Testing Firewall Policies using HOL-TestGen

Semesterarbeit Wintersemester 05/06

Lukas Brügger
Betreuer: Achim D.Brucker, Burkhart Wolff, Diana Senn

Information Security
ETH Zürich

10. März 2006

Inhalt

Aufgabenstellung

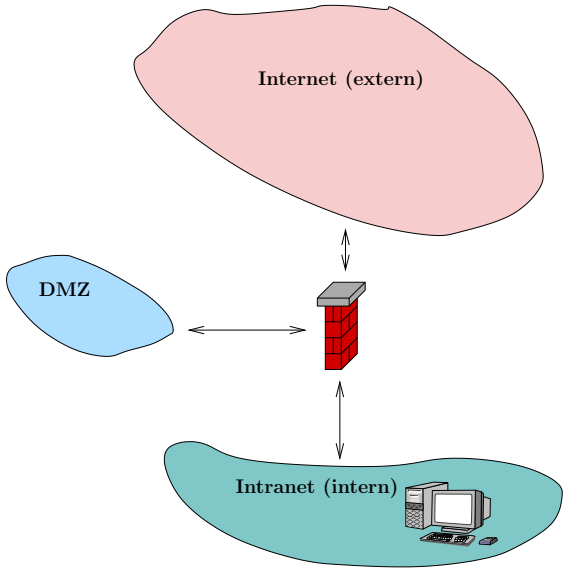
Ziel:
Semiautomatische Generierung von Daten um zu Testen, ob eine Firewall eine gegebene Sicherheitspolicy richtig implementiert.

Aufgabenstellung

Ziel:
Semiautomatische Generierung von Daten um zu Testen, ob eine Firewall eine gegebene Sicherheitspolicy richtig implementiert.

- Entwickeln eines formalen Modelles einer zustandslosen und einer zustandsbehafteten Firewall in Isabelle/HOL
- Ausarbeiten von Fallstudien
- HOL-TestGen auf diese Fallstudien anwenden

Ein typisches Szenario



$\longrightarrow$	Intranet	DMZ	Internet
Intranet	-	smtp, imap	all protocols except smtp
DMZ	$\emptyset$	-	smtp
Internet	$\emptyset$	http,smtp	-

Firewall

- Eine Firewall ist ein *zustandsloser* oder *zustandsbehafteter Packetfilter.*
- Das Filtern (`accept` oder `deny`) geschieht basierend auf
  - Quelladresse
  - Zieladresse
  - Protokoll
  - evtl. Zustand

HOL-TestGen

HOL-TestGen ist eine interaktive Testumgebung, basierend auf der interaktiven Beweisumgebung Isabelle/HOL.

Es erlaubt

- die Formulierung der Testspezifikation in HOL
- die automatische und interaktive Entwicklung von abstrakten Testfällen
- deren Verfeinerung zu konkreten Testdaten
- die Generierung eines Testskriptes zur Testausführung

Wurde bisher nur zum Testen von Datenstrukturen verwendet.

Inhalt

Das Modell

Grundlage des Modelles ist ein Packet.

types (α,β) packet =
"id $\times$ protocol $\times\alpha$ src $\times\alpha$ dest $\times\beta$ content"

- `id` ist ein Integer
- `protocol` ist ein Aufzählungstyp, z.B. ftp, http, smtp
- α `src` und α `dest` ist eine abstrakte Adressangabe. z.B. IPv4:

  types
  ipv4_ip = "(int $\times$ int $\times$ int $\times$ int)"
  ipv4 = "(ipv4_ip $\times$ int)"

  Modell kann einfach um z.B. IPv6 Adressen erweitert werden
- β `content` ist der abstrakte Inhalt.

Das Modell

Die Firewall kann ein Packet entweder akzeptieren oder ablehnen.

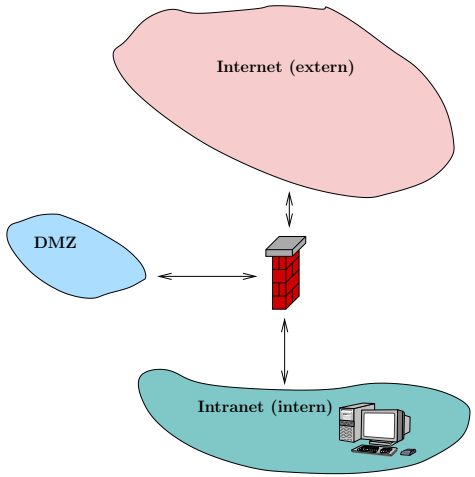
datatype α out = accept α | deny

Eine Policy ist eine partielle Abbildung (Map) von `packet` zu `packet out`.

types (α, β) Policy =
"(α, β) packet $\rightharpoonup$ ((α, β) packet) out"

Ein Kombinatorsatz erlaubt die einfache Spezifikation einer solchen Policy.

Fallstudie



$\longrightarrow$	Intranet	DMZ	Internet
Intranet	-	smtp, imap	all protocols except smtp
DMZ	$\emptyset$	-	smtp
Internet	$\emptyset$	http,smtp	-

Fallstudie

src	dest	protocol	action
Internet	DMZ	http	*accept*
Internet	DMZ	smtp	*accept*
⋮	⋮	⋮	⋮
$*$	$*$	$*$	*deny*

Fallstudie

src	dest	protocol	action
Internet	DMZ	http	*accept*
Internet	DMZ	smtp	*accept*
⋮	⋮	⋮	⋮
∗	∗	∗	*deny*

```
constdefs Internet_DMZ :: "(ipv4, content) Rule"
          "Internet_DMZ ≡
              (allow_prot_from_to smtp internet dmz) ++
              (allow_prot_from_to http internet dmz)"
```

Fallstudie

src	dest	protocol	action
Internet	DMZ	http	*accept*
Internet	DMZ	smtp	*accept*
⋮	⋮	⋮	⋮
∗	∗	∗	*deny*

constdefs Internet_DMZ :: "(ipv4, content) Rule"
"Internet_DMZ ≡
(allow_prot_from_to smtp internet dmz) ++
(allow_prot_from_to http internet dmz)"

Die gesamte Policy ist dann:

constdefs test_policy :: "(ipv4,content) Policy"
"test_policy ≡ deny_all ++ Internet_DMZ ++ ..."

Fallstudie

Die Testspezifikation sieht anschliessend folgendermassen aus:

test_spec "FUT x = test_policy x"

Fallstudie

Die Testspezifikation sieht anschliessend folgendermassen aus:

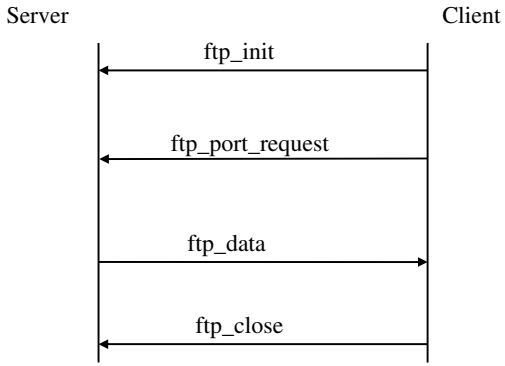
test_spec "FUT x = test_policy x"

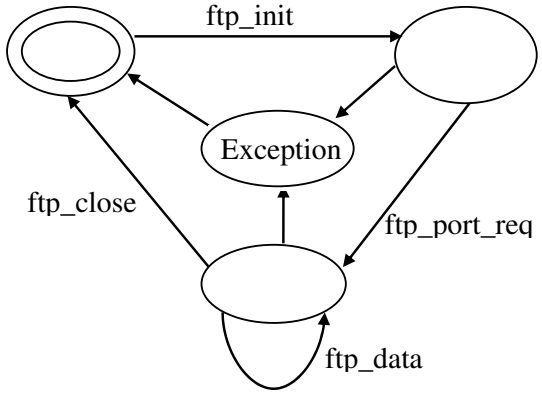
Ergebnisse:

- Benötigte Zeit: 19h 11'
- Anzahl generierter Testfälle: 485
- Die generierten Testdaten sehen folgendermassen aus:
  - `FUT (6,smtp,((192,169,2,8),25),((6,2,0,4),2),data) = Some (accept (6,smtp,((192,169,2,8),25),((6,2,0,4),2),data))`
  - `FUT (2,smtp,((192,168,0,6),6),((9,0,8,0),6),data) = Some deny`

Inhalt

FTP-Protokoll (vereinfacht)





Das Modell

- Eine zustandsbehaftete Firewall ist eine Erweiterung des zustandslosen Falles.
  Idee: Eine Policy ist immer noch dasselbe, jedoch kann sie sich nun dynamisch ändern!

Das Modell

- Eine zustandsbehaftete Firewall ist eine Erweiterung des zustandslosen Falles.
  Idee: Eine Policy ist immer noch dasselbe, jedoch kann sie sich nun dynamisch ändern!
- Ein Zustand ist ein Tupel der History und der aktuellen Policy.

  types (α,β,γ) FWState = "$\alpha \times (\beta,\gamma)$ Policy"

Das Modell

- Eine zustandsbehaftete Firewall ist eine Erweiterung des zustandslosen Falles.
  Idee: Eine Policy ist immer noch dasselbe, jedoch kann sie sich nun dynamisch ändern!
- Ein Zustand ist ein Tupel der History und der aktuellen Policy.

  types (α,β,γ) FWState = "$\alpha \times (\beta,\gamma)$ Policy"

- Eine FWStateTransition ist ein Map von einem eintreffenden Packet und einem Zustand zu einem neuen Zustand.

  types (α,β,γ) FWStateTransition =
  "((β,γ) In_Packet $\times$ (α,β,γ) FWState) $\rightharpoonup$
  ((α,β,γ) FWState)"

▸ Mehr

Testdaten

Die Testdaten sind nun Listen von Packeten, z.B.:

```
FUT [(6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), close),
     (6, ftp, ((4, 7, 9, 8), 21), ((192, 168, 3, 1), 3), ftp_data),
     (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), port_request 3),
     (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), init )] =
([(6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), close),
  (6, ftp, ((4, 7, 9, 8), 21), ((192, 168, 3, 1), 3), ftp_data),
  (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), port_request 3),
  (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), init )],
new_policy)
```

Auf der linken Seite haben wir eine Liste aller Packete, welche die Firewall erhält, auf der rechten Seite eine List aller Packete die sie akeptiert und die neue Policy (normalerweise aufgefaltet).

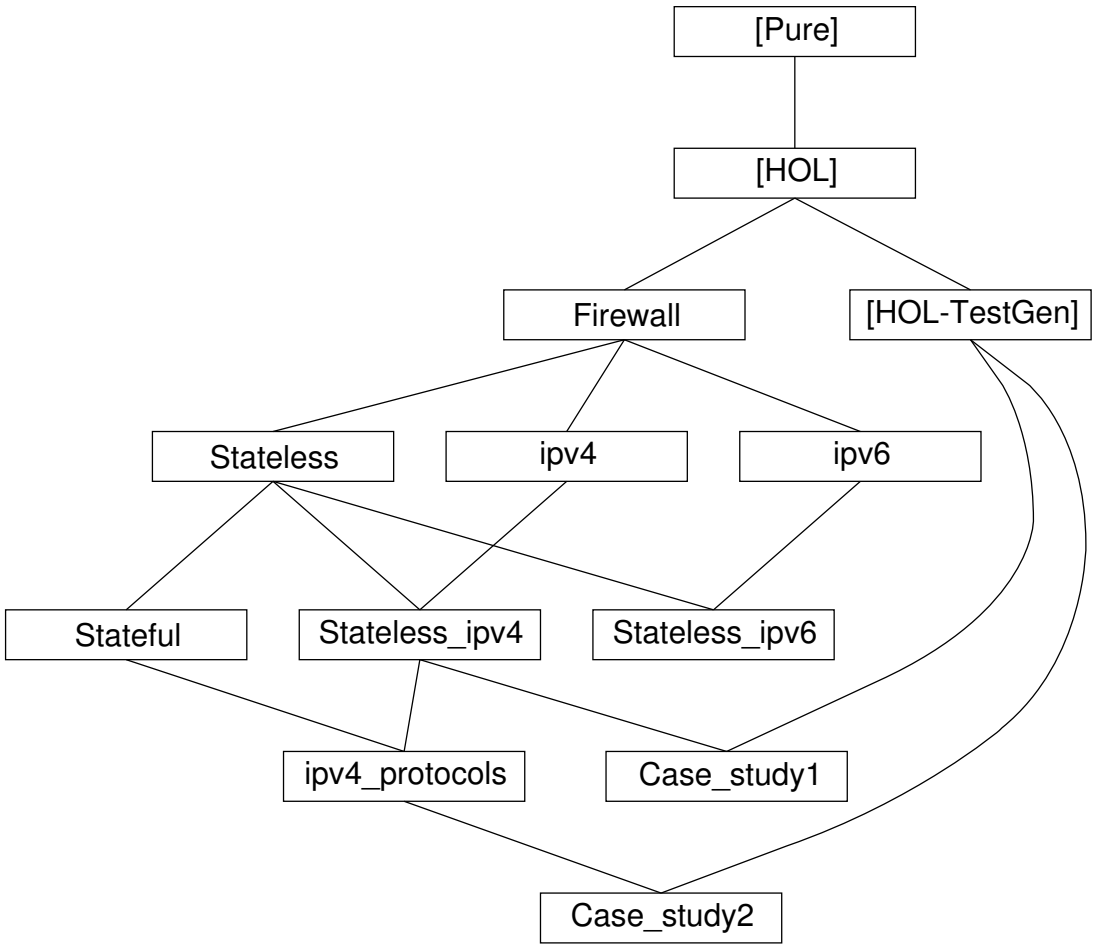
Ergebnisse

Mit Suchtiefe vier:

- Benötigte Zeit: 7 Minuten
- Anzahl Testfälle: 4

Die benötigte Zeit ist vor allem von der Tiefe der Traces abhängig, bei grösseren Protokollen steigt sie schnell an.

Theoriegraph



Inhalt

Fazit

- Formales und realistisches Modell einer Firewall in HOL
- Modell ist sehr generisch und einfach erweiterbar
- Gut aussehende und verwertbare Testdaten
- Anstoss zur Weiterentwicklung von HOL-TestGen

Future Work

- HOL-TestGen weiterentwickeln
- Zusätzliche Protokolle (VoIP) integrieren
- IPv6 modellieren
- Verwendung der Testdaten in `fwtest`

Protokollspezifikation

Ein Protokoll wird dann mittels einer FWStateTransition spezifiziert. Dies wird mit einer Monade erreicht.

```
consts FTP_ST_2 :: "((ipv4,ftp_msg) history, ipv4, ftp_msg) FWStateTransition"
recdef FTP_ST_2 "{}"
  FTP_ST_2_1: "FTP_ST_2 ((a,ftp,c,d,port_request port_r), (InL,policy )) =
       ( if  p_accept (a,ftp,c,d,port_request port_r)  policy  then
          ( if  not_before  (is_close  a)  (is_init  a)  InL  then
                Some ((a,ftp,c,d,port_request port_r)#InL, policy  ++
           (allow_prot_from_to_port ftp (subnet_of d) (subnet_of c) port_r))
                else  Some ((a,ftp,c,d,port_request port_r)#InL,policy))
           else  Some (InL,policy))"

FTP_ST_2_2 : "FTP_ST_2 (x, (InL,policy)) = None"
```

Protokollspezifikation

Ein Protokoll wird dann mittels einer FWStateTransition spezifiziert. Dies wird mit einer Monade erreicht.

```
consts FTP_ST_2 :: "((ipv4,ftp_msg) history, ipv4, ftp_msg) FWStateTransition"
recdef FTP_ST_2 "{}"
  FTP_ST_2_1: "FTP_ST_2 ((a,ftp,c,d,port_request port_r), (InL,policy )) =
       (if p_accept (a,ftp,c,d,port_request port_r) policy then
          (if not_before (is_close a) (is_init a) InL then
                Some ((a,ftp,c,d,port_request port_r)#InL, policy ++
         (allow_prot_from_to_port ftp (subnet_of d) (subnet_of c) port_r))
                else Some ((a,ftp,c,d,port_request port_r)#InL,policy))
           else Some (InL,policy))"

FTP_ST_2_2 : "FTP_ST_2 (x, (InL,policy)) = None"
```

Das gesamte FTP-Protokoll ist dann einfach:

```
consts FTP_ST::"((ipv4,ftp_content) history,ipv4,ftp_content)FWStateTransition"
defs FTP_ST_def: "FTP_ST ≡FTP_ST_1 orelse
                          FTP_ST_2 orelse FTP_ST_3 orelse
                          FTP_ST_4 orelse FTP_ST_E"
```

Nächste Schritte

Operator `Mfold`, der eine History, eine Policy und eine FWStateTransition nimmt und einen FWState daraus generiert.

consts Mfold :: "(β,γ) history $\Rightarrow$
(α,β,γ) FWState $\Rightarrow$
(α,β,γ) FWStateTransition $\Rightarrow$
(α,β,γ) FWState "

primrec
"Mfold [] S ST = S"
"Mfold (a#H) S ST = (case ST (a,S) **of** None $\Rightarrow$ Mfold H S ST
| Some Sn $\Rightarrow$ Mfold H Sn ST)"

Traces, die man Testen möchte

constdefs

```
NB_ftp :: "'a src ⇒ 'a dest ⇒ id ⇒ port ⇒ ('a,ftp_msg) history set"
"NB_ftp s d i p ≡{x. (is_ftp S0 s d i p x)}"
```

Testspezifikation

test_spec "t ∈NB_ftp c s i p ∧ is_in_intranet c ∧
is_in_internet s ∧ (snd s) = 21 ⟹
FUT t = Mfold (rev t) ([], ftp_policy) FTP_ST"

▸ Back

is_ftp

consts

is_ftp :: "ftp_states ⇒ α adr ⇒ α adr ⇒ id ⇒ port ⇒ (α,ftp_msg) history ⇒ bool"

primrec

"is_ftp H c s i p [] = (H=S3)"

"is_ftp H c s i p (x#lnL) = (λ (id,pr,sr,de,co). (((pr = ftp ∧ id = i ∧(
(H=S2 ∧sr = c ∧de = s ∧co = init ∧ is_ftp S3 c s i p lnL) ∨
(H=S1 ∧sr = c ∧de = s ∧co = port_request p ∧ is_ftp S2 c s i p lnL) ∨
(H=S1 ∧sr = s ∧de = (fst c,p) ∧co=data ∧is_ftp S1 c s i p lnL) ∨
(H=S0 ∧sr = c ∧de = s ∧co = close ∧ is_ftp S1 c s i p lnL)))))) x"

▸ Back

orelse

```
consts orelse :: "(α ⇀ β) ⇒ (α ⇀ β) ⇒ (α ⇀ β)"
  (infixl "orelse" 100)
defs orelse_def: "(f orelse g) x ≡ case f x of None ⇒ g x
                                             | Some y ⇒ Some y"
```

allow_prot_from_to

constdefs
allow_all :: "(α, β) Rule"
"allow_all p ≡Some (accept p)"

constdefs
allow_prot_from_to :: "protocol ⇒ α::net set set ⇒ α::net set set ⇒

"allow_prot_from_to prot src_net dest_net ≡allow_all |‘
{pa. src pa ⊏ src_net ∧
dest pa ⊏ dest_net ∧
protocol pa = prot}"

map_add

```
consts map_add :: "(α ~=> β) => (α ~=> β) => (α ~=> β)"
          ( infixl  "++" 100)
defs    map_add_def: "
      m1++m2 == %x. case m2 x of None => m1 x
                                | Some y => Some y"
```