Automated Software Vulnerability Testing Using In-Depth Training Methods

Alexandr Kuznetsov [1[0000-0003-2331-6326]], Oleksiy Shapoval [1[0000-0003-4478-3193]],

Kyrylo Chernov [1[0000-0003-2417-3793]], Yehor Yeromin [1[0000-0001-9720-746X]],

Mariia Popova [1[0000-0002-9751-1717]], Olga Syniavska [2[0000-0002-7507-3541]]

[1]V. N. Karazin Kharkiv National University, Svobody sq., 4, Kharkiv, 61022, Ukraine
kuznetsov@karazin.ua, alex.shapoval@protonmail.com,
kirillfilippsky@gmail.com, suvenick2@gmail.com,
mariia.popova26@gmail.com
[2]Sumy State University, Rymskogo-Korsakova st., 2, Sumy, 40007, Ukraine
o.syniavska@uabs.sumdu.edu.ua

Abstract. The article provides a view on modern technologies, which are used for automatic software vulnerability testing in critically important systems. Features of fuzzing realization (which is based on making many inputs with different mutated data) are also studied. As a result, testing algorithm picks input data that is more likely to cause a fail or incorrect work of software product. Deep learning algorithms are used to decrease the computational complexity of testing process. The use of simple fuzzer and Deep Reinforcement Learning algorithm shows that the amount of mutations necessary to find vulnerabilities decreases by 30%.

Keywords. Fuzzing, Testing, Reinforcement Learning, Q-Learning, Software Security.

1 Introduction

The development of modern computer technology leads to the emergence of new high-quality information services and their implementation in all spheres of human activity [1-5]. The development of the IT-industry has led to the construction of global computer networks, extensive data warehouses, automated control systems, including critical infrastructures, Smart Grid and much more [6-8].

In the age of the Internet and global implementation of information technologies, information security is more important for critical infrastructures [9, 10]. Complex solution for problems related to informational security is connected with solving different objectives in cryptography [11-14], computations optimization, technical and physical security as well as many others [10, 15-17].

This paper focuses on the problem of automated software vulnerability scanning [18-22]. As practice shows, computer programs are the most exposed fragment of modern IT infrastructure. Failure or configuration errors and undeclared operations can lead to disastrous consequences. Development and research of methods and tools for automated software vulnerability scanning is extremely important and relevant task.

2 Known Fuzzers and Their Analysis

Barton Miller, who is a professor at the University of Wisconsin, Madison, was the first to introduce the term "fuzzing" together with his students in 1989 [23]. Henceforward, the development of automated testing continued and the creators of fuzzing used this method to search for vulnerabilities in software using different operating systems including UNIX, Windows and Mac OS [23-26]. Today fuzzing is most commonly used in Software Quality Assurance. It is also one of the key steps of Microsoft Security Development Lifecycle (SDL). Software experts from Microsoft consider that deliberate input of wrong or random data is a sufficient and non-costly way to detect potential errors before product release [18-22].

Fuzzing testing has several advantages among which [23-26]:

- high speed (usually much higher than manual code review);
- no need to involve human work;
- fuzzer does not need to be controlled, while human capabilities are finite;
- scalability, i.e. if there is a need to find more vulnerabilities, the only thing needed is more fuzzers.

However, fuzzing has several disadvantages. For example, when fuzzing is used, it is very difficult to discover deep errors, such as business logic errors etc. [27]. It is relevant to conduct comparative analysis of different fuzzing methods and experimental researches using the most common software products as an example.

This work presents the results of analysis and comparative research of automated vulnerability search technologies. Particularly, the following common fuzzing utilities are reviewed [28-32]: American Fuzzy Lop, MiniFuzz, and Peach. Experimental researches are conducted for such everyday software products as Google Chrome; Notepad++; Winamp; Microsoft Paint.

American Fuzzy Lop (AFL) is an open-source fuzzer, which was developed by Polish computer security expert Michał Zalewski [29, 30]. The program uses genetic algorithms to automatically look for test cases. This fuzzer's main goal is to cause unexpected behavior of target programs by changing or shifting input channel bytes.

MiniFuzz was developed by Microsoft. This fuzzer is intended for simple and routine usage [31]. Its operating principle is forming data beforehand, then passing them to the target and catching errors. This utility belongs to dumb fuzzers category which conduct fuzzing randomly.

Peach is more advanced tool for "intellectual" fuzzing developed by Michael Eddington [32]. It supports not only mutation mode but also fuzz-file generation. Since program needs to know the structure of target files, specialized XML-documents are used as input. Peach can fuzz applications, servers, network protocols, drivers, internal protocols, devices, systems and so on.

During experimental research of automated vulnerability search effectiveness MiniFuzz was used. Fuzzing process was conducted for several common desktop applications, including Google Chrome; Notepad++; Winamp; Microsoft Paint.

For Google Chrome testing several dozens of different html-files were selected. Aggressiveness (how much of input data is mutated) was alternately set to 5%, 15%, 25%, 35%. Testing results for this case are shown in Table 1. As can be seen from the table, fuzzing testing can help in discovering "File Not Found" type errors. Apparently, higher aggressiveness leads to more errors of this type.

Table 1. Expected value of "File Not Found" error and confidence interval (for significance level $\alpha = 0{,}01$ sample size $N = 1000$)

Error	Aggressiveness Level			
	5%	15%	25%	35%
"File Not Found"	$0{,}239 \pm 8{,}23 \cdot 10^{-4}$	$0{,}269 \pm 2{,}6 \cdot 10^{-4}$	$0{,}295 \pm 3{,}49 \cdot 10^{-4}$	$0{,}320 \pm 8{,}28 \cdot 10^{-4}$

After several hours of fuzzing for Notepad++ no errors occurred even with 100% aggressiveness. This shows high level of stability and security of this application. Testing of Winamp led to similar results.

To test Microsoft Paint images of different formats and sizes were selected. Testing results are shown in Table 2. As it can be seen from the results, selective testing with fuzzing revealed errors "File Not Found" and "Wrong Format". Frequency of these are almost similar and increase with higher aggressiveness.

Table 2. Expected value for errors and confidence interval (for significance level $\alpha = 0{,}01$ sample size $N = 1000$)

Error	Aggressiveness Level			
	5%	15%	25%	35%
"File Not Found"	$0{,}13 \pm 2{,}6 \cdot 10^{-4}$	$0{,}15 \pm 2{,}58 \cdot 10^{-4}$	$0{,}18 \pm 2{,}64 \cdot 10^{-4}$	$0{,}22 \pm 1{,}58 \cdot 10^{-4}$
"Wrong Format"	$0{,}09 \pm 2{,}96 \cdot 10^{-4}$	$0{,}16 \pm 3{,}77 \cdot 10^{-4}$	$0{,}2 \pm 3{,}58 \cdot 10^{-4}$	$0{,}23 \pm 4{,}57 \cdot 10^{-4}$

Conducted tests show that the majority of common applications are secured and cannot be crashed using primitive fuzzing. In the case of Google Chrome, for example, it is not a surprise, because software engineers and testing professionals at Google use much more complex fuzzing during the development cycle of their products [28]. The same goes for Microsoft.

Thus, the analysis of different fuzzers in the area of automated testing shows that this approach to software vulnerability search can vary depending on the goal, tester's skills, data format and other factors. Some applications have privilege separation system, which depends on user level. Using fuzzer as a tool for automated vulnerability search, it is possible to find errors in software products, which let attacker gain full or partial control over the system. Some low-lever errors are very similar to each other so it is possible to use the same logic to find vulnerabilities in more applications.

Relying on conducted experimental research for selective testing it can be said that fuzzing is quite promising method of automated vulnerabilities search. We were able to find errors even in reliable and tested applications, like Google Chrome and Microsoft Paint, as we managed to discover random input data, which would cause errors. However, these are not critical for application functioning and/or operating system, their handling is correct that, apparently, is stipulated behavior for such kind of corrupted input data.

It is worth mentioning that fuzzing has some limitations when it comes to practical use and have not gained wide popularity for automated vulnerability testing yet. However, considering the fact big companies, such as Google and Microsoft are using fuzzing as a part of their methodology and vigorously work on its development, it can be safely said that fuzzing has quite strong potential [29-32].

The most promising direction of future development of automated vulnerability search methods is fuzzing intellectualization [18-22]. This is about using deep learning methods to improve computational procedures for automated vulnerability search. It is believed that such approach can significantly improve the process of selecting input data, which will cause failure or errors in the target application.

3 Intellectual Fuzzing Algorithm

Fuzzing is a method of software and security vulnerabilities testing which is conducted by making multiple tests using mutated input data [21]. Repeated testing is performed with random mutation, and usually testing time is far from optimal. This article considers the problem of intellectual fuzzing and tries to find a solution [22]. The main goal is to develop a technology that can be guided and will make decisions based on the experience it gained during testing. The solution to this question lies in machine learning and reinforcement learning based on the deep Q-learning algorithm [33]. It uses maximum possible rewards, which are defined during development process by analyzing program source data and available rewards. This allows to apply optimal input data mutations. Thus, agent gets an opportunity to learn to formulate an optimal action policy for obtaining maximum reward. In this paper, we propose an algorithm and a computer model of the in-depth training as well as research of automated vulnerabilities testing effectiveness in comparison with the random mutation test. During testing, the "black box" method is used. It means that the available information represents only results of the program's work and the input data that it needs to perform [21].

During research, we realized there is a serious problem with randomized fuzzing: if it works with randomly generated input data, the time will not be optimal, since the process is performed blindfold. There may be a lot of testing rounds resulting in huge amounts of mutations that do not provide any progress. The process of fuzzing is an execution of a task cycle in certain defined program, where input is a sequence that was changed by some mutation. The ideal solution to solve such problem is machine-learning technology called reinforcement learning. The best example of using this algorithm is the AlphaGO developed by Google DeepMind in 2015. It becomes the world's first program to win the game of "Go" with a top-ranked professional Lee Sedol [34].

As a combination of fuzzing and reinforcement training, a system capable of changing the rules for selecting specific mutation was created. It sends mutated data to input and, depending on the program's source data, generates a reward in order to rely on its own experience and select optimal mutations for particular case upon further testing. Thus, the amount of mutations does not contribute to the testing process significantly reduces. This makes testing process faster. Schematic diagram of the developed system is shown in Fig. 1. Testing process begins by defining the original non-mutated input data. Its format depends on fuzzing kind. Packages are submitted to the program's input channel and the response of the program is determined using special debugging software. From obtained data (this may be the program execution time, code coverage, code completion, etc.) system's State is formed. It will be preprocessed and presented to the Deep Q-learning model's input. The system then decides what Action should be taken next.

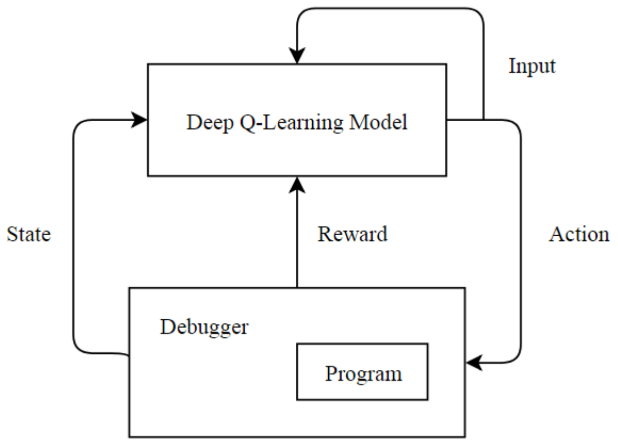


Fig. 1. Simplified diagram of the intellectual fuzzing algorithm

At the same time, depending on the selected action and the obtained program state, system forms the Reward for the algorithm. Using this reward, algorithm understands assigned task and determines optimal behavior for its implementation. In addition, algorithm remembers which actions brought it to the maximum reward (finding a mistake or a program failure, etc.) and, in the following testing rounds, decides what action should be taken based on its already gained experience. To calculate the next step, the previous inputs are mutated according to the action that was selected. Program input receives new mutated data. This procedure is repeated until the algorithm reaches its goal. Developed model uses Markov decision-making process – deep Q-learning [35].

4 Reinforcement Learning

This section provides necessary data about the algorithm of reinforcement training. Reinforcement learning is a computational approach to understanding and automating targeted learning and decision making. It stands out among other machine learning algorithms by the fact that agent learns directly by interacting with the environment without referring to examples [35]. This algorithm is primarily aimed to solve problems that arise during interaction with the environment to achieve long-term action. It uses formal structure of Markov decision-making process [35], defining the interaction between agent and environment in terms of states, actions and rewards. These features include understanding of causes and effects, as well as presence of clear goals. The notion of value and function of value is the main features of reinforcement training methods.

As noted earlier, the interaction between agent and environment can be described using Markov decision-making process $M = (S, A, P)$, where S – a set of system's states, A – action's set, P – transition probabilities set. For each state-action pair $(s,a) \in S \times A$, P is a set of probabilities $P(s' \vee s, a)$, where s′ corresponds to the next system's state. Agent considers possible system states for the selected action, where each transition has its own reward $r(s,a)$, and studies the optimal behavior for maximizing the reward.

During the learning process, the main goal is to maximize the finite amount of rewards:

$$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1} ,$$

where $\gamma \in (0,1)$ – discount rate, which determines the remuneration priority over time. Action a_t at state s_t is determined by the policy of action $a_{t\pi}(\vee s_t)$. Policy π attaches considered possible states to action, which in turn determines agent's behavior. Expected cumulative reward for the policy-maker π is defined as:

$$Q^{\pi}\left(s,a\right) = E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \vee s_0 = s, a_0 = a\right].$$

The problem of finding the optimal value $Q_{\pi}(s,a)$ can be reduced to the procedure of function approximation. To achieve this $Q_{\pi}(s,a)$ needs to be updated after each iteration of receiving the award [36]. It is defined as

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha ,$$

where α – learning rate. The entire procedure can be stated in the following sequence: agent gets the status s_t, takes action

$$a_t = \arg\max(Q(s_t, a)) ,$$

which defines the reward r_t, and causes the system to go to the state $s_t + 1$.

After receiving a reward r_t and state $s_t + 1$, agent determines the best possible effect

$$a_t + 1 = \arg\max(Q(s,a)).$$

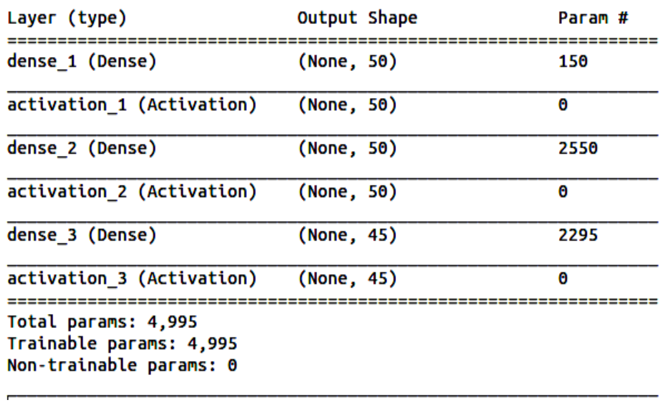
Then it updates the value $Q(s_t, a_t)$. To approximate the function $Q(s_t, a_t)$, deep neural networks are used (it defines the name of deep Q-learning), which in turn aim to minimize the loss function:

$$L = (r + \gamma \cdot \max(Q(s_t + 1, a_t)) - Q(s_t, a_t)^2.$$

5 Simulation Results

To conduct an experiment simple fuzzer was used. Its main function is to implement input data selection for automated software vulnerability search. Software launch process was simulated and special logic tests were developed. The program could return an error code when certain data is received to compare fuzzing with and without artificial intelligence. Possible states of the system are presented in the form of data generated after the completion of the program. This data is transmitted by neural network, which consists of one input layer, two hidden layers with 50 neurons each and activation functions (Rectified Linear Unit): $f(x) = \max(0, x)$.

The output of the neural network has 45 elements, representing the number of possible mutations. Complete scheme of neural network for function approximation $Q(s_t, a_t)$ is shown in Fig. 2.



```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 50)                150
_________________________________________________________________
activation_1 (Activation)    (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2550
_________________________________________________________________
activation_2 (Activation)    (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 45)                2295
_________________________________________________________________
activation_3 (Activation)    (None, 45)                0
=================================================================
Total params: 4,995
Trainable params: 4,995
Non-trainable params: 0
_________________________________________________________________
```

Fig. 2. Neural network scheme Network training is performed using Adam optimization algorithm [36]

Let $f(\theta)$ be noisy target function: a stochastic scalar function is differentiated relatively to the parameter θ. The expected cost of this feature, $E\left[f(\theta)\right]$ relatives to θ must be minimized. Using $f_1(\theta),\ldots,f_T(\theta)$ the implementation of a stochastic function in the next steps $1,\ldots,T$ is denoted. Stochasticity can be based on the estimation of random subsets (mini-groups) of data points or noise of a function. At $g_t = \nabla\theta f_t(\theta)$ the gradient is denoted – the vector of partial derivatives f_t relatively to θ, which is estimated by time t.

The algorithm updates exponential moving average values of the gradient (m_t) and the square of the gradient (v_t), where hyper parameters $\beta_1, \beta_2 \in [0,1]$ control the exponential velocity of decomposition for these moving averages. The most moving averages are the estimations of the first moment (mean value) and the second moment (uncentered dispersion) of the gradient. However, these moving averages are initialized as zero vectors, which lead to the estimation of moments moving in zero direction, especially in the initial time steps and when expansion rates are small (for example, the value β_s close to 1). The good news is this bias initialization can be easily prevented by getting bug fixes m_{bt} and v_{bt}.

The algorithm itself has the following form:

Required input data:

- α : Learning speed
- $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for moment estimates (standard settings $\beta_1 = 0.9, \beta_2 = 0.999$)
- $f(\theta)$: Stochastic function with parameter θ
- θ_0 : Vector of initial parameters
- Algorithm:
- $m_0 \leftarrow 0$ (Initialize the first moment vector);
- $v_0 \leftarrow 0$ (Initialize the second moment vector);
- $t \leftarrow 0$ (Initialize the time);
- while θ_t not converged:
    - $t \leftarrow t+1$;
    - $g_0 \leftarrow \nabla\theta f_t(\theta_t - 1)$ (Take the gradient relative to the stochastic function during t);
    - $m_t \leftarrow \beta_1 \square m_t - 1 + (1-\beta_1)\square g_t$ (Update rejected first moment);
    - $v_t \leftarrow \beta_2 \square v_t - 1 + (1-\beta_2)\square g_{2t}$ (Update the second estimation for a rejection);
    - $m_b t \leftarrow \frac{m_t}{1-\beta_{t_1}}$ (Calculate the corrected bias estimation of the first moment);

    - o $yb_t \leftarrow \frac{y_t}{1-\beta_{t_2}}$ (Calculate the corrected bias estimation of the second moment);
    - o $\theta_t \leftarrow \theta_t - 1 - \alpha\square\frac{mb_t}{\sqrt{vb_t}}$ (Update parameters);

  end while

- return θ_t (Calculated parameters).

Input mutations were selected based on the standard list: increasing and decreasing line length, integer insertion, adding special characters (for example, "%s", which may also cause errors). In sum, 45 functions were created and placed in a dictionary for further use.

The system receives an award if execution time was greater than the previous or if there were errors occurred during testing. When an error occurs, the algorithm finishes its work. While forming this type of award, there was a problem when algorithm already found one error and started calling it repeatedly to get maximum reward. To avoid this problem, two constants must be set: $\gamma \in (0,1)$ – discount rate and $\varepsilon \in (0,1)$ – intelligence speed. The first constant was already being mentioned before. The second one determines how the algorithm is capable in terms of discovering new solutions. Random action will be chosen with probability ε, and the most profitable action will be selected with probability 1 – ε. The hypothesis is a scientific assumption that made to explain any phenomenon and requires testing on theoretical basis in order to become a reliable scientific theory [37]. Statistical hypothesis is any assertion (assumption) concerning the type or distribution parameters of a certain feature of the objects being studied [37].

The following sequence of actions was necessary to test the hypotheses:

1. Make calculations of certain statistics, the distribution of which is known.
2. Find the P-value for the calculated results.
3. Make appropriate conclusions depending on the significance criterion and P-value.

A special test for identifying an error was developed. The hypothesis of the experiment is Q-learning fuzzing works faster than randomized one. The discount rate is set at 0.9, and the exploration speed is equal to 0.5. The latter parameter decreases 0.99 times after each era. The Student's t-test was used [38] to test the hypothesis.

Student's t-test – the general name for the class of methods for statistical criteria testing. It is based on comparison with the Student's distribution. The most common application of the criteria is related to checking the equality of mean values in two samples [38]. To use this criterion, some conditions must be met: the initial data must have normal distribution and dispersion must be equal.

The first group contains the results of testing using developed algorithm, while the second one presents the results of random mutations. Testing was performed in the following sequence: generate 15 experiments, and record the amount of mutations necessary to find an error at the end of each. The results of the experiments are shown in Table 3.

The results of Student's t-test calculation: $t = -12.40$. The number of degrees of freedom:

$$v = 2n - 2 = 2 \times 15 - 2 = 28$$

Considering significance criteria

$$\alpha = 0.01 \text{ and } P - value = 2.763,$$

while $t < P - value$, the hypothesis is valid.

The result is statistically significant for a given criterion, if the probability of accidental occurrence of the same or extreme result is less than the given level (0.01) under the condition of loyalty of the null hypothesis.

Table 3. Experimental Results

№	Deep Q-learning model	Random selection of mutations
1	3671	3686
2	1191	1897
3	1879	3164
4	1640	3233
5	1966	10446
6	1585	5358
7	1135	1134
8	4877	752
9	2465	2157
10	3266	3684
11	1895	2026
12	2093	2993
13	1150	295
14	1181	3381
15	1153	358

The testing time of developed algorithm is better than the time of random mutations testing considering the fact the algorithm did not learn before. On average, developed algorithm finds an error after 2076 mutations, whereas random testing needs 2832 mutations. Represented algorithm finds error 30% faster.

This result proves the direction of research was chosen right. Particularly, it shows that the main problem of fuzzing (large amount of mutations) can be solved using artificial intelligence methods. Actually, if input data is changed directionally depending on previous results, it can speed up mutation process and receive results (which are finding the vulnerability in certain software product) after less amount of mutations.

6 Conclusions

Due to conducted research, one of promising automated software testing methods in critically important systems was analyzed. Fuzzing bases on multiple input of different (mutated) data to find parameters, which will cause failure or incorrect functioning of software. Repeated testing is usually carried using randomized mutations and the time of testing is very high in the most cases. This article researches the problem of intellectual fuzzing – the technology, which uses previous testing experience to make choices, related to mutation, and reduce testing time.

Deep Reinforcement Learning algorithm was used to implement intellectual fuzzing. With the use of simple fuzzing app, it becomes possible to prove that testing time decreases by 30%. This result was received because fuzzer used previous experience to adjust mutations.

This research may continue in other spheres. For example, Intrusion Detection and Prevention Systems [39-42] are also can be built using some elements of artificial intelligence. Critically important information systems in different spheres, including banking, industrial facilities management and Smart Grids are especially interesting for further research [43-50].

References

1. Bitner, M.J., Zeithaml, V.A., Gremler, D.D., Maglio, P.P., Kieliszewski, C.A., Spohrer, J.C.: Technology's Impact on the Gaps Model of Service Quality" in Handbook of Service Science - Service Science: Research and Innovation in the Service Economy, US, Boston, MA: Springer (2010)
2. Brocke, J., Thurnher, B., Winkler, D.: Improving IT-Service Business Processes by Integrating Mobility: The Role of User Participation-Results from Five Case Studies. In: International Conference on eLearning e-Business Learning e-Business Enterprise Information Systems and e-Government (EEE 2008) (2008)
3. Budzik, J., Hammond, K.: Watson: Anticipating and Contextualizing Information Needs. In: Proc. of the 62 Annual Meeting of the American Society for Inform. Science, 1999, vol. 36, pp. 727–740 (1999)
4. World Manufacturing Production 2012. United Nations Industrial Development Organization, 2013
5. Structural Change in the World Economy. United Nations Industrial Development Organization, 2009
6. Bush, S.F., Goel, S., Simard, G.: IEEE Vision for Smart Grid Communications: 2030 and Beyond. In: IEEE Vision for Smart Grid Communications: 2030 and Beyond, 2014, pp. 1492–1499. (2014) doi:10.1109/IEEESTD.2013.6690098
7. Albarakati, A., Moussa, B., Debbabi, M., Youssef, A., Agba, B.L., Kassouf, M.: OpenStack-Based Evaluation Framework for Smart Grid Cyber Security. In: 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart

Grids (SmartGridComm), Aalborg, 2018, pp. 1–6. (2018) doi:10.1109/SmartGridComm.2018.8587420

8. Hariri, M.E., Youssef, T., Habib, H.F., Mohammed, O.: A Network-in-the-Loop Framework to Analyze Cyber and Physical Information Flow in Smart Grids. In: 2018 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia), Singapore, 2018, pp. 646–651 (2018)
9. Sokolov, S.S., Glebov, N.B., Antonova, E.N., Nyrkov, A.P.: The Safety Assessment of Critical Infrastructure Control System. In: 2018 IEEE International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT&QM&IS), St. Petersburg, 2018, pp. 154–157. (2018) doi:10.1109/itmqis.2018.8524948
10. Krasnobayev, V., Kuznetsov, A., Koshman, S., Moroz, S.: Improved Method of Determining the Alternative Set of Numbers in Residue Number System. In: Recent Developments in Data Science and Intelligent Analysis of Information. ICDSIAI 2018. Advances in Intelligent Systems and Computing, Springer, Cham, 2019, vol. 836, pp. 319–328. (2019) doi:10.1007/978-3-319-97885-7_31
11. Cybersecurity for Smart Grid Systems (25 June 2018). https://www.nist.gov/programs-projects/cybersecurity-smart-grid-systems
12. Andrushkevych, A., Gorbenko, Y., Kuznetsov, O., Oliynykov, R., Rodinko, M.: A Prospective Lightweight Block Cipher for Green IT Engineering. In: Green IT Engineering: Social, Business and Industrial Applications. Studies in Systems, Decision and Control, Springer, Cham, 2019, vol, 171, pp. 95–112. (2019) doi:10.1007/978-3-030-00253-4_5
13. Ferguson, N., Schneier, B.: Practical Cryptography. John Wiley & Sons (2003)
14. Kuznetsov, O., Potii, O., Perepelitsyn, A., Ivanenko, D., Poluyanenko, N.: Lightweight Stream Ciphers for Green IT Engineering. In: Green IT Engineering: Social, Business and Industrial Applications. Studies in Systems, Decision and Control, Springer, Cham, 2019, vol. 171, pp. 113–137. (2019) doi: 10.1007/978-3-030-00253-4_6
15. Yinghua, H., Yanchun, M., Dongfang, Z.: The Multi-join Query Optimization for Smart Grid Data. In: 2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA), Nanchang, 2015, pp. 1004–1007. (2015) doi:10.1109/ICICTA.2015.255
16. Du, Z.: Intelligence Computation and Evolutionary Computation. Springer-Verlag Berlin Heidelberg (2013)
17. Tong, C., Gao, Y., Tong, M., Li, J., Wang, Q., Chen, K.: Application of lightning detection in Smart Grid dynamic protection. In: 2012 International Conference on Lightning Protection (ICLP), Vienna, 2012, pp. 1–13. (2012) doi:10.1109/ICLP.2012.6344209
18. Choi, S.Y., Lee, H.Y.: Toward Automated Scanning for Code Injection Vulnerabilities in HTML5-Based Mobile Apps. In: 2016 International Conference on Software Security and Assurance (ICSSA), St. Polten, 2016, pp. 24–24. (2016) doi:10.1109/ICSSA.2016.11
19. Peng, H., Shoshitaishvili, Y., Payer, M.: T-Fuzz: Fuzzing by Program Transformation. In: 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, 2018, pp. 697–710. (2018) doi:10.1109/SP.2018.00056
20. Zhao, J., Pang, L.: Automated Fuzz Generators for High-Coverage Tests Based on Program Branch Predications. In: 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), Guangzhou, 2018, pp. 514–520. (2018) doi:10.1109/DSC.2018.00082
21. Sutton, M., Greene, A., Amini, P.: Fuzzing: Brute Force Vulnerability Discovery. 1st ed. Boston, MA, USA: Addison-Wesley Professional (2007) http://bxi.es/Reversing-Exploiting/Fuzzing%20Brute%20Force%20Vulnerability%20Discovery.pdf

22. Böttinger, K., Godefroid, P., Singh, R.: Deep Reinforcement Fuzzing. Submitted on 14 Jan 2018. (2018) https://arxiv.org/abs/1801.04589
23. Miller, B.P., Fredriksen, L., Barton, B.S., Miller, P.: An Empirical Study of the Reliability of UNIX Utilities (1989) ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf
24. Miller, B.P.: Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services (1995) ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz-revisited.pdf
25. Forrester, J.E., Miller, B.P.: An Empirical Study of the Robustness of Windows NT Applications Using Random Testing. Computer Sciences Department University of Wisconsin Madison, WI 53706–1685 (2000) ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz-nt.pdf
26. Miller, B.P., Cooksey, G., Moore, F.: An Empirical Study of the Robustness of MacOS Applications Using Random Testing. Computer Sciences Department University of Wisconsin Madison, WI 53706–1685 (2006) ftp://ftp.cs.wisc.edu/paradyn/technical_papers/Fuzz-MacOS.pdf
27. René, F.: The Art of Fuzzing (2017) https://www.sec-consult.com/wp-content/uploads/files/vulnlab/the_art_of_fuzzing_slides.pdf
28. Guided in-process fuzzing of Chrome components. Google Security Blog. https://security.googleblog.com/2016/08/guided-in-process-fuzzing-of-chrome.html
29. American fuzzy lop. http://lcamtuf.coredump.cx/afl/
30. Böck, H.: How Heartbleed could've been found (2015) https://blog.hboeck.de/archives/868-How-Heartbleed-couldve-been-found.html
31. Dallman, J., Pulikkathara, G.: MiniFuzz File Fuzzer Overview (2011) https://msdn.microsoft.com/en-us/biztalk/gg675011
32. Peach Fuzzer Community Edition. http://www.peach.tech/resources/peachcommunity/
33. Li, Y.: Deep Reinforcement Learning: An Overview. Submitted on 25 Jan 2017 (v1), last revised 26 Nov 2018 (this version, v6) (2018) https://arxiv.org/abs/1701.07274
34. AlphaGo Games. https://deepmind.com/research/alphago/match-archive/alphago-games-english/
35. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press Cambridg (1998) http://incompleteideas.net/book/bookdraft2017nov5.pdf
36. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization". Submitted on 22 Dec 2014 (v1), last revised 30 Jan 2017 (this version, v9) (2017) https://arxiv.org/abs/1412.6980
37. Statistical hypotheses and their verification (in Ukrainian). http://stat.org.ua/statclasses/hypotheses-testing/
38. t-criterion for Student (in Ukrainian). http://fpo.bsmu.edu.ua/static/t-kryteriy-styudenta
39. Muller, S.: Risk Monitoring and Intrusion Detection for Industrial Control Systems. Docteur de l'Université du Luxembourg en Informatique, University of Luxembourg, Luxembourg (2018)
40. Kuznetsov, A.A., Smirnov, A.A., Danilenko, D.A., Berezovsky, A.: The statistical analysis of a network traffic for the intrusion detection and prevention systems. Telecommunications and Radio Engineering, 2015, vol. 74(1), pp. 61–78. (2015) doi: 10.1615/TelecomRadEng.v74.i1.60
41. Jahan, S., Habiba, R.: An analysis of smart grid communication infrastructure & cyber security in smart grid. In: 2015 International Conference on Advances in Electrical Engineering (ICAEE), Dhaka, 2015, pp. 190–193. (2015) doi:10.1109/ICAEE.2015.7506828
42. Wang, Y., Zhang, B., Lin, W., Zhang, T.: Smart grid information security - a research on standards. In: 2011 International Conference on Advanced Power System Automation and Protection, Beijing, 2011, pp. 1188–1194. (2011) doi:10.1109/APAP.2011.6180558

43. Rassomakhin, S., Kuznetsov, A., Shlokin, V., Belozertsev, I., Serhiienko, R.: Mathematical Model for the Probabilistic Minutia Distribution in Biometric Fingerprint Images. In: 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP), Lviv, Ukraine, 2018, pp. 514–518. (2018) doi:10.1109/DSMP.2018.8478496
44. Borkar, A., Donode, A., Kumari, A.: A survey on Intrusion Detection System (IDS) and Internal Intrusion Detection and protection system (IIDPS). In: 2017 International Conference on Inventive Computing and Informatics (ICICI), Coimbatore, 2017, pp. 949–953. (2017) doi:10.1109/ICICI.2017.8365277
45. Moskovchenko, I., Pastukhov, M., Kuznetsov, A., Kuznetsova, T., Prokopenko, V., Kropyvnytskyi, V.: Heuristic Methods of Hill Climbing of Cryptographic Boolean Functions. In: 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 2018, pp. 1–6. (2018) doi:10.1109/INFOCOMMST.2018.8632017
46. Lei, H., Chen, B., Butler-Purry, K.L., Singh, C.: Security and Reliability Perspectives in Cyber-Physical Smart Grids. In: 2018 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia), Singapore, 2018, pp. 42–47. (2018) doi:10.1109/ISGT-Asia.2018.8467794
47. Kuznetsov, A., Kavun, S., Panchenko, V., Prokopovych-Tkachenko, D., Kurinniy, F., Shoiko, V.: Periodic Properties of Cryptographically Strong Pseudorandom Sequences. In: 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 2018, pp. 129–134. (2018) doi:10.1109/INFOCOMMST.2018.8632021
48. Kavun, S.: Conceptual fundamentals of a theory of mathematical interpretation. Int. J. Computing Science and Mathematics, 2015, vol. 6(2), pp. 107–121. (2015) doi:10.1504/IJCSM.2015.069459
49. Kavun, S.: Indicative-geometric method for estimation of any business entity. Int. J. Data Analysis Techniques and Strategies, 2016, vol. 8(2), pp. 87–107. (2016) doi: 10.1504/IJDATS.2016.077486
50. Kovtun, V., Kavun, S., Zyma, O.: Co-Z Divisor Addition Formulae in Homogeneous Representation in Jacobian of Genus 2 Hyperelliptic Curves over Binary Fields. International Journal Biomedical Soft Computing and Human Sciences, 2012, vol. 17(2), pp. 37–43. (2012) doi:10.24466/ijbschs.17.2_37