Méthodologie d'enseignement du développement et de l'évaluation d'application carte par un outil venant de la recherche académique

Germain Jolly[1], Sylvain Vernois[1] and Christophe Rosenberger[1]
[1]*Normandie Université; ENSICAEN; UMR 6072 GREYC, Caen, France*
{*germain.jolly, sylvain.vernois, christophe.rosenberger*}*@ensicaen.fr*

Keywords: Sécurité, Analyse, Application carte, Enseignement, Évaluation, Framework WSCT

Abstract: Les cartes à puce sont des éléments sécurisés par de nombeures spécifications et standards. Cependant, des vulnérabilités sont parfois découvertes. Durant nos recherches sur la sécurité des cartes à puce, nous avons étudié l'évaluation des applications cartes. Dans ce papier, nous présentons un outil d'aide au développement d'applications cartes basé sur l'observation et la détection d'anomalie présente dans l'application développée par les élèves ingénieurs de l'ENSICAEN. Durant leur cursus, les élèves apprenent à développer une application carte à partir de spécifications, c'est-à-dire une applet JavaCard. La méthode d'aide au développement proposée ici est en deux parties : la première est un terminal de test permettant de détecter si une carte est correcte à chaque étape de développement, et la seconde permet d'avoir plus d'information sur une erreur possiblement existante afin de permettre le guidage du développement et de la correction d'erreurs possibles. Outre l'enseignement du développement JavaCard, l'élève acquiert, par cette méthodologie, une formation pratique axée sur la validation de développement, la détection d'erreur d'implémentation par l'utilisation d'un oracle et plus généralement l'évaluation.

1 INTRODUCTION

Alors que les cartes à puce ont envahi nos modes de vie, la sécurité des applications demeurent au centre des questionnements. En effet, EMVCO annonce 1.62 milliards cartes EMV de paiement et 23.8 millions de terminaux EMV en circulation dans le monde (EMVCo, 2013). Le développement d'une application carte n'est pas facile et les développeurs se heurtent à certaines difficultés.(Rankl, 2007). Les spécifications peuvent paraitre trop importantes ou trop flous selon l'application. Le test est une technique de découverte d'anomalie dans un système donné par rapport à la connaissance de son comportement théorique. Néanmoins, le test complet de tous les cas possibles semble couteux en temps et en ressources, surtout si le testeur n'a pas grande expérience dans le domaine.

Pour le test fonctionnel et sécuritaire, il est nécessaire de valider chaque étape et chaque élement d'une carte à puce. Il n'est pas réaliste d'avoir une carte à puce sécurisée et certifiée et de lui installer un programme mal implémenté. La partie logiciel doit être vérifiée et certifiée. Il est donc question de l'évaluation logicielle durant la phase d'implémentation de l'application dans ce papier.

Alors que dans le domaine insdustriel, l'évaluation est princiapelement permise par des outils de test, le milieu acadmique propose des techniques moins conventionnelles. Voici quelques techniques de vérification et de validation (V&V) formant la base de cette étude : contrôle (reviews), analyse (vérification mathématique) ou test (entrée/sortie d'un système). La plupart des travaux académique sont des méthodes boite blanche.

- La vérification statique est une méthode d'analyse ((Distefano and Parkinson J, 2008), (Philippaerts et al., 2014) ou (Ahrendt et al., 2005)) permettant de vérifier que les bonnes pratiques de développement sont suivies.
- Le model checking est une méthode consistant à vérifier si un model d'un système suit les spécifications ((Sabatier and Lartigue, 1999)). Nous visitons tous les chemins possible de la machine à états correspondant au système cible.
- Le test, méthode la plus utilisée dans le monde industriel, peut être effectué à différentes phases du cycle de vie de la carte (par exemple, lors de la phase de développement ou lors de la certification) ((Philipps et al., 2003), (van Weelden et al., 2005)).

- Le fuzzing, apparu plus récemment, est une technique de génération automatique de cas de test ((Lancia, 2011), (Bekrar et al., 2012) ou (Alimi et al., 2014).) afin d'automatiser la recherche d'anomalie.

Certaines méthodes formelles étant trop lourdes (pour le concepteur mais surtout pour l'utilisateur), nous avons opter pour une méthode plus légère tout en cherchant une efficacité au moins similaire au test (le fuzzing pouvant être assimilé celui-ci. La méthode mise en place durant nos recherches a pour objectifs d'être configurable simplement afin de créer des cas de tests ou des comportements locaux et attendus de la carte à puce et utilisable facilement en vue de l'évaluation d'une application carte durant son développement. En représentant l'application par comportements locaux et attendus, nous pouvons analyser à la volée la conformité de l'implémentation par rapport aux spécifications fournis au développeur, ce qui complète le test effectué sur un système.

Ainsi, nous nous sommes placés dans le cas d'aide au développement dans un cas concret : l'enseignement du développement d'une application carte dans le cadre d'une formation d'ingénieur. Cette méthode est configurable par l'enseignant ayant défini le contenu du TP et ayant une expérience plus importante dans ce domaine et est utilisable par l'éléve durant la phase de développement ainsi que par l'enseignant pour la notation des TPs. La méthodologie d'énseignement proposée est divisée en trois étapes : pour commencer, l'élève développe une partie de ce qui est demandé, ensuite, il lance un terminal de test afin de vérifier son avancement de développement et enfin, si nécessaire, il peut obtenir plus d'informations sur l'erreur qu'il a fait durant le développement. Cette méthodologie permet d'apprécier le développement d'une applet JavaCard de manière autonome et sécurisée mais aussi permet d'aborder les notions de sécurité, d'évaluation, de test et d'observation d'anomalie. La méthode et l'outil proposé permet un double enseignement (évaluation et développement) ce qui permet d'éviter de dissocier ces domaines indissociables et d'apporter une certaine rigueur à la formation informatique.

Dans la seconde section, nous posons les bases des applications cartes et de leur certification. La méthodologie est définie dans la troisième section. Finalement, une illustration concrète est fourni dans la dernière partie sur une application de type Porte Monnaie Electronique (PME).

2 CONTEXTE

Nous nous concentrons sur une application JavaCard. Par exemple, pour l'application de paiement, EMVCo fournit les spécifications EMV (Europay MasterCard Visa) et les spécifications CPA (Common Payment Application) (Watanabe et al., 2006). De plus, les constructeurs les complètent par des spécifications propriétaires.

Durant le cycle de vie d'un produit, nous devons nous poser plusieurs questions (Radatz et al., 1990) :

- **Vérification :** Avons-nous construit le système correctement ?
- **Validation :** Avons-nous construit le bon système ?
- **Évaluation :** Comment évaluer les résultats des précédentes questions ?
- **Certification :** Ces résultats sont-ils en accord avec les critères d'une structure de certification ?

Afin d'évaluer et de valider un produit, plusieurs tests sont faits durant les étapes de vie de la carte à puce. (Rankl and Effing, 2010). Tous les aspects de la carte (application, puce et données de personnalisation) sont étudiés et doivent être validés pour assurer un produit sécurisé. La certification permet de reconnaitre le niveau de sécurité d'un produit par un laboratoire externe au constructeur.

Selon ISO 8402 (for Standardization, 1994), la qualité est définie par la totalité des fonctionnalités et des caractéristiques d'un produit ou d'un service mais aussi par l'absence d'erreurs et la résistance aux comportements anormaux (un comportement anormal n'étant pas prévu dans le cahier des charges du produit et pouvant aboutir à une utilisation inattendue du système). Dans ce domaine, le test est principalement utilisé pour s'assurer de la qualité d'un produit, c'est-à-dire effectuer son évaluation. On va donc parler d'évaluation logicielle lorsque l'on effectue cette analyse sur la partie logicielle de la carte à puce : l'application. Même si le test permet la découverte d'erreurs (plus l'expérience du testeur est élevée plus l'évaluation sera complète), il ne permet pas de couvrir la totalité des cas possibles. Le niveau minimum demandé pour la couverture des instructions est de 95% et le niveau demandé pour la couverture des chemins est de 80%. (Rankl and Effing, 2010).

Un terminal communique avec une carte à puce en envoyant des commandes de type Application Protocol Data Unit (APDU) et en recevant des réponses APDU d'après ISO/IEC 7816 . Sur la figure 1, nous pouvons voir la structure de ces messages. CLA indique le type de la commande; INS correspond à l'identifiant de la commande; P1 et P2 sont les

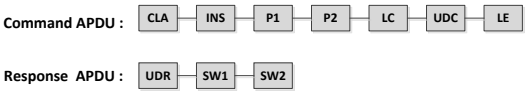

Figure 1: Communication entre la carte à puce et le terminal

paramètres, LC est la longueur des données fournies avec une commande (ceci est optionel) et Le indique la longueur des données devant être contenue dans la réponse. La rponse contient trois éléments : SW1 et SW2 correspondent à l'état de traitement de la commande et les données contenues dans UDR sont des données envoyées par la carte au terminal pour la suite de la transaction.

3 MÉTHODOLOGIE DE VÉRIFICATION PROPOSÉE

3.1 Objectives

À partir d'une carte certifié (ou de spécifications), nous pouvons générer un ensemble de propriétés qui représente le comportement local et attendu de l'application carte. L'objectif est de détecter à la volée si l'application ne suit pas son comportement normal. Nous avons donc créer un language de propriété permettant de représenter l'application de manière globale par sa machine à état ou de manière locale par une propriété.

3.2 Le langage de représentation

Afin de détecter un comportement anormal et de donner plus d'informations sur les causes qui ont déclenchées une erreur, nous avons représenté chaque élément transitant entre la carte et le terminal.

3.2.1 Discussion

Contrairement aux méthodes telle que le model checking ou le théorème par preuves, nous vérifions les propriétés par l'observation de la communication durant la transaction. En fait, dans les études comme (Posegga and Vogt, 1998) (Sabatier and Lartigue, 1999) (Lanet and Requet, 2000) (Jacobs et al., 2004) ou (Haneberg et al., 2007), la vérification a besoin d'un modèle formelle ou l'accès au code source. Avec notre langage, nous pouvons représenter et vérifier le comportement attendu en utilisant seulement les données transmises. Ces éléments sont bien s$\hat{}$ur configurables par exemple pour la gestion des données pouvant être cryptées.

3.2.2 Définitions des balises

Pour représenter le comportement, nous avons commencé par définir les balises représentant la communication APDU de base.

- **Instruction :** contient les champs CLA et INS
- **Parameters :** contient les champs P1 et P2
- **Status :** contient les champs SW1 et SW2

Pour tous les messages transitant, nous pouvons récupérer des données supplémentaires. Avec ces balises, nous pouvons travailler sur la totalité des données capturées ou bien sur une valeur spécifique d'un objet de type TLV (Tag Longueur Valeur).

- **cdata :** contient la totalité des données contenues dans la commande
- **rdata :** contient la totalité des données contenues dans la réponse
- **Tag :** permet de récupérer une valeur d'un tag spécifique
  - name : nom du tag (exemple : 0x9F27)
  - value : valeur associée au tag (exemple : 0x40)
- **mask :** permet d'appliquer un masque
- **call :** permet de lier un élément à une fonction afin d'ajouter une fonctionnalité sur celui-ci (exemple : vérifier la validité d'une donnée spécifique par rapport sa valeur précédente). Cet balise permet une diversité des traitements à faire sur ces éléments.

Pour les liens logiques entre les commandes et les réponses, nous avons besoin d'éléments simples :

- **And :** tous les éléments sont vrais
- **Or :** un élément doit être vrai
- **Nor :** tous les éléments sont faux

Pour définir une machine à états, nous avons besoin de ces balises :

- **cardstate :** représente l'état de l'application
  - name : nom de l'état de la carte
  - default : correspond à l'état initial de l'application
- **transition :** une transition permet de passer d'un état à un autre
  - name : nom de la transition
  - from : état de départ
  - to : état d'arivée
- **from / to :** Permet d'indiquer la direction
  - name : l'état associé avec la direction

Pour terminer, quelques éléments sont nécessaires pour représenter les propriétés :

- **property :** représente un comportement local et attendu de l'application
  - name : nom de la propriété
  - explanation : description de la propriété
- **step :** indique une suite de comportements attendus

3.3 Comparaison des outils à notre disposition

Plusieurs outils existent afin de travailler avec une carte à puce. Voici une liste d'outils indépendants ou académiques capable de communiquer avec une carte à puce et de créer des programmes de type terminal simplement et efficacement.

- SmartCard Framework : un outil indépendant pour développer une application pour la carte (Rouit, 2011)
- Open SCDP : un ensemble d'outils pour le développement, le test et le déploiement de carte à puce (CardContact, 2012)
- Javaemvreader : un outil pour communiquer et lire des données d'une carte EMV (Sasc, 2014)
- PCSC_sharp : un outil pour travailler avec des cartes à puce (Mueller, 2012)
- CardPeek : un outil afin de lire le contenu des cartes à puce (Pannetrat, 2010)
- OPAL : une bibliothèque pour développer des outils pour les cartes à puce (Bkakria et al., 2011)
- WSCT : une bibliothèque capable de travailler et d'explorer les cartes à puce (Vernois and Alimi, 2010)

	Fonct.*	Doc.	Maint.
SmartCard Framework	⋆	⋆⋆	⋆
OpenSCDP	⋆⋆	⋆⋆⋆	⋆⋆
Javaemvreader	⋆ ⋆	⋆ ⋆	⋆
PCSC_Sharp	⋆	⋆	⋆
OPAL	⋆ ⋆	⋆ ⋆	⋆ ⋆
CardPeek	⋆	⋆ ⋆	⋆ ⋆ ⋆
WSCT	⋆⋆⋆	⋆	⋆⋆⋆

* Fonctionnalités contient les mécanismes d'observation, d'extension et de rejeu. Plus il y a d'étoiles, plus l'outil semble intéressant pour notre étude.

3.4 Discussion

WSCT (Vernois and Alimi, 2010) est un framework développé par Sylvain Vernois et al. au laboratoire GREYC depuis plusieurs années. il est écrit en C# et permet de travailler avec les cartes à puce. Les deux objectifs principaux de cet outil est de fournir :

- **une interface de programmation orienté objet** pour l'accès à un lecteur de carte.
- **une interface graphique évolutive** par une création de plugin simple et accessible.

Nous avons utilisé cet outil afin de mettre en place notre langage de propriété et d'observer la communication entre une carte à puce et un terminal et suivre le comportement de l'application carte grâce aux capacités du framework mais aussi de détecter les comportements anormaux grâce à la surveillance d'un oracle simple et efficace basé sur des propriétés.

4 ILLUSTRATION

Dans cette partie, l'utilisation de la méthode est montré dans le cadre d'un enseignement du développement JavaCard. Pour se faire, nous utilisons trois environnements : Eclipse permet le développement de l'application, un terminal de test est effectué avec le framework WSCT et permet d'envoyer des plans de test par étape de développement à la carte et enfin un observateur, aussi basé sur le framework WSCT, utilisant le language vu dans la section 3 qui permet l'analyse plus poussée de l'anomalie détectée.

4.1 Représentation de l'application cible

La représentation de l'application est l'oracle à utiliser pendant l'évaluation de l'application. Dans cette étude, l'enseignant fourni aux élèves des spécifications mais aussi des plans de test, une machine à états et une collection de propriétés. Le premier document permet aux élèves de débuter le développement de l'application. Grâce aux éléments supplémentaires, l'étudiant peut valider son développement et se laisser guider par les plans de test, ou plutôt les résultats de ces plans de test, mais aussi par la découvertes des erreurs d'implémentations par l'observation des communications ce qui permet un guidage plus précis par la détection de violation de propriétés. Ceci permet à l'enseignant d'appréhender la notion de sécurité et d'évaluation d'un système basée sur une méthode de

test complété par un outil créé dans un contexte de recherche académique.

Listing 1: machine à états partielle de l'application de paiement EMV

```
<machine>
 <cardstate name="Selected"/>
 <cardstate name="Initiated"/>
 <transition text="GPO">
  <from name="selected" dir="Down" />
  <to name="initiated" dir="Up" />
  <and>
   <instruction="0x80A8"/>
   <status="0x9000"/>
  </and>
 </transition>
 <transition text="else">
  <from name="initiated" dir="Right" />
  <to name="selected" dir="Right" />
 </transition>
</machine>
```

Voici un exemple de propriétés :

Listing 2: Quelques exemples de propriétés de l'application EMV

```
<properties>
     <property name="blocageOK">
    <step>
      <instruction instruction="0x9020"
          />
      <status status="0x9170" />
    </step>
    <step>
      <instruction instruction="0x9020"
          />
     <require>
      <status status="0x9170" />
      </require>
   </step>
  </property>

  <property name="blockOneLeft">
    <step>
      <instruction instruction="0x9020"
          />
      <status status="0x9111" />
    </step>
    <step>
      <instruction instruction="0x9020"
          />
      <status status="0x9170" />
      <nor>
        <cdata cdata="01020304"/>
      </nor>
    </step>
  </property>

  <property name="block">
    <step>
      <instruction="0x9020" />
     <nor>
      <status="0x9000" />
     <nor>
    </step>
    <step>
      <instruction="0x9020" />
     <nor>
      <status="0x9000" />
     <status="0x9170" />
     <nor>
     <require>
      <call method="control" />
     </require>
    </step>
   </property>
</properties>
```

Un compteur permettant de bloquer la carte (et ainsi éviter l'attaque de type brute force) doit être présent sur l'application. Ces trois propriétés permettent de détecter une anomalie et même de spécifier la cause de l'erreur.

- La première permet de savoir si le blocage est effectif et non pas seulement affiché.
- La seconde vérifie que l'application retourne bien le message correspondant au blocage (le blocage semble alors être pris en compte).
- la dernière permet de savoir si le compteur se décrémente correctement.

4.2 Développement JavaCard

L'élève peut alors développer son application sur eclipse. Au fur et à mesure qu'il avance, il peut lancer son application via JCOP.

4.3 Connexion entre l'application et le framework WSCT

Ensuite, le framework WSCT peut être lancé afin de se connecter à la carte émulée, ce qui ne change rien pour nous car nous nous focalisons sur la partie logicielle. La figure 3 nous donne l'interface de connexion de WSCT.

4.4 Terminal de test

Un plugin nommé PME Terminal permet d'appliquer à l'application des plans de test pour chaque étape de développement. Ceux-ci sont bien sûr créés par l'enseignant et doivent correspondre aux spécifications fournis l'étudiant. Cette méthode est donc une méthode de développement guidée en partie par le test.

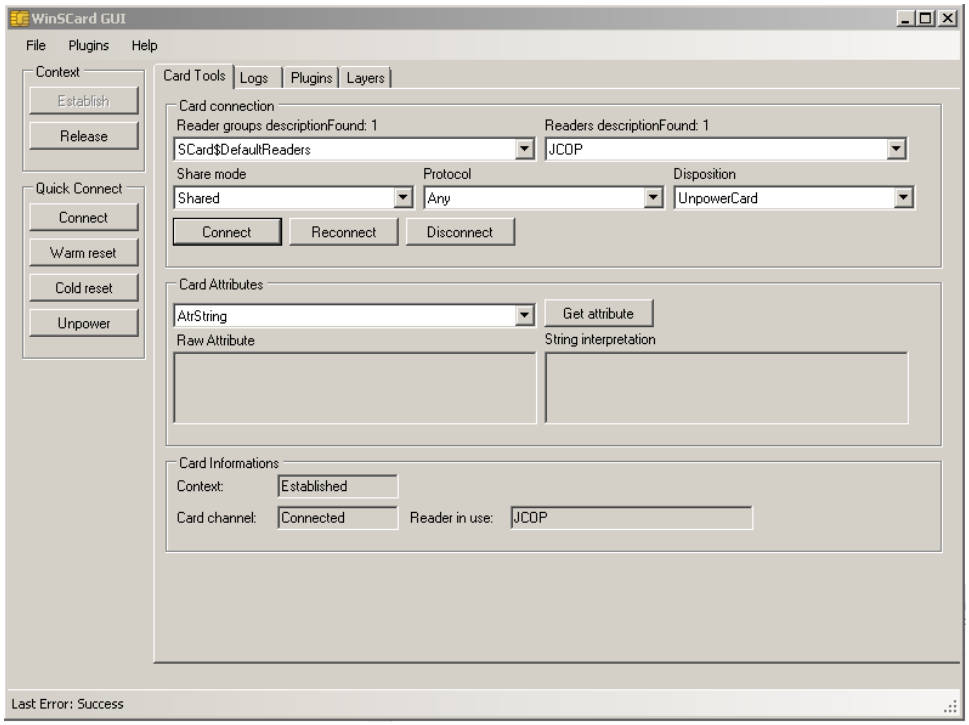

Figure 2: Environnement de connexion de WSCT

Sur la figure 4, nous pouvons voir que la seconde étape de développement n'est pas validée. En effet, un test n'est pas passé.

4.5 Analyse des erreurs d'implémentations

Un second plugin, créé dans le cadre de travaux de recherche peut alors être lancé en parallèle afin de détecter les anomalies le plus précisemment possible. Il s'agit de la partie complémentaire que l'enseignant peut générer manuellement ou de manière plus automatique (des méthodes de génération sont en cours de développement : une méthode triviale, une méthode intelligente par un algorithme génétique et une méthode formelle).

Sur la figure 5, nous avons une capture du plugin correspondant à la fin de transaction, c'est-à-dire dans notre cas la fin du plan de test. On observe une violation de propriété qui permet de savoir quelle fonctionnalité n'est pas valide. Le blocage de la carte n'est donc pas effective. Il s'agit bien sûr d'un exemple. Plus l'enseignement est préparé par l'enseignant, plus les propriétés seront précises. Tout est configurable selon la volonté de l'enseignant.

4.6 Résultats

Voici les résultats sur quatre implémentations :

- A : sans erreur, implémentation de référence.
- B : avec erreur sur le compteur.
- C : avec erreur sur le blocage.
- D : implémentation incluant une backdoor.

	A	B	C	D
Test	OK	ECHEC	ECHEC	OK
Propriété	OK	ALERTE	ALERTE	ALERTE

* OK signifie que tous les plans de test sont passés avec succès ou qu'aucune violation de propriété n'est visible, ECHEC signifie qu'un test n'est pas passé et ALERTE signale une violation de propriété.

A l'inverse du test, les propriétés sont plus génériques et plus simples (elles ne sont définies que sur quelques couples commandes/réponses). Un plan de test implique de rejouer la transaction à partir du début. De plus, dans les cas A, B et C, nous concluons que la détection par le test est équivalente la détection par les violations de propriétés. Les plans de tests permettent de savoir si l'application est correcte ou non. Mais c'est en regardant l'analyse de communication et la violation des propriétés que nous pouvons

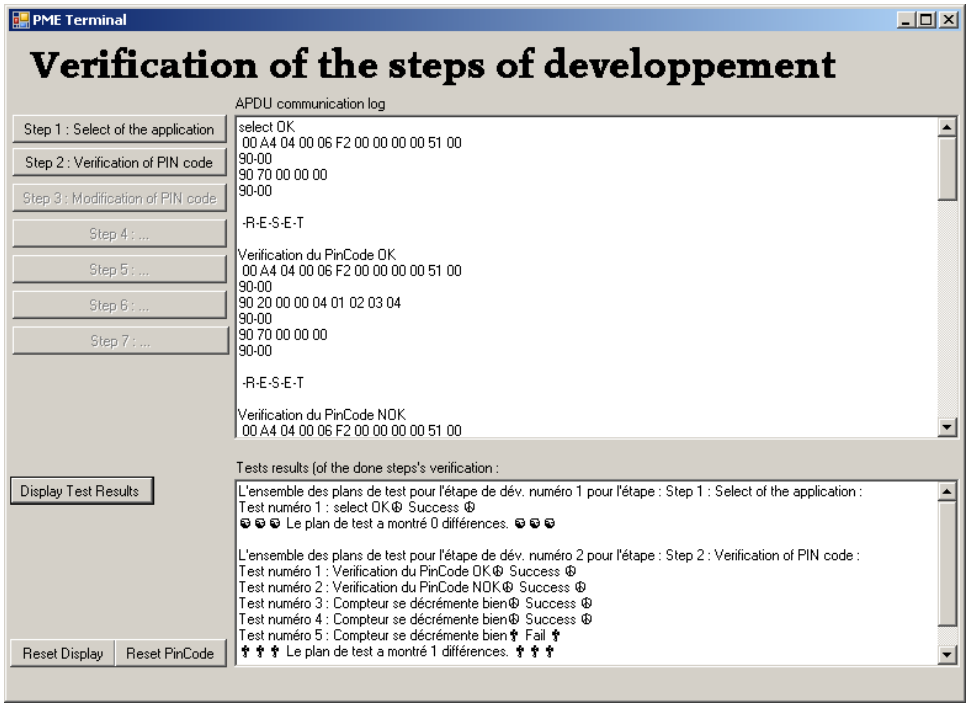

Figure 3: Outil pour effectuer le test

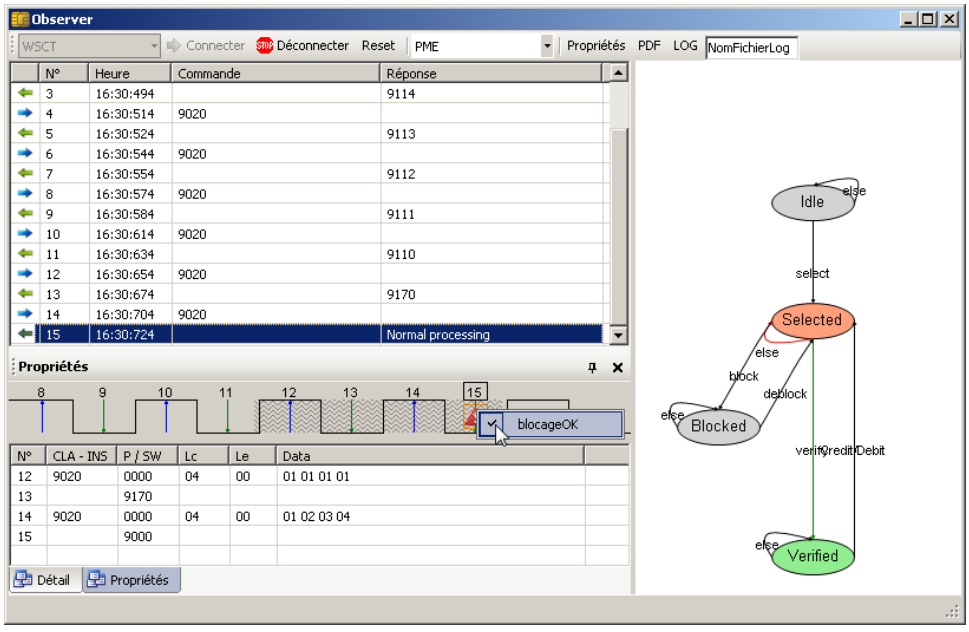

Figure 4: Outil de détection d'anomalie

connaitre l'endroit exact de l'anomalie (une description pouvant être ajoutée chaque propriété). Cependant, concernant le cas D, on obtient une détection supplémentaire. En effet, ce cas n'étant pas désiré, il est très peu probable et donc non géré par le test. Il s'agit ici d'une propriété que l'on pourrait caractériser de propriété de contrôle : elle va vérifier la bonne validité du code PIN si la commande VERIFY est acceptée.

4.7 Discussion

Cette méthodologie d'enseignement renforce une rigueur dans le développement JavaCard. De plus, nous pouvons la caractériser de méthode modulaire, simple et accessible. Son ouverture sur le monde de la recherche et son lien avec la pédagogie de sécurité logicielle ne font que renforcer l'intérêt de cette méthodologie. Cependant, cette technique peut s'avérer longue à en place, surtout la première année. Cependant, si l'enseignant a les compétences de représenter le programme (ce qui est normalement le cas), il n'est pas difficile de prendre en main l'outil et de le configurer. Le lien entre développement et évaluation est vue de manière concrète et permet de faciliter et d'illustrer la méthode de notation des TPs auprès des élèves (la relation serait comparable à la relation entre le constructeur et le laboratoire qui doit noter l'application).

5 CONCLUSIONS

Ce document résume des travaux de recherche ayant été utilisé dans le cadre de l'enseignement du développement d'applications JavaCard. Il permet d'appréhender l'enseignement de développement en le couplant à l'enseignement de la sécurité logicielle et plus particulièrement l'évaluation logicielle (durant la phase de développement pour les élèves et durant la phase de notation pour l'enseignant). Ces enseignements ne peuvent pas être dissocier vu l'ampleur de l'utilisation des applications JavaCard pour de multiples usages. Grâce aux travaux de recherche et au framework existant dans notre laboratoire, nous avons pu mettre en place un outils (tiré directement de la recherche académique) à vocation pédagogique.

Les améliorations possibles seraient de compresser ces outils en un seul outil plus accessible et utilisable par un non initié mais aussi de catégoriser les propriétés (selon leur rôle et leur priorité) afin d'affiner le paramétrage de l'outil.

REFERENCES

Aarts, F., De Ruiter, J., and Poll, E. (2013). Formal models of bank cards for free. In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, pages 461–468. IEEE.

Ahrendt, W., Baar, T., Beckert, B., Bubel, R., Giese, M., Hähnle, R., Menzel, W., Mostowski, W., Roth, A., Schlager, S., et al. (2005). The key tool. *Software & Systems Modeling*, 4(1):32–54.

Alimi, V., Vernois, S., and Rosenberger, C. (2014). Analysis of embedded applications by evolutionary fuzzing. In *High Performance Computing & Simulation (HPCS), 2014 International Conference on*, pages 551–557. IEEE.

Bekrar, S., Bekrar, C., Groz, R., and Mounier, L. (2012). A taint based approach for smart fuzzing. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 818–825. IEEE.

Bkakria, A., Bouffard, G., Iguchi-Cartigny, J., and Lanet, J.-L. (2011). Opal: an open-source global platform java library which includes the remote application management over http. In *e-Smart 2011*.

CardContact (2012). http://www.openscdp.org/.

Distefano, D. and Parkinson J, M. J. (2008). jstar: Towards practical verification for java. In *ACM Sigplan Notices*, volume 43, pages 213–226. ACM.

EMVCo (2013).

for Standardization, I. O. (1994). *ISO 8402: 1994: Quality Management and Quality Assurance-Vocabulary*. International Organization for Standardization.

Haneberg, D., Grandy, H., Reif, W., and Schellhorn, G. (2007). Verifying smart card applications: an asm approach. In *Integrated Formal Methods*, pages 313–332. Springer.

Jacobs, B., Marché, C., and Rauch, N. (2004). Formal verification of a commercial smart card applet with multiple tools. In *Algebraic Methodology And Software Technology*, pages 241–257. Springer.

Jolly, G., Vernois, S., and Lambert, J.-L. (2014). Improving test conformance of smart cards versus emv-specification by using on the fly temporal property verification. In *Recent Trends in Computer Networks and Distributed Systems Security*, pages 192–201. Springer.

Lancia, J. (2011). Un framework de fuzzing pour cartes à puce: application aux protocoles emv. In *Symposium sur la Sécurité des Technologies de lInformation et des Communications (SSTIC)*, page 82.

Lanet, J.-L. and Requet, A. (2000). Formal proof of smart card applets correctness. In *Smart Card Research and Applications*, pages 85–97. Springer.

Mueller, D. (2012). https://code.google.com/p/pcsc-sharp/.

Pannetrat, A. (2010). https://code.google.com/p/cardpeek/.

Philippaerts, P., Mühlberg, J. T., Penninckx, W., Smans, J., Jacobs, B., and Piessens, F. (2014). Software verification with verifast: Industrial case studies. *Science of Computer Programming*, 82:77–97.

Philipps, J., Pretschner, A., Slotosch, O., Aiglstorfer, E., Kriebel, S., and Scholl, K. (2003). Model-based test case generation for smart cards. *Electronic Notes in Theoretical Computer Science*, 80:170–184.

Posegga, J. and Vogt, H. (1998). Byte code verification for java smart cards based on model checking. In *Computer SecurityESORICS 98*, pages 175–190. Springer.

Radatz, J., Geraci, A., and Katki, F. (1990). Ieee standard glossary of software engineering terminology. *IEEE Std*, 610121990:121990.

Rankl, W. (2007). *Smart Card Applications: Design Models for Using and Programming Smart Cards*. Wiley Online Library.

Rankl, W. and Effing, W. (2010). *Smart card handbook*. John Wiley & Sons.

Rouit, O. (2011). http://www.codeproject.com/articles/17013/smart-card-framework-for-net.

Sabatier, D. and Lartigue, P. (1999). The use of the b formal method for the design and the validation of the transaction mechanism for smart card applications. In *FM99Formal Methods*, pages 348–368. Springer.

Sasc (2014). https://github.com/sasc999/javaemvreader.

van Weelden, A., Oostdijk, M., Frantzen, L., Koopman, P., and Tretmans, J. (2005). On-the-fly formal testing of a smart card applet. In *Security and Privacy in the Age of Ubiquitous Computing*, pages 565–576. Springer.

Vernois, S. and Alimi, V. (2010). Winscard tools: a software for the development and security analysis of transactions with smartcards. *Norsk informasjonssikkerhetskonferanse (NISK)*.

Watanabe, T., Howell, P., and Pugh, S. (2006). Easing emv: Emvco's new common payment application. *Card Technology Today*, 18(2):12–13.