Baseline report – Identification of Vulnerable web application

Saeid Tizpaz Niari, Shyam sundar Ramamoorthy, Yue Zhang

1) **Data Extraction**: Data is extracted from two different benchmark application. We are using instrumentation tools to extract program features like function calls and scripts to record time of execution for each input. Considering time as normal random variable, we record time of execution for the same inputs 10 times to calculate mean and standard deviation.

2) **Applications**:
The applications are two micro-benchmark programs which are vulnerable to side-channels. As It takes different time for different function calls, the programs are potential to leak secrets. In this case, the inputs are a set of bit sequence of 0 and 1. Given the inputs, the programs will call functions which are determined by pattern of the benchmark:
   a) MSB: Only one function will be called for every user input. The function is corresponding to most significant bit 1 in the given program input. The vulnerability is the ability of an observer in time to detect the location of program most significant 1.
   b) RegEx: Input triggers particular set of functions corresponds to a pattern of bits in the input. For this example, we use 101 as the pattern. Given an input of a bit sequence, three functions correspond to the position of most significant pattern of 101 will be called.

3) **Linear Regression**: We apply linear regression, but we could not get convincing results out of it. We will continue to resolve the issues and figure out if it can be useful for our settings.

4) **Clustering and classification:**
K-means algorithm is used for cluster the data. We use time as the only feature to cluster samples (time domain). The mean of 10 recorded time is used as the feature for clustering. We considered weights for each samples such that a sample can belong to different clusters with different weights. This weights come from the normal probabilistic assumption about the time. The clustering step produces labels for each sample and its corresponding weights to belong to that label. As a result, the input data of classification step will include data labels, weights of each sample, and the corresponding function call. CART decision tree will be used to discriminate function calls based on the given labels (labels will be corresponding to time ranges).

	MSB	RegEx
Number of features	10	20
Number of samples	213	400
K for K-means	10	17
Accuracy	100%	100%
K for k-fold validation	5	5

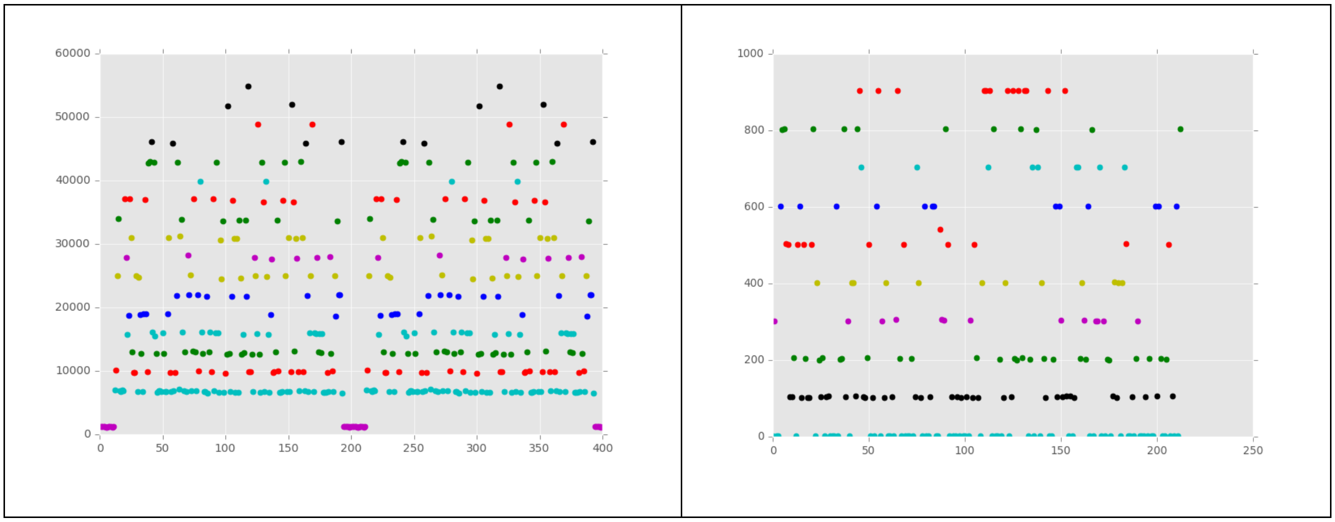

Fig: Clustering for MSB and RegEx benchmark program using time feature versus ID of samples

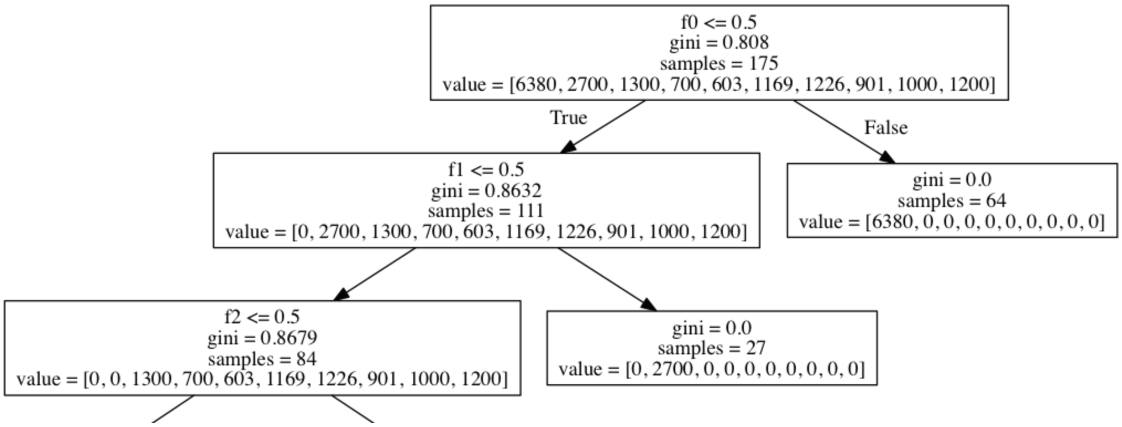
f0 <= 0.5
gini = 0.808
samples = 175
value = [6380, 2700, 1300, 700, 603, 1169, 1226, 901, 1000, 1200]
True
False
f1 <= 0.5
gini = 0.8632
samples = 111
value = [0, 2700, 1300, 700, 603, 1169, 1226, 901, 1000, 1200]
gini = 0.0
samples = 64
value = [6380, 0, 0, 0, 0, 0, 0, 0, 0, 0]
f2 <= 0.5
gini = 0.8679
samples = 84
value = [0, 0, 1300, 700, 603, 1169, 1226, 901, 1000, 1200]
gini = 0.0
samples = 27
value = [0, 2700, 0, 0, 0, 0, 0, 0, 0, 0]

Fig: Part of Decision tree produced to discriminate function calls in MSB benchmark pattern

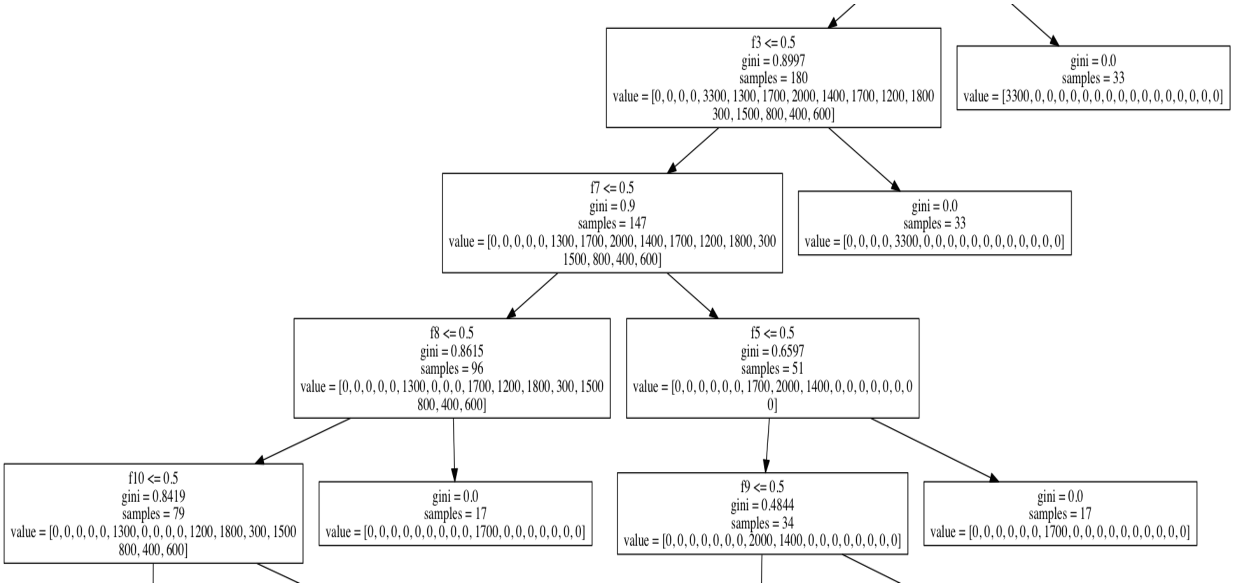
f3 <= 0.5
gini = 0.8997
samples = 180
value = [0, 0, 0, 0, 3300, 1300, 1700, 2000, 1400, 1700, 1200, 1800
300, 1500, 800, 400, 600]
gini = 0.0
samples = 33
value = [3300, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
f7 <= 0.5
gini = 0.9
samples = 147
value = [0, 0, 0, 0, 0, 1300, 1700, 2000, 1400, 1700, 1200, 1800, 300
1500, 800, 400, 600]
gini = 0.0
samples = 33
value = [0, 0, 0, 0, 3300, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
f8 <= 0.5
gini = 0.8615
samples = 96
value = [0, 0, 0, 0, 0, 1300, 0, 0, 0, 1700, 1200, 1800, 300, 1500
800, 400, 600]
f5 <= 0.5
gini = 0.6597
samples = 51
value = [0, 0, 0, 0, 0, 0, 1700, 2000, 1400, 0, 0, 0, 0, 0, 0, 0
0]
f10 <= 0.5
gini = 0.8419
samples = 79
value = [0, 0, 0, 0, 0, 1300, 0, 0, 0, 0, 1200, 1800, 300, 1500
800, 400, 600]
gini = 0.0
samples = 17
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1700, 0, 0, 0, 0, 0, 0, 0]
f9 <= 0.5
gini = 0.4844
samples = 34
value = [0, 0, 0, 0, 0, 0, 0, 2000, 1400, 0, 0, 0, 0, 0, 0, 0, 0]
gini = 0.0
samples = 17
value = [0, 0, 0, 0, 0, 0, 1700, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Fig: Part of Decision tree produced to discriminate function calls in RegEx benchmark pattern