An Aspect-Oriented Model to Monitor Misuse

K. Padayachee
School of Computing, University of South Africa
Pretoria, 0003 South Africa

J.H.P. Eloff
Department of Computer Science, University of Pretoria
Pretoria, 0002 South Africa

Abstract- **The efficacy of the aspect-oriented paradigm has been well established within several areas of software security as aspect-orientation facilitates the abstraction of these security-related tasks to reduce code complexity. The aim of this paper is to demonstrate that aspect-orientation may be used to monitor the information flows between objects in a system for the purposes of misuse detection. Misuse detection involves identifying behavior that is close to some previously defined pattern signature of a known intrusion.**

I. Introduction

The efficacy of the aspect-oriented paradigm has been well established within several areas of security, such as authentication, access control, encryption and software tampering. Typically security concerns tend to crosscut objects, resulting in code tangling. However the aspect-oriented paradigm facilitates the abstraction of these security-related tasks to reduce code complexity. Crosscutting concerns are related issues that are scattered throughout the functionality of an application [1]. The aim of this paper is to demonstrate that aspect-orientation may be used to monitor the information flows between objects in a system for the purposes of misuse detection. Misuse-detection involves identifying behavior that is comparable to some previously defined pattern signature of a known intrusion.

Application-level bugs are often exploited to compromise the security of a system and it is vital to correct these vulnerabilities instantaneously. For instance, 'design-level problems accounted for about 50% of the security flaws uncovered during Microsoft's "security push" in 2002. Sufficient protection of software applications from attacks, however, is beyond the capabilities of network and operating system-level security approaches (e.g. cryptography, firewall and intrusion detection) because they lack knowledge of application semantics' [2].

Detecting programming attacks should ideally follow an approach similar to the course of action advocated by Newsome and Song [3]: A detection mechanism should detect unknown attacks early, before the system is compromised. Secondly, once a new exploit attack is detected, attack signatures must be developed that can be used to filter out those attacks efficiently until the vulnerability can be patched. This paper focuses on the latter event, when an attack is known and where an interim measure must be instituted until the problem is resolved.

The use of aspect-orientation in information security has been validated by several studies ([4-6]). Aspect-orientation promotes reusability since a security aspect may be reused for other applications [1]. For example, access control has similar requirements for most applications. Vanhaute and De Win [7] have derived reusable generic aspects from typical security concerns. As aspectual components do not need to have hard-wired names of objects, an aspect may be easily reused [8]. Furthermore, the aspect-oriented paradigm is highly extensible as it is flexible enough to accommodate the implementation of additional security features after the functional system has been developed, as crosscutting concerns may be added or removed without making invasive modifications to original programs [9].

This paper examines the strategy of using the aspect-oriented paradigm to reveal patterns of information flow to detect programming attacks. The first two sections explore the concepts of misuse detection systems and information flow control respectively, while the discourse of the subsequent two sections focuses on aspect-oriented programming and its influence on software security. Section 6 demonstrates how an aspect-oriented methodology may be used to detect information flow patterns that signify programming attacks. Sections 7 and 8 conclude with directions for future work and insights gathered from the experiment conducted.

II. Misuse Detection Systems

Anomaly detection relies on identifying all behavior that is abnormal for an entity. While misuse detection involves flagging behavior that is close to some previously defined pattern signature of a known intrusion. The disadvantage of the first approach is that it does not necessarily detect undesirable behavior, and that the false alarm rates can be high. The problem with tracking all information flows would be the difficulty in identifying an anomaly as these logs will probably be large. The problem with misuse-based detection is that the anomaly must be known in advance.

While misuse-based detection cannot detect new intrusions, in actual systems, anomaly detection systems have the advantage of detecting previously unknown intrusions[10]. Anomaly detection methods involve various machine learning and statistical techniques [11]. This paper will not examine the issues surrounding pattern recognition.

Recently there has been a trend towards using hybrid frameworks combining both misuse detection and anomaly detection components which, in effect, reduces the inefficiencies and maximizes the strengths of both techniques (see [12]). The other significant trend is the movement towards the inclusion of intrusion detection systems on the application-level. Most intrusion detection

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 273–278.

systems are essentially based at the network-level or operating system level. It has been noted recently that there is a need to consider the application-level, in terms of monitoring the interaction between the user and the application [13]. 'Application-level bugs are more frequent than kernel-level bugs and, therefore, applications are often the means to compromise the security of a system. Detecting these attacks can be difficult, especially in the case of attacks that exploit application-logic errors' [14]. However 'to detect attacks exploiting application-logic errors, it is desirable to be able to perform selective, application specific auditing in certain points of the application's control flow. The problem is that few applications provide hooks for instrumenting [sic] their control flows, and, even if these hooks are available, they may not be in the right places. In addition, the instrumentation technique would be application-specific and not easily portable to different applications' [14]. It is evident that aspect-orientation may be ideal for providing these 'hooks' through the use of pointcut designators [15].

Another newly identified trend is the use of information flow control to support misuse detection. In this research attempts are made to find a solution within the aspect-oriented paradigm while incorporating some of these trends. The model presented here, is based on misuse detection within the application-level using information flow control analysis.

III. Background On Information Flow Control

Information is exchanged among variables in procedural programs and by messages in object-oriented systems. An illegal flow arises when information is transmitted from one object to another object in violation of the information flow security policy [16]. A transfer of information does not necessarily occur every time a message is passed. An object acquires information by changing its internal state, as a result of changing the values of some of its attributes. Thus, if no such changes occur as a result of a message invocation in response to a message, then no information has been transferred [17]. There have been two basic types of information flow controls available within the object-oriented perspective, namely language-based information flow controls [18] and information flow controls based on message filtering [16, 19].

Language-based information flow controls are enforced through the use of security-typed languages where program variables and expressions are augmented with annotations that specify policies on the use of the typed data. Language-based information-flow techniques necessitates that the programmer must not only understand the algorithm to be implemented but must also understand what the desired security policy is and how to formalize it using annotations [20]. Further security policies may not be available during functional design, thereby resulting in inconsistencies. The aspect-oriented paradigm enables security policies to be separated from the code and accordingly security policies may be coded independently of other requirements [21]. In general, information models are difficult to implement. Hence the message filtering model developed by Jajodia and Kogan [19] considers only primitive operations such as *read* and *write* methods. As the aspect-oriented paradigm facilitates genericity through the use of wildcards, it may extend the message filter model beyond considering only primitive operations [22].

The methodologies presented above, are solely based on preventing illegal information flow. However, Masri and Podgurski [23] presented a novel approach to detect attacks against application software using dynamic information flow analysis. Where certain patterns of information flow may be used to detect vulnerabilities and possible attacks. This paper shares this notion but surveys an aspect-orientation implementation of information flow, as an alternative technology. Due to the genericity offered by aspect-orientation, the model presented here may be used within other contexts during security risk analysis, where the illumination of specific information flows to detect vulnerabilities is required.

IV. Background On Aspect-Oriented Programming

In every object-oriented software design there are core concerns. In a robotic system, for instance, these concerns involve motion management and path computation. The concerns are located in a particular scope and are not required in any other scope. Other concerns are common to many of a system's modules like logging, authorization and persistence. These system-wide concerns are called 'crosscutting concerns' and the re-implementation of one issue in different modules is called 'code scattering' [24]. Aspect-oriented programming addresses the problem of code scattering by localizing these crosscutting concerns into a modular unit called an aspect.

An aspect is a modular unit of a crosscutting implementation that is provided in terms of pointcuts and advices, specifying what (advice) and when (pointcut) its code is going to be executed [25]. In terms of codification, aspects are similar to objects. However, aspects observe objects and react to their behavior [26]. An aspect is a piece of code that describes a recurring property of a program and can span multiple classes, interfaces or aspects [8]. Unlike a class though, aspects are injected into other types. Aspects improve the separation of concerns by making it possible to cleanly localize crosscutting design concerns. They also allow programmers to write, view and edit a crosscutting concern as a separate entity.

During program execution, there will be certain well-defined points where calls to aspect code would be inserted [25]. These are known as join points. Aspects introduce their supplemental functionality at these join points [26]. A pointcut is a set of join points described by a pointcut expression. An advice declaration is used to specify code that should run when the join points specified by the pointcut expression are reached [27]. The advice code will be executed when a join point is reached, either before or after the execution proceeds. For example, AspectJ supports *before*, *after* and *around* advices, depending on the time the code is executed [28]. A *before* (*after*) advice on a method execution defines code to be run before (after) the particular