Detection of the Operating System Configuration Vulnerabilities with Safety Evaluation Facility

Peter D. Zegzhda, Dmitry P. Zegzhda, Maxim O. Kalinin

Information Security Centre of Saint-Petersburg Polytechnical University,
P.O. Box 290, K-273, Saint-Petersburg, 195251, Russia

Abstract. In this paper, we address to formal verification methodologies and the system analyzing facility to verify property of the operating systems safety. Using our technique it becomes possible to discover security drawbacks in any IT-system based on access control model of 'state machine' style. Through our case study of model checking in Sample Vulnerability Checking (SVC), we show how the evaluation tool can be applied in Microsoft Windows 2000 to specify and verify safety problem of system security.

1 Introduction

Assurance that system's behavior will not result in an unauthorized access is fundamental to ensuring that the enforcing of the security policy will guarantee system security. The greater the assurance, the greater the confidence that the security system will protect its assets against the threat with an acceptable risk. A fall down occurs in suitably secure commercial operating systems, applications, and network components, especially with respect to security. Commercial offerings have serious security vulnerabilities. The sources of security faults are concealed at abundant errors of the system designing, coding, and administrating processes. The widely known lacks of security are those of programming origin, and they are resolved with regular patches and service packs. At the same time, sophisticated analysis of the well-known operating systems made by the world-renowned organizations, e.g. *CERT* or *Secunia*, testify to the 20 percents of vulnerabilities caused by incorrect security configuring and adjustment arranged by users or administrators. We consider the errors made at the time of security administrating as reasons of variety of *operating system configuration vulnerabilities* (*OSCV*).

The OSCVs take after:

- ignoring the security bulletins, published by vendors or security experts;
- setting the different security adjustments that unobviously may conflict with each other or alternate other settings;
- setting the security settings that may be contradicting to the applied security policy.

The most typical examples of the OSCVs are using of trivial passwords, default security settings, or accidental system folder permissions. For instance, in Microsoft Windows 2000, if there is a shared folder created by administrator, the system grants

D. Zegzhda P., P. Zegzhda D. and O. Kalinin M. (2005).
Detection of the Operating System Configuration Vulnerabilities with Safety Evaluation Facility.
In *Proceedings of the 3rd International Workshop on Security in Information Systems*, pages 263-276
DOI: 10.5220/0002564902630276

'*Full Access*' to new object for '*Everyone*'. If administrator is a novice in security, she could miss such fault and others could access to somebody's private files. Another example is that *Dr.Watson*, the built-in debugger in Microsoft Windows, starts every time after system fault. This program creates a dump file, e.g. *C:\Winnt\user.dmp*. Now imagine that *OE* falls down, and *Dr.Watson* runs. The dump corresponding to *OE* includes all mail accounts and passwords as plain text. Besides this, the NT file system (NTFS) creates a new file with default properties (the default access permissions among them) taken from the parent folder, e.g. *C:\Winnt*. The '*Everyone*' group thus has a '*Full Control*' over the dump file. And consequently, all private email passwords saved in the file becomes open for every user. The Linux-style operating systems obtain the OSCVs of the same sort: for instance, they have a *SUID*-related problem. Such kind of mistakes in protection environment reduces every solid and well-engineered security to 'zero'.

To eliminate the OSCVs, administrator has to know the operating system on-the-fly, observe its structure permanently, and analyze the security bulletins and updates operatively. Therefore, administrator needs to be in good knowledge of system inside. She should control countless numbers of the system securable objects. For instance, there are 36 types of the Microsoft Windows 2000 entities that are used with access differentiation. Among them there are 9 entities of the user level (e.g. group accounts, NTFS objects, system registry), and 27 kernel-level objects (e.g. jobs, processes, threads, objects of synchronization). Each object refers to the access control list (ACL), every entity (ACE) of which is a 32-bit mask. Users and groups obtain up to 37 privileges. What is more, 38 local security settings specify the system-native security policy. Thus, even in the isolated station, a number of security combinations exceeds *tens of millions*, and it is a hard work to control them all manually. Consequently, to solve the task of security faultlessness we need a special facility for system security analysis.

This paper discusses a technique of the security system analysis and a formal verification tool, the Safety Evaluation Workshop (SEW). It allows to specify the system's security-related elements and verify the system's safety against the OSCVs.

This paper is structured as follows. *Section 2* addresses the classification of security models and illustrates the safety problem background. *Section 3* analyzes the related works in the field of safety modeling and evaluation. *Section 4* introduces our safety problem resolving approach. *Section 5* gives a review of the SEW's structure to analyze the safety of the operating system. In *section 6*, we explain the example of logical specification and security flaws detection for Sample Vulnerability Checking in Microsoft Windows 2000 using the SEW tool. Finally, *section 7* discusses conclusion and future directions.

2 A Safety Problem

In general, *security* represents the combination of confidentiality, integrity or availability. The term of *security model* [1] could be interpreted as formal representation of secure system's confidentiality, integrity or availability requirements. In

more general usage of the term, a security model specifics a particular mechanism for enforcing confidentiality, called *access control*, which brought over into computer security from the world of documents and safes. Security concerns arise in many different contexts; many security models have been thus developed. Access control models can be grouped into two main classes according to security policies [2]:

- *Mandatory Access Control* (*MAC*): based on mandated regulations determined by a central authority;
- *Discretionary Access Control* (*DAC*): based on the identity of the requestor and on access rules stating what requestors are allowed to do.

Getting assurance that a system behavior will not result in an unauthorized access is called a *safety problem* [1]. For the listed model types, the security problem can be specified in the following manner:

- *MAC*: governs access on the base of classification lattice of subjects and objects in the system. There is no safety problem for MAC, because safety of MAC has been proved theoretically in general case [3]. Verification of the safety of any kind of MAC policy is obvious as mathematical proof, but it becomes necessary in particular cases of real world realizations;
- *DAC*: has a drawback that it does not provide real assurance on the flow of information in a system. Harrison, Ruzzo, and Ullman showed that the safety problem was undecidable *in general case* [4], and research was focused on determining whether safety could be decided for access control models with limited expressive power (e.g. [5][6]). They mathematically demonstrated that an access control model could be designed for which safety is efficiently decidable in polynomial time but given a few restrictions.

Unfortunately, MAC-policies are not particularly well suited to the leading world-known solutions and industry organizations that process sensitive information and. On the contrary, a great majority of the systems (e.g. operating systems, DBMS, firewalls) uses the DAC-based security models as the basis of access control mechanism. For instance, Microsoft Windows, Linux, Trusted Mach, and other protected systems use DAC realizations in form of ACLs, access bits, messages control, etc. Thus, the safety verification is the actual problem for security evaluation, especially for the wide-spread computer systems. In theory, safety can not be resolved for the DAC in general case, e.g. HRU model, on the whole. But the determining whether the system implementing access control model is safe or not can be resolved *for the restricted case*, i.e. *for the definite system and for its definite states*. In this paper, we use this statement to solve the safety problem of DAC-related access control by proposing universal specification and model checking tools. These tools allow the Evaluator to describe the system security elements, calculate the safety estimation for any security computer system based on the 'state machine' model, and detect the vulnerabilities of operating system security configurations.

3 The Related Works

Most of the other works on security resolving relates to safety evaluation and vulnerabilities detection. The ASL [7] is an example technique based on the formal

logic language for security specification. Although it provides support for role-based access control, the language does not scale well to real systems because there is no way of modeling real access control rules. There is no specification of delegation and no way of modeling the rules for groups of identities.

LaSCO [8] is a graphical approach for evaluating the security constraints on objects, in which the policy consists of two parts: domain (the system abstraction) and the requirements (authorization rules). The scope of this approach is good for demonstration, but far from implementation in life systems.

Ponder [9] language is a specification language with object-oriented basis. It is the closest to the safety evaluation purposes. It has JAVA implementation, but it is not prepared for automated evaluation. For example, we need manually compose the system's structure, if we want to specify the hierarchical system. The Ponder-based system does not support the state modification and state transition.

We have observed characteristics of the Windows-oriented vulnerabilities detectors (Enterprise Security Manager, Symantec Corp.; Intrusion SecurityAnalyst, Intrusion Inc.; NetIQ Security Analyzer, NetIQ Inc.; XSpider, Positive Technologies; Microsoft Baseline Security Analyzer, Microsoft Corp., etc). After analyzes, we have made some conclusions:

- no solution investigates the system inside. For example, the known products have an eye on the well-known file paths or the security-critical folders. No one looks at security of the kernel mode objects;
- no product allows composing the detection criteria. For example, the analyzed solutions use either the predefined checks or the scripts of the check sequence;
- no detector predicts an effect of the security settings upon the reachable states of the system.

However, to our knowledge, the general problem of the evaluation of the operating system security enforcement including weakness detection has never been addressed by any author.

4 Safety Evaluation Resolving

According to principals of the computer system modeling, we use the term of *security model* as the combination of *system security states* and transitions through *access control rules*. To check the safety of the system, we add to this schema of security model the term of *constraints* like a set of access restrictions given by user. If the constrains are set properly, there will be no OSCVs in the system. Criteria for the OSCVs checking we will call *OSCV-criteria*.

The term of *safety* states that "*something bad never happens*". If a security system has safety problem in security, it means that the OSCV exists and secret information is leaked by unauthorized access. Assurance that a system behavior will not result in the unauthorized access is fundamental for ensuring that the enforcing of the access control model will guarantee the system security. An important feature of an access control model is the ability to verify the safety of its configurations. Using the

approach of the system states, the definite system is *safe* according to the DAC model, if the following conditions met:

1. The security state corresponding to the given initial system state conforms to the OSCV-criteria.
2. The system access control mechanism realizes the access control rules.
3. Every security state reachable from the initial one keeps the OSCV-criteria fair.

The process of producing of the sets of the reachable states and the evaluating of the criteria is called a *safety problem resolving*. Formally, procedure of the states verification can be presented in the following manner.

A system Σ is a state machine: $\Sigma = \{S^{\Sigma}, T, s_{init}^{\Sigma}, Q\}$, where:

- S^{Σ} denotes the set of the system states;
- Q denotes the set of the queries executed by the system;
- T denotes state transition function, T: $Q \times S^{\Sigma} \rightarrow S^{\Sigma}$ that moves the system from one state to another. A request q issued in the state s^{Σ} moves the system to the next state $s^{\Sigma *} = T(q, s^{\Sigma})$;
- s_{init}^{Σ} is the initial system state.

A system Σ is a finite machine: a state s^{Σ} is reachable iff there is a sequence $<(q_{init}, s_{init}^{\Sigma}), ..., (q_n, s_n^{\Sigma})>$ such that $s_n^{\Sigma} = s^{\Sigma}$, and $s_i^{\Sigma} = T(q_i, s_{i-1}^{\Sigma})$, $0<i<n$. Note that s_{init}^{Σ} is trivially reachable.

A general security model M is an ensemble of sets, M={S, R, C}, where:

- S denotes the set of the security states defined by the security model;
- R is the set of the access control rules in the form of logical predicates $r(s, s')$ defined on $S \times S$ and checking that the transition from s to s' meets to the security model;
- C is the set of the *OSCV*-criteria in the form of logical predicates $c(s)$ defined on S and checking security of the state s. A state $s \in S$ is secure iff for every criterion c the predicate $c(s)$ succeeds: $\forall c \in C$: $c(s)=true$.

A system safety property Λ can be formulized as $\Lambda=\{M, \Sigma, D\}$, where D is the mapping function, D: $S^{\Sigma} \rightarrow S$, which sets the relation between the system states and the security states.

By the definitions being specified, the *System Safety Statement* can be formulated. The system Σ implementing the model M is *safe* iff:

1. $\forall c \in C$: $c(D(s_{init}^{\Sigma}))=true$,
2. $\forall s_t^{\Sigma}, s_{t+1}^{\Sigma} \in S^{\Sigma}$: $s_{t+1}^{\Sigma} = T(i, s_t^{\Sigma})$ $\exists s_i = D(s_t^{\Sigma})$, $s_{i+1} = D(s_t^{\Sigma})$ and $\forall r \in R$: $r(s_i, s_{i+1})=true$,
3. $\forall s_t^{\Sigma} \in S^{\Sigma}$: s_t^{Σ} is reachable from s_{init}^{Σ}, $\forall c \in C$: $c(D(s_t^{\Sigma}))=true$.

This statement declares an analogous of the *General Security Theorem* [1, 3], but in reference to the safety problem in *real* computer systems. The System Safety Statement allows the system's security to be evaluated in practical realization of the DAC systems (e.g. Microsoft Windows series, Linux, BSD systems, all UNIX-related versions) and all other systems, security of which needs to be checked. Our approach

also makes possible to discover the nature of the OSCVs, because it uses the states and the states transitions for the security evaluation. To make the checking of the System Safety Statement a routine procedure we have constructed a security configuration evaluation facility — *Safety Evaluation Workshop* (*SEW*).

5 The Safety Evaluation Workshop

For safety evaluation of systems security, we have proposed the *SEW's structure* [10] which consists of 7 components. The following figure shows the evaluation framework in the SEW.

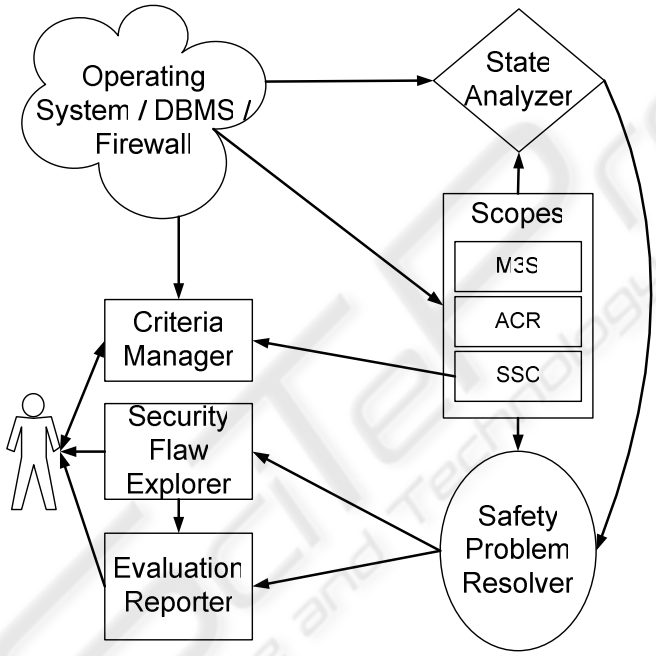

Fig. 1. The Safety Evaluation Workshop

The functions of the SEW's parts are as follows:

- *Safety Problem Specification Language* (*SPSL*): allows to specify the system state, the access control rules, and the OSCV-criteria, and thus to obtain the formal model of the evaluated system for further resolving;
- *Scopes*:
  - *Model-related System Security State Scope* (*M3S-scope*): specifies the system security state in SPSL. For example, the scope for Microsoft Windows 2000 contains the predicates describing all of the securable objects and their attributes, e.g. users, files, processes, kernel objects, ACLs, owners, memberships, etc;

  - *Access Control Rules Scope* (*ACR-scope*): specifies the access control rules in SPSL. For example, in Microsoft Windows 2000, this scope contains the rules that regulate the access control to the securable objects and that are realized in the *system reference monitor* (SRM). Rules have a form of Prolog clauses and allow the state transactions resolving and computing of the authorized accesses for any user;
  - *State Security Criteria Scope* (*SSC-scope*): expresses the OSCV-criteria in SPSL. For example, in Microsoft Windows 2000, this scope allows users to set checking of the Microsoft security bulletins or the firm security policy;
- *State Analyzer*: investigates the system being evaluated and builds the M3S-scope of the system automatically according to the access control model;
- *Criteria Manager*: allows the Evaluator to input the OSCV-criteria in convenient graphical mode and construct the SSC-scope;
- *Safety Problem Resolver*: processes the scopes and detects the security vulnerabilities;
- *Security Flaw Explorer*: demonstrates the sequence of events that lead to the security fault, and finds the security settings that are the reasons of the OSCVs;
- *Evaluation Reporter*: produces the final report containing an initial state (in the terms of security settings), access control rules that lead to the security faults, OSCV-criteria that were resolved and found to be *true*, and an overall safety evaluation result.

The core components of the SEW are the SPSL and the SPR. To specify the system security-related elements we have developed the language — SPSL. SPSL is a logical specification language to express the security states, access control rules, and OSCV-criteria. The expressive power of SPSL allows specification of protection mechanisms of common-used operating systems such as Microsoft Windows 2000, Linux, etc.

We also have composed the SPR tool which helps the Evaluator to process the SPSL-based specifications and to verify whether system security policy involves a safety problem. The SPR interprets the SPSL-based specifications and checks the system to meet the *System Safety Statement*: it evaluates the initial security settings by the given OSCV-criteria; then it generates all reachable states and checks them by the criteria. SPR tool is a logical machine based on the Prolog kernel and supplies with API implemented in C++ programming language.

Other components of the SEW are built around the core modules and make the user's work more handy.

6 Security Evaluation Example

6.1 Sample Vulnerability Detection

For easy understanding of security specification and SEW's functionality technique, we show a *Sample Vulnerability Checking* (*SVC*) applied in Microsoft Windows 2000.

Like in office workstation, our sample computer has MS Office installed. All of the MS Word templates of the user documents are located in the given folder, e.g. *C:\Documents and Settings\Administrator\Application Data\Microsoft\Templates*. Now let's imagine the situation when user named *'Administrator'* shares her template with other users. To do it, she grants the access to read and write the template for the Microsoft Windows built-in group named *'Users'*. If the violator, the member of the *'Users'* group changes the *Normal.dot* template file in the given folder so it contains the malicious code (e.g. macro viruses), all new documents of Administrator will be infected. This is the sample of the OSCV, in which user ignores or forgets the recommendations to protect the MS Word templates.

6.2 The State Scope

Like in well-known security models, our security states are the collections of all entities of the system (subjects, objects) and their security attributes (e.g. ACLs). In our example, we assume that a target of OSCV-criteria is a *C:\Documents and Settings\Administrator\Application Data\Microsoft\Templates* folder. The system security states may be presented as the M3S-scope.

According to M3S-scopc mentioned above, we used the State Analyzer component of SEW to specify SVC's security state. The following code example shows the M3S-scope of SVC generated with the State Analyzer.

```
.........[abbreviation]........
subj('s-1-5-21-73586283-484763869-854245398-500',
[type(user),name('administrator'),
privileges([security,...,remoteinteractivelogon]),
groups(['s-1-5-32-544'])]).
.........[abbreviation]........
subj('s-1-5-32-545',
[type(group),name(['users']),
privileges([shutdown,...,networklogon])]).

.........[abbreviation]........
obj('c:\\documents and settings\\user\\application
data\\microsoft\\templates\\normal.dot',
[type(file),
owner(['s-1-5-21-73586283-484763869-854245398-
500']),inheritance(yes)],
[['s-1-5-21-73586283-484763869-854245398-
500',tnn,[0,1,2,3,4,5,6,7,8,16,17,18,19,20]],
['s-1-5-18',tnn,[0,1,2,3,4,5,6,7,8,16,17,18,19,20]],
```

```
['s-1-5-32-544',
tnn,[0,1,2,3,4,5,6,7,8,16,17,18,19,20]],
['s-1-5-32-545',tnn,[0,1]]]).
.........[abbreviation]........
```

This state specifies three entities: two subjects (one user, one group) and one object (the template file). Every entity is specified in SPSL format. Each predicate declares the security attributes of the corresponding unit. For example, user '*Administrator*' has SID equal to S-1-5-21-73586283-484763869-854245398-500, some system privileges, and owns membership in the '*Administrators*' group. The second predicate specifies the group named '*Users*', which is characterized with some system privileges only. The third expression declares the object, the template file *C:\Documents and Settings\Administrator\Application Data\Microsoft\Templates*, which is the goal of vulnerability evaluation. The attributes of this file are: the owner SID, inheritance flag, and the object's ACL. As in the operating system, ACL is a set of access control entities, presented in the form of " *SID – 'rights delegation' – 'access bits'* ".

In the same manner, using the State Analyzer, we can gather all of the system objects of the user mode as well as of the kernel mode. For example, the following predicate specifies the COM-object and its security configuration:

```
obj('tlntsvr.exe',
[type(com), owner(['s-1-5-32-544']),
appID(['{b8c54a54-355e-11d3-83eb-00a0c92a2f2d}'])],
[['s-1-5-32-544',tnn,[0]],['s-1-5-4',tnn,[0]],
['s-1-5-18',tnn,[0]],
['s-1-5-18',tnn,[1]]]).
```

6.3 The Access Control Rules

Access control rules express the restrictions on a system behavior. The system states transformation is able after the access authorized by the system reference monitor (access control mechanism). It checks the authorization possibility against the security policy requirements represented by access control rules. In SVC example, a subject can have actions of reading or writing according to the SRM policy of Microsoft Windows. To embody this mechanism we have investigated the Microsoft Windows inside (e.g. using the black-box testing strategy) and looked through innumerous Microsoft Press. It made us able to re-compose the protection subsystem in the form of clauses.

Such specification can be called as the ACR-scope. The following code example shows the ACR-scope of SVC. For want of paper space, we do not describe all of the Microsoft Windows ACR-scope in SPSL. We just prepared a sample of the read access checking with some comments describing the SRM working:

```
.....[abbreviation].....
allow_file_read(U, F):-
% System security settings allow user U to traverse
% through containers of file F
   allow_traverse(U, F),
% EPL is effective permissions list
% for user U and file F
   effective_permissions(U, F, EPL),
% Get PL, the list of privileges granted to user U
   privileges_list(U, PL),
% Privilege "Backup files and directories"
% is granted to user U
   ( member(backup, PL), !;
% Permission "Read data" is granted to user U
   member(0, EPL),
% Permission "Read attributes" is granted to user U
   ( member(7, EPL),!;
% P is direct container of file F
   container_of_file(P, F),
% Permission "List folder" is granted to
% user U for direct container of file F
   group_permissions(U, P, 0) ),
% Privilege "Backup files" is granted to user U
   ( member(restore, PL), !;
% Permission "Synchronize" is granted to user U
   member(20, EPL) ) ).
.....[abbreviation].....
```

The 'read' access to the file is granted, if user has a 'traverse' permission for the file, or she has a 'Read Data' bit in her ACE referred to the file, or the user's group membership gives her some abilities to access the file.

6.4 The Criteria

The security criteria allow the customer or evaluator to delimit the secure and insecure states in security model. The criteria have the form of constraints which state

the necessary conditions of the secure state. The system is safe by the OSCV-criteria if all logical goals corresponding to the criteria are true. If some criterion goal is true, then system breaks the safety conditions specified in the criterion. In SEW facility, security criteria can be noted as the SSC-scope. The special component of SEW, the Criteria Manager, allows to compose and edit the vulnerability criteria. This code example shows the SSC-scope written in SPSL.

```
.........[abbreviation].......
criterion('Criterion #1: Users are not allowed to edit
the file Normal.dot',
mask,
[obj('c:\\documents and settings\\administrator\\
application data\\microsoft\\templates\\normal.dot'),
inheritance('tnn'),
's-1-5-32-544'(0,1,2,3,4,5,7,8,6,16,17,18,19,20),
's-1-5-18'(0,1,2,3,4,5,7,8,6,16,17,18,19,20)]).
.........[abbreviation].......
```

The logical predicate denotes one of the criteria to be checked in Microsoft Windows system. It specifies the criterion description: criterion refers to *Normal.dot*. The type '*mask*' points that there is a mask criterion, i.e. the checking of the concrete access rights to the given *Normal.dot* object. There is also a condition of safe system: only *SYSTEM* (its SID equals to S-1-5-18) and '*Administrators*' group (S-1-5-32-544) are allowed to do '*Full Access*' to *Normal.dot*. All other cases are considered to be vulnerable.

In the mentioned style, we can compose a full range of OSCV-criteria. It becomes able to handle even context-related conditions, such as "The system is vulnerable, if *Administrator* can modify object *X*, provided she is connected to the local console". Such conditions are indeed part of Microsoft Windows security model. From the point of security, all kinds of user's activity in the system (such as connection to the local console, applications running, etc) are mapped to Win32API functions calls operated with the Windows recourses. List of functions calls and set of resources maintained by the Windows security (so named as *securable objects*) are defined in MSDN. Because of monitoring a variety of operations over the securable objects, we can analyze the user's activity in the system.

6.5 Safety Evaluation Results Processing

We have SPR input with a triple (M3S-scopc, ACR-scope, and SSC-scope) written in SPSL. Then we have run the resolving program for SVC. After the running

procedure, we have got a result file — the security evaluation *Report*. The following text example shows the report file for SVC.

```
*** SYSTEM SAFETY RESOLUTION ***
CRITERION #1:
Users are not allowed to edit the file Normal.dot

>> VIOLATION DETECTED:
subject            group    <Users>
has unauthorized permissions
                   bits       [0,  1]
                              [Read Data,   Write Data]
for object(s)      file       c:\documents and settings\
                              administrator\
                              application data\
                              microsoft\templates\
                              normal.dot
.....[abbreviation].....
```

The result for checking *criterion 1* is *failed*. It means that there is some vulnerability in the security configuration. After analyzing unsafe state, the SEW discloses nature of security flaw, detecting the subject, object, and their attributes that lead to weakness. The evaluation verdict is "*system is unsafe by this criterion*", because members of group '*Users*' have the '*Read Data*' and '*Write Data*' bits in the ACL, corresponding to the file *Normal.dot*.

7 Conclusion

In this paper, we have addressed to formal specification and vulnerability verification approach for secure operating systems. We discussed a technique of the security system analysis and a formal evaluation tool, the SEW. All these allow to specify the system security-related elements and verify the system safety.

We illustrated the SEW-based checking which enables to verify safety property for secure systems, based on the security scopes. The SEW tool is to be useful for customers and evaluators of secure systems. These kinds of tools are rare in the security field, and no one of the known tools allows the criteria composing, the system inside inspecting, and the configurations influence predicting. The SEW facility brings the safety problem resolving to practice. The targets of its applications

are the computer systems based on state machine presentation: the operating systems, DBMSs, and firewalls.

At last decades, a number of individual countries developed their own security evaluation standards (e.g. [11]). In addition to, opening the way to world-wide mutual recognition of security evaluation results, the new Common Criteria (CC) [12] have been developed. For example, CC define 7 levels (EAL1…7) of assurance for security systems. To get a higher assurance, over EAL5, developers require to specify security model and verify its safety property using formal methods approach. Vendors are discouraged from offering secure systems because significant time and efforts are needed to develop a system capable for meeting the evaluation criteria and to marshal it through the evaluation process. Moreover, because of evaluation delays, an evaluated product is typically no longer the current version of the system, which necessitates repeated reevaluation. For high assurance systems, the difficulties of using formal methods add further complexity to both development and evaluation. However, given the lack of suitable mature, "industrial-strength tools" and the cost of a formal verification activity, informal approach represents a suitable compromise.

The SEW is very useful for administrators and security officers to monitor the system securable resources (files, printers, accounts, processes, etc). The SEW allows any user to discover security of her system in-the-depth, and thus open the 'holes' in the protection. The OSCVs, as mentioned, represent a very serious problem in the modern operating systems. Contemporary systems operate with a huge number of security settings, and the user needs some tools that could explain the whys and wherefores of security weaknesses. The SEW utility makes this process closer to person than ever, because while logical resolving it marks the clause that caused fault of OSCV-criterion, and supplies user with a true reason of the security flaw.

The current versions of the mentioned facility are aimed at such well-known systems as Microsoft Windows 2000/XP and Linux-style systems. For the future works, we will develop and elaborate the SEW components such as the State Analyzer, Criteria Manager, and Security State Explorer to support easy modeling and analyses for safety problem in Microsoft Windows Server solutions (i.e. Active Directory and Group Policy support). Besides, we are targeted at the remote security analysis. To achieve this we develop the Remote Agent module of the SEW which starts the State Analyzer remotely and transmits the scope through the networks.

References

1. J. McLean. Security Model, In Encyclopedia of Software Engineering, Wiley Press, 1994.
2. J. Goguen and J. Meseguer. Security Policies and security models, In Proceedings of the 1982 IEEE Symp. on Research in Security and Privacy, IEEE Computer Security Press.
3. L.J. LaPadula and D.E. Bell. Secure computer systems: A mathematical model, ESD-TR-278, VOL.2, The Mitre Corp., Bedford, MA, 1973.
4. M.H. Harrison, W.L. R.uzzo, and J.D. Ullman. Protection in operating systems, Communications of the ACM, 19(8):461-471, 1976.
5. M. Bishop and L. Snyder. The transfer of information and authority in a protection system, In Proceedings of the 7th ACM Symp. on Operating System Principles, pp. 45-54, 1979.

6. S. Castano, M.G. Fugini, G. Martella, P. Samarati. Database Security, Addison-Wesley, 1995.
7. S. Jajodia, P. Samarati, and V.S. Subrahmanian. A Logical Language for Expressing Authorizations. Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, 1997.
8. 2. J.A. Hoagland, R. Panday, and K.N. Levitt. Security Policy Specification Using a Graphical Approach. Tech. report CSE-98-3, UC Davis Computer Science Dept., 1998.
9. 3. N. Damianou, N. Dulay, E. Lupu, M. Sloman. The Ponder Policy Specification Language. Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 2001.
10. P.D. Zegzhda, D.P. Zegzhda, M.O. Kalinin. Logical Resolving for Security Evaluation, MMM-ACNS, pp. 147-156, 2003.
11. Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, Dec 1985.
12. Common Criteria for Information Technology Security Evaluation, Part, 1: Introduction and General Model, Version 2.1. CCIMB-99, Aug 1999.