KORG: A Communication Channel-Based Attack Synthesizer for Distributed Protocols

Jacob Ginesin
Northeastern University

Max von Hippel
Northeastern University

Cristina Nita-Rotaru
Northeastern University

***Abstract*—Distributed protocols underpin the modern internet, making their correctness and security critical. Formal methods provide rigorous tools for analyzing protocol correctness and cryptographic security, yet existing tools fall short for denial of service (DoS) analysis. We introduce KORG, a tool that synthesizes attacks on distributed protocols by targeting communication channels to violate linear temporal logic (LTL) specifications. KORG provides sound, complete analysis, synthesizing attacks or proving their absence through exhaustive state-space search. With support for pre-defined and custom attacker models, KORG enables targeted DoS analysis and broader LTL-based verification, demonstrated through various case studies.**

***Index Terms*—Protocols, Attack Synthesis, Denial of Service, Model Checking**

I. INTRODUCTION

Distributed protocols are the foundation for the modern internet, and therefore ensuring their correctness and security is paramount. To this end, formal methods, the use of mathematically rigorous techniques for reasoning about software, has been increasingly employed to analyze and study distributed protocols. Historically, formal methods has been employed for reasoning about concurrency and the correctness of distributed systems, and in recent years formal methods has been employed to reason about the security of cryptographic protocols and primitives. However, no previous formal methods tools offered an effective solution for studying denial of service attacks on protocols.

To fill this gap we introduce KORG, a tool for synthesizing attacks on distributed protocols that implements the theoretical framework proposed in Hippel et al. In particular, KORG targets the communication channels between the protocol endpoints, and synthesizes attacks to violate arbitrary linear temporal logic (LTL) specifications. KORG either synthesizes attack, or proves the absence of such via an exhaustive state-space search. KORG is sound and complete, meaning if there exists an attack KORG will find it, and KORG will never have false positives. KORG supports pre-defined attacker models, including attackers that can replay, rearrange, or drop messages on channels, as well as custom user-defined attacker models. Although KORG best lends itself for reasoning about denial of service attacks, it can target any specification expressable in LTL. We present a variety of case studies illustrating the employability and usefulness of KORG.

II. DESIGN METHODOLOGY

In this section we discuss the details behind the design, implementation, and guarantees of KORG.

A. High-level design

TODO: diagram. Promela model, channel selection, gadget selection/definition get put into Korg. Korg spits out another promela model, which is put into Spin along with a property. Then, we get some attacks.

B. The Korg Implementation

We implemented KORG on top of the SPIN, a popular and robust model checker for reasoning about distributed and concurrent systems. Intuitively, models written in PROMELA, the modeling language of SPIN, are communicating state machines whose messages are passed over defined *channels*. Channels in PROMELA can either be unbuffered synchronous channels, or buffered asynchronous channels. KORG generates attacks *with respect* to these defined channels.

```
// channel of buffer size 0
chan msg_channel = [0] of { int }

active proctype Peer1() {
    msg_channel ! 1
}

active proctype Peer2() {
    int received_msg
    msg_channel ? received_msg
}
```

Listing 1. Example PROMELA model of peers communicating over a channel

Following the gadgetry framework as described in Hippel et al., KORG is designed to parse user-chosen channels and generate gadgets for sending, receiving, and manipulating messages on them. KORG has built-in gadgets that are designed to emulate various real-world attacker models, as further described in Section III. Additionally, users can explicitly define which messages a generated gadget can send and receive. Once one or multiple gadgets are generated, KORG invokes SPIN to check if a given property of interest remains satisfied in the presence of the attacker gadgets.

C. Soundness And Completeness of Korg

KORG is an implementation of the theoretical attack synthesis framework proposed by Hippel et al. This framework enjoys soundness and completeness guarantees for attacks discovered; that is, if there exists an attack, it is discovered, and if an attack is discovered, it is valid. However, the attack synthesis framework proposed by Hippel et al. reasons about an abstracted, theoretical process construct. Therefore, in order to correctly claim KORG is also sound and complete, it is necessary to demonstrate discovering an attack within the theoretical framework reduces to the semantics of SPIN, the model checker KORG is built on top of.

Theorem 1. *Checking whether there exists an attacker for a given threat model, the R-∃ASP problem as proposed in Hippel et al., reduces to Büchi Automata language inclusion (which is in turn solved by the* SPIN *model checker).*

Proof. Recalling the definitions from Hippel et al., a *process* is Kripke structure whose transitions are equipped additional input and output operations in the same flavor as a standard I/O automata.[1] Hippel et al. also defines asynchronous composition on processes to match input and output transitions with the same label when constructing the product automata.

Threat models, then, contain a *target process* P that is unmodifiable by an attacker, a set of vulnerable processes $Q_1, \ldots, Q_n$ that are unmodifiable by an attacker, and a Linear Temporal Logic specification ϕ. Let $||$ denote asynchronous composition between processes. For simplicity, let $Q = Q_1 \; || \; Q_2 \; || \; \ldots \; || \; Q_n$. Given this, we initially require $P \; || \; Q \models \phi$ (that is, P composed with Q satisfies the property ϕ.)

Now, our attacker synthesis problem becomes checking whether we can find some process A such that $P \; || \; A \not\models \phi$. Hippel et al. showed finding such an A can be done algorithmically, maintaining soundness and completeness guarantees, given the input and output transition labels of A, denoted $\mathcal{C}(A)$, is a subset of $\mathcal{C}(Q)$. In particular, Hippel et al. describes gadgets dubbed "daisies" which consist of a main state, a recovery state, circular transitions for each input and output label on the main state, and a non-deterministic transition to the recovery state. To construct A, $P \; || \; Daisy(Q) \models \phi$ is checked.

In short, SPIN implements model checking by reducing Promela models to a Büchi Automata(a ω-regular automata), converting a Linear Temporal Logic property into a Büchi Automata, intersecting the two to construct a product automata, and determining if there exists a reachable acceptance cycle [1]. We know by Vardi, we can always generate a Büchi Automatathat accepts the traces of any given Kripke structure [1], [2]. Thus, defining the transition relations in our Büchi Automatato match the I/O transition labels in their respective processes, we can convert P, $Daisy(Q)$, and ϕ to Büchi Automataand intersect them with SPIN. Then, SPIN will soundly and completely search the product automata for acceptance cycles, either finding a counterexample to ϕ or proving the absence of such a trace. □

[1]Modeling processes in this way allows for the simultaneous modeling of message passing while also maintaining the ability to leverage Linear Temporal Logic for specification

From this result, we naturally get a complexity-theoretic result for finding an attacker from a given threat model.

Theorem 2. *Checking whether there exists an attacker for a given threat model, the R-∃ASP problem as proposed in Hippel et al., is PSPACE-complete.*

Proof. By the previous argument, the R-∃ASP problem reduces to intersecting multiple Büchi Automata, which is well-known to be PSPACE-complete [3]. □

Although this result implies KORG has a rough upper bound complexity, in practice due the various implementation-level optimizations of SPIN finding attacks on some ϕ is generally fast, but proving their absence via a state-space search can expensive.

III. Attacker Models

In this section we discuss the various details for each attacker model built into KORG.

A. Custom Attacker Models

B. Replaying Attacker Model

C. Rearranging Attacker Model

D. Dropping Attacker Model

IV. Case Studies

A. SCTP

B. TCP

C. DCCP

V. Usage

VI. Conclusion

References

[1] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification." IEEE Computer Society, 1986. [Online]. Available: https://orbi.uliege.be/handle/2268/116609

[2] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA: MIT Press, 2000.

[3] D. Kozen, "Lower bounds for natural proof systems," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. Providence, RI, USA: IEEE, Sep. 1977, p. 254–266. [Online]. Available: http://ieeexplore.ieee.org/document/4567949/