ATL rules and OCL queries implemented in VAnDroid

Atefeh Nirumand
Bahman Zamani
Behrouz Tork Ladani
ATL rules and OCL queries implemented in VAnDroid

Atefeh Nirumand, Bahman Zamani, and Behrouz Tork Ladani
Department of Software Engineering
University of Isfahan
Isfahan, Iran.
{atefehnirumand, zamani, ladani}@eng.ui.ac.ir

Abstract: This report provides the implementation details of the VAnDroid. VAnDroid is a framework based on Model Driven Reverse Engineering (MDRE) to analyze Android apps. The approach conducted in this framework has three phases. The feasibility and pertinence of this approach is demonstrated by developing an Eclipse-based tool called VAnDroid which implements the three phases of this approach. This tool has been developed by Nirumand et al. VAnDroid automatically identifies the Intent Spoofing and Unauthorized Intent Receipt as two attacks related to the Android application communication model.
Abstract: This report provides the implementation details of the VAnDroid. VAnDroid is a framework based on Model Driven Reverse Engineering (MDRE) to analyze Android apps. The approach conducted in this framework has three phases. The feasibility and pertinence of this approach is demonstrated by developing an Eclipse-based tool called VAnDroid which implements the three phases of this approach. This tool has been developed by Nirumand et al. VAnDroid automatically identifies the Intent Spoofing and Unauthorized Intent Receipt as two attacks related to the Android application communication model.
1 Introduction

VAnDroid is a framework based on Model Driven Reverse Engineering (MDRE). In fact, the MDRE process is used for this framework to analyze Android apps and detect vulnerabilities in the Inter-Component Communication (ICC). The approach conducted in this framework has three phases. The security information is first extracted from the Android application. This is then transformed into a single model called Android Applications Security Aspects using model-to-model transformation of the ATL. Based on this model, some operations, such as queries and model manipulations, can be applied later to analyze and manage the security configurations. In order to detect vulnerabilities, the ATL and OCL rules have been used. In this report, the details of these ATL rules and OCL queries are described.

2 The ATL rules used in the Transformations and Integration phase

In order to extract the needed information and integrate them in the single model, the proposed metamodel is used. This metamodel is shown in Figure 1. In the following, the ATL transformation rules used for the transformations and integration phase are shown.
Figure 1: Android Application Security Aspects Metamodel.
rule Element2SDK{
  from
  s:XML!Element (s.name='uses-sdk')
  to
t:ASEC!SDK
  (targetSdkVersion <-
   if s.getAttribute('android:targetSdkVersion').oclIsUndefined() then
     OclUndefined
   else
     s.getAttribute('android:targetSdkVersion').value
   endif,
   minSdkVersion <- if
     s.getAttribute('android:minSdkVersion').oclIsUndefined() then
     OclUndefined
   else
     s.getAttribute('android:minSdkVersion').value
   endif,
   maxSdkVersion <- if
     s.getAttribute('android:maxSdkVersion').oclIsUndefined() then
     OclUndefined
   else
     s.getAttribute('android:maxSdkVersion').value
   endif)
}

Listing 1: ATL rule to create SDK element.

rule Element2UsesPermission{
  from
  s:XML!Element (s.name='uses-permission')

Listing 2: ATL rule to create UsesPermission element.

```plaintext
rule Element2NewPermission {
from
s:XML!Element (s.name='permission')
to
t:ASEC!NewPermission

(name <- s.getAttribute('android:name').value,
Permisionkind <-
thisModule.stringOfPermission2ProtectionLevel.get(
    s.getAttribute('android:name').value)
)
}
```

Listing 2: ATL rule to create UsesPermission element.
if s.getAttribute('android:protectionLevel').oclIsUndefined() then
    OclUndefined
else
    thisModule.string2ProtectionLevel.get(
        s.getAttribute('android:protectionLevel').value
    )
endif,

PermissionGroup <-
if s.getAttribute('android:permissionGroup').oclIsUndefined() then
    OclUndefined
else
    s.getAttribute('android:permissionGroup').value
endif
}

Listing 3: ATL rule to create NewPermission element.

rule Element2PermissionGroup{
from
s:XML!Element (s.name='permission-group')
t:ASEC!PermissionGroup
( name <- s.getAttribute('android:name').value ,
   Description <-
   if s.getAttribute('android:description').oclIsUndefined() then
     OclUndefined
   else
     s.getAttribute('android:description').value
   endif ,
   icon <- if s.getAttribute('android:icon').oclIsUndefined() then

 rule Element2Application{
  from
  s:XML!Element (s.name='application')
  to
  t:ASEC!Application
  (Label <- s.getAttribute('android:label').value, 
  icon <- if s.getAttribute('android:icon').oclIsUndefined() 
  then 
  OclUndefined 
  else 
  s.getAttribute('android:icon').value
  endif, 
  permission <- 
  if s.getAttribute('android:permission').oclIsUndefined() 
  then 
  OclUndefined 
  else 
  s.getAttribute('android:permission').value
  endif,
  
Listing 4: ATL rule to create PermissionGroup element.
Components ← s.children →

    select ( c | thisModule.name_of_components.includes(c.name) )

Listing 5: ATL rule to create Application element.

rule Element2Component {
    from s:XML!Element (thisModule.name_of_components.includes(s.name))
    to t:ASEC!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,
permission <-
if s.getAttribute('android:permission').oclIsUndefined() then
OclUndefined
else
s.getAttribute('android:permission').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,

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,

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

Listing 6: ATL rule to create Component element.

```atl
rule Element2intentFilter { 
  from s:XML!Element (s.name='intent-filter') 
  to t:ASEC!intentFilter 
  ( 
    actions ← s.children ->select (c | c.name='action'),
    category ← s.children ->select (c | c.name='category'),
    data ← s.children ->select (c | c.name='data')
  )
}
```

Listing 7: ATL rule to create intentFilter element.
3 The ATL rules and OCL queries used in the Analysis phase

All the security information of the application is gathered and represented in the form of an integrated model corresponding to our Android Applications Security Aspects metamodel. The security information obtained in the form of the model is received as input to be evaluated. In order to detect vulnerabilities, the ATL and OCL rules have been used. In the following, some of the ATL transformation rules and OCL queries used for the analysis phase are shown.

```plaintext
rule Component2IntentSpoofing
{
  from
  s:ASEC!Component(s.isPublic and (s.permission.oclIsUndefined() 
  or not thisModule.StrongPermission.includes(s.permission)))
  to
  t:ANO!IntentSpoofingComponent
  
  description <- s.getIntentSpoofing
  =>iterate(up; output: String = '"' | output->concat(up)+'\n')
  do {
    t.trace<-Sequence{s};
  }
}
```

Listing 8: ATL rule to check Intent Spoofing vulnerability.

```plaintext
helper context ASEC!Component
  def:getIntentSpoofing :Sequence(String) =
```
let ComponentsOfApp : Sequence(ASEC!Component) =
self.refImmediateComposite().Components in
ComponentsOfApp =>
iterate(n; output : Sequence(String) = Sequence{} |
if self.haveIntentSpoofing then
  if self.Data=true then
    if self.oclIsTypeOf(ASEC!Activity) then
      output->including(
        ': The Application have
        Activity Launch(with data) in component with name: '+'
        self.name+
        '+''+'''... /the reasons of this vulnerability are :
        this component is public
        and component does not have Strong Permission ')
    else
      if self.oclIsTypeOf(ASEC!Service) then
        output->including(
          ': The Application
          have Service Launch(with data) in component with name: '+'
          self.name+
          '+''+'''... /the reasons of this vulnerability are :
          this component is public
          and component does not have Strong Permission ')
      else
        if self.oclIsTypeOf(ASEC!BroadcastReceiver) then
          output->including(
            ': The Application
            have Broadcast Injection(with data) in component with name:
            '+'
            self.name+
            '+''+'''... /the reasons of this vulnerability are :
            this component is public
            and component does not have Strong Permission ')
        else
          output->reject(c | c.name=self.name)
    endif
  endif
endif
else
  if self.oclIsTypeOf(ASEC!Activity) then

Listing 9: Helper to check Intent Spoofing vulnerability.
rule Intent2UnauthorizedIntentReceipt
{
  from s:ASEC!Intent (not s.Action.oclIsUndefined() and s.IntentKind = #implicit )
  to t:ANO!UnauthorizedIntentReceipt
  ( description <-- s.getUnauthorizedIntentReceipt->
    iterate(up; output: String = '' | output->concat(up)+'n ')
  )
do {
  t.trace <-- Sequence{ s };
}
Listing 10: ATL rule to check Unauthorized Intent Receipt vulnerability.

helper context ASEC!Intent
def: getUnauthorizedIntentReceipt : Sequence(String) =
let IntentsOfApp : Sequence(ASEC!Intent) =
  self.refImmediateComposite().Intents in
IntentsOfApp->iterate(n ; output : Sequence(String) =
  Sequence{} | 
  if self.IntentKind=#explicit then
    output->reject(c | c.name=self.name)
  else
    if self.Action.oclIsUndefined() then
      output->reject(c | c.name=self.name)
    else
      --permission is undefined
      if self.Permission.oclIsUndefined() then
        if self.Data='yes' then
          if thisModule.Method_for_Activity.includes( self.MethodForSend ) then
            output->including(':
            The Application

14
have Activity Hijacking (with data) attack in Intent with name:

': ' +

self.name + ' ' + 'and in Component with Package: ' +
self.SendComponentName +

'... /the reasons of this vulnerability are:
the Intent does not have Strong permission,
while it have Action and it is of implicit type./.../
Intent-sending mechanisms is use for Activity./
')
else
if thisModule.Method_for_Service.includes(self.MethodForSend)
then
output->including(
': The Application
have Service Hijacking (with data) attack in Intent with name:

': ' + 'and in Component with Package: ' +
self.SendComponentName +

'... /the reasons of this vulnerability are:
the Intent does not have strong permission,
while it have Action and it is of implicit type./.../
Intent-sending mechanisms is use for Service./
')
else
if thisModule.Method_for_Receiver.includes(self.MethodForSend)
then
output->including(
': The Application
have Broadcast Theft (with data) attack in Intent with name:

': ' + 'and in Component with Package: ' +
self.SendComponentName +

'... /the reasons of this vulnerability are:
the Intent does not have strong permission,
while it have Action and it is of implicit type./.../
Intent-sending mechanisms is use for Receiver./
')
else
output->reject(c | c.name=self.name)
endif
endif
endif
else

if thisModule.Method_for_Activity.includes(self.MethodForSend)
  then
  output->including('The Application have Activity Hijacking (without data) attack in Intent with name: ' + self.name + ' and in Component with Package: ' + self.SendComponentName + '.../ the reasons of this vulnerability are:
  the Intent does not have strong permission, while it have Action and it is of implicit type. /.../
  Intent-sending mechanisms is use for Activity./')
else
  if thisModule.Method_for_Service.includes(self.MethodForSend)
    then
      output->including('The Application have Service Hijacking (without data) attack in Intent with name: ' + self.name + ' and in Component with Package: ' + self.SendComponentName + '.../ the reasons of this vulnerability are:
      the Intent does not have strong permission, while it have Action and it is of implicit type. /.../
      Intent-sending mechanisms is use for Service./')
  else
    if thisModule.Method_for_Receiver.includes(self.MethodForSend)
      then
        output->including('The Application have Broadcast Theft (without data) attack in Intent with name: ' + self.name + ' and in Component with Package: ' + self.SendComponentName + '.../ the reasons of this vulnerability are:
        the Intent does not have strong permission, while it have Action and it is of implicit type. /.../
        Intent-sending mechanisms is use for Receiver./')
    else
      ;
output->reject(c | c.name==self.name)
endif
endif
endif
endif
else
−−permission is strong
if thisModule.StrongPermission.includes(self.Permission) then
output->reject(c | c.name==self.name)
else
−−permission is not strong
if self.Data=='yes' then
if thisModule.Method_for_Activity.includes(self.MethodForSend) then
output->including('': The Application
have Activity Hijacking (with data) attack in Intent with name:
self.name+' '+'and in Component with Package: '+'
self.SendComponentName+'
'... /the reasons of this vulnerability are:
the Intent have permission but permission is not Strong,
while it have Action and it is of implicit type./.../
Intent—sending mechanisms is use for Activity./'
else
if thisModule.Method_for_Service.includes(self.MethodForSend) then
output->including('': The Application
have Service Hijacking (with data) attack in Intent with name:
self.name+' '+'and in Component with Package: '+'
self.SendComponentName+'
'... /the reasons of this vulnerability are:
the Intent have permission but permission is not Strong,
while it have Action and it is of implicit type./.../
Intent—sending mechanisms is use for Service./' )
else
if thisModule.Method_for_Receiver.includes(self.MethodForSend) then

output->including(
    ': The Application
    have Broadcast Theft (with data) attack in Intent with name:
    ' +
    self.name + ' ' + 'and in Component with Package: ' +
    self.SendComponentName +
    '... /the reasons of this vulnerability are:
    the Intent have permission but permission is not Strong,
    while it have Action and it is of implicit type. .../
    Intent-sending mechanisms is use for Receiver. / ' )
else
output->reject(c | c.name == self.name)
endif
endif
endif
else
if thisModule.Method_for_Activity.includes(self.MethodForSend)
    then
output->including(
    ': The Application
    have Activity Hijacking (without data) attack in Intent with
    name: ' +
    self.name + ' ' + 'and in Component with Package: ' +
    self.SendComponentName +
    '... /the reasons of this vulnerability are:
    the Intent have permission but permission is not Strong,
    while it have Action and it is of implicit type. .../
    Intent-sending mechanisms is use for Activity. / ' )
else
if thisModule.Method_for_Service.includes(self.MethodForSend)
    then
output->including(
    ': The Application
    have Service Hijacking (without data) attack in Intent with
    name: ' +
    self.name + ' ' + 'and in Component with Package: ' +
    self.SendComponentName +
    '... /the reasons of this vulnerability are:
    the Intent have permission but permission is not Strong,
    while it have Action and it is of implicit type. .../


Intent-sending mechanisms is use for Service./ ') else
if thisModule.Method_for_Receiver.includes(self.MethodForSend)
then
output->including(' :
The Application
have Broadcast Theft(without data) attack in Intent with name :
self.name+' '+'and in Component with Package :
self.SendComponentName+
.../the reasons of this vulnerability are:
the Intent have permission but permission is not Strong,
while it have Action and it is of implicit type./.../
Intent-sending mechanisms is use for Receiver./ ')
else
OclUndefined
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
)
->asSet()->asSequence();
Listing 11: Helper to check Unauthorized Intent Receipt vulnerability.