A Framework Based Approach for Formal Modeling and Analysis of Multi-level Attacks in Computer Networks

Gerrit Rothmaier¹ and Heiko Krumm²

¹ Materna GmbH, Dortmund, Germany

gerrit.rothmaier@materna.de

² Universität Dortmund, Dortmund, Germany

krumm@cs.uni-dortmund.de

Abstract. Attacks on computer networks are moving away from simple vulnerability exploits. More sophisticated attack types combine and depend on aspects on multiple levels (e.g. protocol and network level). Furthermore attacker actions, regular protocol execution steps, and administrator actions may be interleaved. Analysis based on human reasoning and simulation only has a slim chance to reveal attack possibilities. Formal methods are in principle well-suited in this situation. Since complex scenarios have to be considered, however, high efforts are needed for modeling. Furthermore, automated analysis tools usually fail due to state space explosion. We propose a novel approach for modeling and analyzing such scenarios. It combines the high-level specification language cTLA with a computer network framework, optimization strategies, a translation tool, and the SPIN model checker. As a proof of feasibility we apply our approach to a multi-LAN scenario.

1 Introduction

Current trends [Ver04] show an increasing number of attacks. Especially attacks that are more sophisticated than just simple vulnerability exploitations are on the rise. Frequently, commonly deployed protocols and services are misused. This is facilitated by the fact that many basic protocols and services have long known security problems (cf. [Bel89]).

In the analysis of such attacks, multiple levels have to be considered simultaneously. For example, an attack on a routing protocol in a multi-LAN scenario may be feasible only with a specific update packet propagation. Propagation in turn depends on low level aspects like network topology and connectivity besides the protocol level flooding algorithm used. Furthermore, attacker position and administrator actions may matter as well. *Multi-level attacks* are attacks which combine and depend on aspects on multiple levels. Because of the complexity involved with simultaneously regarding multiple levels, manual analysis of such attack types is very difficult. Simulation techniques tend to provide good coverage only for very small systems. Thus the likelihood of finding potential attack sequences by means of simulation in a more extensive system is very low.

In this context we recognize the need for a suitable approach to formal modeling and automated analysis of complex attack types, especially multi-level attacks, in

computer networks. We aim to learn about attack processes, their impacts on the network components and their inferences with network administration processes. This shall help to predict relevant attack scenarios and provide means for the clear description of attacks, advice and countermeasures.

For our network modeling and attack analysis we resort to formal techniques. We combine the flexible and modular specification technique *cTLA* [HK00] with the model checker *SPIN* [Hol03], which is currently one of the most powerful and elaborated automated process system verification tools. *cTLA* supports the efficient and structured description of broad models. Using *cTLA2PC*, our translation tool, the *cTLA* specifications are transformed to the more low level process system description language *Promela*, which is input to the automated model checker *SPIN*.

We successfully applied a preliminary version of our approach to the analysis of low-level *ARP* attacks and administrator actions in a single LAN scenario [RPK04]. When starting with larger and more complex examples we, however, experienced high model development efforts. Initial analysis runs showed serious state explosion effects exceeding given memory limits. Therefore we developed further enhancements of our approach with respect to two aspects: First, we developed a *cTLA*-based modeling framework. It defines suitable network model architecture principles and supplies corresponding re-usable model elements thus supporting the efficient development of models. Second, we investigated *SPIN*'s automated model evaluation procedure in order to develop model optimizations. *SPIN* performs model evaluation by means of the computation of the set of reachable system states. This corresponds to the execution of all possible system behavior steps. Thus *SPIN* executes the model, and suitable model optimization conceptions can be adopted from the field of efficient protocol implementation (cf. [Svo89]). In particular, activity thread approaches – coarsening the execution step granularity – and buffer management approaches – reducing the size of interface parameters – were applied. Furthermore, *cTLA2PC* has been optimized to produce *Promela* code with less possible execution traces and to save state components. After these refinements, the modeling and automated analysis of the example scenario described in this paper became possible.

This paper reports on our refined formal modeling and analysis approach. Its main focus is the presentation of the modeling framework and the model optimizations. Section 2 deals with related work. The modeling framework is described in section 3. Section 4 outlines the *cTLA* modeling language. Subsequently, as a proof of feasibility, a medium size example scenario is considered in section 5. Section 6 gives an overview of optimizations on different levels and describes the results of the automated scenario analysis with *SPIN*. Concluding remarks are given in section 7.

2 Related Work

Approaches for the formal analysis and verification of security properties can generally be structured in program and protocol verification. Program verification shall enhance the trustworthiness of software systems (e.g. [BR00]). Protocol verification shall detect vulnerabilities of cryptographic protocols (e.g. [Mea95]). In both fields, a variety of basic methods are applied including classic logic and algebraic calculi (e.g. [KK03]), special calculi (e.g. [BAN89]), and process system modeling techniques

(e.g. [LBL99]). Moreover the approaches differ with respect to tool support. Many tools take advantage of existant more general analysis tools like *Prolog*, expert system shells, theorem provers, algebraic term rewriting systems, and reachability analysis based model checkers like *SPIN* (e.g. [RRC03]). Some powerful tools combine several analysis techniques [Mea96].

Formal modeling and analysis of network attack sequences combining aspects on different levels – e.g. protocol and network level – is a relatively new field. Ramakrishnan and Sekar [RS02] report on the analysis of vulnerabilities resulting from the combined behavior of system components. They consider a single host only, however. A process model is used which is specified in a *Prolog* variant. In the focus are simple security properties, expressed by labeling states safe and unsafe. A custom model checker built on top of a *Prolog* programming environment searches execution sequences which lead to unsafe states and correspond to vulnerability exploitations. Noel et al [NBR02] report on *topological vulnerability analysis*, i.e. checking networks of hosts for vulnerability combinations. A network is statically modeled by a set of tables representing hosts, predefined vulnerabilities, attacker access levels per host, and a connectivity matrix. The analysis investigates the possible combinations of the given vulnerabilities and their stepwise propagation in the network using *SMV* [Ber01]. Protocol level aspects, however, can only be modeled in a very limited way using symbolic constants in the connectivity matrix.

With respect to the simplification of models, our work is on the one hand related to techniques for efficient protocol implementation which were developed about 20 years ago in the course of the upcoming high-speed communication. In particular we learned from the activity thread implementation model which schedules activities of different protocol layers in common sequential control threads [Svo89], and from integrated layer processing which combines operations of different layers [AP93]. On the other hand, approaches for direct *SPIN* model reductions have to be mentioned. Particularly, Ruys [Ruy01] introduces low-level *SPIN*-specific optimizations. Partial order reductions, proposed by Alur et al. [ABH97], have a strong relationship to the activity thread implementation model providing the basis for the elimination of non-deterministic execution sequences.

3 Framework

Designing computer network specifications suitable for automated analysis is difficult. Particularly, the right abstraction level must be chosen. On the one hand, relevant details of a given scenario have to be captured. On the other hand, detailed models naturally have a very large state vector. Furthermore, the number of potential states grows exponentially with the state-vector size. Thus even a small addition to the model can lead to a prohibitive increase in the number of states. This effect is known as *state space explosion*. Automated analysis is rendered impossible because given time and memory constraints are quickly exceeded.

We aim to support the design process for computer network specifications in a way which fosters reuse of tried and tested abstractions and simplifies key design decisions. In the world of object-oriented programming patterns and frameworks are used for this task (cf. Gamma's definition in [GHJ95, pp. 26-27]). Thus we decided to

carry over the framework concept to computer network specifications. While designing computer network specifications for different scenarios, we identified common architectural elements. These elements form the basis of the *cTLA* computer network modeling framework. It defines both basic structure, i.e. typical elements like nodes, interfaces and media with their coupling, and basic behavior, i.e. sending and receiving actions, of computer networks. A specific model has to add its own elements (e.g. *RIP* capable nodes), but the overall architecture is given by the framework.

Fig. 1. Framework's large scale network view

3.1 Fundamentals

Our framework is based on the large scale view of computer networks exemplified by **Figure 1**. It shows two zones, each containing several nodes. The nodes inside a zone can directly communicate with any other node in the same zone. Zones can also be interpreted as network broadcast segments or subnets. All active network elements are modeled by *nodes*. Nodes communicate using *interfaces*. A node is said to belong to a zone if it has an interface in the zone. Interfaces transmit *packets* over *media*. Media is partitioned according to the zones. The two zones are connected by a *transfer node*. A *transfer node* is a special node with at least two interfaces in two different zones. It enables inter-zone communication. A node with just one interface is a *host node*.

On a small scale or node-oriented view packets are processed, sent and received through actions. Inside a node, the packet processing is structured into layers. A valid packet that is received from media by an interface is stored in the interface's receive buffer and then processed through the layers (action *rpcs*). A packet which shall be sent is processed (action *spcs*) the other way around, down the layers until it has reached the interface level. A node's *snd* and media's *in* actions respectively node's *rcv* and media's *out* actions are coupled. If media does not already contain a packet from the zone it can be sent (i.e. moved from the node's to media's packet buffer). The exact layer and address types used in a node vary depending on the specific node and scenario.

Most of today's computer networks employ the internet standard protocols. Thus our framework particularly supports *TCP/IP* based node types.

3.2 Architecture

An overview of the framework as a *UML* class diagram is depicted in **Figure 2**. The most current version of the framework's elements can be retrieved via our web site [Rot04].

According to the syntactical level of the elements, the framework is structured into the packages *Enumerations & Functions* (1), *Data Types* (2), and *Process Types* (3). The package *Enumerations & Functions* is used to define the network topology, initial address assignment and protocols desired for a model. For example, the enumeration *ZoneIdT* contains the model's zones, the function *fSrcToIa* assigns the initial addresses and the enumeration *ProtocolT* lists the required protocols.

The package *Data Types* contains common date types for interfaces, packets and buffers used throughout the framework. For instance, the type *InterfaceT* combines attributes of an interface, *PacketT* is used to represent a packet and *PacketBufT* defines a buffer for packets.

Enumerations & Functions

Data Types

Process Types

Fig. 2. Framework structure

Finally, the package *Process Types* provides process types media and nodes. For example process type *RouterIpNode* contains the behavior of a *IP* transfer node and

`HostIpNode` represents a *IP* host node. Through inheritance behavior is specialized. For example, `ActiveHostIpNode` adds behavior for the processing and sending of packets.

From a functional viewpoint framework elements of several packages usually collaborate to model a conception. For example, a scenario's network topology is modeled by several functions (e.g. `fSrcToZone`) and enumerations (e.g. `ZoneIdT`), together with appropriate handling by processes (e.g. `Media`, `HostIpNode`, `RouterIpNode`) and their actions (e.g. `out`, `rcv`).

Another example is packets and their processing. Packets are sent to and received from `Media` by nodes using interfaces. Interfaces are represented through the `InterfaceIdT` data type, their send and receive buffer's with the `PacketBufT` data type. Packets themselves are mapped to the `PacketT` data type. A packet's interpretation depends on its protocol type (`PacketT.pt`, `ProtocolT` enumeration). Packets currently in processing at a node are stored between layers using the `SystemBufT` data type. The processing actions are defined via the `ActionT` enumeration, usually depending on the protocol type. Similarly addressing depends on intertwined framework elements as well: functions (e.g. `fSrcToIa`), enumerations (e.g. `IpAddressT`), data types (e.g. `PacketT`) and node process types.

4 cTLA

Compositional TLA (cTLA) is based on Lamport's *temporal logic of actions (TLA)* [Lam94]. Explicit notions of modules, process types and composition of process types [HK00] are added, however. The following section gives a short overview of the *cTLA 2003* process types, a more detailed description is contained in [RK03].

The *simple process type* defines a state transition system directly. The state space is spanned by the local variables of the process type. For example, in `HostIpNode` the variable `itf` is a user defined record type containing the network interface's attributes. The set of initial states is defined by the *Init*-predicate (*cTLA* keyword `INIT`). The *actions*, keyword `ACTIONS`, directly constitute the *next-state relation*. Syntactically, actions are predicates over state variables (referring to the current state, e.g. v), so-called primed state variables (referring to the successor state, e.g. v'), and action parameters. Parameters support the communication between processes.

An internal action defines a set of state transitions in exactly the same way as a normal action. Internal actions, however, cannot be coupled with actions of other processes. Thus in the composed system each internal action is accompanied by stuttering steps of all other system components.

A *process extension type* (keyword `EXTENDS`) specializes or augments other process types. This resembles inheritance in object-oriented programming. For example, the `ActiveHostIpNode` process type uses this mechanism to add the action `snd` to `HostIpNode`. The state space is spanned by the vector of all state variables of the extended processes plus extra state variables defined by the extending process. *Init*-predicates and actions are merged through logical conjunction.

The *process composition type* describes systems which are composed from subsystems (keyword `CONTAINS`). These subsystems may in turn be further subsystems or

sets of process instances. For example, the system process `IpMultiLevelExample` of the example scenario is defined using process composition. Its actions are conjunctions of contained processes' actions which have to be performed jointly in order to realize an action of the system. Each process can contribute to a system action by at most one process action. If a process does not contribute to a system action, it performs a *stuttering step*. In *cTLA 2003* [RK03] process stuttering steps do not have to be explicitly listed on the right hand side of system actions.

The state transition system which models an instance of a process composition type is defined indirectly. The state space of the composed process is spanned by the vector of all state variables of the contained processes. Its *Init*-predicate is the conjunction of the *Init*-predicates of the contained processes. The *next-state relation* is the disjunction of the system actions defined in the composing process itself.

The *TLA* formula describing a *cTLA* process system is equivalent to a conjunction of the process formulas and consequently a system implies its constituents.

5 Example Scenario

To demonstrate the feasibility of our approach, we describe the formal modeling and analysis of an example scenario. It consists of multiple hosts in different LANs which are interconnected by routers as depicted in **Figure 3**. In particular, the routers R1, R2, R3 are connected in a triangle fashion forming an internal backbone. We aim to model and analyze this scenario for attack sequences along the lines of the routing and tunneling protocol attack ideas described at the Blackhat Europe Conference [BHE01]. All hosts are *TCP/IP* nodes. The routers additionally run *RIP*, which is still one of the most widely used *interior gateway routing protocols*.

Fig. 3. Example scenario

5.1 Protocol Modeling

The *TCP/IP* modeling is inherited from the framework's `HostIpNode` and `RouterIpNode` process types. So it suffices to implement *RIP* handling with a new process type. `RipRouterIpNode` extends process type `RouterIpNode` – which can

only statically route – with actions for handling *RIP* updates and updating the routing table dynamically.

We only give a short overview of *RIP* here, a more detailed description is given by Perlman [Per00]. Each router uses a *routing information base (RIB)* to store entries of the form $(dst, nhop, itf, met)$. The field *dst* contains the final destination and *nhop* the *IP* address of the *next-hop*, i.e. the next router on the way to the destination. If the current router is directly connected to the final destination's network, this field contains a special value (*NHO_DIRECT* in our modeling). The *itf* field contains the interface connected to the next-hop or the final destination's network. The *met* field is used for storing a *cost metric*, usually the number of *hops*, from the current router to the final destination. A metric of 1 denotes that the current router is directly connected to the final destination's network. On the other hand, a metric of 16 := *MET_INF* means infinite cost.

RIP works with two stages, *input processing* and *output processing*. *Input processing* handles received *RIP* update packets. The critical element of a *RIP update packet* is the pair (dst', met') . A *RIP* update packet may contain several such pairs. For our modeling, however, without loss of generality, we assume that all update packets contain only one pair. The fields describe the best route (in terms of metric) to the destination as known by the router from which the packet originated. If the update packet passes basic sanity checks the packet is considered for updating the router's *RIB*. Usually the *RIB* will only be updated if the update packet's metric is better than the existing metric. A packet from the *next-hop* of an already existing route, however, is allowed to update the *RIB* even if the new metric is worse.

If a *RIB* entry has been changed (no matter which case applied), its *route change flag* is set. *Output processing* will then send a *triggered update*. We only model triggered updates, because regular updates are mainly useful for debugging purposes.

All updates are sent observing the *split horizon* principle. That means the updates are sent to all neighboring routers with the exception of the router from which this route was received (i.e. the *next-hop nhop*).

5.2 Network and System Modeling

We compose our system model using the framework's basic process types (e.g. *Media*) and the specific process types required for the example scenario (e.g. *RipRouterIpNode*). To simplify our modeling, the "straight through" routers *Rx* and *Ry* are not represented by *RipRouterIpNode* instances. We just represent them in the metric of the others routers. Furthermore, we select fixed host roles. *HA* models the attacker host, *H1* is the victim and *H2* its intended communication partner. The attacker *HA* is modeled by type *RipAttackerHostIpNode* which extends *ActiveHostIpNode* with the ability to create and send fake *RIP* update packets. *ActiveHostIpNode* is a basic *TCP/IP* processing node from the framework based on *HostIpNode*. The victim (*H1*) is represented by type *ActiveNonPromHostIpNode* and *H2* by *NonPromHostIpNode* type, both directly from the framework.

For each analysis run, we pick fixed locations for *HA*, *H1*, and *H2*. For example, the attacker, *HA*, is located in LAN 3, the victim, *H1*, is placed in LAN1 and *H1*'s communication partner, host *H2*, is a member of LAN 2 (cf. **Figure 3**).

We label each node's interfaces (e.g. i_1 , i_2 , i_3) and define the assigned *IP* addresses. For each router, the initial routing tables have to be defined. For example, R_1 is directly connected (i.e. metric 1) to zones ZBB12 and ZBB13 over interfaces 2 respectively 3. The route to Z2 is given indirectly via next-hop R_2 (interface 2) with metric 4 (i.e. R_1 , R_2 , R_x , R_y). We start with fully initialized routing tables. Finally, the *cTLA* modeling is completed with the definition of the system process type `Ip-MultiLevelExample`.

We still have to define the security property to be analyzed. *Promela* includes `assert` statements for the *SPIN* model checker. For simplicity, we check the following confidentiality property: `assert( (ha_itf.rpa.pkt.dat_ida == HA_I1_IA) );`

It expresses that host HA only receives packets which are destined for it. After translation of the scenario model to *Promela* with *cTLA2PC*, this assertion has to be inserted into the *Promela* representation of the `rcv_ha` action (i.e. host HA's non broadcast packet receive action). Broadcast packets can not trigger the assertion, because they are received by the `rbc` action instead of `rcv_ha`.

6 Optimizations and Analysis Results

After translation to *Promela* the example system had an initial state vector size of about 630 bytes. That was still too large for successful analysis. In *SPIN*'s full state-space search verification modes memory limits were exceeded quickly. Approximative verification modes did not produce any result in reasonable time. Thus we developed further optimization strategies.

6.1 Optimizations

Optimizations are generally feasible on several levels: Scenario level (1), *cTLA* level (2), *Promela* level (3), and on the verifier level through special *SPIN* verification options (4).

At the scenario level, abstractions are often most helpful. As described in section 5, fixed roles and attacker positions may suffice for each analysis run. Furthermore it may be possible to simplify the modeled protocols (e.g. by allowing only one update pair in each update packet). Such high-level optimizations often yield state space savings even before the specification design is started.

Regarding the second point, the modeling may be optimized at the *cTLA* level. For example, we are able to represent the nodes using fewer buffers and actions per layer with a smarter approach. Our framework's node modeling follows the *activity thread* approach known from efficient protocol implementation [Svo89]. Actions and working buffers of different layers which process the same packet can be combined. This approach requires less state space than a naive layer-by-layer modeling. Because of the framework, this *cTLA* level optimization is inherited by our modeling as well. At this stage, the example scenario modeling has the afore-mentioned state vector of about 630 bytes. This is our starting point for further optimizations.

Action parameters can be optimized on the *cTLA* level, too. First, an equivalent flat system, which contains only a system process and system actions has to be generated. This expansion can be done with the *cTLA2PC* tool. Generally, *cTLA* action param-

ters correspond to existentially quantified variables. But in the flat system, for most action parameters value determining equalities exist. This is due to the fact that all system actions are coupled process actions. Input parameters of one underlying action are output parameters of another. Therefore, it is possible to replace these input parameters with the corresponding symbolic output value. This “*paramodulated*” *cTLA* version usually leads to a noticeable smaller state vector. For example, the example scenario after this step has a state-vector of about 580 bytes.

To enable model checking with *SPIN*, we translate the *cTLA* specification to *Promela* using the *cTLA2PC* tool. Thus we can optimize our scenario modeling at the *Promela* level as well. A comprehensive summary of low-level *Promela/SPIN* optimizations is described by Ruys [Ruy01]. For instance, *SPIN*’s built-in bit-arrays should be avoided. Mapping bit-arrays into bytes using macros decreases the required state space for such variables up to a factor of eight. *cTLA2PC* supports several such optimizations through specific switches during the translation process. With these optimizations the state-vector for our scenario decreases to about 320 bytes.

Furthermore, we acquire extra information from the *cTLA* origin of the *Promela* model. Thus more advanced optimizations are possible. From the *cTLA* specification we have a structuring of the model into parametrized system actions. The execution of these actions can be arbitrarily interleaved, but each action is atomic. So a single *Promela* process embedding all actions in a non-deterministic `do-loop` suffices. Because *cTLA* actions are deterministic, each action is embraced by the *Promela* `d_step` keyword. This optimization does not improve the state-vector, but reduces the required search depth for a specific sequence.

For the action parameters, we at first used *input generator processes*. For each parameter type input generator processes according to the maximum multiplicity of the parameter are needed. Each input generator sets a matching global variable to a random value. Because a model checker has to take each potential execution path into account, all possible parameter values are covered. The input generator approach works fine, but so far automated analysis efforts of the scenario described in this paper failed. We estimated that an attack sequence for the scenario would require about 32 steps, including about 10 *input generator steps* (i.e. value settings). But test runs of our model showed that *SPIN* exceeded 3 GB of memory (i.e. the practical x86 per process memory limit) after reaching search depth 23.

Thus we invented a totally different approach to handle parametrized actions. No input generators and no parameter variables are used. Instead, all parametrized actions are completely “unrolled”. This means that each parametrized action is replaced by copies with fixed parameters for all possible parameter values. Because all variable types in *cTLA* (and in finite-state specification languages) are finite, this is accomplishable.

Of course, the *unroll approach* leads to a lengthy *Promela* level specification, but *SPIN* copes much better with such a specification than with a specification using the input generator approach. The size of the state-vector is not reduced very much, but the number of processes, actual transitions and the search depth required for a specific sequence are distinctly smaller, because no input generator steps exist.

Finally, *SPIN* supports different options during verification. For example, different *verification modes* (i.e. full state space search, approximative bit-state hashing) and state-vector representation schemes (i.e. compression, graph encoding) are available.

In our experience, these options offer only modest improvements relative to the default settings and each other. Thus most of the time the feasibility of automated analysis does not depend on the right choice of *SPIN* options, but much more on careful optimizations of the modeling.

6.2 Results

Following the optimizations described above we were able to reduce the state-vector for the example scenario to about 50% of its initial size (316 vs 630 bytes). Because input generator steps are no longer needed (unroll optimization), the depth required for a specific sequence is reduced as well.

Automated analysis of a scenario with *SPIN* requires that properties which shall not be violated (i.e. invariants) are included in the *Promela* specification. Then, *SPIN* can check if a sequence leading to a violation of such a property exists. The violation is saved to a trail file. Using *SPIN*'s guided simulation feature this trail file can be expanded to the sequence of executed *Promela* statements. A special feature of *cTLA2PC* (option `--trace-points`) helps to map the *Promela* statements back to actions in the original model.

```
pan: assertion violated (ha_itf.rpa.pkt.dat_ida==11) (at depth
    21)
pan: wrote ip-multilevel-example-veri-flat-para.promela.trail
    (SPIN Version 4.2.0 -- 27 June 2004)
[...]
State-vector 316 byte, depth reached 21, errors: 1
5.5768e+06 states, stored
5.5768e+06 nominal states (stored-atomic)
4.28228e+08 states, matched
4.33804e+08 transitions (= stored+matched)
[...]
```

Listing 1. *SPIN* verifier output (assertion violated!)

We analyzed the described modeling for assertion violations using *SPIN* on a standard PC system. After about 40min and requiring slightly under 1 GB of RAM *SPIN* found an attack sequence of depth 21 (cf. **Listing 1**) violating the specified confidentiality property.

Using *SPIN*'s guided simulation feature and mapping of the results back to *cTLA* actions identified the attack sequence depicted in **Figure 4** (slightly simplified for clarity). The discovered attack sequence corresponds nicely with the routing and tunneling protocol attack ideas from the Blackhat Europe Conference [BHE01]. In a nutshell, after system initialization the host H1 submits a packet destined for host H2 to its router R1. Then, the attacker HA broadcasts a *RIP* update packet advertising a new route to zone Z2 with metric 1 from HA. This packet is then accepted and processed by router R3, which updates its routing table (new route to Z2 via HA has metric 2 = HA's metric + 1, existing route via R2 has metric 4) and prepares triggered *RIP* updates to the other zones (without the originating zone because of the *split horizon* principle). R3 then broadcasts the triggered update packet to zone ZBB13. R1 receives and processes the packet and updates its routing table (the new route to Z2 with

metric 3 via R3 is better than the existing route with metric 3 via R2). Then R1 forwards the packet from H1 according to the changed table (next-hop R3) and sends it. R3 receives the packet and forwards it to its next-hop, the attacker, HA. Thus HA receives a packet from H1 to H2, violating the assertion.

Fig. 4. Attack sequence found by SPIN

Attack sequences depend on the attacker position, update propagation, and initial routing tables which in turn depend on low-level network topology aspects. For example, an attack sequence like the one shown in **Figure 4** is not possible if H2 is connected directly to R2. First, the initial routing tables have other metrics. Second, the metric of the forged *RIP* update packet from HA is increased by each router on its way to R3. Thus the new route would not be better than the existing route, and R3's routing table would not be updated.

Because of the breadth first search the described sequence is minimal. Thus only necessary steps are included. Further variants are possible. For example, R3 will usually broadcast the triggered update packet to R2 as well. Because this step is not required for the violation of the stated confidentiality property, it is not included in the 21 step sequence.

Practical tools to facilitate attacks on routing protocols are widely available, e.g. [FX01]. More secure protocols (e.g. *S-RIP* [WKO04]) have been suggested but are rarely deployed. Widespread deployment is hindered by the usually required computationally expensive cryptographic operations. Furthermore, interoperability is a key requirement in heterogeneous networks but standardization of new protocols is a lengthy process.

7 Concluding Remarks

Attack types combining and depending on aspects on multiple levels, especially protocol and network levels, form an interesting class of advanced network attack types.

They can be modeled and analyzed by means of formal process systems. The presented approach combines the high level specification language *cTLA* with a modeling framework, a translator tool (*cTLA2PC*), model optimization strategies, and the well-known model checker *SPIN*. Our results show that the analysis of practically relevant attack types in medium size scenarios is feasible.

Current work aims to a further enhancement of our approach which shall eventually enable the automated prediction of new network attack strategies. For that purpose framework extensions and additional model optimization strategies are under development. Moreover, model development and analysis experimentation will be supported by a workbench assembling the tool set. Another interesting aspect to be studied further is the integration of logical deduction-based analysis procedures which can exploit *TLA*'s symbolic logic proof techniques.

References

- [AP93] M. Abbott, L. Peterson: *Increasing network throughput by integrating protocol layers*. IEEE/ACM Transactions on Networking, 1 (1993) 600–610.
- [ABH97] R. Alur, R.K. Brayton, T.A. Henzinger, S. Qadeer, S.K. Rajamani: *Partial order reduction in symbolic state space exploration*. In Proc. of CAV 97: Computer-Aided Verification, LNCS 1254, pp. 340-351. Springer-Verlag, 1997.
- [BR00] M. Balser, W. Reif et al.: *Formal System Development with KIV*. In: T. Maibaum (ed.), Fundamental Approaches to Software Engineering. Springer LNCS 1783, 2000.
- [Bel89] S. M. Bellovin: *Security Problems in the TCP/IP Protocol Suite*. Computer Communication Review, Vol. 19, No. 2, pp. 32-48, 1989.
- [Ber01] S. Berezin: *The SMV web site*. URL: <http://www.cs.cmu.edu/~modelcheck/smv.html/>, 1999.
- [BHE01] Blackhat Europe Conference: *Routing and Tunneling Protocol Attacks*, URL: <http://www.blackhat.com/html/bh-europe-01/bh-europe-01-speakers.html#FX>, 2001.
- [BAN89] M. Burrows, M. Abadi, R. Needham: *A Logic of Authentication*. In: Proceedings of the Royal Society, Volume 426, Number 1871, 1989, and in: William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.
- [CM88] K.M. Chandy, J.Misra: *Parallel Program Design - A Foundation*, Addison-Wesley, Reading, 1988.
- [FX01] FX of Phenoelit: *Internet Routing Protocol Attack Suite*, URL: <http://www.phenoelit.de/irpas>, 2001.
- [GHJ95] E. Gamma, R. Helms, R. Johnson et al: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [HK00] P. Herrmann, H. Krumm: *A framework for modeling transfer protocols*. Computer Networks, 34(2000)317-337.
- [Hol03] G. J. Holzmann: *The SPIN model checker*. Addison Wesley, 2003.
- [KK03] K. Kawachi, S. Kitazawa et al.: *A Vulnerability Assessment Tool Using First-Order Predicate Logic*. IPSJ SIGNotes Computer Security No.019 (2003)
- [Lam94] L. Lamport: *The temporal logic of actions*, ACM Trans. Prog. Lang. and Sys. 16 (1994) 872-923.

- [LBL99] G. Leduc, O. Bonaventure, L. Leonard et al: *Model-based verification of a security protocol for conditional access to services*. Formal Methods in System Design, Kluwer Academic Publishers, vol.14, no.2, March 1999 p.171-91.
- [Mea95] C. Meadows: *Formal Verification of Cryptographic Protocols: A Survey*. In: Proc. Asiacrypt, Int. Conf. on Theory and Application of Cryptology, LNCS, Springer-Verlag, 1995.
- [Mea96] C. Meadows: *The NRL Protocol Analyzer: An Overview*. Journal of Logic Programming, 26, 2 (1996) pp. 113-131.
- [NB02] S. Noel, B. O' Berry, R. Ritchey: *Representing TCP/IP connectivity for topological analysis of network security*. Computer Society, IEEE (ed.), Proc. of the 18th Annual Computer Security Applications Conference, Dec. 2002, p. 25-31.
- [Per00] R. Perlman: *Interconnections*. 2nd Edition, Addison-Wesley, 2000.
- [RRC03] P. Romano, M. Romero, B. Ciciani, F. Quaglia: *Validation of the Sessionless Mode of the HTTPR Protocol*. Proc. of IFIP 23rd International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'03), LNCS 2767, Springer-Verlag 2003.
- [RS02] C. Ramakrishnan, R. Sekar: *Model-Based Analysis of Configuration Vulnerabilities*. Journal of Computer Security, Vol. 10, Nr. 1, Jan. 2002, p. 189-209.
- [RK03] G. Rothmaier, H. Krumm: *cTLA 2003 Description*. Internal Technical Report, URL: <http://ls4-www.cs.uni-dortmund.de/RVS/MA/hk/cTLA2003description.pdf>, 2003.
- [RPK04] G. Rothmaier, A. Pohl, H. Krumm: *Analyzing Network Management Effects with SPIN and cTLA*. Proc. of IFIP 18th World Computer Congress, TC11 19th International Information Security Conference, pp. 65-81, 2004.
- [Rot04] G. Rothmaier: *cTLA Computer Network Specification Framework*. Internal Report. URL: <http://www4.cs.uni-dortmund.de/RVS/MA/hk/framework.html>
- [Ruy01] T. C. Ruys: *Towards Effective Model Checking*. PhD Thesis, Univ. Twente, 2001.
- [Svo89] L. Svobodova: *Implementing OSI Systems*. IEEE Journal on Selected Areas in Communications 7 (1989), pp.1115-1130, 1989.
- [Ver04] Verisign: *Internet Security Intelligence Briefing / November 2004 / Vol. 2 / Issue II*. URL: http://www.verisign.com/Resources/Intelligence_and_Control_Services_White_Papers/internet-security-briefing.html
- [WKO04] T. Wan, E. Kranakis, P.C. van Oorschot: *S-RIP: A Secure Distance Vector Routing Protocol*. In Proc. of Applied Cryptography and Network Security (ACNS'04), Yellow Mountain, China, 2004.