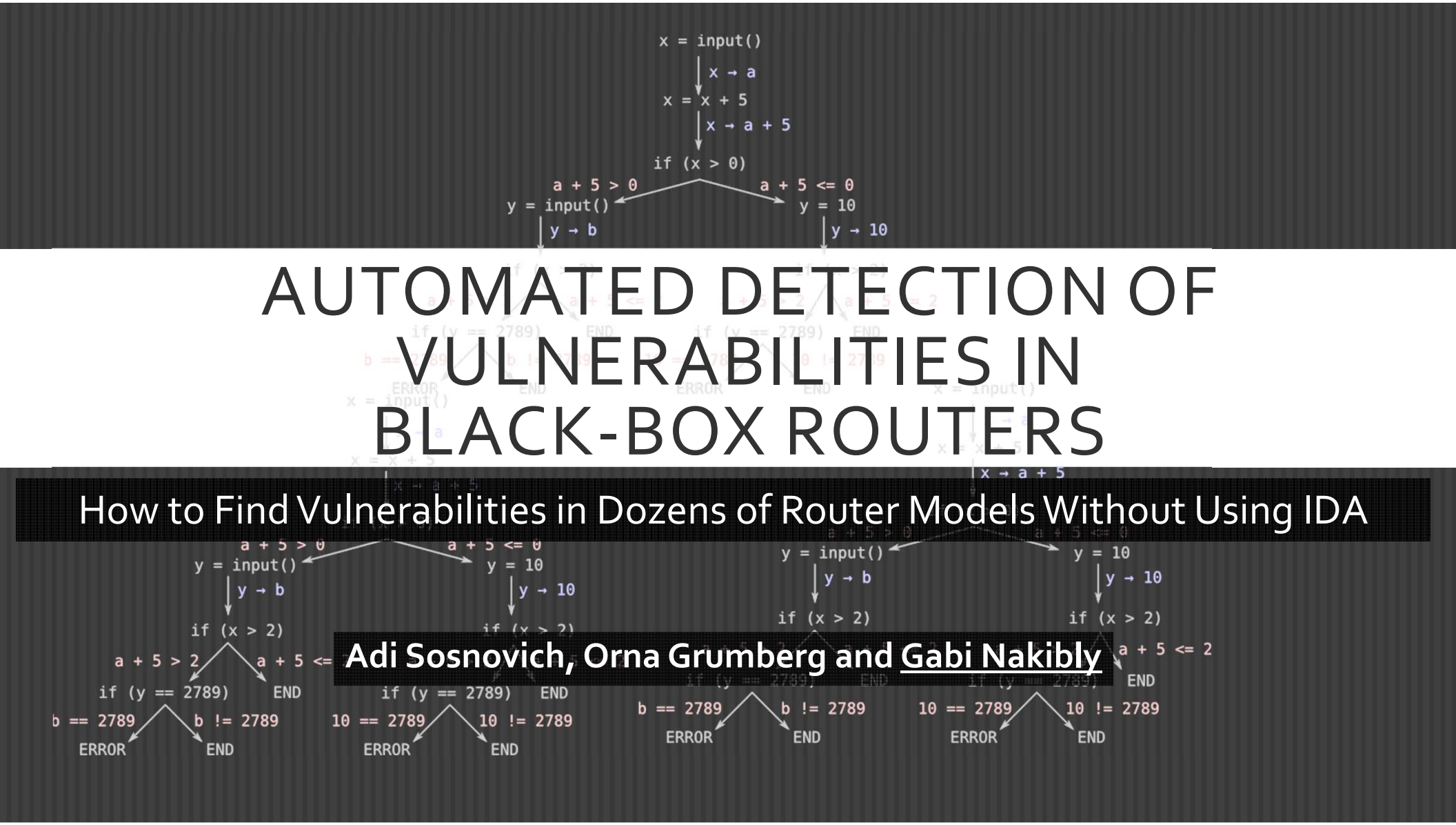


AUTOMATED DETECTION OF VULNERABILITIES IN BLACK-BOX ROUTERS

How to Find Vulnerabilities in Dozens of Router Models Without Using IDA

Adi Sosnovich, Orna Grumberg and Gabi Nakibly

INTRODUCTION – GABI NAKIBLY

- Chief research scientist at the National Cyber and Electronics Research Center
  - Operated by Rafael – Advanced Defense Systems ltd.
- A former Visiting Scholar at Stanford University
- Senior adjunct lecturer and research associate at the Technion – Israel institute of Technology
- I mostly spend my days doing network security research.

OUTLINE

- Motivation
- The method we use
- What is symbolic execution
- Unique optimizations that make our approach work
- Application of the method to Cisco's OSPF
- The vulnerabilities we found

MOTIVATION

- Network protocols are based on open standards
- However, the Internet runs mostly on proprietary and closed-source network devices
- A hidden deviation of a device's implementation from a protocol standard may create a logical vulnerability

MOTIVATION (CONT.)

- However, finding deviations in closed-source routers demands great efforts.
- In this work we present a method that leverages a **formal** **black-box** method to unearth implementation deviations in closed-source network devices
  - No need to access the binary or source code of the device!
  - Just observe the external behavior of the router

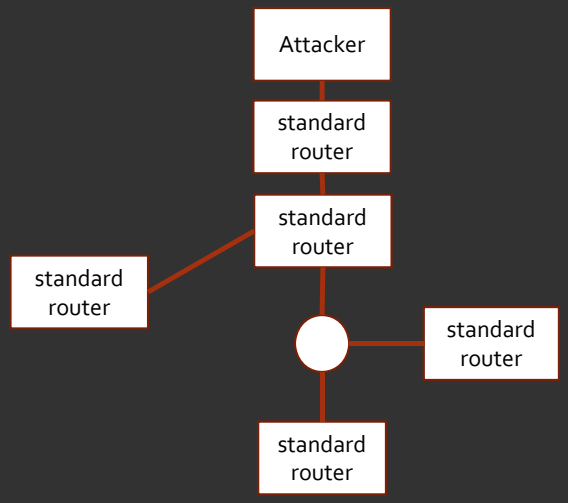
THE BASIC IDEA IN A NUTSHELL

- Compare the behavior of a network device (called SUT) to that of a formal model that captures standard functionality of the network protocol.
  - SUT = System Under Test

THE BASIC IDEA IN A NUTSHELL

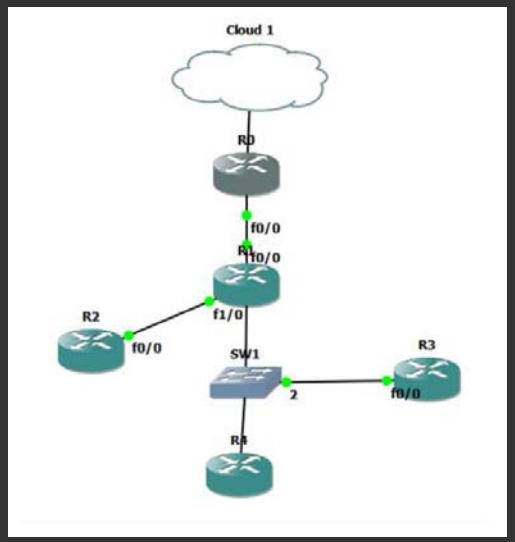
- Compare the behavior of a network device (called SUT) to that of a formal model that captures standard functionality of the network protocol.
  - SUT = System Under Test

Formal model

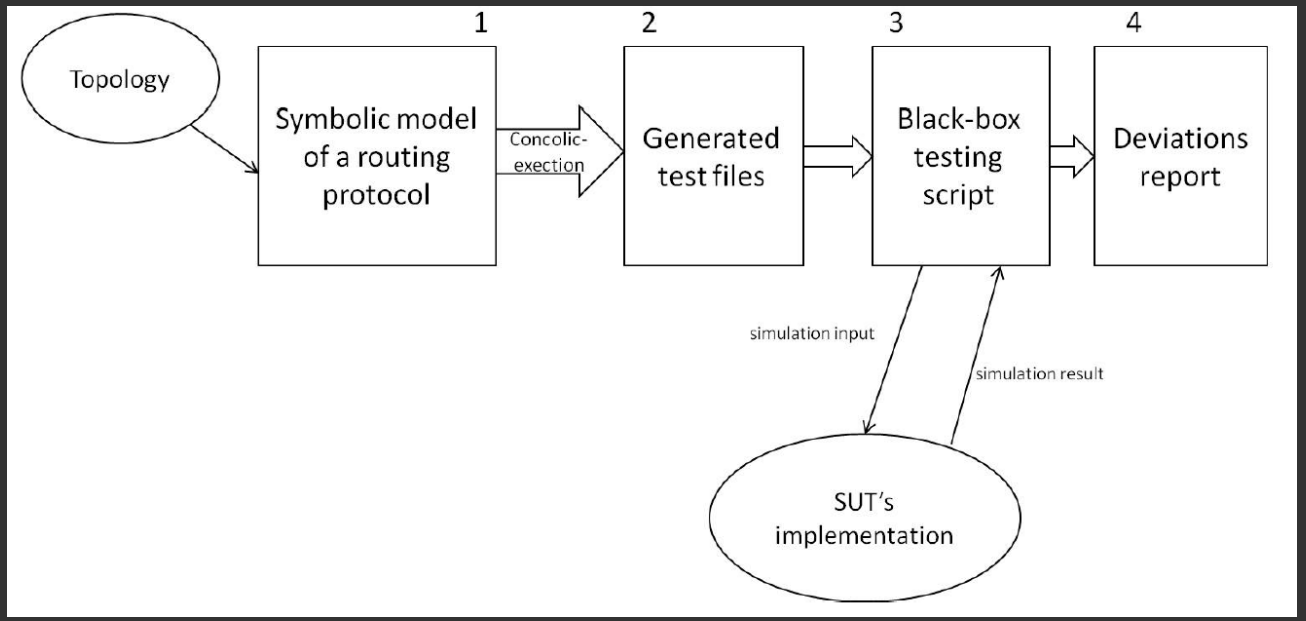


compare

Real devices (SUT)

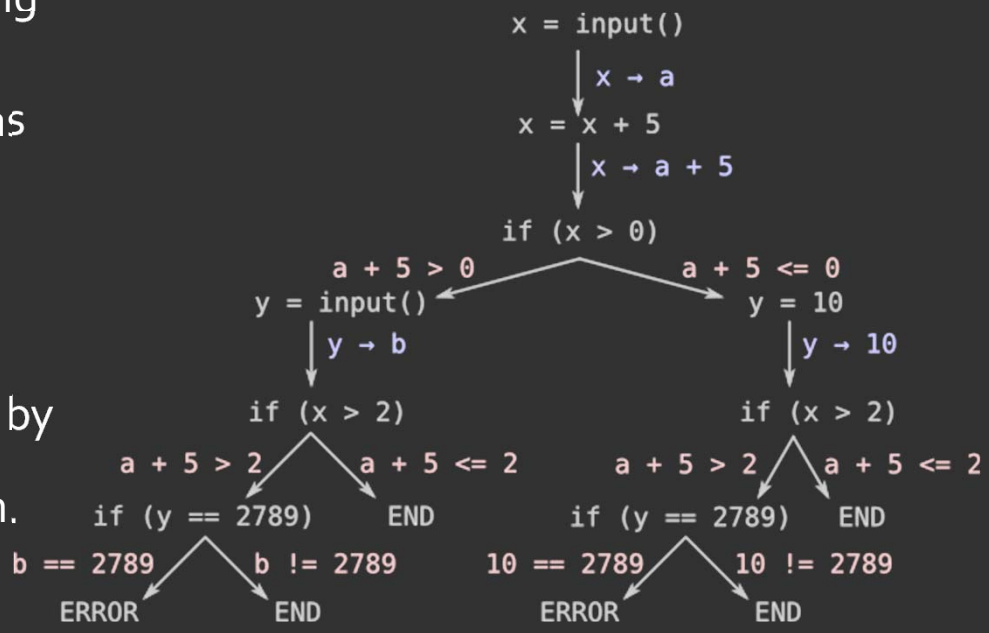


OUR METHOD IN A GLANCE



SYMBOLIC EXECUTION

- Symbolic execution allows analyzing the execution paths of a program and generating corresponding test cases.
- The input variables of the program are defined as symbolic variables.
- Then, the program is symbolically run, where symbolic expressions represent values of the program variables.
- On each execution path a constraint is obtained by collecting all the symbolic expressions that correspond to conditional branches on that path.

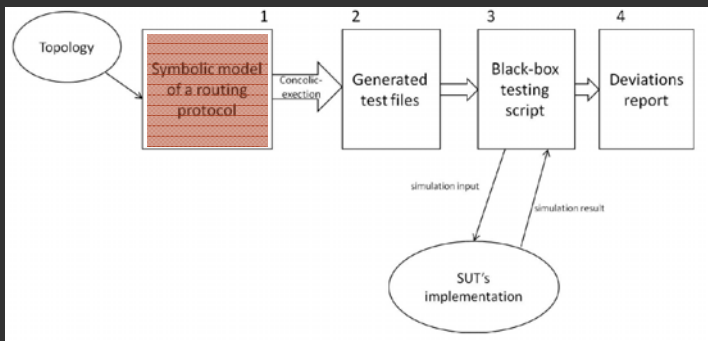


CONCOLIC TESTING

- Concolic execution = symbolic execution + real execution
- Concolic testing is a dynamic symbolic execution technique to systematically generate tests along different execution paths of a program.
- It involves concrete runs of the program over concrete input values alongside symbolic execution.
  - Initially, some random concrete input values are chosen.
  - During a run of the program with this input, symbolic constraints are gathered over the conditional branches of the current execution.
  - A constraint solver is then used to construct the next concrete execution on a different path.

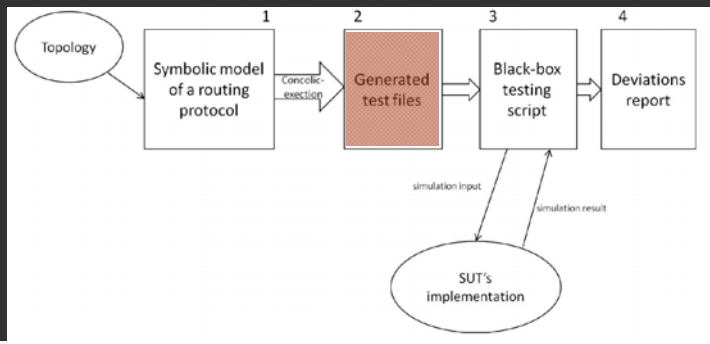
1. CREATE A MODEL OF A PROTOCOL

- The first thing is to create a model of the protocol.
- A model (which can be written in C, Python,….) captures the functionality of the protocol.
  - It can be elaborate or detailed as needed.
  - It written based on the protocol standard or based on existing software.
- The model simulates the execution of the protocol among the network devices in a predefined topology.
  - The network devices we want to test needs to be setup in the above topology.
- The model receives as an input rogue protocol messages sent by an attacker.
  - These messages are symbolic variables.



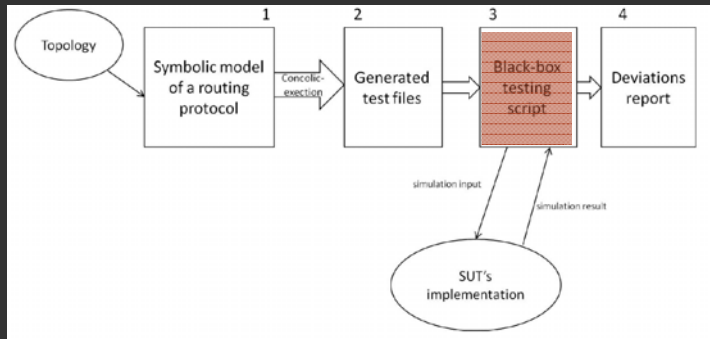
2. GENERATE TEST CASES

- The tool uses concolic execution on the model of the protocol to cover **all** execution paths of the protocol's model.
- Each execution path is driven by a specific sequence of rogue protocol messages sent by the attacker.
  - Each such execution path represents a test case.
- A test case entails sending the sequence of the rogue messages to the network devices we actually.



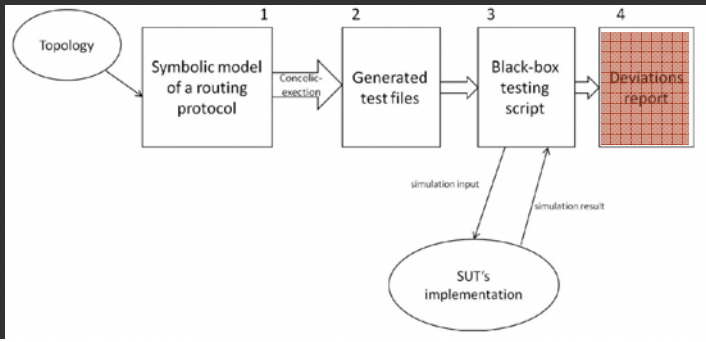
3. EXECUTE TESTS

- Each generated test file is executed on the SUT.
- Before each test the devices are initialized to a predefined initial state.
- During the test execution the sequence of rogue protocol messages are sent to the network devices.
- At the end of the test the state of the devices are extracted.
  - For example, for routers the state is their routing table and routing advertisements DB.



4. FIND DEVIATIONS

- The devices states for each test are compared to the expected state obtained from the model.
- If there is a mismatch – the test fails.
- A failed test represents a deviation of the protocol's implementation from the protocol standard.
  - Which likely represents a vulnerability
- The failed test is accompanied with traces of all messages exchanged between the devices during the run of the test, both on the model and on the SUT.
- Comparing these traces facilitates the analysis of the vulenrability.



IMPORTANT CHARACTERISTICS

- **Logical vulnerabilities** – our tool does not aim to find technical vulnerabilities(such as buffer overflows or UAF), but rather logical vulnerabilities of the protocol implementation.
  - Such logical vulnerability stem from the fact that the protocol was not implemented as it should.
  - They pose a serous threat to the robustness of the protocol's design.
- **Automatic** – once the protocol's model and topology are determined, the identification of deviations is fully automatic.
- **Full coverage** – the generated tests cover the entire functionality of the protocol's model.
- **Modularity** – the analyzed model need not detail the protocol in its entirety. The model may only include parts of the protocol deemed relevant to the security analysis or parts that may be considered more prone to deviate from the standard.

SCALABILITY ISSUES

- If the protocol model is complex i.e. we model large chunk of the standard or we model many devices in the topology, many tests may be needed to cover the entire functionality model.
- This is called the **path explosion problem**.
- We deal with it using a unique optimization that is tailored to testing network protocol in general – and routing protocols in particular.
- Our optimizations allow to **dramatically reduce** the number generated tests **without reducing the coverage** of the protocol's model.

OPTIMIZATION

- We reduce execution paths that cross the same intermediate states.
- Let's examine all test cases that have a sequence of N messages of the following form:

- We can reduce them to following single test without loosing functionality coverage:

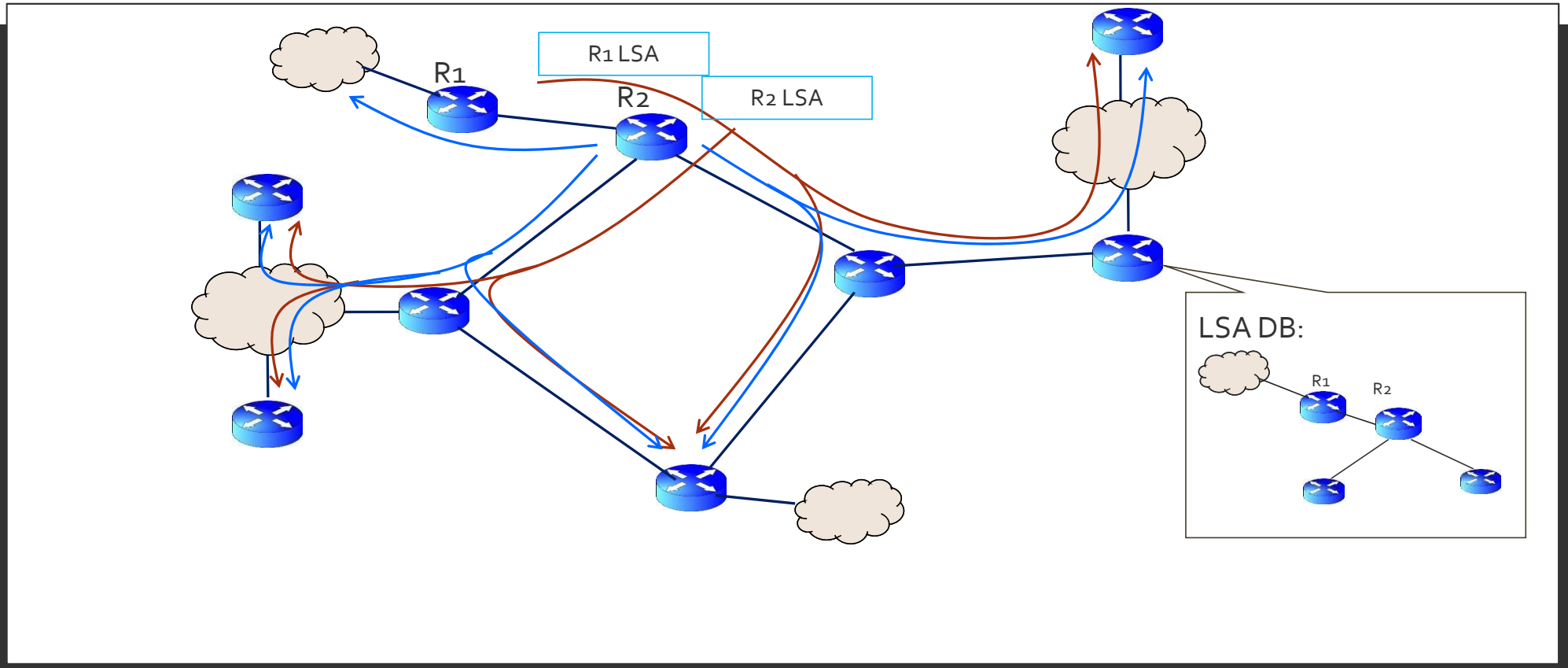
OSPF ANALYSIS

- We now describe how we applied our method to find vulnerabilities in the OSPF implementation of routers.
- OSPF is one of the most widely used and most complex routing protocols on the Internet.

OSPF 101

- Every router advertises it's link state (i.e. "who are my neighbors?").
  - This is called Link State Advertisement (LSA).
- The LSAs are flooded throughout the autonomous system hop-by-hop.
- Every router receives the LSAs of all other routers.
  - This allows to build the topology map of the AS.

OSPF 101 (CONT.)



THE ATTACKER

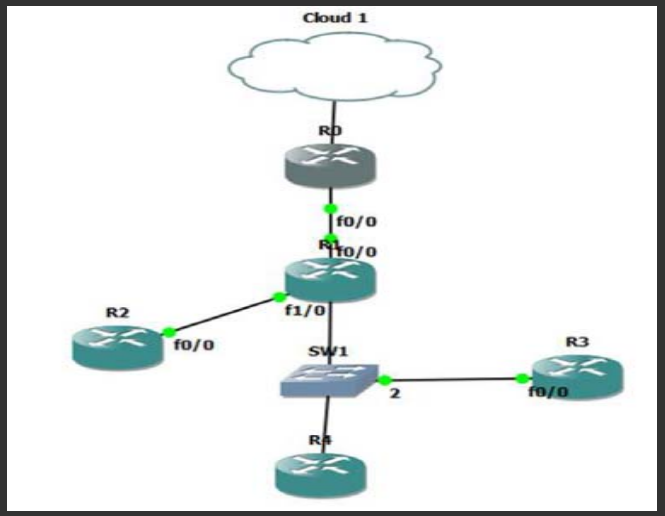
- Location: inside the AS
  - Controls a single legitimate router in an arbitrary location
  - This means it can flood LSAs to its neighbors
- Goal: Significant and persistent control of the routing tables of other routers in the AS.

OSPF FORMAL MODEL

- To serve the purpose of finding security vulnerabilities our OSPF model focuses on the core parts of the protocol that are relevant to its security against the above type of attack.
- We modeled the OSPF using a Python code having roughly 1000 LoC.
- The symbolic variables are the fields of the LSA:
  - Sequence number
  - Destination
  - Advertising Router
  - LSID
- For each symbolic LSA, the LSA is sent to its destination and then a loop is applied. On each loop iteration every router runs its procedure once.
- The router's procedure implements the core functionality of OSPF.

OSPF FORMAL MODEL (CONT.)

- The model included 5 routers in the following simple topology:
- The symbolic LSAs are sent from cloud 1.



TESTBED (FOR CISCO)

- To test Cisco's OSPF implementation we used alternately two network simulation software: GNS3 and VIRL.
  - Both software suites allow to simulate a network of multiple routers, each running an emulation of an actual IOS image (identical to the images used in real Cisco routers).
- We used the following IOS versions:

IOS Version	Release date
15.1(4)M, release software (fc1)	Mar. 2011
15.2(4)S7, release software (fc4)	Apr. 2015
15.6(2)T, release software (fc4)	Mar. 2016

This is the latest IOS version

RESULTS

- For Cisco we discovered 7 deviations in all three IOS versions.
- 2 of those deviations were new and exist in the most up-to-date IOS version.
- The new vulnerabilities allow an attacker to remotely erase the routing table of a remote router
  - And even update the routing table with arbitrarily false information.

VULNERABILITIES - DETAILS

- Incomplete fight-back for Rogue LSA with maximum sequence number
  - A spoofed LSA having the maximum sequence number was sent by unicast to a victim router R on behalf of R itself.
  - R did NOT sends a "fight-back", hence the content of the spoof LSA persists.
- Incorrect MaxAge LSA origination during fightback:
  - A spoof LSA having the maximum sequence number was sent on behalf of R2 to R1.
  - R2 originates a fight-back with the incorrect content, hence the content of spoofed LSA persists.

QUAGGA (OPEN SUSE)

- Similar vulnerabilities have been discovered in Quagga.
- However we used a different testbed setup to run those routers.

IN SUMMARY

- Our approach can be adapted to any routing protocol or network protocol for that matter.
- All our tool need is a model of the protocol and you are good to go!
- You do not have to write your own model. An exiting open-source of the protocol can work as well with the right adaptations.
- Once you have a model you can test any implementation of that protocol fully automatically
  - allowing you to discover many vulnerabilities is a short time.