Using ACL2 to Verify Security Properties of Specification-based Intrusion Detection Systems

Tao Song [1], Jim Alves-Foss[2], Calvin Ko[3], Cui Zhang[4], and Karl Levitt[1]
[1] Computer Security Laboratory, University of California, Davis
{songt, evitt}@cs.ucdavis.edu
[2]Center for secure and dependable system, University of Idaho
jimaf@cs.uidaho.edu
[3]NAI Labs, Network Associates Inc., Santa Clara, CA
calvin_ko@nai.com
[4]Computer Science Department, California State University, Sacromento
zhangc@ecs.csus.edu

Abstract

Intrusion detection is considered to be an effective technique to detect attacks that violate the security policy of systems. There are basically three different kinds of intrusion detection: *Anomaly detection*, *misuse detection* and s*pecification-based intrusion detection* [MB02]. Specification-based intrusion detection differs from the others by describing the desired functionalities of security-critical entities including system programs, protocols, networks, and application programs [CK97]. This means unknown attacks will be detected as well as known attacks. There is an open question which kind of attacks can be detected by a specific specification-based intrusion detection system. In this paper a hierarchical model is built to reason specifications for different security requirements. A formal framework is built with ACL2 to analyze and improve detection rules of intrusion detection systems [KM00]. SHIM (System Health and Intrusion Monitoring) is used as an example to show the validation of our model and framework [CK01]. We formalize all specifications of SHIM and a trusted file policy and we reason about the soundness and completeness of the specifications by proving the specifications satisfy the policy with various assumptions. These assumptions are properties of the system that are not checked by the intrusion detection system. Analysis of these assumptions shows the role of SHIM in improving the security of the system.

1 Introduction

Intrusion detection systems are widely deployed as effective mechanisms to detect exploitations of vulnerabilities of computer systems. There are basically three different kinds of intrusion detection systems [MB02]: anomaly detection, misuse detection and specification-based intrusion detection. Specification-based intrusion detection is based on the specifications of normal behavior of security-critical entities. These specifications are mainly developed manually, based upon an expert understanding of characteristics of

the programs. The specifications can also be discovered by machine learning and this will not be a topic for this paper.

Specification-based intrusion detection has better performance compare with anomaly detection and misuse detection. Most of anomaly detections are statistic-based. It is difficult to identify unknown attacks from behaviors of the system, so anomaly detection usually detects unknown attacks with a high false positive rate. Misuse detection is based on signatures of known attacks. It is still an open question whether general signatures can catch unknown attacks, but we believe the answer is mostly “no.” Specification-based intrusion detection is considered to have better performance in detecting unknown attacks or variants of known attacks [MB02]. Testing is currently being used to evaluate the soundness of the specifications. But testing is usually performed according to the tester’s understanding of known attacks. It is difficult to verify the effectiveness of an intrusion detection system in detecting unknown attacks. New approaches are needed to verify the soundness and completeness of given specifications.

SHIM, a specification-based intrusion detection system, focuses on the behavior of privileged programs that grant root privilege to normal users [CK01]. In SHIM, specifications are developed in SHIM to constrain the behaviors of privileged programs to the least privilege that is necessary to complete the functionality of the program. It is still difficult to address the completeness of these specifications and the role they play in improving the security of the system. A formal framework is needed to analyze the detection rules in intrusion detection systems.

In this paper, ACL2 is used to develop an abstract system model that can be used as the basis for different intrusion detection systems. A hierarchical model is built to generalize the verification of specifications. As an example, we formalize specifications of SHIM and a security policy, trusted file access policy, and prove that these specifications can satisfy the policy with various assumptions. We also discuss the use of these specifications to enforce a well-known integrity security policy, the Clark-Wilson integrity policy [CW89].

The paper is structured as follow: section 2 introduces intrusion detection. Section 3 describes a hierarchical model of verification. Section 4 shows an example of our approach. We formalize specifications of SHIM and prove that these specifications satisfy various trusted file access policies with assumptions. In section 5 we discuss our results and the limitation of the verification we proved. The last section is our conclusion and recommendation for future work.

2 Related Work

Specification-based intrusion detection was proposed by Calvin Ko in 1996 [CK96]. Several approaches were developed thereafter [CK97, CK01, US01]. In [CK97], Parallel Environment Grammar (PE Grammar) is used to describe the specifications of privileged programs. In [US01], Behavior Model Specification Language is used to describe security-relevant behavior of the system.

The specifications have been analyzed and improved using different methods. In [HFS+98], the normal behavior representations are developed by analyzing normal runs of programs. A machine learning method, inductive logic programming (ILP), is used to construct valid behavior specifications of programs automatically [CK00].

Different approaches are used to specify and analyze the intrusion signatures and detection rules of misuse detections have been [LWJ98,RG01, PD02]. A declarative language, MuSigs, is proposed in [LWJ98] to describe the known attacks. Temporal logic formalas with variables are used to express specifications of attack scenarios [RG01]. Pouzol and Ducasse formally specify attack signatures and proved the soundness and completeness of their detection rules. In addition, data mining techniques and other AI techniques such as neutral network are used to refine and improve intrusion signatures[GS99, LSM99, SEZ+01].

Our approach is different from these approaches in different ways. First we developed a framework to evaluate detect rules of different intrusion detection systems. We formalized security-relevant entities of an UNIX-like system as well as access logs. Detection rules including intrusion signatures and specifications can be formalized and reasoned in the framework.

Second we proposed a way to verify security properties of intrusion detection systems with assumptions and security policies. Security polices are always satisfied with some assumptions. An attack can violate a security policy by breaking its assumptions. So it is possible to verify the improvement of security by proving the *weakening* of assumptions. For example, assuming a policy P is satisfied with assumptions A and with the deployment of the mechanism m , P is satisfied with assumption B where A implies B, then we can say m improve the security because attacks violate assumption B will violate A but attacks violate assumption A may not violate B .

The last but not the least, in our preliminary result, we verify a significant property of specification-based intrusion detection systems: the capability to detect unknown attacks. In our verification, the specifications of SHIM satisfy a *passwd* access policy with some assumptions. This means any attacks, including known attacks and unknown attacks, violate the policy can be detected by SHIM.

3 Approach

In this paper we present a framework to analyze and improve the detection rules of intrusion detection systems. These detection rules include specifications of a specification-based intrusion detection system and attack signatures of misuse detection systems. Anomaly detection systems will not be analyzed because most of them are statistics-based while specifications and signatures are declarable.

We try to answer the question whether given intrusion detection rules can satisfy the security requirements of the system. Security polices and properties of attacks are used to describe the security requirements of the system. The satisfaction of the security

requirement determines whether violations of security policies or instances of attacks can be detected by the detection rules.

3.1 Hierarchical Model of Verification

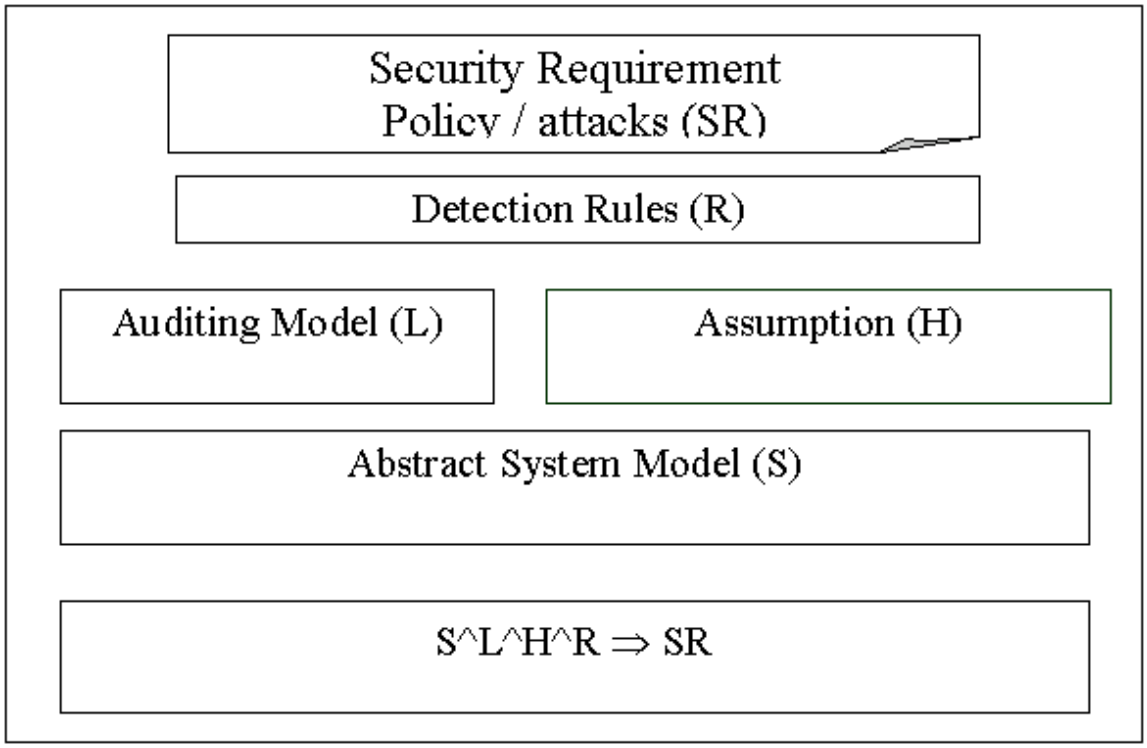

Figure 1: Hierarchical model of verification

A hierarchical model is developed for the verification. The model consists of an abstract system model, auditing model, detection rules, assumption and security requirements. The basis of the model is an abstract system model (S) in which security-critical entities of the system are formalized. An auditing model (L) is necessary for the model because almost all the intrusion detection systems are based on the analysis of audit trail from system, application and network. Detection rules (R) are different according to different intrusions detection systems. In SHIM, detection rules are specifications of normal behavior of privileged programs. Security Requirements (SR) define properties that should be kept to guarantee the security of the system. Assumptions (H) are necessary for the verification. Some security properties that we are not sure and properties that cannot to be monitored will be declared as assumptions (e.g. kernel of the system is not subject to attack).

All specifications of a system are rested on assumptions. A system specification will have assumptions of how the system and programs behave. The specifications cannot be declared as complete before all assumptions of the specifications are identified. In some cases, once the assumptions are declared as required by the verification

approach, an intrusion detection system may not to monitor the properties which are defined by these assumptions.

We use ACL2 theorem prover to specify the verification. ACL2 provides a formal language for specifying data and related operations. At the same time ACL2 provides a basis for formal verification. For a given security requirement (SR), such as a security policy, and some specifications (L), the completeness of the specifications is proved by addressing assumptions (H) of specifications such that the policy (SR) will be guaranteed by specifications(L) and assumptions(H). Such verification can be specified by proving a theorem in ACL2, in which H and L implies SR.

Our verification can also be used to verify security improvements of a system. The security of a system is always based on some assumption. A successful attack will break some of the assumptions. If the deployment of a security mechanism can protect some assumptions from being violated, such mechanism can been verified to improve the security of the system. Supposed A is the set of assumptions and s is a specific assumption in set A ($s \in A$), s cannot be implied from other elements of set A. The security of a system is improved if a mechanism m can imply s. This means s can be protected and any attacks that break assumption s will be detected by m. In addition, m can improve the security of the system by replacing A with a *weaker* assumption B where A implies B. This means some attacks violated A but not violate B will be detected by m. The security improvements will be discussed in section 4 with some examples.

In the abstract system model, some security-critical components are formalized, including Users, Processes, Files, Environment variables and Access control mechanism.

```
system: (proglist calllist filelist userlist envlist)
proglist:((pname pdir)...)
calllist:((callname)...)
filelist:((path ouid ogid pmode inodeid)...)
pmode: ((r w x)(r w x)(r w x)(dir reg socket pipe))
userlist:((uid uname gid homedir)...)
envlist:((envname envvalue)...)
```

Figure 2: Abstract System Model

Auditing plays an important role in recording the behavior of a system and detection of security violations. Most intrusion detection systems analyze audit data of systems to determine any potential attacks to the system.

log record: (procobj fileobj syscall newprop)
Procobj: (prog ruid pid euid egid)
Fileobj:(name ouid ogid pmode nodeid)
Syscall:(syscall flags)
Newprop: (newowner, newmode, newpath, chpid)
pmode: ((r w x)(r w x)(r w x)(dir reg socket pipe))

Figure 3: Audit Model

4 Specification and Verification of SHIM

We developed a hierarchical verification model that can be used to reason different kinds of intrusion detection systems. As an example, we formalized the specifications of a specification-based intrusion detection system, SHIM, and analyzed them according to different security policies and attacks.

4.1 Formalization of SHIM

SHIM developed specifications for privileged programs of UNIX systems. These specifications mainly focus on the valid operations of a privileged program. If the program was compromised by some attacks (e.g. buffer overflow attacks), and tried to invoke any system calls that violate the specifications, an alert would be raised for the violation of the specification.

Parallel Environment Grammar (PE grammar) is used in SHIM to describe all valid operations of a program. In our verification, functions are defined to check the audit trail according to specifications. All valid operations are mapped to functions of ACL2. For example, in the specification of ftp daemon, eight system calls are valid, including *open*, *read*, *write*, *chmod*, etc. We define a function *spec_ftpd_rec* which accept audit trail as a parameter and return *nil* if any system calls of the trail are not valid.

4.2 Security requirements

Security requirements are used to describe some properties that need to be kept to satisfy the security of the system. There are basically two ways to present the security requirements: one is to define security policies, the other is to describe attack scenarios. In our verification, different attacks and security policies are formalized to analyze the specifications of SHIM.

There are two ways to verify whether an attack can be detected by a specific IDS. The first method is to formalize possible audit trails, which include the attack scenarios, and then analyze the audit data according to the specification of the program for the violation. Such verification can be used to prove the capabilities of the specifications to detect known attacks. A more general one is we describe the security property that will be violated by the attacks instead of particular audit trails. Then we develop a proof that the

formalized specifications will always monitor that property. For example, in an ftp-write attack, an attacker takes advantage of a normal anonymous ftp misconfiguration. If the ~ftp directory and its subdirectories are owned by the ftp account or in the same group as the ftp account, the attacker will be able to add files (such as the .rhosts file) and eventually gain local access to the system.

Security policies are also formalized to reason the security properties of specifications. Trusted file access policies are security policies that we developed to keep trusted files from unauthorized access. In UNIX systems, discretionary access control model decide whether a subject can access an object depend on the privilege of the subject and access permission of the object. Some files are supposed to be accessed by specific users or using specific programs. For example, the passwd file of a UNIX system should be edited by root or by another user using the Passwd program. So, file access policies are defined in the format as: *(trusted file, authorized user, program, access)* where "*trusted file*" is the file to be protected, "*authorized user*" defines the user that can access the file with any programs and "*program*" define the program that can be used by other users to access the file. The passwd file access policy is defined as:

(/etc/passwd, root , passwd, (open-wr,create, chmod, chown, rename…))

This policy can be formalized as a function in ACL2:

```
(defun access-logrec (logrec)
        (if (and (not (equal (getprocruid logrec) 0))
        (equal '(/ etc passwd) (getfilename (getfile logrec)) )
        (or (equal 'open (getcallname logrec))
            (equal 'chmod (getcallname logrec))
            (equal 'chown (getcallname logrec))
            (equal 'rename (getcallname logrec))
            (equal 'delete (getcallname logrec)) ))
t  nil ) )
```

Figure 4: Passwd file access policy

4.3 Verification of passwd file access policy

According to the discretionary access control model of the UNIX system, any user without root privilege cannot edit passwd file unless using a privileged program. If the kernel of the system is well implemented, the behavior of unprivileged programs will never violate this policy. So, we need to verify that behavior of all the privileged programs satisfies the policy.

In SHIM, audit filter is used to get the audit trail of a specific program from the audit data of the system. Two levels of verification will be used to verify the satisfaction of the passwd file access policy: verification against a specific privileged program and verification against concurrent execution of different privileged programs.

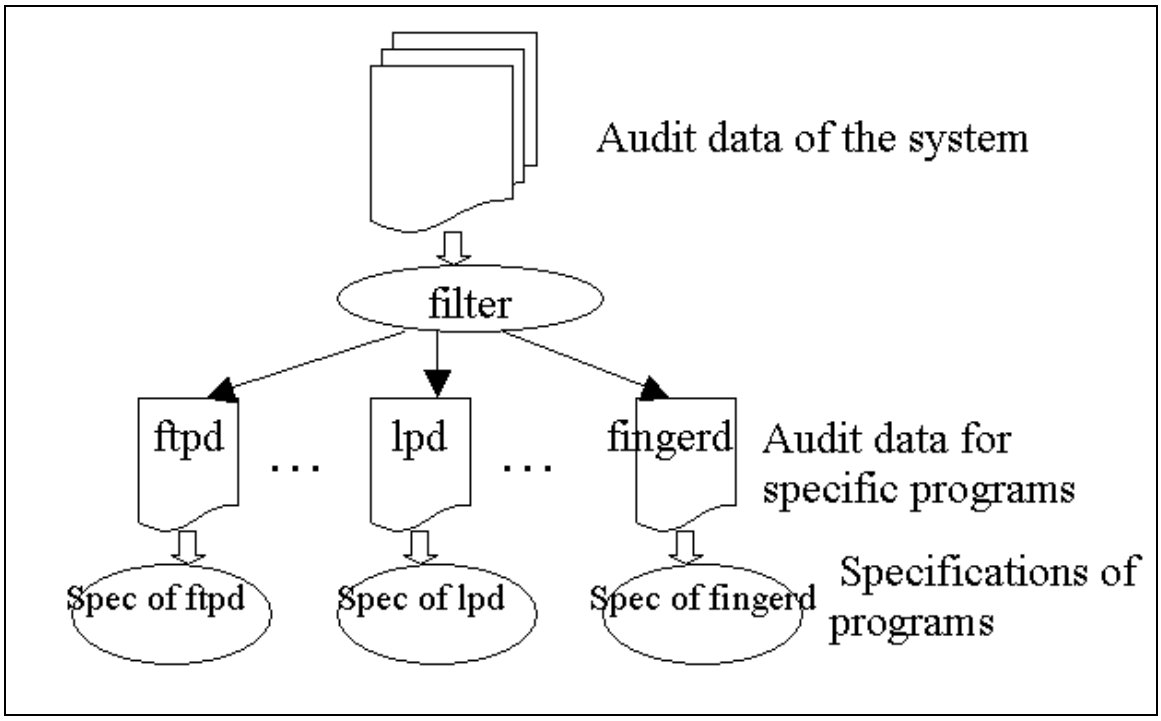

Figure 5: Mechanism of SHIM to filter concurrent execution audit log

4.3.1 Behavior of each privileged program satisfies the policy

Given audit trail of a specifc privileged program, we try to prove any audit trail that passes the specification check will satisfy the passwd file access policy. We use ftp deamon as an example to show how it works.

The proof is defined as a theorem which is listed below. The formalization of the abstract system model *sys* and audit data *log* are used in this theorem. We may notice that some assumptions are added to complete the proof. Such assumptions include the format of data, system assumption and verification assumption. Two important verification assumptions are made in this proof and they are closely related to the specification and the policy. The first assumption is about the access permission of passwd file. The passwd file can only be protected when it has proper access permission. The other assumption is about the setting of the home directory of the user that tres to access the passwd file. If a user can access passwd file his home directory is set as “/etc”. The reason is that the specification of ftp deamon allows the user to access the files under his home directory. In fact, such an assumption can be guaranteed by deploying some configuration checking tools such as COPS. But in SHIM such property of the system are not monitored. With these assumptions, any audit data that passes specification check of ftp deamon will satisfy passwd file access policy.

```
(defthm passwd-ftp
  (implies
        (and(not (member '(/ etc passwd) created))
        (consp log)(consp sys)(logp log)(consp created)(sys-p sys) (validuser sys log)
        (passwdsafe log)(homedirsafe sys)       ; assumptions
        (spec_ftpd sys log created))       ; specification check
     (not-access-passwd log)     ; passwd file access policy
  )
)
```

Figure 6: Verification of passwd file access policy with specification of ftp daemon

4.3.2 Concurrent execution of programs satisfy the policy

In SHIM, a filter is used to map the audit trail of the system to a subset, audit trail of a specific program. We simulate the filter using a function *filter(prog, log)* in ACL2 where prog is the name of the program and log is the audit trail of the system. A question is whether the filter will change the security property of the audit trail. If the filter maps the data trail of a few privileged programs to the audit trail of each program and all the subset of the data trail satisfy the passwd access policy, does this mean the audit trail satisfies the policy? We just analyze the audit trail of two privileged programs. Suppose *log* is the audit trail of ftpd and lpd. We try to prove that if audit trail of ftpd, *filter('ftpd, log)* can pass the specification check of fptd and the audit trail of lpd, *filter('lpd, log),* can pass the specification check of lpd, the audit trails of ftpd and lpd satisfy the passwd access policy.

```
(defthm passwd-specs
  (implies
   (not (member '(/ etc passwd) created))
   (implies
    (and (logp log) (consp log) (consp sys) (sys-p sys) (procsafe log)
          (passwdsafe log) (homedirsafe sys) (validuser sys log)  ;assumptions for ftpd
          (validenv sys 'printerspool)    ;assumptions for lpd
          (spec_ftpd sys (filter 'ftpd log) created)
          (spec_lpr sys (filter 'lpr log) created))
    (not-access-passwd log)))
  )
```

Figure 7: Verification of passwd file access policy with concurrent execution

5 Discussion

A more complex security policy, the Clark-Wilson integrity, can be enforce with SHIM. The Clark-Wilson integrity policy concerns the integrity of the data in a system

[CW89]. This model uses transactions for the basic operation of the system and uses verification procedures to verify the integrity of constrained objects. The standard UNIX operating system is not a mechanism of the Clark-Wilson integrity policy. SHIM can be used to enforce the Clark-Wilson policy. In this mechanism, a privileged program will be monitored as Transformation Procedures (TP) in the CW policy. SHIM will act as Integrity Verification Procedure(IVP) that monitors the behavior of TP. Files play the role of Constrained Data Items (CDI) and the integrity of files will be guaranteed by the CW policy. Access Triple, an important concept of the CW policy, defines associate access of CDI to TP that means a user can only access CDI with specific TP. Specifications of SHIM also define access constraint of privileged programs, so specifications are used as Access Triples. These access triples can be defined with:

(defun AccessTriple(user process file) (spec_process user file))

There are various certification rules and enforcement rules in the CW policy. Our approach satisfies most of them except behavior of root should be limited [SL02].

In the verification, we introduced assumptions needed to satisfy the passwd file access policy. These assumptions related to access permission of target objects (e.g., passwd file cannot be world-writable), proper configurations (e.g., home directories of users cannot be /etc/), and so on. SHIM is not capable of monitoring these basically static properties of the system. But these assumptions can be checked by deploying other security tools such as Tripwire and Kuang [KS93 ZK96]. As a security-critical component, the security of passwd file can be considered as one of the assumptions on the security of the system. Suppose proper access permission of passwd file is assumption *A*, and satisfaction of passwd file access policy is assumption *B*, *A* is a *weaker* assumption compared with *B* because *B* always implies *A*. Also, proper configuration is considered as a normal assumption for the security of the system. So the verification of the ftp deamon proved that any attacks on the ftp deamon will be detected by SHIM if they access the passwd file without authorization.

Beside these assumptions we also have some assumptions relating the “system”. These system assumptions are very important although they are not listed in our verification. We assume the system kernel is correctly implemented and, thus, is not subject to attacks. If the access control mechanism is not well implemented and a user can access some objects for which he is not authorized, it is impossible to protect these objects by only adding constraints on privileged programs. As a hypothesis of the intrusion detection system, audit logs should record the trace of attacks so analysis of the audit logs may detect such attacks. If an attack eliminates its trace from the audit logs before the intrusion detection system analyzes these data, it is impossible to detect such an attack.

6 Conclusion and Future Work

In this paper, we presente the preliminary result of our work. We use formal methods to analyze and improve the detection rules of intrusion detection systems. We

used ACL2, a theorem prover, to verify the completeness of the specifications of SHIM and, in process, show the capability of SHIM in detecting attacks – essentially any activities that causes the security policy not to be satisfied. In subsequent work, we will analyze some misuse detection systems, and signatures characteristic of such systems will be formalized to address which kind of attacks can be specified by these signatures and whether it is possible to detect some unknown attacks with a general attack signature.

Reference

[CK96] C.C.W. Ko , "Execution Monitoring of Security-Critical Programs in a Distributed System: A Specification-Based Approach", Ph.D. Thesis, August 1996

[CK97] C. Ko, M. Ruschitzka, and K. Levitt, “Execution Monitoring of Security-critical Programs in Distributed Systems: A Specification-based Approach”, Proc. of the 1997 IEEE Symposium on Security and Privacy, Oakland, California, May 1997, pp. 134-144.

[CK00] C. Ko, “Logic induction of valid behavior specifications for intrusion detection”, Proc. of IEEE Symposium on Security and Privacy 2000

[CK01] C. Ko, J. Rowe, P. Brutch, K. Levitt, “System Health and Intrusion Monitoring Using a hierarchy of Constraints”, Proceeding of 4th International Symposium, RAID, 2001

[CW89] Clark,D. , Wilson, D., “Evolution of a Model for Computer Integrity, Report of the Invitational” Workshop on Data Integrity, 1989

[DD87] D. Denning, “An Intrusion-Detection Model”, IEEE Transactions on Software Engineering 13(2), pp. 222-232 (Feb. 1987)

[FHS+98] Forrest, S.; Hofmeyr, S.A.; Somayaji, A.; Longstaff, T.A; “A sense of self for Unix processes”, Proc. of IEEE Symposium on Security and Privacy 1998

[GS99] Anup K. Ghosh and Aaron Schwartzbard , “A Study in Using Neural Networks for Anomaly and Misuse Detection”, Proc. of USENIX Security Symposium, 1999

[HW02] H. Chen, D. Wagner , “MOPS: an infrastructure for examining security properties of software”, Technical Report UCB//CSD-02-1197, UC Berkeley, 2002

[KM00] M. Kaufmann, P. Manolios, J S. Moore, “Computer-Aided Reasoning : An Approach”, Kluwer Academic Publishers, 2000

[KS93] G. Kim, E. H. Spafford, “The design of a system integrity monitor: Tripwire,” Technical report CSD-TR-93-071, Purdue University, November 1993

[JA80] J. P. Anderson, "Computer security threat monitoring and surveilance," Technical report, James P. Anderson Co., Fort Washington, PA, April 1980.

[LWJ98] Jia-Ling Lin; Wang, X.S.; Jajodia, S., “Abstraction-based misuse detection: high-level specifications and adaptable strategies”, Proc. of IEEE Computer Security Foundations Workshop, 2002.

[LSM99] Wenke Lee; Stolfo, S.J.; Mok, K.W., “A data mining framework for building intrusion detection models”, Proc. of IEEE Symposium on Security and Privacy, 1999

[MB02] Matthew A. Bishop, Computer Security: Art and Science, Addison Wesley Longman 2002

[MR99] M. Roesch, “Snort: Lightweight Intrusion Detection for Networks”, Proc. of USENIX LISA ’99, Seattle, Washington, November 1999, pp. 229-238.

[PD02] J.P. Pouzol, M. Ducasse, “Formal specication of intrusion signatures and detection rules”, Proc. of IEEE Computer Security Foundations Workshop, 2002.

[PN97] P.A. Porras and P.G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", Proc. of the 20th National Information Systems Security Conference, Baltimore, Maryland, October 1997, pp. 353-365.
[RS02] C.R. Ramakrishnan and R. Sekar, "Model-Based Analysis of Configuration Vulnerabilities", Journal of Computer Security, Vol. 10, No. 1-2, pp.189-209.
[SEZ+01] Schultz, M.G.; Eskin, E.; Zadok, F.; Stolfo, S.J., "Data mining methods for detection of new malicious executables", Proc. of IEEE Symposium on Security and Privacy,2001
[SL02] T. Song, K. Levitt, "Using Specification-based Intrusion Detection to Enforce Clark-Wilson Integrity Model on UNIX," Proc of student workshop of UC Davis, 2002
[US01] P. Uppuluri, R. Sekar, "Experiences with Specification-based intrusion detection," Proc of Recent Advances in Intrusion detection, 2001
[WB89] William R. Bevier, "Kit: A Study in Operating System Verification", Proc of IEEE Transactions on Software Engineering, 1989
[ZL96] D. Zerkle, K. Levitt, "NetKuang-A Multi-host Configuration Vulnerability Checker," Proc of Sixth USENIX Security Symposium, 1996