CellScope: Automatically Specifying and Verifying Cellular Network Protocols

Yinbo Yu[†‡], You Li[‡], Kaiyu Hou[‡], Yan Chen[‡], Hai Zhou[‡], and Jianfeng Yang[†*]
[†] School of Electronic Information, Wuhan University, China
[‡] Department of Computer Science, Northwestern University, USA
{yyb,yjf}@whu.edu.cn,{youli,kyhou}@u.northwestern.edu,{ychen,haizhou}@northwestern.edu

CCS CONCEPTS

• **Networks** → **Protocol correctness**; Mobile and wireless security; • **Security and privacy** → *Logic and verification.*

KEYWORDS

Cellular network; protocol specification and verification; model checking

1 INTRODUCTION

Towards a secure cellular network, researchers are spending efforts to manually identify vulnerabilities, *e.g.*, [3] (here we call this approach as *human investigation*). Such a practice demands both time and patience. It is only suitable to be applied on a few procedures of large cellular network protocols. *Dynamic testing* [2], on the other hand, invokes test cases to check if the actual behavior of the network meets security criteria. It identifies vulnerabilities during execution. This approach, however, is depending on the quality of input test cases. As a result, its completeness can never be proved.

The use of formal methods brings a systematic and solid approach to cellular network security research [1, 4]. Nevertheless, researchers have encountered a common and critical problem: *specification*. Protocols of cellular network are documented in natural languages. Lots of human efforts are required to convert protocols into formal models. Such *manual-crafted specifications* are error-prone and only capable of describing small pieces of protocols.

Being different from existing methods (see Table 1), CellScope automatically extracts formal models of cellular network protocols from software implementations. By applying improved counter-example-guided abstraction refinement (CEGAR), CellScope efficiently inspects cellular network protocols in large scale.

Three major challenges are posed to our CellScope framework. First, there are multiple software entities run independently in a cellular network. Therefore, we setup software message channels to deliver plain messages. Second, the state space in an implementation of cellular network is too large for a model checker. For instance, a prevailing open source implementation of LTE, OpenAirInterface (OAI), has more than one million lines of code. We abstract low-layer protocols and build straightforward message channels between high-layer protocols. Program slicing is then applied to pruning program states according to used properties. Third, general purpose software model checkers cannot fully exploit information from specification. We figure out ways to exchange information between the frontend and the backend, such that the verification process can be accelerated.

*Jianfeng Yang (yjf@whu.edu.cn) is the corresponding author.

SIGCOMM '19, August 19–23, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6886-5/19/08…$15.00
https://doi.org/10.1145/3342280.3342294

Table 1: Existing methods vs. CellScope.

Methodology	Generalizable	Trustworthy	Complete	Automated
Human investigation	○	●	○	○
Dynamic testing	◐	◐	◐	●
Manual specification	◐	◐	◐	◐
CellScope	●	◐	●	●

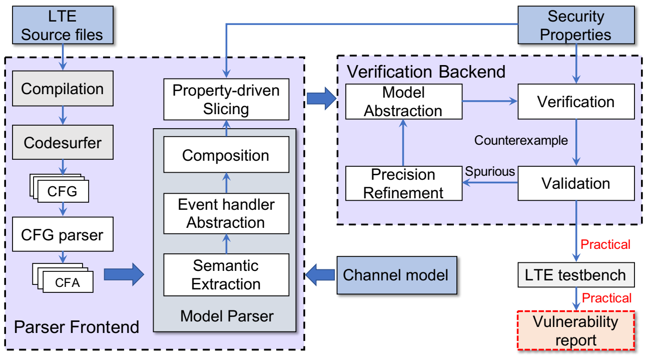

Figure 1: The architecture of CellScope.

In summary, we present our specification and verification framework, CellScope (shown in Figure 1), for exploiting design flaws in cellular network protocols. CellScope consists two major components: a *parser* frontend and a *verification* backend. The parser frontend automatically translates LTE source code to formal models and instruments channel and adversary models into LTE models. By slicing the model depending on a property, CellScope performs the CEGAR-based verification backend on it. Once a counterexample is identified, it will be validated on a real OAI testbed platform.

2 AUTOMATED SPECIFICATION

A cellular network has three major entities: user equipment (UE), base station (eNB), and core network (CN). We model each entity as a control flow automata (CFA). A CFA consists of (1) integer variables, (2) control locations with start and exit points, and (3) directed edges between locations. It is used in *verification backend* as the formal specification. Leveraging a program-understanding tool, CodeSurfer, along with code compilers, CellScope extracts

CFGs from source code. Then, it uses our implemented CFG parser to translate CFGs to CFAs.

2.1 Model Construction

Entities in the cellular network run independently. To facilitate model checking, we design a software message channel for message exchange between entities and for fusing the entities together. We focus on detailed procedures in protocols above the link layer (*e.g.*, NAS, RRC, and S1AP), while we abstract the protocol stack below that layer. With such a channel, CellScope can construct a complete yet simplified model for the whole cellular network.

As shown in Figure 2, our message channel model has two functionalities. First, the channel is built to simplify message exchange. Messages are encoded and exchanged between event handlers either within an entity (through multi-thread interface calls) or between entities (through physical signals). To facilitate model checking, the channel caches plain text message instead of bitstreams generated by one handler. The message is then translated as an input to the target handlers. Second, the channel is used to express adversarial behavior. We consider a Dolev-Yao-style adversary in the channel model with the following capabilities: it can drop or modify message in the channel or impersonate one legitimate participant to inject messages into the channel. These capabilities are implemented with operations on message caches and message exchanges in channels.

2.2 Property-driven Slicing

CellScope focuses on interactions among three major entities in RRC and NAS layer protocols. In that sense, it sets the slicing criteria to be a set of program instructions that send and receive messages. To reduce the sizes of the models, we design a property-driven slicing approach. Our approach slices models guided by each individual property to be checked by our verification backend.

Depending on the specific property, the slicing criteria used by CellScope may include configurations of entities, messages appear in protocols, and delivery instructions in between protocols and entities. Starting from each slicing criteria, CellScope performs backward slicing and tracks both intra-process and inter-process, as well as inter-entity. The more precise the slicing is, the more states it can prune statically. Nevertheless, to avoid false positives in model checking, CellScope performs an over-approximated slicing instead of precise point-to analysis. Avoiding the very expensive point-to analysis makes it possible for CellScope to prune the system model for each individual property.

3 FORMAL VERIFICATION

A major challenge in formal verification is the scalability problem: the model checker may not be able to terminate when the model gets large. This problem is more severe, as our model is extracted from a largest software. To address this challenge, we propose two new techniques: prioritized counterexample guided abstraction refinement (P-CEGAR) and model decomposition with weakest precondition.

3.1 P-CEGAR

As the original concrete model could be too complex to handle, in CEGAR, a *sound* yet *incomplete* abstract model is built. If the abstract model is safe, the concrete model should also be safe. However, when a counterexample is found on the abstract model, it is either feasible on the concrete model, or a *spurious* example due to a high abstraction level. Therefore, CEGAR further refines the abstract model by generating new predicates to rule out the current spurious example. A number of heuristics have designed to generate predicates. Unfortunately, because the specification is usually written by humans, none of the heuristics are using any knowledge in addition to the explicit model itself.

As a united platform, CellScope is able to share knowledge in between its specification and verification parts. Particularly, when constructing CFAs, CellScope distinguishes *protocol related variables* from program control related variables. In addition, *dummy adversary variables* are recorded by CellScope when building message channels. An abstract model consists of these variables can largely capture the behavior of the underlying protocol yet remains small in size. Therefore, after finding the set of contradicting predicates, our model checker prioritizes more essential variables over other variables. In this way, CellScope captures the essence of the concrete model and selects new predicates more wisely. Consequently, the model checking will terminate faster.

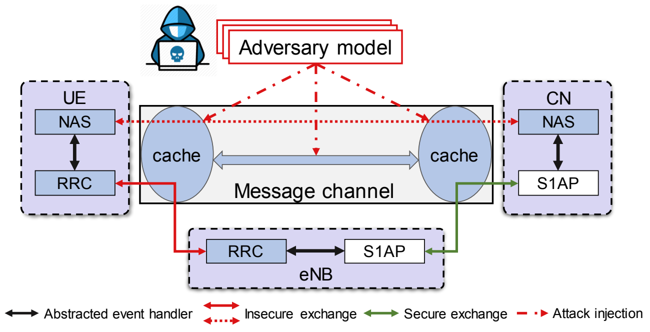

Figure 2: Message channel model.

3.2 Model Decomposition with Weakest Precondition

The execution time of verification grows fast as the size of the model grows. Nevertheless, formal models are always entangled, making decomposing them into separate ones infeasible. Cellular network models, on the other hand, can be easier decomposed into protocol layers. Meanwhile, a single layer can further be divided into function modules. The interactions in between are limited to a few messages.

To verify a safety property, CellScope starts from verifying the function module in which the violation to the property can occur. Then, CellScope proceed by constructing the set of weakest preconditions on the interface between modules, from which the violation can be reached. In the same way it propagates backward, until reaches the initial module.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. 2018. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In *Network and Distributed Systems Security (NDSS) Symposium 2018*.
[2] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. 2019. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. In *IEEE Symposium on Security & Privacy (SP)*.
[3] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. 2019. Breaking LTE on layer two. In *IEEE Symposium on Security & Privacy (SP)*.
[4] Guan-Hua Tu, Yuanjie Li, Chunyi Peng, Chi-Yu Li, Hongyi Wang, and Songwu Lu. 2014. Control-plane protocol interactions in cellular networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. ACM, 223–234.