Increasing the reliability of information management systems for complex objects

A Levchenkov^{1,*}

¹Don State Technical University, 1, Gagarin sq., Rostov-on-Don, 344003, Russia

Abstract. This article describes one of the possible approaches to the implementation of a set of mechanisms and tools for assessing and monitoring software safety based on static and dynamic analysis of programs, which provides an effective check of the operating and developed software. To solve this problem, it is proposed to use mathematical models of program circuits, modern software development technologies and methods of software analysis.

1 Research problem statement

The aggregate component of the reliability of modern information systems, especially those focused on the control function, includes both the reliability of the software and the reliability of the technical means for supporting the computing process [2,9]. Currently, there is a fairly extensive list of methods for maintaining and ensuring a given level of reliability of IS objects, especially the hardware component.

It is obvious that the reliable functioning of various information systems, including analytical, intellectual and expert IS, is based precisely on their software component. The reasons for this conclusion are, on the one hand, their complex software implementation and, on the other hand, the use of foreign development software that did not pass the certification process for the presence of software backlogs. All this dictates the need to control the quality parameters of software, especially for those information systems that provide vital functions for managing both people and complex technical equipment. Moreover, the fulfillment of this condition is equally relevant both for operating IS and for newly developed ones [3].

The importance of such a study is based on the direct dependence of the operability of the functioning of the IS on the qualitative indicators of the reliability and safety of the software used.

The current state of the art in solving this problem indicates that the "test and debug" methodology does not give the desired result, because the health check procedure cannot detect a "hole" in software security vulnerability.

Therefore, we can conclude that such a property of reliability as the operability of an information system has nothing to do with security.

The simplest and most reasonable way to check the security of software is to evaluate each element for the absence of vulnerabilities based on the brute force method. Existing

* Corresponding author: levkov53@mail.ru

security protocols and cryptographic encryption methods do not solve this problem due to the lack of the possibility of full coverage of all stages of IC design, such as development of specifications, design, implementation, development of program code, etc [10].

The lack of full-scale checks and tests of information systems manifests itself in the format of the occurrence of security errors quite often. One of the ways to eliminate this condition is the organization of the regulatory process in the implementation of the software quality assessment system. An attempt to solve it by controlling the absence of undeclared opportunities does not sufficiently reduce both the labor intensity of the process itself and its cost.

The relevance of research on this topic is due to the widespread use of unsafe software in critical systems and the need to develop new models and fairly effective methods for assessing and monitoring the safety of operated and implemented information systems in complex information-calculation and information-control software systems [3].

The existing methods for assessing and controlling software security have serious shortcomings that prevent their widespread use.

A method is needed that makes the most complete use of all information about the program being tested, based on a combination of static and dynamic analysis and supplemented with the mechanisms of expert systems and systems for formal proof of the correctness of programs. On the basis of such an integral method, it is planned to develop a set of automated tools that ensure the assessment and control of program security, if possible, without human intervention [6].

To develop an integral method, a *software vulnerability model* and a *model of information security* threats are needed, which would provide a convenient apparatus for analyzing programs, both in their text and in dynamics, and would allow the use of formal proof methods [4-5].

In order to develop a set of mechanisms and tools for assessing and controlling software security, it is necessary to build a model of information security threats arising from software vulnerabilities. To analyze software tools and identify vulnerabilities in them, it is required to develop a software model. These models should be developed within the general information technology model.

The basis for the development of these models should be the concept of modeling the information process based on its model representation. This approach should provide a formal apparatus for constructing static and dynamic methods for program analysis. In addition, we get the opportunity to implement the process of automatic correctness proof [7-8].

When developing the models, the following assumptions were made:

- the hardware architecture follows the von Neumann architecture;
- source codes of programs are written in procedural or object-oriented high-level languages.

2 Information process model

To build a model of the information process, it is necessary to present it in the form of a triple:

$$\langle AU, CU, MU \rangle,$$

where AU - is a computing device;

CU - control device;

MU - is a mass storage device.

The way to access this information is the names of the memory locations. We denote the set of names $A = \{a_i, i \in \mathbb{N}\}$. Obviously, each site has an identifier with a specific address.

Let us denote the sets of these information objects, respectively, C, D, N . Let us call the state of the corresponding set its subset available to the calculator at a given time. The state will be denoted by the letter q with the index of the corresponding set. For example, q^C, q^D, q^N .

The sets of all states will be denoted by Q^C, Q^D, Q^N , respectively. We denote all possible states of a collection of sets

$$C \cup D \cup N; Q = Q^C \times Q^D \times Q^N.$$

The set of all information objects available to the calculator at a particular time is denoted by

$$q = q^C \cup q^D \cup q^N.$$

The information process can be in various states. By a state we mean a triple of the form (q^D, L, q^N) ,

where $q^D \in Q^D$;

$q^N \in Q^N$;

$L \in Q^C \times A; L = \{(q^C, a_c)\}$.

Let us introduce a one-to-one mapping when each element of one set corresponds to only one element of another set. Such a mapping is called bijective $\text{elem} : Q \times A \rightarrow Q$.

In addition, we also introduce a binary relation of linear order $\prec$. Let us define a function $\text{next} : A \rightarrow A$ such that for

$$\forall a_i, a_j, a_z \in A (i, j, z \in \mathbb{N}),$$

where $\text{next}(a_i) = a_j \mid ((a_i \prec a_z) \bullet (a_z \prec a_j) \rightarrow (a_i = a_z) + (a_j = a_z)) \equiv 1$, ($\bullet$ - logical "and", $+$ - logical "or").

Then the set of values of information objects will be

$$V = \{v_i, \Omega \mid i \in \mathbb{N}\},$$

where Ω - means undefined value.

Let us also denote operations (arithmetic and logical) e . Let's introduce three types of operations:

- $e(a_1, a_2), a_1, a_2 \in A$ - the value of e - the value of a two-place operation on information objects located under the names a_1, a_2 ;

- $e(a_1), a_1 \in A$ - the value of the e -value of the unary operation on the information object located under the name a_1 ;

- $e(\text{const})$ the value of e is equal to $\text{const} \in V$.

We also introduce the write function: $Q \times A \times V \rightarrow Q$.

Let us denote $q(a_u) = v, v \in V$ the operation of assigning the information object $\text{elem}(q, a_u)$ to the value $v \in V$ $q' = \text{write}(q, a_u, v) = q(a_u) = v$.

That is, write means that an object named a_u is assigned the value v when the set of available objects is q . If c is an assignment, condition, or transition operator, then we can respectively write:

cases c :

$$u := e \rightarrow f_1(u, e)$$

$$\text{IF } b \text{ THEN } c \rightarrow f_2(b, c)$$

$$\text{goto } a \rightarrow f_3(a),$$

where f_1, f_2, f_3 - are expressions that determine the result in each specific case. The conditional expression is written like this:

$(b \rightarrow \dots$

$T \rightarrow \dots),$

if b - is true, then the result is taken from the first line, otherwise - from the second.

Let's set the function of transitions between states of the information process

$$f : Q^D \times (Q^C \times A) \times Q^N \rightarrow Q^D \times (Q^C \times A) \times Q^N;$$

$$f(q^D, L, q^N) = \text{cases elem}(q, a_c) :$$

$$\text{comp}(a_u, e) \rightarrow \text{write}(q, a_u, e); (q^{D'}, (q^C, a'_c = \text{next}(a_c)), q^N)$$

$$\text{IF } a_u \text{ THEN } c \rightarrow$$

$$((a_u \rightarrow (q^{D'}, c); (q^C, a'_c = \text{next}(a_c)), q^N)$$

$$T \rightarrow (q^D, (q^C, a'_c = \text{next}(a_c)), q^N))$$

$$\text{goto } a_l \rightarrow (q^D, (q^C, a_c = a_l), q^N)$$

default: error.

This transition function is common to models at different levels of detail. In this model, the set Q^C denotes the software component of the information process, and Q^D denotes the information being processed.

During the work of the calculator, the elements of the Q^C set are selected and interpreted into actions on the elements of the Q^D set. Access control rules and security policy are implemented in the software component of information processes based on the properties of the information being processed.

3 Information Threat Model

This model is designed to formalize the information process in automated systems and develop dynamic, in particular, emulating, methods for identifying vulnerabilities in software.

Thus, the threat Y can be represented as a triple of sets:

- a set of objects O ;
- many sources of I ;
- many points of application T ,

$$Y = \{O, I, T\}.$$

Let's consider the structure of these sets in more detail. The set of objects O , to which security threats are directed, consists of information objects, which are pieces of information and rules for their processing. Actually, the threat to information security is access or modification in violation of these rules.

Information processing rules can be represented in the form of logical expressions or predicates that determine the permitted actions on information objects [7]. Those. the information process $\langle \Pi \rangle$ can perform the operation $\langle O_n \rangle$ over the information object $\langle O \rangle$ only if the predicates $\langle \Pi \rangle \in \langle O \rangle$ take values "true" for $\langle O_n \rangle$ and $\langle \Pi \rangle$.

In addition to the rules that determine the allowed operations, there are also predicates that describe the internal properties of information, for example, its integrity and consistency.

These rules are set in the design of information technology and are reflected in the documentation and specifications for information arrays and software. When transferring information between information flows, all predicates must have the value "true". Within threads, rules can be violated [8].

Control over the implementation of information processing rules is carried out by a dedicated group of information processes called the security core. They work with information objects called security descriptors, which are predicates of other information objects. Predicates describing the rules for working with descriptors are set during the design of information technology and remain unchanged during the entire period of its use. Thus, the model distinguishes two groups of processes - normal and related to the security core. The former perform the functional tasks of the information system, while the latter ensure the security of the processed information.

4 Software Vulnerability Model

Multiple threat sources Π are the causes of information security breaches. Such sources can be random factors of the external environment or the system itself, as well as intruders. What is important for us is not their subjective or objective nature, but the fact that these sources initiate the process of implementing the threat.

Application points T are vulnerabilities on the exploitation of which a specific information security threat is based. These include organizational flaws, errors, hardware failures and failures, software vulnerabilities.

To consider the software vulnerability model, a software tool model was built and examined. This model makes it possible to describe and analyze both machine codes and source codes of programs based on the use of the apparatus of logical schemes of programs and algorithms, which allows presenting software in a general, formalized form and unifying the apparatus of subsequent analysis. For each new language, it is only necessary to develop a module for translating program schemes into the language, and the implementation of the rest of the analysis methods will remain unchanged [4-6].

Vulnerabilities are characterized by both random errors and errors such as undeclared capabilities.

The number of random errors that appear during the operation of a software tool depends on the effort spent on debugging and testing the software, as well as on how much the tests correspond to real operating conditions.

The program scheme was used as a model of the software tool. Let's denote the program scheme as

$$\mathfrak{S}(p_1, \dots, p_k, A_1, \dots, A_n),$$

where - $p_1, \dots, p_k$ - are boolean variables,

- $A_1, \dots, A_n$ - functional objects.

Let the diagram $\mathfrak{S}$ represent the software component of the information flow I , the hardware component is implemented by the calculator A , the information object IO with the descriptor D acts as the processed information. The specification S for the software component of the information flow I is given in a formal language. Descriptor D is:

$$D = \{d_1, \dots, d_m\}, \quad (1)$$

where $d_1, \dots, d_m$ - are predicates describing the requirements and rules for working with the IO information object.

A program tab is a set of program statements

$$\mathfrak{T}_b = \{p_{l_1}, \dots, p_{l_m}, A_{i_1}, \dots, A_{i_n}\} \subset \mathfrak{T}, \quad (2)$$

receiving control in the case when some subset of logical variables

$$\{p_{j_1}, \dots, p_{j_v}\} \subset \{p_1, \dots, p_k\} \in \mathfrak{T} \quad (3)$$

takes on the value "true".

Let us call the set of operators (2) the body of the program tab, and the set of variables (3) - the activating event. Expressions (2) and (3) are suitable for describing both "hatch" type tabs and "logic bombs" type.

A self-propagation block in viral-type tabs is a set of program operators and logical variables

$$\{p_{h_1}, \dots, p_{h_m}, A_{t_1}, \dots, A_{t_n}\} \subset \mathfrak{T}_b, \quad (4)$$

providing writing $\mathfrak{T}_b$ to the information object IO and changing the descriptor D so that IO becomes a software component for some information flow.

Let's designate O^S the set of all specifications, $O^\mathfrak{T}$ - the set of program schemes. We introduce the function

$$V : O^S \rightarrow O^\mathfrak{T}, \quad (5)$$

matching $S \in O^S$ the schema to $\mathfrak{T} \in O^\mathfrak{T}$ the specifications. Let's call the class of circuits equivalent $\mathfrak{T}$ to those deducible from S . All programs whose circuits are not deducible from their specifications contain errors or undeclared capabilities.

5 Conclusion

It is obvious that an information flow is a potential threat if its software contains a vulnerability. The presence of a vulnerability is determined by the fact that the program scheme cannot be deduced from the specifications.

Thus, the model of threats to information security, considered in the context of information technology, allows a comprehensive approach to the description and analysis of existing and possible threats. The most common and dangerous threats are those arising from software vulnerabilities. The proposed model of software vulnerability using program schemes is focused on the development of tools for automatic control and assessment of software security. It allows the translation of a formal description of a program, given in any procedural programming language, into a program schema language for subsequent analysis to identify existing vulnerabilities.

References

1. Landyshev V A Levchenkov A N Landysheva O N 2017 *Proceedings 8th International Scientific Conference* 29-31
2. Lipaev V V 1981 *Reliability of ACS software* (M: Energoizdat) p 241
3. Lipaev V V 2013 *Reliability and functional safety of real-time software complexes* p 176

4. Levchenkov A N 2017 *Proceedings of the IV International Scientific and Practical Conference "Actual directions of scientific research: development prospects"* **1S** 283-288
5. Levchenkov A N 2019 *Proceedings of the All-Russian Scientific and Practical Conference "National Security in Russia: Actual Aspects STI "NATSRVITIE"* 40-47
6. Lobko V T 1997 *News of higher educational institutions North Caucasian region Technical sciences* **2** 12-19
7. Lyapunov A A 1968 *About logical schemes of programs In the collection Problems of Cybernetics* (M: State Publishing House of Physical and Mathematical Literature) p 320
8. Yanov Y I 1958 *On logic circuits of algorithms In the collection Problems of Cybernetics* (M: State Publishing House of Physical and Mathematical Literature) p 19
9. Myers G 1980 *Reliability of software* (M: Mir)
10. Liskov B Gateg J 1989 *Using abstractions and specifications in software development: Per from English* (M: Mir)
11. Guttag John V, Horning James J, Wing Jeannette M 1985 *Larch in five easy pieces. Technical report 5* (DEC Systems Research Center)