Cilt 11, Sayı 4, Sayfa 117-125, 2022 Volume 11, Issue 4, Page 117-125, 2022

Araştırma Makalesi https://doi.org/10.46810/tdfd.1201248 Research Article

Türk Doğa ve Fen Dergisi
Turkish Journal of Nature and Science
www.dergipark.gov.tr/tdfd

Optimization of Software Vulnerability with the Meta-Heuristic Algorithms

Canan BATUR ŞAHİN[1*]

[1] Malatya Turgut Özal University, Faculty of Engineering and Natural Sciences, Software Engineering Department, Malatya, Türkiye
Canan BATUR ŞAHİN ORCID No: 0000-0002-2131-6368

**Corresponding author: canan.batur@ozal.edu.tr*

(Received: 08.11.2022, Accepted: 13.12.2022, Online Publication: 28.12.2022)

Keywords
Meta-heuristic algorithm, Optimization, Clock-work mechanism, Software vulnerability

Abstract: In order to ensure the development of secure software, it is essential to predict software vulnerabilities. Nevertheless, there can be considerable losses in case of an attack on an information system. Detecting a dangerous code, which can lead to severe unknown consequences, requires great effort. There is a strong need to devise meta-heuristic-based approaches to provide effective security and prevent vulnerabilities or mitigate them. The primary focus of studies on software vulnerability prediction models is to specify the best set of predictors that are related to the presence of vulnerabilities. However, the existing vulnerability detection methods suffer from coarse detection granularity and a bias toward global features or local features. The framework proposed in the present work improves optimization algorithms for the best set of optimized vulnerability patterns correlated for software vulnerabilities based on a clock-work memory mechanism. Using the proposed framework, we found vulnerable optimized patterns based on clock-work memory mechanism feature representation learning that directly. The effectiveness of the developed algorithm was further improved with the clock-work memory mechanism based on 6 open-source projects, such as LibTIFF, Pidgin, FFmpeg, LibPNG, Asteriks, and VLC media player datasets.

Meta-Sezgisel Algoritmalar ile Yazılım Güvenlik Açıklarının Optimize Edilmesi

Anahtar Kelimeler
Meta-sezgisel algoritmalar, Optimizasyon , Saat-hafıza mekanizması, Yazılım güvenlik açığı

Öz: Yazılım güvenlik açığının tahmini, güvenli yazılım geliştirmek için önemli bir husustur. Ancak, bir bilgi sistemine saldırı yapıldığında büyük kayıplara neden olabilir. Tehlikeli kodun tespiti büyük çaba gerektirir ve bu da bilinmeyen ciddi sonuçlara yol açabilir. Etkili güvenlik sağlamak ve güvenlik açıklarının oluşmasını önlemek veya güvenlik açıklarını azaltmak için meta-sezgisel tabanlı yaklaşımlar geliştirmeye güçlü bir ihtiyaç vardır. Yazılım güvenlik açığı tahmin modelleri üzerine yapılan araştırmalar, temel olarak, güvenlik açıklarının varlığı ile ilişkili en iyi tahmin ediciler kümesini belirlemeye odaklanmıştır. Buna rağmen, mevcut güvenlik açığı algılama yöntemleri, genel özelliklere veya yerel özelliklere yönelik önyargı ve kaba algılama ayrıntı düzeyine sahiptir. Bu yazıda, önerilen çerçeve, bir saat-çalışma belleği mekanizmasına dayalı yazılım güvenlik açıkları ile ilişkili en iyi optimize edilmiş güvenlik açığı kalıpları kümesi için optimizasyon algoritmalarını geliştirmektedir. Geliştirilen algoritmanın etkinliği, LibTIFF, Pidgin, FFmpeg, LibPNG, Asteriks ve VLC medya oynatıcı veri kümeleri gibi 6 açık kaynak projesine dayanan saatli çalışan bellek mekanizması ile daha da artırılmıştır.

1. INTRODUCTION

Exploitable vulnerabilities in software considerably weaken computer systems' security and pose a threat to the information technology infrastructure of numerous government organizations and sectors. Software vulnerabilities represent exploitable weak points in a source code with the objective of causing harm or loss. Software vulnerabilities are also the root cause of cyberattacks. Detecting vulnerabilities means revealing code snippets that induce errors in particular cases in large code chunks. It is still difficult and requires a lot of time to detect vulnerabilities to date.

A number of techniques are available for detecting software vulnerabilities. It is possible to identify them at design time (without executing the source code) or at run time (while executing the software). Static code analysis (SCA) takes place among the most common design-time techniques and involves code analysis without executing the program for the purpose of identifying possible problems (alerts). It is possible that a part of the above-mentioned alerts are software vulnerabilities. The said process is realized using static analysis tools (SATs), which are either open-source or commercial.

The major branch of detection approaches is the discovery of possible vulnerabilities in the source code. Nevertheless, they have weaknesses, such as high false-positive rates and low efficiency. Whereas the vulnerability detection method that employs machine learning technology has advanced considerably in accuracy and automation, the problems specified below create obstacles to its performance: (1) Long-term dependency between code elements. There is valuable information for detecting vulnerabilities in the dependencies between elements. Nevertheless, elements that are related in semantic terms can be located far from each other. Hence, we suggest an automated software vulnerability framework based on clock-work memory mechanism recurrent neural networks for a representation method. We believe that deep learning algorithms have the capability to capture complex vulnerability patterns.

The need for optimization techniques with higher reliability, particularly meta-heuristic optimization algorithms, has recently arisen because of the constantly increasing complex nature and difficulty of real-world problems. The said techniques are mainly stochastic and perform the estimation of optimal solutions for various optimization problems. Reasoning about processes at multiple time scales is facilitated by Clock-Work RNN (CW-RNN) models. The hidden layer in a CW-RNN is separated into various modules. Each of these modules processes inputs at its temporal granularity, making calculations solely at the prescribed clock rate. Forward connections are present in the CW-RNN, from the input to the hidden layer and from the hidden to the output layer. CWRNN models primarily contribute to discussing long-term dependencies. The architecture of the CW-RNN is similar to that of a simple RNN with an input, output, and hidden layer. There are g modules in the hidden layer, and each of these modules has its clock rate. The neurons within each module are completely interconnected, meaning that the connectivity among neurons of various modules is set on the basis of the modules' clock periods.

The current work makes the following main contributions:

1. The study creates the metaheuristic algorithm-based vulnerability detection system with metaheuristic optimization algorithms,
2. A framework is proposed, improving the detection capability of heuristic approaches based on a clock-work memory for learning optimized patterns to extract the optimized features for detecting software vulnerable codes.
3. Our framework's design is validated by conducting experiments, and the usage of clock-work memory is shown as optimized-feature representations.

The rest of the paper is organized as follows: Section 2 describes the Material and Method Section 3 describes the the proposed Model used in the study. Section 4 describes the Results and Discussion. Section 5 describes the conclusion.

2. MATERIAL AND METHOD

The current part contains the background of the most frequently employed techniques in the literature. .

2.1. Meta-Heuristic Algorithms

In this part, the bio-inspired metaheuristic algorithms used are given as follows.

2.1.1. Whale optimization algorithm (WOA)

The Whale Optimization Algorithm (WOA) has been newly developed, and its basis is whales' hunting behavior. The mentioned algorithm includes the following three stages: circling hunting, bubble-net attacking, and prey hunting.

In circling hunting, whales first circle the prey and thus set the trap. Afterward, a search agent is selected according to the distance of an individual whale from the prey. After identifying the search agent, the positions of all whales in the group are updated according to the search agent's position, which can be expressed in mathematical terms, as shown below:

$$\vec{D} = [\vec{C}.\ \vec{X} * (t) - \vec{X}(t)] \quad (1)$$

Where $C = 2 * r$, r denotes a random number in the range of 0-1; $\vec{X}$ refers to the local optimal position; $\vec{X}(t)$ current denotes the current position; refers to the iteration number; and represents the distance between every whale and the search agent.

Afterward, whales perform bubble-net attacking by utilizing the spiral around and spiral update methods [15]. They move in the prey's direction spirally according to the search agent. It is possible to determine the updated position of other search agents moving toward the best agent by Equations 2-3 :

$$\vec{X}(t+1) = \vec{X} * (t) - \vec{A}.\vec{D} \quad (2)$$

$$\vec{A} = 2.\vec{a}\ \vec{r} - \vec{a} \quad (3)$$

Eqs. (4) and (5) are used to find the search agent's random position:

$$\vec{D} = [\vec{C}.\ \vec{X}_{rand} - \vec{X}] \quad (4)$$

$$\vec{X}(t+1) = \vec{X}_{rand} - \vec{A}.\vec{D} \quad (5)$$

The shrink position in the movement with a helix shape toward the prey is updated by the whale, as shown in Equation 6:

$$\vec{X}\,(t+1) = \vec{D}\,.\,e^{bL}\ .\cos(2\,\pi\,L)\ + \vec{X}\ *\ t \qquad (6)$$

Where $\vec{X}$(t+1) Updated denotes the whales' updated position; b refers to a constant representing the logarithmic spiral's shape; l represents the distance between the whale and the food. L = −1 denotes the minimum distance to the food, and L = +1 refers to the maximum distance to the food. It was assumed that the possibility of selecting a method for a certain case was 50%, and Equation 7 expresses the chance of choosing the path:

$$\vec{X}\,(t+1) = \begin{cases} \vec{X}*\,(t)\ -\ \vec{A}\,.\vec{D} & p < 0.5 \\ \vec{D}\,.\,e^{bL}\ .\cos(2\,\pi\,L)\ + \vec{X}\ *\ t & p > 0.5 \end{cases} \qquad (7)$$

Here, p refers to a number chosen in a random way between 0 and 1.

2.1.2. Multi-verse optimizer (MVO) algorithm

The Multi-Verse Optimizer (MVO) takes place among the new swarm intelligence algorithms. Its source of inspiration is the multiverse theory discussing how the big bangs generate multiple universes and the interaction of the said universes with each other via various hole types. In the MVO algorithm, the "white hole" and "black hole" concepts with the objective of exploring the wormholes to utilize the search spaces for formulating a population-based algorithm and considered that every solution was a universe and every variable/attribute in the solution denoted an object in the said universe. Furthermore, there is a fitness value (inflation rate) in every solution, reflecting the solution quality, which is computed by the corresponding objective function.

A solution receives a good objective value in case white holes appear, whereas the solution receives a worse objective value in case black holes appear. With a higher number of interactions between white holes and black holes, the movement of the variable values of the good solutions to poor solutions occurs.

2.1.3. Grey Wolf Optimizer (GWO)

The Grey Wolf Optimizer (GWO) represents a meta-heuristic optimization algorithm. Grey wolves' hunting strategy and leadership hierarchy are mimicked in the GWO. The leadership hierarchy comprises four wolf types, including alpha (the fittest solution), beta (the second-best solution), delta (the third-best solution), and omega (the remaining part of the candidate solutions). In practice, the prey is encircled by grey wolves, who march during the hunt, which is expressed with the equations below:

$$\vec{D} = |\vec{C}.\ \vec{X}_p\ (t) - \vec{X}\,(t)| \qquad (8)$$

$$\vec{X}\,|(t+1) = \vec{X}_p\ (t)\ - \vec{A}.\,\vec{D} \qquad (9)$$

Here, t represents the current iteration, D displays the movement vector, $\vec{X}_p$ denotes the prey's position vector, A and C refer to the coefficient vectors, and $\vec{X}$ displays a grey wolf's position vector. The calculation of the coefficient vectors (A and C) is performed by means of the equations below:

$$\vec{A} = 2.\vec{a}\ \vec{r}_1\ - \vec{a} \qquad (10)$$

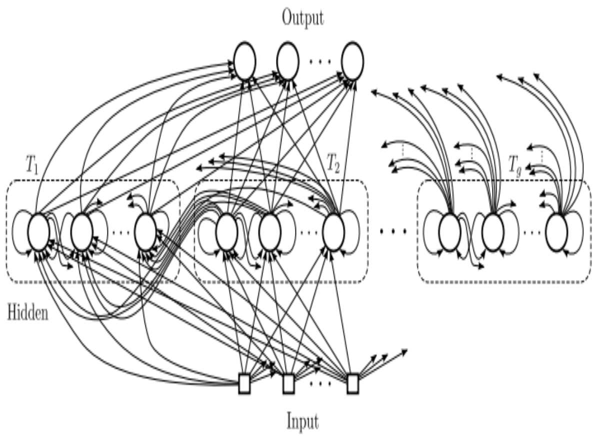
$$\vec{C} = 2.\ \vec{r}_2 \qquad (11)$$

where r1 and r2 are selected in a random manner in the normal range from zero to unity. During iterations, the components of a decrease in a linear way from 2 to 0. By utilizing Equations (10-11), a grey wolf is capable of getting closer to the prey by altering its position around the prey in a random manner.

3. THE PROPOSED METHOD

3.1. Methodology

The objective of the current work is to enhance the effectiveness of meta-heuristic algorithms with the clock-work memory mechanism for predicting software vulnerabilities. The optimized software patterns that were the most appropriate for vulnerability prediction in software systems were obtained. Reasoning about processes at multiple time scales is facilitated by Clock-Work RNN (CW-RNN) models, making calculations solely at the prescribed clock rate. Neurons of various modules are connected on the basis of the modules' clock periods [14].



In the CW-RNN, the speed of the clocks is the same all the time, but sometimes they run at a slower speed and sometimes at a faster one. At each CW-RNN time step *t*, just the outputs of module *i*, satisfying(t MOD T_i) = 0, are active. It is arbitrary to choose the set of periods $\{T_1, \ldots, T_g\}$. In the present work, the exponential series of periods is utilized; the ith module has a clock period of $T_i = 2_i{-}1$. In the proposed framework, each metaheuristic algorithm's metadynamics uses the clock-work memory mechanism as a logging function for the optimized best candidate patterns. For each heuristic algorithm, the information is aggregated from generations using a clock-work memory logged mechanism based on time scales.

Algorithm 1. Pseudo-code of the proposed Clock-Work Memory Mechanism

Input : *Set of vectors of vulnerable code :* $X= [X_1, X_2, . . . , X_N]$

Output : *Set of optimized best patterns:* $S_{best}=\{S_1, S_2, ..., S_N\}$;

BEGIN

Step 1: *{Initialize Metaheuristic Algorithms' parameters}*

Step 2: *[1,2,...N] Initialize the solutions' positions randomly.*

Step 3: *Calculate the fitness of each search agent*

Step 4: For *each iteration,* ***do****:*

Step 4.1: *[Train Clock-Work Network]*

Step 4.1.1: For *each search agent* ***do****:*

Step 4.1.2: *update the position of each current search agent*

Step 4.1.3: *Hidden dimensions are updated in groups at time period clock rates.*

Step 4.1.4: *create the clock-work memory based on time scales* $\{T_1, . . . ,T_g\}$ *for each optimized search agents (candidate solutions)*

Step 4.1.5:*Calculate the fitness of each search agents*

Step 4.1.6:END For

Step 4.1.7: *[END Train Clock-Work Network]*

Step 5: END For

Step 6: *Add List optimized best search agents stored in clock-work memory*

Step 7: END For

Step 8:END

CW-RNN separates the hidden recurrent units into 10 g modules, each runs their own computation at specific, hidden layer units as 32, 64 and 128 rates.The explanation of the general experimental methodology is presented in Algorithm 1, designed based on the each baseline metaheuristic algorithms.

Binary encoding is employed for the purpose of representing feature selection or exclusion in the solution set. Every candidate solution is expressed as a bit string having a length *n*, where n refers to the total feature number. Feature *j* was retained in case of the j^{th} bit being equal to 1, whereas it was removed in case of the jthbit being equal to 0.

The fitness function is employed with the objective of showing the quality of each candidate optimized pattern. The fitness of a candidate solution of each nature-inspired algorithm is proportional to the classification error rate of the model.

4. DISCUSSION AND CONCLUSION

The data source includes vulnerable and non-vulnerable functions from the six open-source projects, such as LibTIFF, Pidgin, FFmpeg, LibPNG, VLC media player, and Asterisk. The vulnerability labels were acquired from the National Vulnerability Database (NVD) [11] and the Common Vulnerability and Exposures (CVE) [12] websites. The algorithms are designed for the collective extraction of beneficial information from real-world vulnerability datasets in order to enhance vulnerability detection performance. The Word2vec [1] model is employed in the embedding layer of the Clock-Work Recurrent Neural network in order to convert an input sequence to meaningful embeddings.

Table 1. Dataset

Data source	Datasource/Collection	#of functions used/Collected	
		Vulnerable	**Non-Vulnerable**
Real-world Open Sources	FFmpeg		
	LibTIFF	213	5701
	LibPNG	96	731
	Pidgin	43	577
	VLC Media Player	29	8,050
	Asteriks	42	3,636

4.1. Results

In Tables 2-7, we compared the performances of the improved heuristic algorithms for detecting vulnerabilities based on the FFmpeg, LibTIFF, LibPNG, Pidgin, Asterisk, and VLC media Player datasets. EvoloPy toolbox contains twenty three benchmarks (F1-F23). In the optimizer.py you can setup your experiment by selecting the test sets. In this study on five test modules (F1-F5). The results demonstrate that the Asterisk dataset displayed the best performance with a 0.029643 error rate for hidden layer unit 128 and test F4, based on the CW-MVO algorithm, compared to the other vulnerability datasets. Nevertheless, according to the results, the worst error rate was found in the LibTIFF dataset with a 0.063467 error rate for hidden layer units 32 and test F5 based on the WOA algorithm. Generally, the FFpmeg, LibTIFF and Pidgin datasets exhibited close error rate performances, except for MVO algorithm. Concerning the other datasets, it was observed that the improved algorithm achieved the highest performance results in the Asteriks, VLC media player, Pidgin, LibPNG, LibTIFF, and FFpmeg datasets, respectively.

Table 2. Error Rate of compared Algorithms for FFpmeg Dataset

Test Benchmark	Hidden Layer units	WOA	CW-WOA	GWO	CW-GWO	MVO	CW-MVO
Test F1	32	0.054853	0.052401	0.0575321	0.04920	0.050653	0.05096
	64	0.053425	0.04912	0.0445252	0.04612	0.049034	0.04742
	128	0.047965	0.041231	0.0453258	0.041875	0.047532	0.040094
Test F2	32	0.046744	0.045536	0.0564363	0.050919	0.0553286	0.057168
	64	0.0478532	0.044321	0.0516742	0.047903	0.057754	0.054721
	128	0.0435731	0.04132	0.050584	0.04566	0.053522	0.053663
Test F3	32	0.0606471	0.056726	0.0534211	0.050791	0.0543457	0.049463
	64	0.057854	0.0541267	0.0543245	0.048925	0.056732	0.045412
	128	0.050765	0.052288	0.0513856	0.047164	0.0483878	0.0432609
Test F4	32	0.0564325	0.0517321	0.0564356	0.052452	0.056057	0.056463
	64	0.055736	0.0498425	0.05345673	0.049756	0.055743	0.052557
	128	0.049732	0.045733	0.0494565	0.047654	0.0564537	0.0534435
Test F5	32	0.0614543	0.057841	0.055843	0.054876	0.052345	0.05086
	64	0.0553561	0.059625	0.055372	0.05321	0.049872	0.04773
	128	0.0542423	0.051097	0.0508490	0.047535	0.0415678	0.042195

Table 3. Error Rate of compared Algorithms for LibTIFF Dataset

Test Benchmark	Hidden Layer units	WOA	CW-WOA	GWO	CW-GWO	MVO	CW-MVO
Test F1	32	0.0576353	0.050203	0.054352	0.04964	0.0575353	0.050649
	64	0.0512432	0.04734	0.050543	0.04682	0.055356	0.052134
	128	0.048676	0.043651	0.045684	0.043636	0.0523907	0.050036
Test F2	32	0.049756	0.044792	0.053453	0.048659	0.0598543	0.055804
	64	0.046532	0.042143	0.0504221	0.0440867	0.0558641	0.052178
	128	0.041344	0.040974	0.0478942	0.039435	0.052578	0.050932
Test F3	32	0.055632	0.052367	0.05673221	0.049543	0.0578975	0.050754
	64	0.057437	0.0521358	0.0523624	0.0485867	0.0545789	0.052468
	128	0.054633	0.051579	0.0485784	0.0443234	0.0534218	0.050732
Test F4	32	0.052459	0.050952	0.0597428	0.0546573	0.0575432	0.053494
	64	0.0513493	0.0470328	0.0534647	0.050535	0.0538098	0.049053
	128	0.049064	0.044573	0.050432	0.045867	0.0498752	0.046256
Test F5	32	0.063467	0.0597538	0.0545789	0.0519754	0.0508124	0.04572
	64	0.060342	0.0557321	0.05458445	0.05296365	0.048753	0.043723
	128	0.056313	0.0501735	0.0513461	0.04642805	0.0445809	0.0414695

Table 4. Error Rate of compared Algorithms for LibPNG Dataset

Test Benchmark	Hidden Layer units	WOA	CW-WOA	GWO	CW-GWO	MVO	CW-MVO
Test F1	32	0.044853	0.03772	0.0475732	0.0366264	0.0456772	0.039963
	64	0.037833	0.034085	0.0413855	0.0347854	0.0406432	0.0383445
	128	0.035356	0.0327045	0.0408253	0.0335466	0.03784214	0.0368952
Test F2	32	0.0495321	0.040558	0.0512345	0.0456779	0.0586328	0.0536874
	64	0.045364	0.041589	0.0509427	0.0437643	0.05743462	0.0527895
	128	0.039752	0.039753	0.0424525	0.039034	0.0528474	0.050643
Test F3	32	0.0543527	0.052356	0.0583252	0.0518514	0.0464632	0.0436784
	64	0.054523	0.0507543	0.0534653	0.0507432	0.0445639	0.0427895
	128	0.049792	0.047059	0.0519478	0.0457322	0.0413563	0.0403468
Test F4	32	0.0567943	0.049743	0.060642	0.049732	0.0584636	0.0516733
	64	0.0512428	0.0469325	0.0574736	0.046457	0.0556473	0.050634
	128	0.0465374	0.043582	0.0508321	0.0413468	0.053452	0.0498368
Test F5	32	0.056975	0.053623	0.054996	0.0506435	0.0595736	0.0458537
	64	0.054245	0.0525672	0.051847	0.0524632	0.0524573	0.0432466
	128	0.049802	0.039953	0.048735	0.0376784	0.0486352	0.035653

Table 5. Error Rate of compared Algorithms for Pidgin Dataset

Test Benchmark	Hidden Layer units	WOA	CW-WOA	GWO	CW-GWO	MVO	CW-MVO
Test F1	32	0.0598224	0.0526843	0.0535784	0.047535	0.0574531	0.054566
	64	0.05465893	0.051246	0.0524462	0.043567	0.0534625	0.051457
	128	0.0513750	0.05074	0.0467848	0.0424653	0.0507436	0.048965
Test F2	32	0.0512463	0.046445	0.0575743	0.053546	0.0619357	0.055684
	64	0.0508396	0.0434562	0.0547362	0.050434	0.0587485	0.052356
	128	0.0447497	0.042567	0.0534639	0.051467	0.0553568	0.050754
Test F3	32	0.0587942	0.052435	0.0567387	0.0497543	0.0596492	0.0445663
	64	0.0553683	0.050476	0.05432842	0.046543	0.0535783	0.0421455
	128	0.0507354	0.051389	0.0532424	0.0434656	0.04784281	0.040754
Test F4	32	0.0587639	0.050643	0.0565743	0.0507546	0.0556437	0.053784
	64	0.0528436	0.045345	0.0534564	0.045726	0.05547326	0.0507643
	128	0.04576932	0.042566	0.0475832	0.0416434	0.0497432	0.045878
Test F5	32	0.0609643	0.055743	0.058473	0.052455	0.0565493	0.053561
	64	0.0612485	0.053465	0.0513452	0.0506754	0.05178458	0.0496433
	128	0.0508467	0.048954	0.0487638	0.0464667	0.0478353	0.044527

Table 6. Error Rate of compared Algorithms for VLC Media PlayerDataset

Test Benchmark	Hidden Layer units	Improved Algorithms					
		WOA	CW-WOA	GWO	CW-GWO	MVO	CW-MVO
Test F1	32	0.0456352	0.395433	0.0498532	0.042456	0.0479425	0.041673
	64	0.0425739	0.040754	0.0453694	0.040643	0.043689	0.0398573
	128	0.0389431	0.037955	0.04052783	0.038954	0.0375392	0.03589
Test F2	32	0.0475378	0.0408753	0.0475489	0.042453	0.0497875	0.045643
	64	0.0439625	0.039855	0.0432563	0.0408674	0.0476542	0.0425824
	128	0.0356382	0.0348457	0.0387426	0.035353	0.0343637	0.0398756
Test F3	32	0.0538032	0.0499484	0.0568324	0.048873	0.0537509	0.043673
	64	0.0514587	0.047745	0.0553572	0.045635	0.0446982	0.041566
	128	0.0468743	0.040937	0.0445638	0.042546	0.0408532	0.040753
Test F4	32	0.0538721	0.0468476	0.0486379	0.040742	0.0546848	0.0464095
	64	0.0517939	0.0473456	0.0465395	0.04287567	0.0534743	0.0459372
	128	0.0459372	0.0428457	0.0443761	0.0413456	0.05075298	0.043524
Test F5	32	0.0578463	0.0513674	0.0597463	0.051456	0.0489573	0.044355
	64	0.0548790	0.0478473	0.0565302	0.0508474	0.04574712	0.042466
	128	0.0516840	0.048763	0.0535726	0.045245	0.0423524	0.040837

Table 7. Error Rate of compared Algorithms for Asteriks Dataset

Test Benchmark	Hidden Layer units	Algorithms					
		WOA	CW-WOA	GWO	CW-GWO	MVO	CW-MVO
Test F1	32	0.0457943	0.040536	0.04336456	0.0356466	0.04795821	0.040633
	64	0.0424672	0.037899	0.04074351	0.0367847	0.0445793	0.039745
	128	0.0409201	0.035854	0.03854974	0.0335366	0.0409536	0.0346783
Test F2	32	0.0425565	0.0316783	0.0409732	0.038646	0.04357893	0.0368476
	64	0.0389532	0.0390624	0.0397327	0.0375673	0.0416897	0.03357221
	128	0.0307432	0.031735	0.0335912	0.032573	0.0479434	0.0314742
Test F3	32	0.0407245	0.0375467	0.03356362	0.0324567	0.0485892	0.030635
	64	0.0375372	0.031455	0.0306361	0.0308422	0.043680	0.039654
	128	0.0386847	0.0396325	0.0237975	0.039644	0.04168361	0.035689
Test F4	32	0.0376893	0.0324578	0.0398526	0.03254673	0.04876483	0.033567
	64	0,03482562	0.313657	0.0346938	0.0397455	0.0436789	0.036642
	128	0.03047314	0.306773	0.0297476	0.0345632	0.0456834	0.029643
Test F5	32	0.0384630	0.0346746	0.0386953	0.0339572	0.0435893	0.0377593
	64	0.0335693	0.0345664 3	0.0326891	0.0397455	0.4075256	0.0324567
	128	0.0313574	0.0335736	0.02975327	0.0300484	0.0426894	0.031546

The improved CW-WOA model achieved the best results as a 0.306773 error rate based on test F4 and a 0.0327045 error rate based on test F1 for the Asteriks and LibPNG datasets, respectively, using 128 hidden layer units. Moreover, it was observed that the CW-GWO model achieved the best performance results, such as a 0.0300484 error rate based on test F5 and a 0.0335466 error rate based on test F1 for the Asteriks and LibPNG datasets, respectively, using 128 hidden layer units. The improved CW-MVO model obtained a 0.029643 error rate in test F4 and a 0.035653 error rate in test F5 for the Asteriks and LibPNG datasets, respectively, using 128 hidden layer units. The obtained results indicate that the Asteriks and Pidgin datasets achieved the highest performance for Test-F3 benchmark. However, the findings demonstrate that the best classification error rate performance exhibited for FFmpeg and VLC Media Player datasets based on the Test-F2 benchmark. Furthermore, the best results showed for LibPng and LibTIFF datasets based on Test-F1 benchmark.

All experimental results show that low hidden layers process, retain, and output high error rates. Meanwhile, high hidden layers generally concentrate on the local, high-frequency information with low error-rate performances.

5. CONCLUSION

The application of nature-inspired metaheuristic optimization algorithms for vulnerability detection is an immature area of research having numerous problems waiting for a solution. The representation learning capability of nature-inspired algorithms to optimize patterns of software vulnerabilities and their customizable structure are promising for the automated learning of complex vulnerable patterns, which will motivate and attract a higher number of researchers to ensure a contribution to the said field with high potential. According to the findings acquired, the proposed

framework leverages the detection rate of the optimized patterns well, which ensures that vulnerable programming patterns learned from software source projects facilitate the representation generation on a target project to predict vulnerabilities better.
Future studies may include effectively optimized representations with the updated vulnerability dataset to achieve recently improved vulnerability detection performance.

Acknowledgement

The present paper does not include any research with human participants conducted by any of the authors.

REFERENCES

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. Efficient Estimation of Word Representations in Vector Space.doi.org/10.48550/arXiv.1301.3781, 2013.

[2] Shi, Y., Wang, Y., & Zheng, H. Wind Speed Prediction for Offshore Sites Using a Clockwork Recurrent Network. Energies, 2022, 15(3), 751.

[3] Koutník, J., Greff, K., Gomez, F., & Schmidhuber, J. A Clock-Work RNN, doi.org/10.48550/arXiv.1402.3511. 2014.

[4] Khurma, R.A., Aljarah, I., Sharieh, A., Mirjalili, S. EvoloPy-FS: An Open-Source Nature-Inspired Optimization Framework in Python for Feature Selection. In: Mirjalili, S., Faris, H., Aljarah, I. (eds) Evolutionary Machine Learning Techniques. Algorithms for Intelligent Systems. Springer, Singapore.https://doi.org/10.1007/978-981-32-9990-0_8, 2020.

[5] Özlem B. D., Canan B. Ş., Prediction of phishing websites with deep learning using WEKA environment, Avrupa Bilim ve Teknoloji Dergisi, vol. 24, pp. 35-41, doi:10.31590/ejosat.901465, 2021.

[6] Guha, R., Chatterjee, B., Khalid Hassan, S.K., Ahmed, S., Bhattacharyya, T., Sarkar, R. Py_FS: A Python Package for Feature Selection Using Meta-Heuristic Optimization Algorithms. In: Das, A.K., Nayak, J., Naik, B., Dutta, S., Pelusi, D. (eds) Computational Intelligence in Pattern Recognition . Advances in Intelligent Systems and Computing, vol 1349. Springer, Singapore. https://doi.org/10.1007/978-981-16-2543-5_42, 2022.

[7] Riyahi, M, Rafsanjani, MK, Gupta, BB, Alhalabi, W. Multiobjective whale optimization algorithm based feature selection for intelligent systems. Int J Intell Syst. 37: 9037- 9054. doi:10.1002/int.22979, 2022.

[8] Abu Khurma, R.; Aljarah, I.; Sharieh, A.; Abd Elaziz, M.; Damaševičius, R.; Krilavičius, T. A Review of the Modification Strategies of the Nature Inspired Algorithms for Feature Selection Problem. Mathematics,10,464.https://doi.org/10.3390/math10030464. 2022.

[9] Rohlfs, C. Generalization in Neural Networks: A Broad Survey. 2022, arXiv. https://doi.org/10.48550/arXiv.2209.01610.

[10] Zhang, G, Ding, Z, Xu, J, Zhong, G, Jiang, N, Zhang, Y. Reasoning and tracing of information security events in the expressway networking system based on deep learning. Int J Intell Syst., 37: 8988- 9012. 2022, doi:10.1002/int.22977.

[11] [Internet]. [cited 2022 November 27]. Available from: https://nvd.nist.gov/

[12] [Internet]. [cited 2022 October 13]. Available from: https://cve.mitre.org/

[13] Batur Şahin, C., Learning Optimized Patterns of Software Vulnerabilities with the Clock-Work Memory Mechanism. Avrupa Bilim ve Teknoloji Dergisi,156-165.2022, https://doi.org/10.31590/ejosat.1159875.

[14] C. B. Şahin, DCW-RNN: Improving Class Level Metrics for Software Vulnerability Detection Using Artificial Immune System with Clock-Work Recurrent Neural Network, 2021 International Conference on INnovations in Intelligent SysTems

and Applications (INISTA), pp. 1-8, 2021, doi: 10.1109/INISTA52262.2021.9548609.

[15] Tansel D., Ayça D. and Hakan K. A Comprehensive Survey on Recent Metaheuristics for Feature Selection. , Neurocomputing, 494, 269-296, 2022.

[16] Abd Elaziz, M., Dahou, A., Abualigah, L. et al. Advanced metaheuristic optimization techniques in applications of deep neural networks: a review. Neural Comput & Applic 33, 14079–14099, 2021, https://doi.org/10.1007/s00521-021-05960-5.

[17] Şahin, C.B., Dinler, Ö.B. & Abualigah, L. Prediction of software vulnerability based deep symbiotic genetic algorithms: Phenotyping of dominant-features. Appl Intell 51, 8271–8287. https://doi.org/10.1007/s10489-021-02324-3. 2021.

[18] Singh, S.K., Chaturvedi, A. Applying Deep Learning for Discovery and Analysis of Software Vulnerabilities: A Brief Survey. In: Pant, M., Kumar Sharma, T., Arya, R., Sahana, B., Zolfagharinia, H. (eds) Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, vol 1154. Springer, Singapore. 2020, https://doi.org/10.1007/978-981-15-4032-5_59.

[19] Dinler, Ö.B., Nizamettin A. An Optimal Feature Parameter Set Based on Gated Recurrent Unit Recurrent Neural Networks for Speech Segment Detection. Appl. Sci., 10, 1273. 2020, https://doi.org/10.3390/app10041273.

[20] Ullah, A., Aznaoui, H., Sahin, C. B, Sadie, M., Ozlem Dinler, Ö.B, Imane, L, Cloud computing and 5G challenges and open issues,11-3, 2022, http://doi.org/10.11591/ijaas.v11.i3.pp187193.

[21] Batur Şahin C. , Batur Dinler Ö. , Abualigah L. Analysis of Risk Factors in the Scope of Distributed Software Team Structure. EJOSAT. (28): 417-424, 2021.

[22] Batur Şahin, C., Abualigah, L. A novel deep learning-based feature selection model for improving the static analysis of vulnerability detection. Neural Comput & Applic 33, 14049–14067, 2021. https://doi.org/10.1007/s00521-021-06047-x.

[23] Batur Dinler Ö. , Batur Şahin C. Prediction of Phishing Web Sites with Deep Learning Using WEKA Environment. EJOSAT. 2021; (24): 35-41.

[24] Batur Şahin C. , Diri B. Sequential Feature Maps with LSTM Recurrent Neural Networks for Robust Tumor Classification. Balkan Journal of Electrical and Computer Engineering. 2021; 9(1): 23-32.

[25] Ullah A. , Batur Dinler Ö. , Batur Şahin C. The Effect of Technology and Service on Learning Systems During the COVID-19 Pandemic. EJOSAT. 2021; (28): 106-114.

[26] Ullah A. , Aznaoui H., Batur Şahin C. , Batur, Daanoune I., Dinler Ö. ,CloudIoT paradigm acceptance for e-learning: analysis and future challenges. Jurnal Informatika. 2022; 16(3): 154-168.ISSN 1978-0524.