Kritik der Protocol Composition Logic(PCL)

Seminar Sicherheit und Zuverlässigkeit
WS 10/11

Jules Rasetaharison

19. Januar 2011

Übersicht

1. Einführung Protocol Composition Logic
2. Probleme in der der Protocol Composition Logic
3. Zusammenfassung

- Definition von Termen als Basis für Sicherheitsprotokolle.
- Als Term werden alle im Protokoll vorkommenden Primitiven sowie aus Einzeltermen zusammengesetzte Terme bezeichnet
- Die Primitiven umfassen die in Sicherheitsprotokollen übliche Objekte: nonces, names, keys, variables...
- Komplexe Terme: tuples, encryptions, signatures.

- Ein Protokoll ist definiert als eine Menge von Rollen
- Eine Rolle $\theta \in$ Q ist eine Liste von Aktionen
- Beispiele für Aktionen: $Send\ X', Y', m$; $Receive\ X', Y', m$; $New\ x$; $enc\ m, K$; $dec\ t, K$; $Verify\ t, m, K$;
- Beispiele für Aktionsprädikate im Thread: $Send(X, m)$; $Receive(X, m)$; $Gen(X, x)$; $Verify(X, y)$; $Has(X, m)$; $Honest(X')$

- Jede Rolle setzt sich aus einer Menge von Basissequenzen *BS* zusammen
- Eine Basissequenz BS_1 einer Rolle θ ist eine zusammenhängende Teilsequenz von θ
- Basissequenzen beginnen mit dem Start von θ oder einer *Receive* Aktion.
- die restlichen Aktionen einer Basissequenz sind keine *Receive* Aktionen.

- Jedem Protokoll wird eine Menge von möglichen Ausführungstraces zugeordnet. Diese Menge wird mit runs bezeichnet.
- Eine Protokoll-Logik wird etabliert, um über Mengen von Ausführungstraces Aussagen zu machen.
- Es wird die Korrektheit des Systems bewiesen

1. Ausführungsmodelle werden durch cords definiert.
2. Ein Cord ist eine Sequenz von Aktionen der einem Thread zugeordnet wird, wobei alle Variablen gebunden sind.
3. Aktionen werden durch Threads ausgeführt, jeder Thread ist mit einem Agent assoziiert.
4. Eine Untermenge der Agenten sind die ehrlichen Agenten $Honest(C)$, diese führen das Protokoll nach Spezifikation aus. Die Übrigen Agenten sind nicht an die Spezifikation gebunden und können Angreifer nach dem Dolev-Yao-Modell darstellen.
5. $[P]_X$ bezeichnet die Ausführung der Aktionssequenz P durch Thread X. Der zum Thread X gehörende Agent wird als X' bezeichnet.

Rekapitulation Protocol Composition Logic
Protokoll-Logik

- Für jede Aktion die in einer Ausführung eines Protokolls vorkommen kann wird ein entsprechendes Prädikat definiert.
- Die Protokoll-Logik wird um logische Konnektive erweitert, insbesondere $<$ für die temporale Abfolge.
- Die Protokoll-Logik wird um Prädikate erweitert, die Schlüsse über das Wissen von Threads ermöglichen

Rekapitulation Protocol Composition Logic
Protokoll-Logik

- Ein Run R eines Sicherheitsprotokolls Q erfüllt die Eigenschaft ϕ: $Q, R \models \phi$.
- Falls alle Runs eines Sicherheitsprotokolls die Eigenschaft ϕ erfüllen: $Q \models \phi$.
- Wenn eine Formel ϕ immer deduzierbar ist, ohne die *Honesty*-Regel: $\vdash \phi$.
- Wenn eine Forme ϕ immer deduzierbar ist, mit Hilfe der *Honesty*-Regel: $\vdash_Q \phi$.

- *Honesty*(X'): Agent X gehört zu den Protokollteilnehmern, die das Protokoll nach Vorschrift ausführen
- *Contains*(x, m) : Term x enthält Term m als Substring.
- *Gen*(Y, m) : Thread Y hat im Laufe der Protokollausführung term m erzeugt.

Probleme der PCL

Die PCL ist nicht in der Lage, die Korrektheit von Authentifikationsprotokollen zu beweisen, wenn diese keine Signatur benutzen.

- Das Problem besteht darin, dass die Existenz von Aktionen einer weiteren Protokollpartei nur unter bestimmten einschränkenden Vorbedingungen gelingt.
- Es existiert nur ein Axiom, welches Schlüsse über die Existenz von Aktionen eines zweiten Protokollteilnehmers zulässt,
- Das Axiom hat eine Vorbedingung, die nur von Protokollen erfüllt werden kann, welche Signaturen benutzen.
- Eigenschaften die die Existenz eines Kommunikationspartners voraussetzen: aliveness, agreement, matching conversations, synchronisation.

Probleme der PCL

- Eigenschaften die die Existenz eines Kommunikationspartners voraussetzen: aliveness, agreement, matching conversations, synchronisation.
- All diese Eigenschaften haben die Form: $\phi(x) \vdash \exists Y.\theta(Y)$
- typischerweise annotiert $\phi(X)$, dass Thread X eine Rolle ausführt, welche die Identität der zweiten Rolle verifiziert.
- Insgesamt sind diese Formel wie folgt zu verstehen: Wenn ein Thread X eine bestimmte Rolle ausführt, dann existiert ein zweiter Thread Y, der Aktionen einer anderen Rolle ausführt bzw. für den bestimmte Eigenschaften gelten, was in $\theta(Y)$ steht.

Probleme der PCL

Das einzige Axiom, welches Schlüsse über die Existenz eines Threads ermöglicht ist

$$\textbf{VER: } \mathit{Honest}(X') \wedge \mathit{Verify}(Y, \mathit{SIG}_{X'}\{|x|\} \wedge X' \neq Y') \supset$$

$$\exists X.\mathit{Send}(X, m) \wedge \mathit{Contains}(m, \mathit{SIG}_{Y'}\{|x|\})$$

Da dieses Axiom das **Signatur-Verifikations-Prädikat** *Verify* benutzt, kann es nur für Signatur-Protokolle benutzt werden.

Verbesserungen PCL

- Es wird eine **konsistenter** Mechanismus zum Umgang mit symmetrischer und asymmetrischer Kryptographie benötigt
- Die Logik muss erweitert werden um Axiome die Schlüsse über die Existenz eines Threads zulassen, wenn dass Protokoll **symmetrische Kryptographie** benutzt.
- Probleme mit **Public-Key** Protokollen müssen behoben werden.

Verbesserungen der Basis PCL

konsistenter Umgang mit Symmetrischer und Asymmetrischer Kryptographie

- Untergliederung der Verschlüsselung
- Differenzierung zwischen den Aktions-Typen encrypt/decrypt.
- Unterteilung der Schlüsselmengen
- Die Wahl der Aufteilung beeinflußt die Aktionsmengen, das *Has* Prädikat, weitere **ARX**-Axiome müssen hinzugefügt, dass Sec Axiom muss modifiziert werden.

Verbesserungen der PCL

- Symmetrischer Schlüssel K wird von den beiden Protokollteilnehmern X', Y' **geteilt**.
- Wenn der Schlüssel in einer Nachricht des Threads X vorkommt gibt es zwei mögliche Erzeuger des Schlüssels.
- Kann ausgeschlossen werden, dass Thread X den Schlüssel erzeugt hat folgt, dass ein anderer Thread den Schlüssel erzeugt hat.

Verbesserungen der PCL
Umgang mit Public-Key-Verfahren

- Eine Modifikation die den konsistenten Umgang mit symmetrischer sowie asymmetrischer Kryptographie sicherstellt ist nicht ausreichend.
- Mangel der Möglichkeit des Nachweises der Existenz eines Threads
- Es ist keine einfache Lösung des Problems möglich, da die public keys öffentlich, also auch jedem Angreifer bekannt sind. Beim Erhalt einer Nachricht die mit dem public key verschlüsselt worden ist, sind keine Schlussfolgerungen über den Absender möglich.
- Lösung würde weitere Axiome wie beispielsweise

$$[receivet]_X \exists Z.Send(Z, t)$$

benötigen, sowie weitere Mechanismen zum Umgang mit diesen.

Probleme der Basis PCL
Keine Schlüsse über vorherige Aktionen in einem Thread

Wenn festgestellt wird, dass ein Thread X zum **Zeitpunkt** n eine bestimmte Aktion(nach seiner Rolle) ausgeführt hat, ist kein Rückschluss über die Ausfürung der in der Rolle vorgeschriebenen Vorgängeraktion durch den Thread möglich.
Die PCL-Logik muss also um einen *Mechanismus der solche Schlüsse ermöglicht* erweitert werden.

Probleme der Basis PCL

Keine Schlüsse über vorherige Aktionen in einem Thread

Aus dem **Unvermögen** eine zeitliche Ordnung auf den Aktionen eines Threads zu etablieren resultiert, dass einige Beweise falsch sind.

Ein Beispiel hierfür ist die folgende Protokoll-Initiator-Rolle bestehend aus zwei Basissequenzen:

$BS_1 \equiv [newm; sendX', Y', m;]_X$
$BS_2 \equiv [receiveY', X', y, s, verifys, (y, m, X'), Y';$
$r = sign(y, m, Y'), X'; sendX', Y', r;]_X$

Durch Inspektion von BS2 wird **nicht ersichtlich** ob der Term m empfangen oder erzeugt wurde.

Probleme der PCL

Keine Schlüsse über vorherige Aktionen in einem Thread 3

Theorem: 'If Basic PCL as defined in [9,13,14,17] is sound, then the authentication properties of of the challenge response protocol CR from[14] cannot be proven by using basic PCL without using protocol composition rules.'

Beweisskizze:
Man beginnt mit der Betrachtung eines Challenge-Response-Protokolls CR'. Dieses besteht aus denselben Basissequenzen wie ein zweites Challenge -Response-Protokoll CR. Da ein Beweis in der PCL-Logik keine Kompositionsregeln benutzt, kann sich ein solcher Beweis nur auf Basissequenzen beziehen. Aus $\vdash_{Q_{CR}} \phi$ folgt nun $\vdash_{Q_{CR'}} \phi$ Dieses Wissen wird benutzt um die Weak-Authentification-Rule für CR' zu widerlegen. Folglich existiert kein Beweis$\vdash_{Q_{CR}} \phi$, der keine Kompositionsregeln benutzt.

Modifikation der PCL
Einfürung einer Präzedenzregel

- Gebraucht wird eine Inferenzregel die sicherstellt, dass gegeben die Aktion eines Threads zum Zeitpunkt t, die durch die Rolle vorgegebene *Vorgängeraktion auch im selben Thread* stattgefunden hat.
- eine solche Regel kann auch Nachteile mit sich, so könnten sich zusammengesetzte Beweise mit der PCL-Logik komplizierter machen.

Probleme mit der PCL
Keine Schlüsse über den Typ einer Variable

In der PCL existieren keine Konstrukte die das Schliessen über Typen ermöglichen. Es gibt Protokolle die in einem Modell mit Typen korrekt sind, in einem Modell ohne Typen jedoch Sicherheitslücken haben.

Ein Beispiel für ein solches Protokoll: Die Invariante γ_1 besagt, wenn der Thread eines ehrlichen Agenten eine Nachricht sendet, welche den Signaturterm S und den Term m beinhaltet, dann gilt entweder:

1. m wurde im Thread Y erzeugt.
2. Thread Y führt *receive* m, mit anschliessendem *send* (y, S) aus.

$$\gamma_1 \equiv (Send(Y, t) \wedge Contains(t, SIG_{Y'}\{|y, m.X'|\})) \supset$$
$$(Gen(Y, m) \vee (Receive(Y, (X', Y', m)) <$$
$$Send(Y, (Y', X', y, SIG_{Y'}\{|y, m, X'|\}))))$$

Als Nächstes betrachtet man die folgende Basissequenz:

$$BS_3 \equiv [receive(X', Y', x); newn; r = sign(n, x, X'); sendY', X', n, d;]_Y$$

BS3 entspricht einem Responder, der eine Ihm unbekannte Nachricht erhält, beispielsweise eine Nonce, der Responder antwortet mit einer signierten Version und einer frischen Nonce.

Probleme mit der PCL
Keine Schlüsse über den Typ einer Variable

Um die Invariante γ_1 zu widerlegen betrachtet man den Agenten Y, welcher den Thread Y' ausfürt. Unter der Annahme, dass $x = SIG_{Y'}\{y, m, X'\}$ gilt, wobei m durch Thread Y erzeugt wurde, folgt der Widerspruch. Es gilt $x = SIG_{Y'}\{y, m, X'\} \supset Contains(x, m)$. Als Resultat folgt, am Ende von BS_3 sendet ein Thread Y mit $Y' \neq Y$ eine Nachricht, die m enthält. Da Y den Term nicht selber erzeugt, noch diesen Term als solches empfangen hat, folgt der Widerspruch.

Diffie - Hellmann - Erweiterung der PCL

Diffie-Hellman ist ein auf asymmetrischer Kryptographie basierendes Authentifikationsprotokoll.

Definition: $g^a \bmod p = g^a$ und $(g^a \bmod p)^b \bmod p = h(a, b)$

1. Die *Fresh*-Regel wird erweitert um $Fresh(X, g(x))$ diese gilt falls $Fresh(X, x)$ gilt.
2. Die *Has*-Regel wird erweitert zu:
   $Has(X, a) \wedge Has(X, g(b)) \supset Has(X, h(a, b) \wedge Has(X, h(b, a)))$
3. Einführung weiterer Axiome:
   $\text{Computes}(X, g^{ab}) \equiv ((Has(X, a) \wedge Has(X, g^b))$
   $\vee (Has(X, b) \wedge Has(X, g^a)))$

Diffie - Hellmann - Erweiterung der PCL

1. $DH1 : Computes(X, g^{ab}) \supset Has(X, g^{ab})$
2. $DH2 : Has(X, g^{ab}) \supset$
   $(Computes(X, g^{ab}) \vee \exists m.(Receive(X, m) \wedge Contains(m, g^{ab})))$
3. $DH3 : (Receive(X, m) \wedge Contains(m, g^{ab})) \supset$
   $\exists Y, m'.(Computes(Y, g^{ab}) \wedge Send(Y, m')) \wedge$
   $Contains(m', g^{ab}))$
4. $DH4 : Fresh(X, a) \supset Fresh(X, g^{a})$

Diffie - Hellmann - Erweiterung der PCL

Das Diffie-Hellmann Protokoll kann auch mit den eben eingeführten Erweiterungen nicht äquivalent in der PCL-Logik ausgedrückt werden.

- Die Sequenz $Send(X, h(a, b)) < Receive(Y, h(b, a))$ kann im Ausführungsmodell nicht durch eine Änderung der Protokoll-Logik ermöglicht werden.
- Gegeben die Beobachtung $Has(X, h(ab))$ ist kein Schluss auf $Has(X, h(ba))$ möglich.

Zusammenfassung

Die PCL Logik hat Schwachstellen:

- Es Fehlen Inferenzregeln und Axiome, die von einigen Protokollen benötigt werden
- Das hinzufügen von neuen Inferenzregeln und Axiomen muss vorsichtig passieren, da ansonsten die Simplizität des PCL Protokolls verloren geht.