Network Protocol Safe Configuration Search in One Shot

You Li, Kaiyu Hou, Hai Zhou, Yan Chen
Northwestern University, USA

CCS CONCEPTS

• **Networks** → **Protocol correctness**; **Protocol testing and verification**; • **Security and privacy** → *Logic and verification.*

KEYWORDS

Network system configuration; Model checking; Protocol security

1 INTRODUCTION

Model checking techniques can verify correctness properties by exhaustively traversing the state space of a formal model. A large number of works [1, 4, 5] done by networking researchers use model checking. Each of the works comes with one or more formal models, which constitutes a valuable asset for the networking security research community. Nevertheless, we find these works do not fully exploit the potentials of their models.

A better way to extract more underlying information from those models is to search for the *secure option space*: given a set of correctness properties, if the model executes from any state within the secure option space, it is guaranteed these properties always hold. Traditional model checkers cannot solve it, as their targeted goal is to decide whether an instance of the model is safe with regard to the given properties. Leveraging recent breakthroughs of symbolic model checkers, we devise algorithms to search for the secure option space. Moreover, the found results can be stored compactly and queried efficiently. We have implemented prototypes of our proposed algorithms and are testing them on existing models of cellular network protocols.

2 MOTIVATION

A network protocol or system can be formalized as a finite state transition system $S : (\bar{i}, \bar{x}, I(\bar{x}), T(\bar{i}, \bar{x}, \bar{x'}))$ consisting of input variables $\bar{i}$, state variables $\bar{x}$, initial state $I(\bar{x})$, and a transition relation $T(\bar{i}, \bar{x}, \bar{x'})$: the function of inputs $\bar{i}$ and current state $\bar{x}$ to determine the next state $\bar{x'}$. A state s is an assignment of values to all state variables $\bar{x}$. A state either satisfies a correctness property: $s \vDash P$, or falsifies it: $s \nvDash P$.

Correctness properties can be divided into two categories: safety properties and liveness properties. Safety properties state that some bad things can never happen, while liveness properties state that some good things should eventually happen.

Traditionally, after the model is specified, researchers select a set of correctness properties the original protocol or system should follow. If the model checker determines the properties are satisfied, one can conclude the original protocol or system is safe with regard to the properties. In the field of networking security, often times the researchers expect to find counterexamples, because counterexamples suggest possible traces of attacks or other vulnerabilities. Regardless of their purposes and methodologies, researchers always choose a designated initial state, so their results only apply to that specific initial state.

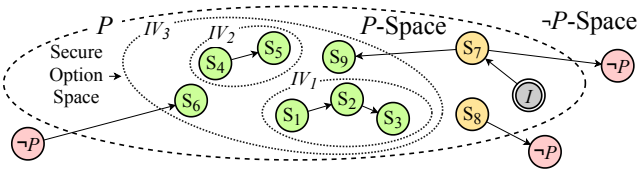

Figure 1: An illustrative example of searching the secure option space for the safety property P. The original initial state I lies in the P-space, but it is not secure as it can reach the $\neg P$-space. The greatest inductive invariant, IV_3, is the secure option space we aim to find. Any state lies in IV_3 is always guaranteed to satisfy P. Note there exist other inductive invariants like IV_1, IV_2, $IV_1 \cup IV_2$, *etc.*

In abstract, the above problem is a decision problem given the model, the initial state, and the correctness properties; the result is either satisfy or falsify. Network protocols and systems are highly complicated. There usually exist more than one way to deploy a single network system. In that regard, we propose to convert the decision problem into a search problem given the model and the properties; the result is a set of states which satisfy the properties.

We call that set of states *secure option space*. The benefits of finding the secure option space are two-fold. For designers, they want to know in what circumstances their designs are secure, especially when a design has multiple asynchronized components, keeps evolving over time, and has more than one contributing team. It is desired by designers that the secure option space covers all use cases and customer requirements. For deployment engineers, they want to configure the systems to meet all security criteria and make sure the systems always execute as expected, provided there are so many configurable parameters and initial states for a network system. In either case, no one will want to enumerate all possible initial states, and run model checker for each of them.

We describe two symbolic model checking algorithms to search secure option spaces for safely and liveness properties, respectively.

3 SECURE OPTION SPACE SEARCH FOR SAFETY PROPERTIES

A safety property P states that bad things can *never* happen. For example, an unauthorized party should not be allowed to broadcast fake alerts by abusing the wireless alert system. On a formal model, it is equivalent to say that none of the bad states ($\neg P$-states) can be reached. Therefore, the problem of searching a secure option space becomes searching for states which cannot lead to any bad states.

An assertion F is an *inductive invariant* if $F \wedge T \Rightarrow F'$: there is no outgoing edges from F-states to $\neg F$-states. Here is the key insight:

SIGCOMM '20 Demos and Posters, August 10–14, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8048-5/20/08.
https://doi.org/10.1145/3405837.3411377

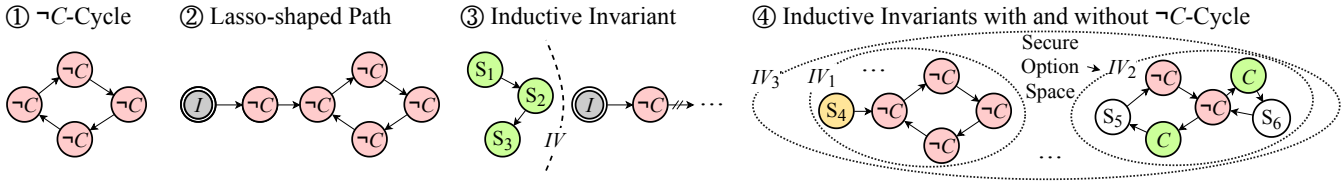

Figure 2: ① A $\neg C$ cycle, in which the condition C of the liveness property P is never satisfied. ② The system is not secure as there exists a path from the original initial state I to the $\neg C$-cycle. ③ Any potential $\neg C$-cycles outside of the inductive invariant IV have no effect to the states within IV. ④ IV_3 and IV_1 have a $\neg C$ cycle in them, so they cannot be determined to be secure. IV_2 is a secure option space we aim to find.

if there exists an inductive invariant F^* which does not intersect with the $\neg P$-space, all F^*-states are secure (Figure 1).

IC3 [2] is the *state-of-the-art* symbolic model checking algorithm. IC3 maintains a sequence of frames $F_1 \cdots F_k$, in which $F_i \wedge T \Rightarrow F'_{i+1}$. During an iteration, if IC3 finds a path from an initial state to a $\neg P$-state, IC3 will return the path as a counterexample; otherwise, it strengthens any frame F_i that have states violating $F_i \wedge T \Rightarrow F'_{i+1}$. After every iteration, if there exists an $F_i = F_{i+1}$, the algorithm returns F_i as an inductive invariant and concludes the model is secure from the initial states; otherwise, IC3 will produce a new frame $F_{k+1} \leftarrow P$ and continue to the next iteration.

Our search for the secure option space can benefit from the design of the IC3 in the following two aspects: *i)* The algorithm will always return an inductive invariant F^* if it exists. *ii)* After every iteration, a new frame $F_{k+1} \leftarrow P$ is produced. Since only P-states can constitute the secure option space, the search for the secure option space always starts from the largest possible candidate, and consequently tends to find a large such space.

The original IC3 algorithm has to be modified to fit our needs. The search should not be depending on any specific initial states, nor should it be interrupted by any found counterexamples. Therefore, the initial condition is set to be *False*, meaning there are no concrete initial states. However, as there are no initial states to constrain abstraction, more states are eliminated from the frames. The resulting inductive invariant would be small or even an empty frame. To deal with this problem, we control the extent of generalization, so that in each iteration a border cube can only be generalized to an extent proportional to its size.

Once an inductive invariant is found, we can expand its size recursively by setting the found F^* as F_0 in the next execution of the above algorithm to find a larger secure option space.

4 SECURE OPTION SPACE SEARCH FOR LIVENESS PROPERTIES

A liveness property states that good things should *eventually* happen. For instance, a phone call to an emergency number should be eventually routed to a public safety center. No liveness properties can be satisfied if a system does not move forward; so liveness properties are usually used in conjunction with fairness constraints[1].

Liveness properties can be refuted if there exists a *lasso-shaped* path starting from an initial state I, and none of the states along that path satisfies the condition of the liveness properties. A lasso-shaped path from I is a path that has a cycle (Figure 2 ①) and a connection from I to any state on the cycle (Figure 2 ②).

[1]Strong Fairness: a process which is *infinitely often* enabled should be executed infinitely often. Weak Fairness: a process which is *always* enabled should be executed infinitely often.

FAIR [3] is an algorithm to find such paths we are looking for. It first queries the SAT solver for a group of states that satisfies a certain fairness condition. Then it queries IC3 for a connection from the initial state to one of the states in the group. If that succeeds, the algorithm queries IC3 for multiple times to build a cycle with the group of states. If all steps above succeed, FAIR finds a lasso-shaped counterexample. However, if any of the queries to IC3 fail, FAIR accumulates a new inductive invariant returned by IC3. Notice that no cycle can go through a boundary depicted by an inductive invariant. FAIR should eventually terminate because it keeps refining the state space with newly found inductive invariants.

In order to find the secure option space with regard to liveness properties, we require the fairness condition to be that no states satisfy the condition C of the liveness properties. That fairness condition applies to both the group-of-state queries and the connection queries. Then we can reach a key observation: if there exists no such cycle within an inductive invariant, every state within that inductive invariant belongs to the secure option space. Figure 2 ③ and Figure 2 ④ demonstrate this observation.

So, we can devise a search algorithm. First it randomly selects some initial states and run the queries to accumulate a number of inductive invariants. Then the algorithm selects a sufficiently large inductive invariant, trying to find a fairness cycle within it. If no such cycle can be found, that inductive invariant is joined with the already found secure option space. If there is such a cycle, one of the states on the cycle is recorded and used as a constraint for future search, so that there will be no repeated efforts.

5 EXPERIMENTS ON CELLULAR NETWORKS

We have implemented our algorithms for safety and liveness properties and tested the algorithms with synthetic models. Given any states, a SAT solver can determine in one shot if they reside in a found secure option space.

At the meanwhile, we collected existing models [4, 5] which were constructed to find vulnerabilities on cellular network protocols. An especially large number of configurable and environment variables exist in the cellular network system, as it is a combination of many subsystems and has evolved significantly for many generations. We built a pipeline to translate the original state transition relations into the format which our programs can parse.

In the past, researchers make efforts to find vulnerabilities within systems. Then they mitigate those vulnerabilities, hoping the systems can become more secure. Such a process could iterate forever. Our proposed method deals with this problem from the reverse direction: efficiently searching for the states and configurations which must be secure. Protocol designers, deployment engineers, as well as researchers, could benefit from the method.

REFERENCES

[1] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. 2018. A formal analysis of 5G authentication. In *2018 ACM SIGSAC Conference on Computer and Communications Security*.

[2] Aaron Bradley. 2011. SAT-based model checking without unrolling. In *2011 International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Springer.

[3] Aaron Bradley, Fabio Somenzi, Zyad Hassan, and Yan Zhang. 2011. An incremental approach to model checking progress properties. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE.

[4] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. 2018. LTEInspector: A systematic approach for adversarial testing of 4G LTE. In *the 25th Network and Distributed Systems Security (NDSS) Symposium*.

[5] Yinbo Yu, You Li, Kaiyu Hou, Yan Chen, Hai Zhou, and Jianfeng Yang. 2019. CellScope: Automatically Specifying and Verifying Cellular Network Protocols. In *2019 SIGCOMM Conference Posters and Demos*. ACM.