Сучасний стан наукових досліджень та технологій в промисловості. 2022. № 1 (19) *ISSN 2522-9818 (print)* *ISSN 2524-2296 (online)*

UDC 004.49 DOI: https://doi.org/10.30837/ITSSI.2022.19.055

W. CAO, V. KOSENKO, S. SEMENOV

STUDY OF THE EFFICIENCY OF THE SOFTWARE SECURITY IMPROVING METHOD AND SUBSTANTIATION OF PRACTICAL RECOMMENDATIONS FOR ITS USE

The **subject** of research in the article is a way for evaluating the effectiveness of the software security improving method. The **aim** of the article – study of the effectiveness of the software security improving method and substantiation of practical recommendations for its use. **Tasks** to be solved: analysis of methods for describing the software security testing process and evaluating its effectiveness, developing a scheme and method for evaluating the effectiveness of a method for improving software security, developing a simulation model for the software security testing process, studying the effectiveness of a method for improving software security, researching and substantiating the reliability of the results obtained, developing practical recommendations for using the method. Applied **methods**: system analysis, project approach, heuristic methods of decision making, process models. The **results** obtained: The analysis of the features of the ways for describing the software security testing process and evaluating its effectiveness showed the possibility of taking into account many factors by using the method of dynamics of averages. A way for evaluating the effectiveness of a method for improving software security has been developed, which differs from the known ones by taking into account the scaling factor of the software development process by introducing security testing specialists. With the help of an improved method, the hypothesis of increasing the efficiency of the security process using the developed method by reducing the relative damage indicator at all stages of the software life cycle, depending on the possible duration of a cyber-intrusion, was proved. The substantiation of the reliability of the results of mathematical modeling has been carried out. A number of practical recommendations on the use of the method of improving software security are given and some shortcomings are highlighted, which allow the conclusion that further research is possible.

Keywords: software safety; efficiency evaluation; reliability of mathematical modeling results; practical recommendations.

Formulation of the problem.

The conducted research of the security testing process [1], as well as the synthesis of the method of automated penetration testing [2], show the impossibility of complete processing of data on SW vulnerabilities and assessing them with 100% accuracy. Therefore, methods for approximate evaluation of the effectiveness of existing testing approaches based on modern approaches to mathematical formalization have become widespread [3, 4]. However, their use also implies the adaptation of the main provisions to changes in the SW testing process, and the reduction of input data uncertainty factors.

Literature analysis.

One of the possible ways to describe the process of SW security testing is the method of dynamics of averages. The advantages of this method are simplicity, the ability to take into account many factors (the availability of active and passive testing tools, the capabilities of penetration testing specialists and DevSecOps, etc.), the availability of analytical solutions [5-9]. Improving the method for evaluating the effectiveness of the developed method for improving SW security based on the method of dynamics of averages is based on the following assumptions. According to the law of large numbers, data on possible SW security threats, as well as means of countering them, are close to average (mathematical expectations), which makes it possible not to consider the details associated with the random state of a single element or SW function, and consider the SW security process as deterministic [5]. With this assumption, all test indicators will also not be random variables - they will be replaced by the corresponding mathematical expectations. The sequence of cyberattacks is represented as a Poisson flow of events [6, 7]. A technique is also used, which consists in the transition from the stream of cyberattacks to the stream of cyberattacks that have achieved results, which is also considered to be Poisson. A cyber attack is called successful if it implements an existing threat. Another possible way to formalize the SW security process is to use the theory of continuous Markov processes [8]. A process occurring in a system is called Markovian if, for each moment of time, the probability of any state of the system in the future depends only on its state at the present moment and does not depend on how the system came to this state. The flow of cyberattacks and the flow of cyberattacks that achieve results are also considered to be Poisson.

Method for evaluating the effectiveness of the developed method for improving SW security

The block diagram of the method for evaluating the effectiveness of the developed method for improving SW security and the labeled graph of system states is shown in fig. 1.

The method for evaluating the effectiveness of the developed method includes the following steps:

1. Analysis of possible SW threats. Meaningful statement of the research problem.

2. Development and synthesis of the main components of the assessed system "SW - Hacker" into a block diagram.

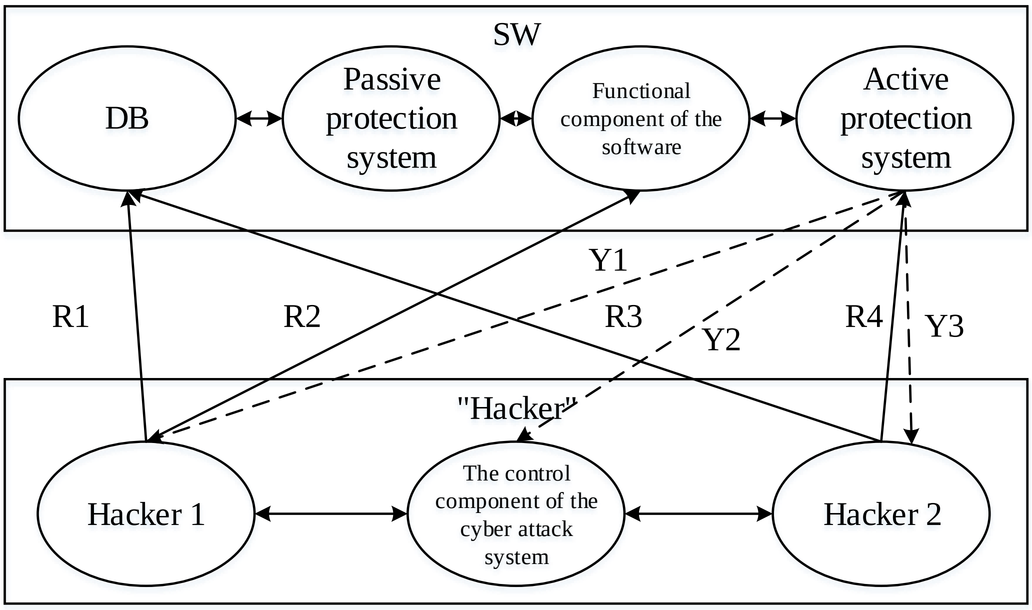
3. Mathematical formalization of the process of finding the numbers of states in differential form in accordance with the method of average dynamics.

4. Formalization of the input data, as well as additional conditions for solving the problem.

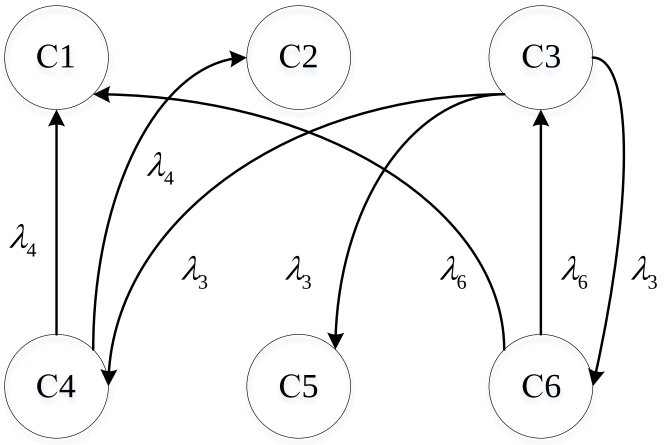
5. Solution of a formalized problem.

6. Fixation and approximation of simulation results.

7. Substitution of results and calculation of performance indicators of the developed method for improving SW security.


SW
DB
Passive protection system
Functional component of the software
Active protection system
Y1
R1
R2
R3
Y2
R4
Y3
"Hacker"
Hacker 1
The control component of the cyber attack system
Hacker 2

a)



b)

Fig. 1. Structural diagram of the method for evaluating the effectiveness of the developed method for improving SW security and a labeled system state graph

The conducted studies and analysis of a number of scientific sources [10, 11] have shown that the effectiveness of the developed method can be quantitatively determined using the loss index ΔQ_i for each SW element or function q_i . This indicator determines the amount of relative damage caused to the tool SW q_i element as a result of cyber attacks on the SW. At the same time, in accordance with scientific works [9, 10], the indicator of relative damage (losses) can be calculated by the formula:

$$\Delta Q_i\left(t^*\right)=\frac{q_i\left(t_0\right)-q_i\left(t^*\right)}{q_i\left(t_0\right)}\cdot 100\%,\ i=1\ldots n, \qquad (1)$$

where t^* is the current moment of time (the moment of the end of the formalization of the process);

$\Delta Q_i\left(t^*\right)$ – relative damage (losses) for an SW element or function q_i at a point in time t^*;

$q_i\left(t_0\right)$ – initial potential of the facility q_i at a point in time t_0;

n – the number of phase coordinates (the dimension of the vector q) of the simulated system "SW - Hacker".

8. Analysis and generalization of the simulation results and preparation of practical recommendations on the use of the SW security enhancement method and the strategy for its use in firms.

The program developed in the Phyton environment made it possible to conduct a series of experiments for given conditions. At the same time, the implemented simulation model of the interaction states of the "SW - Hacker" system made it possible to evaluate the performance indicators of the developed method for improving SW security.

In this model, in order to increase the reliability of the results obtained, the factors and possibilities of active (with the help of DevSecOps, SecDev, etc.) protection and passive protection (using preventive methods of secure programming and testing to detect and localize unsafe elements and SW functions).

For a formalized representation of the dynamic system "SW - Hacker", we introduce variables q_i, $i=1\ldots6$, by which we mean, respectively, the number of states of the following elements: SW installation components, SW database and active protection means, active attack means of the first type, control system and active attack means of the second type of cyber attacks.

The state of the dynamic system "SW - Hacker" as a whole at each moment of time $t\in[t_0,t_N]$ is characterized by a system of ordinary linear differential equations (2), in which the numbers of states q_i, $i=1\ldots6$ are considered as variables:

$$\begin{cases} \frac{dq_1}{dt}=\bar{q}_1(t)=-\lambda_4p_{14}R_1q_4-\lambda_6p_{16}R_3q_6+\delta q_1\gamma_1(q_1,Q_1^*); \\ \frac{dq_2}{dt}=\bar{q}_2(t)=-\lambda_4p_{24}R_2q_4+\delta q_2\gamma_2(q_2,Q_2^*); \\ \frac{dq_3}{dt}=\bar{q}_3(t)=-\lambda_6p_{36}R_4q_6; \\ \frac{dq_4}{dt}=\bar{q}_4(t)=-\lambda_3p_{43}Y_1q_3; \\ \frac{dq_5}{dt}=\bar{q}_5(t)=-\lambda_3p_{53}Y_2q_3; \\ \frac{dq_6}{dt}=\bar{q}_6(t)=-\lambda_3p_{63}Y_3q_3. \end{cases} \quad (2.)$$

Here $\delta q_1\gamma_1(q_1,Q_1^*)$ – mathematically formalized representation of the input of additional resources (cybersecurity specialists) to increase cyber protection q_1;

$\delta q_2\gamma_2(q_2,Q_2^*)$ – mathematically formalized representation of the input of additional resources (cybersecurity specialists) to increase cyber protection q_2;

Q_1^* and Q_2^* – boundary values of the numbers of states q_1 and q_2, accordingly, starting from which a reserve is introduced from the passive protection system;

δq_1 and δq_2 – the intensity of the introduction of reserve funds into the composition q_1 and q_2;

$\gamma_1(q_1,Q_1^*)$ and $\gamma_2(q_2,Q_2^*)$ – signal functions determined by the formulas:

$$\gamma_1(q_1,Q_1^*)=\begin{cases}1, & if\ q_1(t)\le Q_1^*; \\ 0, & if\ q_1(t)>Q_1^*.\end{cases}$$

$$\gamma_2(q_2,Q_2^*)=\begin{cases}1, & if\ q_2(t)\le Q_2^*; \\ 0, & if\ q_2(t)>Q_2^*.\end{cases}$$

The rule for solving differential equations allows you to introduce variables W_k, $k=1\ldots8$ to denote the coefficients of differential equations:

$$W_1=-\lambda_4p_{14}R_1;$$

$$W_2=-\lambda_6p_{16}R_3;$$

$$W_3=0;$$

$$W_4=-\lambda_4p_{24}R_2;$$

$$W_5=-\lambda_6p_{36}R_4;$$

$$W_6=-\lambda_3p_{43}Y_1;$$

$$W_7=-\lambda_3p_{53}Y_2;$$

$$W_8=-\lambda_3p_{63}Y_3,$$

where λ_i, $i=1\ldots6$ – the intensity of the attacks carried out by means of q_i, $i=1\ldots6$ respectively;

$p_{i,j}$ – the probability of penetration (hacking) into the SW or its separate component q_i (for example, a database) as a result of an attack by a mean q_j;

Y_1, Y_2 and Y_3 – coefficients characterizing the level of quality work of SW security testers;

R_1, R_2 and R_3 – coefficients characterizing the degree of training of hackers who overcome SW protection.

It should be noted that the general condition: $Y_1+Y_2+Y_3\le1$.

After transformations of the system of equations (2), we obtain:

$$\begin{cases} \bar{Q}_1(t)=W_1q_4+W_2q_6+\delta q_1\gamma_1(q_1,Q_1^*); \\ \bar{Q}_2(t)=W_3q_6+\delta q_2\gamma_2(q_2,Q_2^*); \\ Q_3(t)=W_4q_6; \\ Q_4(t)=W_5q_3; \\ \bar{Q}_5(t)=W_6q_3; \\ \bar{Q}_6(t)=W_7q_3, \end{cases} \quad (3)$$

where $\gamma_k(t)$ is a function that characterizes the possibility of choosing the interval for introducing a reserve Δt_{add} (cybersecurity specialists) to reinforce the security of SW elements and functions $q_1(t)$ and $q_2(t)$.

The ratios $Y_1/Y_2/Y_3=X_1/X_2/X_3$ determine the strategy of software developers in counteracting cyberattacks of the "Hacker". The ratios $R_1/R_2/R_3=X_4/X_5/X_6$ determine the strategy of the "Hacker" in the organization of breaking SW.

R_1 – share (%) of the “Hacker” funds q_4 involved in a cyber attack on SW elements and functions. It is necessary to note the obligatory fulfillment of the condition: $R_1\le1$. R_2 – the share (%) of the "Hacker" funds q_4 involved in a cyber attack on the elements and functions q_2 of the SW (the main functional component). Also R_3, R_4 are the shares (in %) of the means q_6 of the "Hacker" side participating in the information suppression of the means q_1 and q_3 from the side of the SW developer, respectively. The sums $R_1 + R_2$ and $R_3 + R_4$ must be less than or equal to 1. p_{ij}, $i, j = \overline{1,6}$ is the probability of failure of the *i*-th type device by the *j*-th type device.

Modeling and assessment of relative damage.

When modeling, it is necessary to introduce a number of additional conditions:

1. Replenishment of elements in the state C_1 (adding the active and passive phases of counteracting cyberattacks) is carried out at $q_1(t) \le 80\%\, q_1(t_0)$. Replenishment of elements in the state C_2 is performed when $q_2(t) \le 90\%\, q_2(t_0)$.

2. Each of the elements, functions or systems q_i, $i = \overline{3,6}$, has only two states: safe and dangerous.

3. The input data of the simulation model are as follows:

$$q_1(t_0) = q_2(t_0) = 100, \quad q_3(t_0) = q_5(t_0) = 100\%,$$
$$q_4(t_0) = q_6(t_0) = 50.$$

$$\lambda_3 = 0{,}12 hour^{-1}, \lambda_4 = 0{,}15 c.u^{-1}, \lambda_6 = 0{,}21 c.u^{-1},$$
$$\delta q_1 = 0{,}25 c.u^{-1}, \delta q_2 = 0{,}15 c.u.^{-1}.$$

4. Taking into account the conducted research and expert assessments of specialists from SW development firms, fixed strategies $\{Y\}$ and $\{R\}$ aspects of the "SW - Hacker" system were adopted:

$$Y_1 = Y_2 = 0{,}3; \quad Y_3 = 0{,}25; \quad R_1 = 0{,}5; \quad R_2 = 0{,}25;$$
$$R_3 = 0{,}35; \quad R_4 = 0{,}4.$$

The values of the probabilities of occurrence of a security error SW or successful countermeasures against the "Hacker" will be presented in accordance with table 1.

Table 1. *Values of the probability of occurrence of a security error SW or successful countermeasures against the "Hacker"*

SW				**"Hacker"**		
p_{14}	p_{16}	p_{26}	p_{36}	p_{43}	p_{53}	p_{63}
0,1	0,08	0,09	0,03	0,08	0,125	0,06

5. Numerical solution of differential equations (2) is performed in a cycle on the time interval from 0 to 180 c.u., with a step of 0.1 c.u.

6. The limitations of the simulation, in which it is impossible to continue the given program, are the cases when: $q_1 \le 1$; $q_2 \le 1$; $q_3 \le 10\%$; $q_4 \le 1$; $q_5 \le 10\%$; $q_6 \le 1$.

Calculate the relative damage $\Delta Q_i(t^*)$ for all q_i where *i*=1,…,6.

The calculation results are presented in table 2.

Table 2. *Relative damage* $\Delta Q_i(t^*)$ *for all* q_i *for* t^* *=120 c.u.*

SW			**Hacker**		
$\Delta Q_1(t^*)$	$\Delta Q_2(t^*)$	$\Delta Q_3(t^*)$	$\Delta Q_4(t^*)$	$\Delta Q_5(t^*)$	$\Delta Q_6(t^*)$
29,57	18,31	11,99	64,57	50,45	40,36

In the considered example, the results of the relative damage assessment indicate the feasibility and effectiveness of the security testing system implemented in the model and, at the same time, the possibility of neutralizing the means of the attacking side (using active and passive defense methods).

The effectiveness of the developed SW safety improvement method can be assessed using the curves of the graphs in fig. 2.

As can be seen from the graphs, the use of the developed SW security enhancement method, taking into account the capabilities of automated penetration testing using deep machine learning technology, reduces the relative damage at all stages of the SW life cycle by up to 6 times, depending on the possible duration of the attack.

Justification of the reliability of the results of mathematical modeling.

To substantiate the reliability of the results obtained in sections 2 and 3, a number of experiments were carried out, in accordance with the conditions:

- the SW development team consists of 7 people, including one DevSecOps and one security tester;
- the main SW development management methodology is SCRUM;
- sprints are divided into weeks and rallies are held daily;
- number of experiments N*=500.

Based on the results of the experiment, a histogram of the frequency of correct detection of an attack with a higher probability [12] was obtained, which is shown in fig. 3.

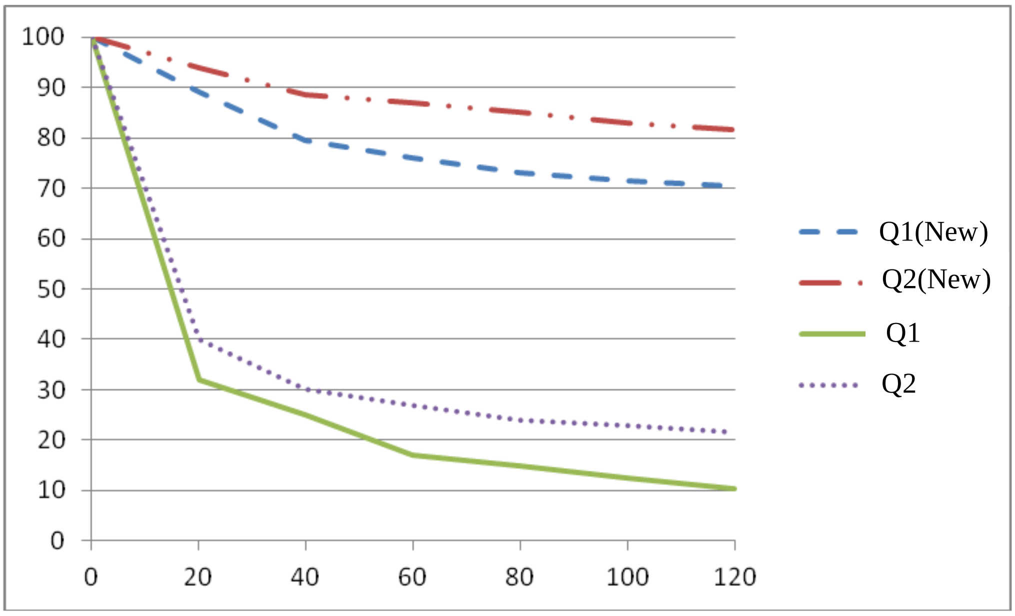


Fig. 2. Graphs of the dynamics of changes $\Delta Q_i\left(t^*\right)$ for systems with DevSecOps and a security tester, and also without them

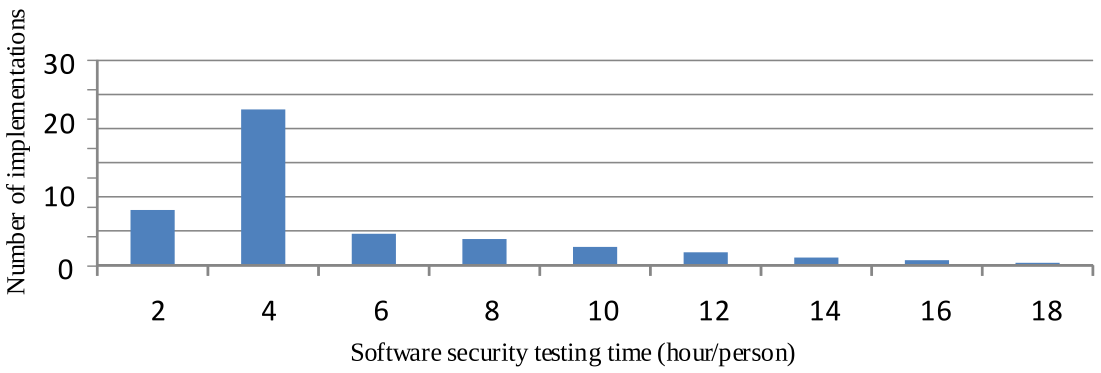


Fig. 3. Histogram of SW security testing time

The hypothesis put forward about the normal distribution of this random variable was tested by Pearson's goodness-of-fit test χ^2 [11]

$$\chi^2 = N^*\sum_{i=1}^{k}(P_i^* - P_i)^2 / P_i \, ,$$

where k is the number of digits (intervals) of the statistical series;

P_i^* and P_i are "statistical" and theoretical probabilities of "hitting" a given indicator in the i-th category.

The conducted verification proved the plausibility of the hypothesis that the value of the frequency of correct detection of an attack with a higher probability of occurrence is distributed according to the normal law.

Estimates $P_{\text{test}}^{(i)}$ of the mathematical expectation $P_{\text{test}}^{(i)}$ and $\hat{D}_{P_{\text{test}}^{(i)}}$ variance ($\hat{\sigma}_{P_{\text{test}}^{(i)}}$ standard deviation) are obtained, a random variable $P_{\text{test}}^{(i)}$ that characterizes the frequency of correct detection of an attack with a higher probability of occurring:

$$P_{\text{test}}^{(i)} = \frac{\sum_{j=1}^{k} P_{\text{test}}^{(i,j)}}{N^*}; \; \hat{D}_{P_{\text{test}}^{(i)}} = \frac{\sum_{j=1}^{k}\left(P_{\text{test}}^{(i)} - P_{\text{test}}^{(i,j)}\right)^2}{N^*\text{-}1};$$

$$\hat{\sigma}_{P_{\text{test}}^{(i)}} = \sqrt{\hat{D}_{P_{\text{test}}^{(i)}}} \, .$$

Using the well-known expression for calculating the confidence probability of the deviation of the relative frequency from the constant probability in independent tests [13], we determine the confidence probability that the value of the characteristic of the frequency of correct attack detection obtained as a result of the experiment with a higher probability of occurrence "does not deviate" from the mathematical expectation $P_{\text{test}}^{(i)}$ by more than 0.05:

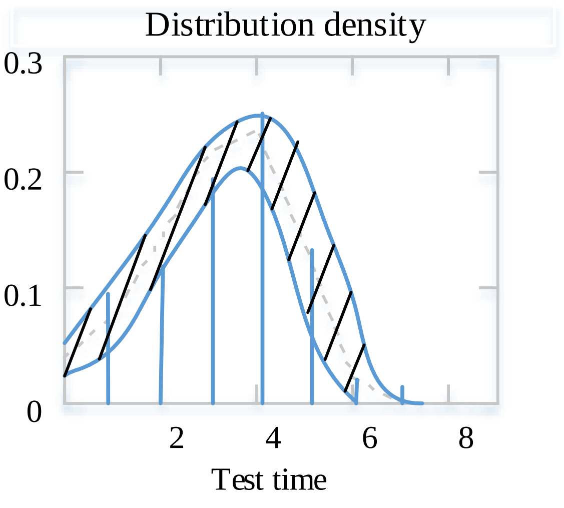
$$P\left(\left|\hat{P}_{test}^{(i)} - P_{test}^{(i)}\right| < 0.05\right) = 2F\left(0.05 \big/ \hat{\sigma}_{P_{test}^{(i)}}\right),$$

where F is a Laplace function of the form $F(x) = \frac{1}{\sqrt{2\pi}}\int_0^x e^{-t^2/2}dt$ [__].

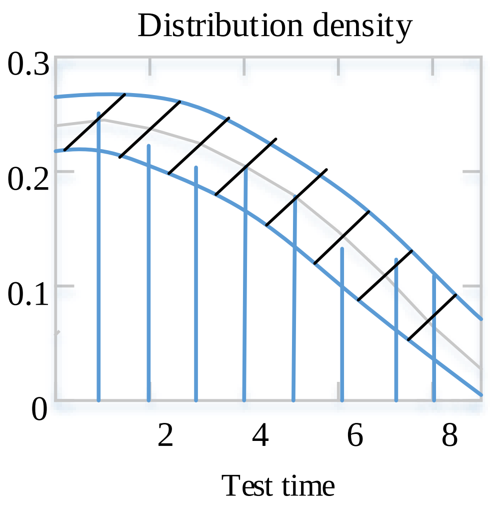
Also, the results of the experiments showed that for all the studied types of data the confidence probability is that the values of the statistical quantity $P_{test}^{(i)}$ will not deviate from the mathematical expectation $\hat{P}_{test}^{(i)}$ by more than 0.05 and is equal to: $P \approx 0.94$.A comparative study of the results of mathematical modeling and experiment has been carried out. The results of the comparison are shown in fig. 4. in the form of a graph of the probability distribution density of the implementation time of penetration testing algorithms, and the boundaries of the confidence interval corresponding to them:

$$I_\beta = \left[\hat{\bar{J}} - \varepsilon_\beta, \hat{\bar{J}} + \varepsilon_\beta\right],$$

in which the true value $\bar{J}$ falls with a confidence probability $\beta = 0,94$ and estimates of its $\hat{t}_{\text{test}}{}^{(i)}$ mathematical expectation.



a)



b)

Fig. 4. Graph of the probability distribution density of the implementation time of penetration testing algorithms, the boundaries of the confidence interval corresponding to them, and estimates of its $\hat{t}_{\text{test}}{}^{(i)}$ of mathematical expectation

It can be seen from the graphs that in the key test situation (time $t_{\text{test}} \approx 4c.u.$) the "calculated" curve J (solid curve), obtained in accordance with the mathematical model developed in the work, in most practical cases fall into the "averaged" confidence interval (shaded area).

This confirms the reliability of the mathematical model and the analytical expression obtained as a result of mathematical modeling.

Substantiation of practical recommendations on the use of the software security improvement method.

The conducted studies have shown that when designing SW, it is necessary to take into account that the fulfillment of any of the requirements of quality indicators during design may affect other requirements. And in this case, it is necessary to analyze the ratio of benefits and losses for the totality of many quality indicators.

As noted in [13], security is the ability of a system to prevent malicious or accidental actions not provided for in the design, or to prevent disclosure or loss of data. Here we are talking about system security in a general sense. At the same time, improving the security of SW also leads to an increase in the reliability of the system as a whole by reducing the likelihood of successful attacks and their negative impact on the system.

The analysis of the literature [9, 13, 14] and the experience of coding and testing in the development of SW made it possible to identify a number of features of secure programming and implementation of security-related control functions:

- coordination of the SW security policy regarding the technical conditions of the object of implementation (implementation), the formation of restrictions on the use of the product in accordance with the security policy;
- signature-heuristic analysis of source and executable code for potentially dangerous operations and coding errors;
- analysis of security subsystems (password protection subsystem tracing), etc.;
- carrying out all types of manual and automatic testing (functional, stress, load testing and performance testing) taking into account increased security requirements;
- structural analysis of distribution redundancy and integrity control;
- analysis of the presence of covert channels;
- update and modification of SW in accordance with the approved security policy.

A qualitative assessment of these factors and features of secure SW programming allowed us to conclude that an integrated approach to identifying vulnerabilities is expedient and to present common security-related errors of programmers, as well as methods for testing and identifying these errors in the form of a table 3.

Table 3. *Code vulnerabilities and SW testing methods*

Programming mistakes	Test method	Note
Errors that occur when entering rarely used input data.	Functional testing of rarely used inputs.	Validation of the functionality declared in the document of rarely used input data using boundary or negative testing methods.
Undeclared input parameters and modes associated with an abnormal state of the system and a possible decrease in its performance.	Performance profiling.	Identification of code fragments in which anomalies are observed (unauthorized transfer, transition to an infinite loop, processing of undeclared input data, etc.) performance decrease.
Errors related to functional safety.	Load and stress testing.	Creation of specific conditions for the program to work with a large amount of input data, increased load on the processor, etc.
Vulnerabilities of the password protection subsystem and other subsystems.	Testing of the authentication subsystem, access control and other information security subsystems.	Defining security management modes, tracing the password system, searching for built-in passwords, etc.
Unintentional (intentional) software bookmarks, bombs.	Signature-heuristic analysis.	Identification of software bookmarks, bombs by signatures of potentially dangerous operations or by read event data.
Incorrect coding.	Signature-heuristic analysis.	Identification of software bookmarks by signatures.
Intentional tabs of a "hooligan" nature, "programmer signatures", "Easter eggs".	Signature-heuristic analysis.	Identification of program bookmarks on the basis of illegitimate identifiers of programmers (messages, constants, hidden "hot keys", etc.).
Hidden channels.	Analysis of traffic and memory in an isolated environment.	Monitoring and auditing logs.
Program bookmarks initiated by a hidden transition.	Redundancy control, signature-heuristic analysis.	Identification of software bookmarks on the basis of hidden data transfer.

Conducted studies have shown that the vast majority of software is based on previously developed software components used for work. For example, programs written in C or C++ depend on the runtime libraries that come with the compiler or operating system. Programs written in Python also include a number of standard libraries, components, or other off-the-shelf software solutions. At the same time, the analysis of the literature [12, 13] showed that there are a number of hidden (undeclared) threats.

One of the undeclared threats to using off-the-shelf software solutions is that even if you write flawless code, your application may be vulnerable due to security flaws in one of the components used. For example, the realpath () function of the C programming language library returns the canonicalized absolute path for the path given as an argument. To get the canonicalized absolute path, the function expands all symbolic links. However, some implementations of realpath () contain a static buffer that overflows when the canonicalized path is longer than MAXPATHLEN. Other common C library functions for which some implementations are known to be prone to buffer overflows are syslog (), getpass (), and the getopt () family of calls.

Since many of these problems have been known for some time, many C libraries now contain fixed versions of these functions. For example, the implementations of libc4 and libcS for Linux contain a buffer overflow vulnerability in realpath (), but the problem is fixed in libc-5.4.13. On the one hand, this creates the prerequisites for eliminating threats and correcting the defect, but, as studies have shown, this problem has not been fully resolved.

Modern operating systems are known to support dynamic link libraries, or shared libraries. In this case, the library code is not statically linked to the executable file, but is searched for in the environment where the program is installed. Therefore, if our development application designed to work with libc-5.4.13 is installed in an environment that has an older version of libc5 installed, the program will be subject to a buffer overflow error in the realpath () function.

One solution is to statically link the safe libraries with your application. This approach allows you to capture the implementation of the library you are using. However, this approach has the disadvantage of creating a large executable image on disk and in memory. It also means that your application will not be able to take advantage of new libraries that may fix previously unknown flaws (both security flaws and others). Another solution is to make sure that the input values passed to an external function stay within limits that are known to be safe for all existing implementations of that function.

It should be noted that similar problems can arise with distributed object systems such as DCOM, CORBA, and other compositional models that are linked at run time.

System integrators and administrators can protect systems from vulnerabilities in off-the-shelf software components (such as libraries) by providing wrappers that intercept API calls that are known to misbehave. Such a wrapper implements the original API functions (usually by calling the original component), but performs additional checks to ensure that known vulnerabilities cannot be exploited by an attacker. To be realistic, this approach requires linking executables at runtime. An example of an approach that implements this method for Linux systems is the libsafe library from Avaya Labs [13].

Shells do not require changes to the operating system and work with existing binaries. They fail to protect against unknown security flaws: if there is a vulnerability in the part of the code that is not caught by the shell, the system will still be vulnerable to attacks.

Related to the approach described is the execution of untrusted programs in a controlled environment that is restricted to specific behavior using custom policies. An example of this approach is Systrace. This approach differs from a secure shell in that it does not prevent the exploitation of vulnerabilities, but it can prevent unexpected secondary actions that exploit authors usually try to perform, such as writing files to a secure location or opening network sockets [14].

Systrace is a policy enforcement tool that provides a way to monitor, intercept, and restrict system calls. Systrace acts as a wrapper for executables, directing them to bypass the system call table. It intercepts system calls and, using the Systrace device, passes them through the kernel and processes system calls [14].

Like a secure shell, a controlled environment does not require any source code or changes to the program being controlled. The disadvantage of this approach is that it is very easy to misformulate policies and break the required functionality of the controlled program. Research has shown that it can be a difficult (and often impossible) task for an administrator to create accurate policy descriptions for large and complex programs whose behavior is not fully understood. Accordingly, the solution of the above problems can become a plan for further research.

Among the main security threats and bugs in Python are the following:

- Command injection (command injection).
- SQL injection.
- XML parsing.
- assert statements.
- Time attacks.
- Polluted site-packages or import path.
- Temporary files.
- Using yaml. load.
- Deserialization of canned data.
- Using the Python runtime system.
- Not installing patches for your dependencies.

Detailed information about each of the listed vulnerabilities is presented in the article [15].

Conclusions

The section contains comparative studies and evaluation of the effectiveness of the method for improving software security. For this purpose, the method for evaluating efficiency has been improved in the work. Its distinguishing feature is taking into account the possibility of scaling the software development process by introducing security testing specialists (DevSecOps, SecDev, as well as penetration testers).

The improved method for evaluating the effectiveness of the software security improvement method is based on the method of dynamics of averages.

With the help of an improved method, the expediency of using the developed method for improving software security is proved, taking into account the capabilities of deep reinforcement learning technology. This will reduce the relative damage at all stages of the SW life cycle by up to 6 times, depending on the possible duration of a cyber-intrusion.

The section substantiates the reliability of the results of mathematical modeling. The results of the experiments showed that for all types of data under study, the confidence probability that the value of the statistical value $P_{test}^{(i)}$ will not deviate from the mathematical expectation $\hat{P}_{test}^{(i)}$ by more than 0.05 and is equal to: $P \approx 0.94$.

The section also provides a number of practical recommendations on the use of the software security enhancement method and highlights some shortcomings, which led to the conclusion that further research is possible.

References

1. Semenov, S., Weilin, C. (2020), "Testing process for penetration into computer systems mathematical model modification", *Advanced Information Systems*, Vol. 4, Issue 3, P. 133–138. DOI: https://doi.org/10.20998/2522-9052.2020.3.19
2. Semenov, S., Weilin, C., Zhang, L., & Bulba, S. (2021), "Automated penetration testing method using Deep machine learning technology", *Advanced Information Systems*, Vol. 5, Issue 3, P. 119–127. DOI: https://doi.org/10.20998/2522-9052.2021.3.16
3. Farchi, E., Hartman, A., Pinter, S. (2002), "Using a model-based test generator to test for standard conformance", *IBM Systems Journal*, Vol. 41, Issue 1, P. 89–110. DOI: https://doi.org/10.1147/sj.411.0089

4. Ali H. Doğru, Veli Biçer (2010), "Modern Software Engineering Concepts and Practices: Advanced Approaches", *IGI Global*, P. 506.
5. Shanahan, L., Sen, S. (2011), "Dynamics of stochastic and nearly stochastic two-party competitions", *Physica A: Statistical Mechanics and its Applications*, Vol. 390, Issue 10, P. 1800–1810. DOI: https://doi.org/10.1016/j.physa.2010.12.041
6. Tze Leung Lai, Haipeng Xing (2008), *Statistical Models and Methods for Financial Markets*, Springer New York Softcover reprint of hardcover 1st ed., 356 p.
7. Stephen Boyd, Lieven Vandenberghe (2018), "Introduction to Applied Linear Algebra Vectors, Matrices, and Least Squares", *Cambridge University Press*. DOI: https://doi.org/10.1017/9781108583664
8. Swart., J., Winter, A. (2010), "Markov processes: theory and examples", available at: // https://www.uni-due.de/~hm0110/Markovprocesses/sw20.pdf
9. Kosenko, Nataliia & Kadykova, Iryna & Artiukh, Roman. (2017), "Formalizing the problem of a project team bulding based on the utility theory", *Innovative technologies and scientific solutions for industries,* P. 53–57. DOI: https://doi.org/10.30837/2522-9818.2017.1.053
10. Khalife, Kassem, Krikhovetskiy H.Ya., і H.A. Kuchuk. (2017), "Evaluation of the system software security"["Ocinka vrazlivosti sistemnogo programnogo zabezpechennya"], *Management systems, navigation and communication. Collection of scientific*, 6 (46), Poltava: PNTU, P. 141–44.
11. Semenov S. G., Khalife Kassem, Zakharchenko M. M. (2017), "An improved way to scale agile software development", ["Usovershenstvovannyj sposob masshtabirovaniya gibkoj metodologii razrabotki programmnogo obespecheniya"], *Bulletin of NTU "KhPI"*, Kharkiv, Vol. 1, No. 1, P. 79– 84. DOI: https://doi.org/10.20998/2522-9052.2017.1.15
12. Gmurman V.E. (2003), *Theory of Probability and Mathematical Statistics,* [*Teoriya veroyatnostej i matematicheskaya statistika*], M., Higher school,479 p.
13. J.D. Meier, David Hill, Alex Homer, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Boucher Jr., Akshay Bogawat. (2009),"Microsoft's Guide to Application Architecture Design", available at: // http://ce.sharif.edu/courses/91-92/1/ce474-2/resources/root/App%20Arch%20Guide%202.0.pdf
14. Robert Seacord (2013), "Secure Coding in C and C++ Addison-Wesley Professional", P. 600.
15. Anthony Shaw (2018), "10 common security gotchas in Python and how to avoid them", *Hakernoon,* available at: // https://hackernoon.com/10-common-security-gotchas-in-python-and-how-to-avoid-them-e19fbe265e03

Received 14.01.2022

Відомості про авторів / Сведения об авторах / About the Authors

Цао Вейлінь – науковий співробітник, викладач, департамент IT інформаційного центру, Нейцзянскій педагогічний університет; e-mail: caowl@njtc.edu.cn; ORCID: http://orcid.org/0000-0001-8230-5235.

Цао Вэйлинь – научный сотрудник, преподаватель, департамент ИТ информационного центра, Нейцзянский педагогический университет, Neijiang, Китай.

Cao Weiling – Intermediate grade of experimenter, teacher, Department of IT information Centre, Neijiang Normal University, Neijiang, China.

Косенко Віктор Васильович – доктор технічних наук, професор, Національний університет "Полтавська політехніка імені Юрія Кондратюка", професор кафедри автоматики, електроніки та телекомунікацій, Полтава, Україна; kosvict@gmail.com; ORCID ID – http://orcid.org/0000-0002-4905-8508.

Косенко Виктор Васильевич – доктор технических наук, профессор, Национальный университет "Полтавская политехника имени Юрия Кондратюка", профессор кафедры автоматики, электроники и телекоммуникаций, Полтава, Украина.

Viktor Kosenko – Doctor of Sciences (Engineering), Professor of Automation, Electronic and Telecommunication Department of National University «Yuri Kondratyuk Poltava Polytechnic, Poltava, Ukraine.

Семенов Сергій Геннадійович – доктор технічних наук, професор, професор кафедри кібербезпеки та інформаційних технологій ХНЕУ ім. С. Кузнеця, Харків, Україна; e-mail: s_semenov@ukr.net; ORCID: https://orcid.org/0000-0003-4472-9234.

Семенов Сергей Геннадьевич – доктор технических наук, профессор, профессор кафедры кибербезопасности и информационных технологий ХНЭУ им. С. Кузнеца, Харків, Україна.

Semenov Serhii – Doctor of Sciences (Engineering), Professor, Professor of the Department of cybersecurity and information technology, Simon Kuznets Kharkiv National University of Economics Kharkiv, Ukraine.

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕТОДУ ПІДВИЩЕННЯ БЕЗПЕКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ І ОБҐРУНТУВАННЯ ПРАКТИЧНИХ РЕКОМЕНДАЦІЙ З ЙОГО ВИКОРИСТАННЯ

Предметом дослідження у статті є спосіб оцінки ефективності методу підвищення безпеки програмного забезпечення. **Мета** статті – дослідження ефективності методу підвищення безпеки програмного забезпечення та обґрунтування практичних рекомендацій щодо його використання. **Завдання**, що вирішуються: аналіз способів опису процесу тестування безпеки програмного забезпечення та оцінки його ефективності, розробка схеми та способу оцінки ефективності методу підвищення безпеки програмного забезпечення, розробка імітаційної моделі процесу тестування безпеки програмного забезпечення, дослідження ефективності методу підвищення безпеки програмного забезпечення, дослідження та обґрунтування достовірності отриманих результатів, розробка практичних рекомендацій щодо використання методу. **Методи**, що застосовуються: системний аналіз, проектний підхід, евристичні методи прийняття рішень, процесні моделі. Одержані **результати**: проведений аналіз особливостей способів опису процесу тестування безпеки програмного забезпечення та оцінки його ефективності показав можливість урахування багатьох факторів шляхом використання методу динаміки середніх. Розроблено спосіб оцінки ефективності методу підвищення безпеки програмного забезпечення, що відрізняється від відомих обліком фактора масштабування процесу розробки програмного забезпечення шляхом впровадження спеціалістів тестування безпеки. За допомогою вдосконаленого способу доведено гіпотезу про підвищення ефективності процесу забезпечення безпеки за допомогою розробленого методу шляхом зниження показника відносної шкоди на всіх етапах життєвого циклу програмного забезпечення залежно від можливої тривалості кібервторгнення. Проведено обґрунтування достовірності результатів математичного моделювання. Наведено ряд практичних рекомендацій щодо використання методу підвищення безпеки програмного забезпечення та виділено деякі недоліки, що дозволило зробити висновок про можливість подальших досліджень.

Ключові слова: безпека програмного забезпечення; оцінка ефективності; достовірність результатів математичного моделювання; практичні рекомендації.

ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ МЕТОДА ПОВЫШЕНИЯ БЕЗОПАСНОСТИ ПРОГРАММНОГО ОБЕПЕЧЕНИЯ И ОБОСНОВАНИЕ ПРАКТИЧЕСКИХ РЕКОМЕНДАЦИЙ ПО ЕГО ИСПОЛЬЗОВАНИЮ

Предметом исследования в статье является способ оценки эффективности метода повышения безопасности программного обеспечения. **Цель** статьи – исследование эффективности метода повышения безопасности программного обеспечения и обоснование практических рекомендаций по его использованию. Решаемые **задачи**: анализ способов описания процесса тестирования безопасности программного обеспечения и оценки его эффективности, разработка схемы и способа оценки эффективности метода повышения безопасности программного обеспечения, разработка имитационной модели процесса тестирования безопасности программного обеспечения, исследование эффективности метода повышения безопасности программного обеспечения, исследование и обоснование достоверности полученных результатов, разработка практических рекомендаций по использованию метода. Применяемые **методы**: системный анализ, проектный подход, эвристические методы принятия решений, процессные модели. Полученные **результаты**: проведенный анализ особенностей способов описания процесса тестирования безопасности программного обеспечения и оценки его эффективности показал возможность учета многих факторов путем использования метода динамики средних. Разработан способ оценки эффективности метода повышения безопасности программного обеспечения, отличающийся от известных учетом фактора масштабирования процесса разработки программного обеспечения путем внедрения специалистов тестирования безопасности. С помощью усовершенствованного способа доказана гипотеза о повышении эффективности процесса обеспечения безопасности с помощью разработанного метода путем снижения показателя относительного ущерба на всех этапах жизненного цикла программного обеспечения, в зависимости от возможной продолжительности кибервторжения. Проведено обоснование достоверности результатов математического моделирования. Приведено ряд практических рекомендаций по использованию метода повышения безопасности программного обеспечения и выделены некоторые недостатки, что позволило сделать вывод о возможности дальнейших исследований.

Ключевые слова: безопасность программного обеспечения; оценка эффективности; достоверность результатов математического моделирования; практические рекомендации.

Бібліографічні описи / Bibliographic descriptions

Цао Вейлінь, Косенко В. В., Семенов С. Г. Дослідження ефективності методу підвищення безпеки програмного забезпечення і обґрунтування практичних рекомендацій з його використання. *Сучасний стан наукових досліджень та технологій в промисловості*. 2022. № 1 (19). С. 55–64. DOI: https://doi.org/10.30837/ITSSI.2022.19.055

Cao, Weiling, Kosenko, V., Semenov, S. (2022), "Study of the efficiency of the software security improving method and substantiation of practical recommendations for its use", *Innovative Technologies and Scientific Solutions for Industries*, No. 1 (19), P. 55–64. DOI: https://doi.org/10.30837/ITSSI.2022.19.055 г