

An Aspect-Oriented Security Framework: Lessons Learned

Viren Shah and Frank Hill

{viren,fhill}@cigital.com

Cigital Labs

Abstract

The security of a software system is an attribute that permeates the whole system. As such, any attempt to address security concerns in a software system must, of necessity, be global in nature, and be applied consistently at every relevant location. To implement a methodology that would encompass these two high-level goals -- separation of security concerns and consistent implementation of security solutions -- we chose to build an AOSD-based framework. In this paper, we describe the framework as well as obstacles we encountered in applying it to commercial projects.

1 Introduction

The paper is meant to serve as a basis of discussion for the workshop concerning potential obstacles that are faced in introducing AOSD to the software security community. In this paper, we first talk about and describe the AOSD-based framework we developed for addressing security issues. Based on our experiences developing this framework and trying to get it used, we present a list of obstacles that we encountered.

The objectives of the project were to devise a technique for addressing security issues in software systems that could be used as a framework for dealing with a wide range of concerns that affect the security of a system. The primary characteristics needed in the framework were:

- Separation of Concerns
- Proactive stance
- Global application
- Consistent implementation
- Adaptability
- Seamless integration

We were fortunate that the features listed above, as ones that needed to be embodied in the security framework, meld well with the strengths of the Aspect-oriented Software Development model.

Aspect-Oriented Software Development (AOSD) has been proposed as a technique for improving separation of

concerns in software. It builds on previous techniques, such as procedural and object-oriented programming, and adds another dimension of modularity. The basic concept behind AOSD is that hierarchical modularity mechanisms are inadequate to deal with concerns that need to be modularized but for which the implementation needs to be spread out over the application. In order to address this class of concerns, AOSD provides specific language mechanisms that make it possible to address these concerns in a modularized manner. This makes AOSD an ideal vehicle for addressing security concerns such that the goals mentioned earlier are met.

The focus of this paper is to describe our framework briefly in order to set the stage to present some of the obstacles we encountered in attempting to use it. Section 2 describes the framework and the security problems we addressed with it. Section **Error! Reference source not found.** lists some of the issues we encountered and have realized must be addressed for the technology to be deployed extensively.

2 The Aspect-Oriented Security Framework

The security framework we have implemented, the Aspect-Oriented Security Framework (AOSF), is simple, flexible and intended to integrate seamlessly into the build process. It was intended as a proof-of-concept to demonstrate the applicability of AOSD to the realm of application security. Our target applications were C-based for 2 reasons: firstly, we felt that this provided us with a wider range of security issues than a type-safe language such as Java (implementation flaws as well as higher-level security abstractions), and secondly, it gave us access to a broader set of applications (from legacy software to new projects). At the time the project was started, there were no other AOSD frameworks targeted at C. Hence, we decided to build the AOSF.

2.1 Framework Characteristics

The AOSF provides a broad base upon which to build solutions to security issues. In addition, it also fulfills the needs of a security framework, as listed in Section 1.

1. **Proactive Stance:** The framework described in this paper is designed to be used as part of the development process such that security can be applied to the software system by default.
2. **Global Application:** By treating security as a cross-cutting concern, the AOSF allows security analysts to apply security solutions globally, while giving them the flexibility to focus on particular parts of the system if needed.
3. **Consistent Application:** The AOSF alleviates the problem of inconsistent implementations of the same solution by automating the process of integrating the security solutions into the software system.
4. **Adaptability:** The framework provides a full-featured transformation engine and an expressive but simple language for encoding generic directives for security solutions. These ensure that the framework can be used to implement a wide-ranging set of security solutions.
5. **Seamless Integration:** One of the strengths of the AOSF tool is the ease with which it can be integrated into the build process. Since the core of the framework, the weaver, can masquerade as a second-stage pre-processor, the only change that the developer has to make is to the compiler directive in the application Makefile.

A large part of the above listed characteristics are inherited due to the use of a AOSD-based approach. As far as the

2.2 Addressing Security Issues

The aspect-oriented framework described above has been used to address several common security problems. The project had two phases. In phase 1, we concentrated on addressing causes of the more prevalent security exploits, such as:

- Buffer overruns
- Time-of-check-to-time-of-use (TOCTTOU)
- Format string vulnerabilities

These types of security issues tended to be straightforward implementation flaws and as such require a simple point solution to fix them. This, under the AOSF, was done through aspects that consisted of simple one-to-one syntactic transformations of function calls and code constructs. The aspects only needed local context at the target locations and thus were simple, small and minimally invasive.

While addressing simple implementation-level flaws is important for creating secure software, it doesn't address the more complex security problems that tend to be more than just simple implementation flaws. The second phase of the project focused on extending the framework to be able to address these more advanced security issues. These issues are ones that require more of a global perspective and context and are more aptly considered to be design or architecture level issues than implementation problems. These types of issues include concerns such as:

- Protection of communications channels
- Event ordering enforcement
- Type safety

Aspects addressing these concerns tend to require more complex logic as well as needing some customization based on the application to which the aspect is being applied. For example, in the aspect dealing with protection of communication channels, the aspect wasn't self-sufficient in that we did not build in all of the required infrastructure for the ftp application we targeted. Similarly, in the event ordering enforcement aspect, we needed to develop a separate notation using which security analysts or developers could describe the event ordering policy for the aspect.

3 Lessons Learned

The experience of developing the framework as well as talking to various developers about it made us realize that there were several obstacles to using our framework in practice.

1. **The KISS Principle:** Introducing a new language (even if it is a minimal superset of the development language) will meet resistance from developers and the QA teams. Aspects tend to be invalidate any concept of well-defined and narrow interfaces. This leads to added complexity for the development team.
2. **Shifting development paradigms:** The biggest obstacle was getting software professionals to see the value in moving to a new development paradigm. While the concept of separating out the security concerns and allowing for security logic to be written by security-aware developers was accepted, the immediate practicalities of the situation gave rise to several problems. These can range from sociological ones such as trusting aspect code that may not be written by the development team to process problems such as the difficulty in moving from non-AOSD to AOSD development.
3. **Traceability:** The framework we provided did not have the infrastructure to allow the

development team to maintain proper traceability throughout the development process. Consider the difficulties inherent in debugging and tracing problems to their root cause: the application code, the aspect, unexpected interactions between the code and the aspect, aspect interactions or a bug in the aspect weaver.

4. **Early lifecycle Security Abstractions:** The AOSF allowed developers to separate out security concerns at the code phase. However, this ability could not easily be reflected in the design and architecture phases i.e. being able to properly annotate architecture and design diagrams to note use of aspects and features being implemented via aspects. The ability to reflect code level security concerns in design (and vice versa) is critical to the proper integration of AOSD-based application security.
5. **Tool Support:** The lack of support tools was not just an obstacle to using the AOSF in practice, but also to the team developing the framework itself. Simple tools such as IDE integration and debugger support were not present and, as such, made the framework almost unusable until they were made available.

4 Conclusion

This paper describes briefly the work we have done with an AOSD-based framework to address security issues. As part of this effort, we encountered several issues that prevented us from fielding the AOSF. Some of the issues were due to the immaturity of our framework and can easily be rectified with more effort. However, some of the issues are of more import to the community as a whole and specifically to expanding the use of AOSD within the software security realm.

Acknowledgment

This work has been funded in part by DARPA contract #F-30602-00-C-0079.