|  |
| --- |
| Evolutionary Fuzzing Framework  Project Documentation  (Pit File Mode) |
| 2013-2014 |
| By Rakotyansky Maria |

Contents

[1. Introduction 4](#_Toc398744500)

[2. Structure 4](#_Toc398744501)

[3. Subproject Organization 5](#_Toc398744502)

[3.1. General Work Plan 5](#_Toc398744503)

[3.2. Meetings with the client 6](#_Toc398744504)

[3.3. Authors and tasks 6](#_Toc398744505)

[4. Theoretical Background 6](#_Toc398744506)

[4.1. Introduction 6](#_Toc398744507)

[4.2. General notions 7](#_Toc398744508)

[4.3. Selection Strategies 7](#_Toc398744509)

[4.3.1. Fitness Proportionate Selection (FPS) aka Roulette 7](#_Toc398744510)

[4.3.2. Tournament Selection 9](#_Toc398744511)

[4.4. Survival strategies 9](#_Toc398744512)

[4.4.1. Elitism 9](#_Toc398744513)

[4.4.2. Aging 10](#_Toc398744514)

[4.4.3. Genitor 10](#_Toc398744515)

[5. Technical Background 10](#_Toc398744516)

[5.1. Entity Relationship Datagram 10](#_Toc398744517)

[5.1.1. Description 10](#_Toc398744518)

[5.2. Data Flow Diagram 11](#_Toc398744519)

[5.2.1. Description 11](#_Toc398744520)

[5.3. Software Requirements Specification 11](#_Toc398744521)

[5.3.1. Description 11](#_Toc398744522)

[5.4. Main Windows Screenshots 12](#_Toc398744523)

[5.5. Peach Fuzzer Platform 15](#_Toc398744524)

[5.5.1. Introduction 15](#_Toc398744525)

[5.5.2. Principles of work 15](#_Toc398744526)

[5.5.3. Peach Fuzzer Version 16](#_Toc398744527)

[5.5.4. Peach Pit files 16](#_Toc398744528)

[6. Implementation and Testing Details 17](#_Toc398744529)

[6.1. Programmer’s Guide 17](#_Toc398744530)

[6.2. Test Cases 17](#_Toc398744531)

[6.3. User’s Guide 19](#_Toc398744532)

[7. Problems and difficulties 19](#_Toc398744533)

[7.1. Peach Fuzzer Platform faults 19](#_Toc398744534)

[7.2. Time of work 21](#_Toc398744535)

[8. Experiment results 21](#_Toc398744536)

[9. Conclusions 32](#_Toc398744537)

[10. References 32](#_Toc398744538)

[11. Appendices 34](#_Toc398744539)

[11.1. Appendix 1: Simple example of Peach pit file 34](#_Toc398744540)

# Introduction

Fuzzing is an approach to software testing where the target software being attacked with test cases generated by another program. The target software is then monitored for any crashes exposed by the processing of this input. The basic principles of fuzzing remain unchanged, but the mechanisms used to make the fuzzing process more effective have undergone serious changes.

Currently fuzzers can be subdivided into 3 groups:

* **Mutation Fuzzers**
  + This type is also called Dumb. It is based on the random mutation of the input.
* **Generation Fuzzers**
  + This type is also called Smart. It defines tests based on predefined input models.
* **Evolutionary Fuzzers** 
  + Here input generation is based on output evaluation.

In our project we tried to develop a simple and effective tool which by means of combination of evolutionary algorithms will be able to produce efficient input used in the process of fuzzing.

After the discussions it was decided that ***Evolutionary*** ***Fuzzing Framework Project*** will have 2 modes of working (subprojects) : ***Pit File Mode*** and ***Modified Input Mode***. The ***Pit File mode*** will use as population modified (mutated) PeachPits. The ***Modified Input Mode*** will use as population modified (mutated) input files created from the same PeachPit.

The current report describes only ***Pit File Mode*** of the ***Fuzzing Framework Project.*** For the purpose of simplicity ***Pit File Mode*** will subsequently be referred as ***Subproject***.

# Structure

The current report is structured as follows:

* **Subproject Organization:** defines the roles and relationships in the development team, subproject timeline and milestones.
* **Theoretical Background:** contains detailed explanation of genetic algorithms technique and, in particular, the ones used in the current subproject.
* **Technical Background:** contains DFD model and SRS documentation used in the product design phase. The chapter also includes description of the Peach Fuzzer Platform and methods of work with it.
* **Implementation and Testing Details:** will describe the steps taken toward implementing the features outlined in the previous sections. The Use Case Diagrams and Activity diagrams will be provided in this section as well. The section will contain class and method signatures, for convenience. Also will include user’s guide and list of test cases.
* **Problems and difficulties**: contains information about technical and external problems and difficulties that influence upon the implementation process and final results
* **Experiment Results:** contains analysis of data produced during the experiment. We will try to interpret results and find out which of the proposed algorithms turned out to be the most effective one.
* **Conclusions:** things learned during the course of the project, recommendations and ideas for improvement.
* **References:** this part includes list of Internet links and literature used in the process of development.
* **Appendices**

# Subproject Organization

# General Work Plan

* Ramp up on “fuzzing” and “evolutionary algorithms”. Read literature.
* Select open source fuzzer. In our case it is a Peach Fuzzing Platform. Learn and start working with it.
* Implement a fuzzing framework.
* Develop “fitness function” (as defined in genetic algorithms)
* Compare between the various genetic algorithms.
* Analyze and prove the chosen approach efficiency.
* Develop automation that uses the chosen technique to generate inputs.
* Prepare final documents (e.g. “User’s Manual”, presentation with explanations, etc.)

# Meetings with the client

Meetings took place on average 2 times a week. Except for this we contacted the client by mail to discuss some points. Each meeting included three stages: discussion of the work done, changes and correction that need to be inserted, discussion of the next project step and creating of an approximate work plan for the next 2-3 weeks. The stages and tasks that were coordinated and discussed with client during these meetings can be seen in the next chapter.

# Authors and tasks

The work on Subproject development was done by Maria Rakotyansky under the supervision of Mr. Neer Roggel (Intel).

The fulfilled tasks were as follows:

* Read necessary literature, study tutorials (References see on pp. 31-32)
* Study work of Peach Fuzzer Platform. Prepare list of PeachPits of different formats and list of programs to be attacked.
* Develop graphics of the framework including Windows of Pit File Mode, Genetic Strategy Selection for Pit File Mode and Statistics for Pit File Mode.
* Develop classes Citizen, Parameters, Statistics, modules GA Framework, Mutation Functions.
* Run experiments with different genetic approaches and analyze results.
* Prepare the following documentation: Evolutionary Fuzzing Framework documentation (Pi t File Mode), Evolutionary Fuzzing Framework Software Requirement Specification (Pit File Mode), Programmer’s Guide (Pit File Mode), User’s Guide (Pit File Mode), presentation.

# Theoretical Background

# Introduction

Genetic algorithms are a model of machine learning which mimics the evolutionary processes in nature. They were introduced in the United States in the 1970s by John Holland at University of Michigan. This method finds application in computational science, economics, chemistry, bioinformatics, etc.

# General notions

* **Population**

To use the evolutionary approach a population of individuals should be created where an individual (creature, citizen) represents solution to the problem.

* **Genome**

Each candidate solution has a set of properties called genomes (chromosomes). These properties can be altered.

* **Fitness**

The fitness value determines how good the individual is at competing at its environment. A good and effective fitness function should take into consideration all characteristics of a desired solution.

* **Selection**

Selection is the choice of candidates for reproduction. The strategy is usually determined by the fitness measure. Very often only a small part of population takes part in breeding and replacement of the previous generation.

* **Crossover**

Crossover is the process of using genes of a pair of existing parents to create a new child. It involves splitting source individuals in accordance to predefined rules and swapping chunks of their chromosomes.

* **Mutation**

It is a random process (probability might be redefined) which slightly modifies the individual’s gene. Mutation is a good way to avoid premature convergence.

# Selection Strategies

## Fitness Proportionate Selection (FPS) aka Roulette

It is a genetic operator used in genetic algorithms for selecting potentially useful solutions for recombination.

In FPS parents are selected in accordance with their fitness. The better the genes are, the more chance has an individual to be chosen. This mimics nature in that fitter individuals will tend to have a better probability of survival and will go forward to form the mating pool for the next generation. But weaker individuals still have a chance to be chosen too. And the also may include some genetic information that might be useful for future generations.

To illustrate this approach let’s imagine a roulette wheel where each individual takes section in accordance to its fitness See **Figure 1**.

The algorithm works as follows:

* Let’s define N as a number of citizens in population
* Calculate sum of expected values in the population. Call the sum R.
* Repeat N times:
  + Choose random integer k between 0 and R.
  + Loop through individuals summing the expected values until the sum is greater or equal to k. elect citizen whose expected value puts the sum over the limit.

FPS method has a very important disadvantage. A few highly fit members can rapidly take over if the rest of population is much less fit. It will prevent GA from any further exploration. In other words evolution will almost halt. The effect is known as “premature convergence”.

To solve the problem “scaling” methods are used. Examples are Exponential Scaling, Sigma Scaling, and Power Law Scaling. All these methods permit to reduce the influence of the strongest subjects.

In the project was used Exponential Scaling proposed by S. R. Ladd, which consists of taking the square roots of fitness.

**Figure 1** Illustration of Roulette Wheel Selection (aka FPS)

## Tournament Selection

Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for breeding.

To understand the principle let’s have a look at the example. Suppose we want to choose a pair of parents.

The algorithm works as follows:

* Randomly choose from the current population a small number of individuals (typically less than 10). Call it K.
* From K choose the fittest citizen. It will be Parent 1.
* Repeat steps for parent 2.

The Tournament Selection is a bit more complicated but provides a higher diversity.

# Survival strategies

## Elitism

Using of mutations and crossover gives a big chance to lose useful genomes. The idea of elitism is in saving the individuals with best chromosomes for future generations. After evaluation of population members N best of them are chosen to be included to the next generation. Very often this method is combined with other approaches. For example, 10% of “elite” citizens and 90% of produced by one of traditional selection strategies. Using elitism increases the effectiveness of genetic algorithms.

## Aging

Age of individuals in the population is used to evaluate their suitability for participation in a new offspring’s’ creation. Age of a newly generated individual is taken as zero and every iteration is increased by one. Citizens of age less or equal to the maximum value are considered more fit for genetic operations than old ones. Such an approach allows individuals with worse genes take part in breeding what maintains more diversity in the population.

## Genitor

In the Genitor model a specific type of the steady state selection. Most genetic algorithms are generational, i.e. each generation consists of offspring produced from parents selected in the previous generation. In the steady state strategy only a small number of citizens is replaced, usually a predefined percent of the worst fit individuals.

The Genitor algorithm works as follows:

* Define the number of citizens to be replaced. Call this number T.
* Repeat T times:
  + Choose randomly 2 individuals from the population
  + Breed them and get an offspring.
  + Evaluate the offspring (determine fitness).
  + Replace the least fit citizen in the population with a newly produced citizen

# Technical Background

# Entity Relationship Datagram

## Description

**Entity Relationship Datagram (ERD)** is a data model for describing the data or information aspects of a business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database (Wikipedia).

The Evolutionary Fuzzing Project is not using data base to keep information, so this point is nor relevant.

# Data Flow Diagram

## Description

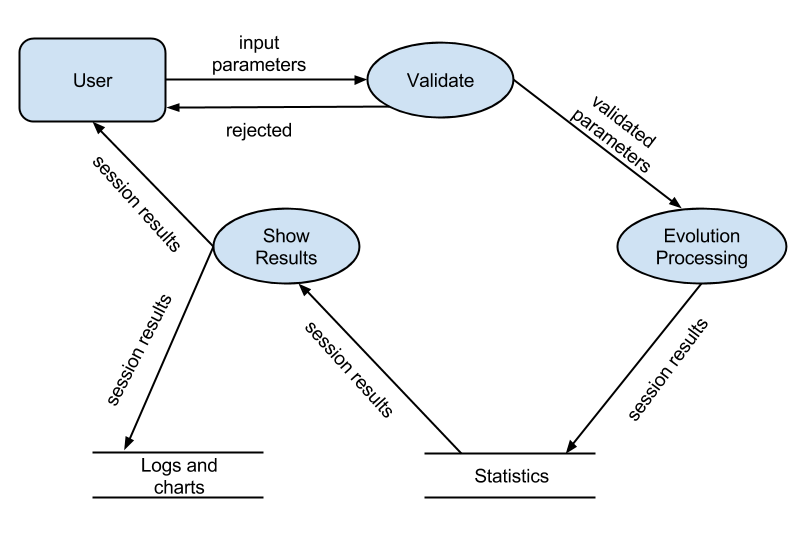
**Data Flow Diagram (DFD)** is a graphical representation of the "flow" of data through an information system modeling its *process* aspects (Wikipedia). See **Figure 2.**

# Software Requirements Specification

## Description

**See FuzzingProjectSRS.docx in the directory *EvolutionaryFuzzingFrameworkDocumentation*.**

**Figure 2:** DFD for the Evolutionary Fuzzing Project



# Main Windows Screenshots

**Window 1: Choose Mode Window**

As it was mentioned above the Project was subdivided into 2 modes. To pass to necessary mode user can choose it from the Dropdown List.

**Window 2: Pit File Mode**

Pit File Mode Window allows to choose paths to save log files, parameters for Peach Fuzzer , number of population.

**Window 3: Select GA Strategy**

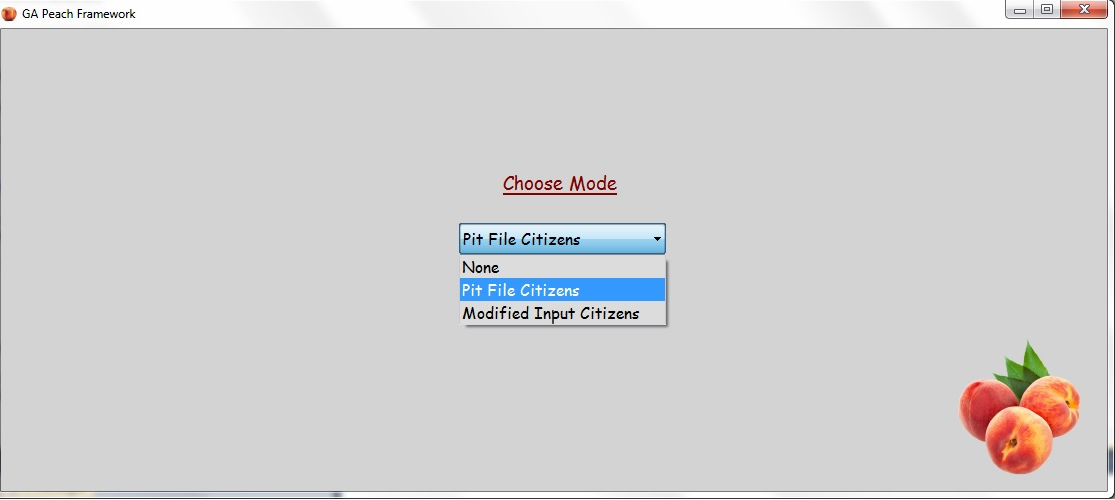
Select GA Strategy Window allows

choosing such parameters as probability of mutation, selection strategy, survival strategy, number of iterations, etc.

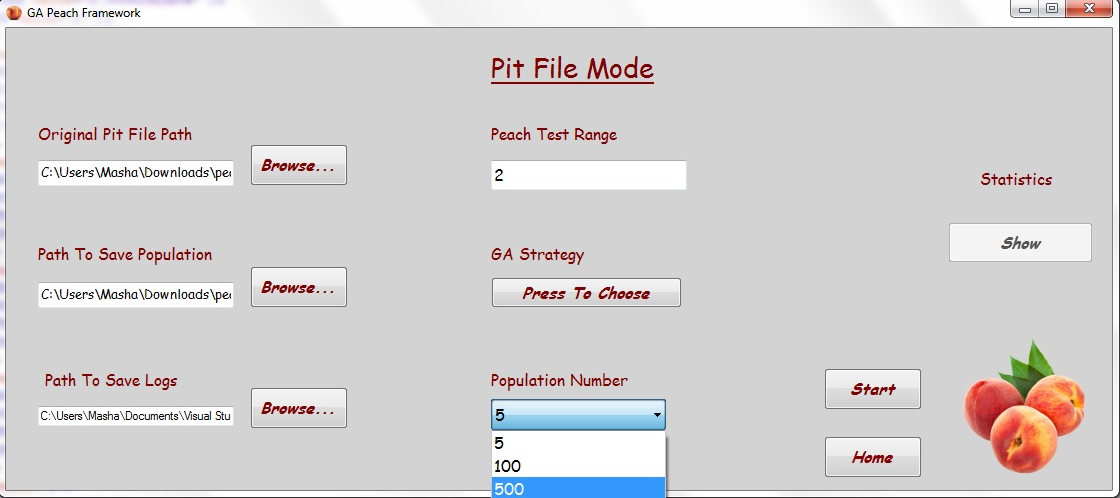
**Window 4: Statistics**

The Statistics window allows seeing all the parameters of the current run and the results in the form of chart.

**Window 1**: Choose Mode Window



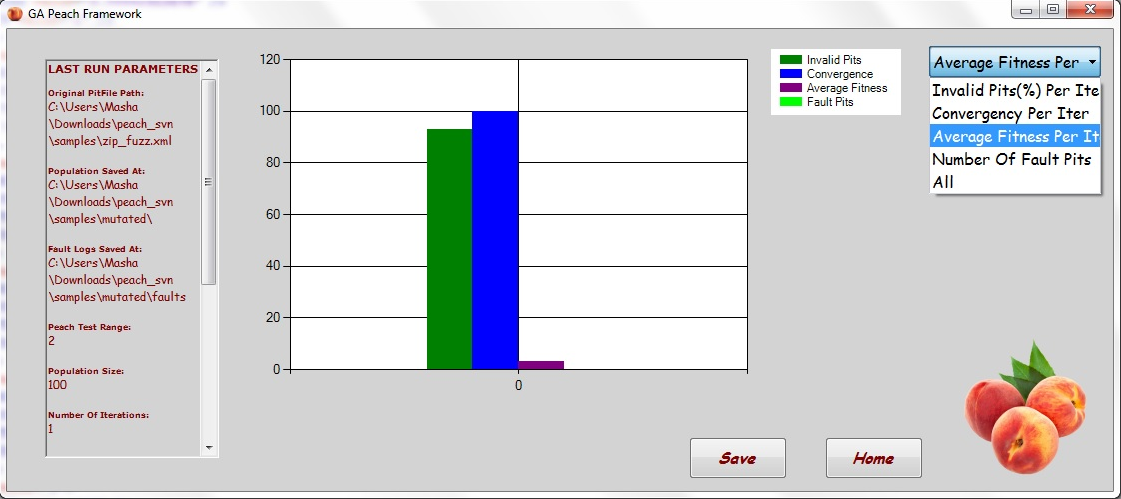
**Window 2**: Pit File Mode

****

**Window 3**: Select GA Strategy

****

**Window 4**: Statistics



# Peach Fuzzer Platform

## Introduction

Peach Fuzzer is an advanced fuzzing platform. It was designed to speed up the development of fuzzers for security researchers, security teams, consultants and companies. The tool is very flexible and allows modeling the data to be used in the process of fuzzing.

All the information about the product can be found on the page of the <http://peachfuzzer.com/> project.

## Principles of work

Peach fuzzes files using three techniques: modeling, mutating and monitoring.

**Modeling**

Peach uses data models which contain data format down to the field level. Based on your data models, Peach knows each data field’s size and format. Each time you test your software, Peach uses state models that tell Peach how to send data to and receive data from your product.

**Mutating**

Peach uses the information in your data models and parameters you supply and mutates your data files. Mutations are designed to reveal security vulnerabilities; they use your data’s existing values and either modify them or produce completely new data.

**Monitoring**

While Peach is running, it monitors your target and environment to identify faulting conditions and collects information you can use to investigate and remediate the faults. During the fuzzing session, Peach uses processes such as fault detection and data collection to monitor the fuzzing run results. Based on the parameters you supply, Peach can capture network traffic during a fuzzing iteration, attach a debugger to a process to detect crashes, or even restart a network service if it crashes or stops.

When fuzzing is complete, all the relevant information is saved in the Peach log files. It can easily be used to reproduce the issue in the code (with a debugger running) for further error analysis. This can help to determine where to correct the problem(s).

## Peach Fuzzer Version

The version used in the Subproject: Peach 2.3.9 (python)

Please take into the consideration the fact that it was slightly modified. Was added ability to return the average test case time through the log file status.txt. The modified version can be found in the directory ***Evolutionary Fuzzing FrameworkDocumentation***. See file **peach\_svn\_2.3.9.rar.** Installation instruction can be found on page <http://old.peachfuzzer.com/v2/PeachInstallation.html>.

## Peach Pit files

Peach Pit files are XML files that contain all of the information needed for Peach to perform a fuzzing run. It is necessary to modify the Peach Pit file to start fuzzing a target. The details and instructions on building valid Peach Pit Files can be found on page <http://old.peachfuzzer.com/v3/PeachPit.html>.

The list of ready Peach Pits which were used in the work on the Subproject is as follows:

* zip\_fuzz.xml – zip file format model
* wav\_fuzz.xml – wav file format model
* avi\_fuzz.xml-avi file format model

All of the templates were taking from this source : http://redmine.corelan.be/projects/corelanfuzztemplates/repository/show/peach and slightly modified for the needs of the Subproject.

Files can be found in the directory ***Evolutionary Fuzzing Framework Documentation***.A simple example of a Peach Pit file can be seen in ***Appendix 1.***

# Implementation and Testing Details

# Programmer’s Guide

Full Programmer’s Guide for a current Subproject can be found in the file **Programmer’s Guide (Pit File Mode).docx** in the directory ***Evolutionary Fuzzing Framework Documentation*.**

# Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Name** | **What to do** | **Expected** | **Result** |
| TC1 | Mode Choice | 1. Start program 2. Choose option “None” 3. Choose option “Pit File Citizens” 4. Choose option “Modified Input Citizens” | 1. Program starts 2. Nothing changes 3. User enters window “Pit File Mode” 4. Message “This part is not implemented in the current version of the program” | OK |
| TC2 | Insert incorrect data to text boxes | All existing text boxes must be tested.   1. Insert empty string. Press “Start” 2. To text boxes expecting paths insert wrong format.   Press “Start”.   1. To text boxes expecting numbers insert wrong format (not numbers). Press “Start”. | 1. In all cases should appear the message “Incorrect parameters. Check data you have inserted” | OK |
| TC3 | Insert correct data to text boxes | All existing text boxes must be tested.   1. Insert correct format parameters to text boxes. Press “Start”. | 1. Program must start session correctly. No error message is displayed. | OK |
| TC4 | Select data from combo-boxes | 1. . Try to select any line from each of combo-boxes. | 1. Data can be selected in each of combo-boxes. 2. Data in each combo-box lists corresponds to predefined range (full info can be found in User’s Guide) | OK |
| TC5 | Check “Browse..” buttons | 1. Click “Browse…” button | 1. All “Browse…” buttons are clickable and allow choosing path. Path automatically appears in a corresponding text box. | OK |
| TC6 | Check “Press to choose” button | 1. Click “Press to choose” button | 1. Appears window “GA Strategy Selection”. No error messages. | OK |
| TC 7 | Check “Apply” and “Return” buttons | 1. Click “Apply” button. 2. Click “Return” button | 1. Buttons must be clickable. 2. If all parameters correct no errors should appear 3. If some parameters are wrong message “Incorrect parameters. Check data you have inserted” | OK |
| TC 8 | Check button “Show” | 1. Try click button before starting the session 2. Try click button after running the session | 1. Button must be unlikable before starting the session. 2. Button must be clickable after the end of running. Should pass to the “Statistics” window without any error messages. | OK |
| TC 9 | Check information in the let text box of the statistics window | 1. Click “Show” button after the run end | 1. Information must coincide with the parameters inserted at the beginning of the session. | OK |
| TC 10 | Check combo-box in upper right corner of the “Statistics window” | 1. Try to select any line from each of combo-boxes | 1. Data can be selected in each of combo-boxes. 2. Data in each combo-box lists corresponds to predefined range (full info can be found in User’s Guide) 3. After selecting appears corresponding chart | OK |
| TC 11 | Check “Save” button | 1. Click “Save” button | 1. Button must be clickable 2. The button “Save” should be changed to “Update”. 3. The directory must be saved in a chosen path. 4. The directory must include pdf file with parameters, xml file with statistics, jpeg file with current chart. | OK |
| TC 12 | Check “Update” button | 1. Click “Update” button | 1. Button must be clickable 2. The current chart must be saved in a previously selected directory | OK |
| TC 13 | Check “Home” button | 1. Click “Home” button | 1. Button must be clickable 2. All parameters should be reset to default | OK |

# User’s Guide

Full User’s Guide for a current Subproject can be found in the file **User’s Guide (Pit File Mode).docx** in the directory ***Evolutionary Fuzzing Framework Documentation*.**

# Problems and difficulties

In the process of the project development it was noticed that the version of the Peach Fuzzer Platform has 2 problems that influence seriously on the expected results: faults and time of work.

# Peach Fuzzer Platform faults

The Peach Fuzzer Platform works in the following way:

It creates the object Template which includes information from the Peach Pit file, and in accordance to the template set defaults values from example given in pit. In case of error crack fails.

The information from example file is evaluated and has rating.

The rating is how well we matched:

1 - BEST 2 - GOOD 3 - OK 4 - MPH

If rating is less than 2 the Peach Fuzzer does not add this block to the template.

Let’s consider example from Peach Pit file for format zip:

**From file incoming.py** **rows 2489-2499**

if (pos+length) > len(buff.data):

try:

buff.read((pos+length) - len(buff.data))

except:

pass

**if (pos+length) > len(buff.data):**

**node.rating = None**

**return (4, pos) //4 is rating**

**Rows from Peach pit file for zip format:**

<String name="lfh\_Signature" valueType="hex" value="504b0304" token="false" mutable="false" />

<Flags name="lfh\_BitFlag" size="16" endian="little" signed="false">

**Explanation**: in String not given length = > used function “LookAhead”, pos (0-

26790) => that (pos+length) > len(buff.data) => Flags get rating 4 => loss of Block

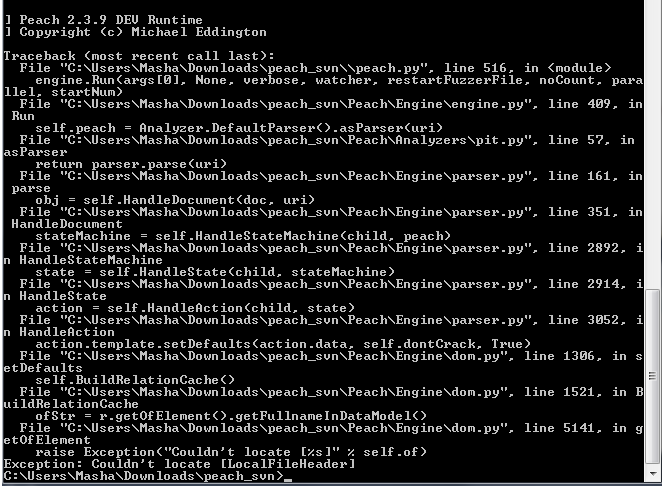
What does loss of Block mean? It means that template some information and in the process of mutation this information will not be found.

The peach version used in the project is not able to deal with this type of problem. And the resulting output can be seen in **Figure 3.**

It’s quite clear that such a fail influences badly upon the evolutionary fuzzing framework performance. Because it does not allow change the structure of the Peach Pit file freely, what seriously decreases the range of mutation.

For example, in case of Peach Pit file for zip format, changes of “size” field in tag “Number”, changes of “null terminated” and “token” fields in tag “String” provoke faults. It means they can’t be mutated in the evolutionary process.

**Figure 3**: Peach Fuzzer Platform fault trace



# Time of work

It also should be taken in consideration that time of test run is quite long. For example for zip format average test case time is about 2-3 seconds. What does it mean?

If we have a population of 100 citizens, and for each citizen peach test range is 30, then an iteration will take 3000\*3=9000 seconds or 50\*3=150 minutes. 10 iterations will run about 25 hours. In many cases such a small peach test range is not enough to provoke target software fault, what seriously complicates the process of data collecting.

# Experiment results

For the experiment Peach Pit file describing zip format was chosen. This Peach Pit file was created by Jason Kratzer. The zip\_fuzz.xml file can be found in the dictionary ***Evolutionary Fuzzing Framework Documentation*.** The latest Zip file format specification can be found here**:** [**http://www.pkware.com/documents/casestudies/APPNOTE.TXT**](http://www.pkware.com/documents/casestudies/APPNOTE.TXT)**.**

For testing purposes we have tries to re-identify vulnerabilities associated with the GlobalScape – Cutezip Archiver. The vulnerable copy was found here**:** [http://www.exploit-db.com/exploits/16162/.](http://www.exploit-db.com/exploits/16162/)

In the process of the experiment were tried different combinations of evolutionary algorithms. The results are described below. Detailed parameters, charts and faults information can be found in the dictionary “**Results**” in the dictionary ***Evolutionary Fuzzing Project Documentation*.**

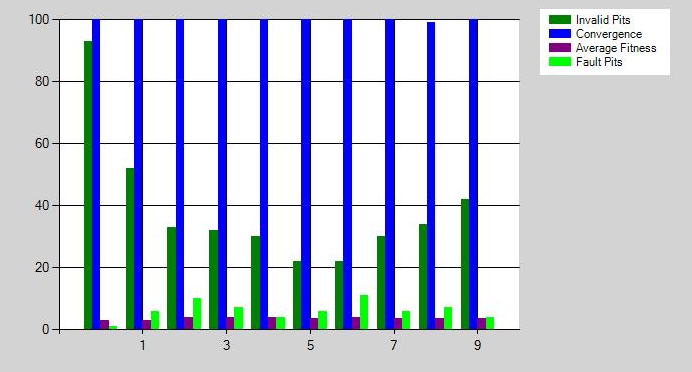
In each experiment:

* peach test range = 30
* population size = 100
* number of iterations = 10.

**Experiment 1:**

|  |  |
| --- | --- |
| Mutation Rate | 0.1 |
| Parents Selection Strategy | Naive |
| Fitness Strategy | All Bad Files |
| Percent of Children with Bad Genes | 100 |
| Survival strategy | Elitism |
| Elitism Rate | 10 |

Results per 10 iterations:

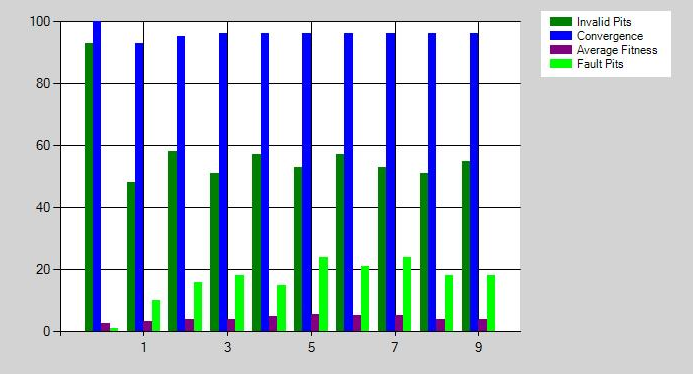


Full results see in directory 24\_08\_2014\_23\_52\_06

**Experiment 2:**

|  |  |
| --- | --- |
| Mutation Rate | 0.5 |
| Parents Selection Strategy | FPS |
| Fitness Strategy | -------- |
| Percent of Children with Bad Genes | ----------- |
| Survival strategy | Aging |
| Elitism Rate | 10 |
| Maximum Age | 3 |
| Scaling Used | False |

Results per 10 iterations:

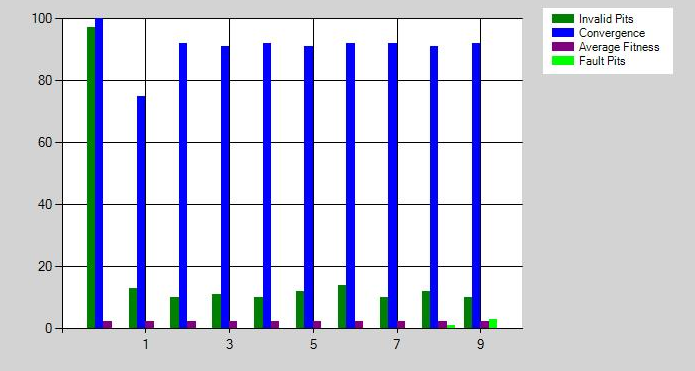


Full results see in directory 23\_08\_2014\_15\_24\_06

**Experiment 3:**

|  |  |
| --- | --- |
| Mutation Rate | 0.1 |
| Parents Selection Strategy | FPS |
| Fitness Strategy | -------- |
| Percent of Children with Bad Genes | ----------- |
| Survival strategy | Aging |
| Elitism Rate | 10 |
| Maximum Age | 3 |
| Scaling Used | False |

Results per 10 iterations:

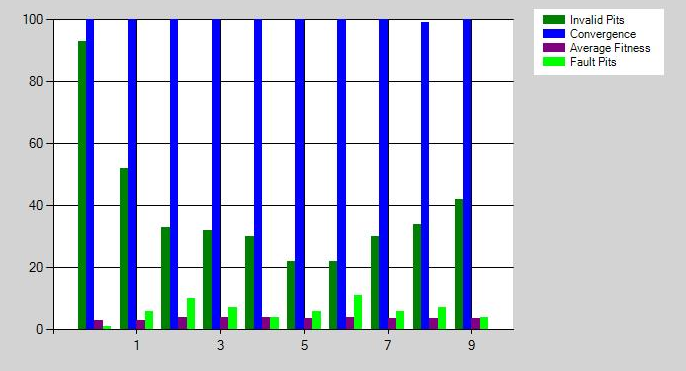
****

Full results see in directory 24\_08\_2014\_07\_34\_37

**Experiment 4:**

|  |  |
| --- | --- |
| Mutation Rate | 0.1 |
| Parents Selection Strategy | Naive |
| Fitness Strategy | Bad Files Partially |
| Percent of Children with Bad Genes | 50 |
| Survival strategy | Elitism |
| Elitism Rate | 10 |

Results per 10 iterations:

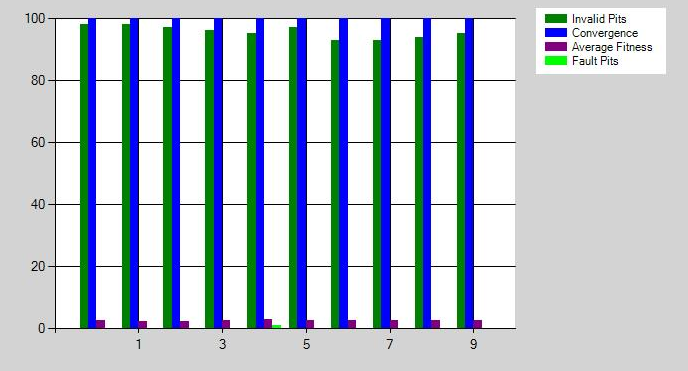
****

Full results see in directory 25\_08\_2014\_15\_05\_31

**Experiment 5:**

|  |  |
| --- | --- |
| Mutation Rate | 0.1 |
| Parents Selection Strategy | FPS |
| Fitness Strategy | ---------- |
| Percent of Children with Bad Genes | ---------- |
| Survival strategy | Elitism |
| Elitism Rate | 10 |

Results per 10 iterations:

****

Full results see in directory 02\_09\_2014\_20\_13\_20

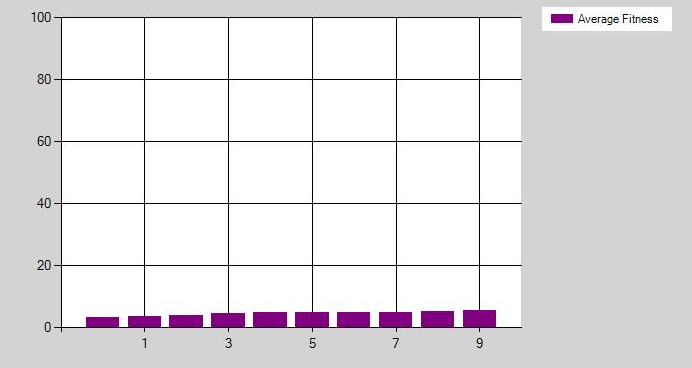
**Experiment 6:**

|  |  |
| --- | --- |
| Mutation Rate | 0.1 |
| Parents Selection Strategy | Tournament |
| Fitness Strategy | ---------- |
| Percent of Children with Bad Genes | ---------- |
| Survival strategy | Elitism |
| Elitism Rate | 10 |

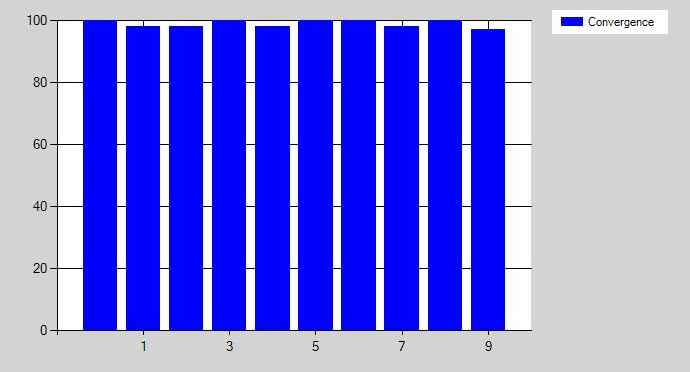
**NOTE**: By mistake the chart “All” was not saved, so below are given 4 charts describing the experiment results.

Results per 10 iterations:

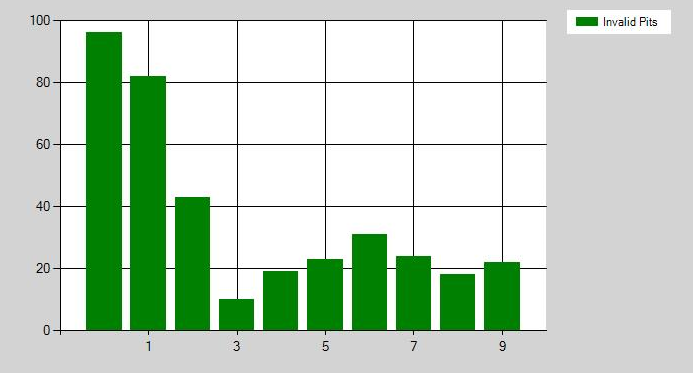
Average Fitness



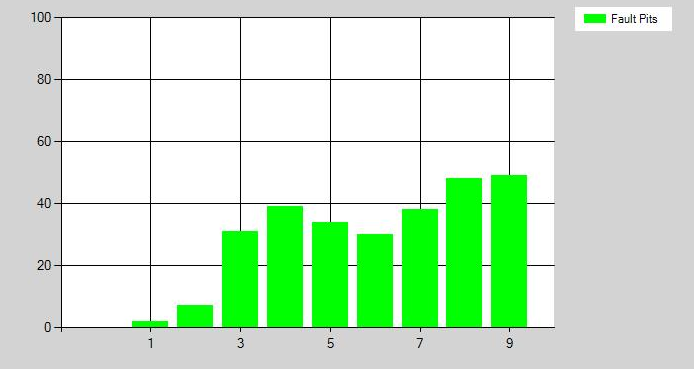
Convergence:



Invalid pits:



Fault Pits:



As we can see not all methods are equally effective. The worst results are given by experiments **3** and **5**. Both of them are using FPS selection strategy and both of them have a very low mutation probability. The problem here might be in the principle of FPS work (see chapter 4.3.1). It is based on breeding the candidates with the best fitness. In experiment 3 the number of valid pits in the 1st iteration is very high, so number of elements taking part in creation new children is very limited. And though the diversity due to the chart is very high, looks like differences are very tiny or having no meaning. The after-breeding mutation probability is very low, so as a result we get population with alike elements unable to provoke fault. Aging here has no special influence, as it starts working only from the 3d iteration.

Almost the same situation we see in experiment **5**.

But if we pay attention to experiment **2**, we’ll see that increasing mutation probability seriously improves final results. Here we found 3 different bugs (full outputs see in directory “Results”):

* EXPLOITABLE\_WriteAV\_0x0a617d5e
* UNKNOWN\_TaintedDataPassedToFunction
* UNKNOWN\_TaintedDataControlsBranchSelection
* UNKNOWN\_ReadAV\_0x445a4901

**NOTE**: Please pay attention that chart shows number of faults per iteration not making difference between types of faults, i.e. different citizens might bring to the same fault.

Now let’s have a look at experiments 1 and 4. The main difference between them is the percent of children that are allowed to inherit genes from invalid Peach pit files.

Experiment 1: all children, experiment 2: only 50% of children. How does that influence?

In the first experiment after 10 iterations total number of Peach Pit files that are able to provoke at least 1 fault is 61.

Found bugs are:

* EXPLOITABLE\_WriteAV\_0x0a617d5e
* UNKNOWN\_TaintedDataControlsBranchSelection
* UNKNOWN\_PosibleStuckCorruption
* UNKNOWN\_TaintedDataPassedToFunction

In the second experiment after 10 iterations total number of Peach Pit files that are able to provoke at least 1 fault is 94.

Found bugs are:

* EXPLOITABLE\_WriteAV\_0x0a617d5e
* UNKNOWN\_TaintedDataControlsBranchSelection
* UNKNOWN\_PosibleStuckCorruption
* UNKNOWN\_TaintedDataPassedToFunction
* EXPLOITABLE\_DEPViolation\_0x4e515c57\_0x521c4e34
* PROBABLY\_EXPLOITABLE\_ReadAVonIP

The result might be explained in the following way: the second experiment provided a good balance between level of mutation (allowed invalid pit files, range of selection is wider) and inheritance of good genes (50% of children inherit genes only from valid files with fitness > 0). The first experiment provided a good level of diversity but is more random and does not guarantee preservation of good genes.

In the last experiment (number 6) was used selection strategy Tournament. It showed the best results. And again this approach provides a good balance between randomness and good genes preservation.

After 10 iterations total number of Peach Pit files that are able to provoke at least 1 fault is 277.

Found bugs are:

* EXPLOITABLE\_WriteAV\_0x0a617d5e
* UNKNOWN\_TaintedDataPassedToFunction
* UNKNOWN\_PosibleStuckCorruption
* UNKNOWN\_TaintedDataControlsBranchSelection
* UNKNOWN\_ReadAV\_0x445a4901
* PROBABLY\_EXPLOITABLE\_ReadAVonIP
* PROBABLY\_EXPLOITABLE\_DEPViolation\_0x4e515c57\_0x521c4e34

In conclusion it can be said that the evolutionary approach might be called promising. But still needs some more experiments with strategies on other software targets to understand dependencies and to be able to choose the most effective configuration to get the best possible results.

# Conclusions

During this year a lot of work has been done. Were learned a lot of new things.

Now we have deeper knowledge of C# (work with graphics, processing of xml files, etc.). We studied a lot of interesting information about software testing techniques, fuzzing in particular. We had a good chance to understand the idea of this approach which both simple and powerful.

Work with an open source project “Peach Fuzzer Platform” was also a very interesting experience. As it requires a lot of preliminary reparations (learn and understand usages of parameters, principles of creating Peach Pit xml files, etc.)

Another advantage is studying Genetics Algorithms: basics, principles, different configurations.

It took us time to learn them theoretically and then implement.

And of course we should not forget about the practice in project management, improvement of writing and presentation skills. We studied standards, principles of projects documentation. But what is more important we got a great opportunity to use our theoretical knowledge in practice.

It is quite clear that the project is not completed yet. And still there are a lot of ideas that can be implemented.

For example, not all the prepared Peach Pit files were used in experiment. And it would be very good to continue working with them and collect more data for analysis.

A good idea is to play with the framework parameters (population number, mutation rate, test range, etc.) and see how it influences the results.

It also might be useful to change and improve the fitness evaluation. For example take into consideration not only the number of faults, but also their quality (for example, exploitable or non-exploitable bugs). Also there’s necessity to think how to use and analyze the fault logs returned by the Peach Fuzzer Platform.

# References

**References for fuzzing testing technique:**

* Fuzzing Techniques: <https://www.owasp.org/index.php/Fuzzing>
* S.Bekhar, C. Bekhar, R. Groz, L. Mounier A Taint Based Approach for Smart Fuzzing
* Smart Fuzzing: <http://trace.tennessee.edu/cgi/viewcontent.cgi?article=2402&context=utk_graddiss>
* Evolutionary fuzzing: <http://www.defcon.org/images/defcon-15/dc15-presentations/DeMott_Enbody_and_Punch/Whitepaper/dc-15-demott_enbody_and_punch-WP.pdf>

**References for Peach Fuzzer Platform:**

* Peach Fuzzer page: <http://peachfuzzer.com/>
* Peach Example.pptx **(**see directory <DocumenationAndInformationUsed>**)**

**References for creating Peach Pit files:**

* Tutorial: <http://www.flinkd.org/2011/07/fuzzing-with-peach-part-1/>
* Peach 2.3: <http://old.peachfuzzer.com/v2/peach23.html>

**References for genetic algorithms:**

* GeneticAlgorithsBasics.pdf (see directory <DocumenationAndInformationUsed>)
* Beginners Guide to Evolutionary Algorithms: <http://www.perlmonks.org/?node_id=298877>
* Evolutionary algorithms: <http://www.talkorigins.org/faqs/genalg/genalg.html>
* Crossover: <http://geneticprogramming.us/Genetic_Operations.html><http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php> <http://cswww.essex.ac.uk/staff/poli/gp-field-guide/24RecombinationandMutation.html>
* Genetic Algorithms overview: <http://www.geocities.ws/francorbusetti/gabeasley1.pdf>

**References for documentation preparation:**

* ProgrammerGuideHowTo.pdf (see directory <DocumenationAndInformationUsed>)
* 830-1998SRS.pdf(see directory <DocumenationAndInformationUsed>)
* Business vs. System Cases.pdf (see directory <DocumenationAndInformationUsed>)

# Appendices

# Appendix 1: Simple example of Peach pit file

