Finding Security Vulnerabilities in a Network Protocol Using Parameterized Systems

Adi Sosnovich[1], Orna Grumberg[1], and Gabi Nakibly[2]

[1] Computer Science Department, Technion, Haifa, Israel
{sadisos,orna}@cs.technion.ac.il
[2] National EW Research and Simulation Center, Rafael, Haifa, Israel
gabin@rafael.co.il

Abstract. This paper presents a novel approach to *automatically* finding security vulnerabilities in the routing protocol OSPF – the most widely used protocol for Internet routing. We start by modeling OSPF on (concrete) networks with a fixed number of routers in a specific topology. By using the model checking tool CBMC, we found several simple, previously unpublished attacks on OSPF.

In order to search for attacks in a *family of networks* with varied sizes and topologies, we define the concept of an *abstract network* which represents such a family. The abstract network $\mathcal{A}$ has the property that if there is an attack on $\mathcal{A}$ then there is a corresponding attack on each of the (concrete) networks represented by $\mathcal{A}$.

The attacks we have found on abstract networks reveal security vulnerabilities in the OSPF protocol, which can harm routing in huge networks with complex topologies. Finding such attacks directly on the huge networks is practically impossible. Abstraction is therefore essential. Further, abstraction enables showing that the attacks are *general*. That is, they are applicable in a large (even infinite) number of networks. This indicates that the attacks exploit *fundamental vulnerabilities*, which are applicable to many configurations of the network.

1 Introduction

This paper presents a novel approach to automatically finding security vulnerabilities in the routing protocol *Open Shortest Path First* (OSPF) [14]. OSPF is the most widely used protocol for Internet routing, thus finding vulnerabilities which are inherent to the design of the protocol is significant for Internet security. Manually identifying vulnerabilities in a complex protocol such as OSPF is a hard task which requires deep understanding and close acquaintance with the protocol.

We propose to find vulnerabilities *automatically* by using model checking techniques. In order to use model checking for our purpose we build a model for the protocol when running on a given network topology; we include in the model an attacker with predefined capabilities; and we specify the absence of a state in which an attack succeeds (to be defined later). If the model checker finds a state violating the specification, it returns a counterexample leading to that state. The counterexample being a run of the protocol is, in fact, an *attack* on the protocol.

A high level description of the OSPF protocol is given below. OSPF runs on each router in a network of routers. Its goal is to distribute the full network topology to all

N. Sharygina and H. Veith (Eds.): CAV 2013, LNCS 8044, pp. 724–739, 2013.

routers. The routers send each other messages describing their partial view of the network topology. When a router gets a message from its neighbor, it updates its database accordingly and *floods* the message on to all of its *other* neighbors. OSPF includes a mechanism for fighting against possible attacks. If a router gets a message in its own name that it did not originate, then the router initiates a *"fight back"* message in order to correct the topology view of all other routers.

We start by modeling (concrete) networks with a fixed number of routers in a specific topology, where each router runs the OSPF protocol. The *attacker* is one of the routers running the same protocol, except that it can also send *fake* messages in the name of other routers, and can ignore messages sent to it. A *state* of the model consists of the databases and message queues of all routers in the network. We say that an *attack succeeds* in a state if (at least) one of the routers has a fake message in its database, and no router has a message waiting to be sent. This means that no fight back is going to change the fake topology view of this router. Thus, the attack is persistent.

We ran the model checking tool CBMC [2] on several topologies. We note that the OSPF protocol is quite elaborate. Further, the size of the database of each router is proportional to the size of the network. We therefore limited the topology sizes in order to fit in the model checker capacity. Nevertheless, we have found several simple, previously unpublished attacks. We also found a more subtle attack which was already published. The vulnerabilities revealed by the attacks we found are known and accepted by OSPF experts.

The limitation of the approach described so far is clear. It can only check a specific and small network topology which may expose only a part of the protocol's functionality. In order to allow for a good coverage of the protocol's functionality many other specific topologies need to be checked, taking more time and computing resources.

We therefore develop an approach which can search for attacks in a *parameterized network*, consisting of a *family of networks* with varied sizes and topologies. We define an *abstract network*, that represents such a family. The abstract network $\mathcal{A}$ has the property that if there is an attack on $\mathcal{A}$ then there is a corresponding attack on *each* of the (concrete) networks represented by $\mathcal{A}$. An abstract network allows to reveal security vulnerabilities in the OSPF protocol, which can harm routing in huge networks with complex topologies. Finding such attacks directly on the huge networks is practically impossible. Abstraction is therefore essential.

The abstraction is defined on all levels of the model: We define an abstract topology which represents a family of concrete topologies. An abstract state represents a set of concrete states. The correspondence between abstract transitions and their concrete counterpart is more subtle. Each abstract transition represents a set of finite concrete *runs*, one in each of the concrete topologies represented by A. As a result, our abstract model is unusual: It under-approximates each member in a family of concrete models. That is, every run of the abstract model has a corresponding run in each of the concrete models represented by it. This is an important characteristics of our abstraction as it allows us to find *general* attacks on an abstract network which are manifested in each of the concrete models it represents. Thus, these attacks are applicable in a large (even infinite) number of networks. This indicates that they exploit fundamental vulnerabilities, which are applicable to many configurations of the network. This is in contrast to finding a specific attack that is only applicable for a single perhaps marginal network configuration.

In this part, we have found attacks on abstract networks manually. However, our abstract model can be implemented for instance in C to be used with CBMC, similarly to our implementation of the concrete model.

It should be noted that in principle, more attacks could be found on a concrete system that belongs to a family. However, in this work we are interested in finding general attacks, that are robust to changes in the topology. These are usually the first attacks a network operator would like to know with regard to its network.

We emphasize that the contributions of this work go beyond the security analysis of OSPF. The abstract concept and definition can be beneficial for finding security vulnerabilities in other protocols as well.
To summarize, the contributions of this work are:

- We analyzed the OSPF routing protocol and *automatically* found attacks on it.
- We found *general* attacks which are applicable to families of networks and demonstrated *security vulnerabilities* in the OSPF protocol.
- We developed a novel technique for *parameterized networks* which is suitable for finding a counterexample (in our case an attack) on each member of the family.
- This work is a first step towards finding security vulnerabilities in other distributed network protocols.

1.1 Related Work

There are a few works that present a security analysis of the OSPF protocol. Most such works (e.g., [17,18,7,15]) focus on LSA falsification attacks. Only two past works ([7] and [15]) present OSPF attacks with a persistent effect while evading a fight-back. This low number of works stands in contrast to the centrality of OSPF to Internet routing. This can be partially explained by the difficulty to do a manual and thorough security analysis of complex distributed network protocols.

There are some works that propose a security analysis of the design of network protocols based on model checking (e.g., [12,13,9]) . All past works check a given network configuration with a predetermined set of participants. In particular, some works (e.g., [11,5,10]) analyzed the security of OSPF and other routing protocols, while considering only a given network model. As other distributed network protocols the functionality of a routing protocol is highly dependent of the number of participants in the protocol and the network topology. Hence, current works that employ model checking for distributed network protocols may not cover the entire protocol's functionality.

Reasoning about families of systems, also known as *parameterized systems*, is a known research area (e.g. [6,8,4,16,1]). Most works present an abstract model which *over-approximates* all members in the family and is used to verify that they all satisfy a given property. We, on the other hand, define an abstract model which *under-approximates* each member in the family. Our abstract model is therefore most suitable for finding attacks on all members. To the best of our knowledge, no similar reasoning has been applied before to parameterized systems.

2 Modeling OSPF

2.1 OSPF Basics

The Internet is clustered into sets of connected networks and routers called Autonomous Systems (AS). Each AS is administered by a single authority, such as a large organization, or an Internet service provider. Within each AS a routing protocol is run. Its aim is to allow routers to construct their routing tables, while dynamically adapting to changes in the AS topology. Open Shortest Path First (OSPF) [14] is currently used within most ASes on the Internet. It was developed and standardized by the IETF organization.

Each OSPF router composes a list of all its links to neighboring routers and their costs. This list is termed *Link State Advertisement* (LSA). Each LSA is flooded throughout the AS. Every router compiles a database of the LSAs from all routers in the AS, thus having a complete view of the AS topology. This allows a router to calculate the least cost paths between it and every other router in the AS. As a result, the router's routing table is formed.

A new instance of each LSA is advertised periodically every 30 minutes, by default. Every LSA has a sequence number which is incremented with every new advertised instance. A more recent LSA instance with a higher sequence number will always take precedence over an older instance with a lower sequence number. An LSA includes the following fields: a) *src* - the router which just sent the LSA; b) *dest* - the router to which the LSA is destined; c) *orig* - the router which first advertised the LSA; d) *seq* - sequence number.

Two routers in the AS may be connected over a *point-to-point* link. A subset of two or more routers may be connected over a *transit network*. One router in every transit network is selected to act as a *designated router*. During the flooding of an LSA each router sends the LSA to all its neighbors (except the neighbor from which the LSA was received). To alleviate flooding load this rule has an exception: a non-designated router may flood an LSA over a transit network only to the designated router of that network. The designated router will send it to all the other routers in that transit network. Note that a router will only receive an LSA from one of its neighbors. An LSA having a *src* that is not one of the router's neighbors will be dropped.

A common goal for an OSPF attacker is to advertise a fake LSA on behalf of some other router in the AS. Such an attack changes the view other routers have of the AS topology and consequently changes their routing tables. The primary measure by which OSPF defends against such attacks is the "*fight-back*" mechanism. Once a victim router receives an instance of its own LSA which is newer than the last instance it originated, it immediately advertises a newer instance of the LSA with a higher sequence number which cancels out the false one. This mechanism prevents most OSPF attacks from persistently falsifying an LSA of another router. Another defense measure is the authentication of LSAs using a secret key shared by the routers of the autonomous system. An outside router that does not know the shared secret can not send LSAs to routers inside the autonomous system.

2.2 The Concrete Model

In the following we present the concrete model for OSPF we used to find attacks. We note that our model is a simplified version of the real OSPF.

Our model assumes as a starting point a stable routing state in the AS. Namely, all the routers advertised their LSAs and calculated their routing tables. In particular, no LSA flooding is in progress or about to start. The LSA databases of all routers are complete and identical. Without loss of generality we assume that the sequence numbers of all the LSAs that have been advertised are 0. In addition, designated routers for all transit networks have been selected. The model is composed of three entities: (AS) *topology* which models a concrete topology of the AS, *Router* which models a legitimate router inside the AS; *Attacker* which models a malicious router inside the AS.

Autonomous System Topology Model. We denote the concrete topology by $T_c = (R, S, E, DR_c)$, where R is the set of routers, $S \subseteq 2^R$ is the set of transit networks, which we refer to as *sub-network*, $E \subseteq R \times R$ is a set of undirected edges, each representing a point-to-point link between two routers, and $DR_c : S \to R$ maps sub-networks to their designated routers. For simplicity of presentation we assume that each router belongs to at least one sub-network. We emphasize that the routers forming a sub-network are directly connected to each other as if they were forming a clique. Nonetheless, those connections are not part of the set E which only includes point-to-point links. Figure 1 depicts an example of a topology.

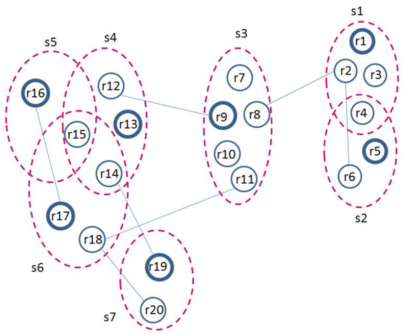

Fig. 1. The concrete topology T_C. The dashed circles marked as s_i are sub-networks, the circles r_i are routers, and lines connecting routers are edges. Bold circles represent designated routers.

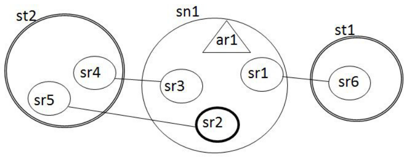

Fig. 2. Abstract topology T_A (see Section 3). The circles marked as sr_i represent singleton routers; the triangle ar_1 represents an abstract router; the circle sn_1 represents an abstract sub-network; and the double circles st_i represent sub-topologies. The bold circle represents a designated router (i.e., sr_2 is the designated router in the sub-network sn_1).

Router Model. The router model executes the standard functionality of the protocol. We model only part of the functionality defined by the OSPF standard since a large model might be infeasible for model checking. Nonetheless, our model captures the protocol's essential operations which any attack must exploit. For example, flooding by its very nature must be exploited by any attack that aims to advertise false LSAs. The functionality we modeled includes: (1) LSA message structure. (2) Flooding procedure. (3) Designated router logic. (4) Fight-back mechanism.

We do not model the actual contents of each LSA, i.e. the list of advertised links and their costs, because the LSA content has no material affect on the attack technique used to advertise a fake LSA. Figure 3 gives a high level overview of the router procedure.

Attacker Model. In our work we assume that an attacker is one of the routers of the autonomous system. Other routers treat the attacker as a legitimate router. The attacker is free from the protocol's standard and is able to ignore incoming messages and to originate messages arbitrarily. In particular, an attacker may originate fake LSAs on behalf of other routers in the topology. The model indicates such LSAs by a special *isFake* flag, which is not part of the OSPF standard, and legitimate routers do not make use of it. This flag allows us to easily define the specifications for the model (see section 2.4). Note that since the attacker has control of a legitimate router, the attacker knows the secret key used to authenticate the LSA messages.

Another important capability of the attacker is sending an LSA to a non-neighbor destination through several links without being opened on the way. Thus, the intermediate routers will not process the message. We call this *unicast* sending. This is a trivial capability that is inherent to any IP network. Every router (malicious or benign) can send messages directly to remote routers. However, regular routers following the OSPF protocol do not use this capability when flooding LSA messages.

```
if (r.Q not empty)
{
  m = pop-head(r.Q)
  if (m.dest ≠ r)
    send m according to r's routing table
  else //m.dest is r
  {
    if (m is newer than the copy in r.DB)
    {
      if (m.orig == r)
        fight-back
      else
        update r.DB and flood m
    }
    else
      ignore m
  }
}
```

Fig. 3. A sketch of the router r procedure. $r.Q$ denotes r's incoming message queue. A message $m = (src, dest, orig, seq)$. $r.DB$ denotes the set of LSA instances currently installed in r's database.

2.3 Formal Model for OSPF

The formal model we use for OSPF is a finite state machine with global states and transitions. In order to obtain a finite model suitable for model checking, we impose a predefined bound SB on the sequence number of messages, and a predefined bound K on the queue size of each router. It should be noted that in real OSPF such bounds exist as well. The queue of each router consists of up to K messages of the form $m = (src, dest, orig, seq, isFake)$, taken from the message domain $M = R \times R \times R \times \{0, ..., SB\} \times \{T, F\}$. The database of router r, $r.DB : R \rightarrow \{0, \ldots, SB\} \times \{T, F\}$, includes for each router r' the sequence number of the last message that was originated by r' and reached r , and the value of the flag *isFake* indicating whether this message was in fact originated by the attacker and not by r'. A global state $\sigma = \{r.DB \,|r \in R\} \cup \{r.Q \,|r \in R\}$ consists of a database and a message queue for each of the routers in the topology, including the attacker.

An r-*transition* between two global states corresponds to an application of the router r procedure (which is either the procedure given in Figure 3 if r is a regular router, or the attacker's procedure if r is the attacker). Note that an r-transition may change, in addition to the queue and the database of r, the queues of some of its neighbors. A *run* of the model consists of a sequence of global states $\sigma_1, \ldots \sigma_n$, such that for each i, a router r from R is chosen nondeterministically, and an r-transition is applied to σ_i, resulting in σ_{i+1}.

2.4 Specification

Our aim is to discover attacks on OSPF that allow an attacker to persistently falsify LSAs of legitimate routers. Our specification for the absence of a successful persistent attack requires that each state will satisfy at least one of the following two conditions:

1. No router has a fake LSA in its database.
2. At least one message resides in a router's queue.

The first condition verifies that the attacker has not fooled another router to install a fake LSA. The second condition relates to the attack's persistency. If not all the routers' queues are empty then the router whose LSA has been falsified might still fight back and revert the effect of the attack. Note that a state which violates the specification defines the outcome of a successful persistent attack regardless of a specific attack technique.

A model checker will search for a violation of the specification. When found, it will return a counterexample in the form of a run of the model which leads to a violating state. This run is actually an *attack* on OSPF.

2.5 Experimental Data

We have implemented in C our concrete model of OSPF, which is a simplified version of the protocol. The implementation is a rather small C program with a few hundreds of code lines. To find counterexamples, i.e. attacks, for which the above specification does not hold we use CBMC, a bounded model checker tool [2]. CBMC can check if a C program satisfies a specification along bounded

Table 1. For CNF formulas encoding topologies of different sizes, the number of variables and clauses in millions and the solving time in hours

#Routers	#Variables	#Clauses	Time
5	8M	21M	3.17h
6	17M	40M	7.07h
7	23M	55M	12.87h

runs. In our model, we bounded the number of cycles by 8, such that in each cycle any of the routers (including the attacker) can run their procedure once. In order to have a finite model which is rather small, we used a bound of $K = 4$ for the queue size, and a bound of $SB = 8$ for possible sequence numbers.

All our experiments were conducted on Intel Xeon X5650 with 32GB of memory. Table 1 details for several different network topologies of different sizes, the number of variables and clauses in the CNF formula generated by CBMC, and the time it took to solve the formula using the solver MiniSAT2 [3] .

2.6 Example of Attacks on OSPF

As mentioned before, when an attack is found the model checker CBMC outputs a path of global concrete states ending with a state that violates the specification. Figure 4 depicts an example of a topology with three sub-networks: $\{r1, r2\}$, $\{r3, r4\}$, and $\{r0\}$. $r1$ and $r4$ act as designated routers. The router $r0$ is attached to $r1$ and $r4$ using point-to-point links. In this topology $r3$ is the attacker. Note that although there are no edges between routers in the same sub-network, they are considered directly connected.

In the following we describe several attacks we found using the above concrete model having the topology depicted in Fig. 4. The first two attacks are simple albeit previously unpublished. The state explosion problem of the model checking impedes finding more complex attacks which may only be exhibited on larger topologies.

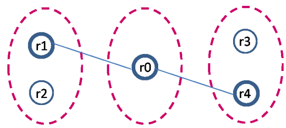

Fig. 4. A concrete topology

Recall that our model is a simplified version of the real OSPF. As the OSPF standard is given in an English manuscript, we cannot formally prove that our model is an under-approximation of the real OSPF. However, an OSPF expert validated that attacks found in our model are also valid in the full OSPF protocol.

Attack #1. The attacker ($r3$) originates a fake LSA on behalf of $r4$ directly to $r2$ (using unicast sending), while falsifying the source to be $r1$. The fields of the fake LSA are: $src = r1$, $dest = r2$, $orig = r4$, $seq = 1$, and $isFake = true$. $r2$ receives this LSA while considering it to be a valid LSA sent by $r1$. Since the sequence number of the attacker's LSA is larger than that of the LSA instance installed in $r2$'s database, $r2$ installs the attacker's LSA in its database. Since $r2$ received the message from $r1$, it does not flood it back to it. Since $r2$ has no other links no further messages are sent in the topology. Hence, the specification of our model is violated.

Attack #2. The following attack relies on the fact that the routers' queues are bounded. Note that any real-life router must bound its queue size that is dependent on the size of memory space in the router. The attacker continuously sends the following message many times: ($src = r3$, $dest = r4$, $orig = r0$, $seq = 1$, $isFake = true$). The number of sent copies should be larger than the bound on the size of the routers' queues.

The messages are received by $r4$ which floods the first message to $r0$. $r0$ then originates a fight-back message m' with $seq = 2$. Since the queue of $r4$ is full, m' will be discarded leaving $r4$ with the fake message installed in the database. All subsequent fake messages flooded to $r0$ will not trigger fight-back, since their sequence number (1) is smaller than that of the last message originated by $r0$ (m' with $seq = 2$). We note that the OSPF standard makes use of a reliable delivery of messages by leveraging acknowledgment messages. Hence a real router retransmits a message until it receives an acknowledgment. Our model does not include this functionality. Nonetheless, this attack would still be feasible in real life if the attacker continued sending messages to keep $r4$'s queue full.

Attack #3. The following attack was first described in [15]. The attacker sends the following two LSA messages: $m1$ = ($src = r3$, $dest = r4$, $orig = r1$, $seq = 1$, $isFake = true$) and $m2$ = ($src = r3$, $dest = r4$, $orig = r1$, $seq = 2$, $isFake = true$). First, $m1$ is received and installed by $r4$. Then, $r4$ floods it to $r0$. Afterward, $m2$ is received by $r4$. Since it has a higher sequence number than $m1$, $m2$ supersedes it in $r4$'s database. $m2$ is also flooded to $r0$. $r0$ processes and sends both messages to $r1$, while $m2$ is the last to be installed in its database. Once $r1$ receives $m1$ it immediately originates a fight-back message $m3$ with $seq = 2$ and floods it to all its neighbors. $r1$ then receives $m2$. Since $m2$ and $m3$ have equal sequence number (2), $m2$ is not considered newer than $m3$, hence $r1$ does not send another fight-back message and ignores $m2$. Once $r0$ receives $m3$, it does not consider it newer than $m2$ which is currently installed in its database. Hence, it ignores $m3$. Since $r4$ installed the fake message $m2$ and no more messages are waiting to be sent the specification of our model is violated.

3 An Abstract Network and Its Matching Concrete Networks

In the previous section we showed how attacks can be found on concrete models. Due to the state explosion problem, the models that can be handled are very small in size and hence restricted in their topologies. We would like to extend our search for attacks to larger and more complex topologies. Further, we are interested in *general* attacks, which are insensitive to most of the topology's details and therefore can be applied in a family of topologies.

In order to achieve that, we define an *abstract model* which can represent a family of concrete models. The models in the family are similar in some aspects of their topologies but may differ in many other aspects.

The abstract model consists of an abstract topology which includes abstract components representing a large number of routers and sub-networks, and of an abstract protocol which is an adjustment of OSPF to the abstract components.

We define several level of abstract components. The most abstract component is the *sub-topology*, which represents any number of concrete sub-networks. The edges between the sub-topology and the rest of the topology are not abstracted. As a result, routers within the sub-topology which are connected to these edges remain unabstracted as well. These routers are called *singleton routers*. The concrete routers they represent are called *visible*. All other routers within the sub-topology and the edges among them are fully abstracted, and are referred to as *invisible*.

Another abstract component is the *abstract router* which represents a set of concrete routers, all contained within the same sub-network, and have no edges outside of the sub-network. An *abstract sub-network* consists of a set of abstract routers and a set of singleton routers. As with sub-topologies, the singleton routers in a sub-network are un-abstracted. They represent a single concrete router whose edges are un-abstracted too. We require that each singleton router belongs to either a sub-topology or a nonempty set of abstract sub-networks.

The intuition behind the definition of an abstract topology is as follows. The un-abstracted routers are those that may participate in an attack. The others are needed to form a topology that brings unabstracted routers to manifest more of their OSPF functionality and thus to possibly expose more security vulnerabilities. Moreover, abstracted routers allow to show that a found attack is general and applicable to a family of topologies.

Clearly, the attacker is always an (un-abstracted) singleton router. Moreover, the messages sent by the attacker are un-abstracted as well. That is, their originator, source, and destination fields refer to singleton routers.

We impose some constraints on abstract sub-topologies, to guarantee that for every abstract transition and every concrete topology represented by the abstract topology, there can be found a corresponding finite concrete run.

For a sub-topology st, recall that each singleton router in st represents a single concrete visible router. We require that in the part of the concrete topology which is represented by st, each of its visible routers must belong to a different sub-network. Also, visible routers in st may not be directly connected to each other, but should be connected to at least one invisible router. Further, the invisible routers in st form a strongly connected component. These constraints guarantee that if a message is flooded to st by a singleton router r, then there is a concrete run along which the message is opened by all invisible routers prior to being opened by any other singleton router.

While these constraints seem quite restrictive, our abstract topologies still represent a large variety of topologies of different sizes. As shown in Section 5, some nontrivial attacks were found on them. Many of these constraints can be removed for the price of much more complex definitions and correctness proof. We choose to present a simpler version here, and to demonstrate its usability.

3.1 Abstract Topology

Formally, an abstract topology is denoted by $T_A = (SR, ST, AR, SN, E_A, DR_A)$ where, SR is a set of abstract singleton routers, $ST \subseteq 2^{SR}$ is a set of sub-topologies, AR is a set of abstract routers, $SN \subseteq 2^{AR \cup SR}$ is a set of abstract sub-networks, and $E_A \subseteq SR \times SR$ is a set of undirected edges, each representing a point-to-point link between two abstract singleton routers. Finally, $DR_A : SN \to SR$ is a function that maps sub-networks to their designated router, which must be from SR. Figure 2 presents an abstract topology. Note that, similarly to the concrete case, connections between routers within the same sub-network are not depicted in the figure.

3.2 Matching Abstract and Concrete Topologies

Next we define a matching relation between abstract and concrete topologies. The matching relation adhere to the intuitive explanation given above. Let $T_A = (SR, ST, AR, SN, E_A, DR_A)$ be an abstract topology and $T_C = (R, S, E, DR_C)$ be a concrete topology. A relation

$$H \subseteq (SR \times R) \cup \left(AR \times 2^R\right) \cup (SN \times S) \cup \left(ST \times 2^S\right) \cup (E_A \times E).$$

is a *matching relation* between T_A and T_C if it satisfies the following constraints:

- H restricted to each one of its domains is a 1-1 function. For instance, $H \cap (SR \times R)$ is a 1-1 function. By abuse of notation we refer to it as $H : SR \to R$.
- A sub-topology st represents a set of concrete sub-networks S'. Each singleton router in st is matched to a concrete router in a sub-network in S'. Different singleton routers in st are matched to routers in different sub-networks in S'.
- An abstract sub-network sn represents a concrete sub-network s such that each singleton router in sn is matched to a router in s, and each abstract router in sn is matched to a set of routers in s. Every router in s has a matched component in sn.
- Each concrete sub-network is matched to either an abstract sub-network or a sub-topology.
- There is an abstract edge between two singleton routers if and only if there is a concrete edge between their matched routers.

For example, the relation H, given below, is a matching relation between T_A from Figure 2 and T_C from Figure 1.

- $H \cap (SR \times R) = \{(sr1, r8), (sr2, r9), (sr3, r11), (sr4, r18), (sr5, r12), (sr6, r2)\}$
- $H \cap \left(AR \times 2^R\right) = \{(ar1, \{r7, r10\})\}$
- $H \cap (SN \times S) = \{(sn1, s3)\}$
- $H \cap \left(ST \times 2^S\right) = \{(st1, \{s1, s2\}), (st2, \{s4, s5, s6, s7\})\}$
- $H \cap (E_A \times E) = \{((sr1, sr6), (r8, r2)), ((sr4, sr3), (r18, r11)), ((sr2, sr5), (r9, r12))\}$

3.3 Global Abstract States

Let T_A be an abstract topology and let $AC = ST \cup AR \cup SR$ be the set of components in the abstract topology. The message domain in the abstract model is $M = AC \times AC \times ORIGS \times \{0, ..., SB\} \times \{T, F\}$, where $ORIGS \subseteq SR$ is a predefined set of originators which can be used by the attacker in its messages. Abstract messages consist of the same fields as concrete messages.

An abstract state is defined by $\sigma_A = \{ac.DB \,|ac \in AC\} \cup \{ac.Q \,|ac \in AR \cup SR\}$, where for every component $ac \in AC$, the structure of its database is identical to that of a concrete component, $ac.DB : ORIGS \to \{0, ..., SB\} \times \{T, F\}$, except that here it is only defined for the subset $ORIGS \subseteq SR$. In addition, for every $ac \in AR \cup SR$, $ac.Q$ is a queue of up to K messages. The database is restricted to $ORIGS$ since in our setting (see section 2.2) only the attacker originates messages, and those messages have $orig \in ORIGS$. Thus, there is no need for $ac.DB$ to contain entries of other originators.

Note that, we do not define a queue for sub-topologies st, since flooding within st is always described as a single abstract transition. Each singleton router in st has a queue. Thus, a queue for st would have represented the queues of all invisible routers, matched to st. However, the queues of all invisible routers are empty whenever the abstract transition begins or ends. Thus, there is no need to represent their content.

3.4 Matching Abstract and Concrete States

Let T_A and T_C be an abstract and concrete topologies and let H be their matching relation. In order to define a matching between abstract and concrete states, we first define a matching between abstract and concrete databases and queues.

We use h to denote a function that matches abstract databases, messages, queues, and global states to sets of their concrete counterparts.

1. An abstract database DB_A matches a concrete database DB_C, denoted $DB_C \in h(DB_A)$, if for each $o \in ORIGS$, the entry for o in DB_A is identical to the entry of $H(o)$ in DB_C.
2. An abstract message m and a concrete message m' match, denoted $m' \in h\,(m)$, if $m'.src \in H\,(m.src)$, $m'.dest \in H\,(m.dest)$, $m'.orig = H\,(m.orig)$, $m'.seq = m.seq$, and $m'.isFake = m.isFake$.
   Since $orig$ is a singleton router and since seq and $IsFake$ are un-abstracted, they have a single matching.
3. An abstract queue matches a concrete queue if
   (a) For a singleton router sr, each message m in its queue is matched with a sequence of (one or more) concrete messages in $h(m)$.
   The reason for matching more than one concrete message with m is that an abstract transition may add only one message to the queue. On the other hand, the concrete run that correspond to this transition consists of several concrete transitions, each of which may add a matching message to the queue. This is because, when sr is part of a sub-topology st, then the invisible routers represented by st may flood the message several times to sr, via different paths in the sub-topology.
   (b) For an abstract router ar, its queue represents the queues of all concrete routers matched with ar. Here the sizes of the queues are identical since a message received by ar corresponds to single messages received by each r in $H(ar)$ from the designated router. No other messages are sent among routers in $H(ar)$.

We can now define matching of abstract and concrete states. $\sigma_C \in h\,(\sigma_A)$ if the following conditions holds

1. $\forall ac \in AR \cup SR\,[\forall r \in H\,(ac)\,(r.Q \in h\,(ac.Q))]$. That is, queues of matching components must match.
2. $\forall ac \in SR \cup ST \cup AR\,[\forall r \in H\,(ac)\,(r.DB \in h\,(ac.DB))]$. That is, databases of matching components must match.

3.5 Abstract Transitions and Their Matching Concrete Transitions

Similarly to the concrete model, an *abstract transition* between two global abstract states corresponds to an application of the procedure of one of the abstract components. The abstract model includes procedures for a singleton router, an abstract router, and an attacker. Our model does not include a procedure for a sub-topology. Instead, its behavior is defined as part of the procedure of singleton routers included in it.

A high-level description of the procedure of a singleton router sr is given in Figure 5. It is similar to the procedure of a concrete router, except that it does not handle messages whose destination is not sr. This is because in the abstract model such messages are sent by unicast directly to their destination. The singleton router procedure can perform either flooding or fight back. Figure 6 describes the flooding procedure performed by a singleton router (as part of its procedure). $FD_A(sr, m.src)$ returns the flooding destinations, i.e. set of abstract components to which sr floods a message m obtained from component src. The fight back procedure is similar, except that FD_A is replaced by the fight back destinations, FBD_A. The statement $ac_1.Q' = ac_1.Q \centerdot \{m_{sr \to ac_1}\}$ performs an update of ac_1's queue. The resulting queue, $ac_1.Q'$, is obtained by concatenating the old queue $ac_1.Q$ with a message which is identical to m, except that its src is sr and its destination is ac_1.

The procedure of an abstract router is simpler. It only installs a message from its queue in its database and does not perform flood or fight back. This is because it is part of a single abstract sub-network, and is not connected by any edges.

An ac-abstract transition corresponds to a single application of the procedure for abstract component ac. This transition may represent either a single concrete transition or a sequence of concrete transitions (i.e., a concrete run), depending on the type of ac and on the message content. Below we detail a few non-trivial cases where abstract transitions correspond to a concrete run. For every concrete topology T_C represented by an abstract topology T_A and for every abstract transition in T_A, a corresponding concrete run as detailed below can be found in T_C.

Case 1. Consider an abstract transition in which a singleton router sr floods a message m, where sr is within a sub-topology st, and st belongs to the flooding destinations of sr. In such a case, the concrete run represented by the abstract transition includes, in addition to the flooding done by sr, the flooding applied by the invisible routers in $H(st)$. By the end of this run, all invisible routers within st have already removed m from their queue, updated their databases (if their databases were less updated), and flooded m further to the rest of the visible routers in $H(st)$.

Case 2. Consider an abstract transition in which a singleton router sr in a sub-topology st floods a message m, where $m.src = st$. This abstract transition represents a concrete run in which $H(sr)$ floods m. In addition, invisible routers in $H(st)$, which are included in the flooding destinations of $H(sr)$, remove m from their queue and ignore it.

Case 3. Consider an abstract transition in which the attacker sends a message m by unicast to a destination which is not one of its neighbors. That is, the message m is added to the queue of its destination. This abstract transition represents a sequence of concrete transitions in which each router on the routing path which is not the destination, sends the message according to its routing table, without opening the message.

Case 4. Abstract transition taken by an abstract router ar represents a sequence of similar concrete transitions taken by each of the concrete routers represented by ar exactly once.

```
singleton router procedure(sr)
if (sr.Q not empty)
{
  m = pop-head(Q)
    if (m is newer than the copy in sr.DB)
    {
      if (m.orig == sr)
        fight back(sr, m)
      else
        update sr.DB and flood(sr, m)
    }
    else
      ignore m
  }
}
```

Fig. 5. Procedure of a singleton router

```
flood(sr, m)
For each
ac1 ∈ FD_A (sr, m.src) ∩ (AR ∪ SR))
{
  ac1.Q' = ac1.Q . {m_{sr→ac1}}
}
For each st ∈ FD_A (sr, m.src) ∩ ST
{
  if (st.DB[m.orig].seq < m.seq)
  {
    st.DB'[m.orig] = (m.seq, m.isFake)
    For each sr1 ∈ FD_A (st, sr)
      sr1.Q' = sr1.Q . {m_{st→sr1}}
  }
}
```

Fig. 6. flooding procedure of a singleton router sr, where m is the message to flood

4 Correctness of the Algorithm

Theorem 1. *Let T_A and T_C be an abstract and concrete topologies and let H be their matching relation. Then, for each finite abstract run $\sigma_1, \ldots \sigma_n$, there exists a corresponding finite concrete run $\sigma'_1, \ldots \sigma'_k$, such that $\sigma'_1 \in h(\sigma_1)$ and $\sigma'_k \in h(\sigma_n)$.*

Corollary 1. *An abstract attack found on an abstract topology T_A, has a corresponding attack on each matching topology T_C .*

Proof Sketch

- We show that for each abstract transition, there is a concrete finite run, such that the initial and final states of the transition and of the run are matching.
- An abstract attack is an abstract run for which the final state violates our specification. A concrete state matching an abstract state which violates the specification, also violates the specification. Thus, the corresponding paths are concrete attacks.
- The proof is based on the matching relation H and on the function h, defined in section 3.

5 Examples of Attacks on OSPF in the Abstract Model

In this section we describe a few attacks, found on different abstract models which we picked manually.

Attack #1. This attack has been found on the abstract topology T_A, presented in Figure 2. The attacker is $sr2$. The set of predefined originators is $ORIGS = \{sr1\}$. The attacker originates a fake message on behalf of $sr1$: $m = (src = sr2, dest = sr5, orig = sr1, seq = 1, isFake = T)$. $sr5$ receives this message while considering it to be a valid message, sent by $sr2$. Since the sequence number of m is larger than that of the message instance installed in $sr5$'s database, $sr5$ installs m in its database, and floods it. The fake message will be flooded and installed in the databases of $st2$, $sr4$, and $sr3$. When m is installed by sr3, it will be flooded to the attacker $sr2$, since $sr2$ is the designated router of the sub-network $sn1$. The attacker will choose to ignore m, thus preventing this message from being flooded to $sr1$, and avoiding fight back. Since no more messages are waiting to be sent, the specification is violated.

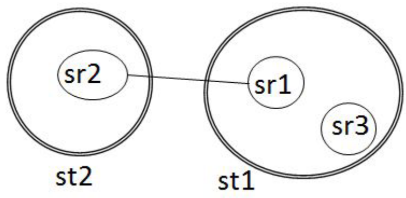

Fig. 7. Abstract topology on which attack #2 is described

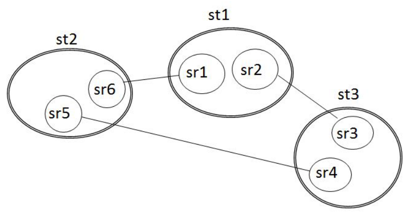

Fig. 8. Abstract topology on which attack #3 is described

Attack #2. T_A is the abstract topology presented in Figure 7. The attacker is $sr3$. The set of predefined originators is $ORIGS = \{sr1\}$. The attacker originates a fake message on behalf of $sr1$: $m = (src = sr1, dest = sr2, orig = sr1, seq = 1, isFake = T)$, which is sent by unicast to $sr2$. $sr2$ installs the fake message in its database and floods it only to the sub-topology $st2$ due to the flooding rules of OSPF. Therefore, in the final state the queues of all abstract components are empty, and the databases of $sr2$ and $st2$ are installed with the fake message. Thus, the specification is violated.

Attack #3. T_A is the abstract topology presented in Figure 8. The attacker is $sr3$. The set of predefined originators is $ORIGS = \{sr2\}$. The attacker sends the following two LSAs (using unicast sending): $m1 = (src = sr3, dest = sr2, orig = sr2, seq = 1, isFake = T)$ and $m2 = (src = sr4, dest = sr5, orig = sr2, seq = 2, isFake = T)$. As a result, $sr2$ sends a fight back message $m3$ with $orig = sr2, seq = 2, isFake = F$, but $sr5$ opens $m3$ after it has already installed $m2$ in its database, and will thus ignore the fight back message and will remain with the fake message.

6 Directions for Future Research

An important direction for future research is to generalize the method for finding general attacks applicable to families of network topologies to other network protocols, in particular routing protocols. Another direction is to develop a methodology for deciding which abstract networks to check, and to automate the abstraction process. Additional direction is to extend the abstraction mechanism for finding attacks which are applicable to a sub-family rather than the whole family, to enable finding more possible attacks.

Acknowledgement. We Thank Manfred Grumberg for initiating the project. This research was conducted as part of the KABARNIT consortium, with the support of the MAGNET founds.

References

1. Abdulla, P.: Regular model checking. STTT 14(2) (2012)
2. Clarke, E., Kroning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004)
3. Niklas Een, N.S.: Minsat 2.0 - (2008), `http://minisat.se/minisat.html`
4. Emerson, E.A., Kahlon, V.: Exact and efficient verification of parameterized cache coherence protocols. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 247–262. Springer, Heidelberg (2003)
5. Fortz, B.: On the evaluation of the reliability of OSPF routing in IP networks. Technical report, Institut dadministration et de gestion (2001)
6. German, S., Sistla, P.: Reasoning about systems with many processes. J. ACM 39(3) (1992)
7. Jones, E., Le Moigne, O.: OSPF security vulnerabilities analysis. Internet-Draft draft-ietf-rpsec-ospf-vuln-02, IETF (June 2006)
8. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich ssertional languages. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 424–435. Springer, Heidelberg (1997)
9. Liu, J., Ye, X., Zhang, J., Li, J.: Security verification of 802.11i 4-way handshake protocol. In: Communications (2008)
10. Malik, S.U.R., Srinivasan, S.K., Khan, S.U., Wang, L.: A methodology for OSPF routing protocol verification. In: 12th International Conference on Scalable Computing and Communications (ScalCom) (2012)
11. Matousek, P., Ráb, J., Rysavy, O., Svéda, M.: A formal model for network-wide security analysis. In: Engineering of Computer Based Systems (2008)
12. John, C.: Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murphi. In: IEEE Symposium on Security and Privacy, pp. 141–151 (1997)
13. Mitchell, J.C., Roy, A., Rowe, P., Scedrov, A.: Analysis of EAP-GPSK Authentication Protocol. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 309–327. Springer, Heidelberg (2008)
14. Moy, J.: OSPF version 2. IETF RFC 2328 (April 1998)
15. Nakibly, G., Gonikman, D., Kirshon, A., Boneh, D.: Persistent OSPF attacks. In: NDSS (2012)
16. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of ad hoc routing protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 18–32. Springer, Heidelberg (2008)
17. Wang, F., Vetter, B., Wu, S.F.: Secure routing protocols: Theory and practice. Technical report, North Carolina State University (May 1997)
18. Wu, S.F., et al.: JiNao: Design and implementation of a scalable intrusion detection system for the OSPF routing protocol. ACM Transactions on Computer Systems 2 (1999)