Наведені результати розробки інструментарію сценарного моделювання на основі формальних граматик. Проаналізовано роботи, пов'язані з різними способами опису сценаріїв в системах їх розробки. Для природно-мовного опису сценаріїв зазначено, що такий підхід є досить прозорим і зрозумілим для користувачів. Однак такий підхід має ряд недоліків для формалізації та уніфікації опису сценаріїв. Зокрема, наявність у мові ряд неоднозначностей робить його неможливим для однозначно-інтерпретується опису, і як наслідок – малопридатним для виконання формальних перетворень над описом. Графічне представлення сценарієм є наочним поданням сценарію. Більш того, наочне уявлення сценарію у вигляді деякої автоматної моделі може бути оцінений як вкрай привабливе для подальшого мультиагентного моделювання його виконання. Недоліком такого опису сценаріїв є, як і раніше, труднощі виконання формальних маніпуляцій і необхідність переходу до більш зручного для маніпуляцій уявлення. Використання формальних граматик для опису сценаріїв є компромісним підходом, який дозволяє описувати сценарії в однозначно інтерпретується формі. Формально-граматичний опис також більш звично фахівцям з комп'ютерних мов. І крім того, є програми, орієнтовані на роботу з формальними граматиками. Показаний перехід від природно мовного опису сценаріїв до його формального поданням у вигляді стандартного опису в Бекуса-Наура формі. Зміна форму подання зроблена на прикладі опису сценарію поведінки учасників кіберконфлікта в системі безпеки. Отриманий опис сценарію використано в аналізаторі контекстно-вільних граматик. Отримані результати показали можливість застосування пропонованого підходу і використовуваного інструментарію для опису і перевірки коректності опису сценаріїв, що відносяться до будь-якої предметної області

Ключові слова: сценарне моделювання, система безпеки, формальна граматика, контекстно-вільна граматика, Бекуса-Наура форма

UDC 681.32:007.5
DOI: 10.15587/1729-4061.2019.184274

DEVELOPMENT OF SCENARIO MODELING OF CONFLICT TOOLS IN A SECURITY SYSTEM BASED ON FORMAL GRAMMARS

O. Milov
PhD, Associate Professor*
S. Yevseiev
Doctor of Technical Sciences, Senior Researcher*
E-mail: serhii.yevseiev@hneu.net
A. Vlasov
PhD
Air Force Science Center**
S. Herasimov
Doctor of Technical Sciences, Senior Researcher
Department of Combat Use of Weapons of Air Defense of the Ground Forces**
O. Dmitriiev
PhD, Head of Department
Department of Flight Operations, Aerodynamics and Flight Dynamics
Flight Academy of the National Aviation University
Dobrovolskoho str., 1, Kropyvnytskyi, Ukraine, 25005
M. Kasianenko
PhD
Department of Radio Technical and Special Troops***
H. Pievtsov
Doctor of Technical Sciences, Professor, Honored Master of Sciences and Engineering of Ukraine**
Y. Peleshok
PhD, Deputy Head of the Research Center
Institute of Special Communication and Information Security
National Technical University of Ukraine "Igor Sikorsky Kiev Polytechnic Institute"
Verkhnokliuchova str., 4, Kyiv, Ukraine, 03056
Y. Tkach
Doctor of Pedagogical Sciences, Associate Professor
Department of Cybersecurity and Mathematical Simulation
Chernihiv National University of Technology
Shevchenka str., 95, Chernihiv, Ukraine, 14035
S. Faraon
Adjunct
Department of Communications and Automated Control Systems***
*Department of Cyber Security and Information Technology
Simon Kuznets Kharkiv National University of Economics
Nauky ave., 9-A, Kharkiv, Ukraine, 61166
**Ivan Kozhedub Kharkiv National Air Force University
Sumska str., 77/79, Kharkiv, Ukraine, 61023
***Ivan Chernyakhovsky National Defense University of Ukraine
Povitroflotskyi ave., 28, Kyiv, Ukraine, 03049

Received date 29.08.2019
Accepted date 01.11.2019
Published date 27.12.2019

1. Introduction

The future is not a static continuation of the past. Scenarios reflect the fact that several potential future options are possible at any given point in time. Scenario studies typically focus on issues that are sensitive to stakeholders and provide the means by which decision makers can anticipate upcoming changes and prepare for them quickly and

in a timely manner. Through the study and assessment of possible future conditions, scenario studies allow us to assess system vulnerabilities and opportunities of adaptation measures. Scenario planning can lead to more informed decisions by bridging the gap between scientists and decision makers, while at the same time highlighting issues of immediate concern.

Scenario generation is a complex process and inherently involves significant interactions between the researcher and stakeholders and/or expert judgment. Leaders and stakeholders do not always trust forecasting and long-term planning, similar to scenarios, because, in their opinion, this method is practical only if the future can be extrapolated from the past. Therefore, there is a real need for the development of formalized methods for constructing scenarios that do not allow subjective approaches in this area.

The larger the scope of the study in scenario building activities, the greater the number of parties involved in the process, especially if the scenarios represent a field of study at the national or regional level. Traditionally, scenario planning has always taken into account specific problems. However, it should be noted that only by integrating strategies and scenarios within a single structure, the full potential of scenarios can be realized. In order to avoid duplication of work and to facilitate collaborative scenario planning in these large-scale applications, a formal structure is needed for a systematic and organizational approach to scenario generation. This approach can be applied to all scenario research in order to exchange relevant information related to scenarios and form a community of script developers.

The formulated statements lead to the conclusion that it is necessary to develop tools for scenario modeling based on the use of formal approaches and methods and to demonstrate their applicability and effectiveness for modeling security system processes.

2. Literature review and problem statement

Publications devoted to formalizing the description of scenarios and their subsequent generation can be divided into three groups. The first group combines publications describing the use of natural languages (or a limited subset of them) to describe those developed by the script. The second group contains publications on various graphical means of representing and manipulating scripts. The publications of the third group describe various linguistic means that are more rigorous in comparison with natural languages.

The most revealing publication related to the first group is [1]. The paper notes the need for understanding between engineers (analysts, programmers, security experts) and users. Engineers should understand, model and analyze the scope of the application for which the scenario is being developed, and clients/users should verify that the engineers' vision is correct. Scenarios describe situations that occur in the subject area. It is understood that the latter includes all sources of information and all people associated with the implementation of the script. This is a reality limited by the set of goals set by those who are customers of the scenario development. In [2], taking into account aspects of use, it is assumed that scenarios allow to find out the problem, unify the criteria, achieve a compromise between customers and users, organize the details involved and train new participants. The use of scripts as a way of understanding the subject area of its use was recommended in [3–5], and these suggestions became very important for expanding the use of scripts in real practice. However, the analysis of recommendations presented in [6, 7] shows a certain degree of contradiction when using scenarios.

The lack of accuracy of when and how to use scripts has spread to engineers who use these methods in the field. Thus, most developers view scripting as a craft rather than an engineering task. Studies regarding the use of scenarios in industrial projects have confirmed this fact and pointed out the need for more detailed definitions of building scenarios to increase their use in real situations.

A variety of interpretations, syntax, and construction mechanisms for scripts comes to identifying the main contradictions. For example, with respect to the scripting process, there is no consensus on whether it should be top-down or bottom-up scenario development.

In [1], a strategy for creating and using scripts is proposed, based on the assumption that scripts should be based on natural language as a means of communication between stakeholders, in particular between clients/users and requirements engineers.

Using natural language to describe the scenario helps the client/user check it and is consistent with the goal of improving stakeholder communication. Using Language Extended Lexicon (LEL) and scripts to identify requirements and their use throughout the development process allows validation with the client/user. The main purpose of the lexicon is to cover the application dictionary and its semantics, postponing the understanding of application functionality. Scenarios are used to understand its functionality: each scenario describes a specific situation, focusing on the behavior of its participants. The script is built on the basis of a dictionary that reflects the specific and most used words or phrases in the use area of the script, which must be present in LEL.

The use of a glossary containing an application dictionary is also proposed. In other words, the proposal is to create not just a glossary, but a lexicon that includes the designation of each symbol found in the form of a word or phrase related to the field of application. The purpose of this lexicon is not only to ensure good communication and coordination between customers/users and the development team, but also to help describe them, which will facilitate the verification process at the beginning of the scripting process. The use of lexicon symbols in scripts allows these symbols to be a natural hyperlink between these two presentation structures, a fundamental characteristic of the concept of basic requirements for a script.

The fundamental characteristic of this approach is that a natural language script is tied to the lexicon of the application language. This characteristic is original and solves the important problem of reducing ambiguity in descriptions in natural language. Since the script uses LEL characters, they become hypertext, and the lexicon characters are hyperlinks between the two representations.

A scenario model is a structure consisting of objects: a title, a goal, a context, resources, actors, episodes, exceptions, and attribute constraint. Actors and resources are an enumeration. The title, goal, context, and exceptions are declarative sentences, while episodes are a set of sentences

expressed in simple language that provide a quick description of behavior.

A fragment of the scenario model is as follows:

Scenario: description of the situation in the domain.
Syntax: Title+Goal+Context+{Resources}+
{Actors}+{Episodes}+{Exceptions}
Title: Scenario identification. In the case of a supporting scenario, the title matches the sentence of the episode without restrictions.
Syntax: Frase ([Actor|Resource]+Verb+Predicate)
Goal: the goal must be achieved in the application domain. The scenario describes the achievement of the goal.
Syntax: [Actor|Resource]+Verb+Predicate

In general, the natural language for describing scenarios is used in the context of a frame model for representing knowledge about a subject area. This combination determines both the advantages and disadvantages of the proposed approach. The advantages include the clarity of the script description language for users. The frame model allows you to structure scenario description and use appropriate tools for working with frame models. Moreover, psychologists believe that a person thinks in frames, so this model is to the best degree consistent with the process of logical inference of a person. The disadvantages include the presence of different types of ambiguity in natural language, which does not allow the creation of unambiguously interpreted structures. An attempt to use a limited subset of natural language can significantly narrow the expressive means of the latter. In addition, the considered approach to the design and use of scripts does not contain any indications of a means of verifying the correctness of the script.

Of greatest interest among the works describing graphical script development tools is the work [8]. In the work, the scenario is defined as follows. A script is a tool for determining the functionality and behavior of a system from a user's perspective. Thus, it is used in most modern methods of object-oriented development to identify and document user requirements. Scripts also form a kind of abstract test scripts for the developed system.

When developing a software system, validation and verification are recognized as vital activities that are especially valuable when applied in the early stages of the development process. This is because errors found in the specification and design phase are much cheaper to fix than errors found in subsequent steps. Early verification, therefore, significantly reduces error correction and the cost of errors.

Under these conditions, the emphasis should be on testing the correctness of the developed scenarios. In addition to well-established methods, such as data flow control and testing or boundary analysis/domain testing, formal languages for specifications and specialized testing languages are attracting increasing attention. Nevertheless, the gap between theory and practice remains. The gap between what theoretically can be done and what is actually done in practice is mainly due to the reasons presented in Table 1.

The described method can be easily integrated with software development methods and help developers create test situations and develop scenarios at the early stages of the development process, supporting the systematic creation of test cases (Table 2).

Table1

Reasons for the discrepancy between theory and practice of scenario modeling

No.	Reason	Description
1	Lack of planning/time and funds	In real projects, scenarios are modeled in the face of enormous time and cost, since often the project at the end of the development process is behind the schedule and already exceeds the budget. Fault detection causes additional delays. As a result, test preparation and execution are often performed only superficially. The costs and time required for testing are difficult to estimate with sufficient accuracy. Moreover, testing is often not well planned and lacks the time and resources
2	Lack of (testing) documentation:	Tests are not prepared properly, test plans are not developed and tests are not documented
3	Hard work:	Testing and developing test cases are tedious, repetitive, error-prone, and time-consuming activities that cause fatigue and carelessness, even if reliable testing strategies and methods are used
4	Lack of instrumental support	For this reason, testing should be supported by tools. But there is only limited support for tools. Extended support for tools and, in particular, automatic generation of test suites, is limited to systems that are formally defined. Even though automatic test case generation can be applied in a formally defined system, the resulting test sets are huge and, as a rule, only poor coverage is achieved
5	Formal/special testing languages required	Many testing methods use formal specification languages or special testing languages (therefore, special training and education are required). Their use is extremely expensive, difficult to apply and/or they can only be used for limited problems or very specific areas
6	Lack of measures, measurements and data for the quantification of tests and assessment of test quality	In most projects, only a small amount of testing data (error statistics, coverage measurements, etc.) is collected during testing or is available from other projects. Due to the lack of data on the benefits and cost-effectiveness of testing, very little can be said, different approaches cannot be compared, and processes can hardly be improved. The quality of the tests, and therefore to some extent the product, is often not evaluated. In addition, missing data further aggravate the problem of accurate test planning and allocation of necessary resources

A script is defined as any form of description or presentation of user-system interaction sequences. The terms scenario, use scenario, and subject are defined as follows.

A scenario is an ordered set of interactions between partners, usually between a system and a set of actors external to it. May contain a specific sequence of interaction steps (instance script) or a set of possible interaction steps (type scenario).

A use case is a sequence of interactions between an actor (or actors) and a system launched by a specific actor, which gives the result for the actor (typical scenario).

Actor is the role played by the user or an external system interacting with the specified system.

Table 2

Key ideas of the proposed approach

No.	Idea	Description
1	Use natural language scripts	Not only to identify and document requirements, to describe the functionality of the system and determine the behavior of the system, but also to verify the developed system in the process of its development
2	Uncover ambiguities, contradictions, omissions, inaccuracy and uncertainty	In descriptions in natural language (like scripts in SCENT) by formalizing storytelling scripts in state diagrams [10]
3	Formulate descriptive scenarios and/or state diagrams	Where necessary, with preconditions and postconditions, data ranges and data values, as well as non-functional requirements, especially performance requirements, to provide all the information needed for testing and make state diagrams suitable for outputting actual, specific test cases
4	Display test cases systematically	For system testing, bypassing paths in state diagrams and documenting test cases

It is noted that the initial formation of a script in natural language preserves the problems of natural language specifications: natural language is not accurate, definite, and unambiguous. Narrative scenarios can be ambiguous, inconsistent, and incomplete. Formalization helps to find and avoid these problems. Formal languages allow formal reasoning, (strict) verification and validation. But formal languages also have their drawbacks: they require knowledge of a special language, they are difficult to understand, and their application may be error-prone.

An intermediate path was chosen in the work, the transformation of the natural language script into semi-formal state diagrams. This formalization helps to find many omissions, ambiguities and inconsistencies, however, the graphical representation of scripts can be well understood by users if there is some guidance from developers. Thus, formalization is a very useful troubleshooting procedure and can be considered as part of static testing.

Creating scripts and state diagrams is an iterative process. State diagrams should be checked with the user. All (important) paths in the graphical representation must be completed. This verification operation works in parallel with the use phase: the paths traveled by the client to check the state diagrams are test cases that need to be tested in the system. Once again, it should be emphasized that this process is not sequential, many activities can, at least partially, be carried out in parallel; they take advantage of each other and use the same artifacts.

State diagrams describe the behavior of the system (how the system behaves in response to events, data, conditions), and generates important information such as data, performance, quality. This additional information is important for testing: many data-related errors can only be detected in test cases, which cannot be directly obtained from state diagrams. Moreover, specific test data, i.e., input values and expected result, can only be obtained directly from state diagrams.

The state diagram showing the "Authentication" scenario is shown in Fig. 1.

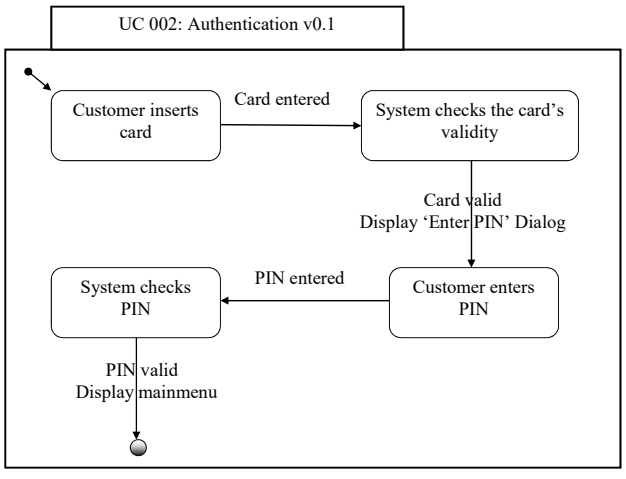

Fig. 1. State diagram representing the "Authentication" scenario [8]

To include information important for testing, it is proposed to expand the notation of state diagrams. In particular, additional testing information may include the following:

– preconditions (and postconditions as necessary);
– input data: input, expected output and ranges;
– non-functional requirements.

Information may be recorded in annotations.

In general, conclusions on the advantages and disadvantages can be formulated as follows.

The advantages include the high visibility of the proposed approach. Using a state diagram allows you to automatically generate a finite state machine that runs on the specified program. The implementation of the state machine, in turn, in the form of a software agent, makes the indicated approach valuable for the implementation of multi-agent systems.

The lack of means for verifying the correctness of scenario formation should continue to be attributed to the disadvantages, since the latter are initially formed in natural language, and the subsequent transition to the formation of a state diagram verifies the correctness of the transition, but not of the initial scenario. In addition, the graphical presentation of scripts is more understandable to UML (or SysML) specialists, rather than the subject area of script usage.

First-order predicate logic can be used as a basic model of knowledge representation for implementing scenario synthesis based on causal relationships. In simple cases, propositional logic can be used to represent knowledge. The scenario synthesis is reduced to constructing a formal generating grammar on a certain set of terminal and nonterminal symbols. The syntax of the propositional logic corresponds to the context-free generating grammar, which is described by the four $G=(V, T, P, S)$, where V is a finite set of nonterminal symbols; T is a finite set of terminal symbols that does not intersect V; P is a finite set of production rules written in the form of Backus notations and representing a decoding of nonterminal grammar symbols with logical expressions containing symbols from T and V; S is the initial character of the grammar.

The term grammar, taken from linguistics, means a set of rules of a language that allow to build and recognize the "correct" phrases in this language.

A phrase is a finite sequence of words that are indivisible elements (terminal symbols). In any language, not

any combinations of words are allowed, therefore phrases corresponding to certain rules of syntax and semantics are considered "correct". A formal grammar is a set of logical rules (syntax) that establish ways to combine words and phrases (non-terminal characters) to form more complex expressions. The rules are applied recursively and allow you to generate an infinite number of phrases (logical formulas) that make up some language. Formal grammar products can be interpreted as a logical formalization of natural language grammar rules, and phrases as subsets of many logical formulas. This mechanism can be used to generate complex systems and objects of any nature with a sufficient degree of knowledge of the studied subject area.

The scenarios are based on the assumption that several potential future scenarios are possible at any given point in time. Scenario studies usually focus on issues that are sensitive to stakeholders, and they provide the means by which decision makers can anticipate upcoming changes and prepare for them quickly and in a timely manner. By examining and evaluating possible future conditions, scenario studies allow us to evaluate system vulnerabilities and opportunities for adaptation measures. For example, decision makers can use scenarios to guide control policies and implement strategic planning for the impacts of an alternative future. Scenario planning can lead to more informed decisions by bridging the gap between scientists and decision makers, while highlighting issues of immediate concern [9–12].

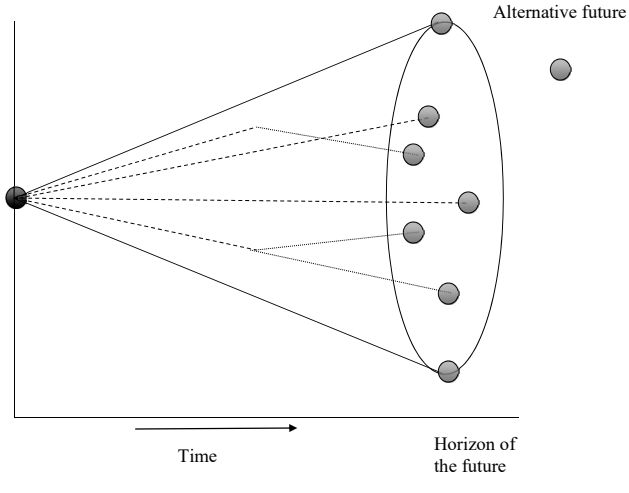

Fig. 2. Conceptual diagram of a scenario funnel. Adapted from [13]

One of the most important characteristics of a scenario is that it should be logically plausible. Plausible scripts provide logical descriptions and explanations of possible events, adding credibility to the main part of the work, which the scripts are designed to complement. In order to further increase the reliability, the probable scenario should also be internally coordinated with the driving forces, which are crucial for the development of the scenario trajectory [14]. To eliminate redundancy, scenarios should be different, focusing on different driving forces and/or goals of the script, but still preserving a set of common input variables so that results from different scenarios can be compared. Useful scenarios should also be creative and check limits when exploring the unknown future, while remaining tied to the purpose of their use and being fully quantified and qualitatively defined [15]. The simplest basic scenario is the "official future" scenario, the "ordinary business" scenario of the widely accepted future state of the world. Most decision makers will not accept future alternatives unless the official future is called into question [11].

3. The aim and objectives of the study

The aim of the work is to develop scripting modeling tools based on formal-language methods, including means for verifying the correctness of the scenario of the interacting parties of cyber conflict behavior in the security system.

To achieve this goal, it is needed to accomplish the following objectives:

– to give a natural language description of the scenario of participants' behavior in a real cyber conflict;

– to analyze existing grammars and choose the most suitable and appropriate for the behavior scenarios of the participants in cyber conflict;

– to show how a natural language description of a script can be converted into a BNF form, which can be considered as the initial description of a script for a program for analyzing the correctness of description of a prototype script, and check the formal-grammatical representation of the script.

4. Natural-language description of the behavior scenario of security system agents

One of the key findings from the economic literature on information security [16] is that attackers who seek to undermine system security act strategically. In addition, information systems are often structured in such a way that the overall security of a system depends on its weakest link [17]. The most careless programmer in a software company may present a critical vulnerability. The global distributed architecture of the Internet leads to the weakest link dominating in the security system – attackers compromise computers hosted by Internet service providers (ISP) or located in countries that do not cooperate with them. Attackers have repeatedly demonstrated their ability to find the easiest way to circumvent a security system, even if the system designer is not aware of this particular vulnerability.

However, systems do not exist in a vacuum; rather, defenders respond to attacks by blocking known holes. And yet, as soon as one weak link is identified and corrective action is taken, another weak point is often discovered and used. Therefore, a strong dynamic component comes into play: attackers find the weakest link, defenders solve the problem, attackers find new holes, which then connect, and so on. It can be seen that this picture appears repeatedly. For example, cybercriminals create networks of compromised machines (called botnets) in order to reach legitimate users by spreading spam, spreading malware, and hosting phishing websites. Attackers concentrate their efforts on the most irresponsible Internet providers, switching to others only after the Internet provider clears its actions or closes. Similarly, technical measures to counter payment card fraud have evolved over time, forcing fraudsters to adopt new strategies, as old flaws are eliminated [18].

The basis of the described behavior scenario is the following mathematical model.

In some confrontation environment, *a defender* protects *an asset* of value *a* from a distributed set of possibly hetero-

geneous *attackers*. There are *n* possible threats that can be considered as separate attack vectors on one system. Each threat can be prevented by investing in appropriate protection (or control). In other words, a one-to-one comparison of threats and defenses is assumed, and also that protection is always effective.

Thus, the model, which reflects the dynamic interaction between attackers and defenders, is focused on the iterative aspect of attack and defense. The case when safety depends on the weakest link is exclusively studied; therefore, the iterative weakest link is modeled. Key model features include:

1. Protective countermeasures can be represented as interdependent; thus, the often mentioned decreasing marginal return on investment in information security [17] becomes endogenous in our model.

2. The defender's uncertainty as to which components are the weakest is fixed.

3. Reconfiguring the game reflects the iterative process of attacking and defending consecutive weak links.

Studying the model leads to several interesting conclusions. A comparison of the static case (one round of attack and defense) with dynamic (several rounds) shows that different strategies of defenders may prevail. When a defender has only one chance to protect the system, growing uncertainty about which link is the weakest forces the defender to defend more assets, but only to a certain extent. When the uncertainty is too high, the defender does not know which asset to protect, and therefore decides not to protect any. If re-investment is allowed instead, the vague defender will initially protect fewer assets and wait for the attacker to "identify" the weakest links that will be fixed in subsequent rounds. Therefore, it may be rational to invest in security until the threats are realized. Unlike other theories, this type of underinvestment is not caused by relationships with other market participants and the resulting incentive systems. Of course, security measures may require significant investments from the very beginning. When unpaid costs are introduced into the model, it is found that, at moderate levels of uncertainty, higher unpaid costs may increase investment in active defense.

Then, conclusions about optimal defensive strategies are converted into accepted safety indicators, such as annual loss expectancy (ALE) and return on security investment (ROSI) [20]. The return on investment decreases as uncertainty about known attacks grows, even as the defender becomes more and more responsive to countering the realized threats.

The model is "launched" in an iterative game with discrete time *t*. In each round, the defender decides to invest in the security system to determine his d_t protection configuration, and extracts net profit $r \cdot a$ from his asset before the attacker penetrates the system and, if successful, receives part *z* of the asset. Gross profit is used or distributed so that the value of assets does not accumulate over time.

Let the elements d_i of the binary vector of the column $d \in \{0,1\}^n$ indicate whether protection against the *i*-th threat (d_i=1) is implemented or not (d_i=0), and let $k = \sum_i^n d_i$ be the number of available countermeasures.

The protection cost c_t in round *t* can be calculated using the matrix *C* of the upper triangular matrix of value $n \times n$, to reflect possible interdependent defenses,

$$c_t = d_t C d_t.$$

The diagonal elements $C_{i,i}$ contain the costs of implementing protection from the *i*-th threat, and the off-diagonal elements C_{ij}, $j>i$ indicate additional costs if the *i*-th protection is implemented together with the *j*-th protection. If all off-diagonal elements are equal to zero, then the protections are independent:

1. An alternative interpretation is that threats are separate targets in a distributed system that together form asset *a*. This interpretation approaches the concept of the weakest goal, in contrast to weak-link games in [21].

2. Higher-order interdependence, such as additional costs, is not taken into account if three or more protections are involved.

Matrix of value for independent defenders

$$C = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Matrix of value for conflicting (or interdependent) defenders

$$C = \begin{bmatrix} 1 & 0 & ½ & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -½ \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Why modeling interdependent defenses? First, organizational science suggests that complexity adds super-linear administrative costs, for example, to pay a manager who directs several employees, each of which is an expert for a certain amount of control. Conflicting protection also occurs in incompatible systems, for example, when two anti-virus scanners are launched on the same computer, the computer slows down and errors occur to increase coverage. Finally, human behavior significantly limits the ability to combine protections: a password policy that requires both special characters and frequent password changes encourages people to attach their password to their monitor.

A good property of the cost matrix is that for positive off-diagonal elements, a decrease in the marginal utility of protection has become endogenous for our model. This compares favorably, say, with the Gordon-Loeb framework, in which this property appears as an assumption [19].

For now, suppose a defender can update his d_t configuration at any time. This may be necessary in order to adapt it to new information about the level of threat or to a change in risk appetite. For example, a start-up company is exposed to so many risks that it can tolerate a moderate level of information security risk. As an enterprise grows and develops a brand, its risk aversion will increase, reflecting more damage caused by potential loss of reputation. Conversely, market competition (along with herd behavior and short-sighted incentives for management) may encourage firms to reduce their risk aversion for higher expected returns.

As stated in [22], a reduction in security investments is often possible to a certain extent, as personnel can be fired or equipment sold (although sometimes with high operating costs). However, for many companies, updating d_t can be very expensive, and the costs are "sunk" in the sense that money is being spent forever. For example, the vast majority of the costs of incorporating new security features into banknotes or payment cards are covered upon the first change. The

production and distribution of specialized tokens or devices among a large and dispersed community are expensive and unlikely to be repeated often. Since unrealized costs significantly affect the security investment strategy in an iterative environment, we include them in our model as follows:

$$s_t = \begin{cases} 0 \text{ if } t=1 \text{ or } d_t = d_{t-1}, \\ \lambda \cdot a \text{ else.} \end{cases}$$

Therefore, the parameter $\lambda \geq 0$ controls the amount of hidden costs.

Although the defender may have some intuition regarding the relative complexity of implementing n threats, such knowledge may well be blurred. To simulate this uncertainty, we order threats 1,..., *n*, increasing the expected cost of the attack (the expectation is taken from the defender's point of view). This constitutes our understanding of the attack profile. We define a simple functional form for the expected cost of attack $\bar{x}_i$ of the *i*-th threat as follows:

$$\bar{x}_i = x_1 + (i-1)\cdot \Delta x \text{ with } \Delta x > 0.$$

However, the unknown true value xi, is modeled as a Gaussian random variable with an average value $\bar{x}_i$ and standard deviation $\sigma / \Delta x$ (with restrictions on the values $x_i \geq 0$),

$$x_i = \sup(0, \chi_i) \text{ with } \chi_i \sim N(\bar{x}_i, \sigma / \Delta x).$$

Note that χ_i implementations remain constant over time. The level of uncertainty can vary by adjusting the parameter σ. Thus, uncertainty modeling is crucial for the model, as it reflects the difficulties that defenders face in anticipating which of the unprotected threats is the weakest link used by the attacker.

The defender's knowledge of the attack profile may increase over time when the observed attacks show which threat is the weakest link with respect to this d_t configuration.

The attacker's model is very simple: the attacker identifies and uses the weakest link, i. e. the threat is the least costly for the attacker. An attacker is not required to compromise between cost and potential benefit, assuming that the same utility is obtained for using all threats. If an attacker succeeds, no matter how he makes a profit $z \cdot a$, this is added to the defender's value. Unlike the defender, the attacker is confident in the cost of each implementation x_i. An attacker does not act indiscriminately; rather, it attacks only when it is profitable, that is, if the member $\max_i (z \cdot a - x_i)$ is not negative.

5. Analysis and selection of formal grammar for describing scenarios

First-order predicate logic can be used as a basic model of knowledge representation for implementing scenario synthesis based on causal relationships. In simple cases, propositional logic can be used to represent knowledge.

In general, there are two main ways to describe individual classes of languages:

– using a generative procedure;

– using a recognition procedure.

The first of them is set using a finite set of rules called *grammar* and generating exactly those chains that belong to the language L. The second – using some abstract recognition device (automaton). When constructing translators, both of these methods are used: grammar as a means of describing *the programming language syntax*, and an automaton as *a model of an algorithm for recognizing language sentences*, which forms the basis for constructing a script analyzer and generator. In this case, methodically (and technologically), a grammar is first constructed, and then, based on it, as a source, a recognition algorithm is constructed [23, 24].

We turn to the formal presentation of the concepts discussed above.

A formal generating grammar is *a quadruple* $G = <N, T, P, S>$, where

T – a finite nonempty set of characters called the terminal (main) dictionary of grammar *G*; the elements of the set *T* are called terminal symbols (*terminals*);

N – a finite nonempty set of characters, called a nonterminal (auxiliary) dictionary of grammar *G*, $T \cap N = \varnothing$, – a joint dictionary of grammar *G*; the elements of the set *N* are called nonterminal symbols (or *nonterminals*);

S – the initial symbol (*axiom*) of the grammar *G*; $S \in N$ denotes the main nonterminal (goal) of the grammar *G*;

P – a finite set of grammar rules, that is, chains of the form $\varphi \to \psi$ and also called *substitution rules* or *productions*, while φ, ψ – the chains in the dictionary $V = T \cup N$ and $\varphi \in (TUN)^* N (TUN)^*$, $\psi \in (NUT)^*$. Final bipartite relation $\to$is interpreted as "replacing" φ with ψ "or substituting φ instead of ψ".

The set of substitution rules *P* is also called a grammar scheme. The chain on the left side of the grammar rule must contain at least one nonterminal character. On the right side of the rule, in the general case, there can be an arbitrary chain of terminal and nonterminal symbols, including the empty chain λ.

The generating grammar, as defined above, is a powerful descriptive tool, but still very general. The practical use of grammars is related to solving the *recognition* problem. The recognition problem is solvable if there exists an algorithm that, in a finite number of steps, answers the question of whether an arbitrary chain over the main grammar dictionary belongs to the language generated by this grammar. If such an algorithm exists, then the language is called *recognizable*. If, in addition, the number of steps in the recognition algorithm depends on the length of the chain and can be estimated before the algorithm runs, the language is called easily recognizable. Otherwise, it makes no sense to talk about building a script generator for the unrecognizable language of its description. Therefore, in practice, such particular classes of generative grammars are considered that correspond to recognizable, and in most cases easily recognizable languages. The most important classes of such languages can be defined within the framework of the classification of languages, which suggests classifying them according to the type of rules of the grammar generating them.

Class 0. Grammar inference rules have the form $\varphi \to \psi$ without any restrictions on the strings φ and ψ. Languages of this class can serve as a model of natural languages.

Class 1. All elements of *P* are obtained from the form $\varphi \to \psi$, where $\varphi = \xi_1 \alpha \xi_2$, $\psi = \xi_1 \beta \xi_2$, and $\xi_1, \xi_2 \in V^*$, $\alpha \in N$, $\beta \in V^*$. The generating grammar with such rules is called *immediate constituent* grammar, or context grammar (ICG). The languages generated by the grammars of this class are called *context-sensitive*. In the ICG grammar, each inference

rule indicates the substitution of some nonempty chain β instead of the nonterminal α provided that the replaced nonterminal α is surrounded by ξ_1 and ξ_2, i. e., the strings ξ_1 and ξ_2 are considered as a context in which α can be replaced with β.

Class 2. All generating rules of the grammar have the form: A→β, where A is a nonterminal symbol, and β is a nonempty chain from *V*, that is $\beta \in V^*$. Replacing the nonterminal A with the string β occurs without regard to the context, therefore grammars of this class are called context-free (CF grammars).

Class 3. All generating rules have the form: $A \to bB$ and $A \to b$, where $A, B \in N$, $b \in T$, that is, the right part of the rule is either a single terminal or a single terminal, followed by a single nonterminal. *Class 3* languages are called *languages with a finite number of states* or *automaton* (*regular*) languages, and the grammars generating them are called automaton grammars (A grammars). A grammars are used mainly at the stage of lexical analysis.

The main classes of languages can be defined by classes of abstract recognition devices (automata), which also form the corresponding hierarchy. Table 3 shows the hierarchy of languages and the corresponding hierarchies of grammars and automata as recognition devices.

Table 3

Hierarchy of languages, grammars and automata

Type	Language	Automate
Type 0	Recursively Enumerable	Turing machines
Type 1	Context-sensetive	Linear-bounded non-deterministic Turing machine
Type 2	Context-free	Non-deterministic pushdown automaton
Type 3	Regular	Finite state automaton

Of the four classes of grammars, context-free grammars are most important when applied to the analysis and design of scripts, given their distribution in the field of programming languages. With their help, it becomes possible to define a large, although not entire, part of the syntactic structure of the scenario.

6. Presentation of the behavior scenario by means of BNF

It was previously shown how the normal Backus forms are used to describe the syntax of a language. It turns out that there is a direct connection between BNF and CF grammars – they are essentially equivalent, the differences concern only the notation. So, the relation "::=" from the BNF corresponds to the relation in the CF grammar – the non-terminal symbols correspond to the metalinguistic variables from the BNF in the CF grammar, the terminal symbols of the CF grammar correspond to the main characters of the programming language from the BNF. In CF grammars, to reduce the notation, rules with identical left parts are also collected into one, using the|(or)symbol as a separator of alternatives.

The scenario synthesis is reduced to constructing a formal generating grammar on a certain set of terminal and nonterminal symbols. The syntax of the propositional logic corresponds to the context-free generating grammar, which is described by the four $G=(V, T, P, S)$, where *V* is a finite set of nonterminal symbols; *T* is a finite set of terminal symbols that does not intersect *V*; *P* is a finite set of production rules written in the form of Backus notations and representing a decoding of nonterminal grammar symbols with logical expressions containing symbols from *T* and *V*; *S* is the initial character of the grammar.

The term grammar, taken from linguistics, means a set of rules of a language that allow to build and recognize the "correct" phrases in this language.

A phrase is a finite sequence of words that are indivisible elements (terminal symbols). In any language, not any combinations of words are allowed, therefore phrases that correspond to certain rules of syntax and semantics are considered "correct". A formal grammar is a set of logical rules (syntax) that establish ways to combine words and phrases (non-terminal characters) to form more complex expressions. The rules are applied recursively and allow you to generate an infinite number of phrases (logical formulas) that make up some language. Formal grammar products can be interpreted as a logical formalization of natural language grammar rules, and phrases as subsets of many logical formulas. This mechanism can be used to generate complex systems and objects of any nature with a sufficient degree of knowledge of the studied subject area.

The language spoken about the language is called a metalanguage. Metalanguages are actually all natural languages. However, using natural language to describe some other language (including scripting language) is not a good solution. This is due, first of all, to the fact that natural language is inaccurate and redundant, and it is difficult to adapt to the formal manipulations necessary in the process of analysis and generation of scripts. Linguists note at least 5 types of ambiguities in natural language: from linguistic to pragmatic. Therefore, one should turn to the existing metalanguages for describing grammars of various types. Such a metalanguage, which has become widespread, is a formal metalanguage for determining immediate constituent grammars, is the Backus-Naur form (BNF).

There should be no confusion between the symbols of the metalanguage and the language being described. The BNF avoids such confusion by using only four metalinguistic symbols (Table 4). The appearance of literal characters without parentheses indicates that these are terminal characters of the language. The grammar is written as a series of provisions, each of which consists of the left side, followed by the metacharacter "::=", and then a list of right parts. The left side is the name of the constituent, and the right parts, separated by the metacharacter |, are strings containing terminal characters or constituent names, or both.

Table 4

BNF metasymbols

Metasymbol	Equivalent in natural language	Application
::=	is by definition	Separates a definable concept from its definition
|	Or	Separates alternative phrase definitions
<character string>	character string	Means that characters occurring should be treated as a whole

Although any immediate constituent grammar can be written in BNF, the Backus notation is not a complete metalanguage – some sets of strings (languages) cannot be specified using immediate constituent grammars. However, in practice this does not preclude the construction of the language. The properties that need to be communicated to the language are often structural (grammatical) in nature, and since the class of immediate constituent languages (languages defined by immediate constituent grammars) is large, we have sufficient freedom to choose one of them.

It should be noted that full grammars for natural languages are unusually complex. Therefore, to set the grammar of the natural language for the natural language description of scenarios is impractical. To formalize scenarios, a different path can be chosen. In the description of the script in natural language, it is necessary to highlight key terms and relationships between them, and then present the selected set of concepts and relationships in the BNF form. By the way, the selection of basic concepts and relations can be carried out in the process of constructing the ontology of scenario modeling [26].

Consider the use of the context-free grammar formalism to build a language with which you can describe scenarios of the behavior of conflicting parties in the security system.

In the given fragment of the grammar, each rule describes possible options for substituting the logical formulas written in the right part (after the arrow) as the value of the statement contained in the left part. Alternative substitution methods are possible. The comma present in the right parts of the grammar rules corresponds to the conjunction operation.

Let us start with a description of the main term – the script. By a scenario we mean a sequence of interrelated events that can take place under certain conditions. Between events, there are causal relationships that can be represented by rules written in the language of logic. The script is synthesized using a knowledge base containing a description of the script elements and the relationships between them. The result of the synthesis is many possible scenarios, the quality and reliability of which depend on the source information.

Scenario description in the BPF form is as follows:

<scenario>::=<list_of_events>.

From a BNF point of view, a script is the initial symbol of grammar and is defined as a recursive sequence of events. Events can be interpreted as actions that can occur when certain conditions are met or unconditionally.

<list_of_events>::=<event>|<event><list_of_events>
<event>::=<factor_action>|<subject_action>.

The action of some factor or active subject that determines the event is written as follows:

<factor_action>::=<factor>|<condition><factor>
<factor>::=price_of_attack|Dismissal_time|
Activate_Uncertainty|Base_Reputation|
Information_Sharing|Time_reputation_loss|
Time_to_build_up_reputation|Time_to_report_atack.

It should be noted that terms that are not enclosed in angle brackets appeared on the right side of the definitions. Such terms are terminal, that is, they define themselves and do not require further elaboration and definition.

The parties to cyber conflict are defined by terminal symbols in the same way:

<subject>::=attacker|defender|user.

However, if it is necessary to detail the features and nature of the side of cyber conflict, the corresponding term is transformed into nonterminal (enclosed in angle brackets), and its definition should be given below.

It is proposed to determine the actions of subjects as follows:

<subject_action>::=<subject><factor>
<condition><action>|<subject><condition>
<action>|<condition><subject><action>.

Note that on the right side there are two different lines <subject><condition><action> and <condition><subject> <action> (that is, the same terms are given in different sequences). This is not a violation of the rules of description and when checking grammar does not lead to an error message.

As for the possible actions of the attacker and the defender, they should be determined by a well-developed threat classifier, which should offer the most effective means of counteracting each of the threats.

<action>::=threat_1|treat_2|treat 3.

An introduction to the description of the conflict of a nonterminal symbol condition allows you to limit arbitrary combinations of actions that do not correspond to the semantics of the script.

<condition>::=<factor><logical sign><value>.

The described grammar used to represent the scripts was presented as an input file for the context-free grammar analyzer program developed on the Delphi platform. It should be noted that for a visual representation of both the input grammar and the output results, the grammar was presented in an abridged form, shown in Fig. 3.

First result. The resulting grammar analysis of scenario description is a representation of the scenario grammar in the usual (as accepted in the BPF) form, presented in Fig. 4.

The analysis program generates a list of terminal and nonterminal symbols (Fig. 5). This separation is useful for scenario analysis. Indications are also generated about which symbol is the target.

The generation of the following table is due to the fact that a contextually free language cannot be analyzed in one pass and implies the use of recursive parsing methods, the intermediate result of which is to put intermediate results into stack (Fig. 6).

The following lists the triples of characters that are generated during the verification process, and the table of constructed characters can be used in the next steps to generate script description (Fig. 7).

The program ends with detailed diagnostics for each terminal and non-terminal symbol. The final message about the absence of errors in the grammar gives confidence that the description of the prototype is made up correctly and descriptions of the script-instances can be generated according to the proposed template.

```
<scenario><list_of_events>
<list_of_events><event>
<event><list_of_events>
<event><factor_action>
<subject_action>
<factor_action><factor>
<condition><factor>
<factor> price_of_attack
Dismissal_time
Activate_Uncertainty
Base_Reputation
Information_Sharing
Time_reputation_loss
Time_to_build_up_reputation
Time_to_report_atack
<subject> attacker
defender
user
<subject_action><subject><factor><condition><action>
<subject><condition><action>
<condition><subject><action>
<action> threat1
threat2
threat3
<condition><factor><logical sign><value>
```

Fig. 3. Presentation of input grammar for CF grammar analysis program

```
PRODUCTIONS
 1 <scenario>::=<list_of_events>
 2 <list_of_events>::=<event>
 3                   |<event><list_of_events>
 4 <event>::=<factor_action>
 5         |<subject_action>
 6  <factor_action>::=<factor>
 7                   |<condition><factor>
 8  <factor>::=price_of_attack
 9           |Dismissal_time
10           |Activate_Uncertainty
11           |Base_Reputation
12           |Information_Sharing
13           |Time_reputation_loss
14           |Time_to_build_up_reputation
15           |Time_to_report_atack
16  <subject>::=attacker
17           |defender
18           |user
19
<subject_action>::=<subject><factor><condition><action>|<subject><condition><a
ction>|<condition><subject><action>
22  <action>::=threat1
23           |threat2
24           |threat3
25  <condition>::=<factor><logical sign><value>
```

Fig. 4. Listing of scenario description grammar in the BNF form (fragment)

	TERMINAL SYMBOLS		NONTERMINALS
1	_|_	17	<event>
2	user	18	<factor>
3	threat1	19	<action>
4	threat2	20	<subject>
5	threat3	21	<scenario>
6	<value>	22	<condition>
7	attacker	23	<factor_action>
8	defender	24	<list_of_events>
9	Dismissal_time	25	<subject_action>
10	price_of_attack		
11	Base_Reputation		
12	Information_Sharing		
13	Activate_Uncertainty		
14	Time_reputation_loss		
15	Time_to_report_attack		
16	Time_to_build_up_reputation		
	<scenario> is the goal symbol		

Fig. 5. List of terminal and non-terminal symbols

C1 matrix for stacking decision

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	_|_		Y					Y	Y	Y		Y	Y	Y	Y	Y	Y	Y
2	user			N	N	N				N		N	N	N	N	N	N	N
3	threat1			N	N	N	N				N		N	N	N	N	N	N
4	threat2			N	N	N	N				N		N	N	N	N	N	N
5	threat3			N	N	N	N				N		N	N	N	N	N	N
6	<value>		N	N	N	N		N	N	N		N	N	N	N	N	N	N
7	attacker			N	N	N				N		N	N	N	N	N	N	N
8	defender			N	N	N				N		N	N	N	N	N	N	N
9	Dismissal_time	N	N					N	N	N	N	N	N	N	N	N	N	N
10	<logical sign>					Y												
11	price_of_attack	N	N					N	N	N	N	N	N	N	N	N	N	N
12	Base_Reputation	N	N					N	N	N	N	N	N	N	N	N	N	N
13	Information_Sharing	N	N					N	N	N	N	N	N	N	N	N	N	N
14	Activate_Uncertainty	N	N					N	N	N	N	N	N	N	N	N	N	N
15	Time_reputation_loss	N	N					N	N	N	N	N	N	N	N	N	N	N
16	Time_to_report_atack	N	N					N	N	N	N	N	N	N	N	N	N	N
17	Time_to_build_up_reputation	N	N					N	N	N	N	N	N	N	N	N	N	N
18	<event>	N	Y					Y	Y	Y		Y	Y	Y	Y	Y	Y	Y
19	<factor>	N	N					N	N	#	Y	#	#	#	#	#	#	#
20	<action>	N	N					N	N	N		N	N	N	N	N	N	N
21	<subject>			Y	Y	Y				Y		Y	Y	Y	Y	Y	Y	Y
22	<scenario>	N																
23	<condition>		Y	Y	Y	Y		Y	Y	Y		Y	Y	Y	Y	Y	Y	Y
24	<factor_action>	N	N					N	N	N		N	N	N	N	N	N	N
25	<list_of_events>	N																
26	<subject_action>	N	N					N	N	N		N	N	N	N	N	N	N
	Table entries summary: 155 49 Y 230 N 8 #																	
	The maximum depth of recursion was 7 levels. 378 sentential forms were examined.																	

Fig. 6. Table C1 (placing intermediate results into stack)

```
C2 production choice function:
user as stack top will cause productions to be checked in this order:
   18  <subject>::=user
       There will be no context check.
threat1 as stack top will cause productions to be checked in this order:
   22  <action>::=threat1
       There will be no context check.
threat2 as stack top will cause productions to be checked in this order:
   23  <action>::=threat2
       There will be no context check.
```

Fig. 7. Diagnostics of the grammar production selection function (fragment)

7. Discussion of scenario grammar analysis results

Using the program of grammatical analysis of the BNF description of the scenario of behavior of the opposing agents of the security system under conditions of cyber conflict allowed us to obtain results that are consistent with the objectives of the program.

First of all, the use of the analysis program is aimed at helping to debug the BNF grammar so that it can be used in the parsing algorithm. The second purpose is to prepare the tables required for subsequent use in the system of automatic generation of scripts-instances, based on the formal description of the prototype scenario.

The analysis of the prototype scenario is carried out according to the results of diagnosis of the BNF description at all stages of processing the formal presentation of the grammar. The restoration of the familiar form of the BNF grammar description shows the absence of syntactic and lexical errors in the task of the analyzed grammar (Fig. 4). The following output

of the tables of terminal and nonterminal grammar symbols (Fig. 5) allows us to verify that the creator of the script correctly defined those concepts of the script that are finite, undetectable (terminal), and those whose definition should be formed. Since in the considered example of the BNF description of the scenario, only a part of the grammar is presented, it is necessary to pay attention to the concepts that fall into the category of terminal. With further expansion of the grammar, for some of them, definitions should be given, as a result of which they will fall into the nonterminal table, and possibly new ones will appear in the table of terminal symbols. Indication of the target symbol "script" indirectly confirms the correctness of description formation, since this concept does not participate anywhere in the definitions of other concepts (it does not appear on the right side of the BNF description of the grammar).

The table of the sentimental set constructed below represents the set of lines (sentimental forms) generated by the initial symbol. The result shows that 378 sentential forms were verified, which can be considered as possible instance scripts, constructed according to the rules presented in the BNF description of the scenario. As noted, for the selected type of grammar, it is impossible to analyze it in one pass and the use of recursive methods is required. This is reflected in the conclusion of the program that the depth of recursion during grammar analysis reached 7 levels. This value should be recognized as such, which makes the manual analysis of scenario description difficult. And also it should be noted that none of the methods mentioned in the paper allowed to perform this type of analysis.

The verification of the script grammar ends with information on the formation of two tables – $C1$ and $C2$. Table $C1$ (Fig. 6) indicates for each character (row of the table) the possibility of non-recursive continuation of grammar analysis upon receipt of the next character (column). The Y symbol indicates the need for recursion during arnalysis and placing the symbol into stack, N indicates the possibility of continuing non-recursive parsing, and the absence of any symbol at the intersection of a row and a column indicates that these two symbols cannot appear in one rule of BNF grammar description (49 pairs of symbols when used to describe the grammar, recursive analysis algorithms are required). Table $C2$ (Fig. 7) indicates the rule that should be used to perform the analysis, as appropriate.

In general, the proposed approach has shown its operability, it allowed to obtain the necessary information about the description of scripting instances. The description of the prototype script obtained and verified from the point of view of correctness of the context-free grammar, as well as the syntax tables $C1$ and $C2$ constructed, can be used in an automatic script generation system. Since the BNF form of presentation of context-free grammar does not impose any restrictions on the subject area, the scope of the presented approach can be quite wide.

As limitations inherent in the proposed method, the following should be indicated:

1. The used context-free grammar and Backus-Naura form of its representation in general allow not the only interpretation of the description. Because of this, when generating scripts-instances, it is possible to form several forms of script representation. This can be interpreted as the presence of a certain redundancy in the system for description analysis and scenario generation.

2. The used scenario description form (BNF) is quite formalized and sets strict restrictions for representing the described subject area. Because of this, a certain BNF pattern limits the analyst's capabilities in the field of scenario modeling. And when it becomes necessary to create a unique script, it is necessary to redefine the script description template itself and re-verify its correctness.

As directions for the development of the study, the following can be formulated:

1. Further improvement of the tools for describing and analyzing scenarios for the automatic generation of scripts-instances from the generated syntactic tables of grammar analysis.

2. To increase the convenience and compactness of behavior scenario description, to implement support for the extended form of metalanguage (BNF), thereby supporting the desire of analysts in the field of scenario modeling to describe projected scenarios in their own way, possibly in a more familiar way.

8. Conclusions

1. A naturally linguistic description of the scenario of participants' behavior in a real cyber conflict implementing the "wait and see" tactics for defenders, and the weakest link tactics for attackers is given.

2. Existing grammars are analyzed and the most appropriate grammar is selected that matches the scenarios of behavior of the participants in cyber conflict. Such a grammar is a context-free grammar, which best describes the scenario of participants' behavior in cyber conflict.

3. It is shown how the natural language description of a scenario can be converted into the BNF form, which can be considered as the initial description of a scenario for the program for analyzing the correctness of prototype scenario description. The BNF representation of the script is given, corresponding to the natural language description of the script. The formal grammatical representation of the script is checked. Conclusions are made about the efficiency of the proposed approach.

References

1. Do Prado Leite, J. C. S., Hadad, G. D. S., Doorn, J. H., Kaplan, G. N. (2000). A Scenario Construction Process. Requirements Engineering, 5 (1), 38–61. doi: https://doi.org/10.1007/pl00010342
2. Carroll, J. (1995). Introduction: the scenario perspective on system development. Scenario-based design: envisioning work and technology in system development. Wiley, 1–18.
3. Potts, C. (1995). Using schematic scenarios to understand user needs. Proceedings of the Conference on Designing Interactive Systems Processes, Practices, Methods, & Techniques - DIS'95, 247–256. doi: https://doi.org/10.1145/225434.225462
4. Booch, G. (1992). Object oriented design with applications. Object-oriented software engineering: a use case driven approach. Addison-Wesley, Reading, MA/ACM Press, New York.

5. Zorman, L. (1995). Requirements envisaging by utilizing scenarios (Rebus). University of Southern California.
6. Rolland, C., Ben Achour, C., Cauvet, C., Ralyté, J., Sutcliffe, A., Maiden, N. et. al. (1998). A proposal for a scenario classification framework. Requirements Engineering, 3 (1), 23–47. doi: https://doi.org/10.1007/bf02802919
7. Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P. (1998). Scenarios in system development: current practice. IEEE Software, 15 (2), 34–45. doi: https://doi.org/10.1109/52.663783
8. Ryser, J., Glinz, M. (1999). A Practical Approach to Validating and Testing Software Systems Using Scenarios. Proceeding 3rd International Software Quality Week Europe 1999 QWE'99.
9. Devillers, F., Donikian, S. (2003). A Scenario Language to orchestrate Virtual World Evolution. SCA '03 Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, 265–275.
10. Godet, M., Roubelat, F. (1996). Creating the future: The use and misuse of scenarios. Long Range Planning, 29 (2), 164–171. doi: https://doi.org/10.1016/0024-6301(96)00004-0
11. Schwartz, P. (2000). The official future, self-delusion and the value of scenarios. Financial Times, 2.
12. Steinitz, C., Arias, H., Bassett, S., Flaxman, M., Goode, T., Maddock III, T. et. al. (2003). Alternative Futures for Changing Landscapes: The Upper San Pedro River Basin in Arizona and Sonora. Island Press, New York.
13. Timpe, C., Scheepers, M. J. (2003). A look into the future: scenarios for distributed generation in Europe. Energy research Centre of the Netherlands ECN.
14. Maack, J. (2001). Scenario analysis: a tool for task managers. In: Social Development Paper No. 36. Social Analysis: Selected Tools and Techniques. World Bank, Washington, D.C.
15. Hulse, D. W., Branscomb, A., Payne, S. G. (2004). Envisioning alternatives: using citizen guidance to map future land and water use. Ecological Applications, 14 (2), 325–341. doi: https://doi.org/10.1890/02-5260
16. Anderson, R., Moore, T. (2006). The Economics of Information Security. Science, 314 (5799), 610–613. doi: https://doi.org/10.1126/science.1130992
17. Varian, H. (2004). System Reliability and Free Riding. Advances in Information Security, 1–15. doi: https://doi.org/10.1007/1-4020-8090-5_1
18. Bohme, R., Moore, T. (2009). The iterated weakest link. A model of adaptive security investment. Workshop on the Economics of Information Security (WEIS). Available at: http://weis09.infosecon.net/files/152/paper152.pdf
19. Gordon, L. A., Loeb, M. P., Lucyshyn, W. (2003). Information security expenditures and real options: A wait-and-see approach. Computer Security Journal 14, 1–7.
20. Purser, S. A. (2004). Improving the ROI of the security management process. Computers & Security, 23 (7), 542–546. doi: https://doi.org/10.1016/j.cose.2004.09.004
21. Grossklags, J., Christin, N., Chuang, J. (2008). Secure or insure? Proceeding of the 17th International Conference on World Wide Web - WWW '08, 209–218. doi: https://doi.org/10.1145/1367497.1367526
22. Gordon, L. A., Loeb, M. P. (2002). The economics of information security investment. ACM Transactions on Information and System Security, 5 (4), 438–457. doi: https://doi.org/10.1145/581271.581274
23. Zhang, Y., Fan, X., Wang, Y., Xue, Z. (2008). Attack Grammar: A New Approach to Modeling and Analyzing Network Attack Sequences. 2008 Annual Computer Security Applications Conference (ACSAC). doi: https://doi.org/10.1109/acsac.2008.34
24. Gorodetski, V., Kotenko, I. (2002). Attacks against Computer Network: Formal Grammar-Based Framework and Simulation Tool. Recent Advances in Intrusion Detection, 219–238. doi: https://doi.org/10.1007/3-540-36084-0_12
25. Yevseyev, S. P., Dorokhov, A. V. (2011). Information threats and safety in Ukrainian bank payment systems. Kriminologicheskiy zhurnal baykal'skogo gosudarstvennogo universiteta ekonomiki i prava, 2 (16), 68–75.
26. Milov, A. V., Korol', O. G. (2019). Razrabotka ontologii povedeniya vzimodeystvuyushchih agentov v sistemah bezopasnosti. 4th International Congress on 3D Printing (Additive Manufacturing) Technologies and Digital Industry 2019, 832–842.