Policy-Based Proactive Monitoring of Security Policy Performance

Vitaly Bogdanov and Igor Kotenko

Computer Security Research Group, St. Petersburg Institute for Informatics and Automation
39, 14 Liniya, St.-Petersburg, 199178, Russia
{bogdanov,kotenko}@comsec.spb.ru

Abstract. One of topical tasks of policy-based security management is checking that the security policy stated in organization corresponds to its implementation in the computer network. The paper considers the suggested approach to proactive monitoring of security policy performance and security mechanisms functioning. This approach is based on the different strategies of automatic imitation of possible users’ actions in the computer network, including exhaustive search, express-analysis and generating the optimized test sequences. It is applicable to different security policies (authentication, authorization, filtering, communication channel protection, etc.). The paper describes stages, generalized algorithms and main peculiarities of the suggested approach and formal methods used to fulfill the test sequence optimization. We consider the generalized architecture of the proactive monitoring system “Proactive security scanner” (PSC) developed, its implementation and an example of policy testing.

Keywords: Security policy, monitoring, test sequence optimization.

1 Introduction

On the computer network exploitation stage the security administrators need to check the correspondence of the security policy formulated on the design stage to its implementation in the computer network. This task is equivalent to the monitoring of correct performance of security tools and services deployed.

To monitor the functioning of security tools and services on the exploitation stage a passive approach is being used as a rule. According to this approach the current system configuration is being periodically compared with the configuration that was installed (by analyzing the settings of operating systems and applications). The differences found say that security policy does not hold true. However such approach can not guarantee correct security policy fulfillment. The complex distributed network system requires a huge quantity of such settings. Not all of them can be under control. The malefactors can bypass given settings or change them dynamically by injecting and using particular applications. As a result the settings of software and hardware may differ from the security policy accepted.

The paper proposes a common approach to the proactive monitoring of security mechanisms. The approach suggested consists in modeling and imitating user actions in the network and evaluating the results of these actions. When using this approach
the administrator acts mainly as observer: he takes a part in the system administration only when the serious violation of the security policy is revealed or its participation is needed to make an important decision. The suggested approach to proactive monitoring is similar to active vulnerability analysis or penetration testing. Their difference is that during proactive monitoring we do not apply exploits in the tested network. All user actions are usual actions and their fulfillment is controlled by the security policy.

The paper is structured as follows. Section 2 analyzes relevant works and describes the essence of the suggested approach. The common monitoring technique and algorithms applied for different security policies are represented in section 3. Section 4 considers the approach features and restrictions as well as formal methods used for test sequence optimization. Section 5 describes the architecture and implementation of the proactive monitoring system developed, as well as an example of policy testing. Conclusion looks round the paper results and further research directions.

2 Related Work and the Approach Suggested

Existing works on security policy, in particular on security policy monitoring, form the necessary basis for the paper. Russel and Gangemi [21] affirm that one of the necessary conditions of system security is users’ actions monitoring and checking the correspondence of actions to the security policy. Marriot and Sloman [17] consider the issues of monitoring the network events and creating the policies to react on events. In [3, 6, 15, 20, 22, 25] the necessity of security policy monitoring is also noted. Carney and Loe [7] suggest using monitoring to select a stronger policy when the current is violated. Chosh et al. [12] consider active tries to violate security policy as possibility to form a security policy for vulnerable application. Gama and Ferreira [11] suggest the methods of policy detailed elaboration to monitor events in information systems. Beigi et al. [4] use network configuration checking to confirm a policy correspondence. Agrawal et al. [2] consider a security policy monitoring system that checks the correctness of configuration changes. Strembeck [24] describes connection between policy rules, user behavior scenarios and user aims.

The proactive monitoring works of Sailer et al. [23], El-Atawy et al. [9] and Wheeler [26] are the closest to our paper.

Sailer et al. [16] use the proactive monitoring approach for testing the IPSec protocol. Authors consider a simple IPSec based VPN realization and all its possible violations. To check that network hosts are configured properly the authors suggest to place on the one host of VPN-connection a network packet generator and on the second - the tool to capture and check the incoming packets. The generator forms ICMP packets and sends them to the host on the opposite connection point. The packet capturing and checking tool captures the packets before they are processed by the network protocol stack and checks if the packets were formed (secured) correctly. Such scheme allows to quickly reveal the incorrect IPSec configurations.

El-Atawy et al. [9] analyze two methods of the proactive firewall testing. The naive approach to testing is based on sending the random packets to the firewall. The second approach is more effective from the point of view of network resource usage. The specification of each rule contains the description of packets sets that this rule affects.
One can determine all possible intersections of such sets. It is suggested to analyze the result of sending one packet for each such intersection.

Wheeler [26] describes a distributed proactive system for firewall testing. He suggests a formal language for specifying the filtering rules and the architecture of the testing system. This system contains three types of modules (Prober, Manager and Wizard) that are connected in a tree-like structure. Probers are tree leafs, Wizard is the tree root and managers are between the root and leafs. Tester is the simplest module. It listens to some network port or sends packets to another tester. Manager controls testers and managers in its sub tree. It sends commands to subordinate modules and transmits answers upward on the tree-like structure. Wizard interprets the policy description, creates a testing plan, sends commands to subordinate managers, receives and analyses testing results and provides the user interface. Two testers are used to test one connection; they try to create connections using TCP or UDP protocols.

A lot of different commercial systems are being used to manage security policies nowadays. Although all these tools include the components of audit and monitoring of security mechanisms, they mainly targeted on the integration and correlation of security events and an integrated reporting about defended system state.

The approach to security policy monitoring suggested in the paper is based on active imitation of users’ actions (as permitted as well as prohibited by the security policy) and on determining the differences between actual system reactions and reactions that are corresponds to the security policy. In existent works this approach was not researched enough. In difference from most relevant papers [9, 23, 26], where monitoring is used to check IPSec and filtration rules, in this paper we suggest the common approach to proactive monitoring of security policy which can be used for different policies (authentication, authorization, filtering, channel protection, etc.). According to the approach suggested the tested system behavior that is initiated by the generated actions is being compared with the behavior of standard system [5]. As a standard a formal model of the security policy is used. This model is built on basis of the security policy rules and network configuration described on special specification languages.

3 Main Stages and Techniques

To test the security policy of the computer network completely one need to model and plan all possible sequences of users’ actions, perform these actions and compare network responses with expected results. Such testing can not be performed manually; furthermore it may take too much time. To make this task be performable in practice it is needed to systematize the testing procedure, introduce some restrictions for the monitoring process and suggest mechanisms that can make checking more effective.

The suggested approach is based on the independent checking of different rule categories. The every category (or policy) - authentication, authorization, filtering, channel protection, etc. - is implemented using different security facilities. Filtering policy is implemented in the network using firewalls, but authentication, authorization and channel protection policies - as network services.

We suppose that all these facilities do not change their state after they perform a user’s action, i.e. the sequence of actions does not affect their results. In this
condition, it is sufficient to check only results of performing each action separately. Such task statement reduces the task complexity and testing time.

To form a list of all possible actions for each policy category, the proactive monitoring system needs information about the policy and the network configuration. This information is input for monitoring. After receiving this information the monitoring system builds the models of the security policy and the network — this is the first stage of the system operation. On the second stage the system generates the sequence of test impacts. This sequence can be formed by various ways for different policy categories. The third stage is optimizing the sequence of test impacts. The fourth stage consists in performing the test impacts in the network and receiving the results of impacts. The fifth (the last) stage is generalizing the monitoring results and forming the report. The results of each action should be analyzed taking into account the policy category of the action.

To check the security policy rules completely, it is needed to check success (or failure) for all user operations on all assets with all user system accounts. Such approach (based on exhaustive search) is a very labor-consuming, and does not always allow check up the policy for an adjusted time. There are other approaches that have no lacks of exhaustive search. They can approve with a certain probability that policy holds true. For example, the “express-analysis) approach checks only for one random object from the class. Such approach implies essential decrease of checked operations. When the rule uses an object of some class, we check up rule only for one random object from this class. In this case we can have a high false negative rate of deviations. The other approach verifies only several objects of the class. The quantity of such representatives depends on quantity of objects in the given class. Thus, exhaustive search is the most laborious. The next is the choice of several random representatives. And the least laborious approach is the choice of one random representative.

The generalized algorithm of authorization policy checking has the following operations: (1) Look through all users; (2) Look through all privileges; (3) Look through all scanners; (4) Check up the presence of the given privilege for the given user from the given scanner. There are four possible check outcomes: (1) If there is an operation permissive rule and the operation was executed successfully then the policy is not broken; (2) If there is an operation permissive rule and the operation was executed with failure then the policy is broken; (3) If there is no operation permissive rule and the operation was executed with failure then the policy is not broken; (4) If there is no operation permissive rule and the operation was executed successfully then the policy is broken. The fourth violation is the most serious violation. In this case the user can execute operation which is actually forbidden by the policy. Another type of violation is the second violation (user can not execute operation permitted by the policy).

The generalized algorithm of authentication policy checking includes the following operations: (1) Look up through all users; (2) Look up through all credentials; (3) Look up through all services; (4) Look up through all operations; (5) Check up the success of the given operation by the given user on the given server without authentication; (6) Check up the success of the given operation by the given user on the given server with the given authentication method.

The generalized algorithm of filtering policy checking contains the following operations: (1) The checked connections set is empty; (2) Look up through all firewall rules; (3) Assign connections set from current rule to the connection set for checking;
(4) Subtract checked connections set from the connections set for checking; (5) Get the random connection from the set of the connections for checking; (6) Look up for the scanner to send packets through firewall; (7) Look up for the scanner or service to receive packets beyond firewall; (8) Check up the possibility of connection establishing between sending scanner and receiving scanner or service; (9) Add connections for checking set to the checked connections set.

The generalized algorithm of channel protection checking has the following operations: (1) Look up through all servers and ports; (2) Look up through all channel protection techniques; (3) Check up the possibility of the connection to the given host without channel protection; (4) To check up the possibility of the connection to the given port of the given host using the given channel protection technique. Note that if a channel protection policy rule uses transport level channel protection protocols (such as IPSec), an application can not choose whether it uses channel protection or not, because the channel protection using depends on the system configuration. To check the protection of transmitted data on the transport level we need to use the following algorithm: (1) Look up through all servers; (2) Look up through all target hosts and ports; (3) Look up through all channel protection techniques; (4) Check up the possibility of connection to the given server on the given port; (5) Check up if the channel protection on transport level was used.

4 Restrictions and Optimization Approaches

There are several important aspects of the security policy monitoring. Let us consider three such aspects:

(1) performing the “dangerous operations”,
(2) possibility of conflicts with working users,
(3) the necessity to use an authentication database.

The dangerous operations are operations that can lead to information integrity violations. For example, for file system such operations are removing or changing a file. To prevent integrity violations as a result of such operations one need take care about the backing up the information before performing the changes. The other way to bypass the dangerous operations is introducing several modes of system operation. One of the modes allows complete testing with dangerous operations included. The other mode allows all operation testing except dangerous. Dangerous operations are most laborious because they require performing the additional actions.

The second important aspect of the proactive monitoring is a possibility of conflicts with other information processing operations and users. Such conflicts may arise as a result of almost any operation both dangerous and safe from the integrity point of view. For instance, reading access to a file can be denied because this file is locked for editing by other user. A possible solution of this problem is introducing a special time to check the network when users do not work or work rarely. The disadvantage of this solution is impossibility to check the rules that have time restrictions. The other solution is to try to determine the conflict reason and use it during checking.

The third important aspect is a need to have a database of authentication data for all users. This aspect is stipulated by the fact that before checking the user’s privileges
for an operation on an object it is necessary to be authenticated in the system. This aspect undoubtedly reduces the applicability of the proactive monitoring.

The proactive monitoring is a very resource-intensive task due to both a large input data variety and a long time needed to receive response from the network. To decrease the testing time we need to optimize the test impacts.

We can optimize the test impacts by different ways: (1) remove the superfluous test impacts; (2) find the optimal impacts subsequence; (3) generate the impacts subsequences that can be performed simultaneously.

Due the limits of the paper, let us consider the procedure of test impacts optimization on an example of filtering policy testing.

To describe optimization methods we need to give a formal description of filtering policy. The total network filtering policy is a set of policies for all firewalls in the network. The methods of removing the superfluous test impacts and optimizing the test impacts sequence that are described below should be applied to each firewall policy.

The filtering policy of each firewall represents a set of filtering rules. Every firewall rule \( R_i = (P_i, A_i) \) determines a set \( P_i \) of network packets, that rule affects, and an action \( A_i \in \{allow, deny\} \) that has to be applied to the packets from \( P_i \). Furthermore it is essential to take into account the ordering of rules because the packets from \( P_i \) may intersect for different rules.

Let us denote the firewall policy as an ordered set of the firewall rules by \( FP_f = \{R_i\}_{i=1}^{N_f} \), where \( N_f \) - the quantity of rules in the policy of the firewall \( f \).

The test impact for the filtering policy represents sending the network packet over the firewall.

To test the filtering policy completely we need to send all possible packets over the firewall and compare that they were passed in compliance with the policy. Such testing will take a long time. We can send one packet for every rule, but this is not a good decision because the packets from \( P_i \) may intersect for different rules.

To remove superfluous impacts it is possible to use the approach suggested El-Atawy et al. [9]. The main idea of the approach named policy segmentation is to break the packets sets \( \{P_i\}_{i=1}^{N_f} \) for all rules on their disjunctive subsets \( S = \{S_j\}_{j=1}^{M} \) of so-called segments. The segment is such subset of the packets sets from one or several rules that all packets from the segment are related to the same rules set. Authors show that it is enough to send over firewall one packet from each segment to test all possible combinations of rules.

The sending of only one packet from each segment reduces the number of test impacts. But there is a possibility of additional optimization by selecting the optimal sequence of test impacts.

Let we have two rules \( R_1 \) and \( R_2 \) such that \( P_{12} = P_1 \cap P_2 \neq \emptyset \) and \( A_1 = allow, A_2 = deny \). We sent packet \( p_{12} \in P_{12} \) first and revealed that it does not arrive at destination point. Consequently the rule \( R_1 \) does not hold and it is not
needed to send the packet \( p_1 \in P_1 \setminus P_2 \). In other side, if we send the packet \( p_1 \) first we will need to send packet \( p_{12} \) in any case to verify the rule \( R_2 \).

The goal of the impacts sequence optimization is to create such packets sequence that will reveal all deviations from security policy by a minimal number of steps.

Let us use binary test questionnaire optimization theory described in [1] and define the set of events as a set of binary vectors \( E = \{ \{ e_i \}_{i=1}^{N_f} \} \).

Let \( e_i = 1 \) if a firewall rule works good and \( e_i = 0 \) if the rule does not work. The test impact \( t_j \) is one packet sending from a segment \( S_j \in S \). Each test impact may be fulfilled successfully (the packet was received at the destination point) or unsuccessfully (the packet was filtered by the firewall).

Depending on the result, the test impact divides the set \( E \) of events on two classes: \( E_{t_j} \) - the events corresponding to the successful test impact \( t_j \) and \( \overline{E}_{t_j} \) - the events corresponding to the unsuccessful test impact \( t_j \).

To identify events from the set \( E \) we can use different binary test questionnaires.

The questionnaire is a set \( T \) of questions and the sequence of their asking. The binary questionnaire can be represented as a binary tree, where the nodes of the tree are the subsets of the set \( E \) and every node \( X \subset E \) has sons \( X_t \) and \( X_r \). The leafs of the tree are nodes \( X \subset E \), for which \( X_t \) or \( X_r \) is empty set.

Performing the test actions and moving at the tree we will come into list that contains the set of the events that take a place at the tested system. When we find optimal test questionnaire for the given set of the events and test actions, we find optimal sequence of the actions performing.

It is necessary to define the optimality criteria of the test questionnaire. Let us define such criteria as minimum of the average count of the test actions need to be performed to identify the event from the \( E \) unambiguously:

\[
C = \sum_{i=1}^{N_f} p(y_i) C(y_i),
\]

where \( y_i \in E \) - an event, corresponding to the firewall state, \( p(y_i) \) - the probability of the event \( y_i \), \( C(y_i) \) - the cost of the \( y_i \) identification.

The cost of event identification is defined as follows:

\[
C(y_i) = \sum_{t_k \in \mu(y_i)} c(t_k),
\]

where \( \mu(y_i) \) - a path from the root of the questionnaire to the event \( y_i \), i.e. a set that contains test actions that need to be performed to identify the event \( y_i \); \( c(t_k) \) - the cost of the question \( t_k \).
Let the costs of every question is 1, then $C(y_i)$ will be as follows:

$$C(y_i) = |\mu(y_i)|.$$  

Let also all events have equal probability then the average cost of the questionnaire will have the following form:

$$C = \frac{1}{N_f} \sum_{i=1}^{N_f} |\mu(y_i)|.$$  

So we can obtain binary test questionnaire where test impacts correspond to questions. If we optimize this questionnaire we will obtain optimal test impacts sequence.

Let us use dynamic programming method for the optimization of binary test questionnaires with the incomplete set of questions.

To describe this method we need to define the concept of situation. Let for the set $E$ of events the set $T$ of questions is defined. Then $E_T$ is a set of subsets $L_\alpha$ on which the sequences of questions from $T$ divide $E$. For each $L_\alpha \subset E_T$ there are a subset of questions $T_\alpha$ that have a sense relative to $L_\alpha$, i.e. these questions divide $L_\alpha$ on two nonempty subsets. The pair $(L_\alpha, T_\alpha)$ is named as a situation, $m_\alpha = |L_\alpha|$ is a situation degree.

Let for each possible situation with degree $m_\alpha$ the corresponding optimal sub questionnaire has been built. Then, using these sub questionnaires, we can build an optimal sub questionnaire for the situation $(L_\beta, T_\beta)$ with the degree $m_\beta > m_\alpha$.

The Bellman’s optimality equation for this case is as follows:

$$C_{opt}(L_\beta, T_\beta) = \min_{t \in T_\beta} \left\{ c(t) + \sum_{n=1}^{2} p_n C_{opt}(L_{\beta_n}, T_{\beta_n}) \right\},$$

$$p_n = \sum_{y \in L_{\beta_n}} \frac{p(y)}{\sum_{y' \in L_\beta} p(y')}$$

where $c(t)$ is the cost of the question $t$, $p(y)$ is the probability of the event $y$, $\beta$ is the conditional probability, $L_{\beta_n}$ are the subsets on which question $t$ divides the set $L_\beta$ $(n = 1, 2)$. Since the costs of questions are equal, we can set them as 1. We will also consider that the probabilities of all events are equal, then the equation will be as follows:

$$C_{opt}(L_\beta, T_\beta) = \min_{t \in T_\beta} \left\{ 1 + \sum_{n=1}^{2} \frac{|L_{\beta_n}|}{|L_\beta|} C_{opt}(L_{\beta_n}, T_{\beta_n}) \right\}.$$  

This equation is the base of the dynamic programming algorithm [1].

The dynamic programming algorithm for the test impacts optimization is as follows:
1. Set $m_\beta = 1$.
2. For each possible situation $(L_\beta, T_\beta)$ with degree $m_\beta$ build optimal sub questionnaire according to Bellman’s equation using sub questionnaires built on the previous steps for situations with degree $m < m_\beta$.
3. If $m_\beta \leq N_f$ then increase $m_\beta$ on 1 and return to step 2, in other case the optimal questionnaire has been built.

The dynamic programming method allows finding the optimal test packet subsequence for firewall testing. The disadvantage of the dynamic programming method is exponential complexity of the algorithm.

Thus we defined test impacts optimization ways to test filtering policy of the single firewall. To optimize complete network filtering policy we need to find parallel test impacts sequences. Obviously each firewall can be tested independently and different firewalls testing results do not affect each other [26].

To illustrate the presented method let us consider a simple network with a boundary host (see figure 1).

![Fig. 1. The test network with a boundary host](image)

Let us consider the filtering policy of the firewall that consists of three rules: (1) allow TCP connections with the host 193.202.13.3 at the port 80; (2) allow TCP connections with the host 193.202.13.3 at the port 110; (3) deny all other connections.

Every rule allows the particular type of packets, and the last rule denies all packets that were not referred. This rules creation method is typical for the firewall policies.

Let us consider how the test actions optimization algorithm works on the given example. We will note every state of the tested firewall by three letters, for example AbC. The capital letter means that rule works correctly in this state and small letter means that rule does not work. For example, at the state AbC the A and C rules work and B rule does not work.

The set $E$ of the all possible states looks as follows: $E = \{ ABC, ABc, AbC, Abc, aBC, aBc, aBc, abc \}$. The set $T$ of the possible test actions contains three elements: $T = \{ Pa, Pb, Pc \}$, where $Px$ corresponds to connection establishing that is processed by the rule $X$. For instance $Pa$ is the establishing of TCP connection with the host 193.202.13.3 at the port 80 and $Pc$ is the establishing of TCP connection with the host 193.202.13.3 at the port different from 80 and 110.

The set of the subsets that are results of partitioning the $E$ by the questions from $T$ looks as follows: $E_T = \{ E, \{ ABC, ABc, AbC, AbC, aBC, aBc, abc \}, \{ ABC, ABc, AbC, aBC, aBc, abc \}, \{ ABC, ABc, abc \}, \{ ABc, abc \}, \{ ABC, abc \}$. 

\[ \]
AbC, aBC, abC }, { aBC, AbC }, { ABC, AbC }, { ABC, aBC }, { abC },
{ AbC }, {abc }, { ABC } ). The test actions divide the set $E$ into the situations of different
degrees.

Let us consider all situations from the first degree situation up to the eight degree
situation and create optimal test plan. First degree situations: ( { abC } , { } ), ( { AbC } , { } ), ( { aBC } , { } ), ( { ABC } , { } ). Because the test actions for all these
situations are empty then $C_{opt}$ for each first degree situation is equal to 0. Second de-
gree situations: ( { aBC, abC }, { Pb } ), ( { ABC, abC }, { Pa } ), ( { ABC, AbC },
{ Pb } ), ( { ABC, aBC }, { Pa } ). For every situation there is only one test action so
for all situations $C_{opt} = c(t) =1$. Fourth degree situations: ( { AbC, ABC, aBc, abc },
{ } ) и ( { ABC, AbC, aBC, abC }, { Pa, Pb } ). For the first situation the test actions
set is empty, consequently $C_{opt} = 0$. Second situation allows two test actions, conse-
quently $C_{opt} = \min (1 + 2/4 + 2/4, 1 + 2/4 + 2/4) = 2$. The optimal test action is any
from two. Let us assign Pa as optimal test action. Fifth degree situations: ( { ABC,
AbC, Abc, aBc, abc }, { Pc } ). $C_{opt} = 1 + 1/5 * 0 + 4/5 * 0 = 1$. There are two sixth
degree situations and each allows two test actions: ( { ABC, AbC, Abc, aBc,
abc }, { Pb, Pc } ) and ( { ABC, AbC, Abc, aBc, abc }, { Pa, Pc } ). Let us con-
sider $C_{opt}$ for the first situation. $C_{opt} = \min (1 + 1/6 * 0 + 5/6 * 1, 1 + 2/6 * 1 + 4/6 * 
0 ) = 1,33$ and optimal test action is Pc. Similarly for the second situation $C_{opt} = 1,33$
and optimal test action — Pc. Let us assign the first situation as optimal and the opti-
mal test action - as Pc. There is only one eighth degree situation: ( E, { Pa, Pb, Pc } ).
$C_{opt} = \min (1 + 6/8 * 8/6 + 2/8 * 1, 1 + 6/8 * 8/6 + 2/8 * 1, 1 + 4/8 * 0 + 4/8 * 2) = 
\min (2 1/4, 2 1/4, 2) = 2$. The optimal test action is Pc. The optimal test tree that was
created in accordance with the given approach is shown in figure 2.

The average cost of the questionnaire that corresponds to the tree is $C = 1/8 * (1 *
4 + 3 * 4) = 2$. Let us try to calculate the difference between optimal and non-optimal
average number of test operations. Our policy will be typical, i.e. will contain
$N_f - 1$ permissive rules and one prohibitive rule like in the example above. Then the
average cost of the optimal tree can be calculated as follows:

$$C_{opt} = \frac{1}{2^{N_f}} (N_f 2^{N_f-1} + 2^{N_f-1})$$
and the average cost of the non-optimal tree will be:

\[ C_{\text{nopt}} = \frac{1}{2^{N_f}} (N_f (2^{N_f-1} + 1) + (N_f - 1)(2^{N_f-1} - 1)) \].

Then the difference between optimal and non-optimal average costs will be:

\[ C_{\text{nopt}} - C_{\text{opt}} = \frac{1}{2^{N_f}} ((N_f - 2)2^{N_f-1} + 1) = \frac{N_f - 2}{2} + \frac{1}{2^{N_f}}. \]

When \( N_f = 22 \) the difference will be \( C_{\text{nopt}} - C_{\text{opt}} = \frac{20}{2} + \frac{1}{2^{22}} \approx 10 \) operations.

5 System Architecture, Implementation and Experiments

The generalized architecture of “Proactive security scanner” system (PSC) that implements the suggested approach is depicted in figure 3. PSC consists of configurator, scanners, correlator and management console.

![Generalized architecture of PSC](image)

The PSC input is the tested system specification, the tested security policy specification and testing parameters (to manage the process of testing). The PSC output is the report about revealed deviations from the specified security policy in the tested system and their estimation (to answer to the question how critical or crucial these deviations are). The PSC should satisfy the following main requirements: tested system and security policy coverage (completeness), deviations estimation fidelity (adequacy), and productivity.

PSC allows revealing the following security policy violations: Authorization policy - the impossibility of authorized operation, the possibility of forbidden operation;
Authentication policy - the possibility of operation performing by unauthenticated user, the impossibility of operation performing by authenticated user; Filtering policy - permitted connection blocking, forbidden connection possibility; Channel protection policy - unprotected channel possibility, the lack of channel protection.

**Configurator (PSC Config)** plans the test impacts sequence to monitor the security policy performance by forming the tasks for scanners (see figure 4). Its input consists of the specifications of the checked security policy in System Description Language (SDL) and the tested network in Security Policy Language (SPL) as well as the values of testing parameters. The testing parameters can be a subset of security rules to check, a part of the network to test, subjects and objects to test, information about scanners locations in the system, internal information for scanner task generation, etc.

![Fig. 4. Generalized architecture of configurator](image)

**Security policy description language (SPL)** [19] is based on the CIM standard [8] and XML technologies. SPL allows setting the grouping, priority and classification of the policy rules and categories. It contains five rules categories: authentication, authorization (access control), filtering, channel protection (IPSec, SSL/TLS) and operational. Syntax of each rule category is determined by a particular scheme that is based on the XML-scheme. SPL schemes were derived from xCIM-schemes that are the representations of CIM using XML-scheme technology.

**The system description language (SDL)** [19] is applied to describe the network configuration and its functionality. Policy rules must use only such devices, services and functions that are contained in the SDL network description. SDL is based on XML scheme and contains the following main parts: network topology description, i.e. how network nodes are connected, connection type (wired, wireless), cable type, etc; network services description, that are available on corresponding servers and also
operating systems installed; devices and interfaces description, i.e. names, addresses, ports, common purpose description, vendor, etc.

Configurator creates the tasks for scanners by processing input data. For example to check the firewall, configurator forms two tasks: (1) to send a packet from IP address before firewall and (2) to receive the packet at IP beyond firewall.

The generalized algorithm of configurator includes the following steps: (1) Receive the input data; (2) Receive information about scanners locations; (3) Construct the tested system model according to the system specification; (4) Designate scanners locations in the model of the tested system; (5) Construct the tested policy model according to the policy specification; (6) Look through all security policy rules; (7) Select appropriate rule checking method according to kind of the rule and testing parameters; (8) Generate the tasks for the given scanners; (9) Transfer generated tasks to scanners for processing. The details of the algorithm realization differ for each policy class check.

Scanners (PSCs) are components which check a part of the security policy in the tested sub-network. The policy part and system fragment are set by configurator. A scanner task is transferred from configurator to scanner and contains test impact instance. Scanner checks the policy by performing impact and sends check results to correlator.

The generalized algorithm of scanner: (1) Execute impact instance run method; (2) If there are deviations from the rule, add check results and deviations conditions at the scanner report; (3) Transfer check results to correlator.

Correlator (PSC Correlation) receives the check results, analyses them and forms summary reports about the security policy violations revealed in the network.

The generalized algorithm of correlator: (1) Receive check results from scanners; (2) Process and generalize check results to form reports about revealed deviations; (3) Generate and send reports to major system.

Management console allows the security administrator to manage all system components, set input data to configurator and look through the reports of correlator.

To simplify configuring and information transferring, configurator, correlator and management console can be incorporated in one module. To effectively monitor the network at least one scanner should be located at each network segment. In addition, for example, there should be a scanner on a workstation outside the network and a scanner on a workstation which is connected to the analyzed network over modem. If the wireless network is used we need a scanner on the wireless network host.

The main screen of the PSC management console (PSC window) is shown in figure 5. Tabs panel on the main user interface screen allows examining the information that is concerned with current network configuration (Network tab), security policy (Policy tab), scanners’ states (Scanners tab), revealed deviations from the security policy (Report tab) and also system working log (Log tab).

In figure 5 (at right-bottom corner) the checking parameters window (PSC Settings window) is showed. Using these parameters one can determine the dangerous operations performed (Actions field), what policy categories will be tested (Policy field), what will be testing accuracy (Accuracy field) and locations of the scanners (Check from field).
Fig. 5. Test network scheme, security policy and scanners state

During the investigation of the suggested approach the experiments for checking PSC functioning with different network configurations, security policies and monitoring methods were performed.

Let us consider only an example of filtering policy testing in the test network.

During checking, planner looks through all filtering policies (each policy contains all rules for particular firewall) and passes each policy to the scanner task generator. Planner uses appropriate method to check filtering policy completely. Scanner task generator forms the set of the checked connections. Generator looks through all filtering rules from the policy. It calculates set of the connections to test by subtracting from the set of the connections the set of the already checked connections. Then generator gets a connection from the set for checking and searches scanners to test the connection. One scanner is needed to send the packets through firewall and other to receive the packets beyond firewall. If there is no receiving scanner then generator can use existing service to check connection possibility. After checking, the report tab will be generated. Log tab will contain the scanner actions log for the testing period. Performed impact steps can be seen. Scanners tab will show the scanners state changing during testing.

6 Conclusion

The paper considered the suggested approach to the proactive monitoring of the network security policy. The suggested approach is based on the imitation of different users’ actions in the investigated network which can disapprove or approve the fact
that security policy holds true. The particularities, advantages and disadvantages of the suggested approach are determined. The advantage of the proactive monitoring approach suggested is that the action results are similar to the results of actions performed by users. It enables to get the representation on actual system behavior. The system configuration analysis does not give such representation because malefactor that controls the system can violate security mechanisms and (or) replace the analysis results [10, 13]. The disadvantages of the approach suggested are high complexity and low speed of checking in most cases, the restrictions due performing the potentially dangerous actions and the possibility of conflicts with users. Furthermore the proactive monitoring approach will not give results if information about violation is not contained in the policy, for example, if a malefactor leaved in the system a back door which can not be revealed by imitating the regular users’ actions. Nevertheless the approach suggested allows confirming that system users have the privileges that were contained in the policy specification.

The approach is implemented in the proactive monitoring system “Proactive security scanner” (PSC). The main features of implementation are as follows: distributed architecture including multitude of scanners, available in different places of the network; maintaining a centralized automatic configuration of different scanners; gathering data from different scanners located in different places and centralized analyzing the scanning results; mechanism of interpreting and transforming system and policy specifications to the scripts of user actions on evaluating conformity of the current policy and system configuration to the specified security policy and system configuration; mechanism of automatic construction and replaying scripts of user actions taking into account various intentions of users and (or) malefactors.

PSC can work in several modes which are differentiated by the accuracy and checking speed: high accuracy checking (exhaustive search), medium and low accuracy checking, as well as optimized test sequences mode. PSC allows demonstrating the approach efficiently on the examples of filtering, channel protection, authorization and authentication policies. The developed optimization methods allow speeding up of the monitoring process greatly. The suggested approach to the monitoring may be used to fulfill the set of additional testing tasks. For instance it can be applied to imitation of DDoS attacks on the network resources. Such situation imitation can be performed for the server testing purpose. The future research directions are improving the suggested solutions and comprehensive theoretical and experimental evaluation.

Acknowledgments. This research is being supported by grant of Russian Foundation of Basic Research (Project No. 07-01-00547), program of fundamental research of the Department for Informational Technologies and Computation Systems of the Russian Academy of Sciences (contract No 3.2/03), Russian Science Support Foundation and partly funded by the EU as part of the POSITIF project (contract No. IST-2002-002314) and RE-TRUST (contract No. 021186-2).

References