Security Monitoring of Privileged Programs: A Specification-based Approach

Calvin Ko
Department of Computer Science
University of California, Davis
Davis, CA 95616

1 Introduction

Specification-based detection is a technique of security monitoring beside anomaly detection and misuse detection. In specification-based detection, an object's expected security-relevant behavior is specified formally. The specification is compared against the actual behavior of the object using audit trails generated by the system. Specification-based detection is a promising approach to detecting exploitation of vulnerabilities in various components of a system (e.g., privileged programs, subsystem, network components, etc.) In addition, the specifications are written in a fairly formal way so that some formal reasoning (with respect to a system policy) can be done.

This document describes our effort to monitor privileged programs in UNIX. The motivation is that most attacks are cause, directly or indirectly, by vulnerabilities in privileged programs. In addition, it is rather difficult to find out these vulnerabilities. Our approach is to tightly specify what a privileged can do (the system calls and their parameters) so that even when these programs go bad, we have a great chance of catching them.

We have developed an Execution Monitor for UNIX systems. It monitors the execution of privileged programs (setuid root programs and daemon programs that run as root) with respect to their specifications using audit trails generated by the operation system. Our system is able to detect previous attacks that exploited vulnerabilities in privileged program. We describe the language we developed for specifying security-relevant program behavior in UNIX. Using the language, we specified most of the privileged programs in a UNIX system. The specification of the programs are precise and concise. Typical length of the specification of a program is a few lines in our language. We talk about our current prototype as well as some future work.

2 Specification Language

Currently, the security-relevant behavior that we are looking at is the set of operations that the program can do. Our specification language is based on predicate logic. Relevant system operations are represented as predicates. A program specification defines the allowable parameters for each operation. Reader should refer to [1] for more detail. Shown below is an example specification of the finger daemon program

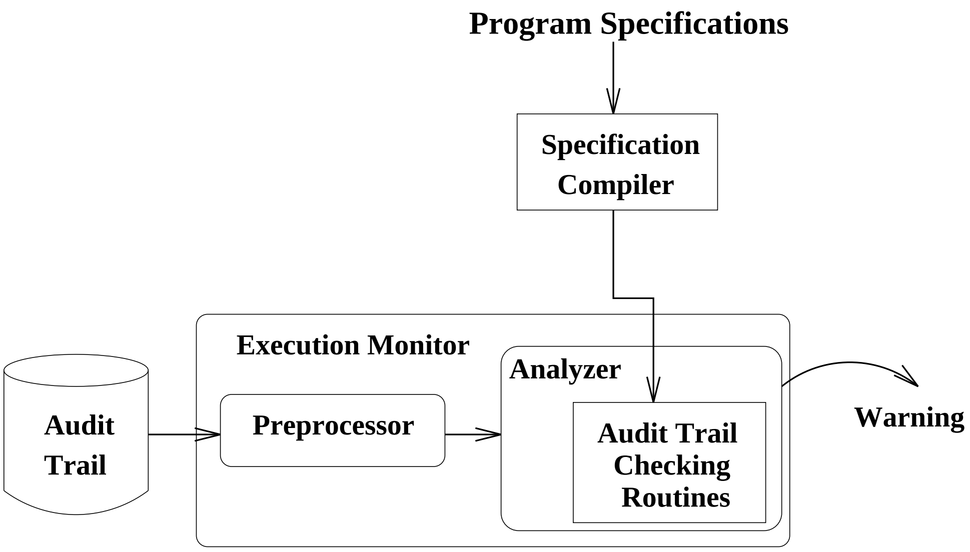

Figure 1: Architecture of the Execution Monitor

```
PROGRAM fingerd(U)
   read(X) :- worldreadable(X);
   bind(79);
   write ("/etc/log");
   exec("/usr/ucb/finger");
END
```

The specification indicates that the finger daemon can read any file that is worldreadable, bind to port 79, write to "/etc/log", and execute the finger program (/usr/ucb/finger). Beside these four operations, it should do anything else. The attack by the Internet Worm should be caught by this specification.

3 Architecture

Figure 1 depicts the architecture of our current prototype. The preprocessor reads the audit records generated by the system, filters irrelevant records, associates a program to each audit record, and produces program audit records (PAR) that will be analyzed by the program execution analyzer. The program execution analyzer analyzes program audit records and determines whether a program violates its specified behavior. One part of the program execution analyzer is a set of audit trail checking routines that describe the specification at the level of abstraction of audit trails. The routines are Perl procedures which take a program audit record and determine the action represented by the record violates the specifications. A routine is generated for each privileged program by specification compiler automatically from the specification of the program.

Our current prototype runs on Solaris 2.3 and 2.4 systems. We have tested it with some existing vulnerabilities; Our prototype is able to detect them instantly. Currently, we have very accurate specifications for some of the problematic privileged programs such as rdist, sendmail, binmail, ps, finger, etc. We plan to specify all the privileged programs in UNIX, which can be used a standard security specification of these programs.

4 Current and Future Work

Extension to our current specification lanaugae: Our current specification is concerned with the operations that the program is allowed to perform regardless of their orders. We are trying to extend the current specification language to capture the ordering of the operations as well as behavior related to synchronization. We attempt to use a grammar (possibly attribute grammar) to describe program execution. A specification of a program is a grammar, which language is the set of all good system call traces of the program execution. Therefore, the analysis is tranformed into language parsing. Our next specificaton language should be able to handle synchronization problems in distributed or concurrent programs.

A Model of UNIX: We are developing a model of UNIX so that we can reason about the security of a UNIX system that uses our execution monitor. The model consists of specifications of system calls and privileged programs. In particular, we want to reason about the integrity of the system.

A Systematic Way for Writing Specification: We are investigating ways for specifying privileged programs systematically. In addition, we want to identify some conceptual features that aids specifying privileged programs.

Reference

[1] *C. Ko, G. Fink, K. Levitt, "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring", in Proceedings of the 10th Computer Security Application Conference.*