Avrupa Bilim ve Teknoloji Dergisi
Sayı 41, S. 156-165, Kasım 2022

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
No. 41, pp. 156-165, November 2022

Research Article

Learning Optimized Patterns of Software Vulnerabilities with the Clock-Work Memory Mechanism

Canan Batur Şahin[1*]

[1*] Malatya Turgut Özal Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, Yazılım Mühendisliği Bölümü, Malatya, Türkiye, (ORCID:0000-0002-2131-6368), canan.batur@ozal.edu.tr.

(First received 9 August 2022 and in final form 18 September 2022)

(**DOI:** 10.31590/ejosat.1159875)

ATIF/REFERENCE: Şahin, B. C. (2022). Learning Optimized Patterns of Software Vulnerabilities with the Clock-Work Memory Mechanism. *Avrupa Bilim ve Teknoloji Dergisi*, (41), 156-165.

Abstract

It is possible to better provide the security of the codebase and keep testing efforts at a minimum level by detecting vulnerable codes early in the course of software development. We assume that nature-inspired metaheuristic optimization algorithms can obtain "optimized patterns" from vulnerabilities created in an artificial manner. This study aims to use nature-inspired optimization algorithms combining heterogeneous data sources with the objective of learning optimized representations of vulnerable source codes. The chosen vulnerability-relevant data sources are cross-domain, involving historical vulnerability data from variable software projects and data from the Software Assurance Reference Database (SARD) comprising vulnerability examples. The main purpose of this paper is to outline the state-of-the-art and to analyze and discuss open challenges with regard to the most relevant areas in the field of bio-inspired optimization based on the representation of software vulnerability. Empirical research has demonstrated that the optimized representations produced by the suggested nature-inspired optimization algorithms are feasible and efficient and can be transferred for real-world vulnerability detection.

Keywords: Feature Selection, Nature-inspired Algorithm, Optimization, Representation learning, Software Vulnerability.

Saat-Hafıza Mekanizması ile Yazılım Güvenlik Açıklarının Optimize Edilmiş Örüntülerini Öğrenme

Öz

Yazılım geliştirme sürecinin başlarında hassas kodları belirleyerek kod tabanının güvenliğini daha iyi sağlamak ve test çabalarını minimum düzeyde tutmak mümkündür. Doğa esinli üstsezgisel optimizasyon algoritmalarının yapay bir şekilde meydana getirilen güvenlik açıklarından "optimize edilmiş örüntüler" elde edebileceğini düşünüyoruz. Bu çalışma, heterojen veri kaynaklarını hassas kaynak kodlarının optimize edilmiş gösterimlerini öğrenme hedefiyle birleştiren doğa-esinli optimizasyon algoritmalarını kullanmayı amaçlamaktadır. Seçilen güvenlik açığı ile ilgili veri kaynakları alanlar arası kaynaklar olup, farklı yazılım projelerine ait geçmiş güvenlik açığı verilerini içeren Yazılım Güvencesi Referans Veritabanı'nın (YGRV) sağladığı verileri kapsar. Bu makalenin temel amacı, son teknolojinin ana hatlarını çizmek ve yazılım güvenlik açığının gösterimine dayalı biyo-esinli optimizasyon alanındaki en ilgili alanlara yönelik mevcut zorlukları analiz etmek ve tartışmaktır. Ampirik araştırmalar, önerilen doğa esinli optimizasyon algoritmaları tarafından üretilen optimize edilmiş gösterimlerin uygulanabilir ve etkin olduğunu ve gerçek dünyadaki güvenlik açığı tespiti için kullanılabileceğini ortaya koymuştur.

Anahtar Kelimeler: Öznitelik Seçimi, Doğa-Esinli Algoritmalar, Optimizasyon, Gösterim Öğrenimi, Yazılım Güvenlik Açığı.

[*] Corresponding Author: canan.batur@gmail.com

1. Introduction

Software vulnerabilities make up security risks for software systems with increasing importance, utilized to attack and damage systems [17]. Managing security in cyberspace must be inspired by systems with advanced complexity. In the process of evolution, natural propensities in complex systems (e.g., plants and animals), enabling survival by adaptation, have been developed by nature.

Algorithms inspired by nature represent population-based metaheuristics inspired by various natural phenomena. Over a very long time, nature has evolved to bring about intelligent behavioral properties and biological phenomena, in which self-learning, adaptability, efficiency, and robustness allow biological agents to undertake complex tasks. Generally, it is possible to categorize nature-inspired algorithms as swarm intelligence and evolutionary algorithms. Natural swarms, e.g., ant colonies, flocks of birds, and schools of fish, are simulated by swarm intelligence algorithms. Cuckoo Search (CS), Multi-Verse Optimizer (MVO), Grey Wolf Optimizer (GWO), Whale Optimization Algorithm (WOA), Moth-flame optimization (MFO), Firefly Algorithm (FFA), Bat Algorithm (BAT), etc. can be listed among the current swarm intelligence algorithms. The majority of the swarm intelligence algorithms obtain the best solution as a result of information exchange between individuals in the swarm.

A number of evolution and natural selection-related concepts in the Darwinian theory have inspired evolutionary-based algorithms. Evolution Strategy (ES), Genetic Programming (GP), and Genetic Algorithm (GA) are among the mentioned algorithms. The algorithms in question employ various strategies for evolving and find good solutions for challenging problems on the basis of evolutionary operators, such as crossover, elitism, and mutation.

As problems become more complex, novel optimization techniques are needed more over the last few decades. This paper presents a simple but powerful modification to the standard Recurrent Neural Network (RNN) architecture, the Clock-work RNN (CW-RNN) with nature-inspired algorithms. Clock-work memory, with the hidden layer divided into modules, makes computations only at its prescribed clock rate. Here, the long-term dependent optimized patterns are captured for each metaheuristic algorithm, with various sections (modules) of the RNN hidden layer operating at various clock periods.

In the current work, we further research the representation learning capability of clever algorithms by learning vulnerable patterns from vulnerability-relevant data sources. It is hypothesized that the source for learning optimized vulnerable code patterns must not be restricted to the historical vulnerability data source that involves real-world software projects. Furthermore, it is necessary that a vulnerability-relevant data source that involves artificial vulnerability samples is utilized as a vulnerability knowledge base.

Our research framework investigates the representation learning capability of clever algorithms based on clock-work memory in order to automatically extract high-level optimized representations, indicating vulnerable patterns from vulnerable-relevant data sources.

The algorithms are compared for the collective extraction of beneficial information from real-world vulnerability data as well as from synthetic data sets to enhance the performance in detecting vulnerabilities. Each algorithm is trained by a historical vulnerability data source, which can be utilized as a feature extractor for producing optimized features that involve the vulnerable information learned from the vulnerability data source. First, meta-heuristic algorithms are fed by means of the vulnerability-relevant data sources above. Afterward, the trained meta-heuristic algorithms are utilized as feature extractors using the long-dependence mechanism of clock-work recurrent neural networks.

The data are fed to every trained meta-heuristic algorithm based on clock-work memory to acquire an optimized subset of vulnerability representations as features. Second, the learned optimized representations are combined as features by concatenating the representations. Three meta-heuristic algorithms were utilized versus to its standard algorithms to predict vulnerabilities using optimized patterns as features.

The current work makes the following main contributions:

- The current work represents the first model, in which the long-term dependency is addressed by the Clock-work Recurrent Neural Network for software vulnerability detection problems using nature-inspired metaheuristic optimization algorithms.
- In the study, the first deep learning-based vulnerability detection system was created with metaheuristic optimization algorithms, aiming to predict vulnerabilities with the aim of learning optimized software features.
- A novel hybrid framework is proposed, improving the detection capability of bio-inspired population-based heuristic approaches based on a clock-work memory for learning optimized patterns to extract the optimized features for detecting software vulnerable codes.
- Our framework's design is validated by conducting experiments, and it is shown that the usage of clock-work memory as optimized-feature representations and a separated classifier with the objective of training on the extracted features enhances the performance in detecting vulnerabilities.

The remaining part of the current work has the following organization. An explanation of the Material and methods is contained in Part 2. Part 3 describes the Proposed Model. Part 4 contains the Results and Discussion. Part 6 summarizes the conclusion and future research.

2. Material and Methods

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

2.1. Nature-Inspired Algorithms

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

2.1.1. Moth-Flame Optimization (MFO)

In MFO, the natural behavior of the actual moth is mimicked. In accordance with the said theory, moths represent

solutions, and their spatial positions represent problem parameters. The best-acquired position (optimal position) by moths is kept in flames. The calculation of the primary MFO algorithm is performed as shown below:

$$MFO = (I, P, T) \quad (1)$$

where I refers to the function utilized for initializing a random moth population and their fitness values; P denotes the primary function moving moths around the search space; T represents the termination function returning true in case of satisfying the termination criterion; on the contrary, it returns false. In the main function P, the flames are updated the position of moths through the following equation:

$$M_i = D_i * e_{bt} * \cos(2\pi t) + F_j \quad (2)$$

Where M_i refers to moth i, and F_j denotes flame j^{th}, D_i refers to the distance between moth i and flame j, *b* represents a constant that defines the logarithmic spiral's shape, and t refers to a random number in [-1, 1]. The computation of D_i is made in the following way.

$$D_i = |F_j - X_i| \quad (3)$$

where D_i denotes the distance between moth i and flame j, F_j refers to flame j, and M_i refers to moth i.

The number of flames is computed based on Eq. 4:

$$\text{Flame number} = \text{round}(N - l * \frac{N-1}{T}) \quad (4)$$

Where *N* refers to the highest number of flames, *l* denotes the current flame number, and *T* refers to the highest number of iterations.

2.1.3. Particle Swarm Optimization (PSO)

Eberhart and Kennedy presented the PSO algorithm in 1995 [33]. In PSO, the solution to every optimization problem is a bird's position in the search space, named a "particle," whereas the problem's optimal solution refers to the corn field's position.

A brief summary of the main steps of the PSO algorithm is shown below:

Step 1: Initialize the parameters, including the particle swarm, position $X_i(t)$, and kinematic velocity $v_i(t)$ of particle i.

Step 2: Assess the fitness of the particle by means of the deployed fitness function.

Step 3: Update the optimal position and the velocity of all particles by utilizing equations (12) and (13).

$$v_i(t+1) = \omega v_i(t) + \varphi_1 \left[p_{best} - x_i(t)\right] + \varphi_2 \left[g_{best} - x_i(t)\right] \quad (12)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (13)$$

The terms p_{bestid} represent the optimal positions of individual particles, while g_{bestid} refers to the swarm's optimal positions. ω denotes the inertia factor controlling the impact of the front velocity on the current velocity. *t* represents the current state of the swarm and particle.

φ1 and φ2 refer to social parameters.

Step 4: Repeat steps 2 and 3 until meeting the terminating conditions.

2.1.5. Firefly Algorithm (FFA)

In the said algorithm, the interaction of fireflies by means of their flashing lights is mimicked. The algorithm accepts that all fireflies are unisex, referring to the possibility of any firefly being attracted by any other firefly. There is a direct proportion between a firefly's attractiveness and its brightness, depending on the objective function. A brighter firefly will attract another firefly. Moreover, there is a decrease in brightness with distance according to the inverse-square law, as shown in Eq. (14):

$$I \prec \frac{1}{r^2} \quad (14)$$

In case of the light passing through a medium with a light absorption coefficient γ, it is possible to express the light intensity at a distance of r from the source, as displayed in Eq. (15):

$$I = I_0 e^{-\gamma r^2} \quad (15)$$

where I0 represents the light intensity at the source. Likewise, it is possible to express the brightness, β, as shown in Eq. (16):

$$\beta = \beta_0 e^{-\gamma r^2} \quad (16)$$

A generalized brightness function for ω ≥ 1 is presented in Eq. (17). Actually, it is possible to use any monotonically decreasing function.

$$\beta = \beta_0 e^{-\gamma r^{\omega}} \quad (17)$$

Concerning the brightest firefly, it will do a local search by moving in a random way in its neighborhood. Therefore, in case of two fireflies, if firefly j has higher brightness compared to firefly i, then firefly i will move in the direction of firefly j by utilizing the updating formula in Eq. (18):

$$x_i := x_i + \beta_0 e^{-\gamma r^2}(x_j - x_i) + \alpha(\varepsilon() - 0.5) \quad (18)$$

where β0 represents the attractiveness of xj at r = 0, and β0 = 1 for implementation, γ refers to an algorithm parameter determining the degree at which the updating process depends on the distance between the two fireflies, α denotes an algorithm parameter for the random movement's step length, and ε() represents a random vector from a uniform distribution with values in the range of 0 and 1.

The said updates of the fireflies' position continue with iteration until meeting a termination criterion.

3. The Proposed Method

Bio-inspired computing is an active area due to its nature, solving numerous real-world problems.

3.1. Problem Formulation and Representation

It was accepted that every data was expressed as $x = [x_1, x_2, \ldots, x_n]$, where n represents the data sample's length, and training data have a corresponding target value of vulnerable or not

vulnerable, which is defined in software vulnerability models. Every candidate solution is expressed with a length n, in which n refers to the total number of features.

3.2. Fitness function

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.

It is possible to consider the optimized pattern with the minimum values of fitness for vulnerabilities as the most representative example of a vulnerability population. The algorithms' fitness function was calculated in line with the classifier equation's error rate.

3.2. 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 are the most appropriate for vulnerability prediction in software systems were obtained. To date, no strategy or idea has been adopted on the Clock-Work memory mechanism-based metaheuristic algorithms for vulnerability detection problems. Reasoning about processes at multiple time scales is facilitated by Clock-Work RNN (CW-RNN) models, making calculations only at the prescribed clock rate. Neurons of various modules are connected on the basis of the modules' clock periods [23].

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 every CW-RNN time step *t*, only the outputs of module i satisfying (t MOD T_i) = 0 are active. It is arbitrary to choose the set of periods {T_1, . . . , T_g}. In the present work, the exponential series of periods is utilized; the i^{th} 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.

In this paper, *Max_iter*, *lb*, *ub*, *dim* and *SearchAgents_no*, *Vmax*, *PopSize*, *wMax*, *wMin*, *c1*, *c2, number of fireflies,WEP_Max, and WEP_Min* parameters for metaheuristic algorithms are set to the values *1000*, *-150*, *+150*, *25*, *30*, *10*, *100*, *0.9*, *0.2*, *3*, *3*, *100, 1* and *0.2* respectively. 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.

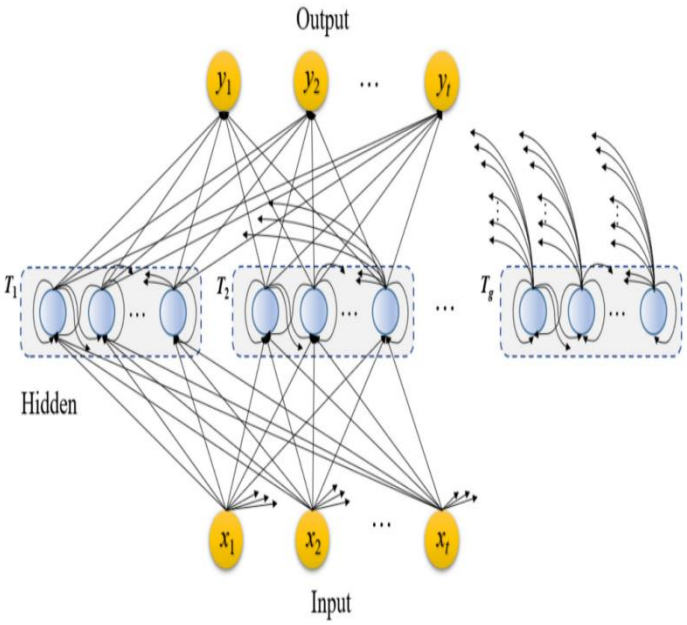

Figure 1: Architecture of the Clock-Work Recurrent Neural Network [4]

4. Results and Discussion

The algorithms' experimental performances in detecting vulnerabilities are described in the present part in order to indicate the efficiency of the compared algorithms.

4.1. Data Collection

The data source includes vulnerable and non-vulnerable functions from the 6 open-source projects, such as LibTIFF, Pidgin, FFmpeg, LibPNG, Asterisk, and VLC media player. The vulnerability labels were acquired from the National Vulnerability Database (NVD) [13] and the Common Vulnerability and Exposures (CVE) [14] websites. The algorithms are designed for the collective extraction of beneficial information from real-world vulnerability data sets in order to enhance vulnerability detection performance. The Word2vec [1] model is used in the embedding layer of the Clock-Work Recurrent Neural network for converting input sequence to meaningful embeddings.

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 {T1, . . . ,Tg} 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

4.2. Results

In Tables 2-7, we compared the performances of the standard and improved heuristic algorithms for detecting vulnerabilities based on the LibTIFF, FFmpeg, Pidgin, LibPNG, Asterisk, and VLC media Player datasets. The results demonstrate that the Asterisk dataset displayed the best performance with a 0.010543 error rate for hidden layer unit 128 and Test F5, based on CW-MFO algorihm, compared to the other vulnerability datasets. Nevertheless, according to the results, worst error rate performance was found in the FFmpeg dataset with 0.0828 error rate for hidden layer units 32 and Test F1 based on CW-PSO algorihm. In point of other datasets, it was observed that the improved algorithm achieved the highest performance results in the Asteriks, LibPNG, VLC media player, LibTIFF, FFmpeg and Pidgin data sets, respectively.

The statistical performance of the improved CW-PSO model outperformed worst results than the other improved algorithms for all datasets. It was observed that the CW-MFO and CW-FFA models generally gave close results than CW-PSO algorithm. In Table 7, there is generally observation about dramatic change for improved algorithms especially for CW-MFO algorithm based on Test F5.

All experimental results show the low-hidden layers process, retain and output the high error rate results, whereas the high-hidden layers focus on the local, high frequency information having the low error-rate performances, generally. Also, improved heuristic agorithms gave better results than the stardard heuristic agorithms.

Table 1. Dataset

Data source	**Data source/Collection**	**#of functions used/Collected**	
		Vulnerable	**Non-Vulnerable**
Real-world Open Sources	FFmpeg	213	5701
	LibTIFF	96	731
	LibPNG	43	577
	Pidgin	29	8,050
	VLC Media Player	42	3,636
	Asteriks	56	14,648

Table 2. Error Rate of compared Algorithms for FFpmeg Dataset

Test Benchmark	Hidden Layer units	Algorithms					
		MFO	CW-MFO	PSO	CW-PSO	FFA	CW-FFA
Test F1	32	0.067895	0.05219	0.0878	0.0828	0.05435	0.0519
	64	0.055744	0.04534	0.074677	0.07105	0.05096	0.04813
	128	0.051643	0.041250	0.068755	0.065856	0.043535	0.040631
Test F2	32	0.05896	0.046725	0.0733456	0.070673	0.048954	0.045455
	64	0.05357	0.045909	0.072245	0.069109	0.04563	0.043768
	128	0.051346	0.044105	0.06775	0.06496	0.04085	0.03912
Test F3	32	0.063245	0.05593	0.072467	0.06714	0.053563	0.050874
	64	0.0608543	0.05334	0.069483	0.065003	0.05134	0.049392
	128	0.058643	0.051325	0.06643	0.066585	0.049753	0.047466
Test F4	32	0.063134	0.053368	0.064837	0.063464	0.047536	0.042789
	64	0.060563	0.051345	0.061864	0.059202	0.045636	0.0408238
	128	0.059533	0.047762	0.065355	0.061543	0.043543	0.0417463
Test F5	32	0.0657843	0.054567	0.077544	0.07321	0.055323	0.052574
	64	0.060424	0.055216	0.07343	0.072464	0.054636	0.054813
	128	0.055733	0.051784	0.07134	0.070567	0.054552	0.052762

Table 3. Error Rate of compared Algorithms for LibTIFF Dataset

Test Benchmark	Hidden Layer units	Algorithms					
		MFO	CW-MFO	PSO	CW-PSO	FFA	CW-FFA
Test F1	32	0.053245	0.05065	0.08197	0.07903	0.051865	0.050619
	64	0.048498	0.04371	0.076539	0.07353	0.049689	0.04725
	128	0.045355	0.040576	0.073256	0.071326	0.046874	0.045327
Test F2	32	0.047841	0.043642	0.06758	0.066436	0.045577	0.04454
	64	0.046524	0.041532	0.065737	0.063431	0.043356	0.041953
	128	0.042452	0.0403526	0.061546	0.060672	0.04087	0.0396304
Test F3	32	0.058643	0.0547213	0.065864	0.063255	0.053573	0.052359
	64	0.053547	0.052932	0.062568	0.061873	0.0508563	0.0495356
	128	0.054564	0.055164	0.064357	0.06232	0.048659	0.046534
	32	0.056432	0.053236	0.05978	0.061245	0.043566	0.040764

Test F4	**64**	0.055323	0.050764	0.061379	0.06327	0.0436789	0.04157363
	128	0.05075	0.0498327	0.064781	0.06048	0.0408765	0.03870357
	32	0.0545634	0.052354	0.075468	0.07255	0.0535667	0.05074732
Test F5	**64**	0.052434	0.0515946	0.072357	0.070732	0.0525608	0.05189543
	128	0.0514789	0.0480953	0.071479	0.07067	0.054672	0.0529764

Table 4. Performance of Algorithms forLibPNG Dataset

Test Benchmark	**Hidden Layer units**	**Improved Algorithms**					
		MFO	**CW-MFO**	**PSO**	**CW-PSO**	**FFA**	**CW-FFA**
	32	0.043688	0.037846	0.064325	0.059636	0.048543	0.0419016
Test F1	**64**	0.041252	0.034247	0.062356	0.0634652	0.0456364	0.0425733
	128	0.034673	0.030421	0.06244	0.060678	0.041356	0.0389311
	32	0.037576	0.036674	0.069086	0.067632	0.0464675	0.0419644
Test F2	**64**	0.034632	0.033755	0.065746	0.062474	0.0446631	0.0407453
	128	0.033525	0.0310357	0.063563	0.06072	0.0408334	0.038647
	32	0.053356	0.050533	0.066576	0.061467	0.057982	0.0517858
Test F3	**64**	0.050734	0.0480536	0.06446	0.06345	0.0515674	0.0508432
	128	0.049874	0.0470734	0.062355	0.061478	0.052578	0.0496433
	32	0.053694	0.051346	0.060843	0.058532	0.0547683	0.0406532
Test F4	**64**	0.052596	0.0490645	0.057574	0.053547	0.042675	0.0396542
	128	0.048464	0.0436875	0.054356	0.052724	0.039868	0.0379432
	32	0.045746	0.041795	0.064632	0.061456	0.0489554	0.045363
Test F5	**64**	0.046467	0.043245	0.062456	0.0608432	0.044678	0.0424842
	128	0.04635	0.0410643	0.059641	0.0579533	0.043678	0.0408426

Table 5. Performance of Algorithms for Pidgin Dataset

Test Benchmark	**Hidden Layer units**	**Algorithms**					
		MFO	**CW-MFO**	**PSO**	**CW-PSO**	**FFA**	**CW-FFA**
	32	0.061567	0.053467	0.059064	0.057532	0.057458	0.053573
Test F1	**64**	0.058642	0.0504323	0.055798	0.054685	0.055742	0.0547894
	128	0.053567	0.049736	0.052356	0.051894	0.052345	0.050543
	32	0.049732	0.048643	0.072533	0.068954	0.051546	0.0469755

Test F2	**64**	0.043467	0.043562	0.067843	0.065764	0.047545	0.0445784
	128	0.044315	0.041374	0.060985	0.060745	0.045643	0.043665
	32	0.054178	0.051456	0.065736	0.063563	0.057535	0.0526674
Test F3	**64**	0.049746	0.046975	0.065743	0.061345	0.054356	0.050853
	128	0.055464	0.050753	0.0608345	0.059043	0.050786	0.0470532
	32	0.059754	0.0565732	0.061356	0.060643	0.044635	0.0414678
Test F4	**64**	0.055548	0.052457	0.058535	0.055356	0.043534	0.0390654
	128	0.054356	0.051473	0.057097	0.056894	0.039643	0.0357547
	32	0.060532	0.052466	0.070643	0.06895	0.054244	0.050746
Test F5	**64**	0.058743	0.050754	0.067843	0.0613456	0.052442	0.051467
	128	0.0524553	0.0517577	0.058546	0.056733	0.048433	0.045783

Table 6. Performance of Algorithms for VLC Media Player Dataset

Test Benchmark	**Hidden Layer units**	**Algorithms**					
		MFO	**CW-MFO**	**PSO**	**CW-PSO**	**FFA**	**CW-FFA**
	32	0.046535	0.042784	0.059646	0.051643	0.0485321	0.042466
Test F1	**64**	0.043586	0.040643	0.048643	0.043677	0.0445632	0.040854
	128	0.040621	0.038975	0.043678	0.039864	0.038743	0.034677
	32	0.048432	0.0428478	0.050749	0.045633	0.458952	0.409847
Test F2	**64**	0.045355	0.038724	0.048695	0.043253	0.4324535	0.3764345
	128	0.041245	0.033859	0.046784	0.040932	0.035652	0.0318563
	32	0.054355	0.050863	0.053563	0.0568732	0.0542454	0.04853
Test F3	**64**	0.051234	0.048762	0.048676	0.0513456	0.0472312	0.0436275
	128	0.054234	0.041456	0.055563	0.0508732	0.046343	0.04146
	32	0.056345	0.048725	0.0497642	0.0425643	0.043522	0.0379464
Test F4	**64**	0.052355	0.046832	0.044774	0.040826	0.363587	0.309825
	128	0.05352	0.0425736	0.047324	0.0386754	0.045673	0.0369378
	32	0.055354	0.0508373	0.045642	0.045763	0.056732	0.050745
Test F5	**64**	0.050743	0.0482895	0.041433	0.0423455	0.053429	0.048725
	128	0.047842	0.046356	0.043566	0.040824	0.048563	0.045256

Table 7. Performance of Algorithms for Asteriks Dataset

Test Benchmark	Hidden Layer units	Algorithms					
		MFO	CW-MFO	PSO	CW-PSO	FFA	CW-FFA
Test F1	32	0.0386432	0.034264	0.054673	0.049827	0.3535783	0.32656
	64	0.035635	0.033653	0.049834	0.0478956	0.036732	0.03368
	128	0.031853	0.0308576	0.045636	0.0412345	0.037433	0.031464
Test F2	32	0.045356	0.038044	0.037982	0.0331075	0.048532	0.0406234
	64	0.0406437	0.0330852	0.034768	0.0315673	0.043525	0.0387542
	128	0.0376324	0.030822	0.031242	0.030827	0.036742	0.0320483
Test F3	32	0.0335663	0.029840	0.032784	0.0274567	0.040873	0.0334556
	64	0.032566	0.027094	0.029842	0.025633	0.0345625	0.0315732
	128	0.028643	0.023635	0.025345	0.0234522	0.025732	0.0245614
Test F4	32	0.034564	0.0247847	0.0286734	0.025784	0.031876	0.0229540
	64	0.034245	0.0235723	0.030742	0.027733	0.027832	0.02084563
	128	0.029632	0.0168767	0.027633	0.022472	0.025353	0.018948
Test F5	32	0.030743	0.0146865	0.034724	0.0282444	0.038632	0.033565
	64	0.027435	0.0126784	0.033826	0.0245673	0.036535	0.030875
	128	0.022563	0.010543	0.024242	0.020743	0.027633	0.022456

5. Conclusion

This paper provides software vulnerability detection by defining a workflow for the learning of optimized feature representations for vulnerability detection by nature-inspired metaheuristic optimization algorithms. The current study researches a new architecture for vulnerability detection and prediction tasks with the clock-work memory mechanism, which operates at different hidden layer units, with nature-inspired algorithms to optimize patterns of software vulnerabilities. According to the findings acquired, the proposed algorithm performs well in terms of the detection rate of the optimized patterns.

Future studies may include semantic representation generation techniques to explore whether these methods can produce more effective optimized representations with more semantic information preserved to achieve improved vulnerability detection performance.

6. Acknowledge

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

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, (2013) “Efficient estimation of word representations in vector space,” arXiv preprint arXiv:1301.3781.

[2] Shi, Y., Wang, Y., & Zheng, H. (2022). Wind Speed Prediction for Offshore Sites Using a Clockwork Recurrent Network. Energies, vol.15, no. 3, 751.

[3] J. Koutnik, K. Greff, F. Gomez, J.Schmidhuber. (2014) A Clockwork RNN," Proceedings of the 31st International Conference on Machine Learning, PMLR vol.32, no. 2, pp. 1863-1871.

[4] Khurma, R.A., Aljarah, I., Sharieh, A.A., & Mirjalili, S.M. (2019). EvoloPy-FS: An Open-Source Nature-Inspired Optimization Framework in Python for Feature Selection. Algorithms for Intelligent Systems.

[5] Ö. B. Dinler, C. B. Şahin, (2021) Prediction of phishing web sites with deep learning using WEKA environment, Avrupa Bilim ve Teknoloji Dergisi, vol. 24, pp. 35-41, 2021. doi:10.31590/ejosat.901465.

[6] Guha, R., Chatterjee, B., Khalid Hassan, S.K., Ahmed, S., Bhattacharyya, T., & Sarkar, R. (2021). Py_FS: A Python Package for Feature Selection Using Meta-Heuristic Optimization Algorithms. Computational Intelligence in Pattern Recognition.

[7] Abu Khurma, R., Aljarah, I., Sharieh, A.A., Abd Elaziz, M., Damaševičius, R., & Krilavičius, T. (2022). A Review of the Modification Strategies of the Nature Inspired Algorithms for Feature Selection Problem. Mathematics.

[8] Xue, B., Zhang, M., Browne, W.N., & Yao, X. (2016). A Survey on Evolutionary Computation Approaches to Feature Selection. IEEE Transactions on Evolutionary Computation, 20, 606-626.

[9] A. Ulah, C. B. Şahin, O. B. Dinler, M. H. Khan, (2021) Heart Dısease Predıctıon Usıng Varıous Machınes Learnıng Approach, Journal of Cardiovascular Disease Research, vol. 12,no.3,pp.379-391. doi:10.31838/jcdr.2021.12.03.58.

[10] Li Z., et al., (2019). VulDeePecker: A Deep Learning-Based System for Vulnerability Detection, Cryptography and Security, Doi: 10.14722/ndss.2018.23158.

[11] Singh S.K., Chaturvedi A., (2020). Applying Deep Learning for Discovery and Analysis of Software Vulnerabilities: A Brief Survey, Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, vol. 1154. Springer, Singapore. https://doi.org/10.1007/978-981-15-4032-5_59.

[12] Batur Dinler, Ö. , Batur Şahin, C. & Abualigah, L. (2021). Comparison of Performance of Phishing Web Sites with Different DeepLearning4J Models . Avrupa Bilim ve Teknoloji Dergisi , Ejosat Special Issue 2021 (ICAENS) , pp. 425-431 . DOI: 10.31590/ejosat.1004778.

[13] Al-Tashi, Q., Abdulkadir, S.J., Rais, H.M., Mirjalili, S.M., & Alhussian, H. (2020). Approaches to Multi-Objective Feature Selection: A Systematic Literature Review. IEEE Access, 8, 125076-125096.

[14] Şahín C. B., and Dírí B., (2019). Robust Feature Selection with LSTM Recurrent Neural Networks for Artificial Immune Recognition System, in IEEE Access, vol. 7, pp. 24165-24178, doi: 10.1109/ACCESS.2019.2900118.

[15] Batur Şahin C., Batur Dinler Ö., Abuagilah L. (2021). Prediction of software vulnerability-based deep symbiotic genetic algorithms: Phenotyping of dominant-features, Applied Intelligence, doi: 10.1007/s10489-021-02324-3.

[16] https://nvd.nist.gov/

[17] https://cve.mitre.org/

[18] Kihel, B.K., & Chouraqui, S. (2019). Firefly Optimization Using Artificial Immune System for Feature Subset Selection. International Journal of Intelligent Engineering and Systems.

[19] Dökeroglu, T., Deniz, A., & Kiziloz, H.E. (2022). A comprehensive survey on recent metaheuristics for feature selection. Neurocomputing, 494, 269-296.

[20] Liu, W., & Wang, J. (2019). A Brief Survey on Nature-Inspired Metaheuristics for Feature Selection in Classification in this Decade. 2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC), 424-429.

[21] Dökeroglu, T., Sevinç, E., Kucukyilmaz, T., & Cosar, A. (2019). A survey on new generation metaheuristic algorithms. Comput. Ind. Eng., 137.

[22] Houssein, E.H., Mahdy, M.A., Shebl, D., & Mohamed, W.M. (2021). A Survey of Metaheuristic Algorithms for Solving Optimization Problems. Metaheuristics in Machine Learning: Theory and Applications.

[23] Şahin C. B., (2021). DCW-RNN: Improving Class Level Metrics for Software Vulnerability Detection Using Artificial Immune System with Clock-Work Recurrent Neural Network, International Conference on INnovations in Intelligent SysTems and Applications 2021 (INISTA'21), pp. 1-8, doi: 10.1109/INISTA52262.2021.9548609.

[24] Meidani, K., Mirjalili, S., & Farimani, A.B. (2022). Online metaheuristic algorithm selection. Expert Syst. Appl., 201, 117058.

[25] Dinler Ö. B., Aydın N., (2020). An optimal feature parameter set based on gated recurrent unit recurrent neural networks for speech segment detection," Applied Sciences, vol. 10, no. 4, pp. 1273, 2020. doi:10.3390/app10041273.

[26] Hailat, M.M., Otair, M.A., Abualigah, L., Houssein, E.H., Şahin, C.B. (2021). Improving Automated Arabic Essay Questions Grading Based on Microsoft Word Dictionary. In: Kadyan, V., Singh, A., Mittal, M., Abualigah, L. (eds) Deep Learning Approaches for Spoken and Natural Language Processing. Signals and Communication Technology. Springer, Cham. https://doi.org/10.1007/978-3-030-79778-2_2

[27] Mirjalili, S. (Ed.). (2022). Handbook of Moth-Flame Optimization Algorithm: Variants, Hybrids, Improvements, and Applications (1st ed.). CRC Press. https://doi.org/10.1201/9781003205326.

[28] Laith Abualigah (2022). The Arithmetic Optimization Algorithm(AOA)https://www.mathworks.com/matlabcentral/fileexchange/84742-the-arithmetic-optimization-algorithm-aoa), MATLAB Central File Exchange. Retrieved July 27, 2022.

[29] Şahin C. B., Dinler Ö. B., & Abualigah, L. (2021). Analysis of Risk Factors in the Scope of Distributed Software Team Structure . Avrupa Bilim ve Teknoloji Dergisi , Ejosat Special Issue 2021 (ICAENS) , 417-424 . DOI: 10.31590/ejosat.1004765.

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

[31] Cihan, P. & Kalıpsız, O. (2016). Öğrenci Proje Anketlerini Sınıflandırmada En İyi Algoritmanın Belirlenmesi . Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi, vol.8, no.1, pp.41-49. Retrieved from https://dergipark.org.tr/en/pub/tbbmd/issue/22248/238831