Program Analysis: Towards the Analysis of CPython Bytecode

Progress Report

André Theuma

Supervisor: Dr. Neville Grech

Faculty of ICT

University of Malta

17/12/2021

Submitted in partial fulfillment of the requirements for the degree of B.Sc. I.C.T. (Hons.)

Abstract

Software systems are heavily relied on by businesses for their day to day management and are also used to protect critical data. In this day and age having a secure system is more important than ever before, thus one must ensure that the system's security is tested out and analysed beforehand.

Program Analysis is a process by which a system's correctness may be tested. In this dissertation, a Static Analysis Framework shall be implemented for its use in the CPython environment.

Contents

1. Introduction & Motivation

Program analysis is the (now automated) process by which a program can be analysed [7]. The results produced by the analysis can then be used for one of two things; Program Optimization or Program Correctness. There are two different types of analyses, Static analysis and Dynamic analysis. Typically both analyses are used in conjunction with each other, but in different production phases.

Static analysis is used in the development phase of a program, prior to testing. With this analysis, the root cause of many basic vulnerabilities is found.

Dynamic analysis is performed during runtime and is very resource intensive, in comparison to Static analysis. Dynamic program analysis is more commonly known as correctness runtime debugging. [2]

In this dissertation, the main topic of focus shall be on disassembling CPython source code into CPython bytecode and performing a relevant analysis on the produced relations. CPython is the most widely used implementation of Python, which is one of the most popular languages. Its use ranges from web-development to enterprise solutions and embedded development. As of today, it is rated as one of the languages used mostly by developers and in several industries [9]. The relations produced after the disassembly will then be broken down into a more elementary form of relation which is simpler to digest, for the user. Different queries will then be performed on these new relations, depending on the use cases needed, finally producing facts. The query system will be of a Soufflé Datalog implementation so as to facilitate the process.

1.1 Non-Triviality

Performing Static Program analysis is non-trivial as it forms an essential part to providing secure software and protecting data from possible attackers. Program correctness essentially provides data confidentiality and data integrity. [6]

2. Background Research & Literature Review

The project is split into two main sections. The disassembly of functions into bytecode with it being processed by the bytecode analyser would form part of the first section, and the Soufflé queries would pertain to the second section of this project.

2.1 Bytecode

In CPython we find that the source code is firstly translated into an intermediate language, known as the `bytecode`, and is then passed through and executed by the virtual machine (PVM) [4], which is written in C, hence CPython. The `bytecode` is exposed by the use of the `dis` module provided by Python [3]. The `dis` module provides the user with the bytecode API wrapper, in which multiple functions may be used to exploit different properties of the source code. For bytecode analysis these functions are used from the aforementioned module; `dis.Bytecode()` [3] and `dis.dis()` [3]. The former iterates over each bytecode instance as an `Instruction` instance [3]. The latter process the source code line by line, and disassembles every object into its corresponding bytecode representation. The bytecode analyser that will be implemented, will disassemble methods and classes. When a class is passed to the disassembler, the disassembler disassembles all the methods in the class object, printing every bytecode instruction on a new line. The `Instruction` instance holds the following attributes; `opcode`, `opname`, `arg`, `argval`, `argrepr`, `offset`, `starts_line`, `is_jump_target` [3].

Each attribute contains information which is relevant to the current bytecode instruction.

2.2 Soufflé

The second part of the project relies heavily on Soufflé which is an open source framework, which performs static program analysis to generate facts [5]. Soufflé is extremely fast in comparison to SQL/Datalog and has very simple syntax, which makes it a very powerful and user friendly tool. The language is based upon Datalog but has state of the art performance [5], allowing analysis to be performed on large libraries, which previously would be infeasible.

Soufflé's fast performance may be credited to its method of evaluating Datalog programs. Soufflé synthesizes and compiles the Datalog program into native C++ code. This is possible by firstly compiling the Datalog program into Relational-Algebra-Machine Programs (RAM) RAM programs are evaluated by elaborate data indexing techniques and multi-threaded query processing. These RAM programs can then be compiled into C++. This is further compiled into machine code, which is what allows for such fast execution. [8]

3. Methods & Techniques Planned

The Methodology planned to carry out this project starts off by creating the CPython bytecode analyser. This analyser will disassemble any function or class object passed to it and its bytecode instructions will be inspected and analysed. The PVM (Python Virtual Machine) stack operations will be taken note of after every bytecode instruction. From this disassembly facts will then be created. The facts produced will form part of the following spec; a fact might make reference to the value pushed to the stack (`PushValue`), its code object reference (`Statement_Code`), a reference to the next instruction (`Statement_Next`), a reference to the instruction's opcode (`Statement_Opcode`), and a reference to the amount of operations performed on the stack (`Statement_Pushes, Statement_Pops`).

These facts are to be stored in a Datalog compatible format (csv) so that a Datalog analysis can be performed on them. The type of analysis that will be done is still yet to be discussed.

4. Evaluation Strategy

The two main evaluation techniques are the following;

- Evaluate bytecode, analyse each bytecode instruction & produce facts.
- Perform meaningful queries on facts generated by the bytecode analyser.

From these evaluation techniques, a report might be produced illustrating the queries & their results.

5. Project Deliverables

- A Final Year Project Report, including any relevant material used in the project.
- The implementation of the CPython Bytecode Analyser, & the function or class objects used.
- The facts generated, the Datalog queries along with their results.

References

[1] J. Aycock. Converting python virtual machine code to c. In *Proceedings of the 7th International Python Conference*, pages 76–78, 1998.

[2] B. Bruegge, T. Gottschalk, and B. Luo. A framework for dynamic program analyzers. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 65–82, 1993.

[3] P. S. Foundation. dis — disassembler for python bytecode, 2021.

[4] O. Ike-Nwosu. Inside the python virtual machine, 2015.

[5] H. Jordan, B. Scholz, and P. Subotić. Soufflé: On synthesis of program analyzers. In *International Conference on Computer Aided Verification*, pages 422–430. Springer, 2016.

[6] A. Møller and M. I. Schwartzbach. Static program analysis. *Notes. Feb*, 2012.

[7] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis.* Springer Science & Business Media, 2004.

[8] R. Shahin and M. Chechik. *Variability-Aware Datalog*, pages 213–221. 01 2020.

[9] I. Spectrum. Top programming languages 2021, 2021.

[7] [2] [6] [9] [1] [4] [3] [5] [8]