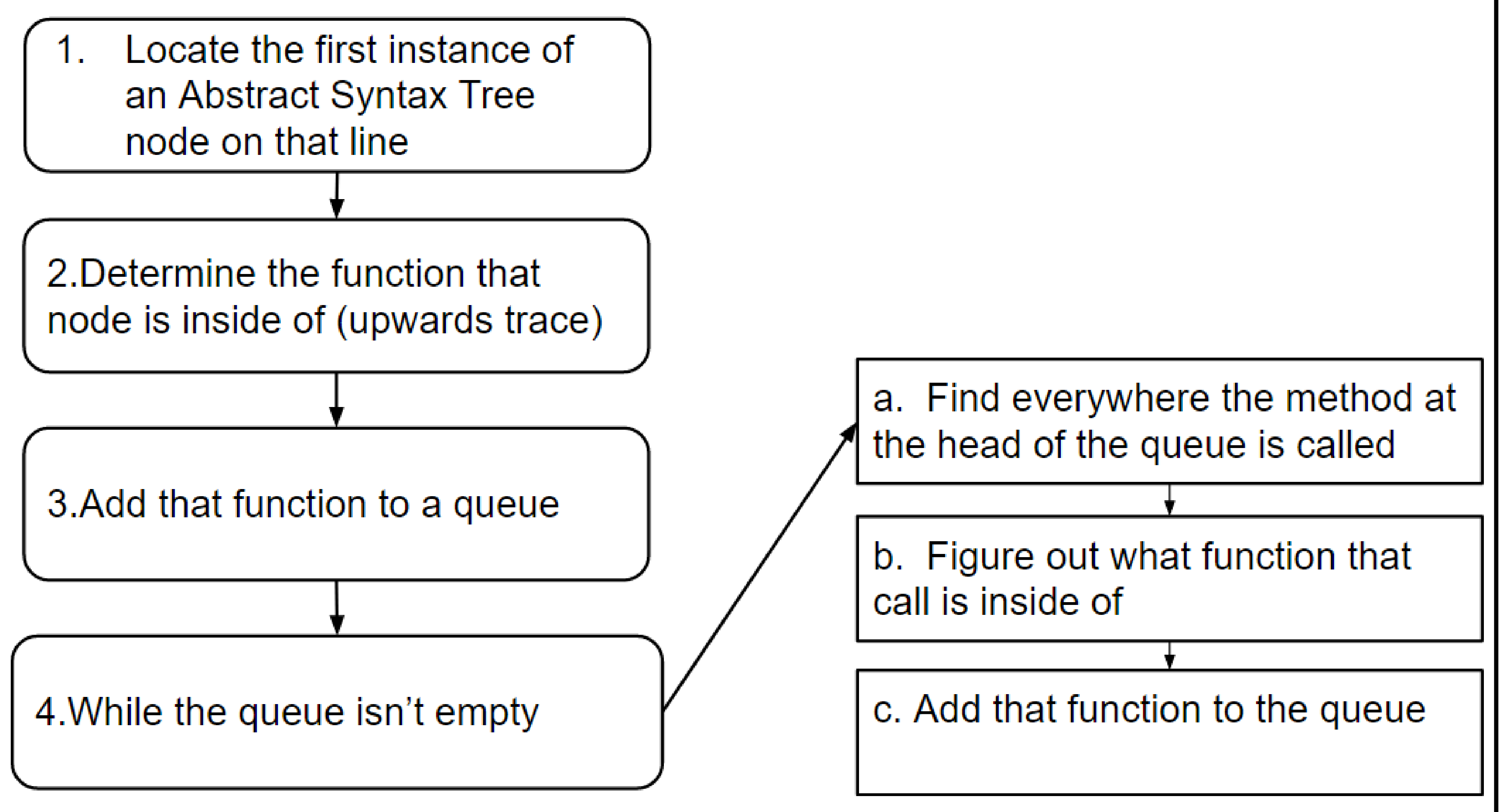
Given a set of source code written in the **C** programming language and a program point with known vulnerability, how can we derive flows to the **vulnerable target** from **start point** of the program?

Introduction

Existing parsers parse the C code to generate an AST of the **complete source code** , this prevents the user from specifying any determined program point(vulnerability) of interest to be analyzed for its reachability from the starting point of program. Thus, our tool will utilize the **AST** to backtrack all the paths from the vulnerable point. This assists in visualizing the reachability of a target point from a **code- security viewpoint**.

Approach

Our approach is a bottom-up one. Identify the vulnerable point by its line number and continually trace upwards to a function definition. When there are no more functions to trace, the flows have been identified.



1. Locate the first instance of an Abstract Syntax Tree node on that line
2. Determine the function that node is inside of (upwards trace)
3. Add that function to a queue
4. While the queue isn't empty
   - a. Find everywhere the method at the head of the queue is called
   - b. Figure out what function that call is inside of
   - c. Add that function to the queue

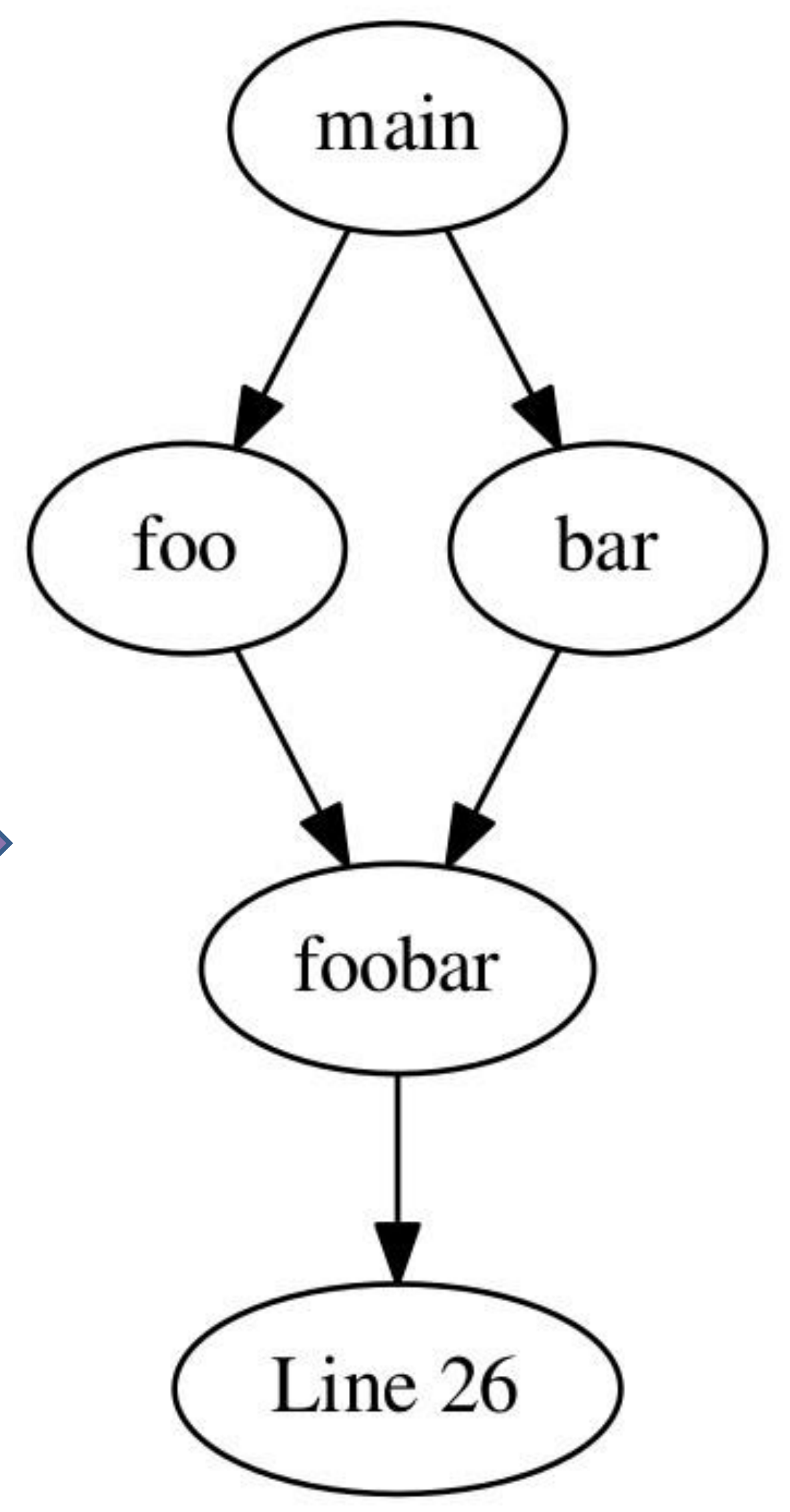
Discussion & Results

Our tool can currently identify flows at the function level. As seen in the below sample code, we are expecting two flows :

$$\text{Main} \rightarrow \text{foo} \rightarrow \text{foobar} \rightarrow \text{vulnerable point}$$
$$\text{Main} \rightarrow \text{bar} \rightarrow \text{foobar} \rightarrow \text{vulnerable point}$$

The Control Flow Graph on the right is the current automated output of our tool.

```
int main(int argc, char* argv[]) {
        int x = 5;

        if (x > 10) {
                foo(argv[1]);
        }
        else {
                bar(argv[1]);
        }

}

void foo(char* b) {
        foobar();
}

void bar(char* f) {
        foobar();
}

void foobar() {
        int vuln = 5;
}
```



Conclusion

Thus, we working towards developing a control flow analyzer to enumerate the flows that reach the vulnerability.

References

Allen, Frances E. (1970, July). Control Flow Analysis. In ACM Sigplan Notices (Vol. 5, No. 7, pp. 1-19). ACM

Khedker, U. P., & Dhamdhere, D. M. (1994). A generalized theory of bit vector data flow analysis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *16*(5), 1472-1511.

Cheatham, T. E., Holloway, G. H., & Townley, J. A. (1979). Symbolic evaluation and the analysis of programs. IEEE Transactions on Software Engineering, (4), 402-417.

This research is a part of the Information Security Research and Education (INSuRE) project. INSuRE is a partnership between successful and mature Centers of Academic Excellence in Information Assurance Research (CAE-R) and the National Security Agency (NSA), the Department of Homeland Security and other federal and state agencies and laboratories to design, develop and test a cybersecurity research network. INSuRE is a self-organizing, cooperative, multi-disciplinary, multi-institutional, and multi-level collaborative research project that can include both unclassified and classified research problems in cybersecurity.