Verification Of Practical Go Programs

Bachelor Thesis Project Description

Luca Halm
Supervised by Prof. Dr. Peter Müller, Felix Wolf, João C. Pereira
Department of Computer Science
ETH Zürich
Zürich, Switzerland

I. Introduction

Go [1] is a compiled, concurrent, statically typed programming language designed by Google. It was designed to simplify the development of software for multiprocessor and network systems. Several companies use Go to provide parts of their service, including Google itself, Netflix, and Dropbox.

The Go programming language is accompanied by a sophisticated testing library. Nonetheless, Go programs still suffer from various kinds of bugs that jeopardize memory safety and functional correctness.

Software verification can help in reducing the risk of unexpected software faults by proving that the program behaves according to some formal specification.

Gobra [2] is a modular verifier for Go programs and includes support for Go's built-in concurrency primitives, most prominently, channels and goroutines, but also mutexes and wait-groups. Gobra works on Go code that is annotated with preconditions, which must hold before the function gets invoked, postconditions, which must hold after the function execution, and proof annotations. Gobra then encodes the annotated source code to a program in the Viper [3] intermediate verification language which gets processed by the Viper toolchain. If verification is successful, the user gets a corresponding message, and in case of a verification failure, Gobra translates the Viper error message back to an error message at the Go level.

The main goal of the project is to evaluate and improve Gobra's ability to verify large Go projects. To that end, we will verify a crucial part of the Go implementation of the SCION [4] protocol. SCION is a new internet architecture designed and developed at ETH Zürich. SCION was developed to be less vulnerable to attacks and to significantly decrease the chance of severe disruptions of the internet. We will verify SCION code in multiple steps. First, the SCION code that we verify needs to be annotated with access permissions to prove memory safety. Then, a functional specification can be added. Finally, Gobra can be used to prove the given specification's correctness. Since the SCION project has a large codebase and is used in practice, we expect that the part that we will verify in the context of this project will utilize all the features that Gobra has to offer.

So far, Gobra has not been applied to a real-world software project of considerable size. Therefore, we do expect some challenges to arise during the project, for example missing type implementations for runes and floats, or difficulties in handling third-party libraries. Gobra does have features to support libraries, however, they have not yet been stress tested. We also expect that Gobra can be greatly improved in terms of productivity by reducing the annotation overhead. For instance, consider nested loops, where the loop invariants of the outer loop often need to be copied to the inner loop. Similarly, many assertions occur as pre- and postconditions and thus have to be copied multiple times. In both of these examples, duplication of annotations can be avoided by introducing a new keyword.

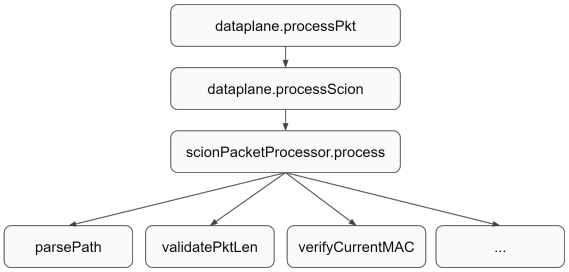

Fig. 1. Call graph of the SCION code to be verified

II. Context

SCION is a new internet architecture that aims to improve on most of the problems that the current internet has, in particular, routing problems with the BGP protocol or DDoS attacks. Unlike current solutions, SCION packets specify not only the intended destination of the packet but also the paths that can be taken by the packet in the network. A path consists of a list of hops that the packet has to traverse to get to the destination. Because the sender is responsible for picking the paths for its packets, there is no need for long inter-domain routing tables and complex longest IP prefix matching. Therefore, forwarding packets is more efficient because the router simply has to

look at the next hop in the path of the packet and send it there.

The SCION architecture can be separated into a control plane and a data plane. The task of the control plane is to discover paths and to make them available to end hosts, essentially providing end-to-end paths through the network. The data plane uses these paths to forward packets to the desired destination. This project will focus on verifying parts of the implementation of the data plane. In particular, the code that we will verify reads an incoming packet, checks whether the header is valid, extracts the next hop from the path, updates the header, and forwards the packet to the next hop.

III. Goals

The overarching goal of this thesis is to verify a substantial part of the SCION data plane. Furthermore, we take this opportunity to improve the productivity of Gobra when applied to real code.

1) First, we will apply Gobra on an interesting small part of the data plane (around 100 lines of Go code). The chosen part should satisfy that the insights gained from verifying that code can be extrapolated to the rest of the data plane. During this process, we will compile a list of features that are useful in terms of productivity, or necessary, for Gobra to be applied to real code of considerable size. The goal is not to list every such feature but in particular, those features which we will add in the scope of this thesis. A preliminary list, without the case study being done, inspired by existing features or other verification tools, is as follows:
   - More fine-grained control over when definitions of functions and predicates are unrolled. This reduces the number of facts the verification backend has to deal with.
   - Support for multiple specifications for a single function to give the user the choice between specifications of different complexity.
   - Different options for specification propagation. For example, we could introduce a keyword that specifies that an assertion is a precondition as well as a postcondition, or that an assertion is part of all invariants of a method.
   - Support for inlined code specifications to avoid moving code into separate methods. For verification, moving code into separate methods has the advantage of simplifying the individual proof obligations, but it significantly changes the code.
   - A feature to split an encoded Viper program into several different Viper programs. The aim is to reduce the number of domain and function definitions for each individual resulting Viper program.
   - A search engine for lemmas.
2) The second goal is to implement the most relevant features on the list resulting from the first goal.
3) The third goal is to verify memory safety, race freedom, and crash safety on a section of the code responsible for packet manipulation and forwarding, as shown in figure 1. Additionally, we will pick and verify suitable data structure invariants. We will add assumptions to keep the verification within the scope of a bachelor thesis. For example, we will assume that all paths have type SCION and that the crossover functionality is never used. We will document all assumptions and discuss their impact in our report. Furthermore, we allow us to modify the code if it is needed for the verification process.
4) The fourth goal is to add functional specification to a small part of the data plane and to document the process of coming up with both the specification and the verification result.
5) The last goal is to evaluate our solution. We will investigate which language features the verifier spends a lot of time on and if this overhead can be easily avoided by slightly modifying the code or the specification. Furthermore, we will analyse the impact of different verification approaches on the verification process, in particular, verification time and specification difficulty.

IV. Extension Goals

1) The first extension is to come up with I/O contracts and verify them. I/O specifications are used to specify which packets are allowed to be sent to the network. I/O specifications are useful to verify that an implementation adheres to a protocol.
2) The second extension goal is to implement more of the features of the list resulting from goal 1 and not implemented in goal 2.
3) The last extension goal is similar to the first two core goals, but with a focus on IDE features. We will come up with a set of IDE features that improve productivity and implement the most relevant features into the Gobra IDE. Furthermore, we will textually evaluate the benefits of our additions.

References

[1] Google LLC. Go programming language. [Online]. Available: https://golang.org/

[2] ETH Zürich. Gobra. [Online]. Available: https://www.pm.inf.ethz.ch/research/gobra.html

[3] P. Müller, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. [Online]. Available: http://pm.inf.ethz.ch/publications/getpdf.php?bibname=Own&id=MuellerSchwerhoffSummers16.pdf

[4] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat, *SCION: A Secure Internet Architecture*. Springer International Publishing AG, 2017.