Research Proposal

Title: Reverse Engineering Devices Through Side Channel Analysis

Author: Johnathan DiMatteo

Supervisor: Sebastian Fischmeister

Motivation

The recent explosion of the Internet of Things (IoT) puts numerous embedded devices at the hands of governments, businesses, and consumers. Their low cost has caused manufacturers to produce and sell them without sufficient security or safety features. If action is not taken, vulnerabilities in IoT devices can lead to harm both economically and otherwise. To enforce security and safety in these devices, we will identify state transitions in order to reverse engineer the protocol specifications. This way anomalies can be identified and corrected to avoid security vulnerabilities. Traditional Hardware-in-Loop (HIL) testing methods involve significant intrusion to the system being observed. Our approach is to monitor the execution of the system to identify transitions without opening up the so called "black box", a technique known as *Side Channel Analysis*.

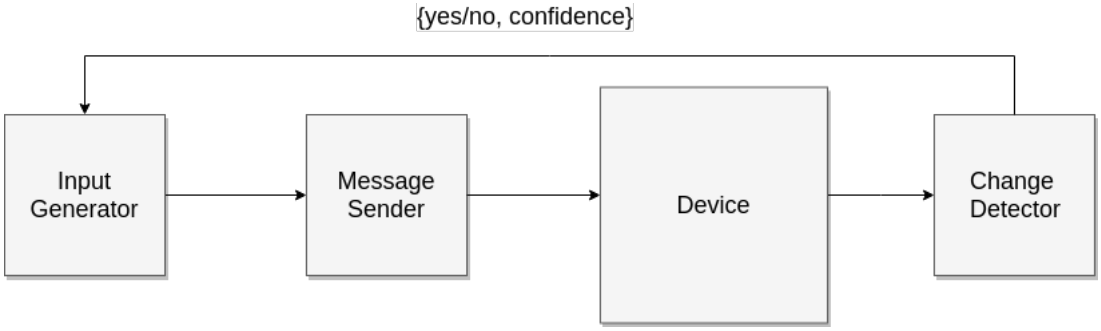
learning an output language as we go behaviour of device based on power trace

time complexity non deterministic restrict to a subset of the input language

Problem Statement

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

System



State Detector

The State Detector determines if the device has changed behaviour based on the response from the system being observed. 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 how many clusters though!!!) DBSCAN is good, no states required, identifies noise, BUT you can't do it online... and labels can be arbitrarily given and we have to recalculate with all data points, every time (BUT only do DTW once, ie. so i made my own. but it also has a problem: makes too many states. A new state in the same msg, even if it only belongs to that msg, is considered an error, because the learning algorithm is deterministic and will think it is a different behaviour. we can join them together as it learns but then it will needs to restart active learning algorithm. we are essentially building the output language as we go.

How we will evaluate? Precision, Recall, F1 score. Additional states in a message are counted seperately as errors. we want to minimize errors while maximizing f1 score. Number of noise samples can increase time complexity

metrics achieved via collecting offline samples from reading registers 1 and 0 while changing states (writing 1 to reg 26 control) and then fed sequentially to the algorithm

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.

The response from the system is a time series signal. The steps of the Change Detector are:

1. Receive a response from the system.
2. Determine if the state has changed.
3. Return {yes or no, confidence }.

There are several ways to analyze time series signals:

1. Distance Based
2. Feature Based
3. Deep Learning

Distance Based: using a metric to determine the similarity between two or more signals. Common metrics include Euclidean distance and Dynamic Time Warping (DTW). Since our response may have slight differences in timing, DTW is the best choice.

Feature Based: using calculated features of the signal such as the mean, standard deviation, skewness, kurtosis, etc. These features are then fed into a machine learning algorithm to determine whether or not the signal has changed.

Deep Learning: the signals themselves are feed into multi-layered neural networks. Similar to feature based methods except that the features are determined by the algorithms themselves. A downside to this method is that significant data is needed for good performance.

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 ?
What message should we send to the system in order to trigger a state change? The Input Generator should keep track of each message and the result from the Change Detector and use this information.

1. Receive answer from Change Detector.
2. Determine which message to query the system in order to try to change the state.

There are several possible methods to determine which message to send next:

1. Enumerate through all possibilities.
2. Active Learning, in particular, uncertainty sampling.
3. Genetic Algorithms to learn the best messages over time.

Additional Modules or Functionality

Protocol Specification: should be easy to switch from protocols.
Message Sender: also determined by protocol, the focus is on I2C for now.
Extract Protocol Method: given a list of messages that cause a state transition in the system, can a protocol be determined?