(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 June 2010 (24.06.2010)

PCT

(10) International Publication Number
WO 2010/069587 A1

(51) International Patent Classification:
G06F 21/00 (2006.01) *G06F 11/36* (2006.01)

(21) International Application Number:
PCT/EP2009/009124

(22) International Filing Date:
17 December 2009 (17.12.2009)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
08 021 967.8 18 December 2008 (18.12.2008) EP

(71) Applicant *(for all designated States except US)*: **NEC EUROPE LTD.** [DE/DE]; Kurfuersten-Anlage 36, 69115 Heidelberg (DE).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BECKERS, Kristian** [DE/DE]; Oberes Bieth 4, 69221 Dossenheim (DE). **SEEDORF, Jan** [DE/DE]; Rathausstrasse 8, 69126 Heidelberg (DE). **HUICI, Felipe** [IT/DE]; Silbershohl 3A, 69221 Dossenheim (DE). **NICCOLINI, Saverio** [IT/DE]; Carl-Philipp-Fohr-Strasse 4, 69121 Heidelberg (DE).

(74) Agent: **ULLRICH & NAUMANN**; Patent- und Rechtsanwälte, Luisenstrasse 14, 69115 Heidelberg (DE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) **Title**: METHOD AND DEVICE FOR SUPPORTING PENETRATION TESTING OF A COMPUTER SYSTEM

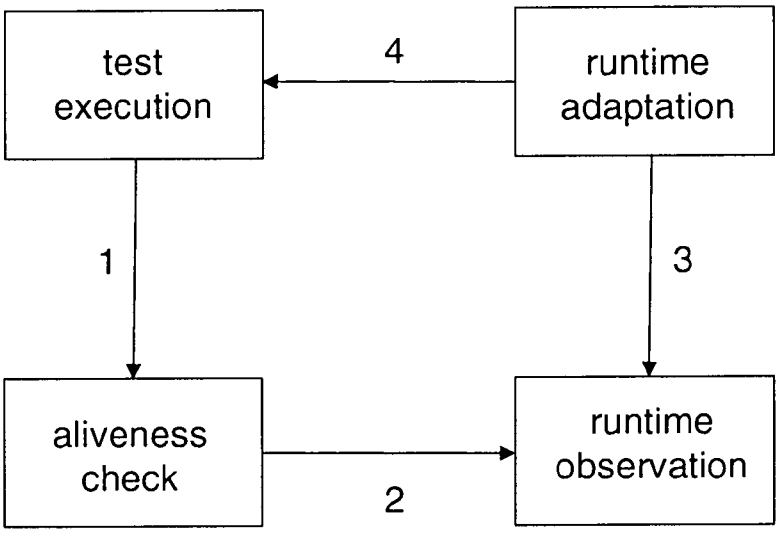

Fig. 1

(57) **Abstract**: A method for supporting penetration testing of a computer system, wherein a fuzzer performs black box fuzz testing for discovering possible vulnerabilities of a target application running on said computer system, wherein input test data is injected into said target application, said input test data being processed by said target application, is characterized in the steps of performing runtime observations of output data of said target application being a result of and/or a reaction to said input test data, and steering the fuzzing process towards possible vulnerabilities by performing a runtime adaptation of input test data in such a way that new input test data is generated in consideration of said runtime observations. Furthermore a corresponding device is described.

WO 2010/069587 A1

METHOD AND DEVICE FOR SUPPORTING PENETRATION TESTING OF A COMPUTER SYSTEM

The present invention relates to a method for supporting penetration testing of a computer system, wherein a fuzzer performs black box fuzz testing for discovering possible vulnerabilities of a target application running on said computer system, wherein input test data is injected into said target application, said input test data being processed by said target application.

Furthermore, the invention relates to a device for supporting penetration testing of a computer system, wherein a target application is running on said computer system, the device comprising a fuzzer being configured to perform black box fuzz testing for discovering possible vulnerabilities of said target application, wherein said fuzzer is further configured to generate input test data for being injected into said target application, and wherein said target application is configured to process said input test data.

In theory no computer system should be exploitable. However in practice systems are so complex that vulnerabilities can not be avoided, that are used to exploit systems. Thus many implementations (if not all) contain security vulnerabilities when they are initially deployed. This applies to software as well as to firmware, i.e. the software running in hardware solutions. Testing techniques for vulnerabilities can be classified into formal verification and penetration testing. Formal verification is a technique that builds a formal model based on the design and implementation of a computer system. Penetration testing is a testing technique that tries to exploit vulnerabilities on a system running under specific system characteristics, environment or state. Formal verification is highly complex and time consuming. In the field of penetration testing the number of possible combinations of input data to a program is so enormous that it is not feasible to test them all, nor exhaustively test manually. Automatic programs called fuzzers are designed specifically to create test data for applications.

White box, grey box and black box fuzzing are penetration testing mechanisms. In white box fuzzing the fuzzer has access to the source code of the application. In grey box fuzzing the tool tries to reverse engineer a model of the program. Black box fuzzing just considers input/output data of an application.

In practice devices and/or computer systems have to be tested in many cases where the source code is not accessible. For instance, companies that deploy third-party software – or hardware like VoIP terminals – have a strong desire to assess the security of such implementations while not having access to the source code. The analysis with grey box fuzz testing is very time-consuming, e.g. creating a control flow graph only from inputs and outputs. Moreover the increased time investment into grey box testing does not guarantee a better likelihood of finding vulnerabilities than other testing mechanisms. Thus black box fuzz testing is often the only sensible and possible method of choice.

The present invention focuses on black box fuzz testing. In the field of black box fuzz testing existing methods either a) rely on human knowledge of the problem domain to steer the fuzzing process or b) generate arbitrary input data or c) try to identify problem domains and generate input data which is described in the PROTOS Test-Suite for SIP (http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/).

Such approaches are very useful in case of testing local applications within a confined environment, but are hardly applicable when testing protocol/application stacks like for instance a SIP (Session Initiation Protocol) stack. Although some results can be obtained, in many cases these are limited to the processing of one single message (INVITE) lacking the capability to track the state of the remote application and to use behavioral level information in the fuzzing process. Hence, these approaches prove to be disadvantageous in terms of efficiency.

Exemplary it is further referred to the Paper *Abdelnur et al.: "KIF: a stateful SIP fuzzer", Proceedings on the 1st international conference on Principles, systems and applications of IP telecommunications, Iptcomm 2007, June 2007,* describing a stateful protocol fuzzer for SIP. The proposed method provides a fuzzer requiring a

state machine for the protocol which is tested and, hence, also proves to be tedious and disadvantageous in terms of efficiency.

It is therefore an object of the present invention to improve and further develop a method and a device of the initially described type for supporting penetration testing of a computer system in such a way that, by employing mechanisms that are readily to implement, an improvement in terms of efficiently discovering possible vulnerabilities is achieved.

In accordance with the invention, the aforementioned object is accomplished by a method comprising the features of claim 1. According to this claim such a method is characterized in the steps of performing runtime observations of output data of said target application being a result of and/or a reaction to said input test data, and steering the fuzzing process towards possible vulnerabilities by performing a runtime adaptation of input test data in such a way that new input test data is generated in consideration of said runtime observations.

Furthermore, the aforementioned object is accomplished by a device comprising the features of claim 17. According to this claim such a device is characterized in that the device comprises means for performing runtime observations of output data of said target application being a result of and/or a reaction to said input test data, and control means for steering the fuzzing process towards said possible vulnerabilities by performing a runtime adaptation of input test data in such a way that new input test data is generated in consideration of said runtime observations.

According to the invention it has first been recognized that in the context of performing black box fuzz testing the discovery of possible vulnerabilities can be considerably improved by taking into consideration the output data of the target application. Further, it has been recognised that this improvement can be achieved in a first step by performing run time observations of output data of the target application which are a result of and/or a reaction to the input test data. According to the invention the fuzzing process is steered in a next step towards possible vulnerabilities by performing a runtime adaptation of input test data. More specifically, the input test data is adapted in such a way that new input test data is generated in

consideration of the runtime observations. As a result, the deployment of the method and the device according to the present invention causes an improvement in terms of efficiently discovering possible vulnerabilities.

It is noted that the invention can be broadly applied to any system depended in any way on software and/or firmware for being functional. Therefore, the term computer system is synonymously used herein for any system, device or such like, which is depended on software and/or firmware for being functional.

Advantageously, the runtime observations may exclusively consider output data of the target application in relation to respective input test data. This means that the entire testing process can be executed without any knowledge of specific source code or any other detailed information of the computer system under test.

According to a preferred embodiment the input test data may consist of and/or include one or more attacks towards the target application. Such an attack may be a specific message, wherein it is assumed that the target application cannot handle this message.

According to a preferred embodiment the runtime observations may analyze the output data by evaluating and/or measuring values – vulnerability indicators – that indicate stress and/or malfunction of the target application as a result of the input test data. In particular, these vulnerability indicators are measured during the test runtime. The runtime observations are just used to figure out a way to a vulnerability by analyzing these indicators, for example indicators for a higher stress level on the computer system under test. These indicators may be used as a compass to direct the subsequent attack even more into the successful direction.

In a specific embodiment the reply time may be considered as a significant vulnerability indicator. Therefore, the runtime observations may include the step of measuring the reply time which is required by the target application to provide output data in response to the input test data. It is assumed that in case the target application requires more time to respond to the input test data, this is caused by difficulties the target application experiences in processing the input test data.

With respect to a further vulnerability indicator the runtime observations may include the step of evaluating the current state of the target application and determining deviations from expected states. In this context it is important to note that many protocols define a state transition diagram. In case of the computer system deviating from this definition, this may indicate a vulnerability. For instance, in case of the Session Initiation Protocol, the fuzzer may be expecting a ringing message, but instead the fuzzer receives an invite message. In order to accomplish these state determinations, the fuzzer requires a state machine for the protocol which is tested.

With respect to another vulnerability indicator the runtime observation may include the step of measuring whether the output data has unusual characteristics and/or obeys syntactic rules. On the one hand output data messages may obey the syntactic rules of the definition of the protocol, but may have unusual characteristics such as out-of-order headers (with respect to previous messages received from the tested computer system), garbled characters, unusual values for certain fields of the message, etc. On the other hand the computer system may reply with a malformed packet that does not comply with the definition of the tested protocol.

According to a preferred embodiment a regular flow of messages may be transmitted between the fuzzer and the target application. This regular flow of messages constitutes a form of "heartbeat". During this regular flow of messages being sent to the target application, the fuzzer sends a separate message flow that constitutes an attack to the target application. The timing of the response of the target application to the regular message flow is analyzed and may be used to assess the potential of the attacks. Thus, the variation in timing of the response to the regular message flow can be interpreted as a malfunction indicator.

Advantageously, aliveness checks may be performed for ascertaining the functional capability of the target application. Thereby it is tested whether an actual vulnerability is found and, hence, the computer system under test is crashed or whether the computer system is still operational.

With respect to variability and penetrating power the input test data may include a plurality of attacks belonging to different attack types within each fuzz testing run. In

a first step the input test data may contain a multifaceted and large number of different attack types which are executed. In the next steps the input test data may be concentrated to the well-proven and well-tried attack types.

To improve and optimize the quality of the attacks, it may be provided that within each fuzz testing run, attacks of the input test data may be rated with respect to their effect on the target application, and wherein the attacks for a subsequent fuzz testing run are generated on the basis of this rating. In particular, it may be provided that a genetic algorithm is employed for creating offspring attacks for a subsequent fuzz testing run.

Concerning the above mentioned creation of offspring attacks it may be provided that the offspring attacks are created from parent attacks (i.e. attacks employed in the preceding fuzz testing run) by means of performing combination, crossover and/or mutation operations. In case of combine and crossover operations offspring attacks are created from at least two parent attacks. The combine operation means the combination of the attacks of the parent attacks. The crossover operation requires common properties in the attack description. In order to generate a new attack a specific property may be randomly chosen from a parent attack. For performing a mutation operation a single parent attack is chosen and the strength of the attack is increased, wherein a description is required for increasing the attacks.

Advantageously, the black box fuzz testing may be terminated by detecting a vulnerability or by attaining a pre-defined number of fuzz testing runs. A maximum number of fuzz testing runs may be defined on the basis of an operator's requirements and/or resources, e.g. based on the desired accuracy of the testing process or the required reliability of the computer system under test.

According to a specific embodiment the attacks may be sorted in different categories, wherein at least one exemplary attack from each category is injected into the target application, and wherein attacks for subsequent fuzz testing runs are only taken from the most promising categories. Thereby an exhaustive testing of all attack types on a protocol may be provided by including an attack classification in the steering process of the fuzzer.

According to a further specific embodiment a protocol independent syntax generator for generating the input test data may be created from an xml-schema based syntax description.

There are several ways how to design and further develop the teaching of the present invention in an advantageous way. To this end, it is to be referred to the patent claims subordinate to patent claims 1 and 17 and to the following explanation of preferred examples of embodiments of the invention, illustrated by the figures. In connection with the explanation of the preferred examples of embodiments of the invention by the aid of the figures, generally preferred embodiments and further developments of the teaching will be explained. In the drawings

Fig. 1 is a flow diagram of an exemplary black box fuzz testing procedure of a method according to the present invention generally illustrating the single procedural steps within one fuzz testing run, and

Fig. 2 is another flow diagram of the fuzz testing procedure of Fig. 1 illustrating the single procedural steps in more detail.

Fig. 1 is a flow diagram illustrating an embodiment of a black box fuzz testing procedure which performs a runtime adaptation. In the illustrated embodiment the procedural steps within one fuzz testing run are depicted. In a first step a fuzzing test is executed against a target application running on a computer system, which is not explicitly shown, but which might be a SIP telephone, for instance. During the runtime of the fuzzing test observation values are measured by runtime observation methods. In a second step an aliveness check is executed for testing whether an actual vulnerability is found or whether the device under test is still "alive".

In a next step, in case the device under test is still alive, the runtime observation values are evaluated by the runtime observation. In this context the executed attacks are rated with respect to their effect on the device under test. Subsequently, in the forth step the runtime adaptation is performed, wherein, based on the rating of the attacks, it is decided which attacks and/or attack types being employed for the next

fuzz testing run. The newly created attacks are executed in the subsequent fuzz testing run and the circuit continues until a vulnerability is detected or a pre-defined number of fuzz testing runs is achieved.

Fig. 2 is a flow diagram illustrating the embodiment of the black box fuzz testing procedure of Fig. 1 employing a genetic algorithm. A genetic algorithm is a type of evolutionary algorithms. These imitate the principles of biological evolution. In each iteration they look at a whole population of possible solutions. These solutions are filtered according Darwins "Survival of the fittest". Genetic algorithms work with vectors that depict elements of the solution space. These vectors are called individuals. A set of individuals is a population. Through each iteration the current population becomes a new generation. The changing of the individuals is achieved by recombination or mutation of the parent individuals into children individuals and the following selection. Genetic algorithms have the best chances of finding the global optimum.

The specific genetic algorithm employed in Fig. 2 for black box fuzz testing has single attacks as individuals. As can be obtained from Fig. 2 the algorithm starts with an initial population of attacks. A fuzzer sends these attacks to the device under test. Subsequently, it is checked whether the objective is reached. The objective is either a found vulnerability or a number of iterations. Reaching the objective stops the execution of the algorithm.

In case the objective is not reached, each attack of the initial population is rated by the way of analyzing the output data generated by the device under test in response to the attack. The rating of an attack is an indicator for the likelihood that the attack may cause a malfunction (in an improved version employed in any of the subsequent fuzz testing runs). According to this rating individuals are selected that are promising candidates for procreating.

An offspring attack is created, for instance, by performing a combine and/or a crossover operator on two parent attacks of the preceding fuzz testing run. Combine is the combination of parent attacks. Crossover requires common properties in the attack descriptions. In order to build a new attack, a specific property, e.g. that the

attack message contains a specific character or character combination is randomly chosen from a parent attack. Mutation chooses a single parent attack and increases the strength of that attack, wherein a description is required for increasing the attacks. Furthermore, attacks may be kept in an unchanged form for a new offspring generation.

In the next steps the created offspring attacks are executed; the objective is checked and the offspring attack are rated in case of the algorithm not being aborted due to a found vulnerability. Furthermore, based on the rating a new population of attacks is build from survivors of the old population (i.e. attacks that are kept unchanged) and newly created offspring. The algorithm continues in a loop with creating offspring attacks again.

In the following a simple application scenario of the runtime adaptation is given. The explanation of this example is using the runtime adaptation steps illustrated in Fig. 1. The example begins with a series of attack types that the black box fuzzer should execute. These are attack types from A to C. The fuzzer builds now five instances for each attack type. In sum fifteen attacks exist now. These attacks are crafted network packets. In the case of stateful attacks one attack can also be a series of network packages. These are several uncrafted network packages to bring the tested device into a specific state and then one crafted network packet follows.

The fuzzer now executes the fifteen attacks. For each attack an aliveness check is executed, testing whether the device under test is still operational. In this example none of the fifteen attacks crashed the device. In the next step the runtime observation methods are evaluated. In this case an active heartbeat is used. This means messages are sent to the device under test during the execution of the attacks. The response times of the messages are used to rate the potential of the attacks. In the next step all fifteen attacks are sorted by their rating value and the worst fife are discarded. From the surviving ten attacks the runtime adaptation builds ten offspring attacks. These are executed against the tested device and rated again. In the next iteration new attacks are built from the best five offspring attacks and the best surviving attacks from the first iteration.

This loop is executed until either a pre-defined number of iterations is reached or a vulnerability is found. At best a vulnerability is found and the iterations show that in the last step of the loop only attacks from type A are used. The other types have not been reused, because their instances did not have promising runtime observation values.

The runtime observation can also be used to exhaustively test attacks sorted in categories. The fuzzer could exemplary test each category and thus focus on only the promising categories. Furthermore the described attacks comprise a function that can increase them in case any of them are tested and their runtime observation values give a reason to increase.

In order to make a method according to the present invention generic, i.e. useful for all possible text based protocols, a syntax generator for generating the input test data can be created from an xml-schema based syntax description. The xml-schema describes the form of the protocol and the lines of the protocol are described by regular expressions encapsulated in xml-schema elements.

Many modifications and other embodiments of the invention set forth herein will come to mind the one skilled in the art to which the invention pertains having the benefit of the teachings presented in the foregoing description and the associated drawings. Therefore, it is to be understood that the invention is not to be limited to the specific embodiments disclosed and that modifications and other embodiments are intended to be included within the scope of the appended claims. Although specific terms are employed herein, they are used in a generic and descriptive sense only and not for purposes of limitation.

Claims

1. Method for supporting penetration testing of a computer system, wherein a fuzzer performs black box fuzz testing for discovering possible vulnerabilities of a target application running on said computer system, wherein input test data is injected into said target application, said input test data being processed by said target application,
c h a r a c t e r i z e d i n the steps of
performing runtime observations of output data of said target application being a result of and/or a reaction to said input test data, and
steering the fuzzing process towards possible vulnerabilities by performing a runtime adaptation of input test data in such a way that new input test data is generated in consideration of said runtime observations.

2. Method according to claim 1, wherein said runtime observations exclusively consider output data of said target application in relation to respective input test data.

3. Method according to claim 1 or 2, wherein said input test data consists of and/or includes one or more attacks towards said target application.

4. Method according to any of claims 1 to 3, wherein said runtime observations analyze said output data by evaluating and/or measuring values – vulnerability indicators – that indicate stress and/or malfunction of said target application as a result of said input test data.

5. Method according to any of claims 1 to 4, wherein said runtime observations include the step of measuring the reply time required by said target application to provide output data in response to said input test data.

6. Method according to any of claims 1 to 5, wherein said runtime observations include the step of evaluating the current state of said target application and determining deviations from expected states.

7. Method according to any of claims 1 to 6, wherein said runtime observations include the step of measuring whether said output data has unusual characteristics and/or obeys syntactic rules.

8. Method according to any of claims 1 to 7, wherein a regular flow of messages is sent to said target application by said fuzzer, wherein said fuzzer sends a separate message flow that constitutes an attack to said target application, wherein the timing of the response of said target application to said regular flow of messages is analyzed.

9. Method according to any of claims 1 to 8, wherein aliveness checks are performed for ascertaining the functional capability of said target application.

10. Method according to any of claims 1 to 9, wherein, within each fuzz testing run, said input test data includes a plurality of attacks belonging to different attack types.

11. Method according to any of claims 1 to 10, wherein, within each fuzz testing run, attacks of said input test data are rated with respect to their effect on said target application, and wherein attacks for a subsequent fuzz testing run are generated on the basis of said rating.

12. Method according to any of claims 1 to 11, wherein a genetic algorithm is employed for creating offspring attacks for a subsequent fuzz testing run.

13. Method according to claim 12, wherein said offspring attacks are created from parent attacks by means of performing combination, crossover and/or mutation operations.

14. Method according to any of claims 1 to 13, wherein said black box fuzz testing is terminated by detecting a vulnerability or by attaining a pre-defined number of fuzz testing runs.

15. Method according to any of claims 1 to 14, wherein said attacks are sorted in different categories, wherein at least one exemplary attack from each category is injected into said target application, and wherein attacks for subsequent fuzz testing runs are only taken from the most promising categories.

16. Method according to any of claims 1 to 15, wherein a protocol independent syntax generator for generating said input test data is created from an xml-schema based syntax description.

17. Device for supporting penetration testing of a computer system, in particular for execution of a method according to any of claims 1 to 16, wherein a target application is running on said computer system, the device comprising a fuzzer being configured to perform black box fuzz testing for discovering possible vulnerabilities of said target application, wherein said fuzzer is further configured to generate input test data for being injected into said target application, and wherein said target application is configured to process said input test data,

c h a r a c t e r i z e d i n that the device comprises

means for performing runtime observations of output data of said target application being a result of and/or a reaction to said input test data, and

control means for steering the fuzzing process towards said possible vulnerabilities by performing a runtime adaptation of input test data in such a way that new input test data is generated in consideration of said runtime observations.

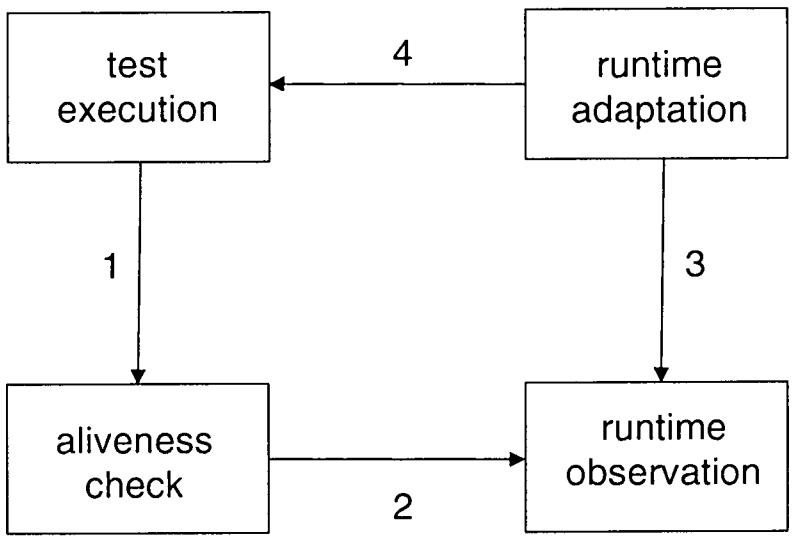
test execution
4
runtime adaptation
1
3
aliveness check
2
runtime observation

Fig. 1

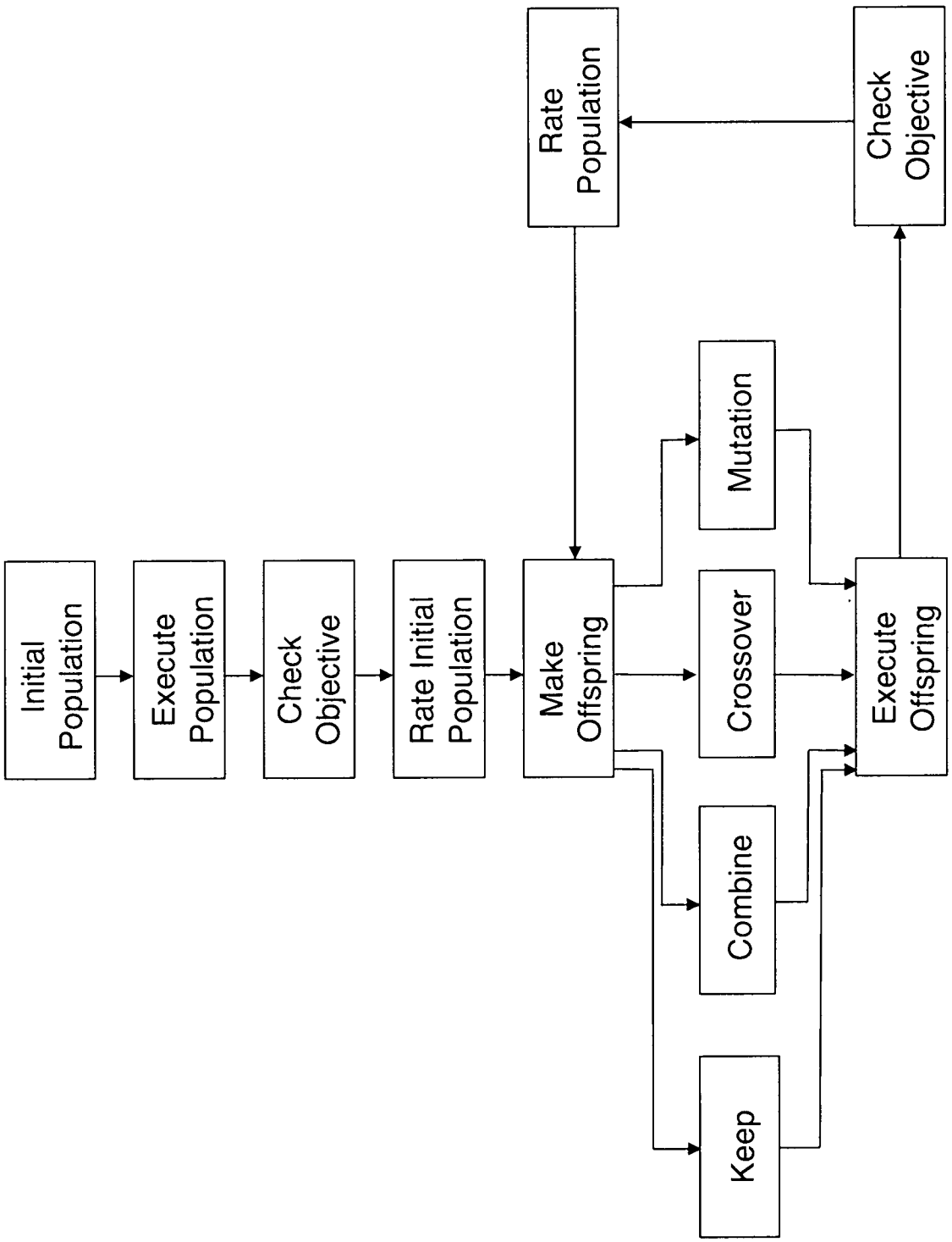
Initial Population
Execute Population
Check Objective
Rate Initial Population
Make Offspring
Mutation
Crossover
Combine
Keep
Execute Offspring
Rate Population
Check Objective

Fig. 2

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2009/009124

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F21/00 G06F11/36

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Timo Tamere: "Automatic Software Testing by Genetic Algorithms" University of Vaasa 8 April 2003 (2003-04-08), XP002573211 Retrieved from the Internet: URL:http://www.uwasa.fi/materiaali/pdf/isbn_952-476-003-7.pdf> [retrieved on 2010-03-11] sections 2.2.2, 2.2.3 page 95 - page 97 ----- -/--	1-6,8-17

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 March 2010	30/03/2010

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Veillas, Erik

1

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Jared D.Demott, Richard J. Enbody, William F. Punch: "Revolutionizing the Field of Grey-box Attack Surface Testing with Evolutionary Fuzzing" Internet 28 July 2007 (2007-07-28), 2 August 2007 (2007-08-02), XP002573212 Retrieved from the Internet: URL:https://www.blackhat.com/presentations/bh-usa-07/DeMott_Enbody_and_Punch/Whitepaper/bh-usa-07-demott_enbody_and_punch-WP.pdf> [retrieved on 2010-03-11] sections "Introduction", "EFS Overview, "EFS Initialization"" -----	1-4,6, 8-10, 12-17
X	Kayacik, H.G.; Zincir-Heywood, A.N.; Heywood, M.;: "Evolving successful Stack Overflow Attacks for Vulnerability Testing" Computer Security Applications Conference, 21st Annual (ACSAC 2005) 5 December 2005 (2005-12-05), 9 December 2005 (2005-12-09), XP002573213 Retrieved from the Internet: URL:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1565250> [retrieved on 2010-03-15] sections "1 Introduction" and "3 Methodology" including sections 3.1 and 3.11 the whole document -----	1,3-4,7, 10, 12-15,17
X	Mark Last, Shay Eyal and Abraham Kandel: "Effective Black-Box Testing with Genetic Algorithms" Springer Berlin / Heidelberg 3 March 2006 (2006-03-03), pages 134-148, XP002573214 ISSN: 1611-3349 ISBN: 978-3-540-32604-5 Retrieved from the Internet: URL:http://www.springerlink.com/content/w316n3854q861050/fulltext.pdf> [retrieved on 2010-03-11]	1-7,10, 17
A	the whole document -----	8

-/--

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Sherri Sparks, Shaw Embleton, Ryan Cunningham, Cliff Zou: "Automated Vulnerability Analysis: Leveraging Control Flow for Evolutionary Input Crafting" Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007) 14 December 2007 (2007-12-14), XP002573215 ISBN: 0-7695-3060-5 Retrieved from the Internet: URL:http://www.cs.ucf.edu/{czou/research/EvolutionaryInputCrafting-ACSAC07.pdf> [retrieved on 2010-03-15] the whole document	1,7
X	C. DEL GROSSO, G. ANTONIOL, M. DI PENTA, P. GALINIER AND E. MERLO: "Improving network applications security: a new heuristic to generate stress testing data" PROCEEDINGS OF THE 2005 CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 25 June 2005 (2005-06-25), - 29 June 2005 (2005-06-29) pages 1037-1043, XP002573368 Washington DC, USA ISBN: 1-59593-010-8 the whole document	1-4,6-7, 9,11-15, 17
X	MIDDLEMISS M J ET AL: "Weighted feature extraction using a genetic algorithm for intrusion detection" EVOLUTIONARY COMPUTATION, 2003. CEC '03. THE 2003 CONGRESS ON CANBERRA, AUSTRALIA DEC. 8-12, 2003, PISCATAWAY, NJ, USA,IEEE, vol. 3, 8 December 2003 (2003-12-08), pages 1669-1675, XP010707247 ISBN: 978-0-7803-7804-9 the whole document	1,10

1