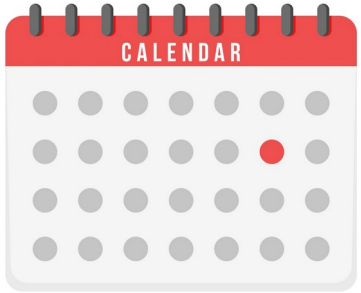
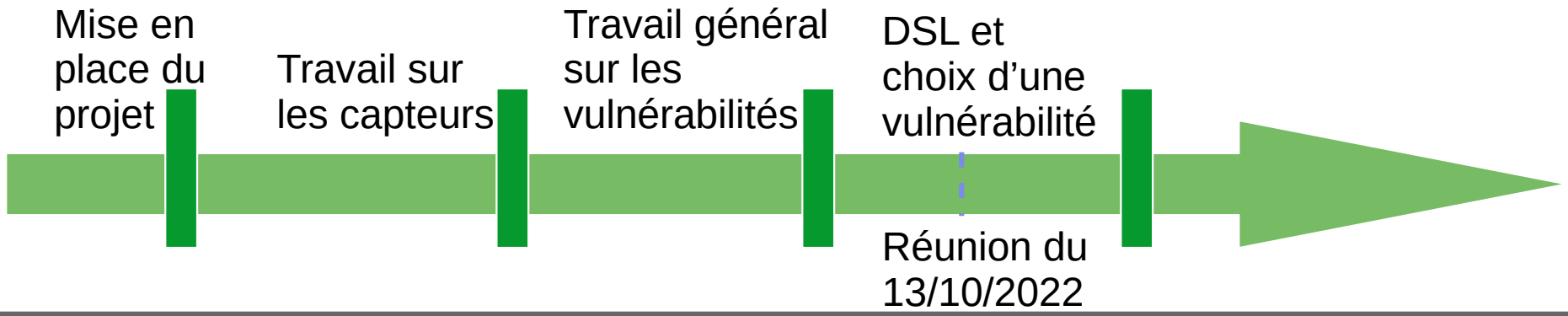
Context-aware security testing of Android applications

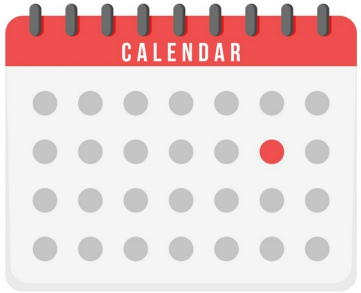
État du projet : au 14/12/2022

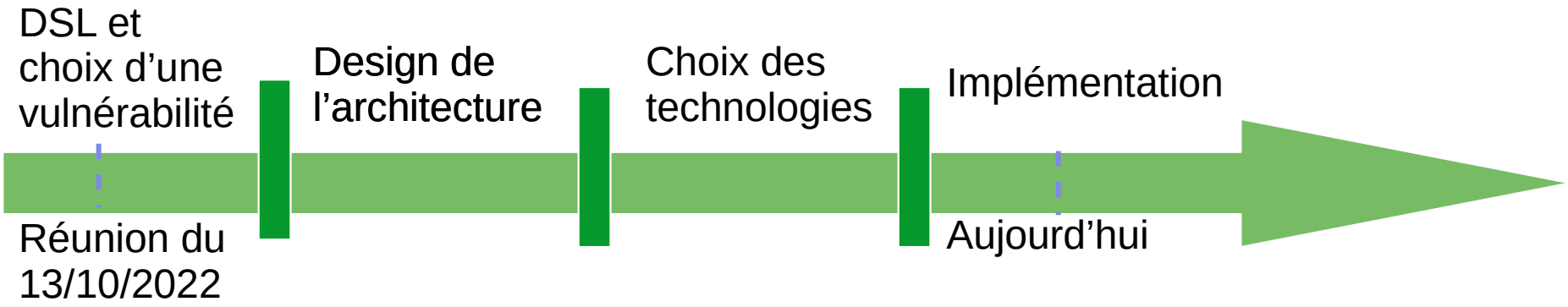




Context-aware security testing of Android applications

État du projet : au 14/12/2022



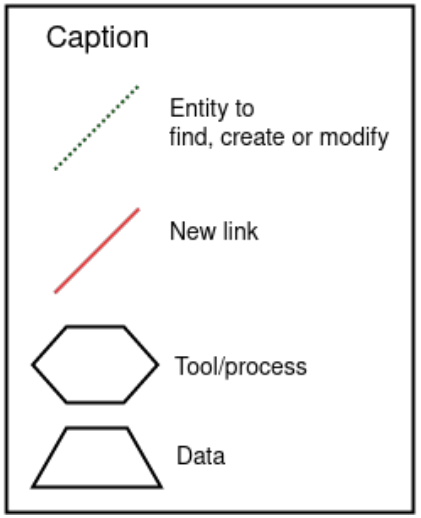


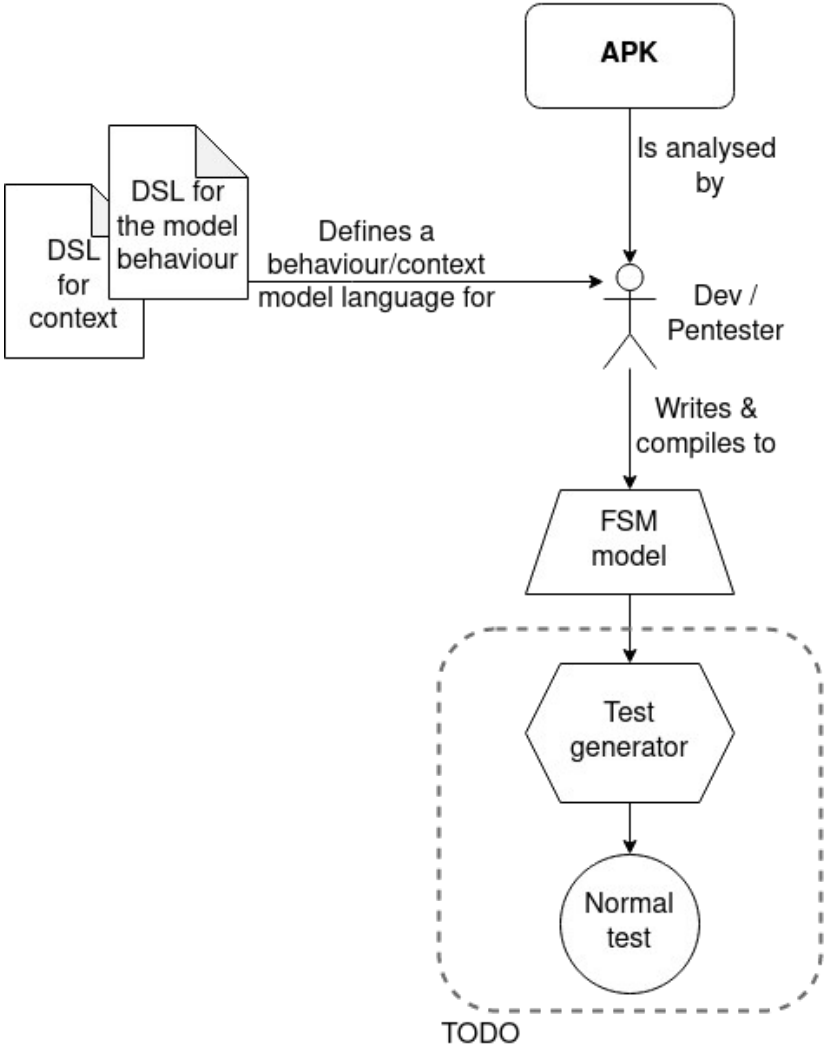
Base idea by Abdallah Adwan

Présentation travail A.Abdallah

But :

- Permettre à un développeur de tester le comportement de son projet en prenant en compte le contexte de l'application

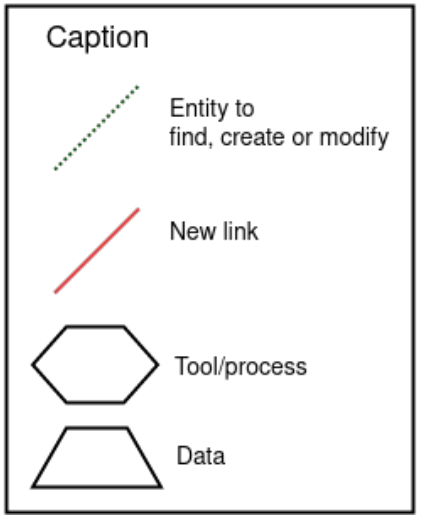


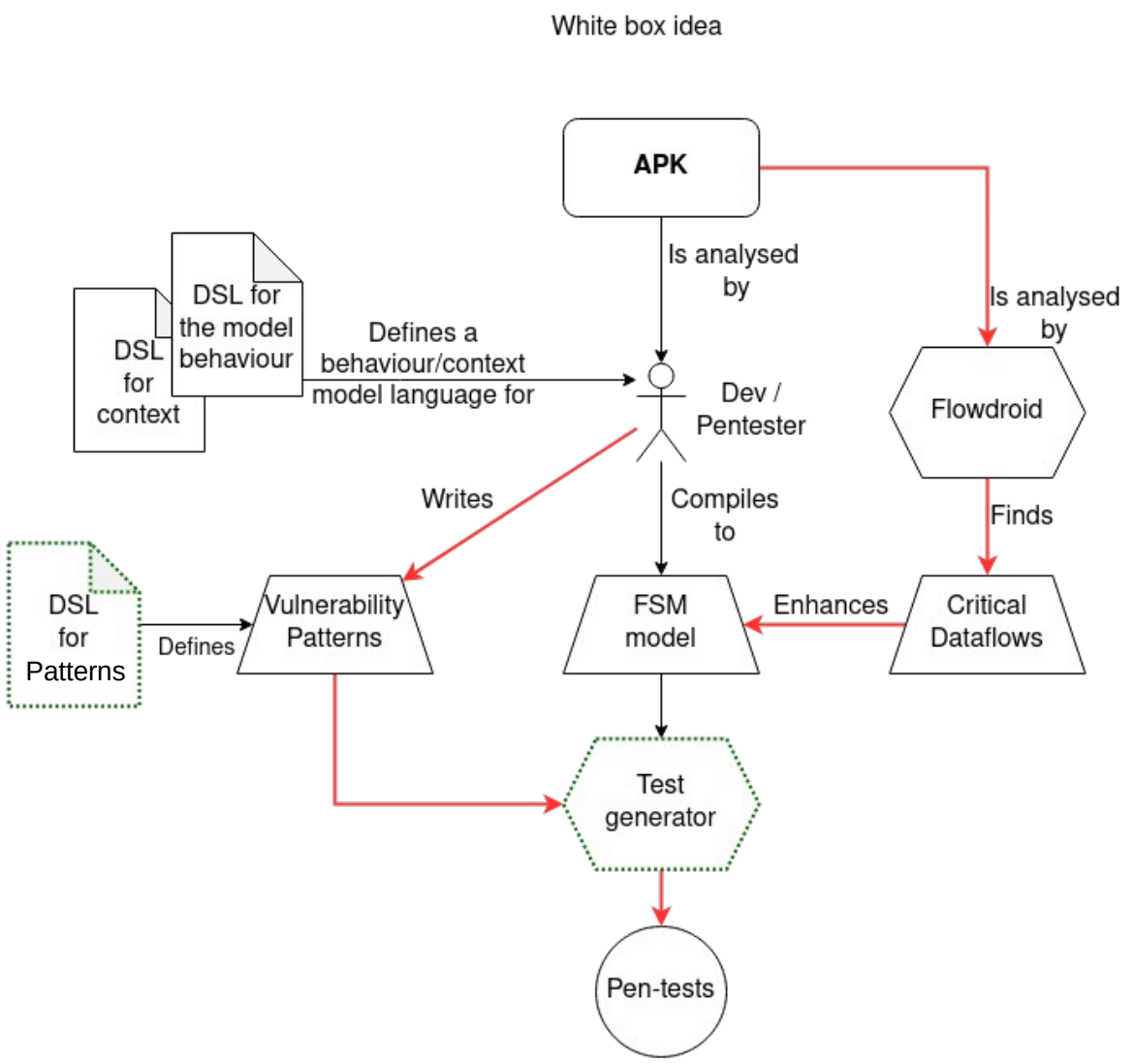


Implémentation choisie

But :

- Permettre à un développeur de tester puis de démontrer la présence des vulnérabilités définies par les patterns en fonction du contexte de l'application

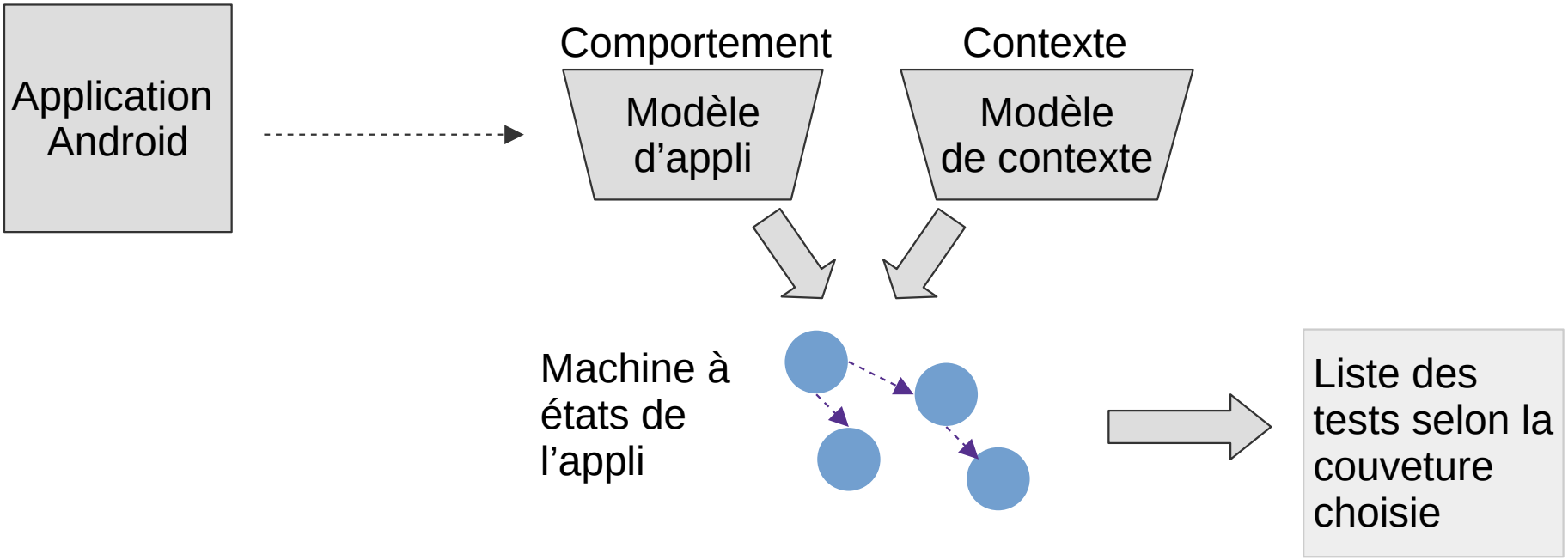




Étapes de ma contribution

- Point de départ : Théorie d’Adwan Abdallah
  - Génération de tests « normaux »
- 1ère étape : Enrichissement des tests via Flowdroid
- 2ème étape : Génération de tests « simples »
- 3ème étape : Écriture des patterns
- 4ème étape : Match entre les pattern et le modèle
- 5ème étape : Génération du test de sécurité

Point de départ : Théorie d'Adwan Abdallah



Example CBML (contextual behaviour ML)

```
contexts {
    INTERNET_CONNECTIVITY,
    COUNTRY,
    LOCATION
}

situations {
    INTERNET_DISCONNECTED : INTERNET_CONNECTIVITY
}

statemachine ABSTRACT_SM {

    state START {
        transition on APP_STARTED -> ENTER_PHRASE_ACTIVITY
    }

    super state ENTER_PHRASE_ACTIVITY abstracts TRANSLATE_PHRASE_ACTIVITY_SM {
        transition on TRANSLATION_SUCCEEDED -> VIEW_TRANSLATION_ACTIVITY
        transition on TERMINATE_BUTTON_CLICKED -> EXIT
    }

    super state VIEW_TRANSLATION_ACTIVITY abstracts VIEW_TRANSLATION_ACTIVITY_SM {
        transition on TERMINATE_BUTTON_CLICKED -> EXIT
        transition on BACK_BUTTON_CLICKED -> ENTER_PHRASE_ACTIVITY
    }

    state EXIT
}
```

```
statemachine TRANSLATE_PHRASE_ACTIVITY_SM {

    state ENTER_PHRASE_ACTIVITY {
        transition on TRANSLATE_BUTTON_CLICKED -> TRANSLATE
        transition on PHRASE_TEXTFIELD_SELECTED -> ENTER_PHRASE
    }

    state ENTER_PHRASE {
        transition on PHRASE_ENTERED -> ENTER_PHRASE_ACTIVITY
    }

    state TRANSLATE awareof COUNTRY, INTERNET_CONNECTIVITY {
    }

}

statemachine VIEW_TRANSLATION_ACTIVITY_SM {
    state TRANSLATION_ACTIVITY {
        transition on NONE -> DISPLAY_TRANSLATION
    }

    state DISPLAY_TRANSLATION {
    }
}

adaptation for INTERNET_DISCONNECTED at TRANSLATE {

    state TRANSLATE {
        transition on TRANSLATION_FAILED -> DISPLAY_WARNING
    }

    state DISPLAY_WARNING{
        transition on WARNIMG_DISMISSED -> external TRANSLATE_PHRASE_ACTIVITY_SM.ENTER_PHRASE_ACTIVITY
    }

}
```

1ère étape : Enrichissement des tests via Flowdroid

Flowdroid est un outil open-source qui permet :

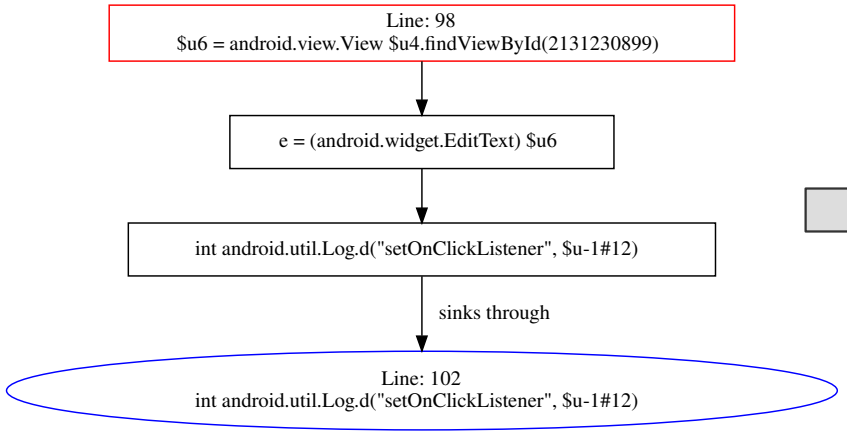
- La détection des liens entre fonctions dites sources et sink
- Générer le graphe d’appel de fonction des liens détéctés

1ère étape : Enrichissement du modèle via Flowdroid

Flowdroid est un outil open-source qui permet :

- La détection des liens entre fonctions dites sources et sink
- Générer le graphe d'appel de fonction des liens détéctés

Passe de détection



Enrichissement du modèle

```
state SEND_MESSAGE awareof INTERNET_CONNECTIVITY {
    transition on SEND_MESSAGE_CLICKED -> HANDLE_SLOW_INTERNET
    dataflows {
        source internet
    }
}

state HANDLE_SLOW_INTERNET{
    transition on NONE -> external SEND_MESSAGE_ACTIVITY_SM.SENDER
    dataflows {
        sink log_d ( source SEND_MESSAGE_ACTIVITY_SM.SEND_MESSAGE.internet)
    }
}
```

2ème étape : Génération de tests « simples »

- On génère des tests d’après le modèle de l’application

```
------------------------------------------------------------

state -> START
  event -> APP_STARTED
state -> ENTER_PHRASE_ACTIVITY
  event -> TRANSLATE_BUTTON_CLICKED
state -> TRANSLATE
  event -> TRANSLATION_SUCCEEDED with context _4g  AND france
state -> TRANSLATION_ACTIVITY
  event -> NONE
state -> DISPLAY_TRANSLATION
  event -> BACK_BUTTON_CLICKED
state -> ENTER_PHRASE_ACTIVITY
  event -> TRANSLATE_BUTTON_CLICKED
state -> TRANSLATE
  event -> TERMINATE_BUTTON_CLICKED with context spain  AND fast3G
state -> EXIT


------------------------------------------------------------

state -> START
  event -> APP_STARTED
state -> ENTER_PHRASE_ACTIVITY
  event -> TRANSLATE_BUTTON_CLICKED
state -> TRANSLATE
  event -> TRANSLATION_SUCCEEDED with context germany  AND high_latency
state -> TRANSLATION_ACTIVITY
  event -> NONE
state -> DISPLAY_TRANSLATION
  event -> BACK_BUTTON_CLICKED
state -> ENTER_PHRASE_ACTIVITY
  event -> TRANSLATE_BUTTON_CLICKED
state -> TRANSLATE
  event -> TERMINATE_BUTTON_CLICKED with context germany  AND high_latency
state -> EXIT


------------------------------------------------------------
```

- Couverture de code : AllTransitions+AllContexts

```
------------------------------------------------------------

state -> START
  event -> APP_STARTED
state -> ENTER_PHRASE_ACTIVITY
  event -> TRANSLATE_BUTTON_CLICKED
state -> TRANSLATE
  event -> TRANSLATION_SUCCEEDED with context _4g  AND uk
state -> TRANSLATION_ACTIVITY
  event -> NONE
state -> DISPLAY_TRANSLATION
  event -> TERMINATE_BUTTON_CLICKED
state -> EXIT


------------------------------------------------------------
```

3ème étape : Écriture des patterns

- Totalement délié de l'application :
  - Un pattern peut-être réutilisé pour toutes les applications

Exemple simpliste

```
Vulnerability "Log.d Leak" {
    description "Log.d kept in code makes it vulnerable to leakage of data"

    function {
        Source private * ,
        Sink "Log.d" {
            parameter {
                private
            }
        }
    }
}
```

Le pattern est valide quand :

Une source privée est redirigée vers le paramètre de la méthode « log.d »

4ème étape : Match entre les pattern et le modèle

Chaque test généré est teste avec une vulnérabilité.
On génère un rapport pour chaque combinaison Test-Pattern

Exemple de rapports positifs :
- Vulnérabilité « log.d leak »

```
Report is positive for Vulnerability [["Log.d Leak"]
    {"Log.d kept in code makes it vulnerable to leakage of data"}
    NO STATICMODEL IMPLEMENTATION FOR NOW
    Sink -> SinkModel [name=log_d, parameters:
---  SourceModel [name=private, AnyValue]]
] - TestPath {fr.castav.lcis.Checker.model.TestPaths$TestPath@4417af13}

Report is positive for Vulnerability [["Log.d Leak"]
    {"Log.d kept in code makes it vulnerable to leakage of data"}
    NO STATICMODEL IMPLEMENTATION FOR NOW
    Sink -> SinkModel [name=log_d, parameters:
---  SourceModel [name=private, AnyValue]]
] - TestPath {fr.castav.lcis.Checker.model.TestPaths$TestPath@67594471}
```

A venir

- Tests complétés d’une preuve d’exploitation

- Automatisation de l’enrichissement du modèle avec flowdroid

-Amélioration des capacités des pattern de vulnérabilité

Point de vue plus global de l'architecture

Modèle de l'application Android

CBML

```
<TestPath>
    <state name="START">
        <transition name="APP_STARTED"/>
    </state>
    <state name="SEND_MESSAGE">
        <transition name="SEND_MESSAGE_CLICKED">
            <contexts>
                <context origin="INTERNET_CONNECTIVITY">offline</context>
            </contexts>
        </transition>
        <dataflows>
            <dataflow name="internet" type="Source"/>
        </dataflows>
    </state>
    <state name="SENDER">
        <transition name="WRONG"/>
        <dataflows>
            <dataflow name="enter_value" type="Input"/>
        </dataflows>
    </state>
    <state name="DISPLAY_WARNING">
        <transition name="BACK_BUTTON_PRESSED"/>
    </state>
    <state name="SEND_MESSAGE">
        <transition name="SEND_MESSAGE_CLICKED"/>
        <dataflows>
            <dataflow name="internet" type="Source"/>
        </dataflows>
    </state>
    <state name="SENDER">
        <transition name="GOOD_MESSAGE"/>
        <dataflows>
            <dataflow name="enter_value" type="Input"/>
        </dataflows>
    </state>
    <state name="SHOW_ANSWER">
        <transition name="SUCCESS"/>
    </state>
    <state name="EXIT"/>
</TestPath>
```

Patterns de vulnerabilité

VPat

generator

Modèle générique de contexte

CDL

Tests sous forme XML

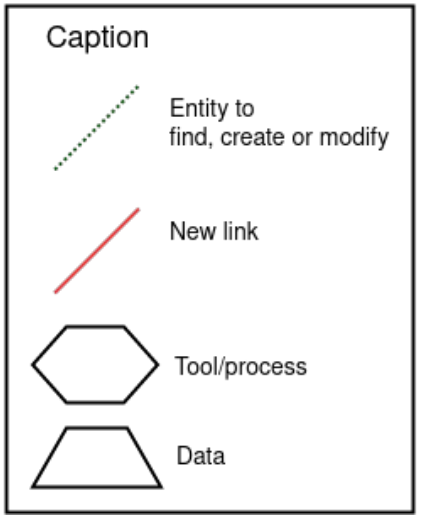
VPatChecker

Rapports

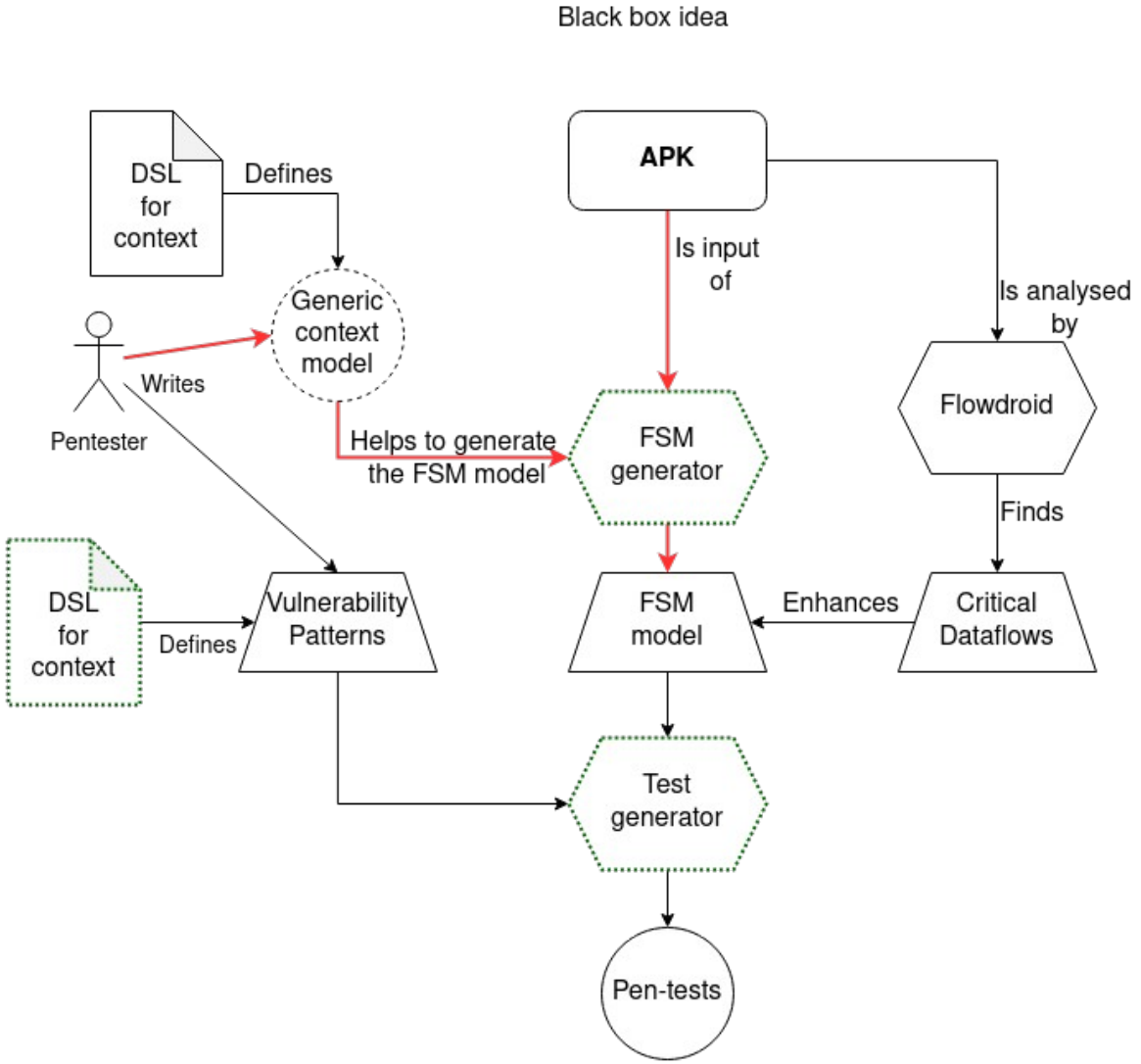
Idée de design pour la suite de ce projet

But :

- Détecter puis démontrer la présence de patterns dans une application quelconque en fonction de son contexte d'exécution



Black box idea



Motivations

Motivations de l'outil :

Objectif philosophique :

- Permettre à un développeur Android de tester puis de démontrer la présence des vulnérabilités définies par les patterns en fonction du contexte de l'application.
- Séparer en trois parties distinctes :
  - Développeur : Programme le code de l'application Android
  - Pentesteur : Écrit des pattern de vulnérabilité (Décrit et classifie les vulnérabilités)
  - Outil de test : Donne le moyen à son utilisateur de vérifier l'existence de vulnérabilités indépendamment de sa connaissance des vulnérabilités
- Simplifier l'ajout de nouvelles vulnérabilités en ayant un langage de pattern simple (à écrire) mais complet (contexte).

Objectif téchnique :

- Séparer le développement de l'outil de test et de la vulnérabilité à tester. → Via des patterns de vulnérabilité
- Possibilité de patterns sensibles au contexte (statique ou dynamique), aux flux de données et aux valeurs d'entrée

Idée globale de l'utilisation du projet