Static Verification of the SCION Router Implementation

Bachelor's Thesis Project Description

Sascha Forster
Supervised by Marco Eilers, Prof. Dr. Peter Müller
Department of Computer Science
ETH Zürich
Zürich, Switzerland

1 Background

1.1 SCION

In today's Internet architecture there are many design flaws that cause problems. To name a few:

- DDoS attacks are very commonly used to make a service or website unavailable
- The BGP protocol causes routing problems as it can be unstable at times
- Security of TLS is questionable, as many certificate authorities (CAs) exist. Additionally, when a CA is compromised, it can take a long time to remove them from the list of trusted roots.

The SCION architecture [1] aims to improve on most of these problems. Most relevant to this project is that SCION does not use BGP, instead it allows clients to choose the route of packets they send. The SCION routers simply forward packets along the route that is defined in the header. This allows SCION routers to be stateless, because they do not need to keep ever growing routing tables. That is beneficial, because looking up the next hop in the header is cheaper than searching for it in the routing table using prefix matching. Furthermore, it prevents hijacking of traffic, avoiding eavesdropping by third parties with malicious intent.

1.2 VerifiedSCION

VerifiedSCION[1] is a joint project of the Chair of Programming Methodology, the Information Security Group and the Network Security Group. It aims to verify the routing protocol SCION, by first verifying that the protocol has certain guarantees, assuming it is implemented correctly, and then showing that the router is implemented correctly. This project aims to contribute to the verification of the implementation.

1.3 SCION Router

As in IP today, the SCION router receives a bit-string, which it needs to parse to make it easier to work with. If the received bit-string is a well-formed SCION packet, it gets parsed successfully. Then, the router will perform some additional processing on the packet to check if it is valid. SCION packets include the entire path from the source to the target address as a hop list in the header. The router looks up the next hop, increases the index for the hop list and then forwards the packet.

2 Router Verification

The aim of this project is to statically verify a part of the SCION router implementation. This means proving safety (no out of bounds errors, no data races),

[1]`http://www.pm.inf.ethz.ch/research/verifiedscion.html`

liveness (no infinite loops) and functional correctness, i.e., the implementation adheres to the specification of SCION. In particular, functional correctness requires proving correct behaviour if a SCION packet is received by the router. Correct behaviour means processing the packet according to the specification and then forwarding it to the next hop. To limit the scope of this project, we choose one path in the router code. Namely, the case when processing a packet whose path has a single up-segment.

2.1 Specification

During verification, it will be a challenge to find the right specification and more specifically the right invariants that are established by the implementation. Specification means annotating the code with concrete pre- and postconditions and loop invariants.

With the right specification using the permission system of Viper one can show that all memory accesses are performed only when a method or function has sufficient permissions to the accessed memory locations. To prove termination, obligations as described in [2] will be used. For showing correct I/O behaviour, a specification according to a Petri net will be used, shown in Figure 1.

In order to write the I/O specification, we define an abstract data type (ADT), which represents a SCION packet. The ADT will only include information that is necessary to show correct I/O behaviour of the router. Using an ADT allows us to get a simplified view of the parsing process. More specifically, we define a function toADT that is equivalent to reversing the parsing of a packet and abstracting the resulting bit-string. We declare, but do not define, functions abstract and concretise, to go from bit-strings to the ADT and back.

In Figure 1, a bit-string is received and stored in bs. Then, if bs is well-formed, the router processes bs to create a bs' which is then forwarded. Otherwise, the packet gets dropped.

Now, assume we have a function $ps = p(x)$ that abstracts the bit-string into the ADT. Instead of $wf(bs)$ and bs', we use $wf(ps)$ and ps'.

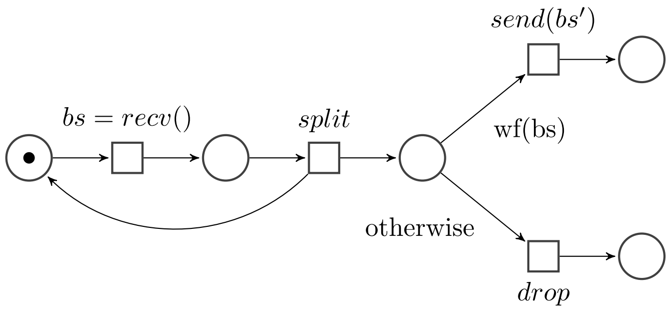

Figure 1: Simple depiction of the Petri net used for I/O specification where:
- bs is the bit-string that is received
- $wf(x)$ says if x is a well-formed SCION packet
- bs' is the bit-string after some processing

2.2 Nagini

Verification of the Python implementation will be performed using the Nagini verifier[2], which is a Python front-end for the Viper [3] verification framework. The tool-chain starts with Nagini, which translates Python code to the Viper language. Then there is a choice between using two different verification techniques, symbolic execution and verification condition generation.

It is also a part of this project to evaluate the performance and correctness of Nagini. If any issues with Nagini arise, we will try to identify and reproduce the bug. However, it is not part of this project to fix bugs in Nagini.

As for library functions that are used in the code, we assume that they are correct. Similarly, we initially assume the correctness of some parts of the SCION code, e.g., the packet parsing code. This means that we will use stubs to represent these functions, as implementation details are not important. Nonetheless, we must annotate them with appropriate contracts such that verification can succeed.

3 Core Goals

1. Prove memory safety of the router by showing

[2]`https://github.com/marcoeilers/nagini`

that no illegal memory accesses can happen. We will attempt to patch any illegal memory accesses in the current implementation, unless the work required exceeds the scope of this project.

2. Prove absence of unintended infinite loops. This means using loop invariants with obligations to show that progress is guaranteed for the router. Due to the nature of a router, there is a loop that repeats the receiving of packets indefinitely. The idea is to show that within that main loop, there are only loops that terminate.

3. Create an abstraction of the packet parsing mechanism of the router and declare a function that maps a SCION packet to the abstraction. The ADT described in Section 2.1 will be used to map the packet to.

4. Write an I/O specification [4] for the SCION router and verify that the implementation complies with it. The Petri net from Figure 1 and the ADT from the previous task will be used for this task.

4 Extension Goals

- Prove correctness of the parsing implementation of the SCION router Python code using the Nagini verifier. Subsequently, we could replace the stub for the parsing with the actual implementation.

- In the core goals, the focus lies on verifying correctness of a specific path in the router code. As an additional step, we could try to generalise this approach to verify other paths that the code can take.

References

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

[2] P. Boström and P. Müller, "Modular Verification of Finite Blocking in Non-terminating Programs," in *29th European Conference on Object-Oriented Programming (ECOOP 2015)* (J. T. Boyland, ed.), vol. 37 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 639–663, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[3] P. Müller, M. Schwerhoff, and A. J. Summers, "Viper: A verification infrastructure for permission-based reasoning," in *Proceedings of the 17th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 9583*, VMCAI 2016, (New York, NY, USA), pp. 41–62, Springer-Verlag New York, Inc., 2016.

[4] V. Astrauskas, "Input-output verification in viper," Master's thesis, Department of Computer Science, ETH Zürich, 2016.