K.G.S. De Silva
140106T
Case Study on **Applying dynamic separation of aspects to distributed systems security & Change management**

Introduction

Distributed systems are commonly required be flexible, expandable. But with increase of number of nodes potential of eavesdropping,spoofing or repudiation of transaction, phishing or denial of service increases. Where various approaches are available to reduce this effect, this study proposes the use of dynamic aspect-oriented software development (AOSD) to implement security mechanisms in distributed systems.
(In this report their methodologies won't be discussed due to the complexity but importance and background information)

Shutting down the system or nodes in such distributed system for a change are not favourable due to two main reasons.

- Shutting down whole distributed system can cause many losses. Ex : Money loss, competitive advantages for competitors, Leaving the system in an inconsistent state
- Shutting single nodes and updating them independently can leave the system in an inconsistent state.

So, it's better in many ways if the changes can be applied to a Distributed system dynamically.
Importance of the method suggested in the case study is that changes can be applied while the system is running.

By using the method authors have implemented solutions for following two problems.

1. Access control and data flow
2. Encryption of transmission

Corrections performed on the running system when,

- Changes to the environment including new vulnerabilities (which were not taken into account at design time)
- Performance (Security options disabling due to performance)

Security of a system depends on its ability to defend itself. This falls under two categories such as,

- What must be protected : Data, Services, Environment resources
- From whom : Threats on C,I,A

There are some Software approaches we can use to implement security when systems are up and running,

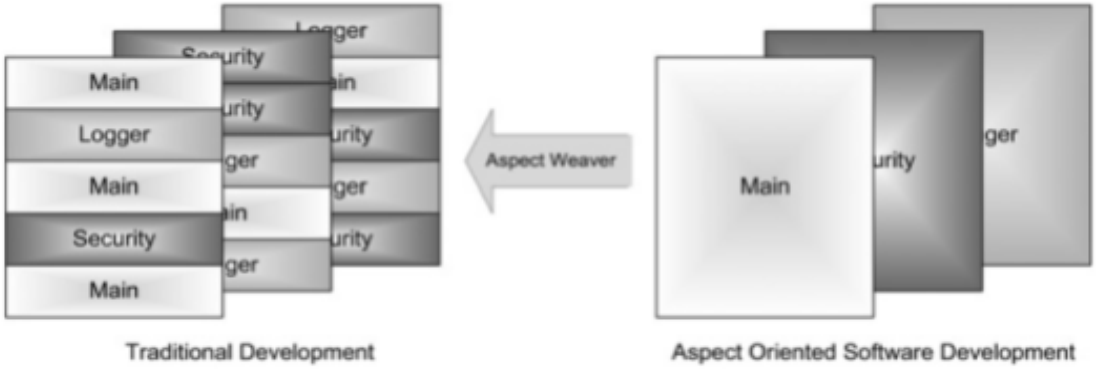
- Rearranging the components - Changing the architecture of a component, changing the connections and communication between components
- Introducing new component / Remove - Introducing redundant components, Adding new layers of security. Some of threats that can be solved using this approach are Preventing Buffer Overflows, DoS.

- Enclosing one or more of the components using a wrapper - Wrapping vulnerable components with standard security layers before opening them to public. Ex: Facade, Limited APIs
- Re-configuring one or more components - Changing the configurations of components. Ex: Changing log level configurations, Switching between development, Testing or deployment configuration, Changing file upload limit, Limiting number of users, Changing Database level configurations
- Directly adapting the code - Change the code and uploading it to the live server. Ex: Replace the compiled classes in the production environment, Changing HTML code.

When implementing above software solutions following state of the system must be carried,

- Reusable and maintainable code
- Dynamic system (Not shutting down the system/ Nodes)

With the AOSD some of the above solutions can be applied while supporting the latter two. The final application is the result of combining several modules. On one side there are the modules that contain the basic functionality; on the other side, the aspects with concrete functionalities that usually cut across the system functionality.



When the functionalities and aspects are developed we have to form the system. This process is called weaving.Traditionally we pass the source code to a compiler to obtain executable. In AOSD the code must also pass through the weaver, to obtain the program with full functionality. Falls under two:

1. Static Weaving : Majority of existing. slower, code is combined and optimized
2. Dynamic Weaving : Adaption, maintain basic functionality

An AOSD platform which support homogeneous static and dynamic weaving have

- Full dynamic weaving
- Platform independence
- Language independence
- Weave-time independence

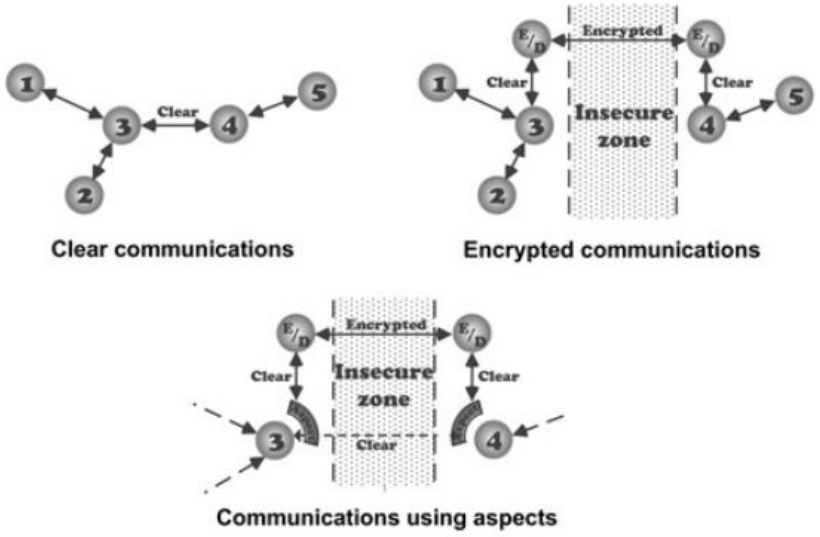
- Wide range of join-point

Access control

Access control /data flow restrict the access to the information, in order to prevent nodes without the appropriate permission. Traditionally solved by one-to-one access controlling/ configuring relationships in such, client server. But problematic in distributed networks such as p2p or fuzzy network which doesn't have fixed topology.
To do this, they use security policies with different levels of authorization for each node. These kinds of security policies are widely used in military systems or for catastrophe management, where access control of information is essential.

1. Authentication to grant the user the appropriate authorization level.
2. Encryption of information to avoid unauthorized access to it.
3. Data tagging to determinate how information flows across the network and to control access to it

Encryption



Above diagram shows their approach on using encryption. Rather than securing all the channels with encryption or leaving all channels un-encrypted they have choose the middle path. There are benefits of this approach.

- Focusing on the parts that needs high security.
- Reducing the performance impact added by encryption
- Reducing the extra overhead on decrypting the channels which does not need encryption.

Suggestions for improvements

The method suggested in the research(not discussed in detail in this report) is a comprehensive and concise implementation for the problem of managing change and maintaining security in a distributed system. Yet, due to the complexity and the less

adaptability of the proposed methodology it needs more improvements to reduce those effects.
Also, the framework should be supplements for documentation, testing, logging and monitoring for the framework to be perfected.

Reference

[1] : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6232029