AOP EXTENSION
FOR SECURITY TESTING OF PROGRAMS

Nadia Belblidia
CSL-CIISE,
Concordia University
Montreal, Quebec, Canada
na.belblidia@ciise.concordia.ca

Mourad Debbabi
CSL-CIISE,
Concordia University
Montreal, Quebec, Canada
debbabi@ciise.concordia.ca

Aiman Hanna
Computer Science,
Concordia University
Montreal, Quebec, Canada
ahanna@ciise.concordia.ca

Zhenrong Yang
CSL-CIISE,
Concordia University
Montreal, Quebec, Canada
zhen_ya@ciise.concordia.ca

Abstract
The purpose of this paper is to use the aspect-oriented programming (AOP) paradigm for security testing. AOP allows security experts to develop and inject separate modules for conducting security testing on the applications independent of their business logic. After an appropriateness analysis of the mostly used approaches for AOP, we retain the pointcut-advice model. The pointcut-advice model is significantly better than the other approaches for security testing. However, the current set of pointcuts is insufficient for the purpose of security testing and needs to be extended with new pointcuts.

Keywords — AOP, Security, Testing, Pointcut, Instrumentation.

1 Introduction

Aspect Oriented Programming (AOP) [14] is a new paradigm that improves software modularity by allowing encapsulation of crosscutting concerns such as security, synchronization, etc. Thankful to a weaving mechanism, AOP languages allow merging codes in a natural and systematic way. AOP is ideal for conducting security testing on free and open source software (FOSS) products. With free and open source software, the code is available and ready to be merged with appropriate security testing aspects. The purpose of this paper is to elaborate a framework for the security testing of free and open source software using aspect oriented paradigm.

One of the security issues is poor memory management, which can lead immediately to security vulnerabilities. For example, by overflowing an internal buffer used by Linux ftplib’s SSL support, a user is able to open a shell on the remote host [2]. One way to test security problems related to run-time memory errors is by instrumenting the source code with monitoring routines in order to facilitate the detection of these errors at runtime. The fact that the source code of FOSS is available makes this instrumented approach feasible. In this paper, our purpose is to elaborate a framework for the security testing of free and open source software using aspect oriented paradigm.

The rest of the paper is organized as follows. Section 2 presents the aspect oriented programming paradigm and its most important approaches. Section 3 reports on security testing. Section 4 discusses the use of the AOP paradigm for security testing and proposes an extension in support of it. Section 5 summarizes the state of the art of works dealing with our research topic. Finally, section 6 discusses the future work and provides a conclusion.

2 AOP Approaches

AOP aims at avoiding scattering and tangling phenomena by separating the basic functionality of the software from other aspects like security, synchronization, etc. Applying AOP leads to improved key software quality attributes as reusability and evolvability. AOP techniques are based on the notion of weaving, which is the process of composing core functionality modules with aspects into one single application. We present in the following the three mostly used AOP approaches: “Pointcut-advice model”, “multi-dimensional separation of concerns” and “adaptive programming”.

The fundamental concepts of the pointcut-advice approach are: join points, pointcuts and advices. A join point is a point in the dynamic call graph of an application. A pointcut is a constructor that designates a set of join points and advices are pieces of code attached to pointcuts. An advice is executed when join points satisfying its pointcut are reached. Generally, three kinds of advices are available: before, after and around. The before advice and the after advice are executed before and after the intercepted join point, respectively. The case for the around advice is more subtle. Inside the around advice, we can choose to resume the intercepted join point or skip it. AspectJ [15], an aspect oriented extension of Java, is probably the most known representative of pointcut-advice languages and AOP languages in general.

Multi-dimensional separation of concerns (MDSOC) allows simultaneous separation according to multiple, arbitrary kinds (dimensions) of concerns [19]. MDSOC treats all concerns as first-class and co-equal, including components and aspects, allowing them to be encapsulated and composed at will. One particular approach to MDSOC is called hyperspaces where the software is modelled as a set of units called hyperslices(concerns). HyperJ [19][24] is the support for hyperspaces in Java. In HyperJ, an hyperslice is a set of modules representing only one single concern and
could be composed with other hyperslices using matching units (e.g. method names).

Adaptive programming (AP), proposed by Demeter group [10], has used the ideas of AOP several years before the name aspect oriented programming was coined. The Law of Demeter is a programming style rule for loose coupling between the structure and behavior concerns. Following this law can result in a large number of small methods scattered throughout the program. Adaptive programming with traversal strategies and adaptive visitors avoids this problem while even better supporting the loose coupling of concerns.

3 Security Testing

Security testing is about validating an application’s security services and identifying potential security flaws. Security testing may seem similar to traditional software testing in terms that testers select input scenarios to torment applications in an attempt to force the software to fail. However, the two disciplines target different problems in a software application. Traditional software testing detects program faults that do not implement the system requirements correctly. On the contrary, security testing looks for “unintended, undocumented, or unknown functionalities”[11] in a software application. For instance, incorrect use of memory such as the infamous format string vulnerability [22] may produce an unintended functionality of a software application such that attackers can execute arbitrary codes by inputting a malicious string. There are many other C/C++ errors such as memory corruption, memory leaks, memory allocation errors, variable initialization errors that may lead to security flaws as well. These security flaws may be exploited, ranging from crashing the application to an attacker inserting and executing malicious code in the application process.

We intend to leverage source code instrumentation in order to detect memory management faults. We inject analysis code to a program and perform dynamic analysis, e.g. probing application context, to expose illegal pointer operations and memory manipulations in the application.

4 AOP for Security Testing

In this paper, we propose to use the AOP paradigm to systemize and automate the injection of analysis code. In the following section, we will choose the most appropriate AOP approach for this purpose.

4.1 AOP Appropiateness Analysis

In this section, we compare the appropriateness of the AOP approaches cited in Section 2 for security testing, more precisely for detecting memory management errors.

The multiple separation of concerns has a serious limitation that it works at the method granularity. It can not operate within a method body. A very basic and simple example of security testing that we can not handle with such approach is to pull apart of security testing code within method bodies.

The adaptive programming is concerned with a loose coupling between structure and behavior and focuses on certain kinds of concerns. They include behavioral concerns that deal with groups of collaborating objects that offer new behavior, the traversal concern and the object structure concern [17]. With this approach, we can not for example probe arguments to method calls.

The pointcut-advice model is the most popular model. It offers a better granularity than multiple separation of concerns approach and considers a more general kinds of concerns than the adaptive programming. Furthermore, the pointcut-advice model takes more of a “pull” approach. It allows to track subtle points in the control flow of the application as for example points where methods are invoked, etc. For these reasons, we choose this approach to facilitate the code instrumentation for security testing. Because our security testing targets C and C++ codes, we decide to use the AspectC++ language [1], which is an aspect-oriented extension for C/C++. However, it needs extension as we will show in the next section.

4.2 AspectC++ Extension

AspectC++ [21] is an open source project of AOP extension for C++. It supports two types of pointcuts: “code pointcuts” and “name pointcuts”. While name pointcuts describe known program entities, code pointcuts describe collection of points in the control flow of an application at runtime. Code pointcuts are created using name pointcuts because they require at least one name to be defined. For example, “\texttt{int C::\%(...)\texttt{}}” is a name pointcut that matches all member functions of the class C that return \texttt{int} and “\texttt{call(int C::\%(...)\texttt{}})” is the code pointcut describing the set of calls of these functions.

None of the code pointcuts given by AspectC++ allows to target local/global variable accesses in C/C++ code. However, for monitoring memory management operations, there is a need to dynamically keep track of local variable at runtime as demonstrated in the following example.

4.2.1 Motivating Example

The example presented in Table I performs simple pointer and string manipulation. Although the code compiles and executes, data are written to unallocated memory as seen in line 12. An attacker may take advantage of this flaw to inject malicious code. In order to detect such vulnerabilities, we need to insert monitoring routines before and after variable accesses. Using AspectC++, this implies the definition and implementation of new pointcuts.

4.2.2 New Pointcuts

We introduce a new name pointcut in order to match variables in C and C++ codes. After this, we define three new code pointcuts \texttt{declare(pointcut), get(pointcut)} and \texttt{set(pointcut)} on this new name pointcut, in order to keep
The most relevant related work are those using AOP for security. However, research results are rare. Among the attempts that have been made to use AOP for security we can cite the Darpa-Funded project of Cigital Labs [16][20][27] that applies AOP to enforce secure code practices. The main outcomes of this project are a security dedicated aspect extension of C called CSAW [16] and a weaving tool. CSAW addresses local, small-sized problems as buffer overflows and data logging. De Win et al [4][6][8][25][26] explored the use of AOP to integrate security aspects within applications. In [7], De Win et al applied AspectJ to enforce access control and in his thesis [5], De Win modularized the auditing and access control features of the FTP server. Another contribution in the use of security aspects is the security aspect library JSAL [12], which is implemented in AspectJ and provides security functions. There is another kind of contribution in the field of AOP for enriching the expressiveness of pointcuts for some goals. In a security perspective, Masuhara and Kawauchi [18] present a new pointcut called dataflow pointcut. They show in their papers, that some security concerns, such as secrecy and integrity, are sensitive to flow of information in a program execution.

None of those previous works dealt with the issue of memory management flaws that we address here.

6 Future Work and Conclusion

In this paper, we study the possibility of using AOP for security testing. It appears that the pointcut advice model is the most appropriate one. However, we have shown that this model still needs extension. We propose new pointcuts to keep track of variable accesses. We plan in near future to finish the implementation of the extension and use the extended AspectC++ to conduct security testing on FOSS applications.

References

### TABLE II: New AspectC++ Pointcuts

<table>
<thead>
<tr>
<th>CodeVariablePointcut</th>
<th>::=</th>
<th>NameVariablePointcut</th>
<th>CodeVariablePointcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeVariablePointcut</td>
<td>::=</td>
<td>get(NameVariablePointcut)</td>
<td>set(NameVariablePointcut)</td>
</tr>
<tr>
<td>NameVariablePointcut</td>
<td>::=</td>
<td>Type</td>
<td>Namespace</td>
</tr>
</tbody>
</table>

650


