Research Proposal

Title: Black Box System Model Learning and Verification.

Author: Johnathan DiMatteo

Supervisor: Sebastian Fischmeister

Degree: MMath

Motivation

To enforce security and safety in devices, models must be validated. Validating a model ensures anomalous states and transitions are corrected to avoid security vulnerabilities. One proposed method to do this is a process of reverse engineering the system, called *Model Learning*. Model learning aims to construct blackbox state diagram models of software and hardware systems by providing inputs and observing outputs. This has been a fundamental research problem studied for decades, but two recent advancements have allowed us to propose a new method. First, *side channel analysis* has allowed us to study a system's response without the need to open the so called "black box". By treating a device as a black box system, one can monitor the power consumption to provide a non-intrusive approach. Traditional modelling tools can introduce timing errors and other anomalies. The downside to this approach is the complexity of the response (ie. it is now a time series signal instead of a binary output). But *machine learning* has opened new data driven approaches for dealing with time series modeling, allowing us to cluster signals to states. Useful in many areas; one possible application could be during Hardware in the Loop (HIL) testing to identify unwanted or undocumented functionality. A well trained model can identify flaws in the design of IoT devices to enforce security and safety. Another application is in protocol fuzzing. This system can determine which inputs will be accepted by the system.

Problem Statement

1. **State Detector**: Given a black box system, determine internal state changes in response to inputs in order to learn the system's behaviour.
2. **Input Generator**: Given the current state and knowledge of previous state changes, what should be the next query to the system?

System

State Detector

The State Detector needs to be able to determine what state the system is in given a response. There are several ways to do this. distance based: measure of how much the signal moved dtw: define a threshold. larger difference = more confident.

clustering algorithms: do they require us to input no. of states? online learning? confidence? options: dbscan, k means (dont know many clusters though) DBSCAN is good, no states required, identifies noise. BUT we have to recalculate with all data points, every time (BUT only do DTW once, ie. receive batch of responses, cluster, return yes + msg that triggered it, or no or maybe or more than one + msgs.

Bayesian approach: DETERMINE WHICH CLUSTER OR MAKE A NEW ONE. can eliminate noise manually or just accept it. for each state, build an expected signal. that way we save out on memory. if the expected signal has an extremely high standard deviation ... it might be time to make a new cluster but we need some way of matching up signals (apparently you can align them using cross correlation, or custom change detection ie. significant change) compute dtw distance to each expected state signal. (or cluster) return (current state) or (new state, msg that triggered it)

1. receive a response
2. calculate dtw to every expected state response
3. if within ϵ of one state, add it to the closest state. else if its within ϵ of two or more clusters, try joining them together. else not within ϵ of any: create a new state (parameter ϵ: HOW CAN WE TUNE ITif
4. align and re-compute expected state response
5. return (state (string), change detected (bool), distance to closest state?).

question: what should ϵ be? how can we tune it?
question: how can we get confidence levels of being in a state?

parameters: ϵ data: expected signals per state, number of signals per state, standard deviation per state

returns: (state(s), confidence) indicating the state change(s) and the message(s) that triggered it.

Input Generator

needs to know current state. enumerate through all possibilities at current state (ie. for I2C, write to each bit of one register, send all at once...) problem: what if we change states and then change again quickly? it would appear as noise. so we need to be careful

with noise. Change Detector returns all states it has observed and the trigger messages. Log states and trigger messages. If last state is a new state, repeat. Elif (we have seen it before): pick a state we have not been to before. Else all states have been reached: end. note: need to verify that all write operations are the same and change with state. have only done so with reads.

1. receive state
2. try to change the state: send a bunch of messages while modifying one bit at a time depending on protocol (character for URL, bit for NMAP packets).

question: how can we determine which msg to send given a state? see Active Learning, in particular, uncertainity sampling. To do uncertainity sampling we need to know the probability of being in a state (the confidence) to be returned by the state detector.

parameters: protocol specification (json?) returns: message to send

notes:
L star algorithm: can't because we don't have an oracle that can give us a counterexample, unless we search and depend on our clustering algorithm? also we dont have a way to deal with non-determinism
randomization is going to be important since we are exploring
There are ways to have error: given state A if we expect to go to B from experience but we go to a new state C, thats an error
perhaps genetic algorithms? first we explore a little to build a model then we train? ie. which messages are trigger messages? start with a random selection, keep ones that trigger a state change, refresh new ones ?

Message Sender

Determined by protocol. Focus on I2C for now.

Additional Modules or Functionality

method to convert states and transitions to mealy machine.
message sender (already implemented for I2C) focus will be on I2C since system is already in place.
function to plot expected signals ie. what do they look like

Data model: json? file that stores all states and state transitions.

Applications

model Verification
model security: can alert users if system is operating outside expected behaviour per state.