Der NRL Protocol Analyzer
-
Seminar Analyse kryptographischer Protokolle

Stephan Allner*
Institut für Informatik
Humboldt-Universität zu Berlin

22. Januar 2003

Zusammenfassung

Die Ausarbeitung ist im Rahmen des Seminars „Analyse kryptographischer Protokolle“ entstanden und beschäftigt sich mit dem NRL Protocol Ananlyzer, einem interaktiven Tool zur Verifizierung kryptographischer Protokolle. Es basiert vorwiegend auf dem Modell von 1992 und versucht die grundlegende Funktionsweise des Analyzers zu beschreiben. Er wurde seitdem jedoch weiterentwickelt und ist aktuell noch in Gebrauch.

Inhaltsverzeichnis

*allner@informatik.hu-berlin.de

1 Einführung

Der „NRL Protocol Analyzer“ ist ein in Prolog geschriebenes interaktives Tool um kryptographische Protokolle auf deren Sicherheit zu überprüfen. Er wurde bzw. wird von Catherine Meadows entwickelt, wobei er erstmals 1994 erwähnt wird. Methoden die in dem Analyzer verwendet werden, wurden aber auch schon früher von ihr, mit der Bemerkung, daß man an einem entsprechenden Tool arbeiten würde, in Ausarbeitungen beschrieben.

Kryptographische Protokolle benutzen kryptographische Elemente, wie Verschlüsselung, um z.B. sicher Schlüssel zu verteilen (key distribution) oder Teilnehmer und Dienste zu authentifizieren. Dies soll über Netzwerke passieren, die möglicherweise von Angreifern belauscht werden, welche genau dies verhindern wollen. So z.B. kann ein Angreifer versuchen bei der key distribution den session key zu erhalten oder einen selbst erstellten key als session key zu verteilen. Den Angreifern wird dies meist durch Fehler in den Protokollen und weniger durch Fehler in den Verschlüsselungsalgorithmen oder durch deren Schwäche ermöglicht. In der Vergangenheit wurden immer wieder Fehler in solchen Protokollen gefunden, nachdem sie veröffentlicht oder gar schon lange in der Verwendung waren und als sicher galten. Um dies zu verhindern und somit ein Missbrauch vorzubeugen versucht man die kryptographischen Protokolle auf deren Sicherheit zu überprüfen. Da informale Analysen oft auch schon bei einfachen Protokollen nicht ausreichen, werden schon seit einiger Zeit formale Techiken zur Verifizierung solcher Protokolle entwickelt. Dazu gibt es verschieden Ansätze, welche mehr oder weniger die „reale Welt“ und die Möglichkeiten des Angreifers abbilden können und überprüfen, ob das Protokoll seine Ziele erfüllt. Diese lassen sich grundlegend in folgende Klassen unterteilen:

1. Logiken des Wissens und Glaubens (belief) (z.B. die BAN-Logik)
2. Algebren (Das Protokoll als algebraisches System modellieren)
3. Zustandsmaschinen (Das Protokoll als Interaktion zwischen einer Menge von Zustandsmaschinen modellieren)

Der „NRL Protocol Analyzer“ verbindet, ähnlich wie der Interrogator oder das Modell von Dolev und Yao[4], auf dem es zudem basiert, den algebraischen Ansatz mit den der Zustandsmaschinen. Er kann verwendet werden um sicherheitsrelevante Eigentschaften eines Protokolles zu überprüfen, als auch Fehler in diesen zu finden.

2 Grundlagen des Modells

Wie im Modell von Dolev und Yao wird davon ausgegangen, daß der Angreifer den gesamten Netzwerkverkehr mitlesen, Nachrichten zurückverfolgen, verändern und löschen und auch selbst eigene Nachrichten erzeugen kann. Weiterhin kann er Funktionen die normalen Nutzern zur Verfügung stehen, wie Ver- und Entschlüsselung, ausführen und evtl. mit einigen Nutzern kooperieren. Es wird jedoch angenommen, daß er einige Mengen von Wörtern nicht kennt und es sein Ziel ist, diese herauszufinden. Dies sind z.B. keys von ehrlichen (honest) Nutzern oder verschlüsselte Nachrichten.

Aus der Tatsache, daß er volle Kontrolle über den Netzverkehr hat, folgt daß jede Nachricht, die ein Teilnehmer erhält, vom Angreifer stammen kann. Daher kann man sich das Protokoll als ein algebraisches System vorstellen, daß der Angreifer manipuliert. Also als eine Menge von Regel zum Erzeugen von Wörtern in einer formalen Sprache. Die Wörter und damit die Regeln, die die Wörter erzeugen, unterliegen einigen Reduktionsregeln, z.B. daß sich Ver- und Entschlüsselung gegenseitig aufheben. Der Angreifer versucht nun herauszufinden, ob ein Wort zu der Sprache gehört oder nicht. Wenn der Angreifer nun kein geheimes Wort aus einer vorbestimmten Menge (z.B. den key eines Nutzers) erhalten kann, ist das Protokoll sicher (analog zum Modell von Dolev und Yao).

Als Erweiterung zum Modell von Dolev und Yao ist es nicht nur Ziel des Angreifers ein solches geheimes Wort zu finden, sondern auch, daß interne Zustandsvariablen bestimmte Werte anzunehmen, also einen einen Teilnehmer davon zu überzeugen, daß ein Wort bestimmte Eigenschaften hat, die es nicht besitzt. Ein Protokoll ist daher unsicher, wenn Angreifer nicht in der Lage ist ein geheimes Wort zu lernen und wenn er ein Wort lernen oder generieren kann und einem ehrlichen Nutzer zu überzeugen, daß das Wort Eigenschaften eines geheimen Wortes besitzt, so z.B. ein von ihm generiertes Wort ein Sessionkey ist.

Ein weiterer Unterschied zum Dolev und Yao Modell ist, daß keine Protokollklassen definiert und effiziente Algorithmen zu deren Verifizierung bestimmt werden, sondern eine allgemeinere Prozedur für verschiedene Arten von Protokollen (key managment- und Authentifizierungsprotokolle) entwickelt wurde.

Es folgt nun eine formale Beschreibung des Modells nach [1], wobei dieses Paper von 1992 stammt und nicht alles abdeckt, was später im Analyzer implementiert wurde, zudem wird der Analyzer dort noch nicht als „NRL Protocol Analyzer“ bezeichnet. Deswegen beschreibe ich im darauffolgenden Kapitel den Analyzer nach [3]. Dies soll nur noch einmal die Verwendung des Analyzers veranschaulichen, weswegen ich dort nicht zu sehr ins Detail gehen werde.

3 Beschreibung des Modells

Aus der Tatsache, daß der Analyzer in Prolog implementiert ist, ergibt sich daß Variablen mit einem großen Buchstaben und Konstanten und Funktionssymbole mit einem Kleinen beginnen, sowie daß alle Variablen universell quantifiziert sind (z.B. für do(X) bedeutet: für alle X do(X)). Die selben Konventionen gelten daher auch für die Protokollspezifizierung und für Beispiele in den folgenden Abschnitten. Hinweis

Desweiteren wird die Verschlüsselung einer Nachricht Y mit einem Schlüssel X, wird mit e(X,Y) angegeben und die Entschlüsselung einer Nachricht Y mit einem Schlüssel X als d(X,Y).

In dem Modell werden Protokollregeln in der Form

$$\text{If } I \subseteq \mathbf{W} \text{ and } A \text{ and } \mathrm{Cond}(I, A) \text{ then } \mathbf{W} := \mathbf{W} \cup O \text{ and } A'$$

ausgedrückt, wobei

- **W** die Menge der Wörter ist, die der Angreifer kennt.
- I (in) die Wörter einer Nachricht angibt, die zu einem Teilnehmer geschickt wurden.
- O (out) die Wörter einer Nachricht angibt, die von einem Teilnehmer geschickt wurden.
- A und A' sind Anweisungen, die aus Gleichungen der Form $T = D$ bestehen, wo T der Name einer Zustandsvariablen ist.
- $\mathrm{Cond}(I, A)$ ist eine Menge weiterer Bedingungen an I und A der Form: $C \neq E$, wobei C ein Subterm aus I oder A ist und E ein Term.

Solche Regeln können verschiedene Ereignisse beschreiben, wie

- einen Teilnehmer, der auf eine Nachricht des Angreifers antwortet;
- die Veränderung einer internen Zustandsvariable eines Teilnehmers;
- ein Teilnehmer, der versucht den Prokollablauf zu initiieren.
- den Angreifer, welcher mit dem ihn zur Verfügung stehenden Mitteln Nachrichten generiert.

Beispiel 1

Eine Regel für einen Angreifer, der in der Lage ist, Daten zu verschlüsseln, kann so dargestellt werden:

$$\boxed{\text{If } \{X, Y\} \subseteq \mathbf{W} \text{ then } \mathbf{W} := \mathbf{W} \cup \{e(X, Y)\}}$$

Anmerkung Wie wir in Abschnitt 4 sehen werden, ist eine solche Angabe in der echten Spezifikation nicht nötig, da diese sich etwas von den theoretischen Grundlagen unterscheidet und solche Regeln von Analyzer selbstständig generiert werden.

Beispiel 2

Nehmen wir an ein Teilnehmer sendet alle Nachrichten die er bekommt mit dem Sessionkey verschlüsselt weiter, so kann man das mit folgender Regel ausdrücken.

$$\boxed{\text{If } \{Y\} \subseteq \mathbf{W} \text{ and KEYSTATE}(a) = X \text{ then } \mathbf{W} := \mathbf{W} \cup \{e(X, Y)\}}$$

Neben den Regeln besteht ein Protokoll in diesem Modell aus einer Menge von Reduktionsregeln, welche die Umformung komplexerer Wörter in Einfachere beschreiben. Beispiele dafür sind

1. $d(X, e(X, Y)) \to Y$, da sich Ver- und Entschlüsselung mit dem selben Key aufheben
2. $p(s(X)) \to X$, wobei s und p die Nachfolger- und Vorgängeroperationen (successor und predecessor) sind.

Definition 1: *Spezifikation* **S** eines Protokolls ist ein Tupel

$[T(\Phi, F), \mathcal{E}, \mathbf{S}, \mathbf{R}, W_0, A_0]$ wobei

- Φ eine abzählbare Menge von Variablen ist;
- F eine endliche Menge an Funktionssymbolen ist;
- $T(\Phi, F)$ die Menge von Termen ist, die durch Φ und F gebildet werden kann;
- $\mathcal{E}$ ist eine Menge an Umschreiberegeln (rewrite rules)
- **S** eine Menge von Zustandsvariablen ist;
- W_0 die Menge der Wörter aus $[T(\Phi, F)$ ist, die der Angreifer zu Beginn kennt;
- A_0 eine Menge von anfänglichen Zuweisungen von Werten auf Zustandsvariablen;

und **R** eine Menge von Protokollregel der Form

$$\boxed{\text{If } I \subseteq \mathbf{W} \text{ and } A \text{ and Cond}(I, A) \text{ then } \mathbf{W} := \mathbf{W} \cup O \text{ and } A'}$$

wie oben, jedoch mit den Einschränkungen, daß

- I und O nur aus Termen von $T(\Phi, F)$ besteht;
- A und A' Anweisungen aus Gleichungen der Form $T = D$ bestehen, wo T der Name einer Zustandsvariablen ist und $D \in T(\Phi, F)$.

Definition 2: Term, Konstanten

Sei Φ eine abzählbare Menge an Variablen und F eine endliche Menge an Funktionssymbolen disjunkt von Φ mit zugehöriger arity (Anzahl der Funktionsparameter). Dann ist ein *Term* entweder eine Variable aus Φ oder ein Funktionsymbol gefolgt von n Termen, wobei n die arity des Funktionssymbols angibt. Ein Funktionssymbol mit einer arity von null ist eine *Konstante*.

Definition 3: Substitution, Unifikator

Sei $T(\Phi, F)$ die Menge von Termen ist, die durch Φ und F gebildet werden kann. Dann ist eine *Substitution* eine Funktion von Φ in $T(\Phi, F)$. Der Term den man durch Anwendung einer Substitution σ auf die Variablen eines Terms T erhält, wird durch σT angegeben.

Ein *Unifikator* von zwei Termen T und U ist eine Substitution σ, so daß $\sigma T = \sigma U$. Es werden also 2 Terme durch entsprechende Substitution in den gleichen Term umgewandelt. Der allgemeingültigste Unifikator von T und U ist ein Unifikator σ, so daß, wenn es einen weiteren Unifikator τ gibt, es noch ein weiter Unifikator μ existiert, so daß gilt $\tau = \mu\sigma$.

Definition 4: term-rewriting system, Normalform

Ein term-rewriting system über $T(\Phi, F)$ ist eine Menge gerichteter Gleichungen $\mathcal{E}$, so daß für $T_1 \rightarrow T_2$ in $\mathcal{E}$, $V(T_2) \subseteq V(T_2)$ gilt, wobei V(T) die Menge der Variablen in T angibt.

Wenn ein Term A in einen eindeutigen nicht reduzierbaren Term durch die Anwendung von Reduktionsregeln gebracht werden kann, so nennt man den erzeugten Term Normalform von A (nf(A)).

Für den Analyzer sind term-rewriting systems interessant, die Noetherian und locally confluent sind. Diese haben die Eigenschaft, daß jeder Term A zu einer eindeutigen nicht mehr weiter reduzierbaren Normalform nf(A) reduzierbar ist.

Definition 5: Noetherian, locally confluent

Sei $\mathcal{E}$ ein term-rewriting sytem über $T(\Phi, F)$. $\mathcal{E}$ ist *Noetherian*, wenn ein Term A, der von einem Term B mittels der Anwendung von Reduktionsregeln erreichbar ist, innerhalb einer endlichen Anzahl von Schritten erreicht wird.

$\mathcal{E}$ ist *locally confluent*, falls wennimmer ein Term A zu den Termen B und C mittels zwei verschiedener Reduktionsregeln reduzierbar ist, dann sind B und C beide zu einem gemeinsamen Term D reduzierbar.

Beispiel 3

Solch ein term-rewriting system kann zum Beispiel wie folgt aussehen. Es handelt sich wieder um die triviale Aufhebung von Ver- und Entschlüsselung mit dem selben Key.

$d(A, e(A, B) \rightarrow B$
$e(A, d(A, B) \rightarrow B$

Es folgen Grundlagen für das Zustandsmodell, es wird beschrieben, was ein Systemzustand und was eine Zustandsbeschreibung ist und wie man eine Zustandsbeschreibung von einer Anderen erreicht.

Definition 6: Systemzustand und Zustandbeschreibung

Ein *Systemzustand* ist ein Paar (Λ, Δ), wobei Λ die Menge der Wörter ist, die der Angreifer kennt und $\Delta = \gamma_1 = \beta_1, \ldots, \gamma_k = \beta_k$ den Zustandsvariablen (γ_l) die Werte (β_l) zuordnet.

Eine *Zustandsbeschreibung* (L, D) ist ein Paar, so daß L eine Menge an Termen aus $T(\Phi, F)$ ist und $D = G_1 = D_1, \ldots, G_t = D_t$ ist die Zuordnung einer Menge von Termen aus $T(\Phi, F)$ (D_i) zu den Zustandsvariablen (G_i).

Ein Systemzustand (Λ, Δ) *genügt* einer Zustandsbeschreibung (L, D), wenn alle Terme in (Λ, Δ) nicht reduzierbar sind und es eine Substitution σ gibt, so daß $\sigma L \subseteq \Lambda$ und $\sigma D \subseteq \Delta$.

Beispiel 4

Nehmen wir die Zustandsbeschreibung (e(k,X),). Dann können wir annehmen, daß der Angreifer ein word e(k,σX) kennt, wobei die Substitution X einen Term zuweist, so daß e(k,σX) nicht weiter reduziert werden kann. Daher könnte er e(k,m) oder e(k,e(l,m)) kennen, nicht aber e(k,d(k,m)).

Die Annahme, daß Term (im Beispiel e(k,σX)) nicht weiter reduzierbar ist, ist daher wichtig, weil wenn der Angreifer e(k,σX) kennt und X=d(k,Y) sein könnte, sich das ganze zu Y reduzieren würde und man daraus erfahren würde, daß der Angreifer irgendein Y kennt. Und zu der Zustandsbeschreibungdiese nutzlose Information hinzukommen würde.

Eine Protokollregel kann nun interpretiert werden, um von einer Zustandsbeschreibung zu einer Weiteren zu gelangen. Die Bedingungen der Regel müssen von einer Zustandsbeschreibung erfüllt sein, damit die Regel angewandt wird. Mit der Folgerung aus der Regel wird die neue Beschreibung konstruiert.

Definition 7: Übergange zwischen Zustandsbeschreibungen

Seien (L, D) und (L', D') Zustandsbeschreibungen. Dann sei R eine Regel der bekannten Form

$$\boxed{\text{If } I \subseteq \mathbf{W} \text{ and } A \text{ and } \mathrm{Cond}(I, A) \text{ then } \mathbf{W} := \mathbf{W} \cup O \text{ and } A'}$$

,so daß eine Substitution σ existiert, die den Variablen von R einen Term zuordnet und die Terme in (L, D), (L', D'), I, A so umformt, daß diese nicht weiter reduzierbar sind und gilt

1. $\sigma I \subseteq \sigma L$
2. $\sigma A \subseteq \sigma D$
3. Cond$(\sigma A, \sigma I)$ enthält keine Ungleichungen der Form $X \neq X$ (not contradicted)
4. $\sigma L' \subseteq \sigma L \cup nf(\sigma O)$ („die übertragenen Wörter werden den, dem Angreifer bekannten Wörtern, hinzugefügt“)
5. $\sigma D' \subseteq (\sigma D - \sigma D'') \cup nf(\sigma A')$, wobei $\sigma D''$ die Menge der Zuweisung aus σD, die durch neue Zuweisung in $nf(\sigma A')$ ersetzt wird. („die neuen Werte für lokalen Zustandsvariablen werden, die aus der Regel folge, werden in der Zustandsvariablenliste des Zielzustandes gesetzt“)
6. Entweder $\sigma L' \cap nf(\sigma O)$ oder $\sigma D' \cap nf(\sigma A')$ ist nicht leer. (Um triviale Übergänge zu verhindern; Es würden keine Terme oder Zustandsvariablen durch eine solche Regel hinzugefügt/geändert.)

Die Zustandsbeschreibung (L', D') ist nun von der Zustandsbeschreibung (L, D) mittels der Regel

R zu erreichen. P' ist von P mittels der Regeln $R_1, \ldots, R_k$ erreichbar, wenn es eine Reihe von Zustandsbeschreibungen $P_0, \ldots, P_k$ gibt, so daß P' von P_k mittels R_k erreichbar ist und so weiter.

Beispiel 5

Gehen wir davon aus, daß der Angreifer das Wort "message" kennt und KEYSTATE(a):=key1, dann kann die Regel

$$\boxed{\text{If } Y \subseteq \mathbf{W} \text{ and KEYSTATE}(Z) = X \text{ then } \mathbf{W} := \mathbf{W} \cup e(X,Y)}$$

mit der Substitution Y=message, Z=a und X=key1 angewendet werden. Man gelangt dann von der Zustandsbeschreibung ({message}, {KEYSTATE(a)=key1}) zu ({message, e(key1,message)}, {KEYSTATE(a)=key1}).

Man kann nun verfolgen wie durch Anwendung der Prokollregeln man von einem Zustand zum nächsten gelangt. Um zu Bestimmen, ob ein Protokoll sicher ist soll gezeigt werden, daß man von bestimmten Zuständen nicht den Initialzustand erreichen kann. Was nun folgt ist genau der umgekehrte Schritt. Es muss anhand der Zustandsbeschreibung bestimmt werden, welche andere Zustandsbeschreibungen die Zustände beschreibt, die direkt davor gewesen sein könnten. Dazu wird im Prinzip die Folgerung jeder Regel mit der Zustandsbeschreibung unifiziert.

Sei die Zustandsbeschreibung (L, D) gegeben. Man sucht nun eine Substitution σ, die I, A und (L, D) so umformt, daß diese nicht weiter reduzierbar sind und im weiteren gilt

1. sich entweder irgendeine Teilmenge von σO zu einer Teilmenge von σL oder irgendeine Teilmenge von $\sigma A'$ sich zu einer Teilmenge von σD reduziert.
2. Cond($\sigma A, \sigma I$) enthält keine Ungleichungen der Form $X \neq X$ (not contradicted)
3. $\sigma A'$ reduziert sich zu einem Ausdruck, der D nicht wiederspricht.
4. Wenn σD eine Anweisung $\sigma(G_i = D_i)$ enthält und σA dieser Anweisung wiederspricht, dann muss $nf(\sigma A')$ diese Anweisung enthalten.

Daraus ergibt sich ein vorheriger Zustand (L', D'), in dem der Angreifer $\sigma I \cup (\sigma L - nf(\sigma O))$ kennt und die Menge der Zustandsvariablen $\sigma A \cup (\sigma D - \sigma D'')$ entspricht, wobei $\sigma D''$ alle Zuweisung aus $nf(\sigma A')$ enthält.

Der Analyzer muss nun alle möglichen Substitutionen σ finden, die die Zustandsbeschreibung mit der Folgerung einer Regel unifiziert. Davon gibt es oft eine unendliche Menge, jedoch eine endlich, ausreichende Menge (complete set), mit deren Hilfe man alle anderen Unifikatoren bilden kann. Zu der Bestimmung solcher Mengen gibt es einige Algorithmen. Da von dem term-rewriting Sytem verlangt wird, daß es Noetherian und locally confluent ist, kann man mit Hilfe eines Algorithmus aus der Klasse der „narrowing algorithms" diese Menge bestimmen. Im Fall des Analyzers wurde eine modifizierte Version des NARROWER Algorithmus [5] eingesetzt, um anhand einer Protokollbeschreibung und der Beschreibung eines Systemzustandes, die Menge der Beschreibungen zu finden, die diesem direkt vorangehen können.

Beispiel 6

Nehmen wir als Beispiel eine Spezifikation mit der einen Regel

$$\boxed{\text{If } Y \subseteq \mathbf{W} \text{ and KEYSTATE}(Z) = X \text{ then } \mathbf{W} := \mathbf{W} \cup e(X,Y)}$$

und fragen den Analyzer danach, wie der Angreifer ein Wort Q finden kann. Dann sucht das Programm nach einer ausreichenden Menge der Unifikatoren mit e(X,Y) und dem term-rewriting system

$d(A, e(A, B) \to B$
$e(A, d(A, B) \to B.$

Es werden dann 2 Elemente der Menge gefunden und zwar Q:=e(X,Y) und Y:=d(X,Q). Wendet man die 1. Unifikation an erhält man die Bedingungen $Y \subseteq \mathbf{W}$ and KEYSTATE(Z) = X und der Angreifer lernt das Wort Q:=e(X,Y), der Vorgängerzustand war demnach (Y, KEYSTATE(Z) = X). Bei der 2. erhält man $d(X, Q) \subseteq \mathbf{W}$ and KEYSTATE(Z)=X und der Angreifer lernt das Wort Q (e(X,Y)=e(X,d(X,Y))), der Vorgängerzustand ist demnach (d(X,Q), KEYSTATE(Z) = X)

Beispiel 7

Als zweites Beispiel dieselbe Spezifikation mit der einen Regel

$$\boxed{\text{If } Y \subseteq \mathbf{W} \text{ and KEYSTATE}(Z) = X \text{ then } \mathbf{W} := \mathbf{W} \cup e(X, Y)}$$

und fragen den Analyzer danach, wie der Angreifer das Wort e(A,B) lernen kann, also den Zustand (e(A,B),) erreicht. Dann werden die Substitutionen 1. X=A,Y=B und 2. Y=d(X,e(A,B) gefunden. Für den ersten ergibt sich der Zustand (B,KEYSTATE(Z)=A) und für den zweiten (d(X,e(A,B)),KEYSTATE(Z)=A).

Ziel des Programmes ist es nun zu zeigen, daß ein Startzustand nicht von einem definierten unsicheren Zustand zu erreichen ist. Würde man das ganze ohne Einschränken durchführen, würde schnell ein unüberschaubare Menge an Zuständen entstehen, meist in Endlosschleifen endend. Man braucht also zum einen Techniken um die Suche einzuschränken und zum anderen zu Zeigen, daß ein Zustand der irgendwo in einer Endlosschleife entsteht kein Startzustand sein kann. Um ein Protkoll zu analysieren, wird nun eine Protokollspezifikation erstell, die unsicherne Zustände bestimmt und gezeigt, daß ein Menge von Zuständen unerreichbar ist. Dann wird über die restlichen Zustände ein komplette Suche laufen gelassen. Hier wird nun noch beschrieben, wie Zustände mit Hilfe formaler Sprachen als unerreichbar bestimmt werden.

3.1 Begrenzung des Suchraumes durch formale Sprache

Die Endlosschleifen in denen das Programm landet, durchlaufen fast nie immer wieder die selbe Menge an Zuständen, statt dessen passiert meist folgendes. Sucht man nach einem Wort t und der Analyzer gibt aus, daß dies nur gefunden werden kann, wenn der Angreifer zuvor e(X,t) kennt. Fragt man nun nach diesem Wort und der Analyzer findet, daß das nur mit dem zuvorigen Wissen von e(X2,e(X,t)) geht. Dies geht dann so weiter. Es ist ersichtlich, daß er die Wörter nicht finden kann und statt dessen in eine Endlosschleife gerät. Für das Beispiel kann man nun die formale Sprache E mit folgenden Erzeugenden entwickeln.

1. $\mathbf{E} \to t$,
2. $\mathbf{E} \to e(\mathbf{X},\mathbf{E})$

wobei $\mathbf{X}$ die Sprache aller nicht reduzierbaren Wörter über $T(\Phi, F)$ ist. Zu zeigen ist nun, daß jedes nicht reduzierbare Wort aus **E** nur dem Angreifer bekannt sein kann, wenn er vorher schon ein anderes Wort der Sprache **E** kennt. Damit ist kein Wort der Sprache **E** erreichbar, und somit auch kein Zustand in dem der Angreifer solch ein Wort kennen müsste. Der Analyzer wird nun auf jeden Erzeugenden der Sprache **E** angewandt, um zu zeigen, daß nicht reduzierbare Wörter der Sprache erzeugt werden. Im Beispiel ist dies trivial und schon gezeigt. Nun kann man bei der Analyse alle solche Zustände ignorieren, die ein solches Wort enthalten oder zu denen man nur gelangt, wenn man in einem vorrigen Zustand solch ein Wort wissen müsste. Sind alle Zustände die zu einem sicheren Zustand führen, unereichbar, so ist das Protokoll für diesen Angriff sicher.

4 Beschreibung des Analyzers

Dieser Abschitt beschreibt kurz den Analyzer, wie er 1994 implementiert war. Das ursprüngliche Modell, wie oben beschrieben, wurde erweitert. Außerdem wird beschrieben, wie die Spezifikation für das Programm aussieht.

Im Gegensatz zu anderen Tool ist der Analyzer weniger automatisiert, wie schon weiter oben gesehen muss der Nutzer die Bearbeitung beeinflussen, dennoch gibt es einen automatischen Modus und immer mehr Teile der Analyse wurden automatisiert.

Die wichtigste Erweiterung in der Protokollspezifikation ist es, daß bei den Übergangsregeln die Ereignisse, die Regel ausführt, mitprotokolliert werden, welche die Übergänge repräsentieren. Diese Events sind dann auch Teil der Zustandsbeschreibung und, damit eine Regel benutzt werden kann um einen Vorgängerzustand zu finden, werden auch die Events der Beschreibung mit dem Event der Regel unifiziert. So kann man nicht nur nachfragen, wie der Angreifer ein Wort findet, sondern wie ein bestimmtes Event stattfinden konnte, etc..

4.1 Spezifizierung eines Protokolls für den Analyzer

Im Analyzer wird das Protokoll als eine Menge von Zustandsübergangsregeln abgebildet, die das Senden und Empfangen von Nachrichten beschreiben. Wobei die Wörter in den Nachrichten natürlich einigen Reduktionsregeln unterliegen. Dies wird vom Nutzer in vier Abschnitten spezifiziert:

1. Übergangsregeln, die das Verhalten der ehrlichen (honest) Teilnehmer bestimmen
2. Operationen die den Teilnehmern zur Verfügung stehen
3. die Wörter die als grundlegende Bausteine dienen !!!
4. die Reduktionregeln, die von den Operationen ausgeführt wird

4.1.1 Übergangsregeln

Die Regeln sind ähnlich, wie oben beschrieben aufgebaut, jedoch durch die events erweitert worden.

Vorbedingungen (if) Gibt an welche Worte der Angreifer wissen muss, die Werte der Zustandsvariablen (lfacts) die gesetzt sein müssen und weitere Bedingungen die erfüllt sein müssen, damit die Regel wirksam wird.

Folgerung (then) Die Bedingungen die erfüllt sind, nachdem die Regel wirksam geworden ist. Gibt die Wörter an die der Angreifer gelernt hat, und gibt Zustandsvariablen neue Werte. Außerdem wird, wenn die Regel ausgeführt wird, ein Zähler des Teilnehmers erhöht, um die Protokollschritte mitverfolgen zu können.

Event-Anweisung Dies ist hier der wesentlichste Unterschied zu der obigen Beschreibung. Die Events werden benutzt um das Auslösen der Regel zu speichern. Sie beschreiben, was die Regel macht. Diese Event-Beschreibung muss natürlich mit den Rest der Regel übereinstimmen. Die Eventanweisung ist ein Funktion mit vier Parameter, der eine Liste relevanter Wörter übergeben wird. Der erste Paramter gibt den Teilnehmer an, der das Event ausführt. Der Zweite, gibt den Zähler des Teilnehmers vor dem Auslösen der Regel und der Vierte den Zähler danach. Der Dritte Parameter dient dazu das Event zu identifizieren (z.B. als Sende eines Schlüssels).

Ähnlich wie die events sind die lfacts aufgebaut. Sprich mit einem Bezeichner zur Identifizierung, dem zugehörigen Nutzer, und den beiden Zählerständen. Die Regeln und ihre Substitution werden dadurch erheblich unübersichtlicher, weswegen ich auf weiterführende Beispiele (ganzer Protokollspezifikationen und dem Finden eines Fehlers) verzichten werde. Zur Veranschaulichung nur kurz ein Beispiel für solch eine Regel. Nutzer können entweder honest oder dishonest sein. Ist der dishonest wird davon ausgegangen, daß er mit dem Angreifer kooperiert und dem Angreifer somit seine Möglichkeiten zur Verfügung stehen.

```
rule(1)
If:
count(user(A,honest)) = [N],
then:
count(user(A,honest)) = [s(N)],
intruderlearns([pke(pubkey(server),rand(user(A,honest),N)),
user(A,honest), user(B,W)])
lfact(user(A,honest),N,step1,s(N))=[user(B,W),rand(user(A,honest),N)]
EVENT:
event(user(A,honest),N,initrequest,s(N))=[user(B,W),rand(user(A,honest),N)]
```

user(A,honest) versucht Verbindung mit einem anderen Nutzer aufzunehmen, dazu verschickt er an einen Keyserver mit dessen öffentlichen Schlüssel verschlüsselt eine Zufallszahl und außerdem noch die Namen beider Nutzer. Er merkt sich in seinen Zustandvariablen, daß er user(B,W) Verbindung aufnehmen wollte und die entsprechende Zufallszahl, welche später als Key verwendet werden soll.

4.1.2 Operationen

Hier werden die Operationen als Funktionssymbole definiert, die von ehrlichen Teilnehmern als auch von Angreifern ausgeführt werden können. Kann sie vom Angreifer ausgeführt werden, so wird sie mit pen markiert, andernfalls mit nopen. Mit Hilfe der Operationen die der Angreifer ausführen kann, werden weiter Übergansregeln definiert, die zuvor (wie oben beschrieben) noch selbsterstellt werden mussten.

```
fsd1:pke(X,Y):length(X)=1:length(pke(X,Y))=length(Y):pen.
```

Der erste Teil steht für „function symbol definition“, der Zweite gibt die Operation und deren Argumente, der Dritte die Bedingungen an die Argumente und der Vierte die Länge des entstehenden Wortes an.

4.1.3 Wortbausteine (Atoms)

Hier werden die grundlegenden Wörter angegeben, wie z.B. Nutzernamen, Keys, und Zufallszahlen. Kennt der Angreifer das Wort wird, es mit known markiert (z.B. ein öffentlichen key), kennt er es nicht, so mit notknown (z.B. hoffentlich bei dem privatekey eines KeyServers).

4.1.4 Reduktionsregeln

Hier werden wieder die von oben bekannten Reduktionsregeln in folgender Form angegeben:

```
rr1:e(X,d(X,Y)) => Y.
rr1:d(X,e(X,Y)) => Y.
```

4.2 Analyse eines Protokolls

Der Analyzer arbeitet prinzipiell wie in 2 beschrieben, jedoch werden bei dem unifizieren der Folgerung mit der Zustandsbeschreibung die event-Anweisungen mit einbezogen. Dadurch kann bei der Angabe des unsicheren Zustandes events angegeben werden, die passiert sein mussten oder Bedingungen.

4.2.1 Beschränkungen des Suchraumes

Teilabfragen: Anstatt zu fragen, wie der komplette Zustand erreicht werden kann, kann der Nutzer nur nach Teilen dieses Zustandes fragen. Kann ein Teil nicht erreicht werden, so kann auch der gesamte Zustand nicht erreicht werden. Kann man den Teilzustand jedoch von einem Anfangszustand erreichen, so muss noch einmal der komplette Zustand überprüft werden. Jedoch weiß man nun schon wie mögliche Pfade zum Startzustand aussehen.

Will man zum Beispiel fragen, wie der Angreifer die Worte d(X,W) und X erhalten kann, kann man zum Beispiel erst fragen, wie er d(X,W) erreicht, da was den Suchbaum erheblich verkleinern würde, da der Analyzer (siehe Definition oben) sowohl für beide Ziel seperat als auch zusammen versucht zu erfüllen und für eine Variable ohne weitere Bedingungen der Angreifer relativ viele mögliche Wörter finden kann.

Language Checker: In dem Analyzer wurde ein Tool integriert, daß dazu genutzt wird um auf die oben beschrieben Weise (3.1) die Unereichbarkeit einer Sprache bewiesen. Diese Sprache kann dann in den Analyzer geladen werden, um die Zustände die ein Wort dieser Sprache als Wissen des Angreifers voraussetzen, als unerreichbar zu markieren. Der Analyzer hört dann an einem solchen Zustand mit der Suche in diesem Zweig auf. In [7] werden Methoden beschrieben, um nicht nur die Sprachen automatisch verifizieren zu lassen, sondern diese auch automatisch zu generieren. Dies wurde gegen 1996 implementiert und soll hier nicht weiter ausgeführt werden.

State Unifier: Hier wird davon ausgegange, daß z.B. bestimmte Zustandsvariablen nur Werte in einer vorgegebenen Form annehmen, können. So z.B. eine Variable W, die einen Sessionkey aufnehmen soll, nur Werte der entsprechenden Form annimmt. Diese Daten werden dann verwendet um, wenn in einem Zustand eine Variable einen Wert annehmen soll, zu überprüfen, ob sich der zugewiesene mit der Beschreibung unifizieren lässt. Geht dies, so wird der Zustand verwendet, andernfalls als unerreichbar markiert. Solche Regel können sowohl automatisch als auch manuell hinzugefügt werden.

4.2.2 Automatische Abfragen

Neben den manuellen Abfragen jedes Zustandes, und dessen Vorgängerzuständen usw., gibt es einen automatischen Modus, in dem der Analyzer von einem angegeben Zustand über eine Suche in einen Startzustand zu gelangen. Der Vorteil an dem manuellen Modus ist, daß der Nutzer auch nur Teilzustände abfragen kann, um den Suchbaum übersichtlicher zu halten. Der vom automatischen Modus erzeugte Suchbaum, kann relativ schnell sehr unübersichtlich werden. Man kann aber jederzeit zwischen den Modi wechseln und Veränderungen vornehmen.

5 Einschränkungen des Analyzers

Es gibt einige Eigenschaften in kryptographischen Systemen, die nicht als rewrite-rules wiedergegeben werden können, so zum Beispiel kommutative Operationen (z.B. bei RSA). Solche können nicht im Analyzer modelliert werden und somit Fehler die damit zusammenhängen nicht gefunden werden. Hier will man aber nach anderen allgemeineren Gleichungstheorien suchen, welche Unifikationen.

Der Analyzer kann nur nach definierten unsicheren Zuständen suchen. So kann man zwar bestimmen, ob bestimmte Fehler existieren, man kann jedoch nicht generell sagen, daß das Protokoll sicher ist. Hier versucht man eine universelle Sprache zu entwickeln, um kryptographische Anfordung anzugeben, die unabhängig vom Protokoll ist.

Zudem kann der Analyzer auch keine Fehler in den verwendeten kryptographischen System finden, da er diese, wie die anderen Tools auch, nicht abbildet.

6 Anmerkung zum Analyzers

Vom Nutzer wird relativ viel Eingreifen verlangt, so muss er nicht nur den Zustand angeben, sondern auch noch Klassen von unereichbaren Zuständen definieren. Zu Beginn der Entwicklung musste er dann manuell die Informationen auswerten, später hat der Analyzer dann selber unereichbare Zustände ausgeschlossen, die dennoch zuerst bestimmt werden mussten. Mit Hilfe des automatischen Modus kann man, so man mit Hilfe des State Unifiers und des Language Checkers, eine Menge unereichbarer Zustände definiert hat, bei vielen einfachen Protokollen, Fehler relativ einfach finden. Der Analyzer wird hier aber immer weiter eintwickelt, wie oben erwähnt, kann er mittlerweile automatisch unerreichbare Sprachen finden. In [8] wurde mit Hilfe des Analyzer das Internet Key Exchange Protokoll untersucht (z.B. in Verwendung bei vpn) und der Analyzer erweitert um auch komplexe Protokolle vernünftig untersuchen zu können. Leider findet man über den genauen aktuellen Stand wenig Information.

Der „NRL Protocol Analyzer“ ist in Quintus Prolog und SWIProlog impelementiert und wurde verwendet um zahlreiche unbekannte Fehler in Protokollen zu finden, wie z.B. im Simmons Selective Broadcast Protocol und im Burns-Mitchell Resource Sharing Protocol.

Literatur

[1] Catherine A. Meadows. *Applying Formal Methods to the Analysis of a Key Management Protocol.* Center for Secure Information Technology, Naval Research Laboratory Washington DC, 20375, 1992.

[2] Catherine A. Meadows. *The NRL Protocol Analyzer: An Overview.* Center for High Assurance Computer Systems, Washington DC, 20375, 1994.

[3] Catherine A. Meadows. *Three Systems for Cryptographic Protocol Analysis.* Journal of Cryptology, 7(2), 1994.

[4] Martin Stiel. *Das Dolev und Yao Model - Seminar Analyse kryptographischer Protokolle.* Institut für Informatik, Humboldt-Universität zu Berlin, 2002.

Folgende Dokumente dienen nur als Verweis und wurden nicht oder nur oberflächlich durchgearbeit.

[5] P. Rety, C. Kirchner, H. Kirchner, and P. Lescanne. *NARROWER: A new algorithm for unification and its application to logic programming.* In Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, New York, 1985. Springer Verlag.

[6] Catherine A. Meadows. *A Model of Computation for the NRL Protocol Analyzer.* Center for High Assurance Computer Systems, Naval Research Laboratory, Washington DC, 20375, 1994. Vergleicht den Analyzer mit einer Erweiterung der BAN-Logik und liefert Ansätze beides zu verbinden.

[7] Catherine A. Meadows. *Language Generation and Verification in the NRL Protocol Analyzer.* Center for High Assurance Computer Systems, Naval Research Laboratory, Washington DC, 20375, 1996.

[8] Catherine A. Meadows. *Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer.* Code 5543, Naval Research Laboratory, Washington DC, 20375, 1999.