The SPHINX Enigma in Critical VoIP Infrastructures: Human or Botnet?

D. Gritzalis, Y. Soupionis
Dept. of Informatics
Athens University of Economics & Business, Athens, Greece
{dgrit, jsoup}@aueb.gr

V. Katos, I. Psaroudakis
Dept. of Electrical & Computer Engineering, Democritus University of Thrace, Xanthi, Greece
{vkatos, jpsaroud}@ee.duth.gr

P. Katsaros, A. Mentis
Dept. of Informatics
Aristotle University of Thessaloniki, Thessaloniki, Greece
{katsaros, mentis}@csd.auth.gr

***Abstract*---Sphinx was a monster in Greek mythology devouring those who could not solve her riddle. In VoIP, a new service in the role of Sphinx provides protection against SPIT (Spam over Internet Telephony) by discriminating human callers from botnets. The VoIP Sphinx tool uses audio CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) that are controlled by an anti-SPIT policy mechanism. The design of the Sphinx service has been formally verified for the absence of side-effects in the VoIP services (robustness), as well as for its DoS-resistance. We describe the principles and innovations of Sphinx, together with experimental results from pilot use cases.**

***Keywords*---SPIT; VoIP; CAPTCHA; Botnet; Security.**

I. INTRODUCTION

Voice over IP (VoIP) is an Internet telephony technology that provides a low-cost, high-quality and high-availability service of multimedia data transmission. Inevitably though, VoIP "inherits" one of the main internet security problems, namely that of Spam over Internet Telephony (SPIT) [1]. SPIT is growing into a serious issue. This is evident in recent reports published by telecommunication companies [2] and from initiatives by major organizations like NEC aiming to develop mechanisms that can tackle the problem [3]. However, there is still need for a complete and effective anti-SPIT mechanism capable to provide robust and usable protection to its users [4].

The Sphinx project focused on one of the dominant VoIP protocols known as Session Initiation Protocol (SIP), which has been found vulnerable to automated SPIT [5-7]. Sphinx developed a service that consists of (a) known efficient mechanisms such as white/black lists, (b) an anti-SPIT policy, which takes into account the user preferences, and (c) an automated Reverse Turing Test (e.g., Captcha) that tells apart the calls made by humans from those made by automated software applications (botnets) [8-10]. Sphinx design combines the mentioned countermeasures, without side-effects to the VoIP services. This has been formally verified along with the service DoS-resistance.

Sphinx development passed through the following phases:

- Design of elementary mechanisms (i.e. black/white lists) and structural units (conditions/countermeasures) for the Sphinx anti-SPIT policy [11].
- Design and implementation of a robust audio Captcha for the specific circumstances of Internet Telephony.
- Design of the policy control-flow that decides whether an incoming call is handled by the audio Captcha or some of the rest anti-SPIT mechanisms.
- Verification of Sphinx robustness by model checking possible side-effects (deadlocks, non-progress cycles etc) and desirable properties like fairness and DoS-resistance.
- Operational evaluation by running experiments over a VoIP infrastructure and tests driven by pilot use cases.

We present the service architecture, the main design principles and experimental results from pilot use cases.

II. SERVICE ARCHITECTURE & POLICY MECHANISM

Fig. 1 illustrates the Sphinx physical architecture, how the servers and end user agents are connected, as well as the call flow within our VoIP infrastructure.

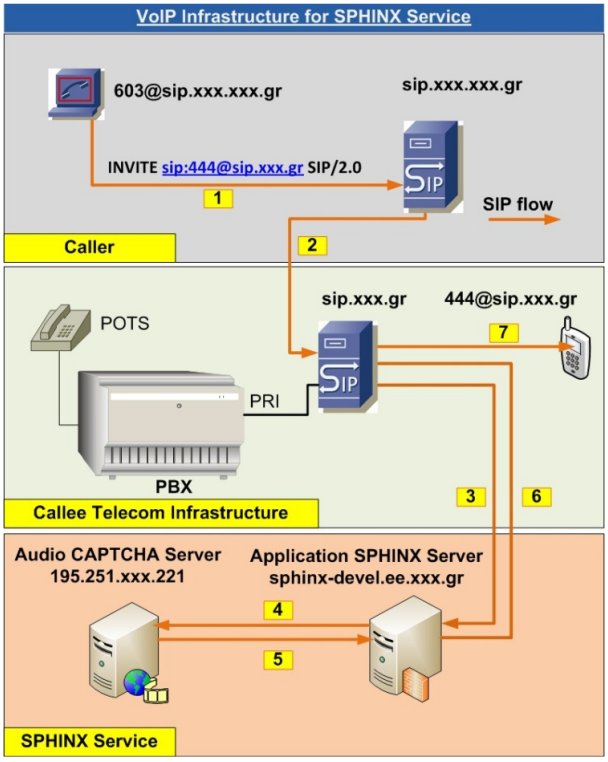

Figure 1: VoIP infrastructure for the Sphinx service

Sphinx runs over an Application Server and utilizes a separate audio Captcha Server. The protected VoIP PBX forwards call establishment requests to Sphinx. An Asterisk SIP

server was used in our VoIP infrastructure that was connected to a classic TDM PBX and a VoIP router gateway for testing purposes. On the callers' side, we set up another SIP server, in order to realize various SPIT scenarios.

Sphinx main functions provide support to manage: (a) all incoming SIP sessions that are forwarded by the Asterisk PBX server, (b) the enforced anti-SPIT policy and the operating preferences related to "bot" identification, (c) the redirection of SIP sessions to the audio Captcha server, if necessary, and (d) the user preferences, which further refine the anti-SPIT policy. Call data logs are kept for performance and incident diagnosis purposes. Sphinx was implemented as a SIP servlet over the JBoss Application Server using the Mobicent[1] communication middleware. Fig. 2 shows the distribution of the service modules over the Application and Audio Captcha servers and their runtime environment. Servers are Linux-based and afford MySQL database services, whereas the Sphinx service is provided by Apache Web Server.

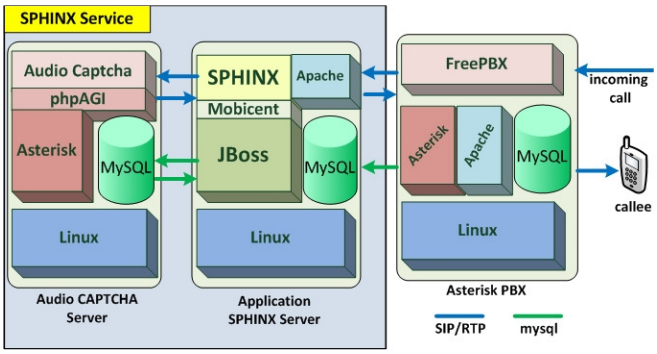

Figure 2: Sphinx service modules

The service database stores operational preferences for the audio Captcha server (Table 1), system and user policy preferences including black/white lists and logic rules, logs, and user data needed for GUI-based user authentication. MySQL's federated database engine connects the database of the Asterisk server with the one residing on Sphinx Application server, thus allowing real time database synchronization.

Table 1: Audio Captcha server operational parameters

Attribute	Value
Enable Captcha	True/False
Difficulty level	Easy/Medium/Hard
Number of concurrent sessions	Integer
Number of failed attempts	Integer
Response time (sec)	Integer
Maximum records in log file	Integer

The audio Captcha functionality may be used by different applications with Sphinx being one of them. This requirement determined the decision to implement audio Captchas as a separate service. The basic algorithm was developed using the php class Asterisk Gateway Interface (phpAGI) that interacts with the asteriskNow[2] software to provide audio Captchas as a standalone service.

[1] http://www.mobicents.org/
[2] http://www.asterisk.org/downloads/asterisknow

AsteriskNow is a widespread open source SIP server implementation and as mentioned we also used it in place of the Sphinx protected VoIP PBX. The provided API supports easy manipulation of SIP headers and allows storing useful metadata in the call-records database. The VoIP PBX runtime environment offers administration access through the FreePBX web-based application over an Apache server and includes a MySQL database that stores operational parameters, such as SIP extensions, voice trunks, call records etc.

A fundamental problem in realizing Sphinx over the described physical architecture is that the asterisk software acts as a back-to-back user agent and changes the SIP session ID every time that a call request is forwarded to an external service (numbered links in the call flow of Fig. 1 correspond to different SIP session IDs). To this end, an extra SIP Header is appended to every incoming call request, such that it can be uniquely identified over the whole duration of the Sphinx-mediated call flow. More precisely, upon receipt of a call request, the Asterisk PBX reads from the SIP headers the SIP session ID, the CallerID and caller's IP address. An additional SIP header named X-Init concatenates the three values and is then propagated by the SIP INVITE message.

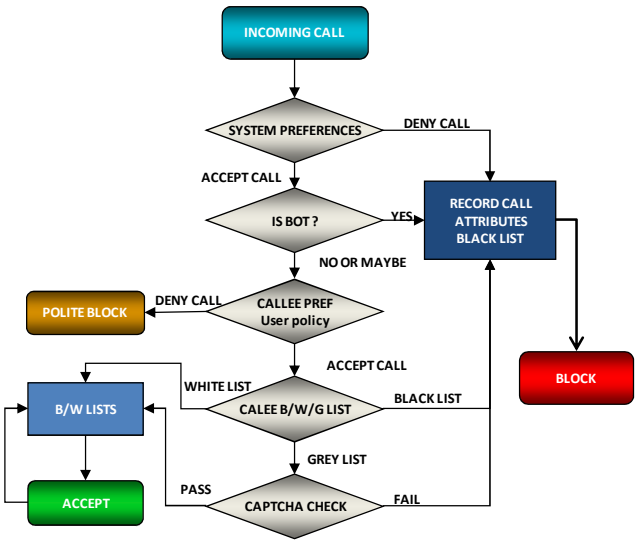

Figure 3: Sphinx policy control flow

Fig. 3 shows the applied policy control-flow. First, every call request is checked against pre-defined system preferences that may include a black list. If the caller has not been blacklisted, a series of additional characteristics are checked that could classify the call request as a bot-originated SPIT.

Table 2 provides a set of metrics that based on our experience can be used to detect calls matching characteristic bot behavioral patterns (e.g. frequent calls with short duration). The average call duration refers to the pure conversation time (from step 7, Fig. 1, call flow) and can be retrieved from the Asterisk SIP server database. For suspicious callers, we opt to temporarily revoke their access to the protected VoIP services.

Table 2: Metrics used to detect bot-originated SPIT

Attribute	Value
Number of calls per hour from same caller	Integer
Number of successive calls from same caller	Integer
Average of call duration per caller (sec)	Integer
Number of callers per minute from same caller	Integer

Call requests that have not been blocked due to system preferences are then checked against a user-defined policy. Such a policy may: (a) impose constraints like the time window in which the user accepts/does not accept calls from (particular) callers, and (b) filter call requests based on the user's black/white lists.

The Captcha test process is then triggered for call requests that have not been previously blocked or accepted (grey listed). Every such request is forwarded to the connected Audio Captcha Server in order to be processed according to operating parameters retrieved from the Sphinx database. Upon completion of the Captcha test, the call request is returned to the Sphinx Application server along with the test result that is recorded in the database. Captcha test failures cause updates of the callees' black lists, in order to block subsequent call attempts by the same callers. If a spitter keeps changing his caller ID, then he is unable to pass the Captcha test and therefore his call characteristics are recorded in order to analyze them for future call blocking. Accepted requests are returned to the Asterisk PBX for further handling. In both cases, all call metadata are logged for diagnostic purposes.

Sphinx operating parameters can be changed through a web application for the service administrator, who can also inspect the available logs for call diagnosis. Sphinx users can manage their own personal black and white lists through a separate application. Finally, they can define new user policies based on a user-friendly graphical interface.

III. Audio Captcha

Our implementation is based on audio Captcha [12]. We developed, apart from evaluating the current audio Captcha implementations, a new audio Captcha for VoIP environments. The proposed Captcha is easy for human users to solve, easy for a tester machine to generate and grade, and hard for a software bot to solve. The validation of its performance was made by two means, i.e., (a) by user tests and (b) by bots configured to solve "difficult" audio Captcha.

Based on these features, we followed an iterative algorithm: (a) we selected a set of attributes that are appropriate for audio Captcha (b) we developed a Captcha that is based on these attributes, and (c) we evaluated the Captcha by calculating the success rates of a bot and of a number of users, until the results were adequately, that is not only the bots success rate was lower than a predefined threshold but also the users' success rate was higher by a another threshold.

As both high user and low bot success rate is a key factor in deciding whether a new Captcha is effective or not, we defined a number of attributes which affect those rates. The main characteristic of these attributes is that they should all be adjusted in the production process of the Captcha.

We classified these attributes into four main categories: (a) vocabulary, (b) background noise, (c) time and (d) audio production. Each one had subcategories, such as the vocabulary and time attributes have the language requirement and total Captcha duration subcategories respectively. The only limitation this audio Captcha have is that the vocabulary should only consist of digits, as it will be used for telephony systems and there are specific phone keyboard constraints.

In order to test the produced Captcha we invited 35 users, who had a university degree and used a computer more than 20 hrs/week, and we used automated audio recognition tools. Most of the users aged between 20-30 years old and 6 persons were older than 40 years oldThe tools were a state-of-the-art open-source speech recognition tool (Sphinx) and a frequency and energy pick detection bot, called DevoiceCaptcha. The bots were selected because (a) they have a known track record for audio Captcha solving, (b) they are widely-used, and (c) they can be adapted in a VoIP environment.

Additionally, we had to integrate the Captcha server in a SIP-based VoIP infrastructure for our tests. We examined and decided that it would include three stages (Fig. 4). When the Asterisk domain receives a message (*Stage 1*), there are 3 possible scenarios, based on the policy outcome: (a) forward the message to the callee, (b) reject the message, and (c) forward the message to Captcha server.

If the INVITE message is forwarded to Captcha, then the *Stage 2* is adopted. In *Stage 2* an audio Captcha is sent to the caller by establishing a VoIP session. Lastly, the caller sends the answer, which is evaluated by the Captcha server. If it is correct the INVITE is forwarded to callee (*Stage 3*), else a new Captcha is send to caller. There is a maximum number of 3-4 retries, according to the implemented policy.

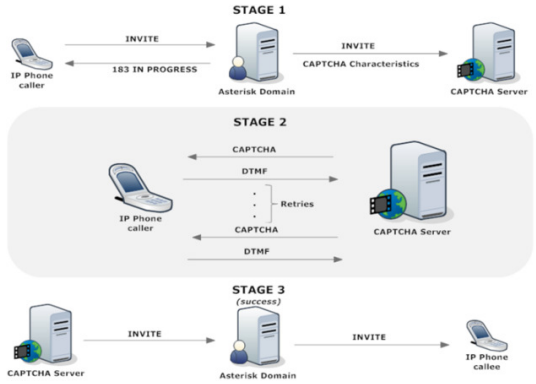

Figure 4: Captcha server integration

When the Captcha is evaluated against a bot attack, the caller was simply replaced by a bot. The bot records the audio Captcha, reforms it to an appropriate audio format (slin - Asterisk compatible) and identifies the announced digits. As soon as the bot has generates an answer, it forms a SIP message and the encoded DTMF answer using the SIPp tool. If the bot's answer is not correct then a new Captcha is sent and the bot starts to record again (*Stage 2*). The procedure depends mainly on the time needed to reform the message. Moreover, the particular bot needs approximately 0.10sec to identify a 3-digit Captcha and 0.15sec to identify a 4-digit one.

Using the above evaluation platform and appropriate number of attributes we can fully control and adjust an effective and user acceptable audio Captcha. Each attribute added strength to the Captcha and directly affected the user and bot success rates. The final Captcha had an average user success rate of 87% - each user solved 5 different Captchas), with an average bot's success rate of <1% (Fig. 5).

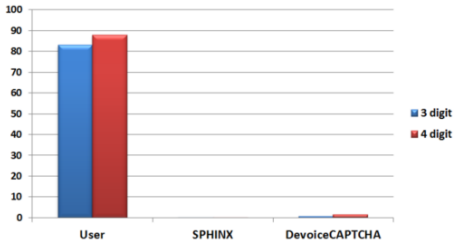

Figure 5: Proposed Captcha success rates

Based on the above attributes and the test outcomes, the characteristics of the proposed Captcha implementation are presented in Table 3. These characteristics were finalized after implementing four different failed Captcha.

Table 3: Proposed VoIP Captcha attributes

VoIP Captcha Attributes	
User success rate	88%
Background noise	music, noise
Intermediate noise	voice, music, noise
Data field	0-9
Spoken characters variation	3-4
Streaming reproduction	yes
Rare reappearance	yes
Production process	automated
Language requirements	multiple languages
Various speakers	yes
Duration (sec)	2-6
Beeps (before/after)	0/0

IV. Formal Verification

Key concerns in the design of Sphinx are its robustness and its resistance against potential DoS attacks. Robustness refers to the avoidance of side-effects in the capability of the SIP protocol to operate as expected, even in the presence of random SPIT calls and communication error messages. DoS-resistance ensures protection against malicious abuse of the Captcha mechanism that may cause exhaustion of limited server resources and in effect render the VoIP server unavailable for legitimate use.

Robustness properties that have been checked include (a) the absence of deadlocks and non-progress cycles (livelocks) in error-free communications, (b) fairness for the service users, (c) guaranteed call establishment for error-free SIP sessions, and (d) absence of message overload that could violate call establishment timeliness. All properties were formally verified by model checking a Sphinx system model developed in the SPIN toolset [13,14]. The model was parameterized based on measurements taken in our VoIP infrastructure, for the SIP message exchange times with (and without) the anti-SPIT policy. The Sphinx system model was checked in execution scenarios of parallel error-free and erroneous SIP communication sessions. At the end, the model was proved correct with respect to the formally stated properties.

The design of DoS-resistance against bandwidth abuse was guided by an evaluation of four different policies for Captcha admission control [15]. Each policy filters excessive call establishment requests, based on a bandwidth preservation criterion for authorized users; therefore, it opens a possibility to drop legitimate Captcha challenges. DoS-resistance is, thus, associated with some cost for each of the considered Captcha admission control policies.

Evaluation of the effects of bandwidth preservation, in terms of the incurred costs and achieved benefits, was based on a probabilistic system model of our Captcha mechanism under DoS attack conditions. The Continuous Time Markov Chain that was developed in the PRISM model checker[3] reflects the race for sharing the available bandwidth of a VoIP server between malicious and legitimate Captcha requests and the needs for servicing authorized users. Parameter values for bandwidth consumption were representative for the demands of the Sphinx audio Captcha, whereas the available bandwidth was set to a value that corresponds to the link capacity of our VoIP infrastructure.

All aspects of cost and benefit for the bandwidth usage [16] were taken into account by selected metrics that altogether avoid quantifying the same effects twice. Two cost metrics were used, namely: the probability to drop a call establishment request by a new client and the percentage of unused bandwidth during a DoS attack. The benefit metric used was the probability to accept a call establishment request by an authorized user.

All metrics were quantified based on reward model structures and properties that were expressed in the PRISM logic query language. Results in Fig. 6 show that threshold-based control and the cutoff policy outperform over the other schemes in terms of their net-benefit value (cost-effectiveness). As a result, Sphinx constantly monitors bandwidth usage. Unidentified call requests may be dropped if they cause exceeding of the defined threshold for efficient bandwidth usage.

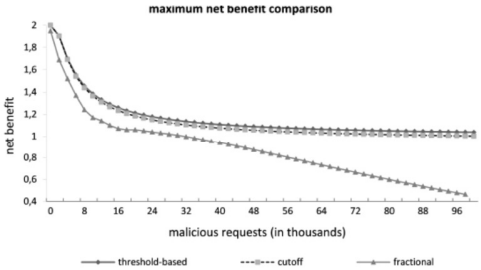

Figure 6: Comparing Captcha admission control policies

V. Use Cases & Experiments

We discuss a series of use cases from the perspective of the Sphinx-protected users. Every such user can define rules for

[3] http://www.prismmodelchecker.org/

blocking calls that violate specific constraints on their characteristic attributes (e.g. caller identifier and time). These calls are dropped without activating the Sphinx Captcha test. If a call cannot be classified as SPIT or unwanted and the caller has not been previously identified as human, then a Captcha test is submitted to the caller. There is only a limited number of tries for the caller to resolve the Captcha test and this number is adjusted by the service administrator. If the caller does not respond within a pre-defined time-span, the Captcha test ends with failure. Upon success, the call is forwarded to the Asterisk PBX, in order to establish the call session.

Captcha failures are recorded, in order to detect callers that repeatedly fail in a number of attempts. These callers can be characterized with high likelihood as bots and can be black-listed through a policy action or manually by the service administrator. On the other hand, callers who consistently pass the Captcha tests for a few times can be safely considered as non-spammers and are therefore white-listed. Identified non-spammers can then access the Sphinx-protected VoIP services in their subsequent attempts, without having to pass through the discussed checks.

Sphinx was tested for a series of protection scenarios including: (a) calls by callers that have not been previously identified as humans, (b) frequent call requests aiming to exhaust the server resources, (c) callers with characteristic bot behaviors, (d) calls from known spammers, and (e) calls that have been marked by the callee as undesirable.

Case study: Flooding attack

SIP has been found vulnerable to flooding attacks. This harmful practice is common in VoIP networks. We distinguish four types of flooding attacks against the SIP services:

Register flooding: The attacker tries to register with a SIP server using either valid or invalid user credentials. This can happen, because SIP registrars often accept connections from public IP addresses.

Call flooding: The attacker sends an excessive number of SIP Invite messages to servers that accept connections from public IP addresses.

Call control flooding: The attacker floods the server with valid or invalid call control messages (e.g., SIP INFO, NOTIFY, Re-INVITE) after the call setup.

Ping flooding (in application layer): The attacker floods the server with SIP OPTIONS messages.

The ultimate goal in the four attack options is the denial of SIP services to the honest VoIP users. In a SPIT attack, the attacker's goal is to deliver an unsolicited message to one or more VoIP service users. Similar to call flooding, the attacker can use a URI combined with a spoofed "FROM SIP" header to evade detection. In the sequel, we report experimental results for the following attack vector:

Step 1: Reconnaissance. First, the attacker tries to discover an open SIP server that accepts SIP Invite messages from public IP addresses. This is possible by using scanning tools such as the well-known *nmap* or the *smap* tool.

Step 2: Enumeration. The attacker finds the numeric range of addresses served by the target SIP domain server, i.e. all valid URI. This may be possible through advanced social engineering techniques or by gathering information from web sites, social media and so on.

Step 3: Launching SPIT. The attacker is ready to launch a SPIT attack based on the same tools used for call flooding. The tool creates a single SIP Invite message with every valid URI and supports the playback of a previously recorded audio spam message. To this end, we used Sipp, a VoIP penetration tool and SIP traffic generator that produced the attack workload for a pool of spoofed Caller IDs.

Sphinx was successful in countering the discussed attack vector, due to the combined action of our anti-SPIT policy with the audio Captcha tests. Malicious calls are blocked before reaching the callees, because the Captcha challenges cannot be solved by the attacker. But even when the attacker backs off before reaching the allowed "Number of failed attempts" in order to avoid the blacklist he cannot bypass the Sphinx anti-SPIT protection. In this case, if the call attempts reach the metric value "number of successive calls from same caller", all subsequent calls by the same Caller ID are directly dropped. Table 4 is a part of the Sphinx service log, for "number of failed attempts" (captcha tries) equal to 3 and "number of successive calls from same caller" equal 5.

Table 4: Attack logs (Step 3)

Call #	Caller ID	Callee	Outcome	Start time	Finish time	Captcha tries	Captcha outcome	Black listed
2993	spiter@ 127.0.1.1	409	SYSTEM_CALLER_MAX_SERIAL_CALLS_EXCEEDED	2012-12-31 13:17:52.0	-	0	NO_CAPTCHA	NO
2992	spiter@ 127.0.1.1	408	SYSTEM_CALLER_MAX_SERIAL_CALLS_EXCEEDED	2012-12-31 13:17:39.0	-	0	NO_CAPTCHA	NO
2991	spiter@ 127.0.1.1	407	SYSTEM_CALLER_MAX_SERIAL_CALLS_EXCEEDED	2012-12-31 13:17:26.0	-	0	NO_CAPTCHA	NO
2990	spiter@ 127.0.1.1	406	SYSTEM_CALLER_MAX_SERIAL_CALLS_EXCEEDED	2012-12-31 13:17:12.0	-	0	NO_CAPTCHA	NO
2989	spiter@ 127.0.1.1	405	IN_SYSTEM_WHITELIST	2012-12-31 13:16:39.0	2012-12-3 13:17:10.(	2	CAPTCHA_FAIL	NO
2988	spiter@ 127.0.1.1	404	IN_SYSTEM_WHITELIST	2012-12-31 13:16:07.0	2012-12-3 13:16:37.(	2	CAPTCHA_FAIL	NO
2987	spiter@ 127.0.1.1	403	IN_SYSTEM_WHITELIST	2012-12-31 13:15:36.0	2012-12-3 13:16:05.(	2	CAPTCHA_FAIL	NO
2986	spiter@ 127.0.1.1	402	IN_SYSTEM_WHITELIST	2012-12-31 13:15:03.0	2012-12-3 13:15:33.(	2	CAPTCHA_FAIL	NO
2985	spiter@ 127.0.1.1	401	IN_SYSTEM_WHITELIST	2012-12-31 13:14:31.0	2012-12-3 13:15:01.(	2	CAPTCHA_FAIL	NO

Step 4: SPIT with different spoofed Caller ID. The attacker now attempts to bypass Sphinx anti-SPIT protection by spoofing the Caller ID for every SIP Invite message. Though he can avoid blocking due to the system policy [12, 17,18], he still fails to solve the audio Captcha test. Moreover, an excessive number of failed Captcha will trigger informing warnings to the service administrator, in order to further investigate it and to apply countermeasures such as

blocking the offending IP. The use of an authentication mechanism is not advisable, since it adds not only significant labor to install it to each participating entity but also important overload to identify each message.

Table 5: Attack logs (Step 4)

Call #	Caller ID	Callee	Outcome	Start time	Finish time	Captcha tries	Captcha outcome	Black listed
3038	spitter410@127.0.1.1	410	IN_SYSTEM_BLACKLIST	2012-12-31 13:36:51.0	2012-12-31 13:37:20.0	3	CAPTCHA_FAIL	NO
3037	spitter409@127.0.1.1	409	IN_SYSTEM_BLACKLIST	2012-12-31 13:36:18.0	2012-12-31 13:36:49.0	3	CAPTCHA_FAIL	NO
3036	spitter408@127.0.1.1	408	IN_SYSTEM_BLACKLIST	2012-12-31 13:35:47.0	2012-12-31 13:36:16.0	3	CAPTCHA_FAIL	NO
3035	spitter407@127.0.1.1	407	IN_SYSTEM_BLACKLIST	2012-12-31 13:35:14.0	2012-12-31 13:35:45.0	3	CAPTCHA_FAIL	NO
3034	spitter406@127.0.1.1	406	IN_SYSTEM_BLACKLIST	2012-12-31 13:34:43.0	2012-12-31 13:35:12.0	3	CAPTCHA_FAIL	NO
3033	spitter405@127.0.1.1	405	IN_SYSTEM_BLACKLIST	2012-12-31 13:34:11.0	2012-12-31 13:34:40.0	3	CAPTCHA_FAIL	NO
3032	spitter404@127.0.1.1	404	IN_SYSTEM_BLACKLIST	2012-12-31 13:33:38.0	2012-12-31 13:34:08.0	3	CAPTCHA_FAIL	NO
3031	spitter403@127.0.1.1	403	IN_SYSTEM_BLACKLIST	2012-12-31 13:33:07.0	2012-12-31 13:33:36.0	3	CAPTCHA_FAIL	NO
3030	spitter402@127.0.1.1	402	IN_SYSTEM_BLACKLIST	2012-12-31 13:32:34.0	2012-12-31 13:33:04.0	3	CAPTCHA_FAIL	NO
3029	spitter405@127.0.1.1	401	IN_SYSTEM_BLACKLIST	2012-12-31 13:32:02.0	2012-12-31 13:32:32.0	3	CAPTCHA_FAIL	NO

VI. Concluding Remarks

As with every security control introduced in an ICT infrastructure, Sphinx aims to manage a tradeoff between user protection and user acceptance with regards to the security technology and service in question. Provided that SPIT is an ever increasing phenomenon due to the reduced telephony costs and improved level of sophistication of the SPIT bots, the perceived added value of VoIP communications may diminish. As such, a systematic treatment of the SPIT problem like the one pursued by Sphinx is expected to become popular in the VoIP domain.

Understanding that telephony is reaching a large number of users with different competencies, needs and cognitive skills, any anti-SPIT mechanism must be effective whilst respecting the user's needs and requirements. Sphinx employs a number of different approaches and our tests show that in order to realistically protect users from automated SPIT bots the protection mechanism should have an adaptable and customizable security policy, as in the opposite case the security mechanism will lead the system to a degenerated state, promoting frustration and exclusion of users.

Acknowledgments

The Sphinx project is co-financed by the European Regional Development Fund and national funds, through the Greek Ministry of Education (Operational Programme "Competitiveness & Entrepreneurship II", Measure "Cooperation").

References

[1] Walsh T., Kuhn D., "Challenges in Securing Voice over IP", *IEEE Security & Privacy*, pp. 44-49, May/June 2005.

[2] Cisco Annual Security Report, http://www.cisco.com/en/US/ prod/collateral/vpndevc/security_annual_report_2011.pdf, 2011.

[3] Seedorf J., d'Heureuse N., Niccolini S., Ewald T., "VoIP SEAL: A research prototype for protecting Voice-over-IP networks and users", in Konferenzband der 4. *Jahrestagung des Fachbereichs Sicherheit der Gesellschaft f`ur Informatik*, April 2008.

[4] Keromytis A., "A Comprehensive Survey of Voice over IP Security Research", *IEEE Communications Surveys & Tutorials*, Vol. 14, No. 2, pp. 514-537, 2012.

[5] Gritzalis D., Marias G., Rebahi Y., Soupionis Y., Ehlert S., "SPIDER: A platform for managing SIP-based spam over Internet Telephony", *Journal of Computer Security*, Vol. 19, No. 5, pp. 835-867, 2011.

[6] Keromytis A., "Voice-over-IP Security: Research and Practice", *IEEE Security and Privacy*, Vol. 8, No. 2, pp. 76-78, 2010.

[7] Rosenberg J., Jennings C., *The Session Initiation Protocol (SIP) and Spam*, Network Working Group, RFC 5039, January 2008.

[8] Bursztein E., Bethard S., Fabry C., Mitchell J., Jurafsky D., "How good are humans at solving CAPTCHA?", in *Proc. of the 2010 IEEE Symposium on Security and Privacy*, pp. 399-413, USA, 2010.

[9] Bigham J., Cavender A., "Evaluating existing audio CAPTCHA optimized for non-visual use", in *Proc. of the ACM Conference on Human Factors in Systems*, pp. 1829-38, ACM Press, USA, 2009.

[10] Markkola A., Lindqvist J., "Accessible voice CAPTCHA for Internet telephony", in *Proc. of the 2008 Symposium on Accessible Privacy and Security*, ACM Press, USA, 2008.

[11] Soupionis Y., Dritsas S., Gritzalis D., "An adaptive policy-based approach to SPIT management", in *Proc. of the 13th European Symposium on Research in Computer Security*, pp. 446-460, Springer, 2008.

[12] Soupionis Y., Gritzalis D., "Audio CAPTCHA: Existing solutions assessment and a new implementation for VoIP telephony", *Computers & Security*, Vol. 29, No. 5, pp. 603-618, 2010.

[13] Gritzalis D., Katsaros P., Basagiannis S., Soupionis Y., "Formal analysis for robust anti-SPIT protection using model-checking", *Int. Journal of Information Security*, Vol. 11, No. 2, pp. 121-135, 2012.

[14] Soupionis Y., Basagiannis S., Katsaros P., Gritzalis D., "A formally verified mechanism for countering SPIT", in *Proc. of the 5th International Conference on Critical Information Infrastructure Security*, pp.128-139, Springer (LNCS-6712), Greece, 2010.

[15] Stachtiari E., Soupionis Y., Katsaros P., Mentis A., Gritzalis D., "Probabilistic model checking of CAPTCHA admission control for DoS resistant anti-SPIT protection", in *Proc. of the 7th Int. Conference on Critical Information Infrastructures Security*, Springer, 2012.

[16] Deshpande, T., Katsaros, P., Basagiannis, S., Smolka, S. "Formal analysis of the DNS bandwidth amplification attack and its countermeasures using probabilistic model checking", in *Proc. of the 13th IEEE Int. High Assurance Systems Engineering Symposium*, pp. 360-367, USA, 2011.

[17] Soupionis Y., Gritzalis D., ASPF: An adaptive anti-SPIT policy-based framework, in *Proc. of the 6th International Conference on Availability, Reliability and Security*, pp. 153-160, Springer, Austria, 2011.

[18] Tam J., Huggins-Daines J., von Ahn L., Blum M., "Improving audio CAPTCHAs", in *Proc. of the 2008 Symposium on Accessible Privacy and Security*, ACM Press, USA, 2008.