# Modelling and analysing security in the presence of compromising adverseries

Hannes Schröder

Universität Potsdam

19. November 2010

# Übersicht

# Motivation

- Kompromittierung ist allgegenwärtig
- Sicherheitsprotokolle haben Schwächen

- Vorhandene Testmodelle oft einschränkend
  $\Rightarrow$ Formalisierung von Angreifer und Angegriffenem

# Motivation

- geeignete formale Darstellung von Umgebung, Kommunikationspartner und Angreifer
- entsprechendes Darstellen von Fähigkeiten des Angreifers als Übergangsregeln

# Notation

- $dom(f)$ und $ran(f)$ als Definitions- und Wertebereich von f
- $f[b \leftarrow\!\shortmid\ a]$ als Update von f
- $f : X \nrightarrow Y$ als partielle Funktion von X nach Y
- für jede Menge S ist $\mathcal{P}(S)$ die Potenzmenge von S und S* die Menge aller finiten Terme in S

# Notation

- $\langle s_0, s_1, ..., s_n \rangle$ als Sequenz von Elementen der Länge $|s|$, $i$ mit $i < |s|$ als $i$-tes Element
- $s \hat{} s'$ ist die Konkatenation von $s$ und $s'$
- $e \in s$ wenn $\exists\, i.s_i = e$
- $union(s)$ als Menge aller Elemente in s
- $last(\langle\rangle) =$ und $last(s \hat{} \langle e \rangle) = e$

# Terme

- Term ::= Agent | Role | Fresh | Var |
  Fresh#TID | Var#TID | (Term,Term) |
  pk(Term) | sk(Term) | k(Term) |
  $\{|\ \text{Term}\ |\}^{a}_{Term}$ | $\{|\ \text{Term}\ |\}^{s}_{Term}$ |
  Func(Term*)

# Terme

- $t \in M \Rightarrow M \vdash t$
- $M \vdash t_1 \wedge M \vdash t_2 \Leftrightarrow M \vdash (t_1, t_2)$
- $M \vdash \{|t_1|\}^s_{t_2} \wedge \Rightarrow M \vdash t_1$
- $M \vdash t_1 \wedge M \vdash t_2 \Rightarrow M \vdash \{|t_1|\}^s_{t_2}$
- $M \vdash \{|t_1|\}^a_{t_2} \wedge M \vdash (t_2)^{-1} \Rightarrow M \vdash t_1$
- $M \vdash t_1 \wedge M \vdash t_2 \Rightarrow M \vdash \{|t_1|\}^a_{t_2}$
- $\bigwedge_{0 \leq i \leq n} M \vdash t_i \Rightarrow M \vdash f(t_0, \cdots, t_n)$

# Events

- Event ::= AgentEvent | AdversaryEvent
- AgentEvent ::=
  create(Role,Agent) | send(Term) | recv(Term) |
  generate($\mathcal{P}$(Fresh)) | state($\mathcal{P}$(Term)) | sessionkeys($\mathcal{P}$(Term))
- AdversaryEvent ::=
  LKR(Agent) | SKR(TID) | SR(TID) | RNR(TID)

# Protokolle

- partielle Funktion von Rollen zu Aktionen, d.h.

$$Protocol : Role \nrightarrow AgentEvent^*$$

- Beispielprotokoll SP:

$$\begin{aligned}&\mathrm{SP(Init)} = \\ &\langle generate(\{key\}), state(\{key, \{|Resp, key|\}^a_{sk(Init)}\}), \\ &send(Init, Resp, \{|\{|Resp, key|\}^a_{sk(Init)}|\}^a_{pk(Resp)}), sessionkeys(\{key\})\rangle\end{aligned}$$

$$\begin{aligned}&\mathrm{SP(Resp)} = \\ &\langle recv(Init, Resp, \{|\{|Resp, x|\}^a_{sk(Init)}|\}^a_{pk(Resp)}), \\ &state(\{x, \{|Resp, x|\}^a_{sk(Init)}\}), sessionkeys(\{x\})\rangle\end{aligned}$$

# Threads

- Protokolle werden von Agenten ausgeführt, die Rollen ausüben $\rightarrow$ jede Instanz einer Rolle ist ein Thread
- neue Terme und Variablen werden eineindeutig an Thread gebunden mit

$$localize(tid) = \bigcup_{cv \in Fresh \cup Var} [cv\#tid/cv]$$

- formale Definition von Thread:

$$thread(l, tid, \sigma) = \sigma(localize(tid)(l))$$

# Threads

- $\mathrm{SP}(\mathrm{Init}) =$
  $\langle generate(\{key\}), state(\{key, \{|Resp, key|\}^{a}_{sk(Init)}\}),$
  $send(Init, Resp, \{|\{|Resp, key|\}^{a}_{sk(Init)}|\}^{a}_{pk(Resp)}), sessionkeys(\{key\})\rangle$

- $thread(SP(init), t_1, [A, B/Init, Resp]) =$
  $\langle generate(\{key\#t_1\}), state(\{key\#t_1, \{|B, key\#t_1|\}^{a}_{sk(A)}),$
  $send(A, B, \{|\{|B, key\#t_1|\}^{a}_{sk(A)}|\}^{a}_{pk(B)}), sessionkeys(\{key\#t_1\})\rangle$

# Der Test-Thread

- Thread in Berechnungsmodell zum Beweisen von Sicherheitseigenschaften, an dem der Angreifer eine *test query* durchführt
- Substitution der Variablen-/Term-Namen durch $\sigma_{Test}$ und Rolle gegeben als $R_{Test}$
- Beispiel: Alice möchte mit Bob reden führt zu $R_{Test} = Init$ und $\sigma_{Test} = [Alice, Bob/Init, Resp]$

# Zustand eines Systems

- $Trace := (TID \times Event)^*$
- Status des gesamten Systems:
  $(tr, IK, th, \sigma_{Test}) \in Trace \times \mathcal{P}(\text{Term}) \times (TID \nrightarrow Event^*) \times Sub$
- Definition: Für ein Protokoll P ist die Test-Substitution $TestSub_P$ definiert als Menge aller Grundsubstitutionen $\sigma_{Test}$, sodass

$$dom(\sigma_{Test}) = dom(P) \cup \{v\#Test | v \in Var\}$$
$$\text{und}$$
$$\forall r \in dom(P).\sigma_{Test}(r) \in Agent$$

# Zustand eines Systems

- Menge der Anfangszustände für ein Protokoll P:

$$IS(P) = \bigcup_{\sigma_{Test} \in TestSub_P} \{(\langle\rangle, Agent \cup \{pk(a) | a \in Agent\}, \emptyset, \sigma_{Test})\}$$

- im Kontrast zu Dolev-Yao-Modellen kennt der Angreifer noch keine Longterm-Secret-Keys

# Transitionsregeln - Teil 1

$$\frac{R \in dom(P) \quad dom(\sigma) = Role \quad ran(\sigma) \subseteq Agent \quad tid \notin (dom(th) \cup \{tid_A, Test\})}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \frown \langle (tid, create(R, \sigma(R))) \rangle, IK, th[thread(P(R), tid, \sigma) \leftarrow\!\!\mapsto tid], \sigma_{Test})} [create]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \quad Test \notin dom(th)}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \frown \langle (Test, create(R_{Test}, a)) \rangle, IK, th[thread(P(R_{Test}), Test, \sigma_{Test}) \leftarrow\!\!\mapsto Test], \sigma_{Test})} [createTest]$$

$$\frac{th(tid) = \langle send(m) \rangle \frown l}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \frown \langle (tid, send(m)) \rangle, IK \cup \{m\}, th[l \leftarrow\!\!\mapsto tid], \sigma_{Test})} [send]$$

$$\frac{th(tid) = \langle recv(pt) \rangle \frown l \quad IK \vdash \sigma(pt) \quad dom(\sigma) = FV(pt)}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \frown \langle (tid, recv(\sigma(pt))) \rangle, IK \cup \{m\}, th[\sigma(l) \leftarrow\!\!\mapsto tid], \sigma_{Test})} [recv]$$

# Transitionsregeln - Teil 2

$$\frac{th(tid) = \langle generate(M) \rangle \hat{} I}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid, generate(M)) \rangle, IK \cup \{m\}, th[I \leftarrow tid], \sigma_{Test})} [generate]$$

$$\frac{th(tid) = \langle state(M) \rangle \hat{} I}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid, state(M)) \rangle, IK \cup \{m\}, th[I \leftarrow tid], \sigma_{Test})} [state]$$

$$\frac{th(tid) = \langle sessionkeys(M) \rangle \hat{} I}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid, sessionkeys(M)) \rangle, IK \cup \{m\}, th[I \leftarrow tid], \sigma_{Test})} [sessionkeys]$$

# weitere Operationen auf Traces

- Such-Operator für Thread-IDs $\downarrow$

$$(\langle (tid', e) \rangle \hat{} tr) \downarrow tid = \begin{cases} \langle e \rangle \hat{} (tr \downarrow tid) & wenn\ tid = tid' \\ tr \downarrow tid & \text{sonst} \end{cases}$$

- Such-Operator für Event-Typen $\lfloor$

$$(\langle e \rangle \hat{} l) \lfloor evtype = \begin{cases} \langle m \rangle \hat{} (tr \lfloor tid) & wenn\ e = evtype(m) \\ l \lfloor evtype & \text{sonst} \end{cases}$$

# Partner-Threads

- Definition einer sog. *matching History* von 2 Threads als
  $MH(l, l') = (l \downharpoonright recv = l' \downharpoonright send \wedge l' \downharpoonright recv = l \downharpoonright send)$

- Definition von Partnern vom Test-Thread in einem gegebenen Trace als
  $Partner(tr, \sigma_{Test}) =$
  $\{tid \mid$
  $tid \neq Test \wedge (\exists a.create(R, a) \in tr \downarrow tid) \wedge$
  $(\exists l.MH(\sigma_{Test}(P(R_{Test})), (tr \downarrow tid)\hat{}\, l)\}$

# Angreifer

- Kompromittierung in drei Dimensionen möglich
  1. *Welche* Daten werden kompromittiert?
  2. *Wessen* Daten werden kompromittiert?
  3. *Wann* werden die Daten kompromittiert?

# Transitionsregeln für den Angreifer - Teil 1

$$\frac{a \notin \{\sigma_{Test}(R) \mid R \in dom(P)\}}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid_A, LKR(a)) \rangle, IK \cup \{LongTermKeys(a)\}, th, \sigma_{Test})} [LKRothers]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \quad a \notin \{\sigma_{Test}(R) \mid R \in dom(P) \setminus \{R_{Test}\}\}}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid_A, LKR(a)) \rangle, IK \cup \{LongTermKeys(a)\}, th, \sigma_{Test})} [LKRothers]$$

$$\frac{th(Test) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid_A, LKR(a)) \rangle, IK \cup \{LongTermKeys(a)\}, th, \sigma_{Test})} [LKRafter]$$

$$\frac{th(Test) = \langle\rangle \quad tid \in Partner(tr, \sigma_{Test}) \quad th(tid) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid_A, LKR(a)) \rangle, IK \cup \{LongTermKeys(a)\}, th, \sigma_{Test})} [LKRaftercorrect]$$

# Transitionsregeln für den Angreifer - Teil 2

$$[SKR]\quad \frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test})}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid_A, SKR(tid)) \rangle, IK \cup union((tr \downarrow tid) \lfloor sessionkeys), th, \sigma_{Test})}$$

$$[SR]\quad \frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test}) \qquad th(tid) \neq \langle\rangle}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid_A, SR(tid)) \rangle, IK \cup last((tr \downarrow tid) \lfloor state), th, \sigma_{Test})}$$

$$[RNR]\quad \frac{}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr \hat{} \langle (tid_A, RNR(tid)) \rangle, IK \cup last((tr \downarrow tid) \lfloor generate), th, \sigma_{Test})}$$

# Zusammensetzen des Angreifers

- einzelne Eigenschaften unabhängig voneinander
- verschiedene Protokolle haben verschiedene Schwächen
- $\Rightarrow$ jede Teilmenge der 7 genannten Transitionen stellt ein sog. *adversary-compromise model* dar und ist einzeln zu betrachten

# Transitionsbeziehungen eines vollständigen Protokolls

- eine Transitionbeziehung $s \rightarrow_{P,Adv,R_{Test}} s'$ ist gültig, wenn es in den Transitionsregeln des Protokolls oder des Angreifers eine Regel gibt, für die alle Prämissen erfüllt sind und die Konklusion $s \rightarrow s'$ ist
- die Menge aller erreichbaren Zustände $RS(P, Adv, R_{Test})$ ergibt sich daher als

$$RS(P, Adv, R_{Test}) = \{s \mid \exists s_0 . s_0 \in IS(P) \wedge s_0 \rightarrow^*_{P,Adv,R_{Test}} s\}$$

# Sicherheitseigenschaften eines vollständigen Protokolls

- mögliche Definition der Geheimhaltungs-Eigenschaften von Protokollen mithilfe der definierten Operationen
- Beispiel: Geheimhaltung von Sessionkeys:

$$th(Test) = \langle\rangle \Rightarrow \forall k \in union((tr \downarrow Test) \lfloor sessionkeys).IK \not\vdash k$$

- andere Geheimhaltungs-Eigenschaften sind analog definierbar

- *Aliveness* definiert durch:

$$th(Test) = \langle\rangle \Rightarrow \exists R_{Test}, R, tid, a.(Test.create(R_{Test}, a)) \in tr \\ \wedge R \neq R_{Test} \wedge (tid, create(R, \sigma_{Test}(R))) \in tr$$

# Erfüllen von Eigenschaften und Widerstehen von Angriffen

- $\Phi$ als Menge aller Zustandseigenschaften, $\phi$ als einzelne Eigenschaft
- ein Protokoll erfüllt eine Sicherheitseigenschaft gegen einen Angreifer, wenn jeder erreichbare Zustand die Eigenschaft erfüllt, d.h.

$$sat(P, Adv, \phi) : \forall R. \forall s. s \in RS(P, Adv, R) \Rightarrow \phi(s)$$

- zusätzliche Definintion einer partiellen Ordnung von Angreifermodellen:

$$Adv \leq_A \Rightarrow \forall(P), R. RS(P, Adv, R) \subseteq RS(P, Adv', R)$$

# Vorbetrachtungen

- Ordnen von Sicherheitsprotokollen in Bezug auf die Angreifer, denen sie widerstehen
- nutzbar für Entscheidungen nach Anforderungen und Worst-Case-Annahmen über Angreifer
- Problem: *jede* Kombination der Angreifer-Transitionen ist ein gültiger und zu testender Angreifer - $2^7 = 128$ verschiedene Modelle

# Automatische Appromixation von Eigenschaften

- automatische Prüfung hat das Problem der Unentscheidbarkeit
- $\Rightarrow$ Hinzufügen einer zusätzlichen Aussage über Korrektheit einer Eigenschaft: $\perp$
- $\Rightarrow$ Nutzen von rekursiver Approximation von sat/3

$$f \in Protokoll \times A \times \Phi \rightarrow \{F, \perp, V\}$$
$$f(P, Adv, \phi) \neq \perp \Rightarrow (f(P, Adv, \phi) = V \Leftrightarrow sat(P, Adv, \phi))$$

# Definition von Hierarchie

- mithilfe der rekursiven Funktion ist eine Definition einer Hierarchie möglich
- Hierarchie als gerichteter Graph $H = (N, \rightarrow)$ mit folgenden Eigenschaften:
  1. N ist eine Zerlegung von H mit $\bigcup_{\pi \in N} = \Pi$ und für alle $\pi, \pi' \in N$ gilt $\pi \neq \emptyset$ und $\pi \neq \pi' \Rightarrow \pi \cap \pi' = \emptyset$
  2. $f$ ist abgeschlossen unter N, d.h.

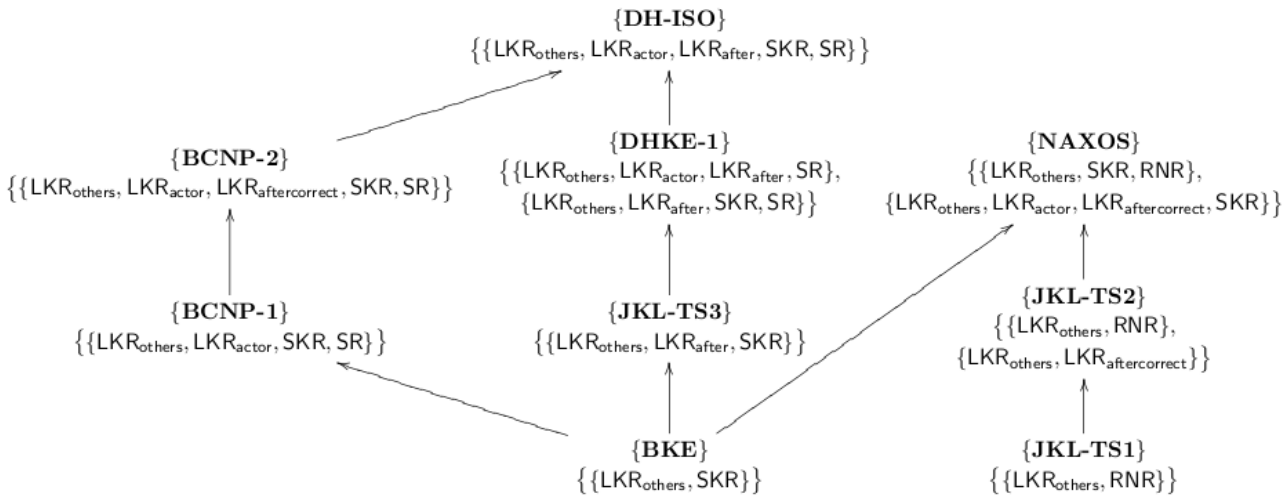
$$(\exists \pi \in N.\{P, P'\} \subseteq \pi) \Leftrightarrow \forall Adv \in A.f(P, Adv, \phi) = f(P', Adv, \phi)$$

  3. $\pi \rightarrow \pi'$ gilt nur, wenn
     $\forall P \in \pi.\forall P' \in \pi'.\forall Adv \in A.$
     $f(P, Adv, \phi) = V \Rightarrow f(P', Adv, \phi) = V \wedge$
     $f(P', Adv, \phi) = F \Rightarrow f(P, Adv, \phi) = F$

# Definition einer partiellen Ordnung auf Basis der Hierarchie

- Lemma: Sei $H = (N, \rightarrow)$ eine Protokoll-Sicherheits-Hierarchie. Sei $\leq_H$ definiert als "Für alle $\pi, \pi' \in N$ gilt $\pi \leq_H \pi'$, wenn $\pi \rightarrow \pi'$." Dann ist $\leq_H$ eine partielle Ordnung.

# Beispielhierarchie von Sicherheitsprotokollen



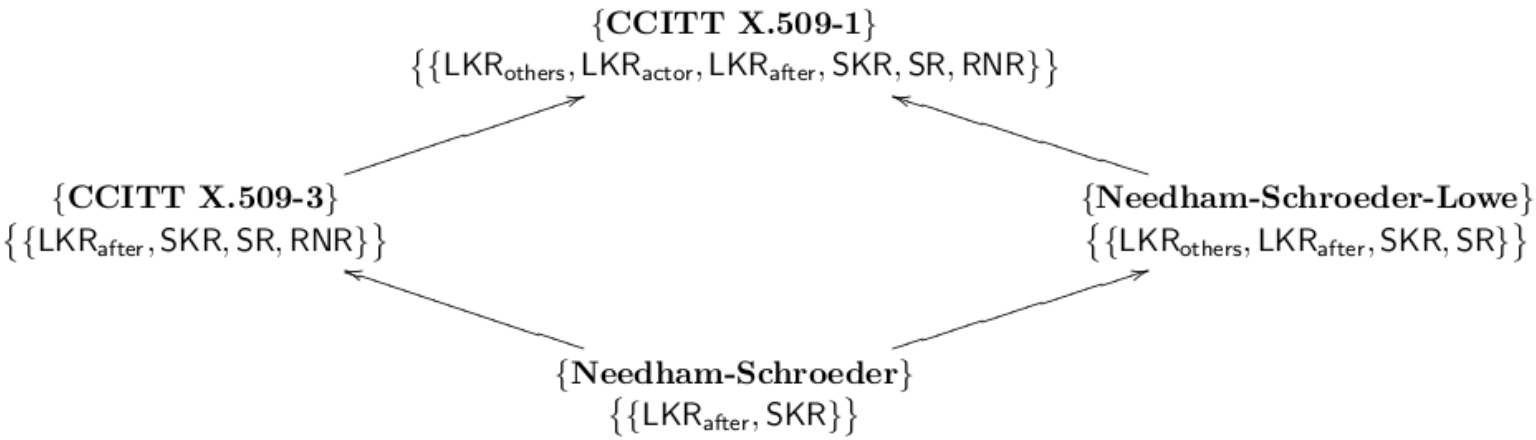

- die annotierten Angreiferfähigkeiten sind angegeben, wenn

$$\forall a' \in A, P \in \pi.(a <_A a' \Rightarrow f(P, a', \phi) = F) \wedge$$
$$(a' \leq_A a \Rightarrow f(P, a', \phi) \neq F)$$

# Einschränkungen der Hierarchie

- nicht alle Kanten basieren vollständig auf sat/3, da die Ausgaben von $f$ nur partiell sind
- Definieren von *Strictness* von Kanten $\pi \rightarrow \pi'$:
  1. Protokolle in $\pi'$ sind mindestens so stark wie Protokolle in $\pi$:
     $\forall P \in \pi, P' \in \pi'. \forall Adv \in A.$
     $f(P, Adv, \phi) \neq F \Rightarrow f(P', Adv, \phi) = V$
  2. Protokolle in $\pi$ sind nicht genauso stark wie Protokolle in $\pi'$:
     $\forall P \in \pi, P' \in \pi'. \exists Adv \in A.$
     $f(P, Adv, \phi) = F \wedge f(P', Adv, \phi) = V$

# Hierarchie von Authentifikationsmethoden



# Erweiterung von Regeln

- Lemma: Ein Hinzufügen von Regeln macht den Angreifer nur stärker:

$$Adv \leq_A Adv \cup \{r\}$$

- Lemma: Die Angreifer-Transitionen sind unabhängig voneinander:

$$(r = LKR_{aftercorrect} \wedge LKR_{after} \in Adv) \Leftrightarrow (Adv \backslash \{r\} =_A Adv \cup \{r\})$$

# Testunabhängige Eigenschaften

- Definition: $\Phi_{PTI}$ als Menge aller testunabhängigen Eigenschaften:

$$\forall P, R, Adv. \forall state \in RS(P, Adv, R). th(Test) = \langle\rangle \Rightarrow \forall (s).(tr, IK, th, \sigma_{Test}) \rightarrow^*_{P,Adv,R_{Test}} s \Rightarrow (\phi((tr, IK, th, \sigma_{Test})) \Leftrightarrow \phi(s))$$

- Theorem: Testunabhängige Eigenschaften widerstehen zukünftigen Fähigkeiten:

$$r \in \{LKR_{aftercorrect}, LKR_{after}\} \wedge sat(P, Adv, \phi) \Rightarrow sat(P, Adv \cup \{r\}, \phi)$$

# Schlussbetrachtungen

- Möglichkeit der formalen Analyse von Sicherheitsprotokollen durch Modellierung von wenigen Regeln
- Ermöglichung und Einsatz von Analyse-Tools zur automatisierten Verarbeitung
- Erstellung einer Sicherheitshierarchie