Technical Report

Information Flow Control within AOP

Keshnee Padayachee
padayk@unisa.ac.za

J.H.P Eloff
eloff@cs.up.ac.za

Abstract: This research demonstrates that aspect-oriented programming can be used to augment an existing system with information flow control controls. Information flow controls primarily aims in preventing information leakage in software systems. This research also considers information flow in differing contexts and consider how aspect-oriented may be improved to further facilitate the implementation of information flow security.

1. Introduction
Developing secure software systems requires more than protecting objects from illegal manipulation but also preventing illegal information flow among objects in a system [Izaki et al. 2001]. This research focuses on the issue of preventing illegal information flows among objects in a system. Incorporating information flow control during software development is extremely challenging. Firstly, the problem with building real applications that have information flow features is interfacing the new application with existing infrastructure that are not designed with information-flow in mind. Secondly it is difficult to manage and assign security policies [Zdancewic 2004] during software development. Thirdly, 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 [Zdancewic 2004]. This research proposes employing the aspect-oriented paradigm to address these problems. It is evident that the aspect-oriented paradigm facilitates the implementation of additional security features to legacy systems without modifying existing code. As roles between application developers and security specialists are separated [Bodkin 2004], it can also simplify the management of security policies. This research considers the feasibility of developing an information flow security aspect to augment an existing system to address illegal information flows.

Several authors have been citing the benefits of using aspect-oriented programming to implement security concerns (see [Viega et al. 2001] and [De Win et al. 2002]). According to Bodkin [2004], Aspect-Oriented Software Development, is relevant for all major pillars of security: ‘authentication, access control, integrity, non-repudiation, as well as for supporting the administration and monitoring disciplines required for effective security.’ The primary
argument supporting aspect-oriented programming is that the average programmer does not have the requisite skills in security. This can be attributed to a lack of expertise and few tertiary institutions offering tuition in software security [Viega et al. 2001]. These types of programming tasks such as authentication, access control, and integrity, should be abstracted away from programmers and allocated to security experts. Secondly, it is observed that security concerns such as encryption and access control tend to crosscut objects. Thirdly, a security aspect can be reused for other applications [De Win et al. 2001]. For instance, access control has the same requirements for most applications. Fourthly, aspect-oriented software design is flexible enough to accommodate the implementation of additional security features after the functional system is developed.

This research investigates the feasibility of using the aspect-oriented paradigm to facilitate the non-intrusive implementation of information flow control features to an existing system as crosscutting concerns can be added without making invasive modifications on the original code. We present two case studies to demonstrate the practicability of the model presented here.

2. Problem Statement

“Many difficulties with information “leakage” arise not from defective access control but from the lack of policy about information flow. Flow controls are concerned with the right of dissemination of information irrespective what object holds the information; they specify valid channels along which information may flow.”[Denning 1982]. Despite its appeal, Information-flow mechanisms have not yet been successfully applied in practice.

This research investigates the feasibility of applying information flow control aspect non-intrusively to an existing system using the aspect-oriented paradigm. As crosscutting concerns can be added or removed without making invasive modifications on original programs. This research will consider information flow in different contexts and consider how aspect-oriented may be improved to further facilitate the implementation of information flow security.

3. Methodology

The research conducted here was based on case-study approach, where simple object-oriented systems will be woven with the message filter aspect and re-compiled under the aspect-oriented paradigm. This research was based on two case-studies. The first one involved the typical information flow analysis, where classes were given classifications. The second one involved discovering information flow patterns to detect programming attacks.
4. Delimitations
Modifications to programs can either be unintentional or intentional (malicious). This study will focus on the former that is, unintentional modifications which include errors or bugs committed by programmers. It is important to distinguish between access control and information flow control. For example, an access policy might specify that user1 can read from file1 and write to file2, while a flow policy might specify that information in file1 is at most confidential and always less than the class of information in file2 [Andrews and Reitman 1980]. This study is primarily concerned with the latter, that is, information flow control, and more specifically information flow analysis and the application of information flow analysis.

5. Literature Survey

5.1. Background work on Information Flow Controls
A software system that manipulates and stores sensitive information such as employee salaries, tax information or identification numbers must prevent such information from leaking during execution [Myers 1999]. Security Controls, such as firewalls, anti-virus software and access control mechanisms, are not sufficient to protect against this type of information leakage. For instance, determining whether a communication violates confidentiality is outside the scope of a firewall mechanism. Similarly with encryption, there is no guarantee that once data is decrypted that its confidentiality will be respected [Sabelfeld and Myers 2003]. An access control policy specifies the rights that subjects such as users have to access objects that contain information [Andrews and Reitman 1980]. These types of controls are placed on the release of data and do not control how data ‘flows’ as each statement in a program executes. ‘Many difficulties with information “leakage” arise not from defective access control but from the lack of policy about information flow. Flow controls are concerned with the right of dissemination of information irrespective what object holds the information; they specify valid channels along which information may flow.’ [Denning 1982]. For example, an access policy might specify that user1 can read from file1 and write to file2, while a flow policy might specify that information in file1 is at most confidential and always less than the class of information in file2 [Andrews and Reitman 1980]. As such, information from file1 can flow to file2, but information flowing from file2 to file1 would be an illegal flow. The issue of secure information flow has become especially important with the growing popularity of mobile code and networked information systems [Sabelfeld 2001]. For example, with the advent of internet banking, the repercussions of a client gaining access to another clients’ account is disastrous for all parties concerned.

An information flow policy specifies a set of security classifications for information and a flow relation defines permissible flows among these classifications of information [Denning and Denning 1977]. Direct flows result from assignment statements and indirect flows result
from conditional statements such as the *if* and *while* programming constructs [Andrews and Reitman 1980]. For instance, example 1 (Figure 1, below), shows information flowing directly from x to z. Information is also flowing *implicitly* from x to y. Example 2, is also an example of information flow, although little can be deduced from the sum, about x and y, if both are unknown. But if one is known then it is easy to determine the other element. Example 3, results in an implicit flow from x to y, because there is no direct assignment statement, but the value of x, can be deduced from y.

<table>
<thead>
<tr>
<th>Example 1:</th>
<th>Example 2:</th>
<th>Example 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>z = x;</td>
<td>z = x + y</td>
<td>if x = 1</td>
</tr>
<tr>
<td>y = z;</td>
<td></td>
<td>y = 1</td>
</tr>
</tbody>
</table>

**Figure 1: Examples of Information Flow**

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 [Samarati et al. 1997]. 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 [Jajodia et al. 1992]. For example in the statements below (Figure 2) information from object \( W \), is being transferred to object \( P \), via two messages namely `getHoursWorked()` and `Get_Hours()`. The former, returns a value which is passed as argument to the latter. This value is used within the context of object \( P \), either by assigning one of its attributes to the value obtained from `getHoursWorked()` or by performing some operation as a result of the value obtained from `getHoursWorked()`. Either way information is flowing into \( P \) via \( W \).

```java
Pay_Info P = new Pay_Info(new Double(1.99));
WorkInfo W = new WorkInfo(new Double(5));
P.Get_Hours(W.getHoursWorked());
```

**Figure 1: Demonstrating Information Flow between objects**

There have been several studies on information flow control policies in the object-oriented paradigm (see [Samarati et al. 1997], [Banerjee and Naumann 2002] and [Myers 1999]) but no articles surveyed so far deal with the issue of information flow in terms of aspect-oriented programming with the exception of a related work by [Masuhara and Kawauchi

4
The next discussion elaborates on the terminology and concepts pertaining to aspect-oriented programming.

5.2. Background work on Aspect-oriented programming
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. In the execution of a program, there will be certain well-defined points called join points where calls to aspect code would be inserted. The pointcut is used to find a set of join points where aspect code would be inserted. An advice declaration can be used to specify code that should run when the join points specified by the pointcut expression is reached. The Advice code will be executed when a join point is reached, either before or after the execution proceeds. A before (after) advice on a method execution defines code to be run before (after) the particular method is actually executed. Around advice defines code which is executed when the join point is reached and has control over whether the computation at the join point (i.e. an application method) is allowed to execute [Kiczales et al. 2001].

The final application is generated taking both the application functional code and its specific aspects. These two entities will be combined at compile time by invoking a special tool called a weaver[Kiczales et al. 1997].

5.3. Related Work on Aspect-Oriented Security
There has been a significant amount of work done in aspect-oriented security to warrant making the process more systematic in terms of software design and development. The subsequent discussion highlights the relevance of aspect-oriented technology in terms of implementing some of the major pillars of security such as access control and authentication, accountability and audit, data protection and information flow controls in software systems.

5.3.1 Access Control and Authentication
De Win et al.[2001] delineated three types of aspects, an Identification aspect, Authentication aspect and an Authorization aspect for access control in the aspect-oriented paradigm. The Identification aspect is used to tag the entities that must be authenticated. The subject (see Figure 3) included in the aspect is used to determine whether access should be allowed or not. The Identification aspect is used as a container for identity information of the subject. The Authentication aspect passes authentication information to the access control mechanism. The Authorization aspect checks access based on the identity information received through the Authentication aspect. As indicated by the code below, De Win et al.[2001] have actually generalized the aspects they developed for access control, so that it is more reusable.
Ramachandran et al. [2006] also addressed authentication and authorization within the aspect-oriented paradigm but they provided a more generic approach. Due to the duality between information flow and access control the model developed here, does share some parallels with algorithms provided by Ramachandran et al.[2006]. However Ramachandran et al. [2006] do not address information flows.

5.3.2 Accountability and Audit
Accountability and audit serve to collect and analyze the activity of an information system. They aim at detection of security violations and defining causes, these can also be easily implemented with aspects [Slowikowski and Zielinski 2003]. Here Slowikowski and Zielinski [2003] demonstrated, that an aspect could keep records when a particular exception is thrown from a component without modifying the component (see Figure 4) below.

5.3.3 Encryption
In an experiment conducted by [Boström 2004], it was found that database encryption can be added after the initial system was already built using aspect-oriented programming. This case study showed that using aspect-oriented programming resulted in better modularity, database independence and less code, but there are instances where the logic developers cannot be totally alienated from the process of encryption because sometimes developing the functionality is dependent on the encryption process.
5.3.4 Information Flow Controls
Masuhara and Kawauchi [2003] found that although sanitizing was a crosscutting concern, there was no possible way to define a pointcut that would be able to detect whether a string was from an unauthorized source or not or contain unwanted information. Hence, they proposed a new pointcut called \textit{dflow} that addresses the dataflow between join points as an extension to the AspectJ Language. Recall that join points are well-defined points where calls to aspect code would be inserted. Although this study is related to information flow, the authors do not address security classifications and their dataflow definition ‘only deals with direct information flow’. Further, they do not comment on the propagation of information among objects in a system. As no studies have performed exclusively on this area and as aspect-oriented programming is an evolution in object-oriented programming, it would be pragmatic to investigate information flow control from this context first.

5.4 Investigating Object Oriented Concepts from an information flow perspective.
It is important to note that access controls ‘place restrictions on release of information but not its propagation’ [Sabelfeld and Myers 2003]. Whereas information flow control is a means of ‘regulating the dissemination of information of objects throughout the system’ [Denning and Denning 1977]. In this context objects implied files, segments or program variables however this notion of information flow is still relevant to object-orientation. There have been two basic types of information flow controls available within the object-oriented perspective, namely language-based information flow Controls [Sabelfeld and Myers 2003] and information flow controls based on message filtering [Jajodia and Kogan 1990, Samarati et al. 1997].

5.4.1 Language-based Information Flow Controls
Information flow control is not a widely used practice compared to access control mechanisms as it was considered to be highly impractical until the use of security typed languages [Sabelfeld and Myers 2003]. In a security-typed language, program variables and expressions are augmented with annotations that specify policies on the use of the typed data (see Figure 5 below). This type of program analysis was pioneered by Denning and Denning [1977]. These security policies are then enforced by compile-time type checking and thus had little runtime overhead [Sabelfeld and Myers 2003].
Language-based information-flow techniques require annotations where 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 [Zdancewic 2004]. For example, JFlow (see Figure 5 above) developed by Myers [1999] and the security typed language proposed by Banerjee and Naumann [2002] require the programmer to be responsible for annotation where data values are labeled with security. Further security policies may not be available during functional design thereby resulting in inconsistencies.

5.4.2 Message Filtering Algorithm

Jajodia and Kogan [1990] first proposed the message filtering algorithm for object-oriented systems where every object is assigned a unique classification. Messages between objects are intercepted by the message filter which decides upon examining the given message and classification of the sender and receiver, whether to permit the information flow or not. This model was extended by Samarati et al. [1997] into a high discretionary access control model for object-oriented systems. The high discretionary access model is able to provide the flexibility of discretionary access control but also the high assurance of mandatory access control.

As the language-based information flow control method forces the programmer to consider the security issues together with the functional issues, it conflicts with the ideology of using aspect-oriented paradigm to resolve security issues. As the aspect-oriented paradigm enables security policies to be separated from the code thereby enabling developers to write the main application and the security experts to specify the security policies [Viega et al. 2001]. Whereas the message filtering algorithm is more conducive to the separation of concerns notion as it is separate modular unit. However the message filtering model developed by Jajodia and Kogan [1990] is relatively complex to implement in the traditional object-oriented paradigm. As Jajodia and Kogan [1990] only consider primitive operations such read and write methods and provide no specification of how the message filtering algorithm can be implemented. The aspect-oriented paradigm allows a more generic implementation of the message filter aspect through the use of wildcards, instead of only considering primitive operations such read and write methods. Using wildcards eliminate the need for explicit naming [Kiczales et al. 2000].

With respect to all possible problems identified with current information flow strategies and the possibility that aspect-orientation might be able to alleviate these issues, an aspect-
oriented information model was developed as described below and a preliminary prototype of the model was developed to demonstrate the practicality of such a model.

5.5. An Aspect-based approach to modeling Information Flow Control

Identifying illegal flows between objects would require an aspect based on the principle of the message filtering algorithm developed by Jajodia and Kogan [1990]. Here pointcuts would be utilized to identify flows between objects. This aspect will observe objects, and intercept messages flowing between them. This aspect’s advice would decide upon examining the given message and classification of the sender and receiver, whether to permit the information flow or not.

In this model, a classification is assigned to the entire object rather than individual attributes of the objects. Jajodia et al. [1992] have demonstrated that the message filter model may be extended to modeling multilevel entities through a process called security inheritance. This study will be limited to considering single-level classifications. It is proposed that these security classifications should be abstracted away from the programmer’s responsibility using the intertype declaration construct which is supported by aspect-oriented programming languages such as AspectJ and Hyper/J [Hanenberg and Unland 2003]. An intertype declaration is generally used to add on information such as methods or fields to an object without modifying the existing class. Here the inter type declaration will be used to add a classification field to each object in the system to store the classification of that particular object. The message filter aspect and the inter type declarations containing classifications for each object in the source code will then will be weaved together with the original source code (see Figure 6 below).

---

**Figure 5: Illustrating Information Flow with aspect-orientation**
The intention of this research was to test this model within existing systems using a case study approach.

5.6. Case Study 1:
With respect to the aspect-oriented flow model described above, a preliminary prototype was built to demonstrate the possibility that such system can be fully implemented. The prototype was built using Aspect J(adjt_1.2_for_eclipse_3.0) as an extension to Java (J2SDK1.4.2_05), and the Eclipse 3.0 IDE. This prototype is essentially a warning system. It merely intercepts flows between objects and outputs whether the flow is from a high security object to a low security object or vice-versa. The classification system added an extra data member to each class, named “tag”, where “tag” was a value denoting the class as 1(secret) or 0(unclassified) (See Figure 8 below) using inter type declarations. For example in the code below (Figure 7), suppose Pay_Info was classified as 1, and WorkInfo and Employee Objects were classified as 0, then Line 5 should be acceptable information flow while Line 7 would result in an information leak. In other words, information from Low Security Object, WorkInfo to a High Security Object Pay_Info is acceptable while information from a High Security Object, Pay_Info to a Low Security Object Employee should issue a warning.

```
1 Employee Micky = new Employee
   ("MStevens","1234","7,23rd Street Menlo Park");
2 Pay_Info P = new Pay_Info(new Double(1.99));
3 WorkInfo W = new WorkInfo(new Double(5));
4 //this should be allowed
5 P.Get_Hours(W.getHoursWorked());
6 //this should not be allowed and a warning issued
7 Micky.pay(P.getPay())
```

**Figure 7: Demonstrating Information Flow between objects**

The Flow aspect considered those actions that resulted in an attribute being assigned (set) or returned (get) from an object. This notion actually took care of all interactions between objects including when objects are being instantiated. When an attribute was returned from a message, the reference of this attribute was stored in a vector. When an attribute in an object was being assigned to a particular value, this vector was inspected to check if the value was obtained from another object. If this value was obtained from another object then a warning message is given to the user if information was flowing from a high security object to a low security object (see Figure below 9).
Several insights were gathered from this experiment. Firstly it is possible to add on information flow to a system without modifying the existing objects. As it is possible to omit the Classifications aspect and the Flow aspect thereby allowing the other objects to function as system without information flow controls.
5.7. Case Study 2:

With respect to the aspect-oriented flow model described above, a small system was built based on the example provided by [Masri and Podgurski 2005] to test the possibility that such system can be fully implemented. The prototype was built using AspectJ(adt_1.2_for_eclipse_3.0) as an extension to Java (J2SDK1.4.2_05), and the Eclipse 3.0 IDE. We did not include the Classifications aspect as this was not necessary for this case study.

We considered a defective server implementation and a simple attack that was adapted from Masri and Podgurski [2005]. The implementation of a server application comprising three classes Server, Session and Account. There is vulnerability in that the server allows a malicious client to avoid getting charged for his/her connection time. This vulnerability was exploited by the following sequence of events:

1) Attacker opens first session (session1) and uses if for a long time.
2) Attacker opens second session (session2)
3) Attacker closes session1 (immediately after step 2)
4) Attacker closes session 2 (immediately after step 3)

This attack basically induces the following information flows:

clientAccount ->Session1
ClientAccount ->Session2
Session1 -> ClientAccount
Session2 -> ClientAccount

This experiment sought to replicate this idea using the aspect-oriented information flow implementation to identify patterns of flow (see Figure 10).

```java
public aspect Flow {
    pointcut getMethods(): get (* ..*);
    before () returning (long x):getMethods() && (within (Session || Account)){
        //Store info about this JoinPoint
    }
    pointcut setMethods(long x):((set (* ..*)||call(* *(..))) && args(x) &&
    within (Session||Account));
    before (long x): setMethods(x){
        // Output flows between Session and Account
    }
}
```

Figure 10: Showing an Aspect could be used to detect flow patterns (Partial implementation provided)
After the aspect indicated in Figure 10 was weaved into the rest of the system, the following code is outputted from the above program. If we isolate the relevant flows, we can see that this reflects the pattern of flow identified by Masria and Podguski [2005]. There were two extra flows detected, but the pattern indicates a vulnerability that is clearly detectable.

Information Flowing From ACCOUNT: ThisUser to SESSION: Session1
Information Flowing From ACCOUNT: ThisUser to SESSION: Session2
Information Flowing From ACCOUNT: ThisUser to SESSION: Session2
Information Flowing From SESSION: Session1 to ACCOUNT: ThisUser
Information Flowing From SESSION: Session1 to ACCOUNT: ThisUser
Information Flowing From SESSION: Session2 to ACCOUNT: ThisUser

The experiment sought to reveal how quickly the model given could be adapted to the problems at hand. In case, it single programmer was able to complete the task of adapting the aspect in 1 hour after the classes were built and tested.

6. Conclusion
With research, we conducted two case studies to prove the information flow control could be implemented within AOP. The first case study showed the information flow control could be implemented within AOP and tested if the code did not violate the information flow policy. We then presented another case study to show that our aspect-oriented information flow control model could be extended to identify patterns of information flow during program attacks. Information flow control is more than access control, as an illegal flow might occur even when only authorized requests are performed on each object. As such most access control models are supplemented with some form of information flow control. Despite its appeal, information flow mechanisms have not yet been successfully applied in practice. As it is difficult to apply and manage in practice and requires programmers to be security experts. The aspect-oriented paradigm can be used to add security to legacy systems and due to the separation of roles between application developers and security it can make the management of security policies easier. In this paper, an information flow control model for existing systems using the aspect-oriented paradigm is posited. Aspects offer several benefits in terms of compact code and increased confidence but they could be drawbacks as it is a new technology. Clearly it is possible to add on information flow without adapting the existing system.

References


Languages and Systems (APLAS'03) Beijing, China 27-29
November,Eds.LNCS2895,105-121.

MYERS, A. C. 1999. JFlow: Practical Mostly-Static Information Flow Control. In
Proceedings of the 26th ACM Symposium on Principles of Programming

RAMACHANDRAN, R., PEARCE, D. J. AND WELCH, I. 2006. AspectJ for Multilevel
Security. In The 5th AOSD Workshop on Aspects, Components, and Patterns for
Infrastructure Software (ACP4IS) Bonn, Germany, 2006

21 March 2006,
SABELFELD, A. 2001. The Impact of Synchronisation on Secure Information Flow in
Concurrent Program. In Proceedings of Andrei Ershov 4th International Conference
on Perspectives of System Informatics Akademgorodok, Novosibirsk, Russia 3-6 July

IEEE Journal on Selected Areas in Communications, 21(1), 5-9.

Flow Control in Object-Oriented Systems. IEEE Transactions on Knowledge and
Data Engineering, 9(4), 624-538.

SLOWIKOWSKI, P. AND ZIELINSKI, K. 2003. Comparison Study of Aspect-oriented and
Container Managed Security. AGH University of Science and Technology.


Workshop on Programming Language Interference and Dependence (PLID) Verona,

We intend publishing the results obtained here at CISSE 2006.