Formal analysis for robust anti-SPIT protection using model checking

Dimitris **Gritzalis**[1], Panagiotis **Katsaros**[2], Stylianos **Basagiannis**[2], Yannis **Soupionis**[1]

[1] Information Security & Critical Infrastructure Protection Research Laboratory
Dept. of Informatics, Athens University of Economics and Business (AUEB)
76 Patission Ave., Athens, GR-10434, Greece
{dgrit, jsoup}@aueb.gr

[2] Dependability & Security Group
Dept. of Informatics, Aristotle University of Thessaloniki (AUTh)
Thessaloniki, GR-54124, Greece
{katsaros, basags}@csd.auth.gr

Abstract

Anti-SPIT policies counter the SPam over Internet Telephony (SPIT) by distinguishing bots launching unsolicited bulks of VoIP calls from human beings. We propose an Anti-SPIT Policy Management mechanism (aSPM) that detects spam calls and prevents VoIP session establishment by the Session Initiation Protocol (SIP). The SPIN model checker is used to formally model and analyze the robustness of the aSPM mechanism in execution scenarios with parallel SIP sessions. In case of a possible design flaw, the model checker provides a trace of the caught unexpected behavior (counterexample), which can be used for the revision of the mechanism's design. Our SPIN model is parameterized, based on measurements from experiments with VoIP users. Non-determinism plays a key role in representing all possible anti-SPIT policy decisions, in terms of the SIP messages that may be exchanged. The model checking results provide evidence for the timeliness of the parallel SIP sessions, the absence of deadlocks or livelocks, and the fairness for the VoIP service users. These findings ensure robust anti-SPIT protection, meaning that the aSPM mechanism operates as expected, despite the occurrence of random SPIT calls and communication error messages. To the best of our knowledge, this is the first analysis for exhaustively searching security policy flaws, due to complex interactions between anti-SPIT measures and the SIP protocol services.

Keywords: *Voice over IP (VoIP), anti-SPIT security policies; robustness analysis; model checking.*

1 Introduction

Spam over Internet Telephony (SPIT) [1-3] refers to all unsolicited and massive scale attempts to establish voice communication with oblivious users of Voice over Internet Protocol (VoIP) services. SPIT is a spreading service abuse phenomenon that comes in three different forms: (a) call SPIT, (b) instant message SPIT, and (c) presence SPIT. The underlying technology that drives the SPIT threat is the Session Initiation Protocol (SIP) [7]. SIP's widespread use shows signs of becoming a de facto industry standard for several forms of interactive communication (voice, video, etc.), despite the findings [8-10], which demonstrate that it is vulnerable to the SPIT threat.

Major IT players and VoIP service providers have already recognized the key value of timely delivering mechanisms that counter SPIT [4-5] effectively. They adopt basic principles and design considerations from policy-based protection measures introduced by the IT security industry.

Policy-based mechanisms work on the basis of predetermined conditions [6], which control the execution of appropriate actions for service protection. Such a structured set of rules defines a security policy. By separating the policy from the VoIP infrastructure we introduce the possibility of potential policy modifications, in order to dynamically change the strategy for managing the system and, hence, to modify the system behavior without changing its underlying implementation. Moreover, such a policy is easily integrated into a VoIP infrastructure and can run along with other security mechanisms of a network domain.

A robust anti-SPIT mechanism should avoid side-effects to the capability of the protected VoIP service to operate as expected. Model checking is an attractive choice, because it offers the advantages of an exhaustive verification technique, where the analyst can use non-determinism to model all possible system behaviors that arise from the interaction of anti-SPIT measures with the VoIP services.

We present the design of an Anti-SPIT Policy Management (aSPM) mechanism and an abstracted finite model of its behavior in the SPIN model checker. The mechanism has emerged from an extensive study of the SIP protocol for SPIT-related vulnerabilities/threats and accommodates policies that handle, counter and mitigate potential SPIT attacks. The three aforementioned aims of aSPM [11] are addressed by intervening to the normal SIP message exchanges, in order to check their content and detect potentially spam calls. This unavoidably introduces delays in the processing of SIP protocol messages.

The SPIN model is parameterized with respect to these delays based on results from experiments with VoIP users. It is then used for studying the robustness of our anti-SPIT mechanism by model checking parallel SIP sessions under the presence of random SPIT calls and SIP protocol error messages.

In [30], we have presented initial steps for verifying basic correctness properties of our aSPM mechanism. Herein, we introduce a comprehensive analysis of aSPM robustness, which can be used in formally verifying the functionality of any policy-based anti-spam countermeasure. Our SPIN model now covers the SIP message exchanges in detail, while the system is still studied when serving parallel SIP sessions. This is achieved due to an improved model representation, which avoids the state space explosion, and at the same time allows verifying properties that span to a wider range of robustness characteristics. We provide results for upper bounds in the duration of call session establishment under the aSPM mechanism, which effectively ground the timeliness of SIP sessions.

We also verify the absence of deadlocks and livelocks, as well as the fairness for the service users [31]. Absence of deadlock means that there is no reachable state in which the SIP message exchange terminates with the protocol participants waiting for some message from the other participants. A livelock involves a set of states in which protocol execution may be trapped with no chance to show progress. Expected properties are expressed either as assertions over the reachable states or temporal logic formulae that are algorithmically validated across all possible execution paths. Analysis of additional robustness concerns is also possible, if all related countermeasure behavior has been modeled.

The remainder of this paper is organized as follows. Related work is presented in Section 2, regarding anti-SPIT security policy studies and model checking. In Section 3, we describe the methodology followed for formally analyzing robustness of anti-SPIT measures in the Session Initiation Protocol (SIP). Section 4 presents the aSPM anti-SPIT policy and offers results obtained from a real-time SIP session experiment, in order to formulate valid modeling assumptions and property specifications. Section 5 introduces the `SIP-aSPM` model developed in the SPIN model checking tool [6]. In Section 6 we provide the verification results for the mentioned properties. Finally, the paper concludes with a review of the outcomes and a discussion on future work insights.

2 Related work

Robustness of real-time communication systems refers to the error tolerance of the protocol that provides the communication session establishment. If we take into account security mechanisms like anti-SPIT countermeasures and policies that are expected to intervene in the protocol's services, certain safety and performance guarantees [3, 6] should be preserved. To this end, formal verification techniques such as model checking offer an attractive approach for providing results, which characterize the behavior of these multiple-parameter based communication systems.

2.1 Security policy background in VoIP

Confronting SPIT requires procedures for both detecting and handling SPIT attacks. Due to the real-time nature of VoIP communications, the identification and handling procedures are applied during the initiation handshake of VoIP calls. Thus, anti-spamming techniques like the content analysis with Bayesian filters and other approaches based on voice-communication pattern recognition [13] are excluded. Our proposal, in order to effectively detect and handle SPIT, is an adaptive policy-based SPIT management framework.

In the design process of the aSPM mechanism, we reviewed a series of policy-based systems from diverse domains, such as storage area networks [14], networked systems [15], database management [16] etc. The policies used in these systems can be categorized into two broad categories [17], namely:

1. *Authorization policies* defining access rights for some subject, which may be a role, a domain, or a management agent. Such access control rules may be applied to various context domains and mechanisms, like database access controls and packet filters.
2. *Obligation policies* defining permit (positive) or forbid (negative) actions on a target object. More specifically, obligation policies specify the actions that must be performed within the system, when certain events occur. They also indicate who must execute those actions, i.e., what kind of activities a subject (human or automated manager components) can perform on objects in the target domain. In the network security context, obligation policies are used to specify the functionality of protection mechanisms, such as intrusion detection systems (IDS) [18].

The anti-SPIT policies enforced by the aSPM mechanism are obligation policies that trigger appropriate countermeasures whenever a SPIT call/message is detected. Policy rules define which behavior will not be accepted in a VoIP system, without specifying the actions and the event sequences that actually "produce" the undesired behavior. This declarative approach opens a possibility for relatively easy integration of the aSPM mechanism with existing drastic anti-SPIT countermeasures. At the same time, the data collected from the interconnected anti-SPIT mechanisms can be used for generating appropriate policy rules. The aSPM mechanism also supports the integration of reputation listing measures.

2.2 Model checking background

Model checking has been effective in discovering design flaws in systems and protocols, while it has been particularly successful in the formal analysis of security and communication protocols [21, 23, 35]. The advent of mature model checking tools like SPIN [19-20, 22] brings this "push-button" verification perspective closer to the current system design practice, without underestimating the still outstanding problem of state space explosion. Model checking is defined as follows: given a model M and a temporal logic formula φ determine if $M \models \varphi$, i.e. does M satisfy φ? Model checking communication protocols like the SIP is based on a finite state model that represents at a suitable abstraction level the behavior of a system,

where the protocol runs in one or more concurrent sessions. Operational errors or security flaws are detected in the form of safety or liveness property violations that reflect unexpected behavior. If a property φ is violated, the model checking result is false and the state exploration algorithm returns a counterexample in the form of an execution path in M leading to the violation of φ. One can then use the counterexample to debug the system model and ultimately find a repair for the detected protocol flaw.

In the bibliography there are a few model checking studies of VoIP communication and multimedia protocols like the SIP [7] and H.323 [24]. In [25], Zave presents three different SPIN models for the SIP that revealed five cases of incomplete specification in the protocol standard. She proposed solutions for the detected problems, but her work is not directly related to the aSPM mechanism, since it basically concerns SIP messages that are not involved in SPIT (e.g. the update message for re-negotiation session establishment). In [26], Liu introduces a Colored Petri Net model for SIP-INVITE transactions over an unreliable medium. The performed state space analysis revealed that SIP-INVITE transactions are not free of livelocks and dead codes, as they are when assuming a reliable communication medium. As the messages involved in that study are not adequate for generating SPIT, they are not the ones handled by the aSPM policies; therefore, the results reported are not applicable to our problem.

From the related bibliography on policy verification, we note the Schaeffer-Filho et al. [27], which focuses on the verification of policy interactions for a system of Self-Managed Cells (SMCs). Their model allowed checking the correctness of the anticipated SMC interactions, in the presence of policies, but before deploying them in physical devices (e.g. PDA, mobile phones, sensors, etc.).

3 Anti-SPIT robustness analysis

The IEEE Standard Glossary [38] defines *robustness* as "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions". A system should be robust enough to handle invalid inputs and all possible abnormal situations that can occur in its operating environment.

SIP specifications include optional requirements, as well as partial or ambiguous definitions for how to handle communication errors. In the context of anti-SPIT protection, a robust mechanism should avoid side-effects to the capability of the SIP protocol to operate as expected, even in the presence of random SPIT calls and messages generated from communication errors. Recent proposals on robustness testing [39-42] can be effective into finding bugs in system implementations, but they cannot easily uncover design flaws attributed to complex interactions like the ones of anti-SPIT measures with the SIP protocol services.

We adopt model checking as a mean to formally analyze the robustness of our anti-SPIT protection mechanism, called aSPM. Fig. 1 provides an outline of the aSPM design methodology that develops along the shown sequence of steps, from the leftmost to the rightmost one:

1. In step 1, we have a detailed design of the processes that comprise an anti-SPIT policy, i.e. the SPIT detection and reaction processes. This was accomplished by identifying the SIP-related SPIT vulnerabilities with a thorough analysis of the SIP RFC [9].
2. In step 2, we propose the development of attack scenarios that exploit the previously identified vulnerabilities.
3. The scenarios are then used to define the policy rules and more precisely the conditions upon which the policy actions depend.

4. An XML schema is created in step 4, to link the policy conditions with the employed countermeasures (actions). The resulting policy definition is then integrated into our anti-SPIT Policy management Framework (ASPF), which provides the underlying SIP-based VoIP execution environment.

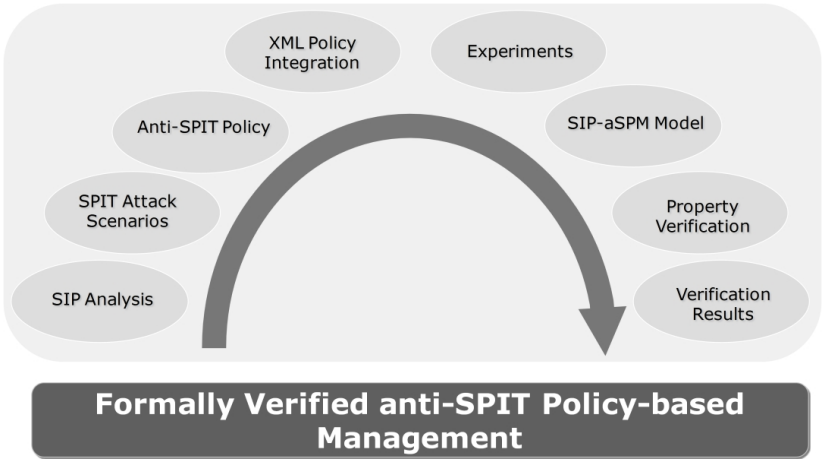

Figure 1: Design of robust anti-SPIT protection over SIP

5. In step 5, a series of experiments with VoIP users takes place, where the ASPF under the control of one or more XML policy schemas generates necessary data for the parameterization of a finite aSPM model in the SPIN model checker.
6. A model of the aSPM mechanism is then developed in SPIN. The model reflects the SIP message exchanges for call session establishment, the behavior of ASPF entities that enforce the anti-SPIT policy and the message overload, due to the policy reaction process. Model parameters for the cost in time of the exchanged messages come from the call session establishment experiments of the previous step.
7. In step 7, we model check the properties need to ascertain aSPM robustness. First, the used anti-SPIT policy should not create 'invalid' states - like deadlocks - and livelocks in the SIP communication process, even upon possible communication errors. Second, we verify that aSPM does not violate timeliness of session establishment, as well as fairness properties with respect to the VoIP service users.
8. The verification results either validate the robustness of aSPM or else yield a counterexample for a robustness property that is not satisfied. In the latter case, the designer studies the counterexample that manifests the result of some design flaw, possibly a policy definition error or an error due to the interactions of the anti-SPIT measures with the SIP protocol services.

4 Security Policy

In this section, we describe two different policy examples, which cover specific SPIT scenarios. The examples illustrate the way that the policy rules have been created and settled to the final rule set. The first example indicates the exploitation of a vulnerability that considers the restrictions that SIP poses in registering an address. The second example illustrates the case of a deficient definition for how a header field can be used and the way that the definition can be exploited in a SPIT attack. The creation of the main policy elements and its representation in a formal XML schema is presented. Finally, the way the policy is enforced in a SIP infrastructure is illustrated.

4.1 Registration of multiple addresses

In the SPIT attack of Fig. 2, a malicious user exploits a vulnerability attributed to an unconstrained change of the registrar server records, which is possible in SIP. Suppose the SIP address `anne@one.com` that a malicious user has got from a legitimate one and that he can register with her address. Such a malicious user can:

- Obtain multiple SIP addresses by sending plain requests to the registration or location servers.
- Change the obtained address to a new one, say `alice@one.com`, which depicts to a recorded SPIT message. In this way, whenever a caller tries to communicate with Anne, his call is redirected and a SPIT attack is thus realized. Such an attack continues until all the friend-SIP-users of Anne have been informed that the Alice address that they use is not valid.

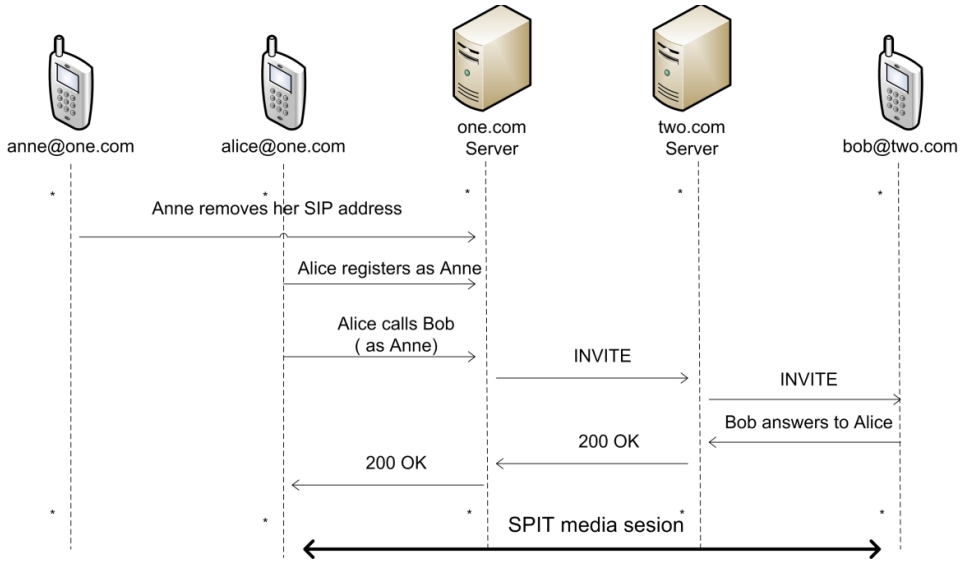

Figure 2: Alteration of a Registrar server's legitimate record

This vulnerability cannot be exploited if the policy clearly states how the domain proxy should treat recurrent requests that aim to new registrations or alteration of existing addresses. This is reflected in rules that define the behavior of the registrar, when it receives registration requests. Such rules define actions like rejecting or accepting to serve the request and answering with an authentication challenge.

4.2 Contact field in message 300

In the second scenario (Fig. 3), the SPIT attack is established using the Contact field. When a request is related to many possible SIP addresses, a SIP message with the code `300` (Multiple Choices) is returned as answer by the domain server. This message contains a Contact field with all the alternative destination addresses. The SIP protocol states that the `Caller` should use one of the addresses included in the Contact field, in order to try a new call.

Thereafter, a spitter is able to automatically redirect the call to a different address from the one chosen by the `Caller` during the call negotiation phase. This intervention could take place in any state of the negotiation and the spitter has just to reply to the original `INVITE` message with a `300` message. A method to prevent SPIT attacks via the `300` message is a policy rule, which states clearly the way such

messages should be handled. The advantage of this approach is that the policy covers all the negotiation process, since the rule is not triggered to a specific point of the communication but is only triggered whenever a new `300` SIP message arrives.

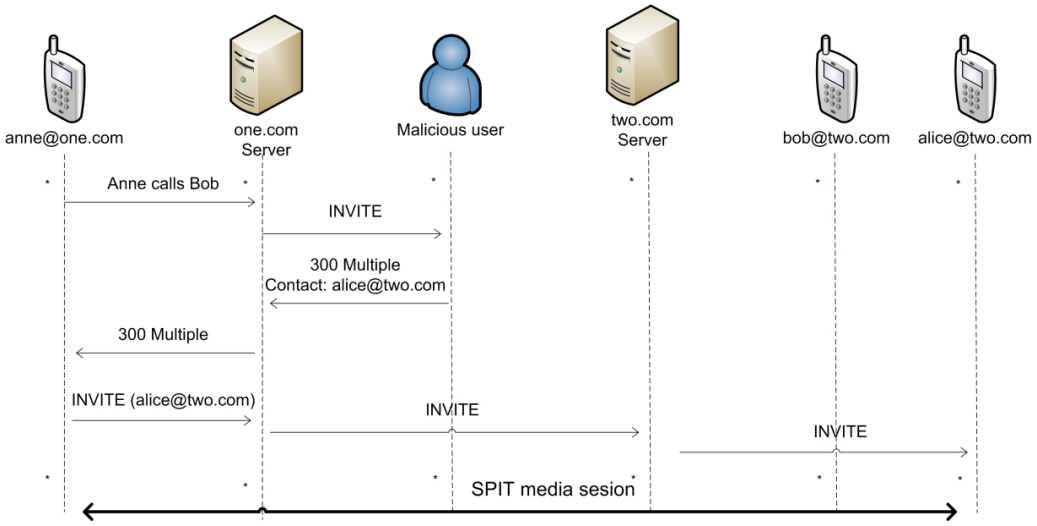

Figure 3: Exploitation scenario of the 300 message Contact field

4.3 Policy approach

We present now how the processed attack scenarios are transformed into SPIT-oriented attack patterns and how this leads to the development of appropriate policy rules.

4.3.1 Policy rules

The policy rule is the key element of a policy. It consists of a Condition, which is a pattern of an identified SPIT scenario extracted from an attack graph, and an Action, which is the reaction of the `Callee`'s domain to the SPIT attack.

The condition is expressed by identifying specific attributes and characteristics of an attack scenario. In order for a condition to be met, we should create precise sub-conditions for each attribute.

Let us focus on the scenario of section 4.2. The attributes that can be handled are: (a) the message code `300`; (b) the multiple contact fields. Each attribute produces a sub-condition and the logical aggregation of these sub-conditions results in the following policy condition:

$$\texttt{Condition} = [\texttt{Code} = 300 \oplus \texttt{Contact} \asymp \texttt{Multiple}]$$

More generally, a condition is defined as:

$$\texttt{Condition} = f(c_1, c_2, \ldots, c_k) = c_1 \bullet c_2 \bullet \ldots \bullet c_k$$

where c_i is a sub-condition and $\bullet$ denotes the logical operator that can be either $\oplus$ (AND) or $\otimes$ (OR). The operators used in sub-conditions are: (1) = equal, (2) $\asymp$ times of header appearance (Multiple, One, None), (3) $\approx$ approximately equal[1], (4) > greater, and (5) < less.

[1] The *approximately equal* operator means that if a header field A has multiple values, then it is approximately equal to header field B, when one of its values is equal to the value of the other field.

The other half of a policy rule is the policy action. The absence of context in which an attack scenario may take place is critical for deciding the reaction of an entity that averts the attack. Different contexts mean different security needs and different security needs translate to different countermeasures. Having them in mind, it is a fact that the countermeasures/actions that can deal with the conditions cannot be strict, as this could be considered as an ASPF limitation. This is why the actions are divided into two categories and a set of suggested actions has been created for each condition that can occur. Moreover, the ASPF can adapt to any future recommended actions and integrate them in the proposed set of countermeasures. The two policy action categories are:

a. *Block.* This countermeasure results in rejecting the SIP message upon which is triggered. The message sent to the `Caller` is "403-Forbidden". Moreover, the action can possibly contain a more informative description for the cause of rejection, in order to help the requestor entity into altering the message such that it meets the necessary requirements of the `Callee` or her/his domain. A typical SIP message could be "488-Not Acceptable Here" with the description "*Inappropriate word in Subject*".

b. *Notify.* This countermeasure is applied, when we are not confident whether the message upon which is triggered is or is not a SPIT. In this case, the message is not rejected, but it is forwarded to where the administrator/user specifies. The requestor entity receives the SIP 183 *"Session in Progress"* message and is thus notified that his/her message has been redirected to a decision-support application. It is possible that the message is forwarded to the `Callee`, after having been altered such that it complies with the policy rules. In the latter case the administrator receives the notification in a log file.

An example of a set of actions for the discussed condition

$$\texttt{Condition = [Code = 300} \oplus \texttt{Contact} \asymp \texttt{Multiple]}$$

appears below:

1. The User Agent Client (`UAC`) uses the previous address of the From field to compose upcoming messages.
2. The `UAC`'s address book is updated with the new addresses.
3. The `UAC` informs the user for the new SIP addresses.
4. The `UAC` rejects the call and returns a Message 403 *"Forbidden"*.
5. The `UAC` rejects the call and returns a 488 "*Not Acceptable Here*" message with description "*Multiple Contact fields"*.
6. The `UAC` forwards the SIP message to another entity and returns a message 183 "*Session in Progress*".

Most of the actions are SIP messages [33-34], because the policy should: (a) be transparent to the administrators and users and (b) keep to a minimum the intervention of other applications during message handling. The final decision for the action to be applied is left to the administrator or to anyone who is responsible for the production of a policy instance, in order to secure a particular VoIP communication.

4.3.2 XML schema - Policy integration

Based on the identified conditions, an XML schema was created. That schema assisted the integration of aSPM into VoIP environments. The reader may obtain the developed aSPM schema in [31]. The XML schema representation includes the identification characteristics of the attack attributes, together with

their relation. Moreover, the XML schema is easily extendable and therefore the domain administrators could add new rules effortlessly.

The main element of the XML policy structure is the *RuleItem*. A *RuleItem* consists of two elements, the Subject, on which the condition is applied (`Caller`, `Callee`, `Caller`'s domain) and the Rule, which records a certain condition and introduces the proper action, when this condition is met. The full version of the XML schema can be retrieved from [31]. After the XML schema creation, the next step is to integrate the XML policy into a SIP infrastructure.

The implementation approach is depicted in Fig. 4 and includes two additional modules:

1. The XML parser which reads the XML policy instance into the memory and provides easy access to tag values of the document.
2. The policy enforcement module, which has as input the parsed XML document, together with the message attributes. This module checks all the policy conditions, so as to find out which are fulfilled (first a SIP message is received and parsed, and then the message attributes are checked against the policy element) and if one or more conditions are met, then the associated action (described in the fulfilled policy element) takes place.

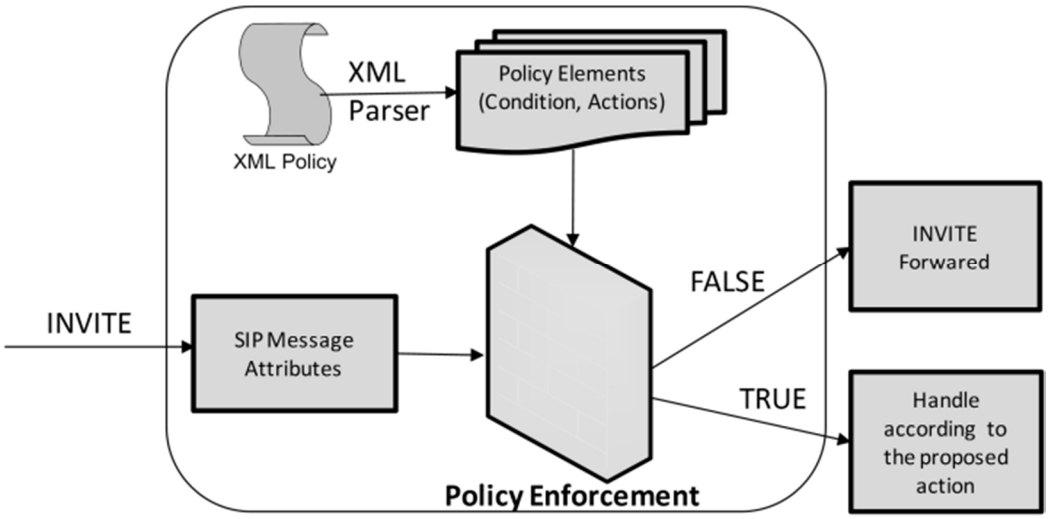

Figure 4: The aSPM integration

4.3.3 Experiments

A requirement that should be validated is the timely session establishment. In order to evaluate this, the impact of policy enforcement in terms of time cost should be evaluated. Therefore, we created a laboratory SIP infrastructure, as shown in Figure 5 which was consisted of: a) two SIP servers implemented by the well-known SER server [28], b) the aSPM policy module, which includes the XML policy document of the protected SIP domain, and c) a client, which was implemented by SIPp [29] and was able to initiate and receive calls.

In order to record the real overload of aSPM, we should first record the SIP infrastructure performance without ASPF and then record the time period for session establishment with the use of aSPM (our policy module accepts all the incoming and outgoing SIP traffic).

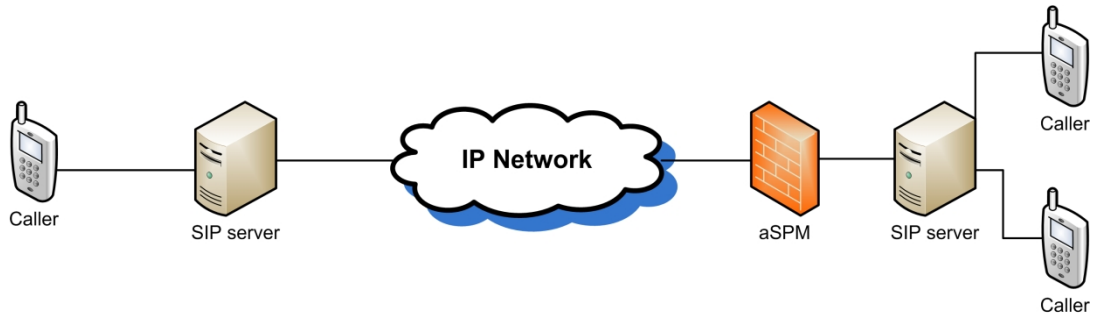

Figure 5: Laboratory Environment

The total number of initiated calls was 10.000. The time needed for the message exchanges between the participating entities is illustrated in Table1. The participating entities are the two domains (D), the User Agent Client (`UAC`), and the policy module (aSPM). The first two columns show the time costs when the aSPM is disabled, while the last two, when the aSPM is enabled. The third column shows the time needed for the `Caller`'s `UAC` to send the final `ACK` message, where the policy does not interfere.

The main assumption is that the time needed to send a message between two communication entities does not depend on the role of the involved entities in the communication, i.e. whether they are the sending or the receiving entity. For example the same time is needed to send a message from Domain1 to Domain2 with the time needed to send a message from Domain 2 to Domain 1.

The times shown in Table 1 have been used as parameters in the formal model, described in the next session, in order to assess how the message exchange affects the requirement of timely session establishment. The reported times may be altered depending on the hardware features that are assumed in the described experiment. However, the proposed verification approach is still effective for other configuration parameter sets.

Table 1: Time needed for exchanging messages

Channel / Time (msec)	UAC – D	D –D	UAC –UAC	UAC – aSPM	D- aSPM
Minimum	97	153	97	286	321
Average	102	154	98	387	428
Maximum	106	162	98	432	642

5 Model checking the SIP based anti-SPIT protection mechanism

Here we describe the steps followed for implementing in SPIN [22] the interactions between the SIP and the aSPM mechanism using the tool's model specification language called PROMELA. The reader may obtain the `SIP-aSPM` model from [32]. This model serves as a formal specification for investigating all possible user-agent behaviors in scenarios of parallel SIP sessions, with respect to a number of correctness properties that guarantee robust anti-SPIT protection.

Fig. 6 provides a schematic representation of our robustness analysis approach. The involved SIP entities, i.e. the Initiator (`Caller`) and the Responder(s) (`Callee`), interact with each other for establishing one or more VoIP communications within a finite time period. The whole analysis encompasses the following steps:

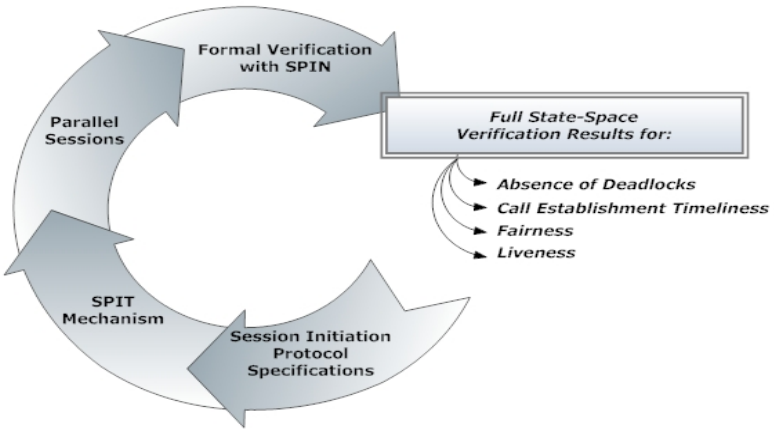

Figure 6: Schematic representation of the SIP-aSPM robustness analysis steps

1. A SPIN model for a single SIP protocol session has been developed, according to the message exchanges prescribed by the SIP specification [7]. While the SIP entities may select among a range of appropriate messages for each protocol step, our model provides an abstract representation of the protocol's operation. This was implemented by having omitted all message manipulation functions with heavy impact in the model's state space that clearly do not influence the intended aSPM analysis.
2. Data for the message latencies and message processing times has been collected, from the measurements of the experiment of section 4. These values are used in place of parameters of executable actions that reflect the entities behavior in a protocol session.
3. A second `Callee` protocol entity has been created that participates in a parallel SIP session with the `Caller`.
4. The aSPM policy has been then integrated into the model, with a focus on the times that the policy needs to process calls. The policy effects on the flow of SIP message exchanges are represented by non-deterministic selection between the messages that the policy can generate in each protocol step.
5. SPIN's state exploration functions have been employed to model check reachability of deadlock, states that violate SIP's functional properties (therefore called invalid states), as well as a Linear Temporal Logic (LTL) [21] formula encoding a requirement for timely session establishment. Robustness analysis also includes fairness guarantees for the service users.

5.1 SIP-aSPM messages

State space explosion was avoided by having omitted the SIP messages that do not affect the SIP execution outcome or the aSPM policy conditions [24]. This modeling approach played a key role in formally validating the robustness of our anti-SPIT mechanism in a scenario with parallel protocol sessions.

More specifically, we ignored the `5xx` messages that concern with server failures and the `1xx` messages, which are used for provisional response(s). The `5xx` messages indicate permanent failures, where the `Caller` should not try to resend the initial request and it is therefore reasonable to consider that they do not affect the policy's implications on the SIP message exchange for session establishment. The `1xx` messages are used exclusively for reporting that a SIP server performs some additional actions, without having concluded to a definite response for a previous message.

In Table 2, we present the communicating entities involved into the modeled SIP-aSPM execution scenarios. For easy reference to the roles of the involved entities, we consider that the Caller belongs to Domain1, while the two Callees named UAC_1 and UAC_2 belong to Domain2.

Table 2: Communication entities that initiate and handle SIP messages

Communication entity	SIP Messages	
	Initiate	Handle
Caller (UAC)	INVITE_1, INVITE_2, REGISTER,OPTION, ACK	2xx, 3xx, 4xx, 6xx
Callee (UAC_1)	2xx (m200_OK)	INVITE_1, INVITE_2, ACK
Callee (UAC_2)	2xx (m200_OK)	INVITE_1, INVITE_2, ACK
Domain1 (Caller's)	-------	REGISTER
Domain2 (Callees')	3xx, 4xx, 6xx	INVITE_1, INVITE_2
aSPM	4xx, 6xx	INVITE_1, INVITE_2, 2xx, 3xx, 4xx, 6xx

In more detail, the messages that the protocol entities initiate or handle (i.e. answer or reject) in the SIP protocol steps are:

1. REGISTER messages created by the Caller
2. OPTION messages created by the Caller
3. INVITE messages that are created by the Caller
4. 2xx successful response message created by the Callee
5. 3xx redirection response messages created by the Domain2 server
6. 4xx request failures messages created by the Domain2 server
7. 6xx global failures messages created by the Domain2 server
8. ACK messages created by the Caller

5.2 Model structure and implementation

The SIP-aSPM model in SPIN comprises six interacting processes, specified as PROMELA proctypes: (i) proctype UAC initiates one or two protocol sessions, (ii) proctypes UAC_1 and UAC_2 that communicate with UAC represent the two Callees, (iii) proctype Domain1 represents the server of the Caller, (iv) proctype Domain2 represents the server of the two Callees and (v) proctype Timer is a stopwatch timer for measuring call establishment time with or without the aSPM policy. Our model also includes rendezvous communication channels, for synchronous message passing between the mentioned processes. The measured message latencies that are shown in Table 1 are encoded as discrete time values attached to message delivery actions. Every message exchange causes a global timer update that is implemented in the Timer proctype.

Time increments occurring along the execution paths that are explored take place according to the measured latencies shown in section 4. For example, a message exchange between UAC and UAC_1 or UAC_2 brings on a time_3 = 98 msec increase in the timer. By tagging the modeled message exchanges with the measured time values, we can estimate the cost in time for call establishment along all possible execution paths that in essence enact the aSPM policy effects. It is thus possible to model check an anticipated upper time bound for call establishment, in model configurations with or without parallel SIP sessions.

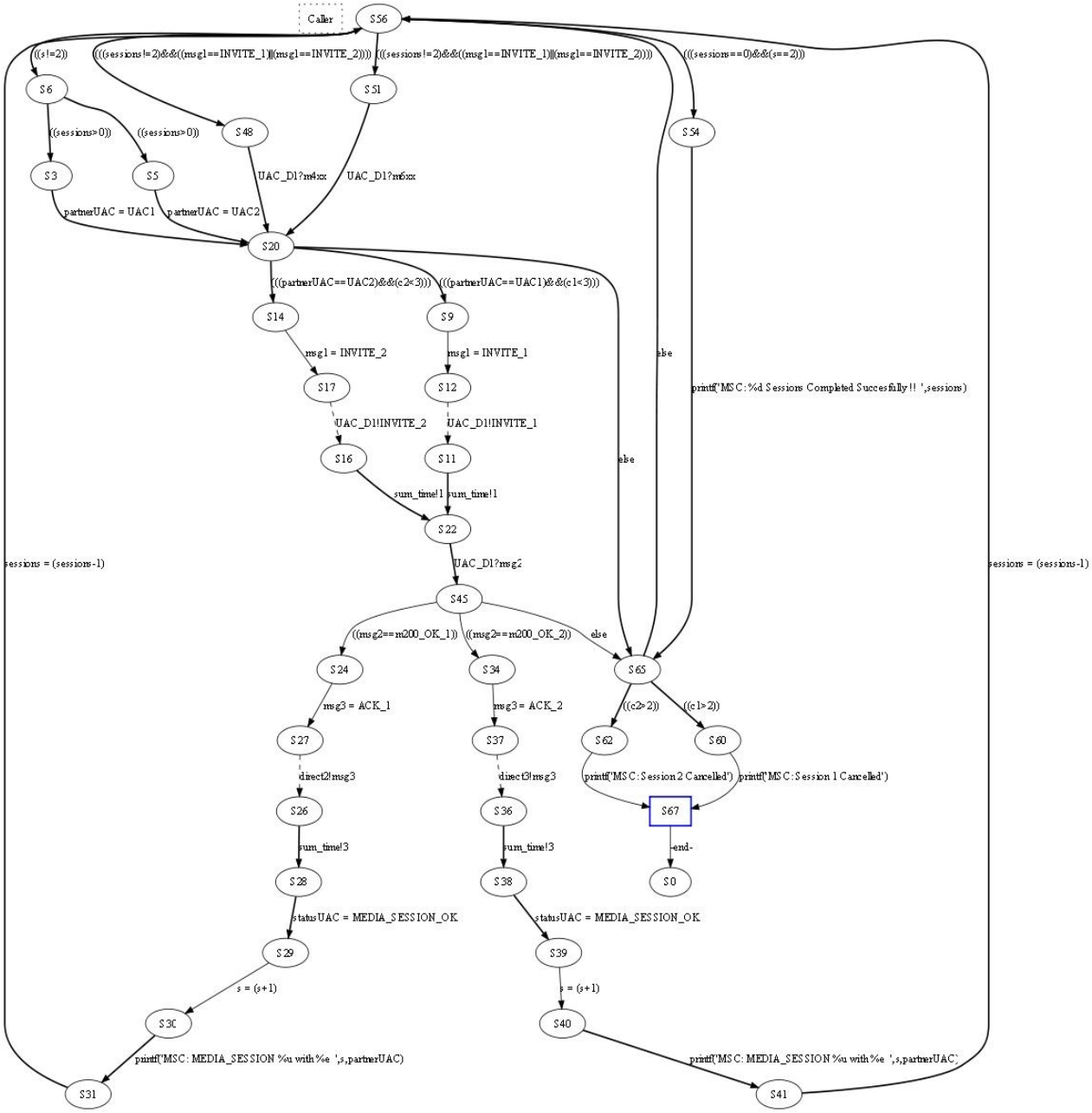

Figure 7: The Caller proctype of the SIP-aSPM model

The `Caller proctype` can be used for analyzing all SIP-aSPM execution scenarios that involve one or at most two parallel sessions. In a two-session execution scenario, the `UAC` non-deterministically selects the first `Callee` (either `UAC_1` or `UAC_2`) and dispatches an `INVITE` message (named `INVITE1` if the recipient is `UAC_1` or `INVITE2` if the recipient is `UAC_2`). The `Domain1 proctype` then forwards the `INVITE` message to `Domain2`. Upon receipt of the message by `Domain2`, there are three possible responses, namely: (i) redirection of the `Caller` entity (message `3xx`), (ii) request failure (message `4xx`) or (iii) global failure (message `6xx`). Redirection involves reform of the received `INVITE` message, in order to incorporate the new `Callee`'s address. The `Caller` responds only to the

possible failure messages (4xx and 6xx) by sending a new INVITE message, if and only if he has not previously received three error messages. In the latter case, Caller drops the call and session establishment fails.

The behavior of the Caller proctype is shown in the finite state automaton of Figure 7. According to this automaton, if there is one opened session, then at any time of the shown procedure it is possible to open a second session with the other Callee. If less than 3 error messages have been received (guards shown in S14 and S9) the 2 sessions with the Callees are successfully completed (guard shown in node S51). Otherwise (node S65), the session for which the number of received error messages exceeds the threshold is eventually terminated.

If there is no error, Domain2 forwards each INVITE message to the Callee's address and waits for his approval. The Callee proctype generates a 2xx response message and sends it to the Caller via the Domain2 server (message shown as m200_OK in the S24 and S34 nodes of Fig. 7). When the Caller receives an expected 2xx message, he responds with an ACK that is sent directly to the Callee, thus establishing a media session (nodes S31 and S41 in Fig. 7, and node S25 in the Callee automaton of Fig. 8).

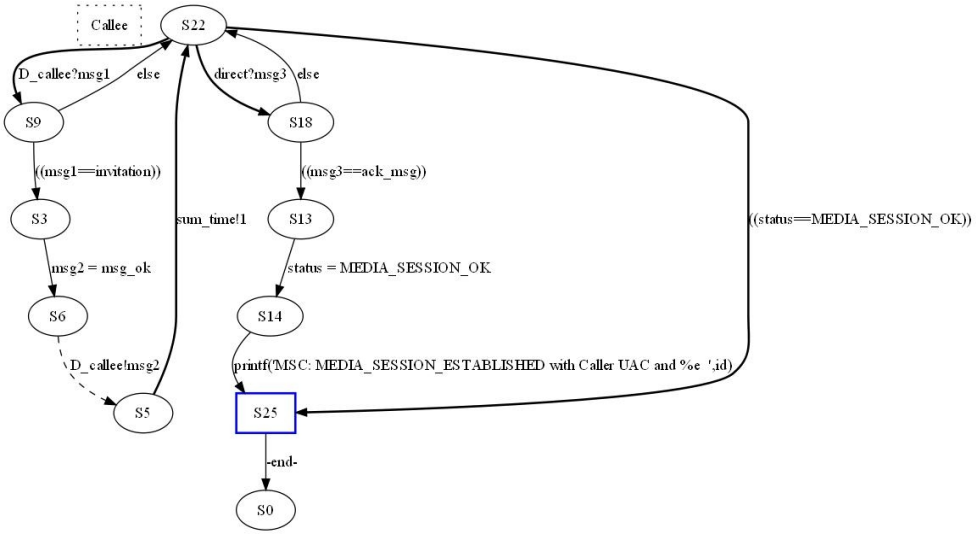

Figure 8: The Callee proctype of the SIP-aSPM model

In the automaton of Fig. 8 for the Callee proctype, the node sequence S22-S9-S3-S6-S5 shows how a Callee handles an INVITE message from his domain. There is only one possibility, i.e. to respond with a 2xx message, while potential errors may be introduced only by the Domain2 proctype, whose behavior is not shown for the sake of brevity. Dashed lines represent the final transitions in cycles that eventually terminate.

The automaton of Fig. 9 introduces the behavior of the Timer proctype and shows the timer updates for all processed messages. Under the given assumption that all SIP messages are delivered to the intended recipient, every message increases the total SIP session time according to the corresponding mean value of the measurements shown in Table 1.

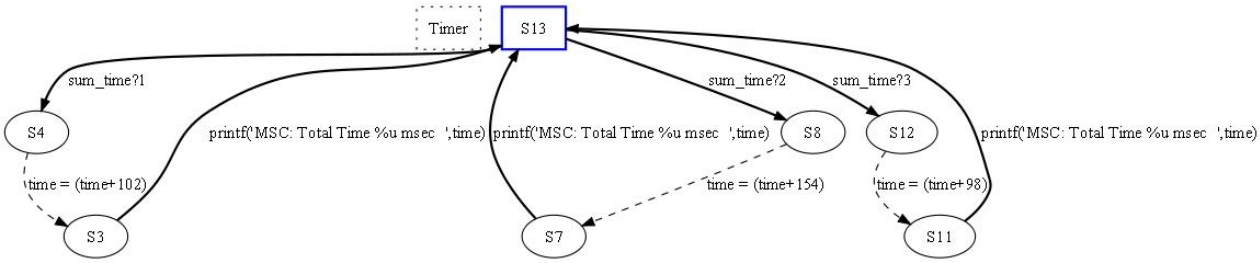

Figure 9: The Timer proctype of the SIP-aSPM model

6 Verification results

Robustness analysis of the SIP-aSPM mechanism is based on the model's state space, which includes the error paths that may occur in a real SIP communication. All aSPM policy effects have been taken into account through the non-deterministic selection between all messages that the policy can generate in each protocol step and by having included in the model, the cost in time for message processing. Error-free paths should not end in deadlock states, meaning that the SIP-aSPM interactions do not cause invalid end (termination) states. When the number of occurred errors can be tolerated (less than two), reachability of end-states within some expected time bound grounds our confidence that the SIP-aSPM mechanism does not involve non-progress cycles (livelocks) or message overload that violates call establishment timeliness. Finally, fairness for the service users ensures that if an UAC initiates a SIP session before some other UAC and no errors occur, then the first UAC establishes a call before the second one.

For model checking the discussed properties:

- we defined assertions that were checked in all reachable states, and
- we developed appropriate formulations in the Linear Temporal Logic (LTL), which is supported by the SPIN model checker.

Assertion monitors that are defined as PROMELA `proctypes` check all reachable states for possible violations of the monitored properties. If an assertion is violated, then the generated counterexample allows the analyst to discover possible aSPM design flaws. Let us consider the condition that at most 2 error messages have been emitted in all sessions. Such an assertion is expressed as follows:

```
active proctype Monitor()
   {
   do
   :: assert(((errors_1<= 2) || (errors_2 <= 2))
         && (sum_time <= TIME_Constraint))
   od
   }
```

where `errors_1` and `errors_2` are the sums of error messages 3xx, 4xx or 6xx for the respective SIP sessions and `TIME_Constraint` is the time period (msec) in which the initiated SIP session(s) are expected to establish the call(s). In Table 3, the results for the property P4 reports reachability of some state, where state exploration stops, due to a detected assertion violation. This result shows the existence of execution paths with more than two error messages for some protocol session, which – as expected – end in abnormal protocol termination.

Timely completion of call session establishment was checked by the LTL formula

```
Q1: [](q → p)
```

where → represents the left associative implication, `[]` is the temporal operator always, and `p` and `q` are propositions defined by the following assignments:

```
#define p time < TIME_Constraint
#define q (sessions==0)
```

Variable `sessions` is initialized with the value 2 and tracks the number of sessions that have been completed by the `Caller`. If `sessions==0` then 2 SIP sessions have been successfully completed. Formula Q1 is therefore interpreted as:

"*If in a reachable state both sessions have successfully finished, this happens in less than* `Time_Constraint` *msec*".

Table 3 shows the results obtained by model checking Q1 for two different time constraints. More specifically, property P1 shows that when the aSPM is enabled, 1 SIP session is successfully completed in less than 6500 msec. P2 refers to the time needed for completing 2 parallel sessions and shows that this is done in less than 10000 msec.

Regarding the fairness characterization of actions, we distinguish between *weak fairness* and *strong fairness*. An action is weakly fair, when if it is *enabled for infinitely long time*, it will be eventually executed. On the other hand, an action is strongly fair, when if it is *enabled infinitely often*, it is eventually executed. In the SIP-aSPM model, session initiation for a `Callee` is a weakly fair action. Let us consider the variables `q_init` and `p_init` with

```
#define q_init init_UAC_1=TRUE & errors_1=0
#define p_init init_UAC_2=TRUE & errors_2=0
```

encoding the two states in which the SIP `Callees` can start interacting with the `Caller` (accept an INVITE message) and no errors have been occurred. The property

```
Q2: [] (((q_init) -> (q && (!p)))  ||  ((p_init) -> (p && (!q))))
```

encodes the weak fairness of session initiation for the `Callee proctype`, where the variables

```
#define q status_UAC_1=TRUE & errors_1=0
#define p status_UAC_2=TRUE & errors_2=0
```

represent a successful SIP session for the respective `Callee`, with no errors encountered. Fairness properties like the one given by Q2 for the SIP protocol show that the process to be executed is fairly chosen between the ones that are simultaneously enabled [36-37]. In the context of the SIP-aSPM model this is interpreted as the fact that the `Caller` selects a `Callee` with equal probability among the ones listening in `Domain2`. Q2 may be also interpreted as follows: if no errors occur for the `Callee` who first interacts with the `Caller`, then the same `Callee` will be the first who will complete the SIP session successfully. Property P7 of Table 3 provides the verification results for query Q2, with the SPIN reporting success in a state space of 2.9e+06 states and 7.022e+06 transitions.

Fairness is a prerequisite for model checking *liveness properties*, which aim to show that a desirable possibility will eventually happen. In our SIP-aSPM mechanism we verified the liveness property Q3 for the `Caller`, in the parallel sessions setting,

```
Q3: [](u -> q_init)->(<>(q && (!p)))||(u -> p_init)->(<>(p && (!q)))
```

with

```
#define u status_UAC=TRUE & sessions > 1
```

representing the `Caller`'s state, in which the first INVITE message has been just sent (therefore `sessions` is >1) and no errors have occurred. With `p`, `q`, `p_init` and `q_init` defined as in Q2, this liveness property is interpreted as follows:

"*If the Caller has initiated a SIP session with one of the listening Callees and no errors occur, then call establishment with the selected Callee will eventually succeed*".

Table 3 summarizes the SPIN results for the mentioned properties (P1, P2, P4 and P7), as well as for the property P8, which shows the truth of Q3. In order to compare how the aSPM mechanism affects SIP timeliness with the case of no anti-SPIT protection, we note the experiments where the aSPM was enabled.

Table 3: Verification results for the robustness analysis of SIP-aSPM

#	Property description	States	Transitions	Memory (MB)	Property definition	Verification result
P1	Full state space search: one completed SIP session within 6500 msec (aSPM enabled)	3.8e +06	7.1811e+06	600.712	Q1	TRUE
P2	Full state space search: two completed SIP sessions within 10000 msec (aSPM enabled)	3.8e+06	7.246e+06	616.11	Q1	TRUE
P3	Full state space search with no errors (absence of deadlock with aSPM enabled or disabled)	3.8e +06	7.1811e+06	570.906	Invalid end-states	TRUE
P4	Partial state space search: assertion that no more than two error messages (3xx,4xx, 6xx) are observed in one of the parallel sessions within 6000 msec (aSPM enabled)	13172	13172	0.924	Assertion	FALSE
P5	Full state space search: one completed SIP session within 3000 msec (aSPM disabled)	3.8e+06	9.238e+06	600.712	Q1	TRUE
P6	Full state space search: two completed SIP sessions within 6000 msec (aSPM disabled)	3.8e+06	7.246e+06	616.114	Q1	TRUE
P7	If no errors occur for the Callee who first interacts with the Caller, then the same Callee will be the first who will complete the SIP session successfully (aSPM enabled or disabled)	3.8e+06	7.022e+06	560.203	Q2	TRUE
P8	If the Caller has initiated a SIP session with one of the listening Callees and no errors will occur, then call establishment with the selected Callee will eventually succeed (aSPM enabled or disabled)	3.8e+06	7.181e+06	585.309	Q3	TRUE

Absence of deadlocks was confirmed by the non-reachability of invalid end states (P3). All model checking results were obtained on a workstation with 3072 MB RAM. Executions of query Q1 were completed in less than 18 sec, occupying an average of 600 MB of memory. Assertion checks for invalid states were completed in less than 6 sec. We believe that such assertion checks are effective for detecting possible 3xx, 4xx or 6xx errors that interfere with the SIP sessions in an unexpected manner.

7 Conclusions and future work

We introduced a policy-based anti-SPIT protection mechanism for VoIP services and its robustness analysis through formal modeling and verification with the SPIN model checker. The so-called aSPM mechanism interacts with the Session Initiation Protocol (SIP) for applying appropriate anti-SPIT measures upon detection of suspicious SPIT message exchanges. In our SPIN model, the policy effects on the flow of SIP message exchanges are represented by non-deterministic selection between the messages that the policy can generate in each protocol step. Model parameters for the time to process the aSPM-generated messages were assigned values from measurements made in an experiment with real VoIP users.

By model checking, we verified the absence of deadlocks in error-free parallel SIP sessions and we confirmed the reachability of failure states, when more than two error messages are emitted in a protocol session. We also showed that the aSPM mechanism preserves weak fairness for the service users and guarantees that error-free SIP sessions eventually complete with success. Finally, aSPM causes negligible overhead, thus preserving robustness with respect to timely call establishment, even in a parallel SIP sessions scenario.

The described analysis can either validate anti-SPIT policy robustness or detect possible property violations. In the latter case SPIN generates a counterexample that can reveal the policy action, which causes the detected problem. We therefore argue that our robustness analysis is a valuable tool in the design of anti-SPIT policies or more generally in the design of security policies that interact with some protocol's services. A hindering factor against the effectiveness of the proposed analysis is the still outstanding problem of state space explosion. We opted to integrate the policy effects in the SPIN model directly, through the use of non-determinism. This may be a limitation, with respect to the applicability of our analysis in policy conditions with complex dependencies.

As a future research prospect we plan to seek for appropriate model abstractions that will enable robustness analysis of complex policies. Finally, it is also desirable to model check the properties related to timely call establishment in scenarios with more than two parallel sessions. This may happen through an improved model representation that opens the possibility to exploit symmetries and other techniques for state space reduction.

Acknowledgements

This work was performed in the framework of the SPHINX (09SYN-72-419) project, which is partly funded by the *Cooperation* Programme of the Hellenic General Secretariat for Research & Technology.

References

1. T. Walsh, D. Kuhn: Challenges in Securing Voice over IP, IEEE Security & Privacy, pp. 44 - 49, Vol. 3 No. 3, 2005.

2. S. Sawda, O. Urien: SIP security attacks and solutions: A state-of-the-art review, in Proc. of the IEEE International Conference on Information and Communication Technologies: From Theory to Applications (ICTTA '06), Vol. 2, pp. 3187-3191, April 2006.
3. J. Rosenberg, C. Jennings: The Session Initiation Protocol and Spam, Network Working Group, RFC 5039, January 2008.
4. J. Quittek, S. Niccolini, S. Tartarelli, M. Stiemerling, M. Brunner, T. Ewald: Detecting SPIT Calls by Checking Human Communication Patterns, in Proc. of IEEE International Conference on Communications (ICC'07), pp. 1979-84, June 2007.
5. D. Graham-Rowe: A Sentinel to Screen Phone Calls Technology, Technology Review (http://www.technologyreview.com/read_article.aspx?id=17300&ch=infotech) 2006 (accessed November 8, 2010).
6. M. Winslett: Policy-Driven Distributed Authorization: Status and Prospects, in Proc. of the 8th IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 12-18, June 2007.
7. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler: Session Initiation Protocol (SIP), RFC 3261, June, 2002.
8. G. Marias, S. Dritsas, M. Theoharidou, Y. Mallios, D. Gritzalis: SIP vulnerabilities and antiSPIT mechanisms assessment, in Proc. of the 16th IEEE International Conference on Computer Communications and Networks (ICCCN 2007), USA, pp. 597-604, August 2007.
9. D. Gritzalis, Y. Mallios: A SIP-based SPIT management framework, Computers & Security, Vol. 27, Nos. 5-6, pp.136-153, October 2008.
10. S. Dritsas, Y. Soupionis, M. Theoharidou, J. Mallios, D. Gritzalis: SPIT Identification Criteria Implementations: Effectiveness and Lessons Learned, in Proc. of the 23rd International Information Security Conference (SEC-2008), Samarati P., et al. (Eds.), pp. 381-395, Springer, September 2008
11. Y. Soupionis, S. Dritsas, D. Gritzalis: An adaptive policy-based approach to SPIT management, in Proc. of the 13th European Symposium on Research in Computer Security (ESORICS 2008), Lopez J., Jajodia S. (Eds.), pp. 446-460, Springer, October 2008.
12. Y. Soupionis, D. Gritzalis: ASPF: An adaptive anti-SPIT policy-based framework, in Proc. of the 6th International Conference on Availability, Reliability and Security (ARES-2011), Pernul G., et al. (Eds.), pp. 153-160, Austria 2011.
13. J. Quittek, S. Niccolini, S. Tarterelli, R. Schlegel: Prevention of Spam over IP Telephony (SPIT), NEC Technical Journal , Vol. 1, No. 2, pp. 114-119, 2006.
14. D. Agrawal, J. Giles, K.-W. Lee, K. Voruganti, K. Filali-Adib: Policy-based validation of san configuration, in Proc. of International Workshop on Policies for Distributed Systems and Networks, June 2004.
15. D. Agrawal, S. Calo, J. Giles, K.-W. Lee, D. Verma: Policy management for networked systems and applications, in Proc. of the IFIP/IEEE International Symposium on Integrated Network Management, May 2005.
16. E. Baralis, J. Widom: An algebraic approach to static analysis of active database rules, ACM Transactions on Database Systems, Vol. 25, No. 3, pp. 269-332, 2000.
17. M. Sloman, E. Lupu: Security and management policy specification, IEEE Network, Special Issue on Policy-Based Networking, Vol. 16, No. 2, pp.10-19, 2002.
18. P. Gama, P. Ferreira: Obligation policies: An enforcement platform, in Proc. of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '05), June 2005.
19. G. Holzmann: The model-checker SPIN, IEEE Transaction on Software Engineering, Vol. 23, No. 5, pp. 279-295, 1997.
20. The SPIN model checker website (http://spinroot.com/) (accessed May 23, 2011).

21. S. Basagiannis, P. Katsaros, A. Pombortsis: Intrusion Attack Tactics for the Model Checking of e-Commerce Security Guarantees, in Proc of the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP '07), pp. 238-252, Springer-Verlag, September 2007.
22. G. Holzmann, The SPIN Model Checker - Primer and Reference Manual, Addison-Wesley, 2003.
23. Basagiannis, S. Katsaros, P., Pombortsis, A.: Synthesis of Attack Actions Using Model Checking for the Verification of Security Protocols, Security and Communication Journal, Vol. 4, No. 2, pp. 147-161, February 2011.
24. ITU-T Recommendation H.323, Packet-based multimedia communications systems, 2009.
25. P. Zave: Understanding SIP through model-checking, in Proc. of the 2nd International Conference on Principles, Systems and Applications of IP Telecommunications, pp. 256-279, Springer-Verlag, July 2008.
26. L. Liu: Verification of the SIP Transaction Using Colored Petri Nets, in Proc. of the 32nd Australasian Computer Science Conference, pp. 63-72, January 2009.
27. A. Schaeffer-Filho, E. Lupu, M. Sloman, S. Eisenbach: Verification of Policy-based Self-Managed Cell Interactions Using Alloy, in Proc. of the 10th IEEE International Symposium on Policies for Distributed Systems and Networks (Policy-2009), pp.37-40, July 2009.
28. SER Server, ver. 2.0 (www.iptel.org/ser) (retrieved May 22, 2011).
29. SIPp traffic generator for the SIP protocol (http://sipp.sourceforge.net/) (accessed August 17, 2010).
30. Y. Soupionis, S. Basagiannis, P. Katsaros, D. Gritzalis: A formally verified mechanism for countering SPIT, in Proc. of the 5th International Conference on Critical Information Infrastructure Security (CRITIS-2010), Xenakis C., Wolthusen S. (Eds.), pp. 128-139, Springer, September 2010.
31. Antispit Policy Schema (http://users.auth.gr/~basags/sip/AntiSpit_Policy_ Schema_1.xsd).
32. The SIP-aSPMv2 Model (http://users.auth.gr/~basags/sip/SIP-aSPMv2.prom).
33. Cisco Systems, Session Initiation Protocol gateway call flows and compliance information SIP messages and methods overview (http://www.cisco.com/application/pdf/en/us/guest/products/ps4032/c2001/ccmigration_09186a00800c4bb1.pdf) (accessed August 07, 2011)
34. Cisco Systems, "SIP Messages and Methods Overview" (accessed August 07, 2011) (http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/rel_docs/sip_flo/preface.pdf).
35. Lowe G., Roscoe A.: Using CSP to detect errors in the TMN protocol, IEEE Transactions on Software Engineering, Vol. 23, No. 10, pp. 659-669, 1997.
36. H. Völzer, D. Varacca, E. Kindler: Defining fairness, in Proc. of 15th International Conference on Concurrency Theory (CONCUR), pp. 458-472, Springer-Verlag, August 2005.
37. A. Sistla: Safety, liveness, and fairness in temporal logic, Formal Aspects of Computing, Vol. 6, pp. 495-511, 1994.
38. IEEE, IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990, 1990.
39. F. Saad-Khorchef, A. Rollet, R. Castanet: A framework and a tool for robustness testing of communicating software, in Proc. of the ACM Symposium on Applied Computing (SAC), pp. 1461-1466, March 2007.
40. X. Yin, Z. Wang, C. Jing, J. Wu: A formal approach to robustness testing of network protocol with time constraints, Security and Communication Networks, Vol. 4, No. 6, pp. 622-632, 2011.
41. F. Belli, A. Hollmann, W. Eric Wong: Towards scalable robustness testing, in Proc. of the 4th International Conference on Secure Software Integration and Reliability Improvement, pp. 208-216, June 2010.
42. N. Laranjeiro, M. Vieira, H. Madeira: Robustness validation in service-oriented architectures, in Architecting Dependable Systems VI, pp. 98-123, LNCS 5835, Springer-Verlag, 2009.