



Seminar:

**Nachweis von Sicherheitseigenschaften für JavaCard
durch approximative Programmauswertung
WS 2001 / 2002**

Veranstalter: Prof. Tobias Nipkow, Martin Strecker

Vortrag:

Sichere Interaktion von SmartCard-Applets

Von Clovis Yemou

6 Dezember 2001

Überblick



Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?

Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?
- Java Card *Sicherheitmechanismus*²
 - Sicherheit der Karte(Verantwortlicher?)
 - Sicherheit der Anwendung(Verantwortlicher?)

Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?
- Java Card *Sicherheitmechanismus*²
 - Sicherheit der Karte(Verantwortlicher?)
 - Sicherheit der Anwendung(Verantwortlicher?)
 - sichere *Interaktion*³ von Applets(Verantwortlicher?)

Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?
- Java Card *Sicherheitmechanismus*²
 - Sicherheit der Karte(Verantwortlicher?)
 - Sicherheit der Anwendung(Verantwortlicher?)
 - sichere *Interaktion*³ von Applets(Verantwortlicher?)
 - * *electronic purse*³

Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?
- Java Card *Sicherheitmechanismus*²
 - Sicherheit der Karte(Verantwortlicher?)
 - Sicherheit der Anwendung(Verantwortlicher?)
 - sichere *Interaktion*³ von Applets(Verantwortlicher?)
 - * *electronic purse*³
- *Applet Certification*⁴
 - Multiapplication Security Policy
 - * electronic purse: Security *Policy*⁵, Security *Properties*⁶

Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?
- Java Card *Sicherheitmechanismus*²
 - Sicherheit der Karte(Verantwortlicher?)
 - Sicherheit der Anwendung(Verantwortlicher?)
 - sichere *Interaktion*³ von Applets(Verantwortlicher?)
 - * *electronic purse*³
- *Applet Certification*⁴
 - Multiapplication Security Policy
 - * electronic purse: Security *Policy*⁵, Security *Properties*⁶
- *Model checking*⁹ für die Fallstudie

Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?
- Java Card *Sicherheitsmechanismus*²
 - Sicherheit der Karte(Verantwortlicher?)
 - Sicherheit der Anwendung(Verantwortlicher?)
 - sichere *Interaktion*³ von Applets(Verantwortlicher?)
 - * *electronic purse*³
- *Applet Certification*⁴
 - Multiapplication Security Policy
 - * electronic purse: Security *Policy*⁵, Security *Properties*⁶
- *Model checking*⁹ für die Fallstudie
 - Mit der *Assume – guarantee*⁷ Verifikation
 - Mit dem SMV-*Model checker*: *Invariant*⁸

Definitionen



Definitionen



- **SmartCard**: Plastische Karte (einer Bank-Karte ähnlich!)

Definitionen



- **SmartCard**: Plastische Karte (einer Bank-Karte ähnlich!) mit eingebettetem
 - **memory** : Daten können nur gespeichert werden(vgl. Diskette!).

Definitionen



- **SmartCard**: Plastische Karte (einer Bank-Karte ähnlich!) mit eingebettetem
 - **memory** : Daten können nur gespeichert werden(vgl. Diskette!).
 - **microprocessor**: Daten können gespeichert, bearbeitet oder gelöscht werden.

Definitionen



- **SmartCard**: Plastische Karte (einer Bank-Karte ähnlich!) mit eingebettetem
 - **memory** : Daten können nur gespeichert werden(vgl. Diskette!).
 - **microprocessor**: Daten können gespeichert, bearbeitet oder gelöscht werden.
 - * *Mono-applicative*: Anwendung in Betriebssystem integriert

Definitionen



- **SmartCard**: Plastische Karte (einer Bank-Karte ähnlich!) mit eingebettetem
 - **memory** : Daten können nur gespeichert werden(vgl. Diskette!).
 - **microprocessor**: Daten können gespeichert, bearbeitet oder gelöscht werden.
 - * *Mono-applicative*: Anwendung in Betriebssystem integriert
 - * *Multiapplication*: Mehrere Anwendungen (Applets) können ,nach der Erstellung der Karte,nachgeladen und gleichzeitig laufen.
(**Interaktionen möglich!**)

Definitionen



- **SmartCard**: Plastische Karte (einer Bank-Karte ähnlich!) mit eingebettetem
 - **memory** : Daten können nur gespeichert werden(vgl. Diskette!).
 - **microprocessor**: Daten können gespeichert, bearbeitet oder gelöscht werden.
 - * *Mono-applicative*: Anwendung in Betriebssystem integriert
 - * *Multiapplication*: Mehrere Anwendungen (Applets) können ,nach der Erstellung der Karte,nachgeladen und gleichzeitig laufen.
(**Interaktionen möglich!**)



Abbildung 1: Multiapplication SmartCart

Definitionen



Definitionen



- **Applet**: Kompakte (java) Anwendung, die von einem fernem, computerfähigem Gerät (multiapplication Smart Card z.B.) schnell geladen und genutzt werden kann.

Definitionen



- **Applet**: Kompakte (java) Anwendung, die von einem fernem, computerfähigem Gerät (multiapplication Smart Card z.B.) schnell geladen und genutzt werden kann.
- **Java Card**(virtuelles Betriebssystem) : Menge von Spezifikationen, um java auf einem (java) SmartCard zum Laufen zu bringen

Definitionen



- **Applet**: Kompakte (java) Anwendung, die von einem fernem, computerfähigem Gerät (multiapplication Smart Card z.B.) schnell geladen und genutzt werden kann.
- **Java Card**(virtuelles Betriebssystem) : Menge von Spezifikationen, um java auf einem (java) SmartCard zum Laufen zu bringen
- Der **Application provider** bietet Anwendungen an, mit denen der **Card provider(issuer)** smart cards erstellen kann und den **Card holder(users)**(Benutzern) anbietet.

Definitionen



- **Applet**: Kompakte (java) Anwendung, die von einem fernem, computerfähigem Gerät (multiapplication Smart Card z.B.) schnell geladen und genutzt werden kann.
- **Java Card**(virtuelles Betriebssystem) : Menge von Spezifikationen, um java auf einem (java) SmartCard zum Laufen zu bringen
- Der **Application provider** bietet Anwendungen an, mit denen der **Card provider(issuer)** smart cards erstellen kann und den **Card holder(users)**(Benutzern) anbietet.
Die Trennung OS, VM und applikative Code

Definitionen



- **Applet**: Kompakte (java) Anwendung, die von einem fernem, computerfähigem Gerät (multiapplication Smart Card z.B.) schnell geladen und genutzt werden kann.
- **Java Card**(virtuelles Betriebssystem) : Menge von Spezifikationen, um java auf einem (java) SmartCard zum Laufen zu bringen
- Der **Application provider** bietet Anwendungen an, mit denen der **Card provider(issuer)** smart cards erstellen kann und den **Card holder(users)**(Benutzern) anbietet.
Die Trennung OS, VM und applikative Code führt zur Frage des **Sicherheitsbedarfs!** (zum Überblick).

Sicherheitsmechanismus



Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)

Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**

Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**
- Frage der Sicherheit der Anwendung (Applet) und damit der Karte
 - **key confidentiality**
 - **Schützt** vor aggressiven Applets
 - **Integrität** von bestimmten Datenfeldern...

Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**
- Frage der Sicherheit der Anwendung (Applet) und damit der Karte
 - **key confidentiality**
 - **Schützt** vor aggressiven Applets
 - **Integrität** von bestimmten Datenfeldern...Der (Applet) **application Provider**

Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**
- Frage der Sicherheit der Anwendung (Applet) und damit der Karte
 - **key confidentiality**
 - **Schützt** vor aggressiven Applets
 - **Integrität** von bestimmten Datenfeldern...
Der (Applet) **application Provider**
- Frage des privacys(:?)

Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**
- Frage der Sicherheit der Anwendung (Applet) und damit der Karte
 - **key confidentiality**
 - **Schützt** vor aggressiven Applets
 - **Integrität** von bestimmten Datenfeldern...
Der (Applet) **application Provider**
- Frage des privacys(:?)
Der end **user**

Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**
- Frage der Sicherheit der Anwendung (Applet) und damit der Karte
 - **key confidentiality**
 - **Schützt** vor aggressiven Applets
 - **Integrität** von bestimmten Datenfeldern...
Der (Applet) **application Provider**
- Frage des privacys(:?)
Der end **user**
- Frage der Kontrolle des Informationsflusses

Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**
- Frage der Sicherheit der Anwendung (Applet) und damit der Karte
 - **key confidentiality**
 - **Schützt** vor aggressiven Applets
 - **Integrität** von bestimmten Datenfeldern...
Der (Applet) **application Provider**
- Frage des privacys (:?)
Der end **user**
- Frage der Kontrolle des Informationsflusses
Der end **Card issuer**

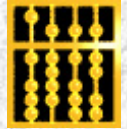
Sicherheitsmechanismus



- Frage der Sicherheit der Karte (HW, OS und VM)
Der **Card provider**
- Frage der Sicherheit der Anwendung (Applet) und damit der Karte
 - **key confidentiality**
 - **Schützt** vor aggressiven Applets
 - **Integrität** von bestimmten Datenfeldern...
Der (Applet) **application Provider**
- Frage des privacys(:?)
Der end **user**
- Frage der Kontrolle des Informationsflusses
Der end **Card issuer**
Java Card Sicherheitsmechanismus verbietet zwar einem geladenen Applet Informationen und Ressourcen unerlaubt zu beobachten, verändern, benutzen
aber was ist mit Informationsfluß bei **Interaktion** zwischen geladenen Applets?

(zum **Überblick**).

Electronic Purse



Electronic Purse



- Wurde von Gemplus implementiert.
- 1 purse Applet + 2 loyalty Applets (Air France+RentaCar).
- Verwaltet Soll und Haben, Konvertierung(z.B francs+euro) und Log von allen Transaktionen.
- Ein Loyalty(Applet) wird dem Benutzerwünsch nach auf die Karte nachgeladen, die Punkte (z.B. Meilen fürs Air France Loyalität-Programm) je nach Einkauf gewährt.
⇒ Wechselwirkungsbedarf zwischen Applets!

Electronic Purse



- Wurde von Gemplus implementiert.
- 1 purse Applet + 2 loyalty Applets (Air France+RentaCar).
- Verwaltet Soll und Haben, Konvertierung(z.B francs+euro) und Log von allen Transaktionen.
- Ein Loyalty(Applet) wird dem Benutzerwünsch nach auf die Karte nachgeladen, die Punkte (z.B. Meilen fürs Air France Loyalität-Programm) je nach Einkauf gewährt.
⇒ Wechselwirkungsbedarf zwischen Applets!

Wechselwirkung:

- **purse-loyalty!**
↔ purse-applet bietet den loyalty-applets einen „logfull“-Dienst an!
- **loyalty-loyalty** auch vorgesehen!
↔(z.B. Air France Flugticket = RentaCar Punkte + Air France Meilen)!

(zum Überblick).

Electronic Purse : Applets-Interaktionen



Wie wird es verhandelt?

Electronic Purse : Applets-Interaktionen



Wie wird es verhandelt?

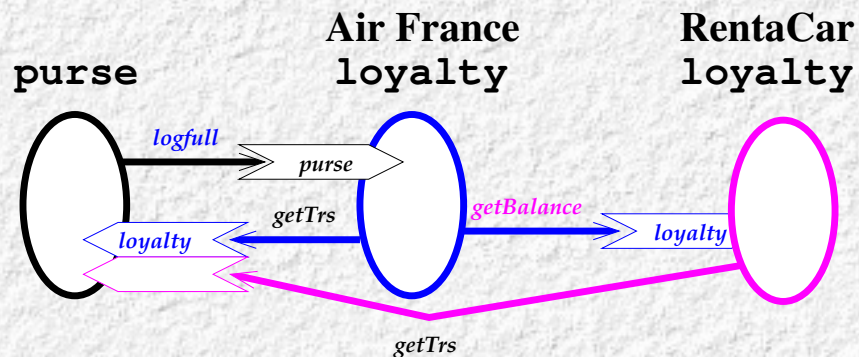


Abbildung 2: Applets Interaktion

Electronic Purse : Applets-Interaktionen



Wie wird es verhandelt?

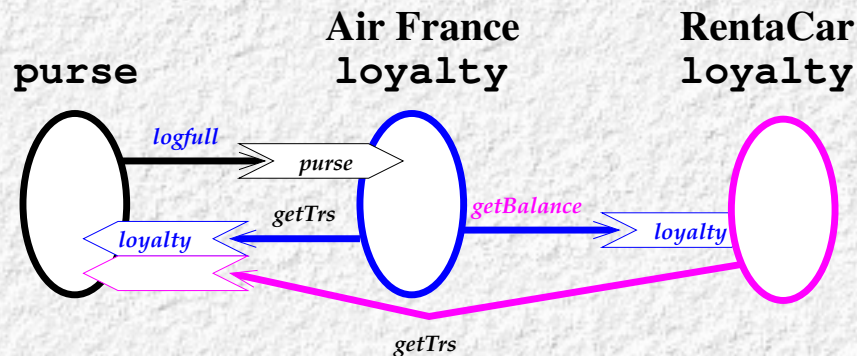


Abbildung 2: Applets Interaktion

Die Methode *logfull* vom Air France-Applet wird vom Purse-Applet gerufen.
↔ Das Air France-Applet triebt mit *getTrs* vom Purse-Applet seine Punkte ein.

Electronic Purse : Applets-Interaktionen



Wie wird es verhandelt?

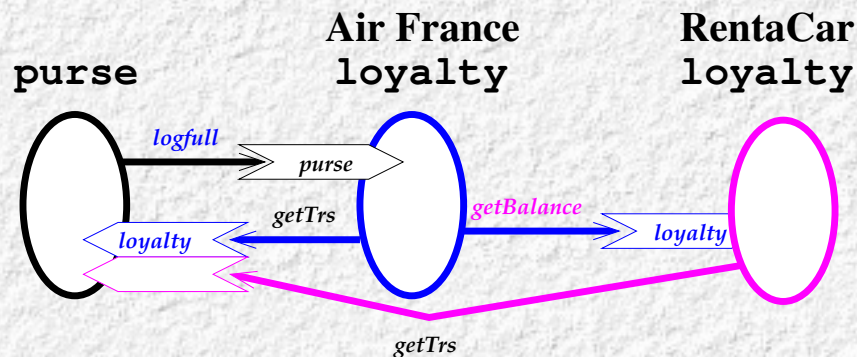


Abbildung 2: Applets Interaktion

- Die Methode *logfull* vom Air France-Applet wird vom Purse-Applet gerufen.
- ↪ Das Air France-Applet treibt mit *getTrs* vom Purse-Applet seine Punkte ein.
- ↪ Das Air France-Applet will aber seine Punkte mit *getBalance* vom RentaCar-Applet (Partner!) eintreiben!
- ↪ Das RentaCar-Applet treibt mit *getTrs* vom Purse-Applet seine Punkte ein.

Electronic Purse : Applets-Interaktionen



Wie wird es verhandelt?

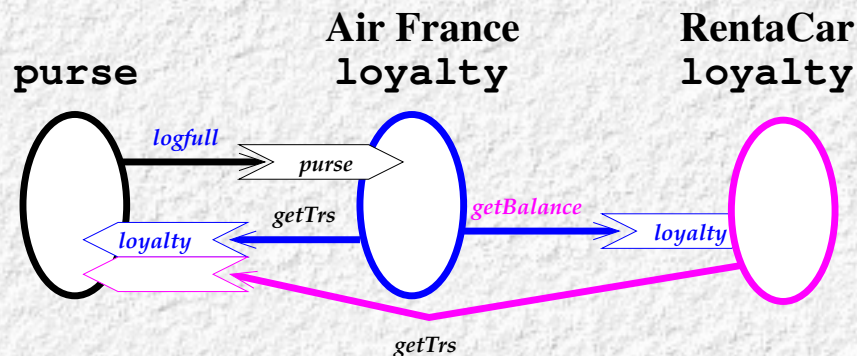


Abbildung 2: Applets Interaktion

- Die Methode *logfull* vom Air France-Applet wird vom Purse-Applet gerufen.
- ↳ Das Air France-Applet treibt mit *getTrs* vom Purse-Applet seine Punkte ein.
- ↳ Das Air France-Applet will aber seine Punkte mit *getBalance* vom RentaCar-Applet (Partner!) eintreiben!
- ↳ Das RentaCar-Applet treibt mit *getTrs* vom Purse-Applet seine Punkte ein.

!: Ein Haufen bestimmter vertraulichen Informationen fließen (unerlaubt!) vom Air France-Applet zum RentaCar-Applet (Partner!):-((((zum Überblick).

Applet certification



Nur erlaubte Informationsflüsse gewährleisten: Wie kann Der Card issuer
unerlaubte Informationsflüsse kontrollieren?

Applet certification



Nur erlaubte Informationsflüsse gewährleisten: Wie kann Der Card issuer unerlaubte Informationsflüsse kontrollieren?

Ziel: Techniken und tools finden, die es erlauben, solche unerlaubte Informationsflüsse aufspüren zu können, um sicherzustellen, daß es nur erlaubte Informationsflüsse zwischen neuen Applets und schon geladenen Applets **tatsächlich** existieren!

Applet certification



Nur erlaubte Informationsflüsse gewährleisten: Wie kann Der Card issuer **unerlaubte Informationsflüsse** kontrollieren?

Ziel: **Techniken** und **tools** finden, die es erlauben, solche unerlaubte Informationsflüsse aufspüren zu können, um sicherzustellen, daß es nur **erlaubte Informationsflüsse** zwischen neuen Applets und schon geladenen Applets **tatsächlich** existieren!

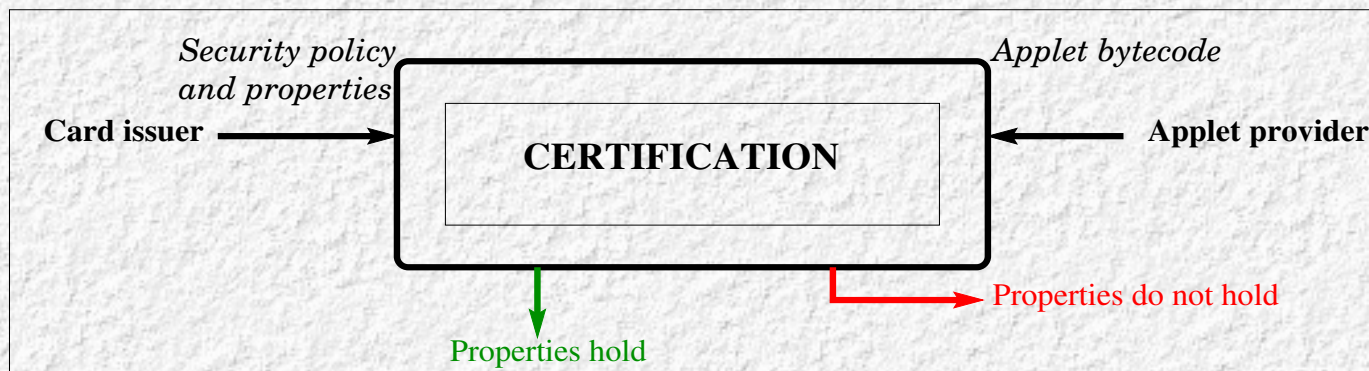


Abbildung 3: Applet Certification

(zum Überblick).

Policy



Eine Politik festlegen:

Policy



Eine Politik festlegen:

- Programme: $\{Purse\} \cup \underbrace{\{AirFrance, RentaCar\}}_{loyalties}$
- Stufen: $\{P, AF, RC\} \cup \underbrace{\{P + AF, P + RC, AF + RC\}}_{shared} = L$

Policy



Eine Politik festlegen:

- Programme: $\{Purse\} \cup \underbrace{\{AirFrance, RentaCar\}}_{loyalties}$
- Stufen: $\{P, AF, RC\} \cup \underbrace{\{P + AF, P + RC, AF + RC\}}_{shared} = L$
- Policy:
 $P + AF \preceq P$, $P + AF \preceq AF$
 $P + RC \preceq P$, $P + RC \preceq RC$
 $AF + RC \preceq RC$, $AF + RC \preceq AF$
d.h. (L, \preceq) hat eine "lattice" Struktur:
Obere Stufen *private* und untere Stufen *public*!

Policy



Eine Politik festlegen:

- Programme: $\{Purse\} \cup \underbrace{\{AirFrance, RentaCar\}}_{loyalties}$
- Stufen: $\{P, AF, RC\} \cup \underbrace{\{P + AF, P + RC, AF + RC\}}_{shared} = L$
- Policy:
 $P+AF \preceq P$, $P+AF \preceq AF$
 $P+RC \preceq P$, $P+RC \preceq RC$
 $AF+RC \preceq RC$, $AF+RC \preceq AF$
d.h. (L, \preceq) hat eine "lattice" Struktur:
Obere Stufen *private* und untere Stufen *public*!

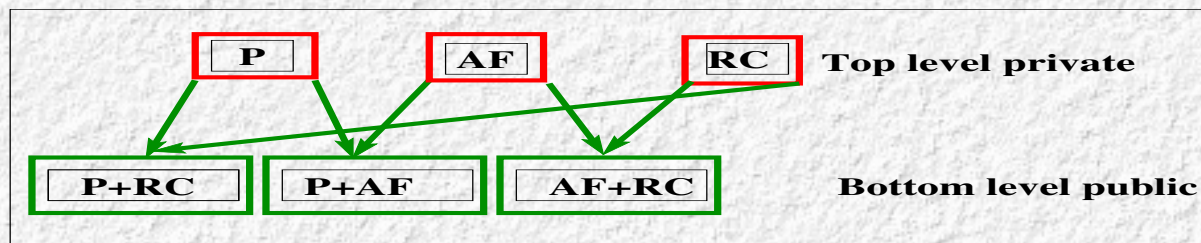


Abbildung 4: lattice-Struktur

properties



Eigenschaften definieren:

properties



Eigenschaften definieren:

„*Secure dependency model*“: Abhängigkeiten zwischen System Objekte können nicht ausgenutzt werden, um indirekte Kanäle zu gründen!

- Wir definieren:
P \equiv Menge aller Programme (purse+loyalties)
L \equiv $\{L_1, L_2, \dots, L_n\}$ zugeordnete Stufen
V \equiv Menge aller System-Variablen

properties



Eigenschaften definieren:

„*Secure dependency model*“: Abhängigkeiten zwischen System Objekte können nicht ausgenutzt werden, um indirekte Kanäle zu gründen!

- Wir definieren:

$P \equiv$ Menge aller Programme (purse+loyalties)

$L \equiv \{L_1, L_2, \dots, L_n\}$ zugeordnete Stufen

$V \equiv$ Menge aller System-Variablen

– Sei nun $p \in P$ und $l \in L$. Wir definieren:

* $v_l^{aus}(p) \equiv$ Ausgabe-Variable aus P mit der Stufe l

* $v_l^{ein}(p) \equiv$ Eingabe-Variable aus P mit der Stufe l

* $v^{int} \equiv$ Interne System-Variable (*Stufe?*)

properties



Eigenschaften definieren:

„*Secure dependency model*“: Abhängigkeiten zwischen System Objekte können nicht ausgenutzt werden, um indirekte Kanäle zu gründen!

- Wir definieren:
 - $P \equiv$ Menge aller Programme (purse+loyalties)
 - $L \equiv \{L_1, L_2, \dots, L_n\}$ zugeordnete Stufen
 - $V \equiv$ Menge aller System-Variablen
 - Sei nun $p \in P$ und $l \in L$. Wir definieren:
 - * $v_l^{aus}(p) \equiv$ Ausgabe-Variable aus P mit der Stufe l
 - * $v_l^{ein}(p) \equiv$ Eingabe-Variable aus P mit der Stufe l
 - * $v^{int} \equiv$ Interne System-Variable (*Stufe?*)
- Eigenschaft: Sei $v_l^{aus}(p) \in V, v_1, \dots, v_k \in V$ und $I = \{1, \dots, k\}$

properties



Eigenschaften definieren:

„Secure dependency model“: Abhängigkeiten zwischen System Objekte können nicht ausgenutzt werden, um indirekte Kanäle zu gründen!

- Wir definieren:
 - $P \equiv$ Menge aller Programme (purse+loyalties)
 - $L \equiv \{L_1, L_2, \dots, L_n\}$ zugeordnete Stufen
 - $V \equiv$ Menge aller System-Variablen
- Sei nun $p \in P$ und $l \in L$. Wir definieren:
 - * $v_l^{aus}(p) \equiv$ Ausgabe-Variable aus P mit der Stufe l
 - * $v_l^{ein}(p) \equiv$ Eingabe-Variable aus P mit der Stufe l
 - * $v^{int} \equiv$ Interne System-Variable (*Stufe?*)
- Eigenschaft: Sei $v_l^{aus}(p) \in V, v_1, \dots, v_k \in V$ und $I = \{1, \dots, k\}$

$$v_l^{aus}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k)) \Rightarrow \forall i \in I, l_i \leq l. \quad (1)$$

! :Es ist nicht aber immer möglich allen Variablen (v^{int}) eine Stufe zuzuordnen!
(zum Überblick).

Computed level



Internen Variablen können nicht immer eine feste Stufe vergeben werden!

Computed level



Internen Variablen können nicht immer eine feste Stufe vergeben werden!

Wir definieren:

cl \equiv computed level(abgeschätzte Stufe).

Computed level



Internen Variablen können nicht immer eine feste Stufe vergeben werden!

Wir definieren:

cl \equiv computed level (abgeschätzte Stufe).

- $\forall v_l^{ein}(p) \in V, cl(v_l^{ein}(p)) = l.$

Computed level



Internen Variablen können nicht immer eine feste Stufe vergeben werden!

Wir definieren:

cl \equiv computed level (abgeschätzte Stufe).

- $\forall v_l^{ein}(p) \in V, cl(v_l^{ein}(p)) = l.$
- $\forall v_l^{int}(p) \in V, cl(v_l^{int}(p)) = \sup((l_i)_{1 \leq i \leq k}),$ mit $v_l^{int}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k)).$

Computed level



Internen Variablen können nicht immer eine feste Stufe vergeben werden!

Wir definieren:

cl \equiv computed level (abgeschätzte Stufe).

- $\forall v_l^{ein}(p) \in V, cl(v_l^{ein}(p)) = l.$
- $\forall v_l^{int}(p) \in V, cl(v_l^{int}(p)) = \sup((l_i)_{1 \leq i \leq k}),$ mit $v_l^{int}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k)).$
- $\forall v_l^{aus}(p) \in V, cl(v_l^{aus}(p)) = \sup(l_i)_{1 \leq i \leq k},$ mit $v_l^{aus}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k))$

Computed level



Internen Variablen können nicht immer eine feste Stufe vergeben werden!

Wir definieren:

$cl \equiv$ computed level (abgeschätzte Stufe).

- $\forall v_l^{ein}(p) \in V, cl(v_l^{ein}(p)) = l.$
- $\forall v_l^{int}(p) \in V, cl(v_l^{int}(p)) = \sup((l_i)_{1 \leq i \leq k}),$ mit $v_l^{int}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k)).$
- $\forall v_l^{aus}(p) \in V, cl(v_l^{ein}(p)) = \sup(l_i)_{1 \leq i \leq k},$ mit $v_l^{aus}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k))$
- **Erweiterte Eigenschaft:**
Sei $v_l^{aus}(p) \in V, v_1, \dots, v_k \in V.$
 $v_l^{aus}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k)) \Rightarrow \sup(l_i)_{1 \leq i \leq k} = cl(v_l^{aus}(p)) \leq l.$

Computed level



Internen Variablen können nicht immer eine feste Stufe vergeben werden!

Wir definieren:

cl \equiv computed level (abgeschätzte Stufe).

- $\forall v_l^{ein}(p) \in V, cl(v_l^{ein}(p)) = l.$
- $\forall v_l^{int}(p) \in V, cl(v_l^{int}(p)) = \sup((l_i)_{1 \leq i \leq k}),$ mit $v_l^{int}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k)).$
- $\forall v_l^{aus}(p) \in V, cl(v_l^{aus}(p)) = \sup(l_i)_{1 \leq i \leq k},$ mit $v_l^{aus}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k))$
- **Erweiterte Eigenschaft:**
Sei $v_l^{aus}(p) \in V, v_1, \dots, v_k \in V.$
 $v_l^{aus}(p) = f(v_{l_1}(p_1), v_{l_2}(p_2), \dots, v_{l_k}(p_k)) \Rightarrow \sup(l_i)_{1 \leq i \leq k} = cl(v_l^{aus}(p)) \leq l.$

D.h.

Die abgeschätzte Stufe einer Output-Variable (obere Schranke der Stufen der Variablen von denen sie syntaktisch abhängig ist) ist **höchstens** ihre Sicherheitsstufe.

Globale Analyse



Wie gehen wir vor?

Globale Analyse



Wie gehen wir vor?

- Ein Applet (*)
- Alle Attribute von einem Applet
 - Eine Methode von einem Applet(Stufe für ihre Invokation)
 - * Kontroll der Stufe des Ergebnisses
 - Alle Methoden in einem Applet
- Alle Methoden in allen Applets auf der Karte.

Globale Analyse



Wie gehen wir vor?

- Ein Applet (*)
- Alle Attribute von einem Applet
 - Eine Methode von einem Applet (Stufe für ihre Invokation)
 - * Kontroll der Stufe des Ergebnisses
 - Alle Methoden in einem Applet
- Alle Methoden in allen Applets auf der Karte.

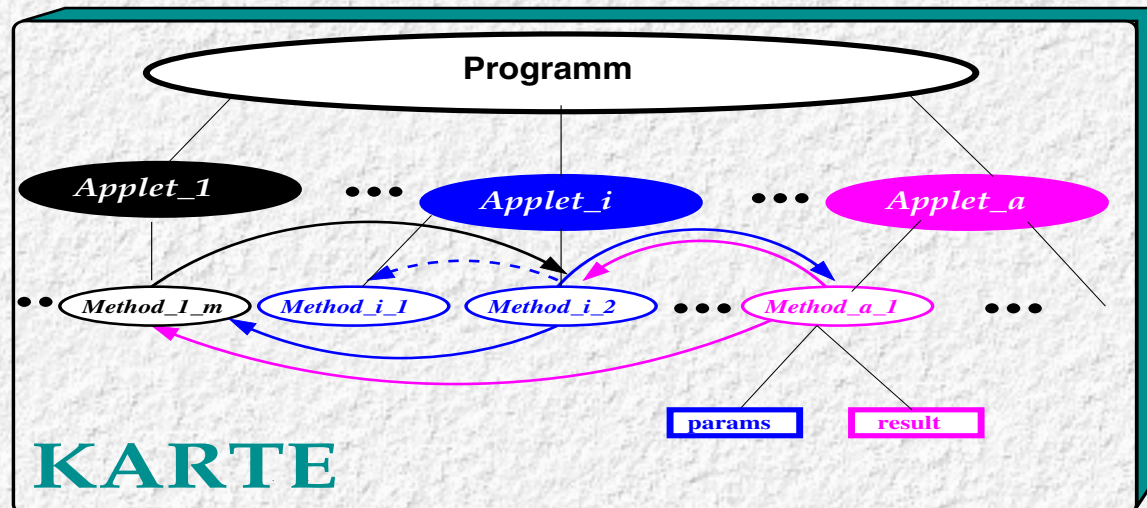


Abbildung 5: Globale Analyse-Technik

Assume-guarantee



Verifikation für die Methode *getBalance* (aus *logfull*)



Assume-guarantee

Verifikation für die Methode *getBalance* (aus *logfull*)

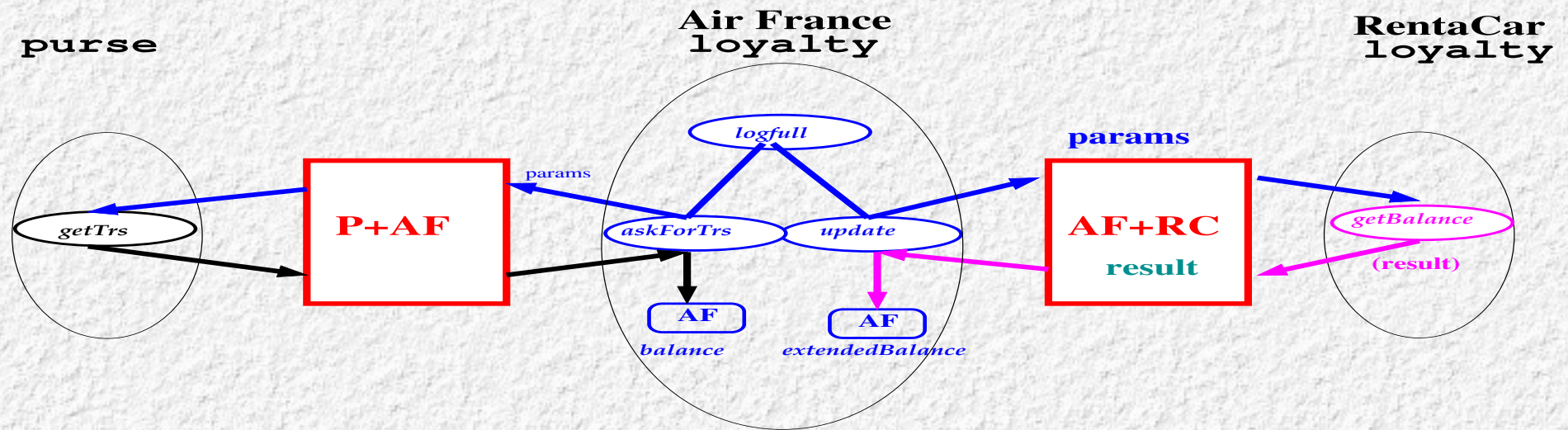


Abbildung 6: Assume-Guarantee Verifikation



Assume-guarantee

Verifikation für die Methode *getBalance* (aus *logfull*)

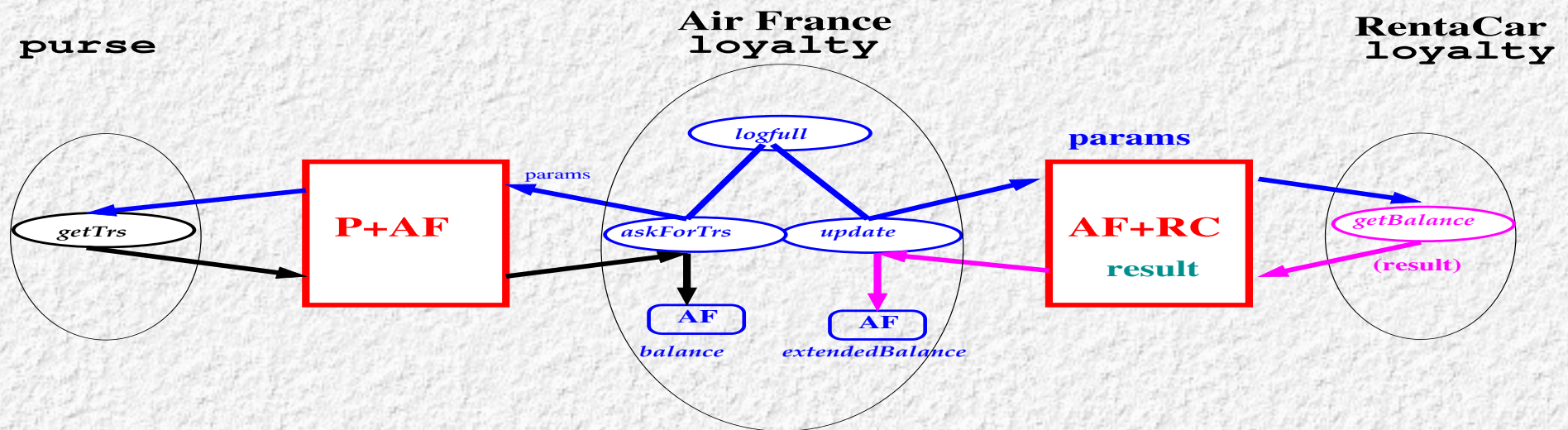


Abbildung 6: Assume-Guarantee Verifikation

logfull ruft *update*, die *getbalance* ruft.

Annahme: *result* hat die Stufe $AF + RC$.

Wir kontrollieren bei der Analyse, die Stufe von den Parameter von *getbalance* $params \leq result$ (eigentlich **result** mit **result**) die Stufe von dem Ergebnis von der Interaktion ($AF + RC$).

Gleiche Regel für das Lesen eines Attributs innerhalb einem Applet (zum Überblick).

Byte-Code



Lokale Verifikation der Eigenschaft (2) auf den Java Byte-Coden

Byte-Code



Lokale Verifikation der Eigenschaft (2) auf den Java Byte-Coden
vereinfachte Version von logfull: *logfull* ruf die Methode *getbalance* von
RentaCar-applet und aktualisiert das Attribut ExtendedBalance

Byte-Code



Lokale Verifikation der Eigenschaft (2) auf den Java Byte-Coden vereinfachte Version von logfull: *logfull* ruf die Methode *getbalance* von RentaCar-applet und aktualisiert das Attribut *ExtendedBalance*

Byte-Code *Methodvoidlogfull()*

0 *aload_0*

1 *invokespecial_108*<*Methodintgetbalance()*>

4 *aload_1*

5 *aload_0*

6 *dup*

7 *getfield_220* <*FieldintExtendedBalance*>

10 *iload_1*

11 *iadd*

12 *putfield_220* <*FieldintExtendedBalance*>

15 *return*

Abstraktion



Abstraktion



L : levels; *pc* : $-1..9$; *mem* : array 0..1 of boolean; *stck* : array 0..1 of boolean; *sP* : $-1..1$;
ByteCode : {*invoke_108*, *load_0*, *return*, *nop*, *store_1*, *dup*, *load_1*, *getfield_220*, *op*, *putfield_220* };

Abstraktion



L : levels; *pc* : $-1..9$; *mem* : array 0..1 of boolean; *stck* : array 0..1 of boolean; *sP* : $-1..1$;
ByteCode : {*invoke_108*, *load_0*, *return*, *nop*, *store_1*, *dup*, *load_1*, *getfield_220*, *op*, *putfield_220* };
init(*pc*) := 0; *init*(*sP*) := 0; *init*(*mem*[0]) := *L.AF* & *L.P*;
for(*i* = 0; *i* < 2; *i* = *i* + 1) {*init*(*stck*[*i*]) := *L.AF* & *L.P*;}

Abstraktion



L : levels; pc : $-1..9$; mem : array $0..1$ of boolean; $stck$: array $0..1$ of boolean; sP : $-1..1$;

ByteCode : {*invoke_108*, *load_0*, *return*, *nop*, *store_1*, *dup*, *load_1*, *getfield_220*, *op*, *putfield_220* };

init(pc) := 0; *init*(sP) := 0; *init*($mem[0]$) := $L.AF \ \& \ L.P$;

for($i = 0$; $i < 2$; $i = i + 1$) {*init*($stck[i]$) := $L.AF \ \& \ L.P$ };

switch(*ByteCode*) {

nop ::

load_0: {*next*($stck[sP]$) := $mem[0]$; *next*(sP) := $sP - 1$ };

load_1: {*next*($stck[sP]$) := $mem[1]$; *next*(sP) := $sP - 1$; }

store_1: {*next*($mem[1]$) := $stck[sP + 1]$; *next*(sP) := $sP + 1$ };

dup_0: {*next*($stck[sP]$) := $stck[sP + 1]$; *next*(sP) := $sP - 1$ };

op_0: {*next*($stck[sP + 2]$) := $stck[sP + 1] \mid stck[sP + 2]$; *next*(sP) := $sP + 1$ };

invoke_108 : {*next*($stck[sP]$) := $L.AF \ \& \ L.P$; *next*(sP) := $sP + 1$ };

getfield_220 : {*next*($stck[sP]$) := $L.AF$; }

putfield_220 : {*next*(sP) := $sP + 2$; }

return ::}

Invariant



Invariant



Abb.: bildliche Darstellung des Invariants.

Invariant:

Smethod_108 :

assert G (ByteCode = invoke_108 ->stck[sP + 1]->L.AF & L.RC));

Sfield_220 :

assert G (ByteCode = putfield_220 ->stck[sP + 1]->L.AF));

Sresult :

assert G (ByteCode = return->stck[sP + 1]->L.AF & L.P));

(zum Überblick).

Model checking



Model checking



Erfüllt das abstrahierte Modell die festgelegte Eigenschaft (Invariant)?

Model checking



Erfüllt das abstrahierte Modell die festgelegte Eigenschaft (Invariant)?

Verifikation mit dem SMV-*Model checker* : wenn die Eigenschaft nicht erfüllt werden kann, generiert der *Model checker* ein Gegen-Beispiel, deren Ausführung gegen die Regel verstößt!

Der SMV-*Model checker* wird mit Sicherheit, bei der Verifikation der Eigenschaft *Smethod_108*, ein **Sicherheitsproblem** aufspüren:

Die Interaktion *logfull* zwischen *purse* und *Air France* hat die Stufe $P + AF$, der Kanal *getBalance* die Stufe $AF + RC$.

Übersichtlich hängt die Invokation der Methode *getBalance* von der Invokation der Methode *logfull* ab.(fig)

Es entsteht dann eine unerlaubte Abhängigkeit einer Variable von Stufe $P + AF$ nach einer Variable der Stufe $AF + RC$.

Model checking



Erfüllt das abstrahierte Modell die festgelegte Eigenschaft (Invariant)?

Verifikation mit dem SMV-*Model checker* : wenn die Eigenschaft nicht erfüllt werden kann, generiert der *Model checker* ein Gegen-Beispiel, deren Ausführung gegen die Regel verstößt!

Der SMV-*Model checker* wird mit Sicherheit, bei der Verifikation der Eigenschaft *Smethod_108*, eine **Sicherheitsproblem** aufspüren:

Die Interaktion *logfull* zwischen *purse* und *Air France* hat die Stufe $P + AF$, der Kanal *getBalance* die Stufe $AF + RC$.

Übersichtlich hängt die Invokation der Methode *getBalance* von der Invokation der Methode *logfull* ab.(fig)

Es entsteht dann eine unerlaubte Abhängigkeit einer Variable von Stufe $P + AF$ nach einer Variable der Stufe $AF + RC$.

Mögliche **Lösung**: Verbot des Aufrufs von Methoden(*getBalance*) anderer *Loyalty* bei der Ausführung von der Methode *logfull*.

(zum **Überblick**).

Vielen Dank!



Auf Wunsch geht es zurück zum **Überblick**.

Überblick



- *Definitionen*¹
 - was ist ein smart card, ein Applet, Java Card?
 - wer ist der Application provider, der Card provider, der Card holder?
- Java Card *Sicherheitsmechanismus*²
 - Sicherheit der Karte(Verantwortlicher?)
 - Sicherheit der Anwendung(Verantwortlicher?)
 - sichere *Interaktion*³ von Applets(Verantwortlicher?)
 - * *electronic purse*³
- *Applet Certification*⁴
 - Multiapplication Security Policy
 - * electronic purse: Security *Policy*⁵, Security *Properties*⁶
- *Model checking*⁹ für die Fallstudie
 - Mit der *Assume – guarantee*⁷ Verifikation
 - Mit dem SMV-Model checker: *Invariant*⁸

