Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 332 (2017) 3–20

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Controlling File Access with Types

Rakan Alsowail and Ian Mackie

Department of Informatics
University of Sussex
Falmer, UK

Abstract

Accidental misuse of shared files by authorised users is a predominant problem. This paper proposes a well-known static analysis approach, namely a type system, to prevent such accidental misuse. We develop a type system that intercepts commands issued by users in a file system and enforces policies on each file. Commands issued by users to manipulate files will be subject to type checking by the type system. Type-checked commands are then guaranteed to not violate policies of the files. The focus of this paper is on a particular policy that allows owners of files (users who created files) to specify the number of times a file can be read by limiting the number of times a file can be copied. Therefore, a file can be read as much as it can be copied. If the file cannot be copied, then it can be read only once. This approach can be extended to other properties.

Keywords: File sharing, security types, type checking

1 Introduction

File sharing has become an indispensable part of our daily lives. The shared files might be sensitive, thus, their confidentially, integrity and availability should be protected. Such protection might be against external threats that are initiated by unauthorised users or insider threats that are initiated by authorised users. Our main interest is with insider threats, in particular trusted authorised users who might accidentally violate files policies. The most widely used technique to protect shared files is access control such as Discretionary Access Control (DAC) [9,7] and Role-based Access Control (RBAC) [14]. Although access control is useful to specify who can access which information, it cannot protect sensitive information against legitimate users. Access control is concerned with the release of information but not its propagation. It provides a guarantee that information is released only to authorised users. However, once information is released to authorised users, it might be leaked maliciously or accidentally to unauthorised users without any further control. Information flow control is a complementary approach to access control to prevent information leakage. It tracks how information propagates through a program during execution to ensure the program does not leak sensitive information.

http://dx.doi.org/10.1016/j.entcs.2017.04.002

Information flow control can be enforced statically [4,5,15,16] or dynamically [13,2]. The former analyses information flow within a program prior to executing, while the latter analyses information during the execution. The dominant approach for enforcing secure information flow statically is the use of type systems.

This paper presents a novel approach of using a type system to solve the problem of accidental misuse of shared files by trusted authorised users. Such misuse occurs, for example, when a trusted authorised user accidentally disseminate a file to unauthorised users, write to a file that is meant be read only, or copy a file that is meant to be read once, after which the file should be erased. Hence, misuse is action that violates files policies. We design a language of commands to manipulate files and specify their policies in a Unix-like file system, and a type system to enforce these policies. In this setting, files are associated with security types that represent security policies, and programs are sets of commands to be issued on files such as read, copy, move, etc. The type system plays the role of a reference monitor that intercepts and statically analyses each command to be issued on a file and determines whether or not the command is safe to be executed. Safe commands are those which do not cause errors during execution. Such errors might be caused by commands that violate the security policies associated with the files or violate its own requirements (e.g., a file must exist to be removed). Therefore, if commands are type-checked, then files and commands policies are not violated and can be executed safely. In this paper, we focus on enforcing a particular policy, namely, limiting the number of times a file can be read. However, the same basic ideas can be extended to enforce other policies as pointed out in Section 6.

The rest of this paper is organised as follows. Section 2 presents the security types and policies of files. Section 3 describes the language syntax and semantics for manipulating files, defines security errors and an algorithm for checking syntactical errors. Section 4 describes the type system, and includes properties. Section 5 introduces a type checking algorithm and proves its soundness and completeness. In Section 6 we give a brief review of related work, and finally we conclude the paper in Section 7.

2 Security Types and Policies

Our approach to limiting the number of times a file can be read is by limiting the number of copies the file can produce. Therefore, a file can be read as much as it can be copied. If a file cannot be copied, then it can be read once. To enforce this policy, we need to restrict the access to *copy* operations and restrict the information flow caused by all operations such that restrictions of files over *copy* operations are not violated. To control the access to copy operations on files we define three security types which are UC, LC^n, and NC each of which specifies a distinct policy of how copy operations can be performed on them. UC stands for Unrestricted Copy, which means that a file associated with this type can be copied without restriction. The copied version of a file of type UC should be allowed to be copied in the same way, so should also be of type UC. LC^n stands for Linear Copy, which means that a