

Assignment 2

DUE DATE: 11:59PM MELBOURNE TIME, THURSDAY MAY 1ST, 2025

Introduction

This handout is the Assignment 2 sheet. The assignment is worth 20% of your total mark. You will carry out the assignment in the same pairs as for Assignment 1, *unless you were allocated to a new pair in case your partner withdrew from the subject.*

In this assignment you will use Alloy to diagnose and fix a vulnerability in a secure audio calling software package. This vulnerability is inspired by a real vulnerability that was present in the *Signal* end-to-end encrypted messaging application (and has since been fixed) that runs on many popular platforms including iPhone, Android, desktop PCs etc.

We will consider a simplified version of the audio calling protocol for this assignment. Specifically the real protocol uses multiple network transport mechanisms and distinguishes various call set-up stages including signalling and media transport. However, for this assignment we will consider a simple protocol whose intended sequence of interactions is depicted in fig. 1.

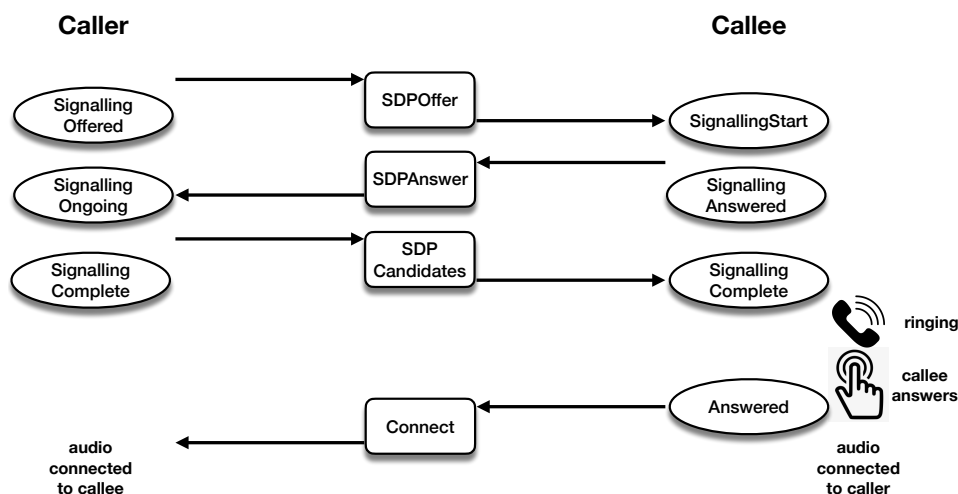


Figure 1: Intended sequence of interactions in the protocol.

For this assignment we will ignore video media and assume that the call transmits only audio between the participants.

We refer to the party who is initiating the call as the *caller* and the party who is receiving the call as the *callee*. Ovals in the figure represent different *states* that each party is in at different points during the call set-up. The rectangles with rounded corners represent messages exchanged between the two participants.

The caller starts by sending an *SDPOffer* message to the callee and in doing so the caller moves to the *SignallingOffered* state. When the callee receives the *SDPOffer* message they move to the

SignallingStart state. From this state the callee then sends a *SDPAnswer* message and moves to the *SignallingAnswered* state. When this message is received by the caller, the caller moves to the *SignallingOngoing* state. From this state the caller then sends an *SDPCandidates* message to the callee and moves to the *SignallingComplete* state. When this message is received by the callee the callee also moves to the *SignallingComplete* state.

At this point the two parties have exchanged enough information to allow the audio to be transferred between them. So the callee's device (e.g. phone) starts ringing, telling them that a call has come in from the caller. When the callee answers the call (e.g. by pressing a button on their phone) they move to the *Answered* state. From this state, the callee then sends the *Connect* message to the caller and the callee then connects their audio device to the caller (i.e. starts transmitting to the caller the audio recorded by the phone's microphone). Once this *Connect* message is received by the caller, the caller also connects their audio device to the callee, and both parties can now exchange audio.

The Formal Model You are provided with a partial Alloy model of this protocol. Part of your tasks for this assignment is to complete this model using the description above and the following information.

The Alloy model formally describes the state of a participant in the protocol. We refer to this participant as the *User*. Messages are sent between participant *addresses*. The address of the user is *UserAddress*. Each Message contains a *source* and a *destination* address, as well as a *message type* field (e.g. *SDPOffer* etc. above).

The Alloy model also describes the state of the *network*, specifically by remembering the last message that was sent on the network between participants.

The Alloy model of the User remembers the last participant that the User decided to call (when they are acting as the caller), and the last participant they decided to answer a call from (when they are acting as the callee). These two pieces of information we call the *last called* and the *last answered*.

For each address it also remembers the state (e.g. *SignallingOffered*, *SignallingAnswered* etc.) of the call (if any) for that address.

Messages sent by the User have their *source* address set to *UserAddress*. Their *dest* field is set to the address of the destination participant for whom the message is sent.

Sending a message places it onto the network. The initial *SDPOffer* message can be sent only to the address that is the *last called* address (i.e. its *dest* address must be the *last called* address) and only when there is no call state recorded by the User for that address.

The User can receive a message only when its *dest* address is *UserAddress* and only when that message is currently on the network. Receiving a message removes it from the network. The *SDPOffer* message can be received only when no call state is recorded for the source address of the message (i.e. for the caller).

The other messages can be sent by the User only when the User's recorded call state for the message's destination *dest* address is as indicated in fig. 1 (e.g. *SDPAnswer* can be sent to some address only when the User's call state for that address is *SignallingStart*). Similarly, messages can be received by the User only when the User's recorded call state for the message's source *source* address is as indicated in fig. 1 (e.g. *SDPAnswer* can be received from some address only

when the User's call state for that address is *SignallingOffered*).

Sending a message changes the User's call state for the message's *dest* address as indicated in fig. 1 (e.g. when the User sends the *SDPAnswer* message, this changes the User's call state recorded for the destination participant to be *SignallingAnswered*). Similarly, receiving a message changes the User's call state for the message's *source* address as indicated in fig. 1 (e.g. when the User receives the *SDPAnswer* message, the recorded call state for the message's source (sender) is updated to become *SignallingOngoing*).

The *Connect* message can be received always from any *source* address whenever the User's recorded call state for that *source* address is *SignallingComplete*.

When the User sends or receives a message, no part of the system is changed except:

- the network (sending a message adds it to the network; receiving a message removes it from the network),
- the User's recorded call state for the relevant participant (the message's source or destination, depending on whether the User is receiving or sending respectively),

and noting the following exceptions:

- Receiving the *SDPCandidates* message causes the *ringing* state to be updated to refer to the message's *source* address (i.e. to the caller).
- Sending the *Connect* message causes the *audio* to be connected to the message's destination (i.e. to the caller).
- Similarly, receiving the *Connect* message causes the *audio* to be connected to the message's *source* address.

The Alloy model also includes actions that model the user deciding to answer a ringing call, and deciding to call another participant. These actions update respectively the *last answered* and *last called* parts of the state.

The Attacker The Alloy model also includes the behaviour of potential *attackers*. The set of addresses controlled (owned) by the attacker(s) is *AttackerAddress*. The only other participants that the User can interact with are potential attackers, i.e. addresses from the set *AttackerAddress*.

Attackers can place any message onto the network whose *source* address is an address from *AttackerAddress*. They can perform no other actions nor modify any other state in the system.

Your Tasks

Task 1: Completing the Initial Model; Finding an Attack (8 marks)

Your first task is to complete the parts of the model that are missing. Doing so will require you to carefully understand the model you are given so you can work out how to fill in the missing parts.

You should not add any additional *sig* declarations nor modify any of the existing *sig* declarations during this assignment.

1. [5 marks] Complete the `user_recv_post` predicates to complete the initial model.
2. [2 mark] There is a vulnerability in the protocol as described above. Specifically, an attacker can cause the User to reach a state in which their audio is connected to the attacker but the user has not yet decided to call that attacker or to answer a call from that attacker.

Write an alloy assertion `assert no_bad_states` that asserts that this can never happen.

3. [1 mark] Use Alloy to discover this vulnerability. Describe the vulnerability in comments above your `check` command that produces the counter-example to this assertion. That is, describe what is wrong with the original protocol and how the vulnerable behaviour arises.

Task 2: Fixing the Model (4 marks)

1. [2 marks] Design a *minimal* fix for the protocol to prevent the vulnerability. That is the smallest change you can think of that would rule out the vulnerability without changing how the protocol functions when used as intended.

Add a comment *to the bottom* of your Alloy file describing how you fixed the vulnerability and what part you changed.

You should not need to add any extra messages, **sigs**, or any extra information to any messages to implement a minimal fix for the protocol as modelled here.

2. [2 marks] Show that your `no_bad_states` assertion now holds. Use a suitably high bound when carrying out this check.

Add comments justifying / explaining your choice of bound and, specifically, what guarantees you think are provided by this check.

To obtain full marks here we want to see you choose a bound that is large enough to provide some real guarantees but not larger than necessary to obtain those guarantees. You need to think carefully about what your bound means, i.e. what behaviours are covered by that bound and why showing the absence of the attack for all of those behaviours provides assurance about the fixed protocol.

If you think that your chosen bound does not provide good guarantees you should say so and explain why.

Task 3: Model Checking (8 marks)

Your fixed model should be capable of simulating protocol executions where the User acts as either the callee or the caller. It should also support other plausible execution traces.

An example predicate, `successful_run`, is provided in the Alloy file to animate behaviours where the User initiates the call (i.e., is the caller).

1. [2 marks] Study the semantics of the **eventually** keyword and the given `successful_run` predicate. Then define an equivalent predicate, `successful_run2`, that does **not** use the **eventually** keyword.
2. [2 marks] Write an assertion named `equivalent` to verify that `successful_run` and `successful_run2` are exactly equivalent. Use an appropriate bound for checking this assertion.

Note: Do not modify the original `successful_run` predicate.

3. [4 marks] Write a predicate `two_rings`, and use the Alloy **run** command to explore protocol behaviours where the User rings for one caller and then immediately rings for a different caller. In other words, there should exist two sequential states: the first in which the User is ringing for one caller, and the second in which the User is ringing for a different caller.

*Hint: A solid understanding of the **eventually** keyword from tasks 3.1 and 3.2 may be helpful here. A suitable bound for **run** is essential to produce example here.*

Academic Misconduct

The University misconduct policy applies to this assignment. Students are encouraged to discuss the assignment topic, but all submitted work must represent the pair's understanding of the topic.

The subject staff take plagiarism very seriously. In the past, we have successfully prosecuted several students that have breached the university policy. Often this results in receiving 0 marks for the assessment, and in some cases, has resulted in failure of the subject.

Submission

Submit your Alloy file using the link on the subject LMS.

Only *one* student from the pair should submit the solutions, and each submission should clearly identify **both** authors.

Late submissions Late submissions will attract a penalty of 10% (2 marks) for every day that they are late. If you have a reason that you require an extension, email Toby *well before the due date* to discuss this.

Please note that having assignments due around the same date for other subjects is not sufficient grounds to grant an extension. It is the responsibility of individual students to ensure that, if they have a cluster of assignments due at the same time, they start some of them early to avoid a bottleneck around the due date.