ASPF: An Adaptive anti-SPIT Policy-based Framework

Yannis SOUPIONIS, Dimitris GRITZALIS
Information Security and Critical Infrastructure Protection Research Laboratory
Dept. of Informatics, Athens University of Economics and Business (AUEB), Greece
e-mail: {jsoup, dgrit}@aueb.gr

Abstract - **SPam over Internet Telephony (SPIT) is a rising IP voice telephony threat. Voice over IP enables the transmission of telephone calls over the Internet, as opposed to plain old telephone service. Internet Telephony essentially means low-cost phone calls, i.e. a clear benefit for both consumers and businesses, which may also lead to cheap methods of mass advertising. Still, industry observers warn that VoIP's low-cost and openness makes it relatively easy for spammers to send unsolicited audio-commercials to VoIP voice-mail inboxes, in much the same way they currently bombard e-mail inboxes. In this paper we set the foundations of an adaptive approach that handles SPIT through an adaptive anti-SPIT policy-based framework (ASPF). ASPF incorporates a set of rules for SPIT detection, together with appropriate actions and controls that should be enforced, so as to counter these attacks. ASPF is formally described through an XML schema. A working prototype is also demonstrated for evaluating ASPF. The prototype is able to make policy alterations, based on abnormal network events.**

Keywords: Security Policy, Spam over Internet Telephony (SPIT), Voice over IP (VoIP), Session Initiation Protocol (SIP).

I. INTRODUCTION

The Internet Telephony (Voice over IP) is a developing technology that promises a low-cost, and high-quality and availability service of multimedia data transmission. Inevitably though, VoIP "inherited" not only these positive features of internet services, but also some obvious disadvantages [1]. One of the main disadvantages is Spam over Internet Telephony (SPIT) [2,3], which is the popular expression of Spam in VoIP network environments. SPIT is a potentially serious issue that IP telephony will be facing in the near future. This is the reason why several organizations have already started developing mechanisms to tackle SPIT [4,5]. Additionally, VoIP implementations are mainly based on a multimedia protocol, i.e., Session Initiation Protocol (SIP) [3]. Recent research shows that this protocol is vulnerable to SPIT [7,8]. Thus, a mechanism for adequately preventing and treating this phenomenon is needed.

Systems, which are designed and implemented to provide protection against threats like SPIT, follow common practices and adopt similar principles, with respect to IT Security. A common practice is to develop a policy-based mechanism, which is founded on predetermined conditions [6]. These conditions can be formalized in a structured security policy, which refers to a set of rules defined by an organization. These rules usually describe how one can access or use a network element, a service, or a general collection of physical or logical objects, under protection.

Although policy based frameworks have been successfully implemented in various domains, such as storage area networks [10], networked systems [11] and database management [12], the existing solutions are often restricted to condition-action rules, where conditions are just matched against incoming traffic flows [10, 11]. This results in static policy configurations, where the policy cannot be altered according to context changes. What is needed is an adaptive policy rule set. Moreover, policy mechanisms [6] are independent of the platform implementation and do not inherit any defect of any platform. They are, also, easy to be integrated to any context.

In order to meet the above requirements, we herein propose an adaptive policy-based SPIT management framework, which is able to detect and handle SPIT attacks. The proposed anti-SPIT policy is as an obligation policy [13,14]. It includes a set of policy rules, together with the countermeasures that should be executed whenever a SPIT call/message is detected. Policy rules do define which behavior is not desired ("illegal") in a VoIP system. On the other hand, they do not describe the actions and event sequences that actually "produce" the undesired behavior. Therefore, our approach can collaborate with any existing anti-SPIT mechanism. Also, it can collect data from other anti-SPIT mechanisms, which produce decisions about SPIT incidents, and create the appropriate policy rules. This kind of mechanisms could be those based on reputation or listing (black, grey) methods.

The paper is organized as follows. First, we describe the research work done on SPIT management. In section 3, we describe the methodology we followed to develop the proposed policy-based anti-SPIT framework. In section 4, we present the foundations of our work, which are based on an extensive study of the SIP protocol with an eye towards identifying the SPIT-oriented threats and vulnerabilities. In section 5, we describe how ASPF can be practically applied to a SIP environment. Then, we introduce the evaluation process and we present its effectiveness, as well as the computational resource needs of the proposed framework. Finally, we end with our conclusions and plans for future work.

II. RELATED WORK

In this section, we briefly survey techniques and approaches that focus either on preventing SPAM/SPIT, or on implementing policy infrastructure to VoIP environments.

A. Anti-SPIT Mechanisms and Techniques

A number of anti-SPIT mechanisms and techniques have been proposed so far. Dantu et al. [15] proposed a multi-stage Voice Spam Detection (VSD) filter for VoIP networks. VSD employs Bayesian inference technique to compute the spam probability of an incoming call. During the learning period, human intervention is required to mark unsolicited calls. Balasubramaniyan et al. [16] proposed a technique that uses call durations to build social network linkages and global reputations for users. Long call duration can serve as a call credential

from the caller Alice to the call recipient Bob. Then Bob could leverage this call credential to call user Charlie, who called Alice recently. This scheme assumes that each VoIP user has a public/private key pair and can generate digital signatures. However, VoIP clients in most of the existing VoIP services do not own a private key [17].

Patankar et al. [18] proposed to employ buddy lists of a VoIP network in order to to establish a trust chain between the caller and the recipient. In their model, intermediate nodes are required to generate some messages to help the caller build the trust chain. They assume that a VoIP client shares a separate symmetric key with each user in his buddy list, which is not easily applicable due to the problem of the key exchange and key secrecy.

Shin et al. [19] proposed graylisting for fighting voice spam. The grey level of a caller determines whether the call is accepted. If a caller launches numerous calls in a certain time span, his gray level will increase. Once the gray level exceeds a threshold, all later calls from the caller within a given period, will be blocked. After the caller stops making calls, the grey level will decrease and eventually become below the threshold. The problem of graylisting is its efficiency when spammers exploit numerous VoIP accounts to launch SPIT, or when a Sybil attack is occurred. Mathieu, et al. [20] present an approach which focuses on detecting and mitigating SPIT by using a sniffing-oriented network-level entity. This entity captures filters, analyses the network traffic and identifies SPIT attacks based on specific criteria, from which a weighed sum (spitLevel) is created. The sum is compared to a threshold and if it is exceeded, then the call is classified as SPIT.

B. *VoIP Policy Infrastructure Methodologies*

Existing methodologies for developing anti-SPIT policies are herein described at an abstract level. They focus mainly on high level aspects of security. Regarding the anti-SPIT policy-related work, there are three main implementations. The first one [21] proposes an authorization policy and recommends an XML structure, which identifies possible SPIT and suggests countermeasures. The main drawback of it is that the identification is based mainly on users URI and not on other SIP protocol aspects. Also, it does not include the recommended rules and conditions. An alternative implementation [22] is the Call Processing Language (CPL), which describes and controls Internet Telephony services. It is developed for either network servers or user agents. It provides an XML schema, as well as the proposed values for initial configuration. This approach is quite generic. It focuses on how one can represent VoIP services, but ignores the SPIT phenomenon and how it can be prevented.

III. METHODOLOGY

The specific SPIT management methodology, that we have adopted, is depicted - in a functional manner - in Fig. 1.

A. *Foundation Step*

The first step of the methodology aims at creating the basic elements (conditions/countermeasures) of the anti-SPIT Policy (partially presented in [23]).

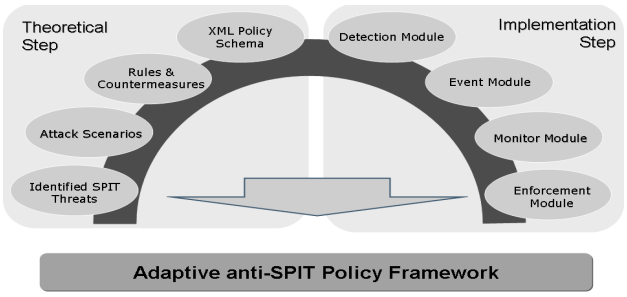

Figure 1. ASPF methodology

The first sub-step was to identify the SIP-oriented SPIT threats by examining and analyzing the SIP protocol [9]. The result of the SIP analysis was a number of well-defined SPIT-related threats and vulnerabilities, in accordance with the SIP RFC [9]. Before doing so, we studied the email SPAM phenomenon, so as to indicate the ways and techniques that the attackers and the defenders used through time [24,25].

The second sub-step was to develop attack scenarios that can exploit the existing vulnerabilities. We used SPIT-oriented attack graphs to determine how vulnerable the SIP protocol is. We also use it to determine attack scenarios, which were used as input to the next sub-step, the definition of the policy conditions.

The identified conditions are the main element of our policy. When a condition is fulfilled, then a SPIT attack is identified. Moreover, there should be actions capable of countering each and every attack. The majority of actions are SIP messages [26,27], because the policy should be transparent to the administrators and users and keep to a minimum the participation of other applications during message handling.

The final step of this section is the development of an XML schema. The schema links attack scenarios (conditions) and countermeasures (actions) in a formal policy. It, also, produces a flexible policy description, which could be adopted not only by our anti-SPIT module but also by most SIP infrastructures.

B. *Implementation Step*

The way to integrate, enforce, and dynamically update the XML policy is described in the implementation step. The implementation step describes the execution environment of the policy-based framework (Section 6). Although it is based on known policy infrastructures [11,12], specific adjustments have been made, so as to fulfill the needs of a VoIP system. The components of the proposed anti-SPIT policy framework are:

1. *The Policy Decision Module (PDM).* The input to it is the parsed XML document, together with the message attributes. Here all the conditions are checked, so as to find out which are satisfied and which not (actually, first a SIP message is received and parsed, and then the message attributes are checked against the policy elements).
2. *The Event Module (EM).* The EM evaluates if an event is applicable for policy monitoring. Then, it stores it for future evaluation in history log. These events are specified by a custom event set. The policies are updated and altered by the policy optimizer, which does not act in real-

time and is independent from detection and enforcement modules.

3. *The Policy Enforcement Module (PEM)*. If one or more conditions are met, then the proposed action (described in the fulfilled policy element) does take place. The PEM is part of the VoIP infrastructure runtime environment, because it should be able to interfere in the communication process and enforce the policy actions.
4. *The Monitor Module (MM)*. It monitors, in real-time, a number of system attributes. When an undesired behavior occurs, it informs the Event module or Decision module of its characteristics.

IV. FOUNDATION STEP

A. Vulnerabilities and SPIT Criteria Identification

Because of the real-time nature of VoIP sessions and services, handling SPIT is more efficient in the signaling phase - the SIP protocol phase - rather than in-call. An analysis of the SIP protocol yielded the following categories of SPIT-related SIP vulnerabilities and threats [7,8,28]: (a) exploitation of messages and header fields structure, (b) header fields of messages, and (c) general vulnerabilities. For the analysis we considered threats related to SIP protocol vulnerabilities, as well as SIP RFC optional recommendations. We did not take into account any vulnerabilities based on the interoperability with other protocols. This categorization accounts for the various points in the communication process where anti-SPIT policy actions are enforceable, in particular for those entities involved in the SIP session establishment.

Any identified vulnerability actually affects every participating entity of a SIP session. Any participating entity does not have a particular status for the entire session. For example, an entity that participates to a SIP negotiation process might also do so in the receiving/transmitting of SIP messages (requests/responses). This should help the administrator/user to adjust the policy according to his preferences.

B. Attack scenario development

Attack graphs can be primarily exploited for intrusion detection, defense, and forensic analysis [29-31]. In [32] a SIP-oriented SPIT attack model, based on attack graphs, was presented. In this paper, the SIP-oriented SPIT attacks were first described in abstract level, called abstract attacks, which constitute the attack graph nodes. Then, the attack graph was derived from the steps followed to accomplice a SPIT attack and the relationship among the abstract attacks. The attack graph nodes are depicted in Table 1.

TABLE 1 ATTACK GRAPH NODES

Node	Description
1	Find and collect users' addresses
2	Send bulk messages
3	Proxies-in-the-middle
4	Maximize profit
5	Hide identity-track when setting-up an attack
6	Hide identity-track when sending a SPIT call/message
7	Encapsulate SPIT in SIP messages

In Fig. 2 we illustrate the attack graph (left sub-image) and an attack tree example based on node 7. The arrows depict the possible connections/relations between the attacks. The graph does not have a single start-node or end-node; it just demonstrates the exploitation of SIP protocol vulnerabilities, so as a malicious user can conduct a SPIT attack.

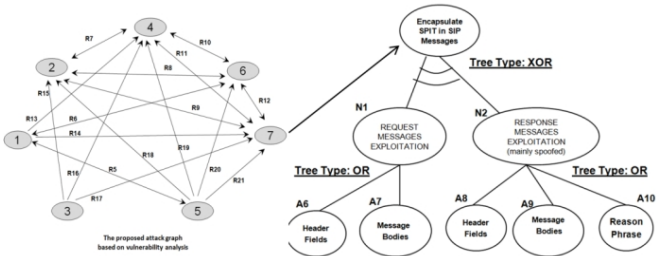

Figure 2. SPIT attack graph and an example of an attack tree

The path followed to produce a SPIT attack leads to a SPIT scenario. A specific attack scenario, which is extracted by the attack graph (attack graph path), follows: In order for the attacker to start sending SPIT message, he has to collect valid SIP URIs (node 1). For this cause, the attacker could start ambiguous requests (identified vulnerability) to proxies (node 2), so as to collect SIP URIs addresses of his potential SPIT victims. Then the "spitter" a) could simple start an attack, or b) could start sending the 300 response messages with multiple addresses in Contact field, so as to add malicious URIs to proxies or UAS (node 5).

C. Policy Approach

Now we present how the attack scenarios can be transformed into SPIT-oriented attack patterns and how this leads to the development of policy rules.

1. Policy Rule

The policy rule is the key element of the policy. A policy rule consists of a Condition, which is a pattern of an identified SPIT attack scenario, and an Action, which is the reaction of the callee's UAC or domain to the SPIT attack.

a. Policy Condition

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

Let us focus the last part of the scenario of previous section. Its attributes are: (a) the message code is 300, (b) there are multiple Contact fields. Each attribute produces a sub-condition. The logical aggregation of these sub-conditions results to the following policy condition:

$$Condition = [Code = 300 \oplus Contact \asymp Multiple] \quad (1)$$

More generally, a condition is defined as:

$$Condition = f(c_1, c_2, , c_k) = c_1 \diamond c_2 \diamond \diamond c_k$$

where c_i is a sub-condition and $\diamond$ denotes the logical operator to be $\oplus$ (AND) or $\otimes$ (OR). The operators that are used in sub-conditions are: (a) = equal, (b) $\asymp$ times of header appearance

(Multiple, One, None), (c) ≈1 approximately equal, (d) > greater, and (e) < less.

b. Policy Action

The other half of a policy rule is the policy action. The absence of the context (inside which each scenario is taking place) is considered critical in deciding the reaction of the corresponding entity that need to be taken. Different contexts mean different security needs. Different security needs translate to different countermeasures. That said, the countermeasures/ actions that can deal with the conditions cannot be strict, as this could be considered an ASPF limitation.

That is why the actions were divided into two categories and a set of suggested actions, that can be taken when each condition occurs, was produced. Moreover, the ASPF can adapt any future recommended actions and integrate them in the proposed set of countermeasures. The policy action categories are:

1. *Block*. This countermeasure leads to the rejection of the SIP message. It occurs when a SPIT condition is true. The SIP message sent to the caller is "403 -Forbidden". Moreover, the action could contain a more informative reason for the rejection. This can aid the request entity to alter the message and meet the necessary requirements of the callee, or her/his domain. A typical SIP message could be "488 - Not Acceptable Here" and description "Inappropriate word in Subject".
2. *Notify*. This countermeasure means that we are not confident whether this message is SPIT or not. Therefore, the message is not rejected, but it is forwarded wherever the administrator/user desires. The request entity receives a SIP 183 "Session in Progress" message, in order for the request entity to be notified that his message has been redirected to a decision-support application. It may be the case that the message is forwarded to the callee after it is altered, so as to comply with the policy rules (the administrator receives the notification in a log file).

An example of a set of actions proposed for the above condition (1) is:

1. The UAC address book is updated with the new addresses.
2. The UAC informs the user for the new SIP addresses
3. The UAC rejects the call and returns a Message 403 "Forbidden"
4. The UAC rejects the call and returns a 488 "Not Acceptable Here" message with description "Multiple Contact fields"
5. The UAC forwards SIP message to another entity and returns a message 183 (Session in Progress)

The final decision of which action will be taken is left to the administrator or to anyone who is responsible to produce the policy instance and secure the particular communication.

2. Policy Instance Creation

The entities that participate in a communication can have a different policy, i.e., the policy instance for each entity includes a different set of policy rules. The method to produce a policy instance, based on the development of SPIT condition and countermeasures for a participating entity, is depicted in Fig. 3. Which rules are going to be included is delegated to the person who is responsible for each communication entity.

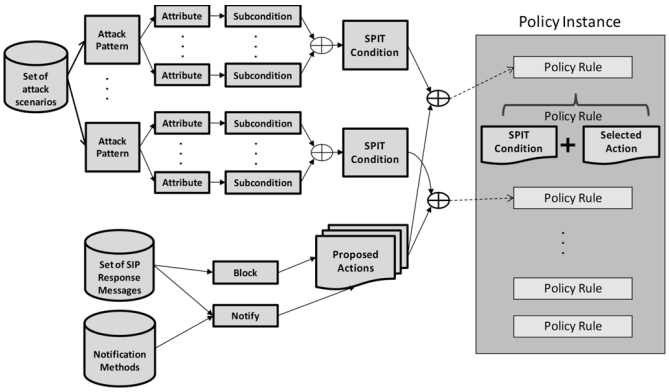

Figure 3. ASPF policy creation

D. .XML representation

The XML schema representation includes the identification characteristics of the attack attributes, together with the relation between them. The schema is produced in order to be easy for the administrator to develop a policy element that is easily processed by an automated procedure. The xml schema is really extended. We now illustrate briefly the main tags.

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, (d) Callee's domain), and the Rule, which obtains the policy element. The second element of the RuleItem is the Rule, which records a certain condition and introduces the proper action, when this condition is met. A Rule element consists of three tags, namely: (a) the Trigger tag, which denotes when the rule is checked, and its values are: Receive Message, Create Message, (b) the Condition tag, and (c) the Action tag, which refers to the action that must be applied.

The second tag of Rule Item, i.e., the condition. The condition can appear one, many, or no times in a Rule, in order to fulfill the produced attack pattern. It consists of: (a) the Item tag, which can be a header field or a request type (INVITE, OPTION, etc.), (b) the Value tag, which is the checked value of the Item, (c) the Relation tag, which is the relation/logical operator between the Item and the proposed value, and (d) the Logical Operator tag, which is an optional element and exists either if the condition includes more than one sub-conditions or if the condition is easier to be expressed like a negation condition. The values of this tag can be only AND, OR, and NOT.

The third part of the Rule tag is the Action. The action element is processed only if the Trigger and the Condition are fulfilled. An action consists of (a) the Obligation tag, which has two distinct values "Must" and "Must Not" whether the action is optional or not, (b) the Actiontype element, which indicates the exact action to be made. The most common actions are (a) Notify, which suggests implies a process that should be done by the proxy, and (b) Return message, which means

[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.

composing of a return message that requests for a new task by the caller's UAC or it just terminates the call.

V. Implementation Step

Once the XML schema is produced, then the VoIP administrator or the user can deploy an XML policy instance, depending on the attributes of the environment. In this section we describe how an XML policy instance is integrated and enforced in a VoIP SIP infrastructure. All the building modules will be presented, in particular those that assist the framework to be dynamically adaptive to the network changes, based on history/time events. Finally, the evaluation of the proposed framework will be also presented.

A. Architecture Components for PolicyEnforcement

We now present the modules that help the policy be enforced in the VoIP infrastructure. The policy enforcement process is based on existing research [33-35], which has been customized to the specific needs of a VoIP environment.

The three main modules used for the policy enforcement are: (a) the Policy Decision Module, (b) the Policy Enforcement Module, and (c) the Policy Repository.

1. Policy Decision Module

The Policy Decision Module (PDM) is responsible to check for "illegal" messages. It retrieves the appropriate XML policy instance document and the SIP message attributes, and decides whether the SIP message is a SPIT threat or not. Every SIP message passes through the PDM, except for those which suffer by syntactical errors, which are rejected by the SIP parser. In this module all the policy conditions are checked. In order to minimize the process time we use Xpath queries [41] on the XML policy for each attribute/sub-condition. Finally, the PDM can send a SIP message for further evaluation to Event Module, and wait for an answer.

2. Policy Enforcement Module

The Policy Enforcement Module (PEM) is the framework entity that enforces the decisions made by the PDM. The involved actions are clearly stated in the XML policy. They mainly refer to a SIP message. When a condition is met, then the proposed action does take place. The PEM acts in the runtime system and sends an operation description (mainly a SIP message) to the application level entity, which is the domain or the corresponding UAC.

3. Policy Repository

The Policy Repository (PR) is responsible for keeping the XML policy instances. The PR is able to keep XML policies not only for domains, but also for each user. Moreover, the PR is able to keep policies to more than one host. Finally, the PR is initiated whenever the Decision module requests a specific policy for a newly arrived SIP message.

B. Architecture Components forAdaptive and Dynamic Policy Management

Dynamic policy management requires a form of history-based event control. The decision whether the calls are characterized as SPIT or not, depends on both, their having already performed related calls (in the same context), and evaluating monitored attributes, which have being changed in a not compliant way. For example, the UAC A might call UAC B several times per day, but it is going to be strange if this number of calls start increasing significantly. Therefore, if the call number exceeds a threshold, then the policy management framework should be alerted and may ban these calls. The two main modules, which support the dynamic nature of the proposed framework, are the Event Module and the Monitor Module. Those two, together with the History/Event module, make the framework effective without producing any significant annoyance.

1. Event Module

The Event Module (EM) is being activated when the Policy Decision Module (PDM) sends it a message. Not all messages are sent to EM, but only those indicated to either produce or be able to produce a SPIT attack or other undesired behavior. The SIP rejected messages are sent to EM, to be stored in the history log for future evaluation. The main procedure of the EM is the evaluation process. This is called policy optimizer. It is initiated by the EM periodically and tries to identify call patterns and characteristics, e.g., large number of errors, unanswered calls from a domain, etc.

SIP messages evaluation is done through the following criteria [37,38]:

1. The number of call messages, in a certain time frame that was made by a specific user.
2. The number of unanswered calls in a certain period of time.
3. The number of errors of the specific domain or user in certain period of time (as those messages were sent to a nonexistent client, there might be a SPIT attack).
4. Identification of the SIP messages pattern, i.e., if the receivers' addresses follow a specific pattern (e.g. alphabetical SIP URI addresses), then the call (message) is flagged as SPIT.

When a specific pattern is recognized, then the module either alerts the policy administrator, or - if it is a simple situation, like blacklisting a domain - it updates the corresponding policies in the policy repository. The EM evaluation process (policy optimizer) allows the decision made to take into account not only the current event, but also the relevant previous events (history-based), in order to meet the SPIT complaint criteria. The policy optimizer runs periodically, but it can also run on demand if the EM receives either a request by the PDM or a notification by the Monitor module. This happens when: (a) the PDM or MM identifies an abnormal event, like an increased network flow, or (b) there is a policy action requesting it, for example this can be adjusted by the administrator for every message from a specific domain.

The number of calls the callee could receive before the Event module reaches a conclusion could be huge. In order to minimize the callee annoyance, our policy we could include the CAPTCHA solution [39] as a notify action. Thus, every time a call reaches the callee and the caller is not known, the policy could redirect the call to a CAPTCHA module to identify whether the caller is a human or bot.

2. Monitor Module

The Monitor Module (MM) helps the proposed framework collect useful data. It does so without overloading the decision

module by messages or affecting the performance of the system.

The Monitor Module supervises the consumption of the computational resources, i.e., it checks the system CPU and memory. The associated consumption involves the execution of specified policy actions in the runtime system and the evaluation process of the EM.

In order for the monitor module to send a notification to the Event Module, the policy administrator should assign two upper thresholds, one for the system memory (MemT) and one for the system CPU (CPUT). When the monitored values reach the thresholds, the event module is informed and it sequentially notifies the decision module. After that, the decision module stops some of the less significant actions, like sending the notification actions to enforcement module and postponing them on-demand evaluation process. When the system comes back to normal, then all the modules are notified by the monitor module.

3. History/Event Module

This module is responsible for storing information regarding SPIT. The History Event Log (HEL) keeps the characteristics of the messages that are blocked or characterized as SPIT. The messages are received either by the Event module or by the Monitor module.

C. The Proposed Platform

The proposed anti-SPIT policy-based framework is depicted in Fig. 4. A run-time example will be herein illustrated.

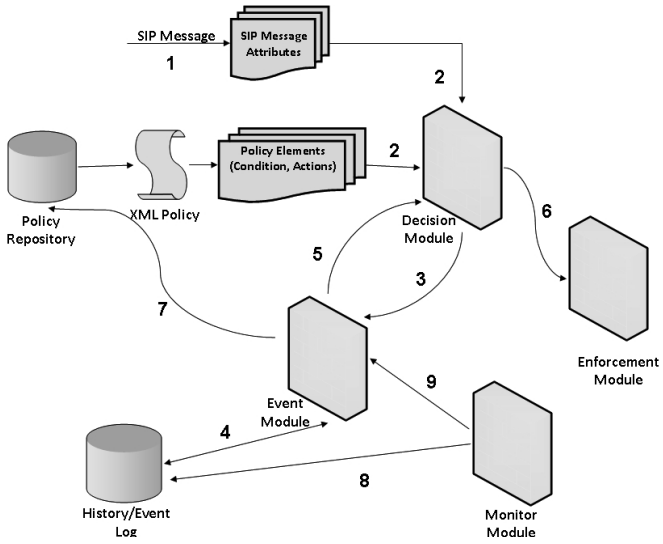

Figure 4. ASPF modules

1. A SIP messages arrives and the SIP parser receives it. The SIP parser is an automated process, integrated in our module. It can scan SIP messages and extract the message attributes (the main attributes are the header fields, such as From, Contact, and SIP-URI and their values).
2. The Policy Decision Module (PDM) receives the extracted SIP messages attributes and the XML corresponding policy instance elements. The policy is taken from the Policy repository or it is received directly by a PDM request. The policy elements are extracted by an XML parser that allows navigation to the entire document, as if it was a tree of "Nodes". The parser reads XML into memory and provides access to tag values of the document.
3. If an event is identified as potential SPIT, then the Policy Decision Module (PDM) informs the Event Module. In this case, PDM may ask for an immediate event evaluation, based on the corresponding previous event.
4. The Event module stores the message details to the History/Event log repository. If there PDM asks for immediate evaluation, then the Event Module informs the decision module if there are any different actions to be taken, but those already written in the policy. Finally, if another policy rule applies for the specific SIP message and this rule makes the message be rejected, then the PDM rejects the message and asks the EM to record this message as rejected.
5. The Policy Decision Module takes the input from the event module and reaches to a verdict if there are any actions to be taken.
6. The PDM returns the decision concerning the execution of the specified action. The policy enforcement module translates the action into the runtime system format (mainly a plain SIP message) and returns it to the corresponding communication entity (SIP server or UAC).
7. Periodically, an Event module process (policy optimizer) evaluates the collected events (step 3), that are stored in History/log event and calculates for each event the aggregated value (e.g. how many times it happened in the last 5 min). If the event value exceeds a specific threshold, then it updates the relevant policy rule in the corresponding policy instance and informs the policy administrator, who can analyze any recorded events and interfere when he feels appropriate.
8. The Monitor Module (MM) examines the SIP messages, based on the policy optimizer criteria. Whenever it identifies an inappropriate message, it stores its attributes to the History/Event log.
9. MM also checks the run-time computational resources. It informs the event module, whenever they pass a specific threshold, thus helping the policy platform (ASPF) react dynamically.
10. Finally, a web view was developed, with which the administrator can monitor the overall system status, calculate event repository size, and update/optimize the policy instances. Steps 4 and 5 are executed only if there is a need for immediate evaluation by the PDM.

VI. EVALUATION SECTION

The ASPF framework is able to handle more than 200 SIP messages simultaneously (an acceptable limitation, as a VoIP provider handles from 40-60 calls/sec according to French Telecom [22]). That said, we will demonstrate the effectiveness of the proposed framework, as well as its impact to the performance of the server (host system).

Performance depends on two metrics: (a). "How much of our system resources are consumed", and (b). "Is the time needed for handling SIP messages a noticeable delay". As per the first question, and in order to quantify how our policy platform affects the overall system performance, we have to record the corresponding values of the monitoring attributes between two timestamps. Let the execution of our policy framework begin at time t1 and end at time t_n. In order to simplify the real overload of ASPF, we should first record the SIP infrastructure performance without ASPF, and then record the values in the same time period with the use of ASPF. The proposed formulas for the system resources consumption are:

$$\text{CPU usage } (\%) : \Delta CPU_i = C'_i - C_i$$
$$\text{Memory usage } (\%) : \Delta Mem_i = Mem'_i - MemC_i,$$

where Mem'_I, C'_I are recorded values with ASFP activated, and Mem_I, C_I are the recorded values without the ASPF use.

The time needed for SIP messages to be processed by our module was calculated by recording the process time of each message by our framework. With these goals in mind, two scenarios were tested for demonstrating the ASPF efficiency and evaluating the metrics. The SIP message flows, which fulfill the scenarios, are based on known SIP call flow examples [40,41].

In order to execute the test scenarios, two laboratories environments were constructed. The first one is a small network environment. The second is an extensive infrastructure network, which was created with simulation software (modelnet [42]). The SIP server implemented is the SER server [43], an open source software product [43].

A. Small Laboratory Environment

The small network environment (Fig. 5) consists of the following fundamental entities:

- A SIP SER server. It has been customized in order to register users, redirect SIP messages, and establish calls. The PC used for setting up the SER server was a Pentium 4, 2.8GHz, 1GB RAM, running the FEDORA 9 Operating System.
- An ASPF platform. It has been installed in the SIP server, at the "logical" entrance of the network environment, and includes the domain and clients policies.
- A number of soft phone clients that have been part of the VoIP domain. The clients are active (i.e., ready to interact with incoming calls). The soft-phone software is the open-source twinkle [44].
- Various external clients. These clients are programmed to make new calls to the internal clients. The exact number of the external clients depends on each use case/scenario. The calls are initiated by using SIPp [45], which establishes calls as well.

1. Proposed Scenario and Results

The scope of the scenario is to identify if the ASPF is able to a) manage an increased flow of calls without abating its functionality or the availability of the VoIP service, b) use extensively the policy repository, which means that when a call is received for specific clients then his XML policy instance is retrieved, and c) characterize a caller as spitter based on the outcomes of the Event module and the History log.

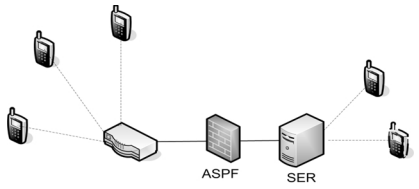

Figure 5. Test laboratory environment

In order to monitor and record the behavior of our platform under heavy load of SIP messages, we produced a high number of incoming calls. In order to do so, we implemented 10 external clients, which are programmed to send 10.000 SIP messages (INVITE), 100 registered clients receiving the messages and a policy file for 80 ones. The callee's policy was retrieved, when a message was received and the callee was identified by the From Field. If a message referred to a client/-recipient who was not registered, the default domain security policy was applied. Moreover, the number of legitimate and non-legitimate messages was random, but only 2 clients were initiating non legitimate call flows. The non-legitimate calls were related to a) blacklisted domains, b) messages with dubious data, like those containing MIME and c) response messages with number 300 like those described in section 4.

The first result of the test was that the ASPF worked properly the whole period of time. That was the case, even though there were queries to the database, I order to identify the malicious users/domains and retrieve the client's policies, which make the computational resources consumptions greater. The SIP messages were generated randomly towards the SIP clients. Each one of the 100 clients has received at least 1 message, thus all the policy files were processed.

As far as ASPF's adaptability is concerned, the experiment showed that 309 messages were forwarded. It took 360.192 msec till the ASPF apprehended that specific clients were conducting a SPIT attack. Afterwards, none of the possible SPIT messages were forwarded and all were banned, because the clients where blacklisted The ASPF has marked all the 309 messages (based on the client's or domain policy) to be forwarded later, either to the client, or to an external module that could be an audio CAPTCHA.

The average message processing time by ASPF was 452 msec. The maximum was 389 msec. Both considered acceptable, since the SIP timeout could not be <0,5 sec.

Even though the CPU and memory usages were increased (Fig. 6), they did not reach any prohibitive level and alert the decision module (the threshold was set 90% of the CPU and memory usage). The CPU usage peak, which occurs from the 15^{th} to the 23^{rd} sec, happens because the ASPF handles simultaneously more than 50 SIP messages/sec. The memory usage increases a few second before, because there are a lot of police files retrieved from Policy Repository. At the end, both CPU and memory usage decreased due to the rejection of the identified spitters.

Finally, all non-legitimate messages were successfully identified (2982 messages) and the overall number of lost messages was 0.

B. *Extensive Laboratory Environment*

An overview of the emulated network topology is shown in Fig. 5. It consists of the following fundamental entities:

- SER SIP servers (3). They have been customized, so that register users can redirect SIP messages and establish calls. Each of those servers creates a VoIP domain.
- ASPF platforms (7). The 2 of them have been installed to SIP server hosts, at the "logical" entrance of the VoIP infrastructure. The other 5 are placed in front of the internal clients. The first 2 ASPFs include all the ASPF modules (extensive version). The remaining 5 (protecting internal clients) include only the users' policy and do not include modules for adaptive management (client version), due to client computational limitations.
- Soft phone clients (9). These clients are active, i.e., they are ready to receive calls. The soft-phones are twinkle open-source soft-phones [44]. The calls are initiated by using SIPp [45].

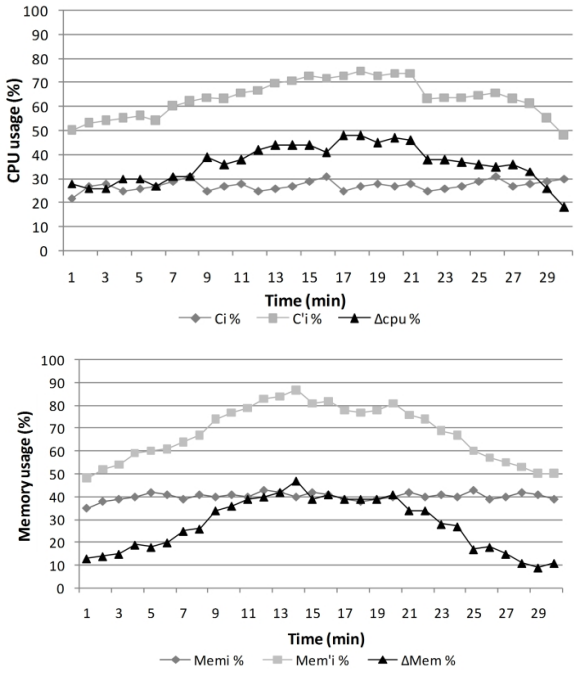

Figure 6. CPU and Memory Usage

For the purpose of the evaluation, we utilized a network emulator called modelnet. By using it we developed a realistic topology consisting of the following network elements: 3 hosts of SER servers, 9 VoIP nodes that host one IP phone each, and 3 access routers. The network link between each client and the respective access routers supports data rates of 10Mbps. The backbone links between the SER servers support data rates of 54Mbps.

This network topology has been e-mulated using a core emulator (running FreeBSD 4.11) and 3 desktop PC, in which we have distributed the 12 virtual hosts (3 servers and 9 IP clients). Summarizing, each physical host ran 1 server and 3-5 virtual nodes.

2. *Proposed Scenario and Results*

We tested the ASPF in a realistic environment. ASPF should demonstrate the ability to manage an increased flow of calls without abating its functionality and affecting the availability of the other IP services (e.g., ordinary file transfer). In order to monitor and record the behavior of our platform, in relationship with other IP services, we simulated a number of ad-hoc flows of file transfers. The flows concluded to a total decrease of 40% of the network bandwidth, i.e., the SIP host where connected to 6Mbps network lines. We did not evaluate any memory or CPU usage, because the simulation tool offered no such functionality; however, as the experiment came to a normal end, we can confidently argue that the platform suffered no visible harm, as a proper VoIP server would have more resources than the simulation tool.

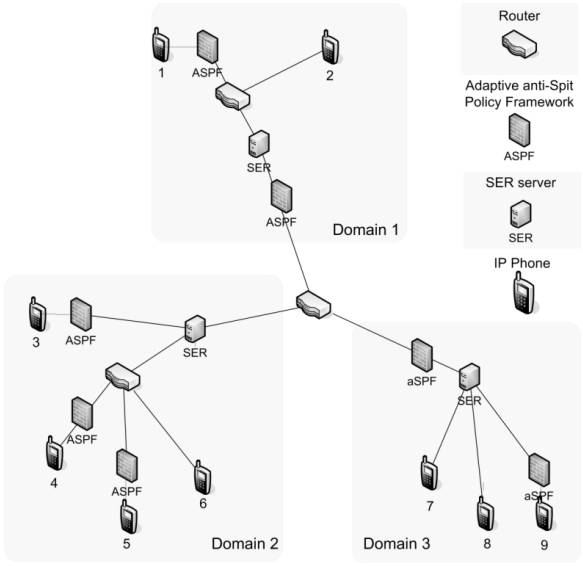

Figure 7. Modelnet topology of a VoIP infrastructure

Taking the above into consideration, we virtualized legitimate and illegitimate SIP messages. The message flows for the simulation scenario are depicted in Table 2. The 1st column shows the IP phones that initiate the calls, the 2nd and 3rd columns show the legitimate and illegitimate SIP messages initiated by each IP phone, and the 4th column shows the domain recipients of the messages. The experiment lasted 30 min (each established call lasted ~3 sec).

Table 2. Call scenario

IP Phone	Messages		Domains
	Legitimate	Non legitimate	
1	5000	0	2,3
2	1000	4000	2,3
3	5000	0	1
4	6000	0	3
5	10000	0	1,3
6	3000	7000	1,3
7	5000	0	1
8	4000	0	2
9	6000	0	1,2

Each domain handled more than 13.000 messages successfully. The ASPF platforms also handled successfully most of the messages. Moreover, the maximum processing time for a SIP message was <0,5 sec, considered acceptable, as the establishment of a VoIP call usually takes 4-6 sec [22]. In Table 3 we present the number of calls served by each ASPF, as well as how many of them were successfully forwarded or rejected.

The domain ASPF confronted every SPIT attack, which was based on SIP messages SPIT characteristics, like inappropriate text in Subject field. Moreover, both ASPF of domains 3 and 4 identified that the IP phones 2 and 6 were "spitters" because they produced too many calls/min. On the other hand, IP phones 3 and 5 received all the spit calls, because their ASPF were client versions, without being able to identify the SPIT attacks based on statistical characteristics.

Table 3. ASPF message handling

ASPFs	Messages		
	Served	Rejected	Lost
Domain 1	16895	3122	79
Domain 3	25598	6643	31
IP phone 1	7112	0	0
IP phone 3	3087	363	0
IP phone 4	2832	874	0
IP phone 5	3100	0	0
IP phone 9	7102	0	0

The only limitation observed is that 120 SIP messages were lost (from a total of 45.000 legitimate SIP messages), meaning that we lost 120 out of 15.000 calls, since it needs 3 exchanged SIP messages to establish a media session.

VII. CONCLUSIONS AND FURTHER RESEARCH

Voice Spam (SPIT) may be a considerable threat in the near future. To face it, appropriate defensive mechanisms should be introduced. In this paper, we provided the reader with a proposed architecture for policy-based management for SPIT prevention and handling, called ASPF, which is based on vulnerability analysis of the protocol SIP. In this paper, we proposed formal conditions, which are fulfilled when a SPIT attack occurs. Based on then, we developed an anti-SPIT oriented policy XML schema.

We evaluated the proposed framework, by implementing a software policy environment, and demonstrated its feasibility by building a working prototype. We showed that ASPF supports automated policy deployment and flexible event triggers, so as to permit dynamic policy configuration. Appropriate experiments were performed, so as to demonstrate the robustness, efficiency, and scalability of this architecture. Finally, we showed that ASPF is made to be easily integrated with other organization security mechanisms, like audio CAPTCHA.

A topic for further research could be the implementation of a module, which assures the scalability of the history-based policy engine by constraining past event repository size. This module could perform lookups over the entirety of the history repository. However, the fact that the periodicity of the purge process can be defined according to system processing capacity, keeps this process from significantly interfering with usual policy evaluations. Thus, two major aspects influence the relevance checking: the number of events in the History Log and the accuracy of the evaluation process. Moreover, a further input for additional conditions would be provided by Zave [46], who identified SIP security vulnerabilities through model-checking.

Our work could also enhance a distributed perspective [47], where the total control remains under the control of the network administrator, but the policies and credentials are distributed (through IPSec, or a web server, a directory-like mechanism or some other protocol) to the users and hosts in the network.

References

1. T. Walsh, D. Kuhn, *Challenges in Securing Voice over IP*, NIST, USA.
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, Vol. 2, pp. 3187-3191, April 2006.
3. J. Rosenberg, C. Jennings, *The Session Initiation Protocol (SIP) 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 the IEEE International Conference on Communications*, pp. 1979-84, UK, 2007.
5. D. Graham-Rowe, "A Sentinel to Screen Phone Calls Technology", MIT Review, 2006.
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, 2007.
7. 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*, USA, pp. 597-604, August 2007.
8. D. Gritzalis, Y. Mallios, "A SIP-based SPIT management framework", *Computers & Security*, Vol. 27, Nos. 5-6, pp.136-153, October 2008.
9. J. Rosenberg, et al, Session *Initiation Protocol* (SIP), RFC 3261, June, 2002.
10. 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.
11. 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.
12. E. Baralis, J. Widom, "An algebraic approach to static analysis of active database rules", *ACM Transactions on Database Systems*, 25(3):269-332, 2000.
13. M. Sloman, E. Lupu, "Security and management policy specification", *IEEE Network*, Special Issue on Policy-Based Networking 16(2), pp.10-19, 2002.
14. P. Gama, P. Ferreira, "Obligation policies: An enforcement platform," in *Proc. of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks*, Sweden, June 2005.
15. R. Dantu, P. Kolan, "Detecting spam in voip networks", in *Proc. of the Steps to Reducing Unwanted Traffic on the Internet Workshop*, USA, July 2005.
16. V. A. Balasubramaniyan, M. Ahamad, H. Park., "Callrank: Combating spit using call duration, social networks and global

reputation", in *Proc. of the 4th Conference on Email and Anti-Spam*, USA, August 2007.
17. R. Zhang, X. Wang, X. Yang, and X. Jiang., "Billing attacks on sip-based voip systems" in *Proc. the lst USENIX Workshop on Offensive Technologies*, USA, August 2007.
18. P. Patankar, G. Nam, G. Kesidisand, C. Das, "Exploring anti-spam models in large scale voip systems," in *Proc. of the 28th International Conference on Distributed Computing Systems*, China, June 2008.
19. D. Shin, J. Ahn, C. Shim, "Progressive multi gray-leveling: A voice spam protection algorithm", *IEEE Networks*, Vol. 20, No. 5, pp. 18-24, Sep. 2006.
20. B. Mathieu, Q. Loudier, Y. Gourhant, F. Bougant, M. Osty, "SPIT Mitigation by a Network-Level Anti-SPIT Entity", in *Proc. of the 3rd Annual VoIP Security Workshop*, Germany, June 2006.
21. Tschofenig, H., Wing, D., Schulzrinne, H., Froment, T., Dawirs, G., A document format for expressing authorization policies to tackle spam and unwanted communication for Internet Telephony (draft-tschofenig-sipping-spit-policy-02.txt).
22. Lennox, J., Wu, X., Schulzrinne, H., *Call Processing Language (CPL): A Language for User Control of Internet Telephony Services*, RFC 3880, Columbia University, October 2004.
23. 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*, J. Lopez S. Jajodia (Eds.), pp. 446-460, Springer, Spain, October 2008.
24. R. Haskins, N. Dale, *Slamming Spam: Guide for Administrators*, Addison-Wesley Professional, 2005.
25. J. Zdziarski, *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*, USA, 2005.
26. 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).
27. Cisco Systems, SIP Messages and Methods Overview (http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/rel_docs/sip_flo/preface.pdf).
28. A. Keromytis, "Voice over IP: Risks, Threats and Vulnerabilities", in *Proc. of the Cyber Infrastructure Protection Conference*, USA, June 2009.
29. V. Mehta, C. Bartzis, H. Zhu, E. Clarke, J., "Ranking Attack Graphs", in *Proc. of Recent Advances in Intrusion Detection Workshop*, pp. 127–144, Springer, Germany, September 2006.
30. P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, graph-based network vulnerability analysis", in *Proc. of the 9th ACM Conference on Computer and Communications Security*, pp. 217-224, 2002.
31. O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated generation and analysis of attack graphs", in *Proc. of the IEEE Symposium on Security and Privacy*, pp. 273-284, 2002.
32. Y. Mallios, S. Dritsas, B. Tsoumas, D. Gritzalis, "Attack modeling of SIP-oriented SPIT" in *Proc. of the 2nd International Workshop on Critical Information Infrastructure Security* (CRITIS 2007), Springer, Spain, October 2007.
33. A. Anderson, "An Introduction to the Web Services Policy Language", in *Proc. of the 5th IEEE International Workshop on Policies for Distributed Systems and Network*, 2004.
34. B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, T. Freeman, "A Multipolicy Authorization Framework for Grid Security", in *Proc. of the 5th IEEE international Symposium on Network Computing and Applications*, IEEE Computer Society, USA, pp. 269-272, July 2006.
35. L. Lymberopoulos, E. Lupu, M. Sloman, "PONDER policy implementation and validation in a CIM and differentiated services framework;" *IEEE/IFIP Network Operations and Management Symposium*, Vol. 1, pp. 31-44, April 2004.
36. A. Berglund, S. Boag, D. Chamberlin, M. Fernandez, M. Kay, J. Robie, J. Simeon, *XPath 2.0 specification*, W3C Recommendation January 2007.
37. S. Dritsas, J. Soupionis, M. Theoharidou, J. Mallios, D. Gritzalis, "SPIT Identification Criteria Implementations: Effectiveness and Lessons Learned", in *Proc. of the 23rd International Information Security Conference*, Samarati P., et al. (Eds.), pp. 381-395, Springer, Italy, September 2008
38. R. Dantu, S. Fahmy, H. Schulzrinne, J. Cangussu, "Issues and challenges in securing VoIP", *Computers & Security*, Vol. 28, No. 8, pp. 743-753, 2009.
39. Y. Soupionis, G. Tountas, D. Gritzalis, "Audio CAPTCHA for SIP-based VoIP", in *Proc. of the 24th International Information Security Conference*, pp. 25-38, D. Gritzalis, J. Lopez (Eds.), IFIP AICT 297, Springer, Cyprus, May 2009.
40. H. Schulzrinne, S. Narayanan, J. Lennox, M. Doyle, *SIPstone – Benchmarking SIP Server Performance*, Columbia University, Ubiquity, April, 2002.
41. A. Johnston, S. Donovan, R. Sparks, C. Cunningham, D. Willis, J. Rosenberg, K. Summers, H. Schulzrinne, *SIP Call Flow Examples*, IETF Internet Draft, December 2001.
42. A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, D. Becker., "Scalability and Accuracy in a Large-Scale Network Emulator," in *Proc. of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.
43. SER server version 2.0 (www.iptel.org/ser).
44. Twinkle softphone (http://www.twinklephone.com) (retrieved 25.08.2010).
45. SIPP traffic generator for the SIP protocol (http://sipp.sourceforge.net/).
46. 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, LNCS 5310, 2008.
47. S. Ioannidis, A. Keromytis, S. Bellovin, J. Smith, "Implementing a Distributed Firewall", in *Proc. of the ACM Computer and Communications Security Conference*, pp. 190-199, Greece, November 2000.