TECHNICAL REPORT

Report No. UI-SE-MDSERG-2022-05
Date: May 26, 2022

ATL Rules and OCL Queries Implemented in VAnDroid2

Atefeh Nirumand
Bahman Zamani
Behrouz Tork Ladani
Jacques Klein
Tegawendé F. Bissyandé

Department of Software Engineering
University of Isfahan
Hezar-Jerib Ave.
Isfahan

Tel: +98-31-37934537
Fax: +98-31-36699529 , +98-31-37932670
zamani@eng.ui.ac.ir
http://mdse.ui.ac.ir/TR

ATL Rules and OCL Queries Implemented in VAnDroid2

Atefeh Nirumand, Bahman Zamani, Behrouz Tork Ladani,
Jacques Klein, and Tegawendé F. Bissyandé
Department of Software Engineering
University of Isfahan
Isfahan, Iran.
{atefehnirumand, zamani, ladani}@eng.ui.ac.ir
{jacques.klein, tegawende.bissyande}@uni.lu

Abstract: *This report provides the implementation details of our proposed approach to detect inter-app vulnerabilities of Android applications. This approach, based on Model Driven Reverse Engineering (MDRE), has three phases. The feasibility and pertinence of this approach are demonstrated by developing an Eclipse-based tool called VAnDroid2 that implements the three phases of the approach. This tool, developed by Nirumand et al, automatically identifies two prominent inter-app vulnerabilities called Intent Spoofing and Unauthorized Intent Receipt.*

ATL Rules and OCL Queries Implemented in VAnDroid2

Atefeh Nirumand, Bahman Zamani, Behrouz Tork Ladani,
Jacques Klein, and Tegawendé F. Bissyandé
Department of Software Engineering
University of Isfahan
Isfahan, Iran.
{atefehnirumand, zamani, ladani}@eng.ui.ac.ir
{jacques.klein, tegawende.bissyande}@uni.lu

Abstract: *This report provides the implementation details of our proposed approach to detect inter-app vulnerabilities of Android applications. This approach, based on Model Driven Reverse Engineering (MDRE), has three phases. The feasibility and pertinence of this approach are demonstrated by developing an Eclipse-based tool called VAnDroid2 that implements the three phases of the approach. This tool, developed by Nirumand et al, automatically identifies two prominent inter-app vulnerabilities called Intent Spoofing and Unauthorized Intent Receipt.*

Contents

List of Figures

1 Introduction

Our proposed approach has three main phases of an MDRE process: (1) *Model Discovery* that extracts the initial models from each app without losing any information. (2) *Transformation and Integration* that uses the chain of model manipulation techniques to transform initial models into more management representations. (3) *Analysis* that uses the processed models in the previous phase to inter-app security analysis, and finally to generate the results in the form of XMI models. To detect inter-app vulnerabilities, the ATL and OCL rules have been used. This report describes the details of these ATL rules and OCL queries.

2 The ATL Rules and OCL Queries Used in the Transformation and Integration Phase

In this phase, the comprehension of the generated initial models takes place by raising the abstraction level of these models using model-to-model (M2M) transformations written in ATL language and obtaining higher-level representations of the Android system.

First, for each app, the security information of initial models is extracted and integrated into a single model called Android Application Security Aspects, using an M2M transformation. This model conforms to the proposed metamodel shown in Fig. 1. This metamodel is an extension of the proposed metamodel in VAnDroid [1].

Second, all Android Application Security Aspects models are received and transformed into a single Inter-Component Communication (ICC) model that conforms to the proposed metamodel shown in Fig 2. This model represents all potential inter-component communication inside the same app or on different apps.

In the following, in Section 2.1, some of the ATL transformation rules for extracting the Android Application Security Aspects model are shown. Then, in Section 2.2, some of the ATL transformation rules for extracting the ICC model are presented.

2.1 Extracting Android Application Security Aspects Model

For each Android app in a bundle, VAnDroid2 extracts all the security information and integrates them into a single model called the Android Application Security Aspects model by using model-to-model (M2M) transformation written in ATL language. This model conforms to the proposed metamodel shown in Fig 1. In the

following, some of the ATL rules used for this M2M transformation are shown.

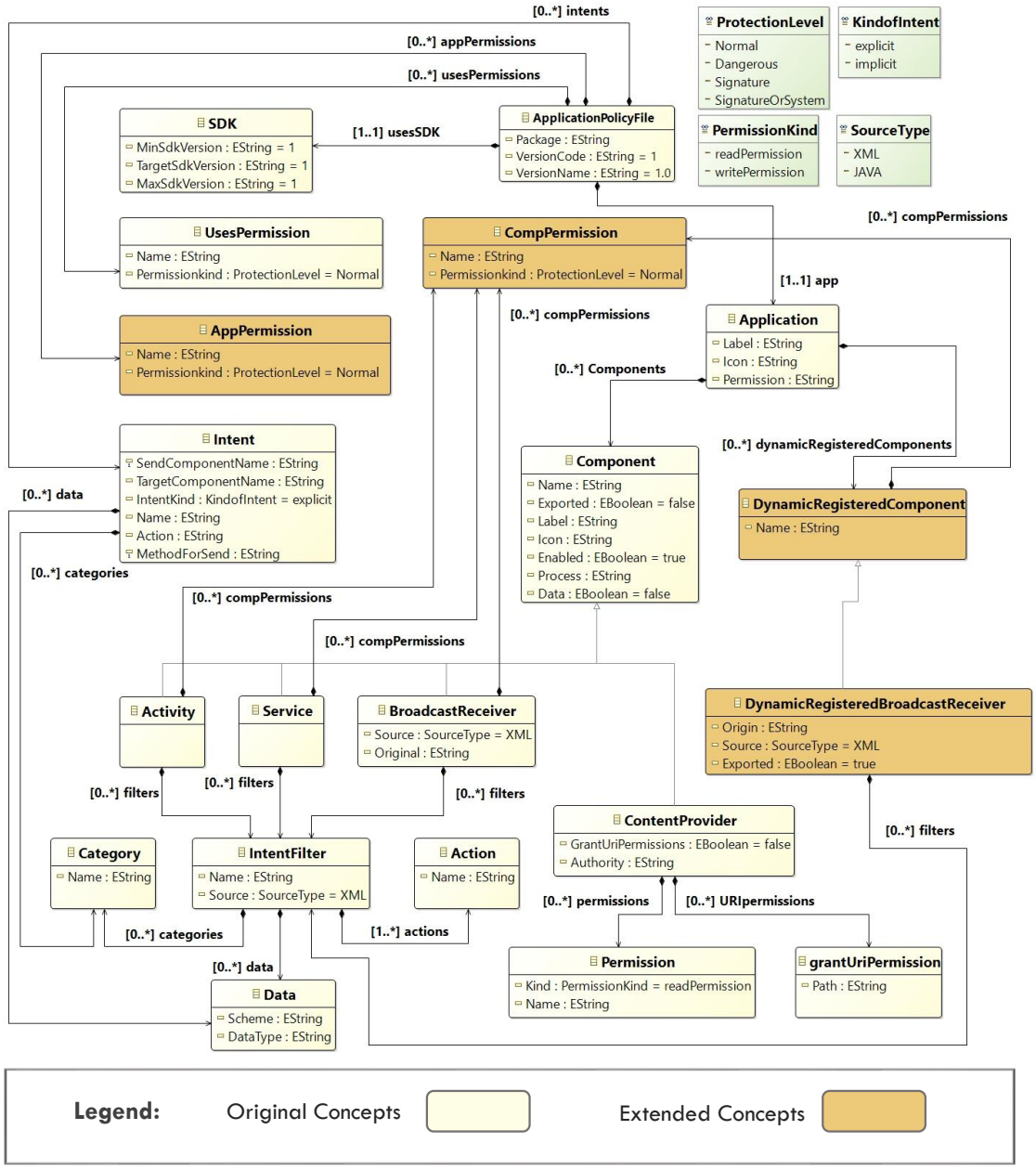

Figure 1: Android Application Security Aspects Metamodel (Extended From [1], The Extended Concepts Are in Dark Color).

Listing 1: ATL Rule to Create ApplicationPolicyFile Element.

```
--for transformation Root TO ApplicationPolicyFile:
rule Root2ApplicationPolicyFile {
from
s: XML!Root (s.name = 'manifest')
to
t: ASAMM!ApplicationPolicyFile (
Package <- if s.getAttribute('package').oclIsUndefined() then
 OclUndefined
else
s.getAttribute('package').value
endif,

VersionCode <-
if s.getAttribute('android:versionCode').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:versionCode').value
endif,

VersionName <-
if s.getAttribute('android:versionName').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:versionName').value
endif,

usesPermissions <-
s.children ->select(c | c.name='uses-permission'
  or c.name='uses-permission-sdk-23'),

app <-
s.children -> select(c | c.name='application'),

appPermissions <-
s.children -> select(c | c.name='permission')->
    collect(c | thisModule.CreateAppPermission(c)),

usesSDK <-
s.children->select(c | c.name='uses-sdk'),
```

```
intents <-
if not thisModule.getIntentInMethodInvocation.oclIsUndefined()
  and thisModule.getClassInstanceCreationOfIntent.notEmpty()
  then
thisModule.getIntentInMethodInvocation->
  collect(c | thisModule.CreateIntentInMethodInvocation(c))
   ->union( thisModule.getClassInstanceCreationOfIntent->
     collect(c | thisModule.CreateIntent(c)))
else
if thisModule.getIntentInMethodInvocation.oclIsUndefined()
  and thisModule.getClassInstanceCreationOfIntent.notEmpty()
   then
thisModule.getClassInstanceCreationOfIntent ->
  collect(c | thisModule.CreateIntent(c))
else
if not thisModule.getIntentInMethodInvocation.oclIsUndefined()
  and not thisModule.getClassInstanceCreationOfIntent.notEmpty()
   then
thisModule.getIntentInMethodInvocation->
  collect(c | thisModule.CreateIntentInMethodInvocation(c))
else
OclUndefined
endif
endif
endif
)
}
```

Listing 2: ATL Rule to Create Application Element.

```
--for Create Application:
rule Element2Application{
from
s:XML!Element   (s.name='application')
to
t:ASAMM!Application(
Label <-
s.getAttribute('android:label').value,
```

```
Icon <-
if s.getAttribute('android:icon').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:icon').value
endif,

Components <-
s.children->
 select(c | thisModule.name_of_components.includes(c.name)),

dynamicRegisteredComponents <-
thisModule.All_Dynamic_broadcast_receiver,

Permission <-
if s.getAttribute('android:permission').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:permission').value
endif
)
}
```

Listing 3: ATL Rule to Create Component Element.

```
--for Create Component:
rule Element2Component {
from
s:XML!Element (thisModule.name_of_components.includes(s.name)
)
to
t:ASAMM!Component (
Name <-
if s.getAttribute('android:name').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:name').value
endif,
```

```
Label <-
if s.getAttribute('android:label').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:label').value
endif,

Icon <-
if s.getAttribute('android:icon').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:icon').value
endif,

Exported <-
if s.getAttribute('android:exported').oclIsUndefined() then
if not s.getIntentFilter.notEmpty() then
false
else
true
endif
else
thisModule.string2Boolean.get(
   s.getAttribute('android:exported').value)
endif,

Enabled <-
if s.getAttribute('android:enabled').oclIsUndefined() then
false
else
thisModule.string2Boolean.get(
   s.getAttribute('android:enabled').value)
endif,

Process <-
if s.getAttribute('android:process').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:process').value
```

```
 endif,

Data <-
if thisModule.getAllgetIntentMethod(
 s.getAttribute('android:name').value+'.java').notEmpty() then
true
 else
false
endif
)
}
```

Listing 4: ATL Rule to Create DynamicRegisteredComponent Element.

```
--for Create DynamicRegisteredComponent:
rule CreateDynamicRegisteredComponent {
from
s:java!MethodInvocation
  (s.method.toString().endsWith('registerReceiver')
  and not s.method.toString().endsWith('unregisterReceiver'))
to
t:ASAMM!DynamicRegisteredComponent
(
Exported <- true
)
}
```

Listing 5: ATL Rule to Create DynamicRegisteredBroadcastReceiver Element.

```
--for Create Receiver from java code:
rule Element2receiverfromjava extends
    CreateDynamicRegisteredComponent {
from
s:java!MethodInvocation
(s.method.toString().endsWith('registerReceiver')
and not s.method.toString().
   endsWith('unregisterReceiver'))
```

```
to
t:ASAMM!DynamicRegisteredBroadcastReceiver(
Name <-
if s.arguments -> select(c | c.oclIsTypeOf(
  java!FieldAccess)).notEmpty() then--for FielAccess
    s.arguments -> select(c | c.oclIsTypeOf(
      java!FieldAccess))
   ->first().field.variable.name
else
  if s.arguments->
  select(c | c.oclIsTypeOf(
    java!SingleVariableAccess)
   ).notEmpty() then--for SingleVariableAccess
    if  s.arguments->select(c | c.oclIsTypeOf(
                  java!SingleVariableAccess))
 ->select(c | not c.variable.oclIsTypeOf(
  java!VariableDeclarationFragment)).notEmpty() then
       s.arguments->select(c | c.oclIsTypeOf(
        java!SingleVariableAccess))
      ->first().variable.name
   else
      if s.arguments->select(c | c.oclIsTypeOf(
       java!SingleVariableAccess))
     ->select(c | c.variable.oclIsTypeOf(
      java!VariableDeclarationFragment))
   ->select(
   c | c.variable.refImmediateComposite().oclIsTypeOf(
     java!FieldDeclaration)).notEmpty() then
            s.arguments->select(c | c.oclIsTypeOf(
             java!SingleVariableAccess))
     ->select(c | c.variable.oclIsTypeOf(
      java!VariableDeclarationFragment))
       ->select(
      c | c.variable.refImmediateComposite().oclIsTypeOf(
       java!FieldDeclaration))->first().variable.name
      else
         if s.arguments->select(
          c | c.oclIsTypeOf(java!ThisExpression)
    ).notEmpty() then-- for ThisExpression
            s.originalCompilationUnit.name.
```

```
regexReplaceAll('.java','')
      else
        if not s.expression.oclIsUndefined() then
          if s.expression.oclIsTypeOf(java!FieldAccess) then
            if not s.expression.field.oclIsUndefined() then
              if s.expression.field.oclIsTypeOf(
                java!SingleVariableAccess) then
               s.expression.field.variable.name
              else
               OclUndefined
              endif
            else
            OclUndefined
            endif
          else
           if s.expression.oclIsTypeOf(
           java!SingleVariableAccess) then
            s.originalCompilationUnit.name.
regexReplaceAll('.java', '')
           else
            s.originalCompilationUnit.name.
  regexReplaceAll('.java', '')
           endif
          endif
        else
         OclUndefined
        endif
       endif
      endif
    endif
  else
    if not s.expression.oclIsUndefined() then
     if s.expression.oclIsTypeOf(java!FieldAccess) then
      if not s.expression.field.oclIsUndefined() then
       if s.expression.field.oclIsTypeOf(
         java!SingleVariableAccess) then
        s.expression.field.variable.name
       else
        OclUndefined
       endif
```

```
            else
             OclUndefined
            endif
           else
             if s.expression.oclIsTypeOf(
               java!SingleVariableAccess) then
              s.originalCompilationUnit.name.
    regexReplaceAll('.java', '')
      else
      s.originalCompilationUnit.name.
      regexReplaceAll('.java', '')
      endif
       endif
   else
     s.originalCompilationUnit.name.
     regexReplaceAll('.java', '')
   endif
 endif
endif,

filters <-
if s.arguments->select(c |
 c.oclIsTypeOf(java!SingleVariableAccess))
 ->select(c | c.variable.oclIsTypeOf(
  java!VariableDeclarationFragment))
  ->select(c | not c.variable.initializer.oclIsUndefined())
   ->select(c | c.variable.initializer.oclIsTypeOf(
    java!ClassInstanceCreation))
    ->flatten()->select(c |
     c.variable.initializer.type.type.name=
      'IntentFilter').notEmpty() then
 s.arguments->select(c |
  c.oclIsTypeOf(java!SingleVariableAccess))
  ->select(c | c.variable.oclIsTypeOf(
   java!VariableDeclarationFragment))
   -> select(c |
    not c.variable.initializer.oclIsUndefined())
    ->select(c | c.variable.initializer.oclIsTypeOf(
     java!ClassInstanceCreation))
  ->flatten()->select(c |
```

```
    c.variable.initializer.type.type.name=
    'IntentFilter')
    ->flatten()->collect(c |
     thisModule.CreateIntentFilterfromJava(c))
else
 if s.arguments -> select(c |
 c.oclIsTypeOf(java!ClassInstanceCreation))
 ->select(c | c.type.type.name=
 'IntentFilter').notEmpty() then
  s.arguments -> select(c | c.oclIsTypeOf(
  java!ClassInstanceCreation))
  ->select(c | c.type.type.name=
  'IntentFilter')
  ->collect(c | thisModule.
   CreateIntentFilterfromJavaForClassInstanceCreation(c))
  else
    if s.arguments -> select(c |
     c.oclIsTypeOf(java!FieldAccess))
  ->flatten()->select(c |
    c.field.variable.refImmediateComposite().oclIsTypeOf(
     java!FieldDeclaration))
    ->select(c |
     c.field.variable.
      refImmediateComposite().type.type.name=
       'IntentFilter').notEmpty() then
    s.arguments
    -> select(c | c.oclIsTypeOf(java!FieldAccess))
    ->flatten()->select(c |
     c.field.variable.refImmediateComposite().
     oclIsTypeOf(java!FieldDeclaration))
     ->select(c |
      c.field.variable.
     refImmediateComposite().type.type.name=
      'IntentFilter')
     ->collect(c |
     thisModule.
     CreateIntentFilterfromJavaForFieldAccess(c))
     else
  if s.arguments
  -> select(c | c.oclIsTypeOf(java!ClassInstanceCreation))
```

```
 ->select(c | c.type.type.name=
   'IntentFilter').notEmpty() then
s.arguments
-> select(c | c.oclIsTypeOf(
 java!ClassInstanceCreation))
 ->select(c | c.type.type.name=
'IntentFilter') ->collect(c |
thisModule.
CreateIntentFilterfromJavaForClassInstanceCreation(c))
else
if s.arguments
-> select(c | c.oclIsTypeOf(
 java!ClassInstanceCreation))
 ->select(c | c.type.type.name=
   'IntentFilter').notEmpty() then
 s.arguments
 -> select(c | c.oclIsTypeOf(java!ClassInstanceCreation))
  ->select(c | c.type.type.name=
     'IntentFilter') ->collect(c |
    thisModule.
      CreateIntentFilterfromJavaForClassInstanceCreation(c))
else
   if s.arguments
  -> select(c | c.oclIsTypeOf(java!ClassInstanceCreation))
   ->select(c | c.type.type.name=
       'IntentFilter').notEmpty() then
 s.arguments
 -> select(c | c.oclIsTypeOf(java!ClassInstanceCreation))
  ->select(c | c.type.type.name='IntentFilter')
  ->collect(c | thisModule.
    CreateIntentFilterfromJavaForClassInstanceCreation(c))
   else
   if s.arguments
  -> select(c | c.oclIsTypeOf(java!FieldAccess))
   ->flatten()->select(c |
     c.field.variable.refImmediateComposite().oclIsTypeOf(
      java!FieldDeclaration))
   ->select(c |
     c.field.variable.
     refImmediateComposite().type.type.name=
```

```
  'IntentFilter').notEmpty() then
s.arguments
-> select(c | c.oclIsTypeOf(java!FieldAccess))->flatten()
->select(c |
 c.field.variable.refImmediateComposite().oclIsTypeOf(
  java!FieldDeclaration))
 ->select(c |
  c.field.variable.
  refImmediateComposite().type.type.name=
  'IntentFilter')->collect(c | thisModule.
    CreateIntentFilterfromJavaForFieldAccess(c))
else
if s.arguments -> select(c |
 c.oclIsTypeOf(java!ClassInstanceCreation))
->select(c | c.type.type.name=
 'IntentFilter').notEmpty() then
  s.arguments -> select(c |
   c.oclIsTypeOf(java!ClassInstanceCreation))
  ->select(c | c.type.type.name='IntentFilter')
  ->collect(c | thisModule.
  CreateIntentFilterfromJavaForClassInstanceCreation(c))
else
 if s.arguments->select(c |
  c.oclIsTypeOf(java!UnresolvedItemAccess))
 ->flatten()->select(c |
 c.refImmediateComposite().oclIsTypeOf(
  java!MethodInvocation))
 ->select(c |
  c.refImmediateComposite().refImmediateComposite().
 refImmediateComposite().oclIsTypeOf(java!Block))
 ->flatten()
 ->select(c | c.getBlockInContainer.statements
 ->exists(d| d.expression.oclIsTypeOf(java!Assignment)
  or d.expression.oclIsTypeOf(java!MethodInvocation)))
   ->select(c | c.getBlockInContainer.statements->
 exists(d| d.expression.IsExpressionForAction
 or d.expression.IsMethodForAction)).notEmpty() then
  s.arguments
   ->select(c | c.oclIsTypeOf(java!UnresolvedItemAccess))
   ->first().getBlockInContainer.statements
```

```
        ->select(c | c.oclIsTypeOf(java!ExpressionStatement))
        ->flatten()->select(c |
          c.expression.IsExpressionForAction
         or c.expression.IsMethodForAction)->flatten()
      ->collect(c | thisModule.
       CreateIntentFilterfromJavaForExpressionStatement(c))
        else
        OclUndefined
        endif
      endif
       endif
      endif
       endif
    endif
     endif
     endif
 endif,

Source <- #JAVA,
Original <- s.originalCompilationUnit.name,
compPermisions <-
if not s.arguments.oclIsUndefined() then
  if s.arguments->select(c |
   c.oclIsTypeOf(java!StringLiteral)).notEmpty() then
     s.arguments->select(c |
     c.oclIsTypeOf(java!StringLiteral))
 ->collect(c | thisModule.
 CreateCompPermissionsForDynamicRegisteredReceiver(c))
  else
   OclUndefined
  endif
else
OclUndefined
endif
)
}
```

Listing 6: ATL Rule to Create AppPermission Element.

```
--for Create AppPermission:
lazy rule CreateAppPermission{
from
 s:XML!Element (s.name='permission')
to
   t:ASAMM!AppPermission(
Name <-
s.children -> select(c | c.name='android:name')
  -> first().value,

Permissionkind <-
if s.getAttribute(
    'android:protectionLevel').oclIsUndefined() then
OclUndefined
else
thisModule.string2ProtectionLevel.get(
 s.getAttribute('android:protectionLevel').value)
endif
)
}
```

Listing 7: ATL Rule to Create CompPermission Element.

```
--CompPermission for Activity, Service, and Receiver:
lazy rule CreateCompPermission {

 from
      s:XML!Attribute (s.name='android:permission')
 to
      t:ASAMM!CompPermission
  (
  Name <- s.value
  )
}
```

Listing 8: Another ATL Rule to Create CompPermission Element.

```
--Create CompPermissions for DynamicRegisteredReceiver
lazy rule CreateCompPermissionsForDynamicRegisteredReceiver{

 from
      s:java!StringLiteral
 to
      t:ASAMM!CompPermission
   (
   Name <- s.escapedValue.regexReplaceAll('\"', '')
   )
}
```

2.2 Extracting ICC Model

By collecting the Android Application Security Aspects model from each app, the comprehension of the Android system is facilitated, and all potential ICCs (at intra- and inter-app communication levels) are extracted and integrated into a single model called ICC. This model conforms to the proposed metamodel shown in Fig 2. Some of the ATL rules used for this M2M transformation are shown in the following.

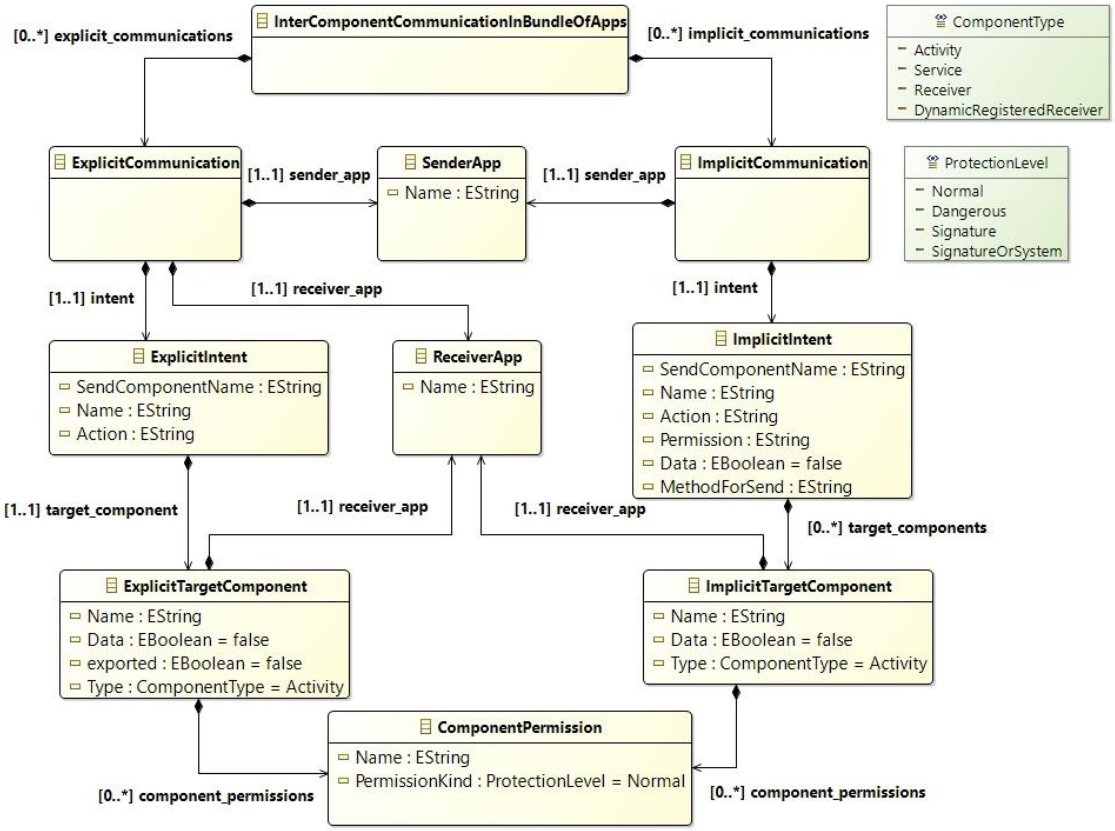

Figure 2: Inter-Component Communication (ICC) Metamodel.

Listing 9: ATL Rule to Create InterComponentCommunicationInBundleOfApps Element.

```
rule CreateInterComponentCommunicationInBundleOfApps {
from
 s:ASAMM!ApplicationPolicyFile (s.Package=
   ASAMM!ApplicationPolicyFile.allInstances().first().Package)
 to
 t:ICCMM!InterComponentCommunicationInBundleOfApps
(
explicit_communications <-
if not thisModule.getAllExplicitIntent.oclIsUndefined() then
thisModule.getAllExplicitIntent->
    collect(c | thisModule.CreateExplicitCommunication(c))
else
 OclUndefined
endif,

implicit_communications <-
if not thisModule.getAllImplicitIntent.oclIsUndefined() then
thisModule.getAllImplicitIntent->
  collect(c | thisModule.CreateImplicitCommunication(c))
else
OclUndefined
endif
 )
}
```

Listing 10: ATL Rule to Create ExplicitCommunication Element.

```
lazy rule CreateExplicitCommunication {
from
 s:ASAMM!Intent (s.IntentKind=#explicit)
to
 t:ICCMM!ExplicitCommunication
(
sender_app <-
 thisModule.CreateApplicationOne(s.refImmediateComposite().app),
```

```
receiver_app <-
if not s.TargetComponentName.oclIsUndefined() then
 if not thisModule.getApplicationTwo(s.TargetComponentName).
         oclIsUndefined() then
  thisModule.CreateApplicationTwo(thisModule.
     getApplicationTwo(s.TargetComponentName))
 else
   OclUndefined
 endif
else
  OclUndefined
endif,
intent <- thisModule.CreateIntentForExplicitCommunication(s)
)
}
```

Listing 11: ATL Rule to Create ExplicitIntent Element.

```
lazy rule CreateIntentForExplicitCommunication{
from
 s:ASAMM!Intent
to
 t:ICCMM!ExplicitIntent
(
Name <- s.Name,
SendComponentName <- s.SendComponentName,
Receiver <- s.TargetComponentName,
Action <- s.Action,

target_component <-
if not thisModule.getActivityReceiverCompForExplicitIntent(
   s.TargetComponentName).oclIsUndefined() then
  thisModule.createReceiverComponentForActivity(
   thisModule.getActivityReceiverCompForExplicitIntent(
     s.TargetComponentName))
else
 if not thisModule.getServiceReceiverCompForExplicitIntent(
 s.TargetComponentName).oclIsUndefined() then
```

```
 thisModule.createReceiverComponentForService(
 thisModule.getServiceReceiverCompForExplicitIntent(
 s.TargetComponentName))
else
 if not thisModule.getBroadcastReceiverCompForExplicitIntent(
 s.TargetComponentName).oclIsUndefined() then
 thisModule.createReceiverComponentForReceiver(
 thisModule.getBroadcastReceiverCompForExplicitIntent(
 s.TargetComponentName))
else
 if not thisModule.
   getDynamicBroadcastReceiverCompForExplicitIntent(
 s.TargetComponentName).oclIsUndefined() then
 thisModule.
    createReceiverComponentForDynamicRegisteredReceiver(
 thisModule.
   getDynamicBroadcastReceiverCompForExplicitIntent(
 s.TargetComponentName))
else
 OclUndefined
endif
endif
endif
endif
)
}
```

Listing 12: ATL Rule to Create SenderApp Element.

```
lazy rule CreateApplicationOne{

from
     s:ASAMM!Application
to
     t:ICCMM!SenderApp
 (
      Name <- s.refImmediateComposite().Package
     )
```

```
}
```

Listing 13: ATL Rule to Create ReceiverApp Element.

```
lazy rule CreateApplicationTwo{

from
    s:ASAMM!Application
to
    t:ICCMM!ReceiverApp
 (
    Name <- s.refImmediateComposite().Package
    )
}
```

Listing 14: ATL Rule to Create ImplicitCommunication Element.

```
lazy rule CreateImplicitCommunication {
from
 s:ASAMM!Intent (s.IntentKind=#implicit)
to
 t:ICCMM!ImplicitCommunication
 (
 intent <- thisModule.CreateIntentForImplicitCommunication(s),
 sender_app <- thisModule.CreateApplicationOne(
    s.refImmediateComposite().app)
 )
}
```

Listing 15: ATL Rule to Create ImplicitIntent Element.

```
lazy rule CreateIntentForImplicitCommunication{
 from
  s:ASAMM!Intent
```

```
 to
  t:ICCMM!ImplicitIntent
  (
Name <- s.Name,
SendComponentName <- s.SendComponentName,
Action <- s.Action,
Permission <-
 if not s.Permission.oclIsUndefined() then
  s.Permission
 else
  OclUndefined
 endif,

Data <-
 if s.ExtraData=true then
  true
 else
  if s.data.notEmpty() then
   true
  else
 false
  endif
endif,

MethodForSend <-
if not s.MethodForSend.oclIsUndefined() then
  s.MethodForSend
else
  OclUndefined
endif,

target_components <-
if not s.Action.oclIsUndefined() and s.categories.notEmpty()
   and not s.data.notEmpty() then
 if not s.TestActionAndCategoryForActivity.oclIsUndefined()
    then
  s.TestActionAndCategoryForActivity
 else
  if not s.TestActionAndCategoryForService.oclIsUndefined()
    then
```

```
    s.TestActionAndCategoryForService
  else
  if not s.TestActionAndCategoryForReceiver.oclIsUndefined()
    then
  s.TestActionAndCategoryForReceiver
 else
   if not s.TestActionAndCategoryForDynamicReceiver.
      oclIsUndefined()
    then
    s.TestActionAndCategoryForDynamicReceiver
   else
     OclUndefined
   endif
  endif
 endif
  endif
else
  if not s.Action.oclIsUndefined() and not s.categories.notEmpty()
     and s.data.notEmpty() then
   if not s.TestActionAndDataForActivity.oclIsUndefined()
     then
  s.TestActionAndDataForActivity
   else
  if not s.TestActionAndCategoryForService.
    oclIsUndefined()
    then
   s.TestActionAndCategoryForService
  else
   if not s.TestActionAndCategoryForReceiver.oclIsUndefined()
     then
    s.TestActionAndCategoryForReceiver
   else
     if not s.TestActionAndCategoryForDynamicReceiver.
    oclIsUndefined()
   then
      s.TestActionAndCategoryForDynamicReceiver
     else
      OclUndefined
     endif
   endif
```

```
endif
  endif
 else
  if s.Action.oclIsUndefined() and s.categories.notEmpty()
    and s.data.notEmpty() then
    OclUndefined
  else
 if s.Action.oclIsUndefined() and not s.categories.notEmpty()
    and s.data.notEmpty() then
     if not s.TestDataForActivity.oclIsUndefined()
   then
   s.TestDataForActivity
  else
   if not s.TestDataForService.oclIsUndefined()
    then
 s.TestDataForService
   else
 if not s.TestDataForReceiver.oclIsUndefined()
  then
  s.TestDataForReceiver
 else
   if not s.TestDataForDynamicReceiver.
    oclIsUndefined()
    then
    s.TestDataForDynamicReceiver
  else
   OclUndefined
  endif
   endif
 endif
  endif
 else
  if s.Action.oclIsUndefined() and s.categories.notEmpty()
    and not s.data.notEmpty() then
if not s.TestCategoryForActivity.oclIsUndefined()
 then
 s.TestCategoryForActivity
else
 if not s.TestCategoryForService.oclIsUndefined()
  then
```

```
  s.TestCategoryForService
 else
  if not s.TestCategoryForReceiver.oclIsUndefined()
   then
   s.TestCategoryForReceiver
 else
   if not s.TestCategoryForDynamicReceiver.
     oclIsUndefined()
    then
 s.TestCategoryForDynamicReceiver
   else
 OclUndefined
      endif
endif
  endif
 endif
  else
if not s.Action.oclIsUndefined() and not s.categories.notEmpty()
 and not s.data.notEmpty() then
  if not s.TestActionForActivity.oclIsUndefined()
   then
    s.TestActionForActivity
  else
 if not s.TestActionForService.oclIsUndefined()
  then
  s.TestActionForService
 else
    if not s.TestActionForReceiver.oclIsUndefined() and
    not s.TestActionForDynamicReceiver.oclIsUndefined()
     then
    s.TestActionForReceiver
      ->union(s.TestActionForDynamicReceiver)
 ->collect(c |
   thisModule.createTargetComponentReceiverAndDynamicReciver(c))
    else
      if not s.TestActionForReceiver.oclIsUndefined() then
         s.TestActionForReceiver
    ->collect(c |
     thisModule.createTargetComponentReceiver(c))
      else
```

```
            if not s.TestActionForDynamicReceiver.
        oclIsUndefined() then
            s.TestActionForDynamicReceiver
        ->collect(c |
         thisModule.createTargetComponentDynamicRegisteredReceiver(c))
            else
              OclUndefined
            endif
     endif
       endif
    endif
     endif
  else
  OclUndefined
  endif
     endif
    endif
  endif
  endif
endif
)
}
```

Listing 16: ATL Rule to Create ImplicitTargetComponent (for Activity) Element.

```
lazy rule createTargetComponentForActivity{
from
s:ASAMM!Activity
to
t:ICCMM!ImplicitTargetComponent
(
Name <- s.Name,
receiver_app <- thisModule.CreateApplicationTwo(
 s.refImmediateComposite()),

component_permissions <-
if not s.compPermisions.oclIsUndefined() then
      s.compPermissions
```

```
  ->collect(c | thisModule.CreateCompPermissions(c))
else
 OclUndefined
endif,

Data <-
if s.Data then
true
else
 false
endif,

Type <- #Activity
)
}
```

Listing 17: ATL Rule to Create ExplicitTargetComponent (for Activity) Element.

```
lazy rule createReceiverComponentForActivity{
from
s:ASAMM!Activity
to
t:ICCMM!ExplicitTargetComponent
(
Name <- s.Name,
receiver_app <- thisModule.
      CreateApplicationTwo(s.refImmediateComposite()),

component_permissions <-
if not s.compPermissions.oclIsUndefined() then
   s.compPermissions
    ->collect(c | thisModule.CreateCompPermissions(c))
else
 OclUndefined
endif,

Data <-
if s.Data then
```

```
true
else
 false
endif,

exported <- s.Exported,

Type <- #Activity
)
}
```

Listing 18: ATL Rule to Create ComponentPermission (for Activity) Element.

```
lazy rule CreateCompPermissions {

 from
      s:ASAMM!CompPermission
 to
      t:ICCMM!ComponentPermission
   (
   Name <- s.Name,
   PermisionKind <- s.Permisionkind
   )
}
```

3 The ATL Rules and OCL Queries Used in the Analysis Phase

By gathering all potential communications between apps and displaying them in a single model called ICC that conforms to the proposed metamodel shown in Fig. 2, it is possible to automatically and effectively perform security analysis at the intra- and inter-app levels. This phase focuses on two prominent inter-app vulnerabilities called Intent Spoofing and Unauthorized Intent Receipt. Some of the ATL transformation rules and OCL queries used for this phase are shown in the following.

Listing 19: ATL Rule to Check Intent Spoofing Vulnerability for Implicit Communications.

```
helper context ICC!ImplicitTargetComponent def :
 haveIntentSpoofing :Boolean=
if not self.component_permissions.notEmpty() then
  true
else
  if self.component_permissions->exists(c |
     not thisModule.NormalPermission.includes(c)) then
   false
  else
   true
  endif
endif;

helper context ICC!ImplicitTargetComponent def:
 getIntentSpoofingCommunications :Sequence(String) =
  let ImplicitTargetComponentonentsOfCommunication :
   Sequence(ICC!ImplicitTargetComponent) =
 self.refImmediateComposite().target_components in
    ImplicitTargetComponentonentsOfCommunication
    ->iterate(n ; output : Sequence(String) = Sequence{} |
if not self.Name.endsWith('MainActivity') then
 if self.haveIntentSpoofing then
  if self.Type= #DynamicRegisteredReceiver then
  output->
   including(': This Bundle
    has Broadcast Injection in the implicit
```

```
      communication with this specification:'+
  ' '+ '/SendComponentName Component:'+ ' '
+ self.refImmediateComposite().SendComponentName +
' /'+ ' ' + '/SendComponentName App:'+
' ' + self.refImmediateComposite().
      refImmediateComposite().sender_app.Name +
' /' + '/Target Component:'+ ' ' + self.Name +
' /' + '/Receiver App:'+ ' ' + self.receiver_app.Name
+ ' /')
 else
if self.Data=true then
  if self.Type=#Activity then
  output->
  including(': This Bundle
  has Activity Launch(with data)
  in the implicit communication
  with this specification:' +
 ' '+ '/SendComponentName Component:'+
 ' ' + self.refImmediateComposite().SendComponentName +
 ' /'+ ' ' + '/SendComponentName App:'+
 ' ' + self.refImmediateComposite().
    refImmediateComposite().sender_app.Name +
 ' /' + '/Target Component:'+ ' ' +
 self.Name + ' /' + '/Receiver App:'+ ' ' +
  self.receiver_app.Name + ' /')
   else
 if self.Type=#Service then
 output->
 including(': This Bundle
  has Service Launch(with data) in
  the implicit communication
  with this specification:' +
  ' '+ '/SendComponentName Component:'+ ' ' +
    self.refImmediateComposite().SendComponentName +
  ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
  self.refImmediateComposite().
    refImmediateComposite().sender_app.Name +
  ' /' + '/Target Component:'+ ' ' +
  self.Name + ' /' +
  '/Receiver App:'+ ' ' +
```

```
  self.receiver_app.Name + ' /')
  else
   if self.Type=#Receiver then
   output->including(': This Bundle has
     Broadcast Injection(with data)
    in the implicit communication
    with this specification:' +
   ' '+ '/SendComponentName Component:'+ ' ' +
    self.refImmediateComposite().SendComponentName +
   ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
   self.refImmediateComposite().
    refImmediateComposite().sender_app.Name +
   ' /' + '/Target Component:'+ ' ' +
   self.Name + ' /' +
   '/Receiver App:'+ ' ' +
   self.receiver_app.Name + ' /')
   else
   output->reject(c | c.Name=self.Name)
   endif
  endif
 endif
else
 if self.Type=#Activity then
  output->including(': This Bundle has
   Activity Launch(without data) in
   the implicit communication
  with this specification:'
  + ' '+ '/SendComponentName Component:'+ ' ' +
   self.refImmediateComposite().SendComponentName +
  ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
  self.refImmediateComposite().
   refImmediateComposite().sender_app.Name +
  ' /' + '/Target Component:'+ ' ' +
  self.Name + ' /' +
  '/Receiver App:'+ ' ' +
  self.receiver_app.Name + ' /')
 else
  if self.Type=#Service then
  output->including(': This Bundle has
  Service Launch(without data) in
```

```
     the implicit communication
    with this specification:' + ' '+
    '/SendComponentName Component:'+ ' ' +
     self.refImmediateComposite().SendComponentName +
    ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
    self.refImmediateComposite().
     refImmediateComposite().sender_app.Name +
    ' /' + '/Target Component:'+ ' ' +
    self.Name + ' /' +
    '/Receiver App:'+ ' ' +
    self.receiver_app.Name + ' /')
     else
    if self.Type=#Receiver then
    output->including(': This Bundle
     has Broadcast Injection(without data) in
     the implicit communication
     with this specification:' + ' '+
     '/SendComponentName Component:'+ ' ' +
     self.refImmediateComposite().SendComponentName +
     ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
     self.refImmediateComposite().
      refImmediateComposite().sender_app.Name +
     ' /' + '/Target Component:'+ ' ' +
     self.Name + ' /' + '/Receiver App:'+
     ' ' + self.receiver_app.Name + ' /')
    else
    output->reject(c | c.Name=self.Name)
    endif
     endif
  endif
 endif
  endif
 else
  output->reject(c | c.Name=self.Name)
 endif
else
 output->reject(c | c.Name=self.Name)
endif
)->asSet()->asSequence();
```

```
rule Communications2IntentSpoofingCommunications
{
from
 s:ICC!ImplicitTargetComponent
 (not s.receiver_app.Name.toString().endsWith(
 s.refImmediateComposite().
  refImmediateComposite().sender_app.Name.toString()))
to
t:ANO!IntentSpoofing
(
description <-  s.getIntentSpoofingCommunications
 ->iterate(up; output: String = '' | output->concat(up)+'\n')
)
do {
  t.trace<-Sequence{s};
   }
}
```

Listing 20: ATL Rule to Check Intent Spoofing Vulnerability for Explicit Communications.

```
helper context ICC!ExplicitTargetComponent def :
 haveIntentSpoofing :Boolean=
if not self.component_permissions.notEmpty() then
  true
  else
    false
endif;

helper context ICC!ExplicitCommunication def:
 getIntentSpoofingForExplicitCommunications :
  Sequence(String) =
 let ExplicitCommunicationsOfBundle :
  Sequence(ICC!ExplicitCommunication) =
  self.refImmediateComposite().
  explicit_communications in
 ExplicitCommunicationsOfBundle
 ->iterate(n ; output :
  Sequence(String) = Sequence{} |
```

```
if not self.intent.Receiver.
 endsWith('MainActivity')then
 if not self.receiver_app.oclIsUndefined() then
   if not self.receiver_app.Name.toString().endsWith(
    self.sender_app.Name.toString()) then
    if self.intent.target_component.
  exported=true then
      if self.intent.target_component.
    haveIntentSpoofing then
       if self.intent.target_component.Type=
        #DynamicRegisteredReceiver then
      output->including(': This Bundle has
       Broadcast Injection
      in the implicit communication
  with this specification:' +
  ' '+ '/SendComponentName Component:'+ ' ' +
  self.intent.SendComponentName + ' /'+
  ' ' + '/SendComponentName App:'+ ' ' +
  self.sender_app.Name + ' /' +
  '/Target Component:'+ ' ' +
  self.intent.Receiver + ' /' +
  '/Receiver App:'+ ' ' +
  self.receiver_app.Name + ' /')
    else
  if self.intent.target_component.Data=true then
   if self.intent.target_component.Type=
    #Activity then
   output->including(': This Bundle has
   Activity Launch(with data)
   in the implicit communication
   with this specification:' +
   ' '+  '/SendComponentName Component:'+ ' ' +
   self.intent.SendComponentName + ' /'+
   ' ' + '/SendComponentName App:'+ ' ' +
   self.sender_app.Name + ' /' +
   '/Target Component:'+
   ' ' + self.intent.Receiver + ' /' +
   '/Receiver App:'+ ' ' +
   self.receiver_app.Name + ' /')
   else
```

```
  if self.intent.target_component.Type=
   #Service then
   output->including(': This Bundle has
   Service Launch(with data)
   in the implicit communication
   with this specification:' +
   ' '+ '/SendComponentName Component:'+ ' ' +
   self.intent.SendComponentName + ' /'+
   ' ' + '/SendComponentName App:'+ ' ' +
   self.sender_app.Name + ' /' +
   '/Target Component:'+ ' ' +
   self.intent.Receiver + ' /' +
   '/Receiver App:'+ ' ' +
   self.receiver_app.Name + ' /')
  else
   if self.intent.target_component.Type=
    #Receiver then
 output->including(': This Bundle has
 Broadcast Injection(with data)
 in the implicit communication
 with this specification:' +
 ' '+ '/SendComponentName Component:'+ ' ' +
 self.intent.SendComponentName + ' /'+
 ' ' + '/SendComponentName App:'+ ' ' +
 self.sender_app.Name + ' /' +
 '/Target Component:'+
 ' ' + self.intent.Receiver + ' /' +
 '/Receiver App:'+
 ' ' + self.receiver_app.Name + ' /')
   else
    output->reject(c |
     c.Name=self.intent.target_component.Name)
   endif
  endif
endif
  else
if self.intent.target_component.Type=
 #Activity then
 output->including(': This Bundle has
 Activity Launch(without data)
```

```
 in the implicit communication
 with this specification:' + ' '+
 '/SendComponentName Component:'+ ' ' +
 self.intent.SendComponentName + ' /'+ ' ' +
 '/SendComponentName App:'+ ' ' +
 self.sender_app.Name + ' /' +
 '/Target Component:'+ ' ' +
 self.intent.Receiver
+ ' /' + '/Receiver App:'+ ' ' +
 self.receiver_app.Name + ' /')
else
 if self.intent.target_component.Type=
  #Service then
 output->including(': This Bundle has
 Service Launch(without data)
 in the implicit communication
 with this specification:' +
 ' '+ '/SendComponentName Component:'+
 ' ' + self.intent.SendComponentName + ' /'+ ' ' +
 '/SendComponentName App:'+ ' ' +
 self.sender_app.Name +
 ' /' + '/Target Component:'+ ' ' +
 self.intent.Receiver + ' /' +
 '/Receiver App:'+ ' ' + self.receiver_app.Name +' /')
 else
  if self.intent.target_component.Type=
   #Receiver then
  output->including(': This Bundle has
  Broadcast Injection(without data)
  in the implicit communication
  with this specification:' + ' '+
  '/SendComponentName Component:'+ ' ' +
  self.intent.SendComponentName +
  ' /'+ ' ' + '/SendComponentName App:'+
  ' ' + self.sender_app.Name + ' /' +
  '/Target Component:'+ ' ' +
  self.intent.Receiver +
  ' /' + '/Receiver App:'+ ' ' +
  self.receiver_app.Name + ' /')
  else
```

```
    output->reject(c | c.Name=
      self.intent.target_component.Name)
   endif
 endif
  endif
endif
  endif
 else
output->reject(c |
 c.Name=self.intent.target_component.Name)
 endif
   else
    if self.intent.target_component.Type=
     #DynamicRegisteredReceiver then
    output->including(': This Bundle has
     Broadcast Injection in the implicit communication
 with this specification:' +
 ' '+ '/SendComponentName Component:'+ ' ' +
 self.intent.SendComponentName +
 ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
 self.sender_app.Name + ' /' +
 '/Target Component:'+ ' ' +
 self.intent.Receiver + ' /' +
 '/Receiver App:'+ ' ' +
 self.receiver_app.Name + ' /')
 else
  if self.intent.target_component.Data=true then
 if self.intent.target_component.Type=
 #Activity then
 output->including(': This Bundle has
  Activity Launch(with data)
 in the implicit communication
 with this specification:' + ' '+
 '/SendComponentName Component:'+ ' ' +
   self.intent.SendComponentName + ' /'+
 ' ' + '/SendComponentName App:'+ ' ' +
 self.sender_app.Name + ' /' +
 '/Target Component:'+
 ' ' + self.intent.Receiver + ' /' +
 '/Receiver App:'+ ' ' +
```

```
self.receiver_app.Name + ' /')
else
 if self.intent.target_component.Type=#Service then
  output->including(': This Bundle has
  Service Launch(with data)
  in the implicit communication with
  this specification:' + ' '+
  '/SendComponentName Component:'+ ' ' +
  self.intent.SendComponentName + ' /'+
  ' ' + '/SendComponentName App:'+ ' ' +
  self.sender_app.Name + ' /' +
  '/Target Component:'+
  ' ' + self.intent.Receiver + ' /' +
  '/Receiver App:'+ ' ' +
  self.receiver_app.Name + ' /')
 else
  if self.intent.target_component.Type=
   #Receiver then
 output->including(': This Bundle has
 Broadcast Injection(with data)
 in the implicit communication
 with this specification:' + ' '+
 '/SendComponentName Component:'+ ' ' +
 self.intent.SendComponentName +
 ' /'+ ' ' +
 '/SendComponentName App:'+ ' ' + self.sender_app.Name +
 ' /' + '/Target Component:'+
 ' ' + self.intent.Receiver + ' /' +
 '/Receiver App:'+ ' ' +
 self.receiver_app.Name + ' /')
  else
   output->reject(c |
    c.Name=self.intent.target_component.Name)
  endif
endif
 endif
  else
    if self.intent.target_component.Type=
     #Activity then
  output->including(': This Bundle has
```

```
  Activity Launch(without data)
  in the implicit communication
  with this specification:' +
  ' '+ '/SendComponentName Component:'+ ' ' +
  self.intent.SendComponentName + ' /'+ ' ' +
  '/SendComponentName App:'+ ' ' + self.sender_app.Name +
  ' /' + '/Target Component:'+
  ' ' + self.intent.Receiver + ' /' +
  '/Receiver App:'+ ' ' +
  self.receiver_app.Name + ' /')
 else
  if self.intent.target_component.Type=
   #Service then
output->including(': This Bundle has
Service Launch(without data)
in the implicit communication
with this specification:' + ' '+
'/SendComponentName Component:'+ ' ' +
self.intent.SendComponentName + ' /'+ ' ' +
'/SendComponentName App:'+ ' ' + self.sender_app.Name +
' /' +
'/Target Component:'+ ' ' +self.intent.Receiver +
' /' +
'/Receiver App:'+ ' ' + self.receiver_app.Name +
' /')
  else
   if self.intent.target_component.Type=
    #Receiver then
 output->including(': This Bundle has
 Broadcast Injection(without data)
 in the implicit communication
 with this specification:' +
 ' '+ '/SendComponentName Component:'+ ' ' +
 self.intent.SendComponentName + ' /'+ ' ' +
 '/SendComponentName App:'+ ' ' + self.sender_app.Name +
 ' /' + '/Target Component:'+
 ' ' + self.intent.Receiver + ' /' +
 '/Receiver App:'+ ' ' +
 self.receiver_app.Name + ' /')
else
```

```
    output->reject(c |
     c.Name=
      self.intent.target_component.Name)
   endif
     endif
    endif
      endif
     endif
    endif
   else
   output->reject(c | c.Name=
    self.intent.target_component.Name)
   endif
 else
 output->reject(c | c.Name=
  self.intent.target_component.Name)
 endif
else
output->reject(c | c.Name=
 self.intent.target_component.Name)
endif
)->asSet()->asSequence();

rule ExplicitCommunications2IntentSpoofingCommunications
 {
from
 s:ICC!ExplicitCommunication
 (not s.intent.target_component.oclIsUndefined() )
to
 t:ANO!IntentSpoofing
(
 description <-  s.getIntentSpoofingForExplicitCommunications
  ->iterate(up; output: String = '' | output->concat(up)+'\n')
)
 do {
 t.trace<-Sequence{s};
    }
}
```

Listing 21: ATL Rule to Check Unauthorized Intent Receipt Vulnerability.

```
helper context OclAny def :
 getContainerImplicitCommunication :
  ICC!ImplicitCommunication =
if self.oclIsTypeOf(ICC!ImplicitCommunication) then
  self
 else
  self.refImmediateComposite().
  getContainerImplicitCommunication
endif;

helper context ICC!ImplicitTargetComponent def:
 getUnauthorizedIntentReceiptCommunications :
  Sequence(String) =
let ImplicitTargetComponentonentsOfCommunication :
 Sequence(ICC!ImplicitTargetComponent) =
  self.refImmediateComposite().
  target_components in
        ImplicitTargetComponentonentsOfCommunication
        ->iterate(n ; output :
         Sequence(String) = Sequence{} |

if self.getContainerImplicitCommunication.
 intent.Permission.oclIsUndefined() then
 if self.getContainerImplicitCommunication.
   intent.Data=true then
  if self.Type=#Activity then
 output->including(': This Bundle has
 Activity Hijacking (with data)
 in the implicit communication
 with this specification:' + ' '+
 '/SendComponentName Component:'+ ' ' +
 self.refImmediateComposite().SendComponentName + ' /'+
 ' ' + '/SendComponentName App:'+ ' ' +
 self.refImmediateComposite().
   refImmediateComposite().sender_app.Name +
 ' /' + '/Target Component:'+ ' ' +
 self.Name + ' /' + '/Receiver App:'+ ' ' +
 self.receiver_app.Name + ' /')
  else
```

```
  if self.Type=#Service then
  output->including(': This Bundle has
   Service Hijacking (with data)
  in the implicit communication
with this specification:' + ' '+
'/SendComponentName Component:'+ ' ' +
self.refImmediateComposite().SendComponentName + ' /'+
' ' + '/SendComponentName App:'+ ' ' +
self.refImmediateComposite().
  refImmediateComposite().sender_app.Name +
' /' + '/Target Component:'+ ' ' +
self.Name + ' /' + '/Receiver App:'+
' ' + self.receiver_app.Name + ' /')
  else
if self.Type=#Receiver then
 output->including(': This Bundle has
  Broadcast Theft (with data)
    in the implicit communication
 with this specification:' + ' '+
 '/SendComponentName Component:'+
 ' ' + self.refImmediateComposite().SendComponentName +
 ' /'+ ' ' +
 '/SendComponentName App:'+ ' ' +
 self.refImmediateComposite().
   refImmediateComposite().sender_app.Name +
 ' /' + '/Target Component:'+ ' ' +
 self.Name + ' /' + '/Receiver App:'+
 ' ' + self.receiver_app.Name + ' /')
else
 if self.Type=#DynamicRegisteredReceiver then
  output->including(': This Bundle has
   Broadcast Theft (with data)
  in the implicit communication
 with this specification:' + ' '+
 '/SendComponentName Component:'+
 ' ' + self.refImmediateComposite().SendComponentName +
 ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
 self.refImmediateComposite().
   refImmediateComposite().sender_app.Name +
 ' /' + '/Target Component:'+ ' ' +
```

```
 self.Name + ' /' + '/Receiver App:'+ ' ' +
 self.receiver_app.Name + ' /')
 else
  output->reject(c | c.Name=self.Name)
 endif
   endif
  endif
 endif
else--not Data
 if self.Type=#Activity then
 output->including(': This Bundle has
  Activity Hijacking (without data)
   in the implicit communication
with this specification:' + ' '+
'/SendComponentName Component:'+ ' ' +
self.refImmediateComposite().SendComponentName +
' /'+ ' ' + '/SendComponentName App:'+
' ' + self.refImmediateComposite().
  refImmediateComposite().sender_app.Name +
' /' + '/Target Component:'+ ' ' +
self.Name + ' /' + '/Receiver App:'+
' ' + self.receiver_app.Name + ' /')
 else
  if self.Type=#Service then
   output->including(': This Bundle has
    Service Hijacking (without data)
       in the implicit communication
 with this specification:' + ' '+
 '/SendComponentName Component:'+ ' ' +
 self.refImmediateComposite().SendComponentName +
 ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
 self.refImmediateComposite().
   refImmediateComposite().sender_app.Name +
 ' /' + '/Target Component:'+ ' ' +
 self.Name + ' /' + '/Receiver App:'+
 ' ' + self.receiver_app.Name + ' /')
  else
if self.Type=#Receiver then
  output->including(': This Bundle has
   Broadcast Theft (without data)
```

```
      in the implicit communication
  with this specification:' + ' '+
  '/SendComponentName Component:'+ ' ' +
  self.refImmediateComposite().SendComponentName +
  ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
  self.refImmediateComposite().
   refImmediateComposite().sender_app.Name +
  ' /' + '/Target Component:'+ ' ' +
  self.Name + ' /' + '/Receiver App:'+
  ' ' + self.receiver_app.Name + ' /')
 else
  if self.Type=#DynamicRegisteredReceiver then
  output->including(': This Bundle has
   Broadcast Theft (without data)
     in the implicit communication
  with this specification:' + ' '+
  '/SendComponentName Component:'+ ' ' +
  self.refImmediateComposite().SendComponentName +
  ' /'+ ' ' + '/SendComponentName App:'+ ' ' +
  self.refImmediateComposite().
   refImmediateComposite().sender_app.Name +
  ' /' + '/Target Component:'+ ' ' +
  self.Name + ' /' + '/Receiver App:'+
  ' ' + self.receiver_app.Name + ' /')
  else
  output->reject(c | c.Name=self.Name)
  endif
 endif
   endif
  endif
 endif
else
output->reject(c | c.Name=self.Name)
endif
)->asSet()->asSequence();


rule ImplicitCommunications2UnauthorizedIntentReceiptCommunications
{
from
```

```
  s:ICC!ImplicitTargetComponent
   (not s.receiver_app.Name.toString().endsWith(
    s.refImmediateComposite().
    refImmediateComposite().sender_app.Name.toString()))
to
  t:ANO!UnauthorizedIntentReceipt
 (
description <-  s.getUnauthorizedIntentReceiptCommunications
->iterate(up; output: String = '' | output->concat(up)+'\n')
 )
 do {
 t.trace<-Sequence{s};
    }
 }
```

References

[1] A. Nirumand, B. Zamani, and B. Tork Ladani, "VAnDroid: A framework for vulnerability analysis of Android applications using a model-driven reverse engineering technique," Software: Practice and Experience, vol. 49, no.1, pp. 70–99, 2019. 2, 3, 4