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

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

method is actually executed. An *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 be executed or not [29] .Combining the application functional code and its specific aspects generates the final application. These two entities will be combined at compile time by invoking a special tool called a 'weaver' [8].

V. Related Work On Applying The Concept Of Aspect-Oriented Security In Creating Secure Software

Security is often extracted as a separable concern, due to its orthogonal nature in respect of the functional requirements of a system, hence the separation-of-concerns principle of the aspect-oriented paradigm is suited to addressing security concerns [30]. Aspect-oriented software development is relevant for all major pillars of security: authentication, access control, integrity, non-repudiation, as well as for the supporting administration and monitoring disciplines required for effective security [31]. Even security-related bugs such as buffer overflows or race conditions can be considered a security-related concern [6]. Security aspects can be used to modularize access control and authentication (see [6], [32] and [33]).

In a study related to information flow control, Masuhara and Kawauchi [34] found 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 contained unwanted information. Hence they proposed a new pointcut called *dflow* that addresses the dataflow between join points as an extension to the AspectJ Language. 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. To address these shortcomings, we have conducted case-studies to demonstrate that the aspect-oriented methodology might be useful for detecting illegal flows between objects [22].

These experiments showed that aspects could be utilized to identify flows between objects. For instance, this aspect's advice could decide, upon examining the given message and classification of the sender and receiver, whether to permit the information flow or not. In an unpublished work by Padayachee, Eloff and Bishop [35], this model was further generalized, so that it may be used in other contexts as well. The *Flow* aspect considered those actions that resulted in an attribute being assigned (set) or returned from an object. This notion actually addresses all interactions between objects, including when objects are being instantiated. When an attribute is returned from a message, the reference of this attribute is stored in an appropriate container. When an attribute in an object is being assigned to a particular value, this container is inspected to check if the value was obtained from another object. If this value was obtained from another object then an appropriate action may be taken if the information flowing should not be permitted according to the information flow policy (see Fig. 1 below).

```
public aspect Flow {
pointcut getMethods():  get(* *.*);
before( )returning(Object x): getMethods() &&
(within (A || B || C //...and other objects being tracked )){
//Information flowing out of this object
//Store information about this joinpoint – Such as the reference of
//this object.
}
pointcut setMethods(Object x):  ((set (* *.*) || call (* *(..))) && args(x)
&& within( A || B || C //...and other objects being tracked ));
before(Object x): setMethods(x) {
//if there is an illegal flow between the sender and receiver object
//  Respond appropriately
//else
//  Allow Process to proceed as normal
}
}
```

Fig. 1. Aspect to determine the flow of information between objects.

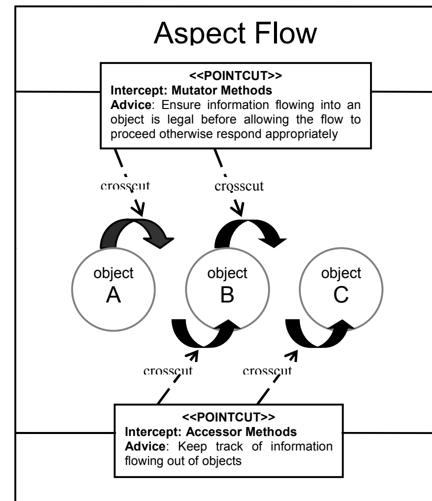

Fig. 2. Aspect Flow intercepts information flow between interacting objects.

For instance, this aspect's advice could decide, upon examining the given message and classification of the sender and receiver, whether to permit the information flow or not. Figure 2 illustrates - when sender object (A) sends information to receiver object (B), this flow is intercepted and tested if it violates the specified information flow policy. If the flow does not disobey the specified information flow policy then the aspect allows the flow to proceed to receiver object (B), otherwise the aspect does not allow the flow to occur or performs a specific action. The model however, has limitations as it only identifies explicit flows [35]. We have also demonstrated via a case study that using aspect-oriented flow model may useful in identifying programming attacks [22]. In this paper we reprised this model in a more generalized context such that it may be useful be in identifying programming attacks.

VI. Case Study To Demonstrate The Practicality Of The Model

With respect to the aspect-oriented flow model described above, a small system was built based on the example

provided by Masri and Podgurski [23] to test the possibility that such system can be fully implemented. The system was built using Aspect J (ajdt_1.2_for_eclipse_3.0) as an extension to Java (J2SDK1.4.2_05), and the Eclipse 3.0 IDE. We wanted to show that aspects could be used to detect vulnerabilities. To this end, we considered a server application comprising of three classes `Server`, `Session` and `Account` (depicted in Fig. 3), where there was vulnerability in that the server allows a malicious client to avoid getting charged for his/her connection time:

1) Attacker opens a first session (session1) and uses it for a long time.
2) Attacker opens a 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 vulnerability by using the aspect-oriented information flow model to identify patterns of flow that indicate misuse.

```
//____________________________________________________________
public class Account {
 public Account (String user){
     name = user;
 }
 public Double getCredit() {
     return credit;
}
 public void setCredit(Double credit){
     this.credit = credit;
}
 public String name;
 private Double credit  = new Double (3600000);//1 hour
}
public class Session {
 Session(Double startCredit){
     this.startCredit = startCredit;
     startTime =  new Double (System.currentTimeMillis());}
 public Double computeCredit(){
return new Double (startCredit.doubleValue() -
(System.currentTimeMillis() -
 (startTime).doubleValue()));
}
 private Double startCredit ;
 private Double startTime;
}
public class Server {
 Session openSession(String user){
     theAccount = new Account(user);
     Account account = theAccount;
     return new Session(account.getCredit());
}
 void closeSession(Session session){
     Account account = theAccount;
     account.setCredit( (session.computeCredit()));
}
 Account theAccount;
}
public class SimpleAttack {
 public void accessServices(Server s, Session session, Double
Duration){
     // Do necessary action
 }
 public static void main(String[] args) {
     SimpleAttack s = new SimpleAttack();
     Double shortDuration = new Double (10);
     Double longDuration = new Double (1000);
     Server server = new Server();
     //OPEN SESSION 1
     Session session1 = server.openSession("User01");
     s.accessServices(server, session1,longDuration);
     //OPEN SESSION 2
     Session session2 = server.openSession("User01");
     //CLOSE SESSION 1
     server.closeSession(session1);
     //USE SESSION2 FOR A SHORT PERIOD.
     s.accessServices(server,session2,shortDuration);
     //CLOSE SESSION 2
     server.closeSession(session2);
 }
}
//____________________________________________________________
```

Fig. 3.Defective server implementation and simple attack.
Note only code relevant to the vulnerability/attack is shown.
(adapted from Masri and and Podgurski [23]).

After the classes were developed and tested, the generalized aspect *Flow* (Fig.4.) was contextualized to identify this type of programming attack. The specific flows between the `Account` class and the `Session` class which were woven into the rest of the system were identified. The specifications involved adjusting the `getMethods()` pointcut to include the `ComputeCredit()` method calls. The *Flow* aspect only identifies flows when an object's data members flow out of an object. As the `ComputeCredit()` method did not involve a data member being returned, it had to be specifically named. The next issue involved identifying the objects involved. As the `Account` class had a name data member, this class did not pose a problem. However, as the `Session` class did not contain a data member that could be used to identify the object, an aspect `AddToSession`, was created specifically for that purpose.

```
//____________________________________________________________
public aspect Flow {
private static Vector References = new Vector();
private static Vector JoinPoint_String =  new Vector();
private static int Count = 0;
pointcut getMethods(): get (* *.*) || execution(Double
Session.computeCredit(..) );
before () returning (Object x):getMethods() && (within (Session
|| Account))
{//Store info about this Join Point
 JoinPoint_String.add(Count, thisJoinPoint.getThis());
 References.add(Count,x);
 Count++;
}
pointcut setMethods(Object x): (set (* *.*) ) && args(x) &&
within (Session||Account);
before (Object x): setMethods(x)
{String Signature = thisJoinPoint.toString();
 for (int i = 0; i < Count; i++){
 if (References.get(i) == x){
System.out.print("Information Flowing From ");
 String PrevSignature = JoinPoint_String.get(i).toString();
 if (PrevSignature.indexOf("Session") > 0 ){
 System.out.print("SESSION: " + ((Session)
```

```
 JoinPoint_String.get(i)).myname);
 }
 else if (PrevSignature.indexOf("Account") > 0){
 System.out.print("ACCOUNT: " + ((Account)
 JoinPoint_String.get(i)).name);
 }
 if (Signature.indexOf("Session") > 0 ){
 System.out.println(" to SESSION: "
 +((Session)thisJoinPoint.getThis()).myname);
 }
 else if (Signature.indexOf("Account") > 0){
 System.out.println(" to ACCOUNT: "+
 ((Account)thisJoinPoint.getThis()).name);
 }
}}}
}
public aspect AddToSession {
 private static int Session.Count = 0;
 public int Session.myname = Count;
 pointcut SessionConstructor(): execution(Session.new(..));
 before(): SessionConstructor() {
 Session.Count++; ((Session) thisJoinPoint.getThis()).myname =
 Session.Count;}
}//
```

Fig. 4. Aspect-oriented implementation to identify patterns of misuse.

The following output (Fig.5.) was produced after the aspects (in Fig. 4.) were woven together with the classes given in Fig.3 using AspectJ. This output reflects the pattern of flow identified by Masri and Podgurski [23] which depicts a user exploiting a program vulnerability.

```
Information Flowing From ACCOUNT: User01 to SESSION: 1
Information Flowing From ACCOUNT: User01 to SESSION: 2
Information Flowing From SESSION: 1 to ACCOUNT: User01
Information Flowing From SESSION: 2 to ACCOUNT: User01
```

Fig. 5. The output produced reflects patterns of misuse detection.

There were a few limitations to this experiment. A few liberties were taken in adapting the aspect to identify the specified flows. It is not the most efficient solution but it rather reflects a 'hacking style' technique. The experiment sought to reveal how quickly the model given could be adapted to resolve the problem at hand. The model proposes that once vulnerability is discovered, an aspect can be used to monitor this vulnerability until a fix or patch is created. Thus this particular monitoring aspect is temporal and will be removed once the program is fixed. In this case, a single programmer was able to complete the task of adapting the aspect in an hour after the classes were built and tested. Several insights were gathered from this experiment. The limitations of AspectJ posed a problem in identifying the objects. It is easy to identify the classes used by referencing a special variable called *thisJoinPoint*, which contains reflective information about the current join point. However, it is difficult to determine the name of the object itself. It would be ideal if the *thisJoinPoint* variable could be expanded to resolve this issue.

A security risk analyst has to trace and track vulnerabilities in a system and this is difficult as he has to relate to the subtleties of the system that are understood by the designer and implementer [36]. Certainly using the model developed above can expose these subtleties without the security risk analyst understanding every program path in the program and constraints may be formed without relying on the programmers to abide by these constraints.

VII. Future Work

We presented an aspect-oriented flow model to identify patterns of information flow. We tested the model on a very small system. In the future a thorough case-study approach would be taken on a more scalable solution to access the flexibility and usability of the model. Although the model was able to produce the flow pattern, the semblance of the pattern reproduced here was discerned manually. A future undertaking may involve developing a tool to automate the process.

VIII. Conclusion

Despite its appeal, information flow mechanisms are difficult to manage in practice and require programmers to be security experts. The aspect-oriented paradigm can be used to add security to existing systems and due to the separation of roles between application developers and security, it can make the management of security polices easier. In this paper, an information flow control model to detect misuse using aspect-orientation is posited. This model may be useful to security risk analysts for identifying security vulnerabilities. Aspects offer several benefits in terms of compact code and increased confidence, but there could be drawbacks as it is a new technology.

References

[1] B. De Win, B. Vanhaute and B. Decker, "Security through Aspect-Oriented Programming," in *Advances in Network and Distributed Systems Security, IFIP TC11 WG11.4 First Working Conference on Network Security*, pp. 125-138, November 2001.

[2] D. Xu and K. Nygard, "A Threat-Driven Approach to Modeling and Verifying Secure Software," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pp. 342-346, November 2005.

[3] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection,Analysis, and Signature Generation of Exploits on Commodity Software," in *The 12th Annual Network and Distributed System Security Symposium*, pp. 1-17, February 2005.

[4] P. Falcarin, M. Baldi and D. Mazzochi, "Software Tampering Detection Using Aop and Mobile Code," in *International Conference on Aspect-Oriented Software*

Development (AOSD'04), pp. 1-6, March 2004.

[5] G. Boström, "A Case Study on Estimating the Software Engineering Properties of Implementing Database Encryption as an Aspect," in *Proceedings of the 3rd international conference on Aspect-oriented software development*, pp. 1-6, March 2004.

[6] B. De Win, W. Joosen and F. Piessens, "Aosd Security: A Practical Assessment," in *Workshop on Software engineering Properties of Languages for Aspect Technologies (SPLAT03)*, pp. 1-9, March 2003.

[7] B. Vanhaute and B. De Win, "Aop, Security and Genericity," in *1st Belgian AOSD Workshop*, pp. 1-2, November 2001.

[8] J.P. Choi, "Aspect-Oriented Programming with Enterprise Javabeans," in *Fourth International Enterprise Distributed Object Computing Conference (EDOC'00)*, pp. 252-261, September 2000.

[9] N. Ubayashi, H. Masuhara and T. Tamai, "An Aop Implementation Framework for Extending Joint Point Models," in *ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution*, 15th of June 2004.

[10] R.A.V. Kemmerer, G., "Intrusion Detection: A Brief History and Overview," *Computer*, vol. *35*, pp. 27-30, April 2002.

[11] G. Adam, "Anomaly Detection in Distributed Computer Communication Systems," *Cybernetics and Systems*, vol. *37*, pp. 635-652, September 2006.

[12] J. Zhang and M. Zulkernine, "Hybrid Network Intrusion Detection Technique Using Random Forests," in *The First International Conference on Availability, Reliability and Security, 2006. ARES 2006.*, pp. 262-269, April 2006.

[13] M. Suydam, "Application-Based Idss: Upping the Application Ante," http://infosecuritymag.techtarget.com/articles/1999/ids.shtml,1999.

[14] J. Zhou and G. Vigna, "Detecting Attacks That Exploit Application-Logic Errors through Application-Level Auditing," in *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pp. 168 - 178, December 2004.

[15] E. Tombini, H. Debar, L. Me and M. Ducasse, "A Serial Combination of Anomaly and Misuse Idses Applied to Http Traffic," in *20th Annual Computer Security Applications Conference, 2004.*, pp. 428- 437, December 2004.

[16] P. Samarati, E. Bertino, A. Ciampichetti and S. Jajodia, "Information Flow Control in Object-Oriented Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. *9*, pp. 624-538, July-August 1997.

[17] S. Jajodia, B. Kogan and R. Sandhu, "A Multilevel-Secure Object-Oriented Data Model," Tech. Rep., pp.596-616, 1992.

[18] A. Sabelfeld and A.C. Myers, " Language-Based Information-Flow Security," *IEEE Journal on Selected Areas in Communications*, vol. *21*, pp. 5-19, January 2003.

[19] S. Jajodia and B. Kogan, "Integrating an Object-Oriented Data Model with Multilevel Security," in *1990 IEEE Symposium on Security and Privacy*, pp. 76-85, May 1990.

[20] S. Zdancewic, "Challenges in Information-Flow Security," in *The First International Workshop on Programming Language Interference and Dependence (PLID)*, pp. 1-5, August 2004.

[21] J. Viega, J.T. Bloch and P. Chandra, "Applying Aspect-Oriented Programming to Security," *Cutter IT Journal*, vol. *14*, pp. 31-39, February 2001.

[22] K. Padayachee and J.H.P. Eloff, "Information Flow Control within Aop, Http://Www.Osprey.Unisa.Ac.Za/Technicalreports/Index.Html," vol., pp.

[23] W. Masri and A. Podgurski, "Using Dynamic Information Flow Analysis to Detect Attacks against Applications," in *Proceedings of the 2005 workshop on Software engineering for secure systems-building trustworthy applications*, pp. 1 - 7, May 2005.

[24] N. Kuntze, T. Rauch and A.U. Schmidt, "Security for Distributed Web Applications

Via Aspect Oriented Security," in *Conference Information Security South Africa*, pp. 1-12, June-July 2005.

[25] F. Ortin and J.M. Cueva, "Dynamic Adaptation of Application Aspects," *Journal of Systems and Software*, vol. *71*, pp. 229-243, May 2004.

[26] J. Viega and J. Voas, "Can Aspect-Oriented Programming Lead to More Reliable Software," *IEEE Software*, vol. *17*, pp. 19 - 21, November 2000.

[27] D. Mahrenholz, O. Spinczyk and W. Schröder-Preikschat, "Program Instrumentation for Debugging and Monitoring with Aspectc++," in *Proceedings of The 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing*, pp. 249-256, April - May 2002.

[28] B. De Win, W. Joosen and F. Piessens. "Developing Secure Applications through Aspect-Oriented Programming." in *Aspect-Oriented Software Development*, Aksit, M., Clarke, S., Elrad, T. and Filman, R.E. Eds., Boston: Addison-Wesley, 2002, pp.633–650.

[29] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten and J. Palm, "Getting Started with Aspectj," *Communications of the ACM*, vol. *44*, pp. 59-65, October 2001.

[30] P. Robinson, M. Rits and R. Kilian-Kehr, "An Aspect of Application Security Management," in *AOSD'04 International Conference on Aspect-Oriented Software Development*, pp. 1-7, March 2004.

[31] R. Bodkin, "Enterprise Security Aspects," in *AOSD'04 International Conference on Aspect-Oriented Software Development*, pp. 1-12, March 2004.

[32] V. Shah and F. Hill, "An Aspect-Oriented Security Framework," in *DARPA Information Survivability Conference and Exposition*, pp. 143, April 2003.

[33] P. Slowikowski and K. Zielinski, "Comparison Study of Aspect-Oriented and Container Managed Security," in *AAOS 2003:Analysis of Aspect-Oriented Software(**Workshop held in conjunction with ECOOP 2003**)*, July 2003.

[34] H. Masuhara and K. Kawauchi, "Dataflow Pointcut in Aspect-Oriented Programming," in *Proceedings of The First Asian Symposium on Programming Languages and Systems (APLAS'03)*, pp. 105-121, November 2003.

[35] K. Padayachee, J.H.P. Eloff and J. Bishop, "Aspect-Oriented Information Flow Control," unpublished.

[36] H. Chivers and J. Jacob, "Specifying Information-Flow Controls," in *Proceedings of the 25th IEEE International Conference on Distributed Computing System Workshops (ICDCSW '05)*, pp. 114 - 120, June 2005.