Fault Injection with a new flavor: Memetic Algorithms make a difference

Stjepan Picek, Lejla Batina, Pieter Buzing, Domagoj Jakobovic

iCIS – Digital Security, Radboud University, The Netherlands

Faculty of Electrical Engineering and Computing, Zagreb, Croatia

Riscure BV, The Netherlands

14th April 2015

Outline

Introduction

- A fault injection (FI) attack is successful if after exposing the device to a specially crafted external interference, it shows an unexpected behavior exploitable by the attacker.
- Insertion of signals has to be precisely tuned for the fault injection to succeed.
- Finding the correct parameters for a successful FI can be considered as a search problem where one aims to find, within minimum time, the parameter configurations which result in a successful fault injection.
- The search space is typically too large to perform an exhaustive search.

Goal

- Use heuristics to find search space parameters that lead to successful attack.
- Use genetic algorithm (GA) and local search (LS), combine them into a memetic algorithm (MA).
- Conduct small number of measurements: 250 for every experiment.
- Perform voltage glitching.
- **Two scenarios**:

Goal

- Use heuristics to find search space parameters that lead to successful attack.
- Use genetic algorithm (GA) and local search (LS), combine them into a memetic algorithm (MA).
- Conduct small number of measurements: 250 for every experiment.
- Perform voltage glitching.
- **Two scenarios**:
  - Finding faults in minimal number of measurements.

Goal

- Use heuristics to find search space parameters that lead to successful attack.
- Use genetic algorithm (GA) and local search (LS), combine them into a memetic algorithm (MA).
- Conduct small number of measurements: 250 for every experiment.
- Perform voltage glitching.
- **Two scenarios**:
  - Finding faults in minimal number of measurements.
  - Characterizing the parameter space, again in a minimal number of measurements.

Smart card details

- Based on ATMega163+24C256 IC, realized in CMOS technology.
- Cards do not deploy any side-channel or fault injection countermeasure.
- We attack a vulnerable PIN authentication mechanism.
- Our approach makes no assumptions on the software that runs on the smart card.

Smart card details

```
for (i = 0; i < 4; i++)
{
    if (pin[i] == input[i])
        ok_digits++;
}
if (ok_digits == 4)  //LOCATION FOR ATTACK
    respond_code(0x00, SW_NO_ERROR_msb, SW_NO_ERROR_lsb);
else
    respond_code(0x00, 0x69, 0x85);
```

Verdict classes

- FI testing equipment can output only verdict classes that correspond to successful measurements.
- Several possible classes for classifying a single measurement:
  1. NORMAL: smart card behaves as expected and the glitch is ignored
  2. RESET: smart card resets as a result of the glitch
  3. MUTE: smart card stops all communication as a result of the glitch
  4. INCONCLUSIVE: smart card responds in a way that cannot be classified in any other class
  5. SUCCESS: smart card response is a specific, predetermined value that does not happen under normal operation

Search space parameters

- We divide search space parameters into two groups.
- The first group consists of parameters that we influence with an external search space algorithm
- The second group consists of parameters that we leave for fault injection framework to set randomly.
- First group include:
  - Glitch length
  - Glitch voltage
  - Glitch offset
- Those parameters define the electrical effect on the target.
- Second group consists of wait cycles and number of glitch cycles parameters.

Approach

- Looking for spots that lead to the successful FI can be considered as an optimization problem.
- We want to find as much "good" spots as possible in the minimal amount of time.
- Recall, there are two scenarios of interest.
- The first one is finding as many successful parameter combinations as possible.
- In this scenario, we can expect that more attempts should be made with parameter values that resemble those that led to a fault.

Approach

- The second one aims to map the parameter space into regions with the same behavior outcome of the smart card.
- We expect that the parameter combinations that result in the same behavior form regions in the search space which are adjacent to regions with different target behavior.

Genetic algorithm

- GA belongs to a subclass of evolutionary algorithms where the elements of the search space are arrays of elementary types.
- We customize the standard version of GA:
  - Map verdict classes to fitness values.
  - Give higher values to verdict classes that are of a bigger importance.
  - For NORMAL class we give the value 1, RESET/MUTE and INCONCLUSIVE classes we consider the same and we give them a value 2, for SUCCESS class we give a value 3.
  - New operator: local crossover.

Genetic algorithm

Input: crx_count $= 0$, mut_count $= 0$

1: **repeat**
2: select first parent
3: **if** first parent of SUCCESS class **then**
4: try to find matching second parent
5: **else**
6: try to find second parent of different class
7: **end if**
8: perform crossover (depending on parent classes)
9: copy child to new generation
10: crx_count $=$ crx_count $+ 1$
11: **until** $p_c * N <$ crx_count
12: **repeat**
13: select random individuals for tournament
14: copy best of tournament to new generation
15: mut_count $=$ mut_count $+ 1$
16: **until** $(1 - p_c) * N <$ mut_count
17: perform mutation on new generation with probability p_m
18: evaluate population

Tabu search

- There exist only a few different verdict classes.
- We do not want to test the same measurements more than once.
- We use a technique from Tabu Search (TS) optimization method.
- TS declare certain solution candidates that have already been visited as *tabu* and therefore not to be visited again.
- We do not use all functionalities from TS.

Local search

- Local search (LS) works on a single solution and generally transcend only to neighbors of the current solution.
- We use one version of the divide-and-conquer algorithm where each new solution is located in the middle of parent solutions (binary search).
- Define distance metrics:
  - **Euclidean distance** between two points $\vec{a}$ and $\vec{b}$ in an n-dimensional space is equals $d(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$.
  - **Manhattan distance** between two points is the sum of absolute differences of their Cartesian coordinates and it equals $d(\vec{a}, \vec{b}) = \sum_{i=1}^{n} |a_i - b_i|$.

First scenario

- Compare with random search and exhaustive search.

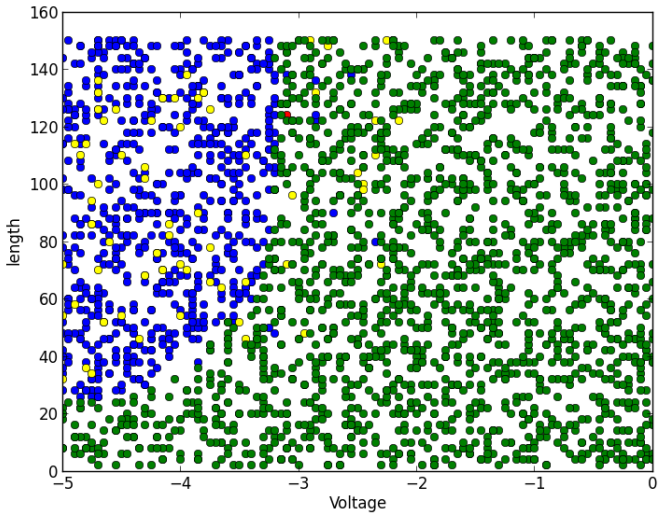

Figure: Random search, 2 500 measurements

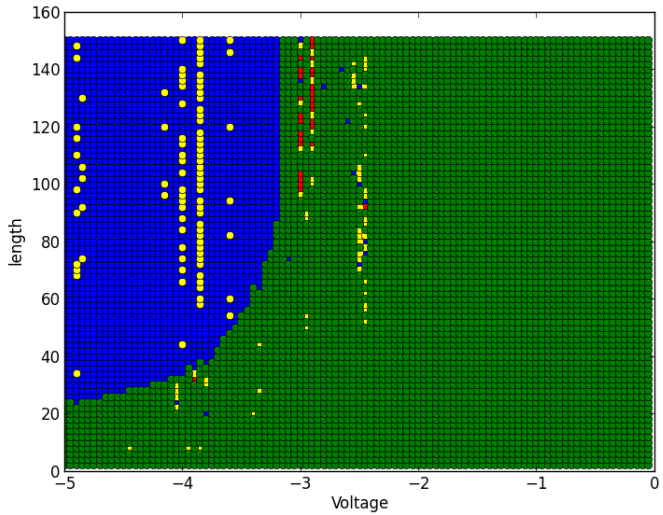

Figure: Exhaustive search, 7 500 measurements

First senario

- We conduct several runs of different algorithm versions and then we present averaged values.
- Columns Normal, Reset and Success show average number of NORMAL, RESET/MUTE and SUCCESS measurements.

Algorithm	Normal (%)	Reset (%)	Success (%)
GA+TS	58.08	39.97	1.94
GA+TS+LS, Euclidean	55.29	41.87	2.84
GA+TS+LS, Manhattan	62.76	36.45	0.78

Second scenario

- Here, we are not interested in SUCCESS points, we treat them as RESET/MUTE.
- We are looking for phase transitions.

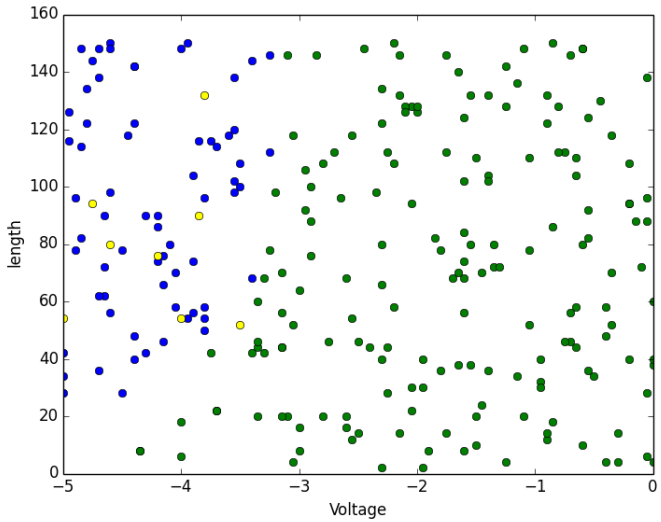

Figure: Random search

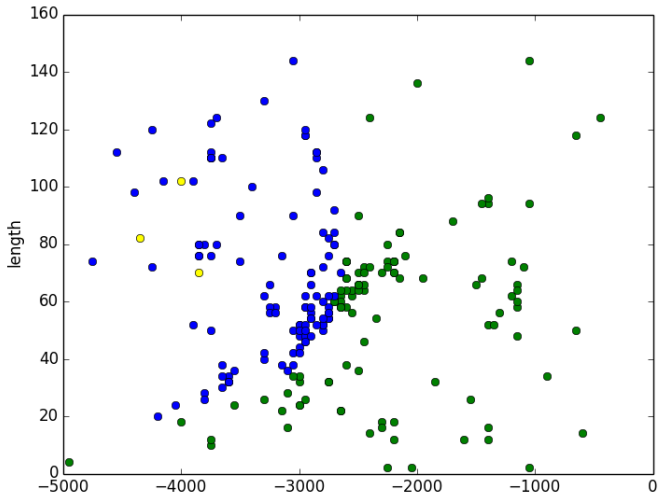

Figure: GA + TS

Second scenario

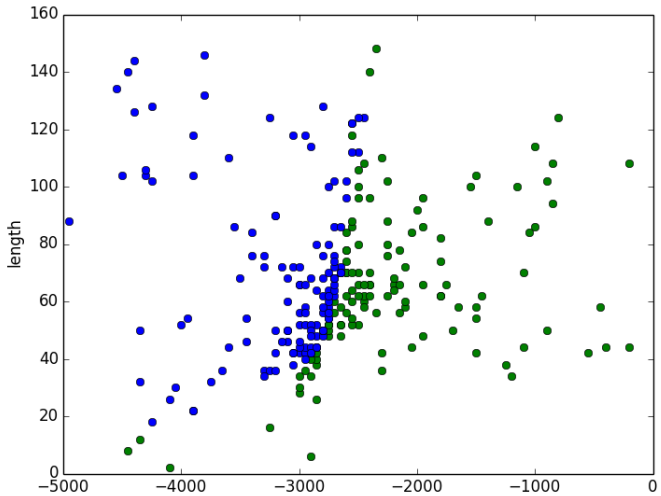

Figure: GA + TS + LS, Manhattan

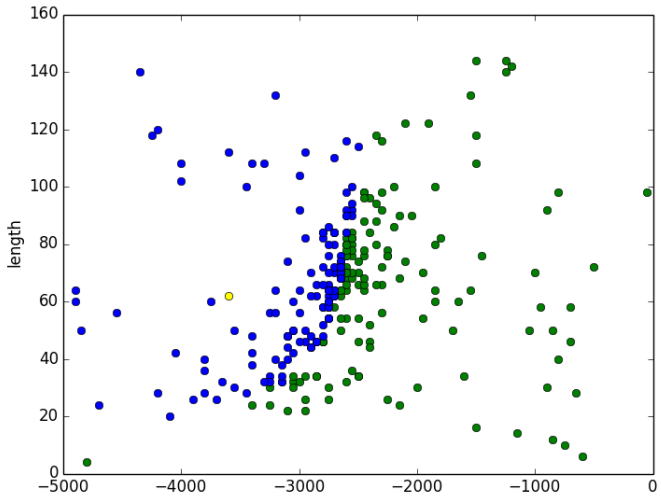

Figure: GA + TS + LS, Euclidean

Conclusion

- Our experiments with the MA show that one can successfully find faults with a limited number of measurements.

Conclusion

- Our experiments with the MA show that one can successfully find faults with a limited number of measurements.
- Additionally, we can characterize interesting search space regions.

Conclusion

- Our experiments with the MA show that one can successfully find faults with a limited number of measurements.
- Additionally, we can characterize interesting search space regions.
- We do not claim this is the best possible method for finding FI search space parameters.

Future work

- Explore more heuristics.

Future work

- Explore more heuristics.
- Compare with previous works.

Future work

- Explore more heuristics.
- Compare with previous works.
- Experiment with smart cards that have implemented countermeasures.

Future work

- Explore more heuristics.
- Compare with previous works.
- Experiment with smart cards that have implemented countermeasures.
- Apply it to other glitching scenarios, like EMFI.

Questions

Thank you for your attention.

Questions?