Protokollsicherheit

F7

Protokollimplementierung und -verifikation

Die Analyse von abstrahierten Protokollmodellen (z.B. mit Proverif) ist nur begrenzt praktikabel:

- Erstellung eines abstrakten Modells für den Code ist kompliziert und aufwändig.
- Ist das Modell korrekt?
- Das Model muss bei Änderungen in der Implementierung stets aktualisiert werden.

Lösungsansätze:

- Automatisierung der Erzeugung von Modellen von der Implementierung.
- Entwicklung von Methoden zur direkten Programmanalyse. Beispiel: Verstärkung des ML-Typsystems, so dass Protokolleigenschaften durch Typüberprüfung verifiziert werden können.

Automatische Erzeugung von Modellen

Beispiel:
Automatische Übersetzung von F#-Programmen in Proverif-Dateien [Bhargavan, Forunet, Gordon, Tse]

- Anwendung; TLS

Grenzen dieses Ansatzes:

- Die gesamte Protokollimplementierung wird in eine einzige große Proverif-Datei übersetzt. Grenzen der Skalierbarkeit mit der TLS-Implementierung bereits erreicht.
- Fragilität bezüglich der erfolgreichen automatischen Analyse
- Interpretation der Ergebnisse bezüglich der Originalsprache

Modulare Verifikation durch Typsysteme

Modulare Verifikation von Eigenschaften von F#-Protokollimplementierungen durch Verfeinerung des F#-Typsystems: **F7** [Bhargavan, Fournet, Gordon]

Der Ansatz greift die Designphilosophie des ML-Typsystems auf, welches für die modulare Verifikation von Programmeigenschaften entwickelt wurde.

Die Programmiersprache ML wurde Ende der 70er Jahre von Robin Milner als Metasprache für den Theorembeweiser LCF entwickelt. Ziele:

- Automatisierung von computergestützten Beweisen.
- ML wurde zur Programmierung von Beweistaktiken entwickelt.
- Das ML-Typsystem stellt sicher, dass nur wahre Theoreme bewiesen werden können.

```
(* ========================================================================= *)
(* LCF-style basis for Tarski-style Hilbert system of first order logic.     *)
(*                                                                           *)
(* Copyright (c) 2003-2007, John Harrison. (See "LICENSE.txt" for details.)  *)
(* ========================================================================= *)

(* ------------------------------------------------------------------------- *)
...
(*  if |- p ==> q and |- p then |- q                                         *)
(*  if |- p then |- forall x. p                                              *)
(*                                                                           *)
(*  |- p ==> (q ==> p)                                                       *)
(*  |- (p ==> q ==> r) ==> (p ==> q) ==> (p ==> r)                           *)
(*  |- ((p ==> false) ==> false) ==> p                                       *)
(*  |- (forall x. p ==> q) ==> (forall x. p) ==> (forall x. q)               *)
 ...
(* ------------------------------------------------------------------------- *)

module type Proofsystem =
   sig type thm
       val modusponens : thm -> thm -> thm
       val gen : string -> thm -> thm
       val axiom_addimp : fol formula -> fol formula -> thm
       val axiom_distribimp :
            fol formula -> fol formula -> fol formula -> thm
       val axiom_doubleneg : fol formula -> thm
       val axiom_allimp : string -> fol formula -> fol formula -> thm
       ...
   end;;
```

Ein Benutzer/“Angreifer” kann nur wahre Theoreme erzeugen, da der Zugriff auf Theoreme nur durch die Methoden einer Signatur möglich ist.

Refinement Types

Bhargavan, Fournet und Gordon haben 2008 eine Sprache mit “Refinement Types” für die Protokollverifikation vorgeschlagen.

F7:

- Basiert auf einer Teilmenge von F# (ohne Klassen, Objekte, etc.)
- Das Hauptmerkmal von F7 ist das gegenüber F# verfeinerte Typsystem.
- Refinement Types erlauben die Modularisierung der Protokollverifikation.

Refinement Types

Ein **Refinement Type** ist ein Typ, der durch eine logische Formel eingeschränkt wird.

Notation in F7: $x : T\{C\}$

Die Werte von $x : T\{C\}$ sind alle $M : T$, für die $C\{M/x\}$ wahr ist.

Beispiele:

- $x : \mathsf{int}\{x > 0\}$
- $k : \mathsf{bytes}\{Key(k, a, b) \wedge Priv(k)\}$

Für die logischen Formeln wird erststufige Logik mit beliebigen Prädikatensymbolen benutzt.

F7

F7 ist ein Typchecker für F#, der mit Refinement Types in Modulsignaturen überprüfen kann.

- Der Programmtext ist F#-Code, der mit dem normalen F#-Compiler übersetzt wird.
- Zusätzlich zum normalen Quelltext schreibt man für jedes Modul eine F7-Signatur, die Refinement Types und logische Ausdrücke enthalten kann.
- F7 kann die F#-Implementierung gegen die F7-Signatur prüfen.

F7 — Minimalbeispiel

Datei `prg.fs`:

```
module Prg

let f x = x + 1
```

Datei `prg.f7`:

```
module Prg

val fact : x : int -> r : int { r > x }
```

F7 — Protokollbeispiel

Quelle: [Fournet, Bhargavan, Gordon: Cryptographic Verification by Typing for a Sample Protocol Implementation, 2010]

Einfaches RPC-Protokoll:

1. $A \to B$: utf8(s) | (hmacsha1 k_{AB} (request s))
2. $B \to A$: utf8(t) | (hmacsha1 k_{AB} (response $s\ t$))

F7 — Protokollbeispiel

1. $A \to B$: utf8(s) | (hmacsha1 k_{AB} (request s))
2. $B \to A$: utf8(t) | (hmacsha1 k_{AB} (response s t))

Die Sicherheit des Protokolls hängt mindestens von folgenden Faktoren ab:

- Die Funktion hmacsha1 ist kryptographisch sicher.
- Weder A noch B gibt k_{AB} bekannt.
- Die Funktionen request und response sind injektiv und ihre Wertebereiche disjunkt.

F7 — Protokollbeispiel

1. $A \to B$: utf8(s) | (hmacsha1 k_{AB} (request s))
2. $B \to A$: utf8(t) | (hmacsha1 k_{AB} (response s t))

Ereignisse im Protokollablauf werden durch logische Prädikate repräsentiert:

- $Request(A, B, s)$ wird direkt vor dem Senden der Nachricht 1 wahr.
- $Response(A, B, s, t)$: wird direkt for dem Senden der Nachricht 2 wahr.
- $KeyAB(k, A, B)$ wird wahr wenn der Schlüssel k wird mit$A$ und B assoziiert wird.
- $Bad(H)$ wird wahr wenn H ein Geheimnis veröffentlicht.

F7 — Protokollbeispiel

1. $A \to B$: $\quad \mathsf{utf8}(s) \mid (\mathsf{hmacsha1}\ k_{AB}\ (\mathsf{request}\ s))$
2. $B \to A$: $\quad \mathsf{utf8}(t) \mid (\mathsf{hmacsha1}\ k_{AB}\ (\mathsf{response}\ s\ t))$

Verifikationsziele:

- $RecvRequest(A, B, s)$ soll in B nach dem Empfang der Nachricht 1 gelten.
- $RecvResponse(A, B, s, t)$ soll in A nach Empfang der Nachricht 2 gelten.

$$\forall A, B, s.$$
$$RecvRequest(A, B, s) \iff Request(A, B, s) \lor Bad(A) \lor Bad(B)$$

$$\forall A, B, s, t.$$
$$RecvResponse(A, B, s, t) \iff (Request(A, B, s) \land Response(A, B, s, t)) \lor Bad(A) \lor Bad(B)$$

F7 — Protokollbeispiel

1. $A \to B$: utf8(s) | (hmacsha1 k_{AB} (request s))
2. $B \to A$: utf8(t) | (hmacsha1 k_{AB} (response s t))

```
let mkKeyAB a b = let k = hmac_keygen() in assume (KeyAB(k,a,b)); k
let request s = concat (utf8(str "Request")) (utf8 s)
let response s t = concat (utf8(str "Response")) (concat (utf8 s) (utf8 t))
```

```
let client (a:str) (b:str) (k:keyab) (s:str) =
  assume (Request(a,b,s));
  let c = Net.connect p in
  let mac = hmacsha1 k (request s) in
  Net.send c (concat (utf8 s) mac);
  let (pload',mac') = iconcat (Net.recv c) in
  let t = iutf8 pload' in
  if hmacsha1Verify k (response s t) mac';
  expect (RecvResponse(a,b,s,t))
```

```
let server(a:str) (b:str) (k:keyab) : unit =
  let c = Net.listen p in
  let (pload,mac) = iconcat (Net.recv c) in
  let s = iutf8 pload in
  hmacsha1Verify k (request s) mac;
  expect (RecvRequest(a,b,s));
  let t = service s in
  assume (Response(a,b,s,t));
  let mac' = hmacsha1 k (response s t) in
  Net.send c (concat (utf8 t) mac')
```

Modellierung des Angreifers

Der Angreifer wird als beliebigies F#-Programm modelliert.

Die Möglichkeiten des Angreifers werden durch Signaturen für Bibliothek und Protokollimplementierung spezifiert.

Angreifer-Interface für das Protokoll:

```
let setup (a:str) (b:str) =
  let k = mkKeyAB a b in
  (fun s -> client a b k s),
  (fun () -> server a b k),
  (fun () -> assume (Bad(a)); k),
  (fun () -> assume (Bad(b)); k)
```

mit Signatur

```
val setup: strpub -> strpub ->
  (strpub -> unit) * (unit -> unit) *
  (unit -> keypub) * (unit -> keypub)
```

Sicherheitsaussage

- Mit diesem Fall kann der Angreifer beliebig viele Clients und Server starten und er kann beide korrumpieren.
- Über das Angreifer-Interface der Bibliothek erhält der Angreifer Zugriff auf Netzwerk und Kryptographiefunktionen.

Durch das Typsystem wird nun eine Sicherheitsaussage der folgenden Form garantiert:

- Ist jedes Modul bezüglich seiner F7-Signatur wohltypisiert
- und ist der Angreifer ein beliebiges F#-Programm, das Zugriff auf das Angreifer-Interface hat,

dann ist das Programm semantisch sicher.

Dabei ist ein Programm semantisch sicher, wenn nur `assert`-Anweisungen ausgeführt werden, die logisch aus den vorher mir `assume`-Anweisungen angenommenen Formeln folgen.

Modularität — Kryptographie

Die Kryptographie-Bibliothek stellt folgendes Interface für MACs bereit:

```
val hmac_keygen: unit -> k:key{MKey(k)}
val hmacsha1:
  k:key ->
  b:bytes{ (MKey(k) /\ MACSays(k,b)) \/ (Pub_k(k) /\ Pub(b)) } ->
  h:bytes{ IsMAC(h,k,b) /\ (Pub(b) => Pub(h)) }
val hmacsha1Verify:
  k:key{MKey(k) \/ Pub_k(k)} -> b:bytes ->
  h:bytes -> unit{IsMAC(h,k,b) /\ MACSays(k,b)}
val hmacsha1Verify1:
  k:key{MKey(k) \/ Pub_k(k)} -> b:bytes ->
  h:bytes -> w:bool{ w = true => MACSays(k,b)}
```

Modularität — Kryptographie

Refinement Types für MACs (idealisiert):

```
val hmac_keygen: unit -> k:key
val hmacsha1:
  k:key ->
  b:bytes{ MACSays(k,b) } ->
  h:bytes
val hmacsha1Verify:
  k:key -> b:bytes ->
  h:bytes -> unit{ MACSays(k,b) }
val hmacsha1Verify1:
  k:key -> b:bytes ->
  h:bytes -> w:bool{ w = true => MACSays(k,b)}
```

Protokollinvarianten

Definition von `MACSays`:

```
!a,b,k,m. KeyAB(k,a,b) => ( MACSays(k,m) <=>
(  (?s. Requested(m,s) /\ Request(a,b,s)) \/
  (?s,t. Responded(m,s,t) /\ Response(a,b,s,t)) \/
    (Bad(a) \/ Bad(b))))
```

Dabei sind `Requested` und `Responded` Nachbedingungen von `request` und `respond`.

Protokollinvarianten

Mit folgenden weiteren Annahmen kann der F7-Typchecker das Protokoll verifizieren:

```
theorem
  !k,a,b,a',b'. KeyAB(k,a,b) /\ KeyAB(k,a',b') => (a=a') /\ (b=b')
```

```
theorem
  !a,b,k. KeyAB(k,a,b) /\ Pub_k(k) => Bad(a) \/ Bad(b)
```

Die Bibliotheken benutzen weitere Prädikate zur Modellierung kryptographischer Protokolle:

- $Pub(x)$ ist wahr wenn x dem Angreifer bekannt ist.
- $Bytes(x)$ ist wahr wenn x im Lauf des Protokolls auftaucht.
- ...