# Formale Modellierung und Analyse protokollbasierter Angriffe in TCP/IP Netzwerken am Beispiel von ARP und RIP

Gerrit Rothmaier, Heiko Krumm
Materna GmbH Dortmund, Universität Dortmund, FB Informatik
(gerrit.rothmaier@materna.de, krumm@cs.uni-dortmund.de)

**Abstract:** Formale Modellierung und Analyse von netzwerkbasierten Angriffen ermöglichen zwar ein tieferes Verständnis von Angriffsfolgen, sind aber meist so komplex, dass die Modellierung sehr aufwändig und eine Analyse mit automatischen Werkzeugen aufgrund der Größe des Zustandsraumes unmöglich ist. Wir stellen einen Ansatz vor, der Modelle für praxisrelevante Angriffe und Szenarien mit vertretbarem Aufwand zu behandeln vermag. Dieser Ansatz kombiniert cTLA, ein Modellierungsframework und Optimierungsstrategien mit dem mächtigen Modelchecker SPIN. Beispiele zu ARP und RIP zeigen die Anwendbarkeit auf.

## 1 Einleitung

Die Auswirkungen netzwerkbasierter Angriffe wie massenhaftes Auftreten von Würmern oder Ausfall populärer Webseiten sind inzwischen häufig für eine breitere Bevölkerungsschicht spürbar. Ein typischer Angriffstyp nutzt Implementierungsschwachstellen, insbesondere. *Pufferüberläufe*, in verbreiteten Softwarekomponenten aus. Solche Angriffe müssen jedoch meist exakt auf eine bestimmte Plattform mit einer gewissen Implementierungsversion abgestimmt sein. Dahingegen sind *protokollbasierte Angriffe* nicht von speziellen Implementierungsschwachstellen einer bestimmten Plattform abhängig, sondern missbrauchen reguläre Eigenschaften der Protokolle. Aufgrund der umfassenden Verbreitung sind Protokolle aus dem *TCP/IP* Umfeld, z.B. *ARP* und *RIP*, besonders lohnende Angriffsziele. Zusammen mit der herausragenden Bedeutung der Interoperabilität in heterogenen Netzwerken ist es kurzfristig meist nicht machbar, ein anfälliges Standardprotokoll durch ein sichereres Protokoll zu ersetzen. Das anfällige Protokoll kann erst dann deaktiviert werden, wenn der Ersatz eine ausreichend weite Verbreitung erlangt hat. Angriffsmöglichkeiten bestehen also für eine längere Zeit fort. Vor diesem Hintergrund untersuchen wir protokollbasierte Angriffe in *TCP/IP* Netzwerken. Dabei überlagern sich Angriffe und Netzwerkadministrationsaktionen als dynamische Prozesse, die in einem Szenario schrittweise betroffene Systemkomponenten verändern. Unser Ziel ist das Erkennen typischer Angriffsszenarien, Auswirkungen und möglicher Abhilfen. Dazu greifen wir auf Techniken zur formalen Modellierung und Analyse funktionaler Eigenschaften nebenläufiger, ereignisdiskreter Prozesssysteme zurück. Die zu analysierenden Netzwerkmodelle sind oft größer (hinsichtlich der Anzahl der Prozesse, der Anzahl der Systemzustände und der auf unterschiedlichen Abstraktionsebenen relevanten Operationen) als in anderen Bereichen des Einsatzes formaler Techniken (z.B. bei der Analyse von Authentifizierungsprotokollen oder der Verifikation vertrauenswürdiger Softwarekomponenten). Entsprechend treten bekannte Nachteile formaler Techniken wie hoher Modellierungsaufwand und enorme Zustandsräume in der automa-

tisierten Analyse verstärkt auf. Um diese Probleme zu bewältigen, haben wir einen Ansatz mit folgenden Grundelementen entwickelt: Zur Modellierung verwenden wir die Spezifikationssprache *cTLA* [HK00]. Zusammen mit einem Framework ermöglicht sie die flexible und kompositionale Modellierung komplexer Systeme. Die *cTLA* Modelle werden mit Hilfe des Übersetzers *cTLA2PC* in die weniger strukturierte, *C*-ähnliche Sprache *Promela* übersetzt. *Promela* ist die Eingabesprache für den zur Analyse verwendeten Modellchecker *SPIN* [Hol03]. Eine entscheidende Rolle für die Analysierbarkeit mit automatisierten Werkezeugen spielen geeignete Optimierungen auf verschiedenen Ebenen.

Der Beitrag soll einen Einblick in unseren Modellierungs- und Analyseansatz geben. Zunächst werden existierende Ansätze und die in den Beispielen verwendeten Protokolle *ARP* und *RIP* kurz eingeführt. Abschnitt 4 umreißt *cTLA*, Abschnitt 5 das *cTLA* Modellierungsframework. In Abschnitt 6 folgen *SPIN*, *Promela* und *cTLA2PC*. Optimierungen werden in Abschnitt 7 vorgestellt. Abschnitt 8 enthält zwei Beispiele zu *ARP* und *RIP*.

## 2 Existierende Ansätze

Ansätze zur formalen Modellierung und Analyse im Bereich Sicherheit existieren schon länger. Sie lassen sich hauptsächlich in die Bereiche Programmverifikation (z.B. [BR00]) und Protokollverifikation (z.B. [Mea95]) einteilen. Dort werden auch bereits zumindest teilautomatisierte Werkzeuge wie Theorem-Beweiser, Modellchecker oder Term-Rewriting-Systeme eingesetzt.

Die Untersuchung netzwerkbasierter Angriffe unter Berücksichtigung von Managementeingriffen und Schwachstellenausbreitung in Netzwerken ist jedoch ein neues Gebiet. In [AR00] (leicht erweitert in [NOR02]) werden Angriffe, als Kombination der Ausnutzung von Schwachstellen auf einem Host und anschließendem Wechsel des Angriffshosts, mit dem Modellchecker *SMV* [Ber01] untersucht. Das zugrundeliegende Modell ist elementar, beispielsweise werden Hosts durch eine vordefinierte boolesche Schwachstellen-Menge (Wert wahr entspricht vorhandener Schwachstelle) und einen Wert repräsentiert, der die momentanen Zugriffsrechte des Angreifers (*keine* oder *Root*) enthält. Ausnutzungen von Schwachstellen erhöhen die Zugriffsrechte und werden als Formeln repräsentiert. Gewünschte Sicherheitseigenschaften werden als Assertions dargestellt. Gegenbeispiele des Modellcheckers sind dann genau die erfolgreichen Angriffe. Beim Ansatz von [RS98] (erweitert in [RS02]) werden Angriffe als Kombination der Interaktion von Systemkomponenten auf einem *einzelnen* Host untersucht. Ein Host besteht aus einer Menge von Prozessen (wie z.B. Dateisystem- und Druckprozess) mit lokalem Zustand, die über synchrone Aktionsaufrufe kommunizieren. Modellierung und Analyse verwenden *XMB*, eine Prologumgebung. Wiederum werden Angriffsfolgen als Gegenbeispiele zu Assertions gefunden.

Bei der Optimierung unserer Modelle greifen wir teilweise auf Techniken zurück, die der effizienten Protokollimplementierung ähneln, beispielsweise den *Activity Thread* Ansatz aus [Svo89]. Desweiteren enthält [Ruy01] einige für uns auf *Promela* Ebene hilfreiche *SPIN* Optimierungen.

## 3 Routing in TCP/IP Netzwerken

Beim Routing lassen sich zwei Fälle unterscheiden: Routing innerhalb eines LAN Segments und segmentübergreifend. In *TCP/IP* Netzwerken wird innerhalb eines LAN Segments das *Address Resolution Protocol (ARP)* (vergl. z.B. [Tan03], [SK91]) zum Routing verwendet. Es stellt einen Mechanismus zur Abbildung von logischen *IP* Adressen auf die Hardwareadressen der Hosts bereit. Ein Host A, der mit einem Host B innerhalb desselben LAN Segments kommunizieren will, sendet zunächst eine *ARP Anfrage* mit seinem eigenen *IP*- und Hardwareadresspaar und Host Bs *IP* Adresse (sog. *Who-Has*) an das gesamte LAN (per *Hardware Broadcast*). Host B erhält diese Anfrage, nimmt Host As *IP*- und Hardwareadresspaar in seinen *ARP Cache* auf und sendet sein eigenes Adresspaar in einer *ARP Antwort* zurück an Host A. Host A nimmt das empfangene Adresspaar in seinen *ARP Cache* auf. Anschließend können die Hosts A und B auf Ebene ihrer *IP Schichten* miteinander kommunizieren. Aufgrund der Speicherung in den *ARP Caches* (mit begrenzter Größe) ist der *ARP Anfrage-Antwort* Vorgang üblicherweise nur vor der ersten *IP* Kommunikation zwischen Host A und B notwendig.

Im Falle der segmentübergreifenden Kommunikation müssen Pakete von Routern weitergeleitet werden. Beim *TCP/IP* Routing hängt der nächste Router (*Next-Hop*) üblicherweise nur von der Zieladresse des Pakets (*Destination-based Routing*) ab. Zur Bestimmung des nächsten Routers wird in der einfachsten Form eine *Routingtabelle* verwendet. Jeder Eintrag (auch genannt *Route*) hat die Form (*Ziel*, *Schnittstelle*). Dabei kann *Ziel* sowohl für einen einzelnen Host als auch ein komplettes Netzwerk stehen, *Schnittstelle* steht für eine Verbindung zu einem weiteren Router oder direkt zum Zielnetz. Schritt für Schritt gelangt das Paket so bis zu einem Router, der direkt mit dem Zielnetz verbunden ist. Innerhalb des Zielnetzes wird das Paket dann wiederum wie oben mit Hilfe von *ARP* an den Zielhost zugestellt. Da statische Routingtabellen aufwändig zu pflegen sind und auf dynamische Veränderungen der Netzwerktopologie (wie Ausfall eines Routers) nicht reagieren können, wurden Protokolle zum Austausch von Informationen zwischen Routern entwickelt. Mit diesen Informationen und einer erweiterten Routingtabelle, der *Routing Information Base (RIB)*, können Router ihre Tabellen dynamisch anpassen. Das heutzutage verbreitetste Protokoll zum Austausch von Informationen zwischen Routern, die unter derselben administrativen Kontrolle stehen (*Interior Gateway*, d.h. zu einer Organisation gehörig), ist *RIP v1* [Hed88]. *RIP* basiert auf einem Distanzvektoralgorithmus [For62]. Die Einträge der *RIB* haben die Form (*Ziel*, *Next-Hop*, *Schnittstelle*, *Metrik*), dabei ist *Metrik* die Entfernung (üblicherweise in Anzahl Router) zu *Next-Hop*. Aktualisierungsnachrichten haben die Form (*Ziel'*, *Metrik'*). Allgemein wird ein *RIB* Eintrag durch eine Aktualisierungsnachricht (die einfache Plausibilitätschecks erfüllen muss) verändert bzw. neu angelegt wenn die Metrik dadurch besser (d.h. kleiner) wird. Stammt eine Nachricht vom *Next-Hop* einer vorhandenen *Route,* wird die Metrik auch aktualisiert wenn sie schlechter (d.h. größer) ist. Empfängt ein Router längere Zeit keine Aktualisierungsnachricht vom *Next-Hop* (z.B. wg. Ausfall), inkrementiert der Router die Metrik. Dies führt schließlich zur Aufgabe dieser Route. Aktualisierungsnachrichten werden zwischen den Routern in regelmäßigen Abständen (*Regular Updates*) und bei Änderungen in der *RIB* (*Triggered Updates*) versendet (unter Berücksichtigung von Sonderfällen wie dem *Split-Horizon* Prinzip). *Triggered Updates* sind im Anwendungsbsp. (s. Abschn. 9) die interessantesten Fälle.

## 4 Modellierungssprache cTLA

Eine *Temporal Logic of Actions (TLA)* [Lam93] Spezifikation beschreibt ein Zustandsübergangssystem durch eine *kanonische Formel* $f_{Sys}$ ::= $Init \wedge \Box [ Next ]_V$ , wobei $V$ die Menge der Zustandsvariablen, *Init* der initiale Systemzustand und *Next* die Zustandsübergänge als Disjunktion von Systemaktionen $sysAct_1 \vee sysAct_2 \ldots \vee sysAct_m$ . Optional kann $f_{Sys}$ auch um *Fairness* Annahmen erweitert werden, für unsere Anwendung stehen jedoch *Safety* Eigenschaften im Vordergrund. Eine Systemeigenschaft $f_{Prop}$ kann durch die Implikation $f_{Sys} \Rightarrow f_{Prop}$ formal verifiziert werden. *Compositional TLA (cTLA)* [HK00] erweitert *TLA* um Möglichkeiten zur Zusammensetzung von Spezifikationen aus Modulen und Prozesstypen. Prozesstypen können aus anderen Prozesstypen zusammengesetzt sein. Durch diese Möglichkeiten wird Wiederverwendung erleichert und der Einsatz eines Frameworks (vergl. Abschn. 5) möglich. Im folgenden werden kurz *cTLA 2003* Prozesstypen vorgestellt, eine ausführlichere Darstellung findet sich in [RK03].

**Einfacher cTLA Prozesstyp** – Der einfache *cTLA* Prozesstyp beschreibt auf naheliegende Weise direkt ein Zustandsübergangssystem. Die Menge der Zustandsvariablen besteht gerade aus den lokalen Variablen, *Init* entspricht dem *cTLA* `INIT` Prädikat und *Next* ist die Disjunktion der lokalen Aktionen. Der einfache *cTLA* Prozesstyp `Media` aus dem Framework (vergl. Abschn. 5) beispielsweise modelliert physikalische Medien, über welche Pakete übertragen werden.

**Erweiternder cTLA Prozesstyp** – Durch den erweiternden *cTLA* Prozesstyp, Schlüsselwort `EXTENDS`, können ein oder mehrere andere Prozesstypen ergänzt und spezialisiert werden. Die Menge der Zustandsvariablen des erweiternden Prozesstyps entspricht der Menge der Zustandsvariablen der erweiterten Prozesstypen ergänzt um die Menge der Zustandsvariablen des erweiternden Prozesstyps. *Init* wird als Konjunktion aus den `INIT` Prädikaten der erweiterten Prozesstypen und dem `INIT` des erweiternden Prozesstypen gebildet. Für *Next* werden zunächst gleichnamige Aktionen sowohl des erweiternden als auch der erweiterten Prozesstypen konjunktiv verknüpft, anschließend werden die so entstandenen Aktionen disjunktiv verknüpft. Damit ähnelt der erweiternde cTLA Prozesstyp der Vererbung aus der *objektorientierten Programmierung (OOP)*, wobei hier jedoch Verhalten nicht ersetzt, sondern spezialisiert wird. Ein erweiternder *cTLA* Prozesstyp ist z.B. `NonPromIpNode` aus dem Framework, welcher `HostIpNode` so spezialisiert, dass Pakete nur im *Non-Promiscuous-Mode* empfangen werden können.

**cTLA System Prozesstyp** – Ein cTLA System Prozesstyp, Schlüsselwort `CONTAINS`, koppelt enthaltene Prozessinstanzen. Die Menge der Zustandsvariablen entspricht den enthaltenen Prozessinstanzen, *Init* der Konjunktion der *Init*s der enthaltenen Prozessinstanzen. *Next* wird als Disjunktion der im Prozesstyp definierten *Systemaktionen* der Form $sysAct_i = P_{j1}.A_{k1} \wedge \ldots \wedge P_{jni}..A_{kni}$, wobei *P.A* für eine Aktion der Instanz *P* steht, gebildet. Jede Instanz darf in jeder Systemaktion höchstens einmal vorkommen, kommt sie nicht vor, so nimmt sie bei Ausführung der Systemaktion einen *Stotterschritt* vor. Ein *cTLA* System Prozesstyp ist z.B. stets der das Gesamtsystem modellierende Prozesstyp, z.B. `IpArpExample` (s. Listing 1), welcher *Node*- und *Media*-Prozessinstanzen im *ARP* Beispiel (s. Abschn. 8) zum Gesamtsystem koppelt (s. Listing 1).

```
PROCESS IpArpExample();
CONTAINS
  med: Media();
  bnA: IpArpNode( NA_ID, NA_MII );
  bnB: IpArpNode( NB_ID, NB_MII );
  bnC: IpArpNode( NC_ID, NC_MII );
ACTIONS  /* send system actions */
  snd_A( pkt: PacketT ) ::= bnA.snd( pkt ) AND med.in( pkt );
  snd_B( pkt: PacketT ) ::= bnB.snd( pkt ) AND med.in( pkt );
  snd_C( pkt: PacketT ) ::= bnC.snd( pkt ) AND med.in( pkt );
         /* receive system actions */
  rcv_A( pkt: PacketT ) ::= bnA.rcv( pkt ) AND med.out( pkt );
  ...
         /* broadcast receive system actions */
  rbc( pkt: PacketT ) ::=
         bnA.rbc( pkt ) AND bnB.rbc( pkt ) AND bnC.rbc( pkt )
  AND med.out( pkt );
  /* internal node actions implicitly added */
END
```

Listing 1: System Prozesstyp IpArpExample

## 5 Modellierungsframework

Im Bereich der *OOP* dienen Frameworks zur Wiederverwendung bewährter Elemente und Strukturen für eine bestimmte Anwendungsdomäne (vergl. z.B. [GHJ97, S. 31f]) und senken so den Aufwand trotz erhöhter Qualität der Lösungen. Eine Übertragung des Framework Konzepts aus der *OOP* auf *cTLA* Netzwerkmodelle bietet sich aufgrund der hohen Aufwände bei der formalen Modellierung und Analyse von Netzwerkmodellen (vgl. Abschn. 1) und der speziellen Eigenschaften der *cTLA 2003* Spezifikationsspache für Wiederverwendung und Komposition (vergl. Abschn. 4) an. Grundidee des *cTLA* Netzwerkmodellierungsframeworks ist die Sicht von Netzwerken als Zonen (typischerweise entspr. LAN-Segmenten), Knoten (`Node`s) und einem Medium (engl. `Media`). Das Medium ist entsprechend der Zonenstruktur partitioniert und kann Pakete übertragen (Aktionen `Media.in` bzw. `Media.out`). Die Knoten sind logisch in Schichten (engl. *Layers*) unterteilt und empfangen (`Node.rcv`), verarbeiten (`Node._pcs_`) und senden (`Node.snd`) Pakete. Einen Überblick über das Framework zeigt Abb. 1, eine Strukturierung nach syntaktischem Typ der Elemente liefert die Pakete *Aufzählungstypen & Funktionen*, *Datentypen* und *Prozesstypen*. Es sei hier nur kurz auf den groben Zweck der Pakete eingegangen, eine ausführliche Beschreibung der Frameworkelemente enthält [Rot04]. Die Elemente des Pakets *Aufzählungstypen und Funktionen* dienen zur Definition der Netzwerktopologie, benötigter Protokolle (je nach Anwendungsszenario) und zur Zuweisung initialer Adressen. Im Paket *Datentypen* sind Elemente für innerhalb des Frameworks immer wieder benötigte Elemente wie Netzwerkschnittstellen, Pakete und Puffer enthalten. Schließlich enthält das Paket *Prozesstypen* die Kernelemente des Frameworks mit `Node`-, `Media`- und `Router`typen. Durch Verwendung des `EXTENDS` Mechanismus lassen sich diese Prozesstypen feingranular aufeinander aufbauen.

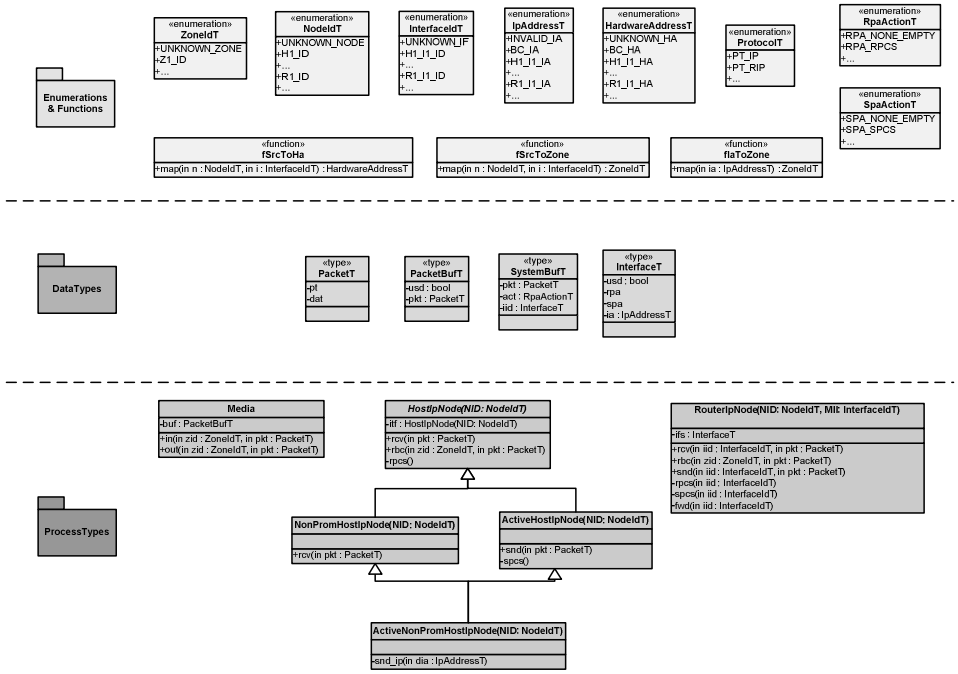


Abbildung 1: Framework Struktur

# 6 SPIN, Promela & cTLA2PC

*SPIN* [Hol03] ist einer der in der Praxis am meisten eingesetzten und mächtigsten *Finite-State* Modellchecker. Seit 1995 gibt es eigene internationale Workshops zu *SPIN* und in 2001 wurde *SPIN* mit dem *ACM Software System Award* ausgezeichnet. Bei der Verifikation unterstützt *SPIN* vielfältige Optionen bezüglich Suchstrategie und -modus. Insbesondere ist auch ein *Guided-Simulation* Modus, der bei der Verifikation aufgefallene Folgen interaktiv nachvollziehen lässt, implementiert.

*Promela*, die Eingabesprache von *SPIN* ähnelt syntaktisch der Sprache *C*, ist aber einerseits z.B. hinsichtlich Arithmetrikfunktionen stark reduziert, andererseits um spezielle Konstrukte zur Erzeugung, Synchronisation und Kommunikation von Prozessen erweitert. Zur Prozesssynchronisation dienen *Guarded Statements*, die Kommunikation zwischen Prozessen kann sowohl über gemeinsam verwendete, globale Variablen als auch über spezielle *Channels* erfolgen. *Channels* entsprechen *FIFO* Warteschlangen, in die Nachrichtenwerte geschrieben bzw. gelesen werden können. Promela unterstützt jedoch im Gegensatz zu *cTLA* keine Aktionen oder Parameter, alle Prozesse bestehen aus einem monolithischen Block von Anweisungen, deren Verlassen den Prozess beendet. Ebenso gibt es keine zusammengesetzten Prozesstypen. Die Stärken von Promela liegen also eher in der effizienteren Analyse aufgrund der einfachen Struktur und der Unterstützung durch *SPIN* als in der leichten Anwendbarkeit für komplexe Modelle.

Um die Vorteile von *SPIN/Promela* hinsichtlich der Analyse als auch die Vorteile von *cTLA* hinsichtlich komplexer Modelle kombinieren zu können, haben wir ein Werkzeug zur Übersetzung von *cTLA* Modellen nach *Promela*, *cTLA2PC*, entwickelt. *cTLA2PC* unterstützt verschiedene Optionen um auf *SPIN* optimierten *Promela* Code zu erzeugen, so z.B. eine verallgemeinerte Version des *Bit-Vector* Ansatzes aus [Ruy01]. Grundsätzlich erfolgt die Übersetzung von *cTLA* nach *Promela* mit *cTLA2PC* wie folgt. Zunächst wird aus dem in der *cTLA* Spezifikation beschriebenen kompositionalen System durch Expansion und Auflösen von Prozesstypen ein äquivalentes *flaches System* erzeugt. Dieses *flache System* enthält nur noch einen einfachen *cTLA* Prozess mit seinen Aktionen. Er wird in einen *Promela* Prozess übersetzt, darf jedoch dazu keine Aktionen und Parameter mehr enthalten. Die *cTLA* Aktionen werden jeweils als eigene Wahlmöglichkeit in eine nicht-deterministische *Promela* Auswahlschleife eingebettet. Nun sind noch die Parameter zu behandeln. In *cTLA* sind Aktionsparameter implizit existenzquantifiziert, d.h. eine Aktion ist ausführbar, wenn eine Belegung der Aktionsparameter existiert, die die Vorbedingungen der Aktion erfüllt. Zur Umsetzung werden zum einen die Aktionsparameter durch globale Variablen ersetzt, unter Zusammenfassung nicht innerhalb einer Aktion verwendeter Parameter gleichen Typs. Zum anderen werden für die dann noch notwendigen globalen Parametervariablen *Eingabegeneratorprozesse* erzeugt. Ein Eingabegeneratorprozess besteht aus einer nicht-deterministischen *Promela* Auswahlschleife, die alle möglichen Wertebelegungen für den Parametervariablentyp enthält. Schließlich werden die Vorbedingungen jeder *cTLA* Aktion in *Promela Guarded Statements* umgesetzt. Insgesamt kann mit der beschriebenen Vorgehensweise eine *cTLA* Spezifikation auf *Promela* abgebildet werden. Weitere Punkte, wie z.B. die Abbildung von Datentypen, Funktionen und Prädikaten, auf die hier aus Platzgründen nicht weiter eingegangen werden soll, müssen jedoch dabei zusätzlich berücksichtigt werden.

## 7 Optimierungen

Selbst die Zustandsvektoren von kleinen Netzwerkmodellen sind oft so groß, dass eine Analyse mit *SPIN* selbst mit 4 GB Hauptspeicher, dem Speicherlimit von 32-Bit Systemen (wie der *x86 Architektur*), nicht mehr möglich ist. Durch geeignete Optimierungen lässt sich jedoch oft schrittweise der Zustandsvektor verkleinern, bis schließlich doch noch eine Analyse durchführbar ist. Solche Optimierungen können, und – im Falle größerer Modelle – müssen auf mehreren Ebenen durchgeführt werden.

**Szenario** – Oft lässt sich auf der obersten Ebene das Anwendungsszenario vereinfachen. Beispielsweise kann eine Betrachtung eines Rechners exemplarisch für alle Rechner eines LANs ausreichen oder durch Fallunterscheidung lässt sich ein Szenario auf mehrere, einfacher zur analysierende Fälle aufteilen.

**cTLA** – Auch auf der *cTLA* Ebene lassen sich Optimierungen erzielen. Beispielsweise können nach dem *Activity Thread* Ansatz [Svo89] schichtübergreifend Puffer und Aktionen eines Nodes, die dasselbe Paket bearbeiten, zusammengefasst werden. Die `Node`-Typen des Frameworks (s. Abschn. 5) verwenden diesen Ansatz bereits. Durch diese Optimierung ließ sich der Zustandsvektor beim *ARP* Beispiel (s. Abschn. 8) von 484 Bytes auf 248 Bytes verringern.

Weiterhin können, ausgehend vom *flachen System* (vgl. Abschn. 6), oft Parameter der Systemaktionen durch feste Werte ersetzt werden. Das liegt daran, dass durch die Kopplung von Prozessaktionen häufig ein Eingabeparameter eines Prozesses ein Ausgabeparameter eines anderen ist. Somit lässt sich der in der Systemaktion vorhandene Eingabeparameter durch den symbolischen Wert des Ausgabeparameters ersetzen. Entsprechend fällt bei der anschließenden Übersetzung nach *Promela* dann auch der Eingabegeneratorprozess für diesen Parameter weg. Im *RIP* Beispiel (s. Abschn. 8) verkleinert dieser *Paramodulations*-Ansatz den Zustandsvektor von 630 auf 580 Bytes.

**Promela** – Wie bereits in Abschn. 6 angedeutet, sind teilweise auch spezielle Optimierungen für *SPIN* Eigenheiten erforderlich und in *cTLA2PC* implementiert. Beispielsweise sollten die in *SPIN* eingebauten *Bit-Arrays* nicht verwendet werden, da sie pro Bit bis zu einem Byte benötigen. Die Verwendung einer eigenen Umsetzung basierend auf dem *Bit-Vector* Ansatz aus [Ruy01] spart für diesen Datentyp daher bis zu 7/8 des Speicherbedarfs ein. Ähnliches gilt teilweise auch für andere Datentypen, das Verhalten von *SPIN* bei solchen Optimierungen sollte jedoch fallweise geprüft werden. Weiterhin lassen sich aus der *cTLA* Herkunft des *Promela* Modells Optimierungen herleiten. So können *cTLA* Aktionen zwar in einer beliebigen Reihenfolge ausgeführt werden, jede Aktion für sich ist jedoch atomar und deterministisch. Also kann in der Einbettung der *cTLA* Aktionen in *Promela* (vergl. Abschn. 6) jede Aktion durch ein *Promela* `d_step` umschlossen werden. Dies verringert zwar nicht den Zustandsvektor, aber die Anzahl der möglichen Zustandsübergänge.

Obige Punkte sind nur eine Auswahl der möglichen Optimierungen auf den jeweiligen Ebenen, so lässt sich auch die für eine spezifische Sequenz benötigte Suchtiefe verringern. Insgesamt ergibt sich durch Optimierungen für das *ARP* Beispiel schließlich ein Zustandsvektor von 168 statt 484 Bytes, für das *RIP* Beispiel 316 statt 630 Bytes.

## 8 Anwendungsbeispiele

Im folgenden werden kurz zwei Beispiele zu protokollbasierten Angriffen in *TCP/IP* Netwerken, auf die unser Ansatz (vergl. Abschn. 1) erfolgreich angewandt wurde, beschrieben.

**ARP Szenario** – Betrachtet wurde ein LAN Segment mit drei Hosts A, B, C. Untersucht wurden die Auswirkungen von *IP* Übernahmen (u.a. durch Administrationsfehler) bzw., wie sich als vergleichbar erwies, *ARP* Angriffen. Details zu diesem Szenario, jedoch mit einer anderen Angriffsfolge, finden sich in [RPK04].

Zur Modellierung verwendet wurden (vergl. `IpArpExample` Prozesstyp in Abschn. 4) für die Hosts drei Instanzen von `IpArpNode`, basierend auf `HostIpNode`, und eine Instanz von `Media`. In `IpArpNode` wurde die Funktionalität zum Senden bzw. Empfangen von *ARP Anfragen* bzw. *Antworten* und zur Aktualisierung des *ARP Caches* integriert (vergl. Abschn. 3). Die *IP* Schicht wurde so modifiziert, dass bei Versand eines *IP* Pakets, zu dessen Ziel-*IP* die zugehörige Hardwareadresse noch nicht im *ARP Cache* ist, das Paket zunächst zurückgestellt und eine *ARP Anfrage Antwort* abgewartet wird.

Angriffsfolgen entsprechen Verletzungen von Assertions, die Sicherheitseigenschaften modellieren (vergl. Abschn. 2). Diese Assertions werden nach der Übersetzung des *cTLA* Modells durch *cTLA2PC* (vergl. Abschn. 6) mit Promela `assert` Anweisungen eingefügt. Sei die Sicherheitseigenschaft „Ein *Node* kann die Kommunikation der anderen *Nodes* nicht abhören.“. Dann sind verschiedene Fälle denkbar, die durch ein oder mehrere Assertions modelliert werden können: z.B. mit *Node* A als Angreifer, Kommunikation zwischen B und C und *Node* C als berechtigtem Empfänger, die entsprechende *Promela* Assertion findet sich in Listing 2.

```
assert(      !(bnA_ifs[1 - 1].spa.pkt.l2t==L2_IP &&
  BVGET(    bnA_ifs[1 - 1].spa.pkt.dat, 3, DI_IDA) ==
            bnC_ifs[1 - 1].ia && bnA_ifs[1 - 1].spa.pkt.dha !=
            bnC_ifs[1 - 1].ha ));
```

Listing 2: Assertion im ARP Szenario

Mit Hilfe des *Guided Simulation* Modus von *SPIN* können Angriffsfolgen betrachtet und auf das *cTLA* Modell zurückgeführt werden. So findet *SPIN* z.B. die in Abb. 2 als Sequenzdiagramm dargestellte Angriffsfolge zu der Assertion aus Listing 2:

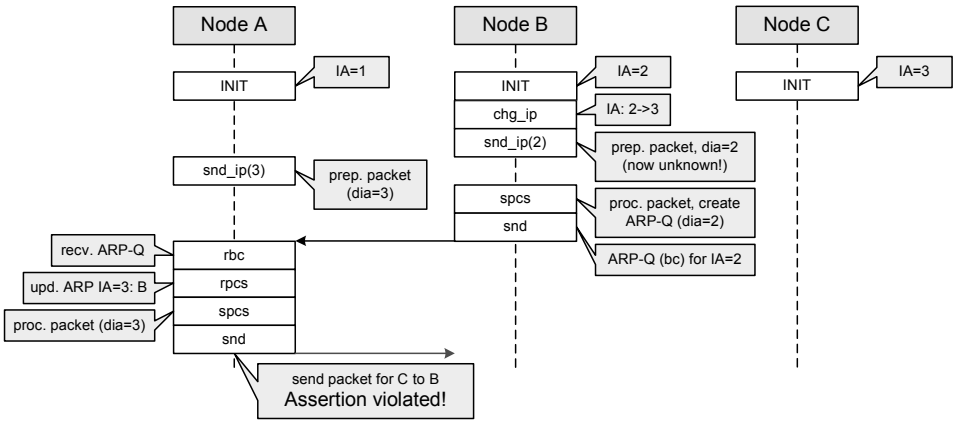


Abbildung 2: Angriffsfolge im ARP Szenario

Dieses Sequenzdiagramm zeigt wie *Node* B durch Übernahme der *IP* Adresse von *Node* C und anschließendem Senden einer „gefälschten“ *ARP Anfrage* den *ARP Cache* von *Node A* so beeinflussen kann (sog. *ARP Cache Poisoning*), dass zukünftige Pakete von *Node* A an *Node* C von *Node* B abgefangen werden. Entsprechende Angriffswerkzeuge sind im Internet frei verfügbar (z.B. in [OV04]). Obige Folge zeigt, dass *ARP Cache Poisoning* nicht nur durch einen bewussten Angriff sondern auch durch einen Administrationsfehler (doppelte *IP* Vergabe) ausgelöst werden kann.

**RIP Szenario** – Betrachtet wurde ein Szenario welches nach Vereinfachung aus LAN Segmenten Z1, Z2, Z3, Hosts H1, H2, H3 mit *Hi* in *Zi, RIP v1* fähigen Routern R1, R2, R3 mit je drei Interfaces und *Backbone* Segmenten ZBB12, ZBB13, ZBB23 wobei $ZBB_{ij}$ die Router $R_i$ und $R_j$ verbindet, bestand (vergl. Abb. 3). Untersucht wurden die Auswirkungen von *RIP Update* Angriffen wie sie z.B. das Werkzeug [Fx01] ermöglicht.

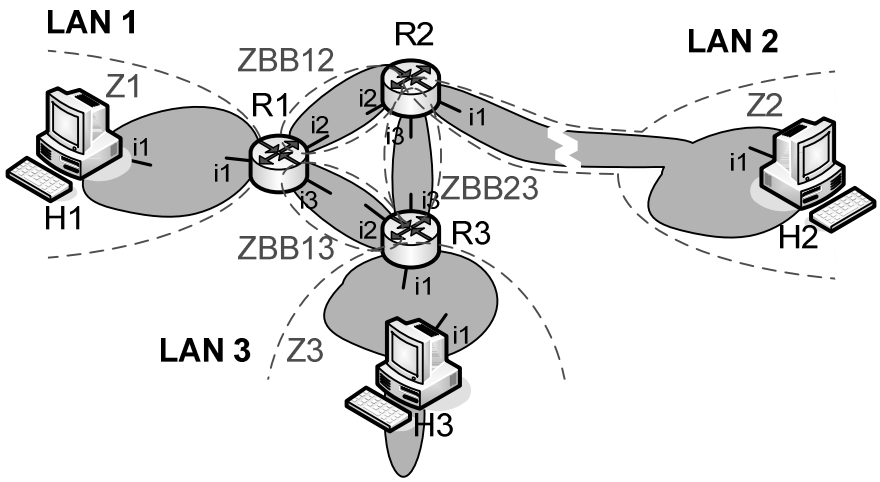


Abbildung 3: Vereinfachtes RIP Szenario

Zur Modellierung wurden für die Hosts jeweils unterschiedliche Rollen gewählt, z.B. H1 normaler Host (Prozesstyp `ActiveHostIpNode`), H2 passiver Host (`NonPromHostIp-Node`) und H3 Angreifer (`RipAttackerHostNode`). Die Router wurden durch den Prozesstyp `RipRouterIpNode`, basierend auf `RouterIpNode` modelliert. Der Basistyp `RouterIpNode` enthält die Funktionalität zur Weiterleitung von Paketen in *TCP/IP* Netzwerken nach einer statischen Routingtabelle. Durch `RipRouterIpNode` wird die Funktionalität zur Verarbeitung und zum Versenden von *RIP Updates*, dem aktualisieren der *RIB* etc (vergl. Abschn. 3) ergänzt. Eine `Media` Instanz verbindet alle *Nodes* entsprechend der LAN Segmentierung.

Die Sicherheitseigenschaft kann ähnlich dem *ARP* Szenario gewählt werden, mit obiger Verteilung der Hostrollen ergibt sich folgende *Promela* Assertion: `assert( (h3_itf.rpa.pkt.dat_ida == H3_I1_IA) )`. Sie wird eingefügt in die *Unicast* Empfangsaktion von Host H3 und drückt aus, dass H3 nur *Unicast IP* Pakete empfangen darf, die auch an die IP Adresse von H3 gerichtet sind.

Bei Untersuchung des übersetzten *cTLA* Modells ergänzt um diese Assertion mit *SPIN* wird eine verletzende Angriffsfolge gefunden, die sich durch das Sequenzdiagramm in Abb. 4 veranschaulichen lässt. Im wesentlichen sendet Host H3 ein gefälschtes *RIP Update* Paket an Router R3 in welchem er vorgibt eine kürzere Verbindung zu Z3 zu haben. Daraufhin aktualisiert R3 seine *RIB* (insb. wird H3 als *Next-Hop* für Z3 gesetzt) und sendet ein *Triggered Update* bzgl. dieses neuen Wegs an R1, der nun ebenfalls seine *RIB* aktualisiert (insb. Next-Hop R3 bzgl. Z3), da auch für ihn der neue Weg immer noch eine bessere Metrik hat. Nach diesen Änderungen an den Routingtabellen wird das Paket von H1 an H2 nicht mehr von R1 über R2 an H2 sondern automatisch von R1 über R3 an H3 weitergeleitet. Auf diese Weise gelingt es H3, ein Paket von H1 an H2 zu empfangen und die Assertion zu verletzen.

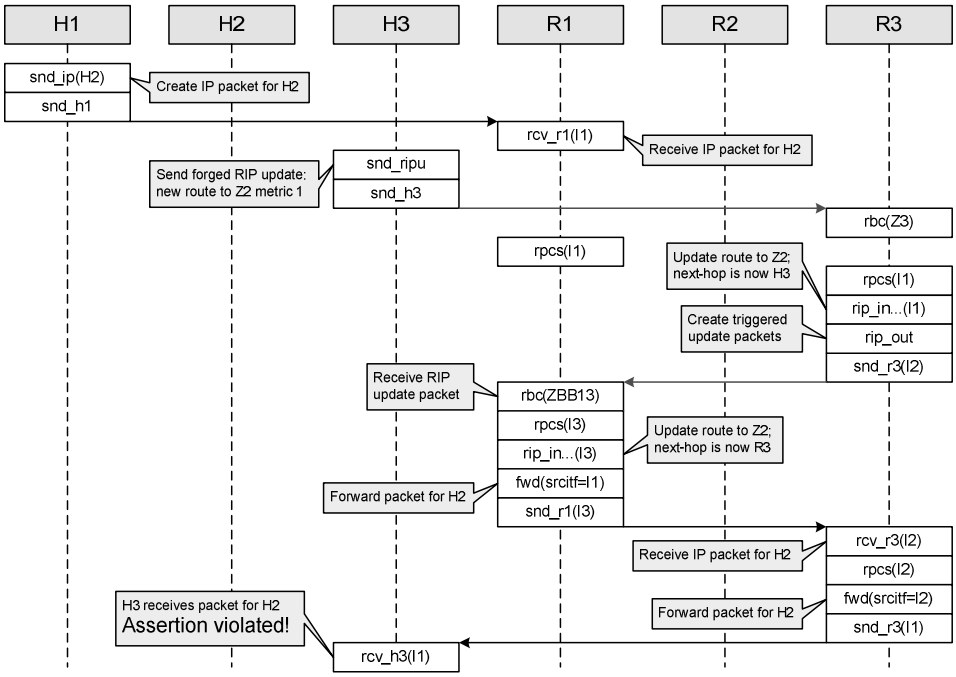


Abbildung 4: Angriffsfolge im RIP Szenario

# 9 Zusammenfassung und Ausblick

Angriffe auf *RIP* und *ARP* sind wichtige Beispiele protokollbasierter Angriffe in *TCP/IP*-Netzwerken, die allgemein keine plattformspezifischen Implementierungsschwächen sondern reguläre Eigenschaften der jeweiligen Protokolle ausnutzen. Die protokollbasierten Angriffe sind eine besondere Teilmenge der netzwerkbasierten Angriffe, die durch formale Modellierung und Analyse untersucht werden können. Der hier vorgestellte Ansatz kombiniert die kompositionale Spezifikationssprache *cTLA* mit einem Modellierungsframework, Optimierungsstrategien auf mehreren Ebenen, einem automatischen Übersetzungswerkzeug und dem Modelchecker *SPIN*. Insgesamt wird die Analyse von Angriffsfolgen möglich, wie sie durch in der Praxis verfügbare Angriffswerkzeuge in realen Netzwerken ausgelöst werden.

Unsere aktuelle Forschung zielt darauf ab, diesen Ansatz zu erweitern und zu verfeinern um schließlich sogar die Analyse von netzwerkbasierten Angriffen, die verschachtelt mit dynamischen Administrationsprozessen einhergehen, zu erlauben. Um dieses Ziel zu erreichen sind weitergehende Optimierungsstrategien, die möglichst auch durch *cTLA2PC* automatisiert durchgeführt werden können, und Ergänzungen des Frameworks geplant. Desweiteren sollen Modellentwurf, -übersetzung und -analyse in eine gemeinsame Umgebung integriert werden, ähnlich dem Konzept einer integrierten Entwicklungsumgebung.

## Literaturverzeichnis


[AR00] Ammann, P.; Ritchey, R.: Using Model Checking to Analyze Network Vulnerabilities. IEEE Symposium on Security and Privacy, 2000.

[Ber01] Berezin, S.: The SMV web site. URL: http://www.cs.cmu.edu/~modelcheck/smv.html/, 1999.

[BR00] Balser, M.; Reif, W. et al.: Formal System Development with KIV. In: T. Maibaum (ed.), Fundamental Approaches to Software Engineering. Springer LNCS 1783, 2000.

[For62] Ford, L. R. Jr.; Fulkerson, D. R.: Flows in Networks, Princeton University Press, Princeton, N.J., 1962.

[Fx01] FX of Phenoelit: Internet Routing Protocol Attack Suite, URL: http://www.phenoelit.de/-irpas, 2001.

[GHJ97] Gamma, E.; Helms, R.; Johnson, R. et al: Entwurfsmuster. Addison-Wesley, 1997.

[Hed88] Hedrick, C.: RFC 1058 - Routing Information Protocol. Network Working Group Rutgers University, URL: http://www.faqs.org/rfcs/rfc1058.html, 1988.

[HK00] Herrmann, P.; Krumm, H.: A framework for modeling transfer protocols. Computer Networks, 34 (2000), S. 317-337.

[Hol03] Holzmann, G. J.: The SPIN model checker. Addison Wesley, 2003.

[Lam93] Lamport, L.: The temporal logic of actions, ACM Trans. Prog. Lang. and Sys. 16 (1994), S. 872-923.

[Mea95] Meadows, C.: Formal Verification of Cryptographic Protocols: A Survey. Advances in Cryptology - Asiacrypt 1994, LNCS 917, Springer-Verlag, 1995, S. 133-150.

[NOR02] Noel, S.; O' Berry, B.; Ritchey, R.: Representing TCP/IP connectivity for topological analysis of network security. Computer Society, IEEE (ed.), Proceedings of the 18th Annual Computer Security Applications Conference, 2002, S. 25-31.

[OV04] Ornaghi, A.; Valleri, M.: Ettercap - Suite for man in the middle attacks on LANs. URL: http://ettercap.sourceforge.net, 2004.

[RS98] Ramakrishnan, C.; Sekar, R.: Model-Based Vulnerability Analysis of Computer Systems. Second International Workshop on Verification, Model Checking, and Abstract Interpretation, Pisa, Italien, 1998.

[RS02] Ramakrishnan, C.; Sekar, R.: Model-Based Analysis of Configuration Vulnerabilities. Journal of Computer Security, Vol. 10, Nr. 1, Jan. 2002, S. 189-209.

[Ruy01] Ruys, T. C.: Towards Effective Model Checking. Doktorarbeit, Univ. Twente, 2001.

[SK91] Socolofsky, T.; Kale, C.: RFC 1180 - A TCP/IP Tutorial. Spider Systems Limited, URL: http://www.faqs.org/rfcs/rfc1180.html, 1991.

[Svo89] Svobodova, L.: Implementing OSI Systems. IEEE Journal on Selected Areas in Communications 7 (7), 1989, S. 1115-1130.

[Tan03] Tannenbaum, A.: Computer Networks. 4th edition, Prentice-Hall, 2003.

[RK03] Rothmaier, G.; Krumm, H.: cTLA 2003 Description. Internal Technical Report, AG RvS, FB Informatik, Universität Dortmund. URL: http://ls4-www.cs.uni-dortmund.de/RVS/MA/hk/cTLA2003description.pdf, 2003.

[RPK04] Rothmaier, G.; Pohl, A.; Krumm, H.: Analyzing Network Management Effects with SPIN and cTLA. In (Deswarte, Y.; Cuppens, F.; Jajodia, S.; Wang, L. Hrsg.) Proc. of IFIP 18th World Computer Congress, TC11 19th Int. Information Security Conference, 2004. Kluwer Academic Publishers, Boston; Dordrecht; London, 2004, S. 65-81.

[Rot04] Rothmaier, G.: cTLA Computer Network Specification Framework. URL: http://ls4-www.cs.uni-dortmund.de/RVS/MA/hk/framework.html, 2004.