

# 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

Die Problemstellung

Der zustandslose Fall

Der zustandsbehaftete Fall

Fazit

# Aufgabenstellung

## Ziel:

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

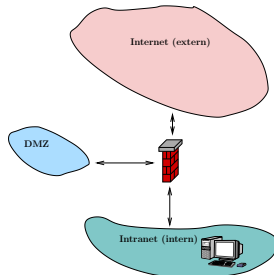
# Aufgabenstellung

## 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



→	Intranet	DMZ	Internet
Intranet	-	smtp, imap	all protocols except smtp
DMZ	∅	-	smtp
Internet	∅	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

Die Problemstellung

**Der zustandslose Fall**

Der zustandsbehaftete Fall

Fazit

## Das Modell

Grundlage des Modelles ist ein Packet.

**types**  $(\alpha, \beta)$  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
- ▶  $\alpha$  src und  $\alpha$  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

- ▶  $\beta$  content ist der abstrakte Inhalt.

## Das Modell

Die Firewall kann ein Packet entweder akzeptieren oder ablehnen.

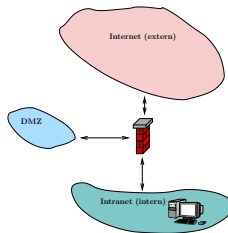
**datatype**  $\alpha$  out = accept  $\alpha$  | deny

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

**types**  $(\alpha, \beta)$  Policy =  
" $(\alpha, \beta)$  packet  $\rightarrow$   $((\alpha, \beta)$  packet) out"

Ein Kombinatoratz erlaubt die einfache Spezifikation einer solchen Policy.

## Fallstudie



→	Intranet	DMZ	Internet
Intranet	-	smtp, imap	all protocols except smtp
DMZ	∅	-	smtp
Internet	∅	http, smtp	-

## Fallstudie

src	dest	protocol	action
Internet	DMZ	http	<i>accept</i>
Internet	DMZ	smtp	<i>accept</i>
⋮	⋮	⋮	⋮
*	*	*	<i>deny</i>

## Fallstudie

src	dest	protocol	action
Internet	DMZ	http	<i>accept</i>
Internet	DMZ	smtp	<i>accept</i>
⋮	⋮	⋮	⋮
*	*	*	<i>deny</i>

```
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	<i>accept</i>
Internet	DMZ	smtp	<i>accept</i>
⋮	⋮	⋮	⋮
*	*	*	<i>deny</i>

```
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

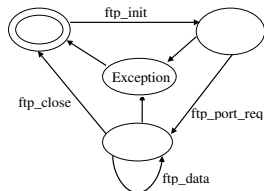
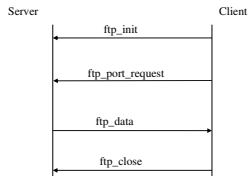
Die Problemstellung

Der zustandslose Fall

Der zustandsbehaftete Fall

Fazit

## 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**  $(\alpha, \beta, \gamma)$  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**  $(\alpha, \beta, \gamma)$  FWState = " $\alpha \times (\beta, \gamma)$  Policy"

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

**types**  $(\alpha, \beta, \gamma)$  FWStateTransition =  
" $((\beta, \gamma)$  In\_Packet  $\times (\alpha, \beta, \gamma)$  FWState)  $\rightarrow$   
 $((\alpha, \beta, \gamma)$  FWState)"

## 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).

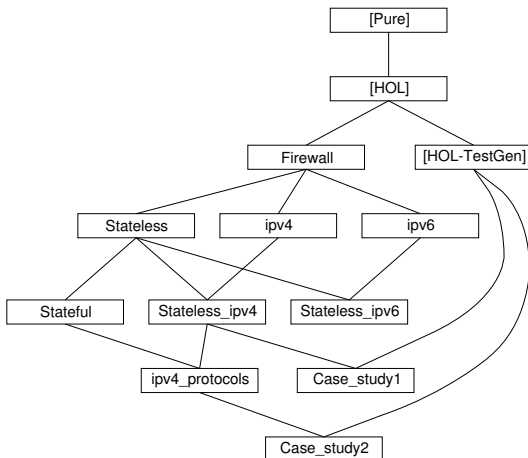
## Ergebnisse

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

Die Problemstellung

Der zustandslose Fall

Der zustandsbehaftete Fall

Fazit

## 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_5"
```

## Nächste Schritte

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

```
consts Mfold :: "( $\beta, \gamma$ ) history  $\Rightarrow$   
                ( $\alpha, \beta, \gamma$ ) FWState  $\Rightarrow$   
                ( $\alpha, \beta, \gamma$ ) FWStateTransition  $\Rightarrow$   
                ( $\alpha, \beta, \gamma$ ) 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  $\Rightarrow$   $\alpha$  adr  $\Rightarrow$   $\alpha$  adr  $\Rightarrow$  id  $\Rightarrow$  port  $\Rightarrow$  ( $\alpha$ ,ftp_msg) history  $\Rightarrow$  bool"
```

## primrec

```
"is_ftp H c s i p [] = (H=S3)"  
"is_ftp H c s i p (x#lnL) = ( $\lambda$  (id,pr, sr,de,co). (((pr = ftp  $\wedge$  id = i  $\wedge$  (H=S2  $\wedge$  sr = c  $\wedge$  de = s  $\wedge$  co = init  $\wedge$  is_ftp S3 c s i p lnL)  $\vee$  (H=S1  $\wedge$  sr = c  $\wedge$  de = s  $\wedge$  co = port_request p  $\wedge$  is_ftp S2 c s i p lnL)  $\vee$  (H=S1  $\wedge$  sr = s  $\wedge$  de = (fst c,p)  $\wedge$  co=data  $\wedge$  is_ftp S1 c s i p lnL)  $\vee$  (H=S0  $\wedge$  sr = c  $\wedge$  de = s  $\wedge$  co = close  $\wedge$  is_ftp S1 c s i p lnL) ))))) x"
```

▶ Back

## orelse

```
consts orelse :: " $(\alpha \rightarrow \beta) \Rightarrow (\alpha \rightarrow \beta) \Rightarrow (\alpha \rightarrow \beta)$ "  
  (infixl "orelse" 100)  
defs orelse_def: "(f orelse g) x  $\equiv$  case f x of None  $\Rightarrow$  g x  
  | Some y  $\Rightarrow$  Some y"
```

# allow\_prot\_from\_to

## constdefs

```
allow_all      :: "( $\alpha$ ,  $\beta$ ) Rule"  
"allow_all p  $\equiv$  Some (accept p)"
```

## constdefs

```
allow_prot_from_to :: "protocol  $\Rightarrow$   $\alpha$ ::net set set  $\Rightarrow$   $\alpha$ ::net set set  $\Rightarrow$ 
```

```
"allow_prot_from_to prot src_net dest_net  $\equiv$  allow_all |"  
    {pa. src pa  $\sqsubset$  src_net  $\wedge$   
      dest pa  $\sqsubset$  dest_net  $\wedge$   
      protocol pa = prot}"
```

## map\_add

```
consts map_add :: "( $\alpha \rightsquigarrow \beta$ ) => ( $\alpha \rightsquigarrow \beta$ ) => ( $\alpha \rightsquigarrow \beta$ )"  
  (infixl "++" 100)  
defs   map_add_def: "  
  m1++m2 == %x. case m2 x of None => m1 x  
                | Some y => Some y"
```