

# **Joys and horrors of aspect-oriented programming**

**Bart De Win**

**Secure Application Development Course, 2008**

## **Outline**

- **Motivation for AOP and Security**
- **AspectJ in a nutshell**
- **AOP and Security in practice**
- **Security implications**
- **Conclusion**

# Causes for software security problems

- Software security is an interesting and challenging area
- Problems herein are related to:
  - Security domain
    - Complexity and sensitivity of theories
    - Bug sensitivity of implementations
  - Secure software characteristics
    - Pervasiveness of security
      - Secure coding
      - Security mechanisms are crosscutting
    - Unanticipated risks and change of environment
  - Trade-offs security-usability, security-performance, and so forth
  - ...

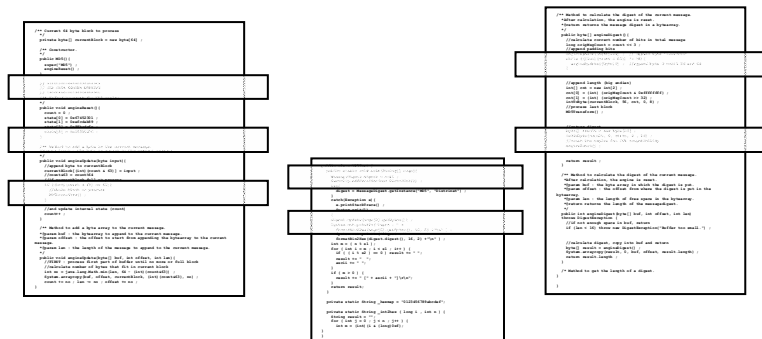
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

3

# Secure coding

- Security is crosscutting in location



March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

4

## Secure coding (ctd.)

- Typical examples:
  - Buffer overflow
  - Input validation
- Often repetitive and, hence, developers tend to forget about it
- Coding guidelines, or compiler and run-time support can be helpful
- No general-purpose solution exists:
  - Canonicalization errors
  - Race conditions

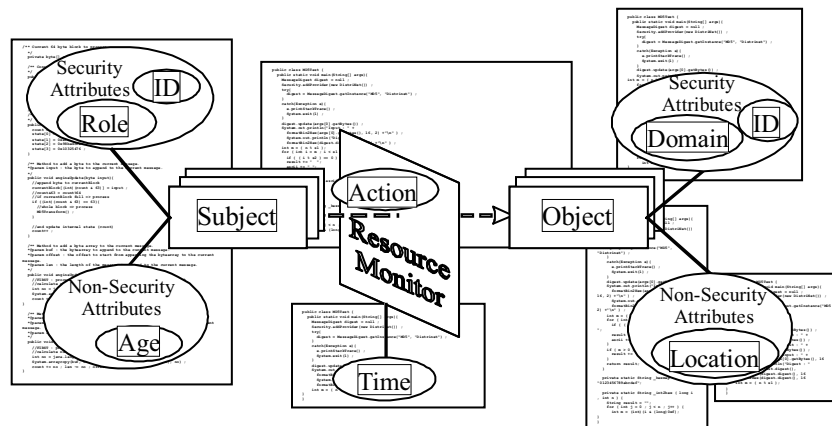
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

5

## Security mechanisms are crosscutting

- Security is crosscutting in structure



March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

6

## **Security mechanisms are crosscutting (ctd.)**

- **Examples:**
  - Access control (e.g., resources within jdk 1.6)
  - Confidentiality
  - Privacy
- **Modular security engines are only a partial solution**
  - Where to invoke ?
  - How to access parameters ?
  - Where to store security state ?
- **Particularly problematic for fine-grained security requirements**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

7

## **Security: an evolving property**

- **Security of a system is often implemented once and for all**
  - E.g., inspired by the Common Criteria
- **Utopic, because of unanticipated changes**
  - Incomplete threat analysis
  - New functional requirements
  - Design optimizations for NFR's
  - Changes in the system's environment

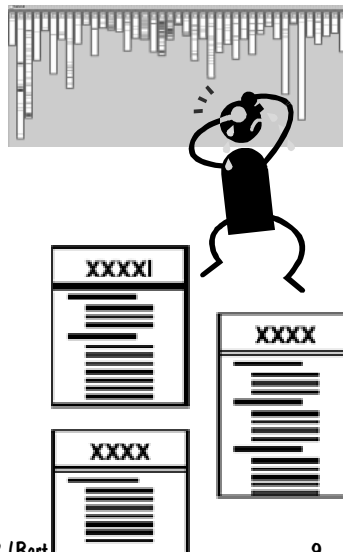
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

8

# Resulting Problems

- **Scattering**
  - The specification of one property is not encapsulated in a single module
- **Tangling**
  - Each module contains descriptions of several properties or different functionalities



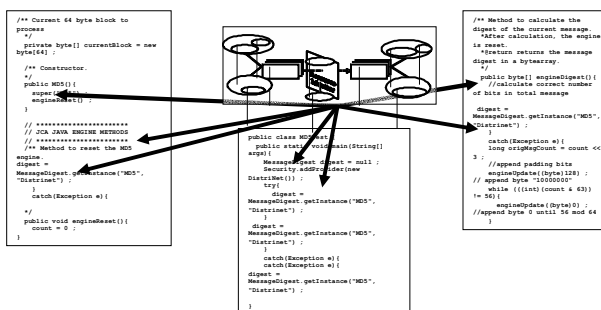
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart

9

# AOP to the rescue ...

- To optimize the modularization of application-level security



March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

10

## **AOP to the rescue ... (ctd.)**

- **Rationale**
  - Combines advantages of declarative and programmatic software security
  - Addresses pervasiveness and evolution issues
  - Application developer is less "bothered" with security
- **Useful for many security problems**
  - Different forms of access control (role based, owner based, ...)
  - Securing sensitive application data bmo. cryptographic functions
  - Privacy and anonymity policies
  - ...

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

11

## **Outline**

- **Motivation for AOP and Security**
- **AspectJ in a nutshell**
- **AOP and Security in practice**
- **Security implications**
- **Conclusion**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

12

## AspectJ in a nutshell

- **A general-purpose AO language**
  - Features AO-specific extensions to Java
  - De facto standard for the core concepts of many AO tools
  - Static and dynamic language features
- **An extension to Java**
  - Outputs .class files compatible with any JVM
  - All Java programs are AspectJ programs
  - Supports source-code and byte-code weaving
  - Support for Java5 annotations
- **Commercial sponsors**
  - It originated from Xerox Parc
  - Currently being maintained by IBM
- **IDE support**
  - Nice Eclipse plugin (AJDT)

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

13

## Joinpoints and pointcuts

- A joinpoint is a point in the dynamic execution of the software
- Different types are supported:
  - Method & constructor call
  - Method & constructor execution
  - Field access (get / set)
  - Exception handler
  - Initialization
  - Advice execution
- A pointcut selects a set of joinpoints based on a number of constraints

```
public class MyPolicy extends Policy {
    private Permissions perms ;

    public MyPolicy(){
        super();
        perms = new Permissions() ;
        try{
            <read permissions from file>
            this.verifyPermissions();
        }
        catch(IOException e){System.err.println(e);}
    }

    private void verifyPermissions(){
        if (perms == null) return false ;
        ...
    }
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

14

## Advice

- **Advice adds behavior to a (set of) joinpoint(s):**
  - Similar to a method
  - Is executed before / after / around the joinpoint
  - For around advice: `proceed()` to resume the action at the specific joinpoint

```
before(): execution(void Foo.m(int)) {  
    System.out.println("M is executed");  
}
```

```
void around(): set(Foo.field) {  
    System.out.println("Are you sure?");  
    if(<confirmed>){  
        proceed();  
        System.out.println("Foo.field changed");  
    }  
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

15

## Advice parameterization

- **Just as regular methods, advice can be parameterized**
  - Values come from the joinpoint context
  - All parameters must be matched within the pointcut
  - Use `this()`, `target()`, `args()`

```
before(int i): execution(void Foo.m(int)) && args(i) {  
    System.out.println("M is executed with argument" + i);  
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

16

# Aspects

- **Any combination of:**

- **Members**
- **Methods**
- **Named Pointcuts**
- **Advices**

```
aspect MyAspect{
  int test;
  int double(int j){return 2*j ;}

  pointcut p(): call(* Foo.*(..) );

  before(): p(){
    System.out.println("Boo") ;
  }
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

17

# Aspects (ctd.)

- **Aspects can be declared 'privileged'**
  - **Have access to protected/private class members or methods**
- **Advices are ordered based on standard rules**
  - **Can be influenced by specifying ordering constraints explicitly**

```
declare precedence: Security, Logging, * ;
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

18

## Aspect instantiation

- **Aspects are instantiated automatically**
  - Cannot be created explicitly by the developer
- **Aspects are associated to a particular 'context'**
  - Normally, one aspect per JVM (`issingleton()`)
  - Alternatives: `perthis()`, `pertarget()`, `percflow()`, `pertypewithin()`
  - Restricts the scope of advice application !
- **Association operators**
  - Requesting reference `MyAspect a = MyAspect.aspectOf(<instance>);`
- **Useful to manage concern-specific state**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

19

## Outline

- **Motivation for AOP and Security**
- **AspectJ in a nutshell**
- **AOP and Security in practice**
- **Security implications**
- **Conclusion**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

20

## Potential usage scenarios

- **Policy enforcement**
  - Implementation (green field or add-on)
    - Also reverse (e.g., disabling license checks)
- **Policy mining and monitoring**
- **Coding guidelines**
  - Implementation
- **Security testing**
- **Verification of correct use**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

21

## Policy Enforcement

- **Most interesting category**
  - Applies the full potential of AOP
- **All about finding ways to 'bind the security engine'**
- **Design activity => many alternative solutions**
  - Consider typical SE properties
  - Non-functional qualities

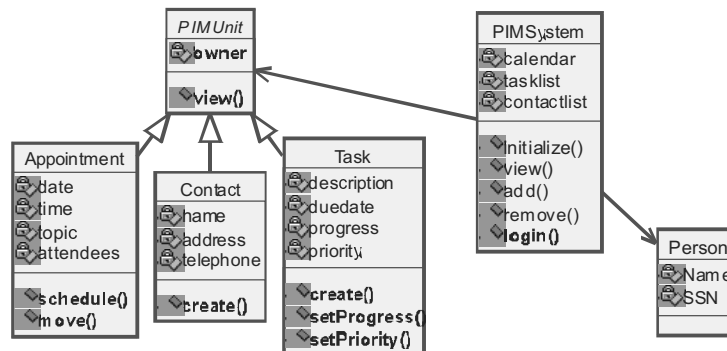
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

22

## Policy Enforcement – PIM

- Policy:
- PIM Unit owners can invoke all operations
  - Contacts only accessible to their owners
  - All other accesses restricted to viewing



March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

23

## PIM security using AspectJ

```

aspect Authentication{
    private static String currentUser ;

    static String getUser(){
        if(currentUser == null){
            currentUser = <login>;
        }
        return currentUser ;
    }
}
    
```

```

aspect OwnerManagement
    perthis(this(PIMUnit)){
        String owner ;

        after(): execution(PIMUnit.new(..)){
            owner = Authentication.getUser() ;
        }
    }
}
    
```

```

aspect Authorization{
    pointcut restrictedAccess():
        execution(* Appointment.move(..) ||
        execution(* Contact.view(..) ||
        execution(* Task.setPriority(..) ||
        execution(* Task.setProgress(..));
    
```

```

    void around(PIMUnit p): restrictedAccess() && this(p){
        //are owners identical ?
        if(! OwnerManagement.aspectOf(p).owner.equals(
            Authentication.getUser()))
            throw new RuntimeException("Access denied !");
        else proceed() ;
    }
}
    
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

24

## Policy Enforcement – PIM w/ JAAS

- With JAAS, Java offers:
  - a pluggable mechanism for authentication
  - an extensible mechanism for authorization based on the subject running the code
- JAAS can be integrated seamlessly using AOP

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

25

## Using JAAS (ctd.)

```
aspect Authentication{
    private static Subject currentUser ; //one per session
    public static LoginContext lc = null ;

    static Subject getUser() {
        if(currentUser == null){
            try{
                lc = new LoginContext("PIM", new TextCallbackHandler()) ;
                lc.login();
                currentUser = lc.getSubject() ;
            }
            catch(Exception e){throw new RuntimeException(e) ;}
        }
        return currentUser ;
    }
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

26

## Using JAAS (ctd.)

```

aspect Authorization{
    pointcut restrictedAccess(): execution(* Appointment.move(..) || execution(* Contact.view(..));

    //Activates a .doAsPrivileged with the currently executing subject
    void around(): restrictedAccess() && !flowbelow(restrictedAccess()){
        try{
            Subject.doAsPrivileged(Authentication.getUser(), new PrivilegedAction(){
                public Object run() {
                    proceed();
                    //No result is required for these particular operations
                    return null;
                } },null);
        }
        catch(Exception e){e.printStackTrace();}
    }

    //Checks whether the correct OwnerPermission is owned
    before(PIMUnit u): restrictedAccess() && this(u){
        Subject owner = OwnerManagement.aspectOf(u).owner;
        OwnerPermission op = new OwnerPermission(owner);
        AccessController.checkPermission(op);
    }
}

```

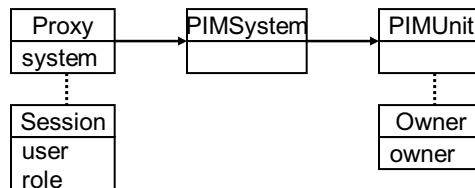
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

27

## Policy Enforcement – PIM w/ sessions

- **Goal:**
  - Introduce roles (simplified RBAC)
  - Support multiple sessions
- **3 steps**
  - Introduce sessions
  - Make session accessible for authorization
  - Adapt authorization and owner management



March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

28

## PIM extended - sessions

```
//A proxy to integrate multiple sessions into PIMSystem.
public class PIMSystemProxy extends PIMSystem{
private PIMSystem pim = null ;

public PIMSystemProxy(PIMSystem s){
pim = s ;
}

public void initialize(){
//initialization is handled in the SessionManager.
}

public void add(PIMUnit u){
pim.add(u) ;
}

public void view(){
pim.view() ;
}
}

//Contains information about the particular session.
public aspect Session perthis(this(PIMSystemProxy)) {
String user ;
String role ;

after(PIMSystemProxy p): execution(PIMSystemProxy.new(..) && this(p) {
//Shortcut for role assignment ...
user = Authentication.getUser() ;
role = RoleManager.getUserRole(user) ;
}
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

29

## PIM extended - sessions

```
//Responsible for managing multiple sessions for the PIMSystem.
//PIMSystem is made singleton.
public aspect SessionManager {
private PIMSystem pim=null ;

//intercept the creation of new PIMSystems in order to return PIMSystemProxys ...
PIMSystem around(): call(PIMSystem.new(..) && ! within(SessionManager){
if(pim == null){
pim = new PIMSystem() ;
pim.initialize() ;
}
return new PIMSystemProxy(pim) ;
}

//ownermanagement adapted to sessions
before(PIMUnit u): execution(* PIMSystem.add(..) && args(u) {
OwnerManagement.aspectOf(u).owner = SessionContextPassing.aspectOf().session.user ;
}
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

30

## PIM extended – making sessions accessible

```
public aspect SessionContextPassing perCflow(execution(* PIMSystemProxy.*(..))) {  
    Session session = null ;  
  
    before(PIMSystemProxy p) : execution(* PIMSystemProxy.*(..) && this(p){  
        this.session = Session.aspectOf(p) ;  
    }  
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

31

## PIM extended – revisit authorization

```
aspect Authorization{  
    pointcut normalAccess():  
        ...  
  
    void around() : normalAccess() {  
        //Secretaries are allowed to view contacts  
        if (thisJoinPoint.getThis().getClass().getName().indexOf("Contact") != -1  
            && SessionContextPassing.aspectOf().session.role.equals("secretary"))  
            proceed() ;  
        //Standard policies  
        else{  
            if(! OwnerManagement.aspectOf(thisJoinPoint.getThis()).owner.equals(  
                SessionContextPassing.aspectOf().session.user))  
                throw new RuntimeException("Access Denied !") ;  
            else proceed() ;  
        }  
    }  
  
    before() : execution(* Appointment.nonexistent(..) || execution(* Appointment.move(..)){  
        System.out.println("Not OK");  
    }  
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

32

## Policy mining and monitoring

- **Goal: instrument the application in order to**
  - deduce information about policy requirements
  - monitor the application to verify whether the current policy meets the risks of the execution environment
- **Heavily dependent on the particular goals and application**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

33

## Coding guidelines

- **Typical usage is insertion of extra security tests**
- **Nature of tests:**
  - Localized, scattered
  - Specific (often difficult to generalize)
- **Example of input validation:**

```
aspect InputValidation {
    pointcut inputcheck(): call (String InputStream+.read(char[])) ;

    after(char[] arr): inputcheck() && args(arr) {
        <validate arr>
    }
}
```

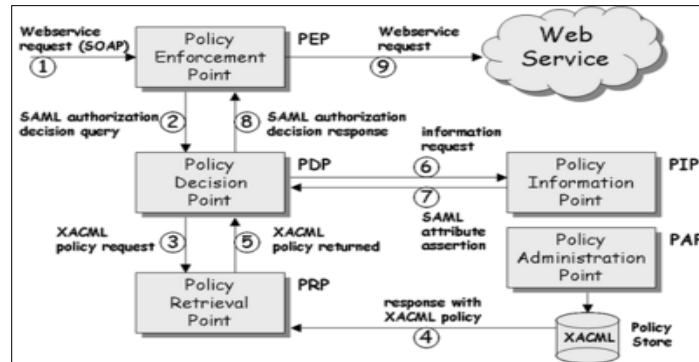
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

34

# XACML binding

- XACML: OASIS standard that specifies
  - Formats for access control policy and message flow
  - A model for access control enforcement



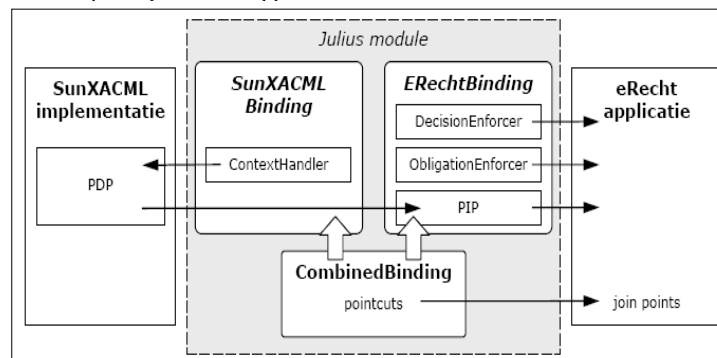
March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win) (Source: <http://www.security.nl/>)

35

# XACML binding

- Goal:
  - Generic, pluggable implementation
  - Easy integration in applications (PEP, PIP)



March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

36

## Outline

- **Motivation for AOP and Security**
- **AspectJ in a nutshell**
- **AOP and Security in practice**
- **Security implications**
- **Conclusion**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

37

## Discussion of AOP benefits

- **Abstraction**
  - Reasoning about one problem (or concern) at a time
  - Caveat: not all AOP tools offer modular reasoning !
- **Verification**
  - Improves inspection capabilities for the security binding
  - Avoids incomplete mediate errors
- **Reuse**
  - Part of the security binding can be made reusable
  - As a result, the security engine/library cannot be composed wrongly
- **Evolution**
  - More localized changes facilitates the maintenance of software
  - Caveat: AOP and the evolution paradox

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

38

## Problem statement

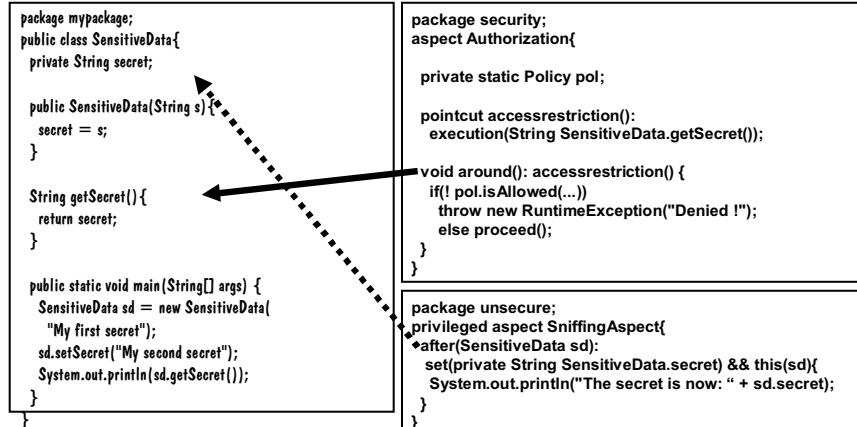
- **Software vulnerabilities are to a considerable degree due to the complexity of:**
  - Software engineering (pervasiveness)
  - Security (algorithms, domain knowledge)
- **Aspect-Oriented Programming (AOP) has shown to be helpful**
  - From a software engineering perspective...
    - Increased modularization improves specialization, verification and manageability
  - But what about the security perspective?
    - Do we really end up with secure software?
    - Statements have been made about this, but little published work is available

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

39

## A motivating example ...



March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

40

## Language-level issues

- Invocation parameters can be modified
  - Imagine the following aspect ...

```
aspect PolicyMod{
    pointcut polcheck(): execution(boolean Policy.isAllowed(..));

    //consult the policy, but always return true
    boolean around(): polcheck(){
        boolean res = proceed();
        return true;
    }
}
```

- Parameters presented to a security engine could be modified as well
- Invocations can be redirected or even discarded entirely:
  - Use a less restrictive Policy object
  - DoS scenarios

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

41

## Language-level issues (ctd.)

- **Privileged aspects**
  - Private internals of classes and aspects can be accessed by privileged aspects
    - Log changes of private variables or executions of private methods
    - Inspect and modify private, security-related attributes
    - Access cflow associations
    - Access inter type declarations
  - As a result, it becomes very hard to protect security-specific information
- **Remark: only possible using weaving-based AOP tools**
  - Allows one to “play” with Java’s type safety rules (at least, from a developer’s perspective)
  - Important to realize the impact on security verification (e.g., information flow)

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

42

## Tool specific problems

- **AspectJ 5 uses dangerous transformations:**
  - When using privileged aspects to access private members, a public method with a 'predictable' name is introduced in the target class !

```
public class SensitiveData{  
  
    //method generated to access the private secret datamember  
    public static String ajc$privFieldGet$unsecure_SniffingAspect$mypackage_\\  
    SensitiveData$secret(SensitiveData sensitivedata){  
        return sensitivedata.secret;  
    }  
  
    <snip>  
}
```

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

43

## Tool specific problems (ctd.)

- Package restricted aspects are transformed into public classes
- Private inter-type declaration members are transformed into public members in the target class
- **AspectJ compiler must control ALL the code in order to guarantee "secure" code**
- **Access modifiers are checked at compile time. What about run-time execution?**
- **Most probably, there will be other issues ...**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

44

## Other risks

- **Use of wildcards in PCD's**
  - Based on syntax instead of semantics
  - Difficult to predict the effect in case of system evolution
- **Aspect circumvention**
  - Based on woven code prediction (possibly multi-pass)
  - Used to be possible in the past, but seems solved with newer compiler versions
- **Load-time weaving**
  - Seems like a small step from a softw. eng. perspective, but from a security point of view it is a different model!
  - The unpredictability increases:
    - What in case of new classes?
    - Can the set of aspects be changed at runtime?
  - The use of LTW should be restricted to systems that have correct compile-time weaving behavior

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

45

## Risk synthesis

- **Security risks are related to:**
  - Modification of the logic of a module
  - Influencing the interaction or composition of modules
  - Enforcement of the aspect model
- **This can occur intentionally or unintentionally**
  - An ignorant developer could introduce security vulnerabilities without even knowing it
  - Addressing these is key

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

46

## **Best practices for implementation**

- **Use specific pcd's (be careful with wildcards)**
- **Avoid the use of privileged aspects**
- **Use aspects that operate at interface level as much as possible (consider to refactor your application)**
- **Structure aspects in packages**
- **Specify aspect ordering, especially for security aspects**
- **Consider verifying coding guidelines to support this**

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

47

## **Best practices for development**

- **Avoid using AOP for high-risk components**
  - E.g., attack surface, security kernel, ...
- **Avoid using different 'sets' of aspects**
  - Pro-actively try to identify feature interactions
- **Make sure that aspects are fully integrated in the development environment**
  - No separate compilation steps

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

48

## Conclusions

- The crosscuttingness of security is an important hurdle in the development of secure software
- AOP can optimize the modularization of application security
  - Improves reasoning and evolution properties
  - Different usage scenarios
- Be aware of the security implications => use wisely !
  - I would advise pro AOP for small, controllable, low/medium-risk projects
- Many issues in the area of AOSD & security are open research problems

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

49

## References

- AOSD & AspectJ
  - The AspectJ programming guide, semantic appendix and quick reference (<http://www.eclipse.org/aspectj/docs.php>)
  - Ramnivas Laddad, "AspectJ in Action", Manning Publications, 2003.
  - Stefan Hanneman and Arno Schmidmeier, "AspectJ idioms for Aspect-Oriented Software Construction", 8<sup>th</sup> EuroPLOP, June 2003.
  - Gregor Kiczales and Mira Mezini, "Aspect-oriented programming and modular reasoning", 27<sup>th</sup> International Conference on Software Engineering, May 2005.
  - Tom Tourwe, Johan Bricchau, and Kris Gybels, "On the existence of the AOSD-evolution paradox", AOSD Workshop on Software-Engineering Properties of Languages for Aspect Technologies (SPLAT), 2003.
- AOSD & security
  - Bart De Win, Frank Piessens, Wouter Joosen, and Tine Verhanneman, "On the importance of the separation-of-concerns principles in secure software engineering", ACSA Workshop on the Application of Engineering Principles to System Security Design, 2003.
  - Bart De Win, Wouter Joosen, and Frank Piessens, "Developing Secure Applications through Aspect-Oriented Software Development", Aspect Oriented Software Development, Addison-Wesley, 2004, pp. 633-650.
  - Viren Shah and Frank Hill. Using Aspect-Oriented Programming for Addressing Security Concerns, International Symposium on Software Reliability Engineering (ISSRE'2002), 2002.
  - Ron Bodkin, "Enterprise Security Aspects", Workshop on AOSD Technology for Application-level Security, 2004.

March 4, 2008

SecAppDev 2008: Joys and horrors of AOP (Bart De Win)

50