

Application of Gaussian Estimation for Devising Reliable Vulnerability Assessment on SIP-based VoIP Infrastructure

Mohammad Z Chowdhury¹, Bon K. Sy^{1,2}, Rasheed Ahmad¹

¹Queens College/CUNY
Computer Science Department
65-30 Kissena Blvd.
Flushing NY 11367
U.S.A.

²Graduate Center/CUNY
Computer Science Department
365 Fifth Ave
New York NY 10016
U.S.A.

emon1928@gmail.com, bon@cs.qc.edu, rasheed790@hotmail.com

Abstract

The Session Initiation Protocol (SIP) is an application-layer protocol commonly used in VoIP for communication over the Internet. In this paper we describe a method to assess a particular kind of vulnerability of SIP implementation so that we could gain insights into its robustness. We used PROTOS as a tool to carry out exception element injection into SIP INVITE method and test it against a SIP implementation in our lab. The purpose of this test is to evaluate implementation level security and robustness of Session Initialization Protocol (SIP). During our test, we noticed that the choice of “wait time” could significantly change the outcome of the assessment. Yet exhaustive trials for finding the optimal wait time are too laborious and expensive. Consequently, we adopted a Gaussian estimation approach for finding the appropriate choice of “wait time.” In this paper we describe the estimation approach and how it is applied to obtain reliable experimental results.

1. Introduction

Like any other system, SIP-based VoIP systems are vulnerable to general attacks. When vulnerabilities exist, there are serious security implications when such vulnerabilities are exploited. For example, these security implications [1] could range from privacy compromise via eavesdropping, service disruption caused by the Denial of Service attack, to unauthorized privileged super-user access during the core dump of the SIP-based VoIP process initiated by the root super-user.

Session Initiation Protocol (SIP) is an application-layer protocol used for end-to-end signaling in Internet communication [3] such as VoIP. It establishes, manages, and terminates multimedia sessions or conferences. SIP supports five aspects of such sessions. These are user location, user availability, user capabilities, session setup,

and session management. SIP [4] is emerging as the standard protocol to provide telephony and enhanced services over the Web.

Currently there are many SIP testing tools available but most of them focus only on the performance and not the robustness and vulnerabilities. Following is an exemplary list of some of the available SIP testing tools: TTsuit-SIP [5], SIP Torture testing tool [6], TTCN-3 for SIP [7], SiVuS - The VoIP Vulnerability Scanner [8], and SIP Forum Test Framework (SFTF) [9].

As typical to any enterprise environment, City University of New York (CUNY) would have to understand the security implication if CUNY is to deploy VoIP. As such, we are interested in the assessment of (1) the robustness of SIP implementation, and (2) possible implementation level security leaks. At the same time, we recognize the scope of such an assessment could be very board. In this research, we focus on assessing only one specific aspect of SIP signaling; namely, injection of exception elements into the SIP signaling message header of the INVITE request. We employed PROTOS [3] to facilitate our assessment. It is our hope that this paper could contribute to the field by answering the following question:

With respect to our implementation of the SIP-based VoIP infrastructure using open source IPTEL [11] — which is replicable in any enterprise environment similar to ours, what is the level of robustness and vulnerability exposure with respect to exception character injection attack on the SIP signaling message header of the INVITE request?

We argue that it is significant if we could answer the question just raised. It is because the answer to the question just raised will provide insights into whether the level of robustness and vulnerability is acceptable for others who may adopt a similar implementation.

The paper is organized as follows: we describe the SIP implementation in our environment in section 2, the underlying assumption of the vulnerability assessment that we conduct in section 3, SIP robustness test suit in section 4, the necessary details of the SIP INVITE method (related to this study) in section 5, statistical methodology used to estimate the wait time for testing, the experimental design and the results in section 6. Finally we conclude the paper by summarizing our findings, observations, and the future work that needs to be done in this area.

2. Some details on infrastructure

Our VoIP infrastructure consists of three major components; namely, SIP/RTP proxy, RADIUS server, and an Oracle database for authentication and accounting purpose. The realization of the SIP/RTP proxy is based on the SIP Express Router (SER) of the open source IPTEL. The realization of RADIUS is based on the open source RADIATOR. VoIP service is provided for authorized users. The backend database maintains the user identity for authentication purpose as well as VoIP usage accounting information.

When a SIP user agent (SIP end point) registers or requests call service, the RADIUS client integrated into the SIP proxy environment will send an authentication request and forward the user identity information to the RADIUS server. Upon the arrival of the authentication request, the RADIUS server will authenticate the user identity against the information stored in the database. If the user identity is not verified successfully, an “access rejected” message will be sent back to the RADIUS client. If the user identity is verified successfully, an “access accepted” message will be sent back to the RADIUS client. The RADIUS client will relay the corresponding message to the SIP proxy for it to take an appropriate action. If an authorization is granted as a result of “access accepted,” an accounting record of “START” will be generated by the RADIUS server to record the relevant information such as the user identity, phone number, timestamp, location information (by IP address), and accounting session ID encoding outbound/inbound phone number if it is a call INVITE request in the SIP environment. Upon service completion (signaled by a BYE/CANCEL request), an accounting record of “STOP” will be generated by the RADIUS server to record the relevant information.

3. Assumption and scope

The vulnerability assessment reported in this paper is focused on the attack due to the injection of exception elements into the SIP signaling message header of a call INVITE request. We assume the VoIP infrastructure is to serve SIP users both within and outside the firewall

perimeter. To overcome the NAT firewall traversal issue [2], the SIP proxy is coupled with a RTP proxy and is placed in the DMZ zone. To provide adequate security measure, host level firewall is enabled in the SIP proxy. Furthermore, to minimize the network delay, our infrastructure realization also assumes the SIP signaling message header transmitted in plain text without encryption. As such, attacks based on exception elements code injection are more than just for theoretical interest, but instead a real threat to a real world deployment.

In our infrastructure, an intruder who fails on authentication will not be able to register successfully. Our implementation of SIP proxy will deny call INVITE request if a SIP user agent/end point fails to register successfully. Since our assessment is focused on call INVITE request, the underlying assumption is the attacks being initiated by the authorized users. According to the report elsewhere [10], “insider” attack accounts for the majority of security exploit.

4. Test Suite – PROTONS: c07-sip

The PROTONS program was developed at the University of Oulu in Finland [3]. This tool is efficient for testing the vulnerabilities by using a black box testing. PROTONS test suit is an open source program having a simple design. Source code of the program as well as the set of injected test cases is available for reference.

The PROTONS test tool contains 4527 test cases, which are classified into 54 test groups. Every test case sends a hostile input to the SIP proxy by injecting exceptional elements in the message. The focus of this test targets at a specific protocol data unit (PDU) i.e. INVITE message including SDP.

5. Some details on SIP call INVITE request

The behavior of the SIP proxy server is dependent on the data it receives in the INVITE messages from the caller. INVITE messages contain many different header fields, which carry the session description data as well. If the message is in the correct form and contains valid information, SIP proxy server parses the request and forwards the request to the destination. But if the INVITE message received by the SIP proxy is corrupt and/or contains data which the server cannot recognize, then the SIP proxy returns an error code. This could happen for various reasons. For example, the soft-phone used by the caller may contain a bug that creates these INVITE messages in an invalid format. Incompatible versions of SIP on the soft-phone and on the proxy server could also cause a similar problem. And heavy congestion over the network or Internet can sometimes lead to the message not being reconstructed properly at the server side if any packets are lost during the transmission. If one of these or similar occurrences corrupts the INVITE message, the

SIP proxy must be capable of detecting a problem and properly handling it. Otherwise the server might crash as a result, and the system will stop functioning properly until it is restarted.

The purpose of the PROTOS Test Suite is to experiment messages with invalid INVITE method to determine the types of errors a SIP implementation can and cannot handle. In order for SIP to become main stream and be adopted for VoIP deployment, thorough tests must be conducted for discovering as many flaws as possible and useful information for fixing them. The PROTOS Test Suite is helpful in finding such flaws. But we investigate the implication of the return codes generated by the test results that could be used as a guideline for developing solutions.

6. Design and setup of experiment

6.1 Environment Specifications

SIP server: SER (0.10.99-dev5 (i386/linux)) of IPTEL
Softphone: CounterPath X-Lite 2.0
eyeBeam 1.1 3007n stamp 17816
Test Suite: c07-sip (Release 2)
Java Version: J2sdk 1.4.2_03
OS: Microsoft Windows XP

6.2 Problem discovered in initial experimentation

In our initial experimentation, all the test cases were run 3 times during different time intervals. Also the infrastructure got reserved only to run these tests so that the SIP proxy is dedicated to deal with only the requests from the experimentation. In this initial experimentation, a complete run of all 4527 test cases using the 100ms default wait time took about 6 hours.

Results of all the three test runs were compiled and we noticed that the majority of the test cases got either rejected or they never finished. Most of the cases got timed out also. Only a handful of cases finished completely. After an investigation, we discovered that the unusual SIP proxy behavior with high failure rate is caused by the improper default setting (100ms) of the “wait time” parameter in the PROTOS test suit. Consequently, there are cases in the PROTOS test suit which require more time to complete. Yet they got terminated prematurely due to the insufficient wait time period. One of the reasons that the default wait time is insufficient is the existence of a large number of test cases involving buffer overflow. Those cases involves the injection of a very long message resulting in a greater message size, thus requiring a longer processing time and a longer wait time for a response.

6.3 Minimum Time Calculation

As observed, not every test case take the same amount of time to finish. This is because the routing of the packets on the Internet is not centrally controlled. So we need to find the minimum wait time required to finish each group. Once the minimum wait time of a group is identified, all the test cases in that group will be run using that minimum wait time. This technique would give more reliable results (in comparison to that of the initial experimentation).

To calculate the minimum wait time, we take a minimum of 15% random cases from each group and run them individually and not in a batch. Following is a process, which was followed to find the minimum wait time. Test cases were run by initially specifying 100 ms wait and delay time (wait (ms) and delay (ms) parameters are available in the PROTOS test software). Every time a single test case was run, its output was observed. If the test finished completely, its time was saved. If a test did not finish in 100 ms then the same test case was run again by specifying 200 ms. Java output was observed again, if the test still did not finish then the same test case was run again by specifying 300 ms. This process was continued until the minimum time was found when the case actually got finished.

Another issue that requires attention is the rarity issue. Some of the groups contain very few test cases; e.g. “SIP-Via-Hostcolon, SDP-Proto-v-Equal, SDP-Attribute-Colon etc.” are three test groups that each has only 16 cases. 15% of 16 cases are only 2 cases. We resolve this issue by adopting the following strategy: whenever 15% of the test cases are less than ten cases, we take at least ten cases from those groups.

A total of 817 (18.05% of the 4527) random cases were run. Each case was run multiple times to find the minimum wait time. After finishing all the individual tests, now we had a range of time available within each group. We then approximate the wait time in terms of a *Gaussian distribution* for estimating the minimum wait time required to finish every individual group with reliable result. Specifically, 54 Gaussian models were derived — one for each one of the 54 groups used in this experiment.

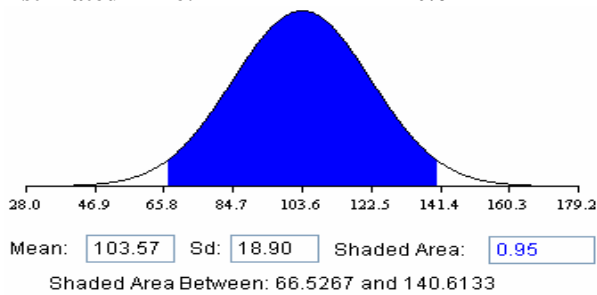
6.3.1 Application of Gaussian Distribution

Since every group has different number of test cases and every case is testing INVITE message by injecting abnormal characters at different location, we choose not to find one wait time for all 54 groups, but rather one wait time for each group. Before applying Gaussian distribution on our time data, mean and standard deviation was calculated. We use 5% significance level on determining the wait time for each one of the 54 test groups. In other words, if the data fits the Gaussian

distribution, the choice of the wait time corresponding to the 5% significance level should be sufficient to cover the wait time needed to obtain reliable results in 95% of the test cases.

Following example will help to understand the process that was followed to calculate the minimum wait time set for testing every case in the group “SIP-Method”.

Group Name: SIP-Method
Total Cases in the Group: 193
% of cases tested: 15%
Total Cases Tested: 28
Mean: 103.57
Standard Deviation: 18.90
Significance level: 5%
Estimated Time: 140.61



The above example illustrates the estimation of the wait time (140 ms) for the Group “SIP-Method” at 5% significance level. After performing the same process on the rest of all other groups, the wait time for each one of the 54 groups is derived. By applying the derived estimated time, only 1.74% (70 cases) of the total test cases got timed out.

The following table describes the detail of the time calculated against each group based on sample test cases after applying Gaussian estimation. Before applying the selected wait time for each one of the 54 test groups in any future testing, it should be mindful that the Gaussian estimation proposed in this study has not yet taken into the consideration of the variability of the network conditions such as delay, packet loss, and the distance (in terms of hops) between a java client and the SIP server.

Groups	Cases Tested	Estimated Time (ms) $1.96 * SD + Mean$
SIP-Method	28	140.61
SIP-Request-URI	10	520.07
SIP-Version	10	305.28
SIP-Via-Host	16	479.09
SIP-Via-Hostcolon	10	824.38
SIP-Via-Hostport	10	100.00
SIP-Via-Version	12	246.83
SIP-Via-Tag	10	398.78

SIP-From-Displayname	28	617.01
SIP-From-Tag	10	704.99
SIP-From-Colon	10	831.06
SIP-From-URI	10	785.67
SIP-Contact-Displayname	28	437.41
SIP-Contact-URI	10	100.00
SIP-Contact-Left-Paranthesis	10	867.83
SIP-Contact-Right-Paranthesis	10	749.31
SIP-To	28	383.96
SIP-To-Left-Paranthesis	10	606.87
SIP-To-Right-Paranthesis	10	1025.98
SIP-Call-Id-Value	23	652.38
SIP-Call-Id-At	10	1110.32
SIP-Call-Id-IP	10	587.26
SIP-Expires	10	100.00
SIP-Max-Forward	10	627.78
SIP-CSeq-Integer	10	243.96
SIP-CSeq-String	27	385.10
SIP-Content-Type	32	307.43
SIP-Content-Length	10	243.96
SIP-Request-CRLF	5	100
CRLF-Request	5	100
SDP-Attribute-CRLF	5	100
SDP-Proto-v-Identifier	25	477.62
SDP-Proto-v-Equal	10	584.74
SDP-Proto-v-Integer	10	243.96
SDP-Origin-Username	25	424.44
SDP-Origin-Sessionid	10	288.59
SDP-Origin-Networktype	26	362.98
SDP-Origin-IP	14	805.59
SDP-Session	28	630.08
SDP-Connection-Networktype	27	511.67
SDP-Connection-IP	12	307.31
SDP-Time-Start	10	100.00
SDP-Media-Media	28	457.58
SDP-Media-Port	10	202.64
SDP-Media-Transport	15	182.30
SDP-Media-Type	10	171.98
SDP-Attribute-	14	204.89

Rtpmap		
SDP-Attribute-Colon	10	714.52
SDP-Attribute-Payloadtype	10	171.98
SDP-Attribute-Encodingname	17	221.25
SDP-Attribute-Slash	10	732.32
SDP-Attribute-Clockrate	10	100.00

6.4 Results and Findings

The results of an exceptional element attack can range from denial of service to unauthorized access. Even if a particular SIP implementation successfully passes the PROTOS or any other robustness software test, it should not be assumed that the SIP implementation is free from vulnerabilities.

PROTOS is using the following criteria to grant **failed** status to a group [3]:

- A device undergoes a fatal failure and stops functioning normally.
- A process or a device crashes or hangs and needs to be restarted manually.
- A process or a device crashes and restarts automatically.
- A process consumes almost all CPU and/or memory resources for an exceptionally long or indefinite time.

According to the above criteria, our SIP implementation did not fail the test. We did not encounter any of the above conditions while testing. Also the SIP proxy was fully operational to respond after we completed all the tests.

6.4.1 Exceptional Test Cases

Every test case sent by PROTOS program receives a response in terms of “return code” from (or relayed by) the SIP proxy. The followings are the list of SIP response codes:

- 1xx – Information Responses
- 2xx – Successful Responses
- 3xx – Redirection Responses
- 4xx – Request Failures Responses
- 5xx – Server Failure Responses
- 6xx – Global Failure Responses

Careful examination of the test results revealed that PROTOS Java program did not send 329 cases to the SIP proxy. Even though the log says “Sending Case#??” but in reality these cases were never sent. These cases were run multiple times but they did not reach SIP proxy. As a result we did not receive any response code from SIP proxy for those cases.

6.4.2. Return Code

The following table explains the unique response codes received after running each test case. Details and implication of these return codes are explained in the following section.

Return Code	Description
200	No more pending branches
400	Transaction tuple incomplete
404	404 Not found
405	Method not allowed
408	Request Timeout
410	410 Gone
478	Unrecoverable destination
479	URI cannot be processed
483	Too many hops
487	Request Terminated
488	Not acceptable here
500	Internal Server Error

Due to the time constraint we could not test the following 3 test groups, SIP-From-URI, SIP-Contact-Displayname and SDP-Origin-Username. As a result our experimental study consists of only 4019 test cases out of the 4527 cases available.

The following is a percentage analysis of the output based on the return codes received from the SIP proxy:

	Return Codes Considered	Detail
Total Groups Available		54
Groups Tested		51
Total Test Cases		4527
Cases Tested		4019
1. Cases of accepted calls	200, 487	3155
% of Accepted Cases		78.50%
2. Cases of rejected calls	400, 404, 405, 408, 410, 478, 479, 483, 488	451
% of Rejected Cases		11.22%
3. Server Side Error	500	84
% of Server Side Error		2.09%
4. Lost Cases		329
% of Lost Cases		8.19%

6.4.3 Implication of Return Codes

200 ok – no more pending branches

When a SIP user agent (endpoint) sends an INVITE request to SIP proxy, it will resend the request if a response is not received by the SIP user agent by a pre-

defined wait time. When multiple requests are sent by the SIP user agent to the SIP proxy, the proxy creates unique branch for each INVITE request and tries to resolve the request. When all the requested methods are resolved (regardless of success), proxy sends '200 ok – no more pending branches' to the SIP user agent (caller) letting it know that proxy has processed all its requests.

PROTOS test tool has a parameter "wait time" with a default value 100 milliseconds. In a valid case (i.e., case 0), java program sent multiple invite requests in an interval of the pre-set wait time. If at the time the SIP proxy was processing a request and the SIP user agent (PROTOS Java program in this case) called the 'teardown' process to generate the CANCEL method, SIP proxy will respond to that request and relay the message to the SIP user agent including a "200 OK, no more pending branches." In other words, a premature teardown process triggered to cancel the pending request is not an expected outcome. Gaussian estimation introduced in this paper is an attempt to remedy this situation. The implication of this is the grade of service that is typically quantified by call loss. From security perspective, SIP implementation is vulnerable to Denial of Service attack that will degrade SIP proxy performance to the level of delay response (beyond the wait time of the SIP user agent), thus resulting in service disruption.

400 Bad requests

As defined in SIP RFC3261 [4] section 21.4.1, "400 Bad request" return code is a syntax problem. Both proxy and endpoint can throw such an exception. This return code is usually generated by the SIP user agent that receives the call. Missing/incorrect syntaxes could occur at the Content-Type, Call-ID header field, or other parts of signaling message header. In addition, this return code may also happen if unexpected contents appear in the Content-Type; i.e., application/sdp. In the case of our SIP implementation based on SIP Express Router, an error string 'Transaction tuple incomplete' will also be generated along with 400 return code and be relayed to the caller (java program) indicating that transaction is incomplete. Ideally, we would like the SIP proxy to be robust enough not to crash when 400 return code occurs, and the SIP proxy not to relay the syntactically incorrect signaling message to the end point.

405 Method Not Allowed

This return code is defined in SIP RFC3261 [4] section 21.4.6 as follows: *'The method specified in the Request-Line is understood, but not allowed for the address identified by the Request-URI'*. In our test cases, SIP-Method, most of the cases had INVITE key word missing even the rest of the message body had correct syntax. SIP proxy generated an error message but also proceeded to relay the message to the endpoint. The

endpoint (eyebeam/x-lite) parsed the message and found no key word INVITE in request-line and generated the return code "405 Method not Allowed." From proxy perspective, it shouldn't relay the message to the endpoint after parsing the message body which has missing method (key word). From endpoint's perspective, this checking mechanism is the result of RFC definition.

Few discrepancies have been observed when we ran the test using two different endpoints, i.e. xlite and eyebeam. Although these two soft phones are from the same developer, versions compatibility might be the underlying reason on the discrepancy of the different return codes.

410 Gone

As it is defined in SIP RFC3261 section 21.4.10, "The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent." Since in our test, every test case is conducted using the minimum wait time derived from the Gaussian distribution, the java program terminates itself when the minimum wait time is over. As such, the server will no longer has the location information of the java client and consequently issues *410 Gone* to other endpoint (callee).

478 Unresolvable destinations

This return code is not defined in SIP RFC. Rather, it was defined by SER. Let's first examine the meaning behind this return code. In the request-uri, user name is followed by the domain name, i.e. bob@domain.com. In SER, the SIP proxy will first try to resolve the domain name. If the resolution is successful, it will forward the message. Otherwise the SIP proxy will generate this return code and return it to the caller. The following is an example that will trigger this return code:
Invalid request-uri: bob@daldalda.adadladl.adadadadl.

479 Regretfully, we were not able to process the URI

This is also not defined in SIP RFC. Rather it was defined by SER. When the SIP proxy fails to parse the request-uri, it generates this message and sends back to the caller and terminates the transaction. Typically, missing 'SIP' keyword in the request-uri will suffice to trigger this return code. Below shows an example of an ill-formed request-uri that will trigger this return code:
Ill-formed request-uri: 492 INVITE
aaaaa:noone@sip.no.invalid SIP/2.0

483 Too Many Hops

SIP RFC, section 20.22 defines Max-Forwards as follows: *'The Max-Forwards header field must be used with any SIP method to limit the number of proxies or gateways that can forward the request to the next downstream server.... The Max-Forwards value is an*

integer in the range 0-255 indicating the remaining number of times this request message is allowed to be forwarded. This count is decremented by each server that forwards the request. The recommended initial value is 70'. When a method or request arrives at a proxy with the *Max-Forwards* value being 0, this return code will be triggered. When the maximum forward is set to be small, it is subject to premature service abortion. On the other hand, if it is set to be a large number, it is subject to the vulnerability exploit if the requested method loops back to the same proxy again and again.

487 Requests Terminated

This return string is generated by the receiving endpoint which receives a cancel request from the other endpoint — the caller. After the callee acknowledges the CANCEL request, it then generates this message. There is a large number of this return in our experiment because of the teardown process in the Java program

488 Not acceptable Here

The user agent was contacted successfully but some aspects of the session description such as the requested media, bandwidth, or addressing style were not acceptable or missing in the message body. Below is an example:

```
m=audio 9876 RTP/AVP 0
a=rtpmap:0 /8000 -- (Voice codec is missing)
```

When this return code occurs, one could anticipate problems in establishing the call connection. As such, this could be considered as SIP proxy server side errors that should be fixed.

500 I am terribly sorry, server error occurred

This is a proxy server side error. In our test cases, it happened for the test case SIP-From-Displayname, where the header field is overflowed. Although in our experiment the SIP proxy implementation did not experience a crash nor did it require restarting, this is a vulnerability exposure that should be addressed from the server side and the SIP proxy implementation

7. Conclusion

Assessment of vulnerability exposure and robustness is an essential step prior to the deployment of a production VoIP system. PROTOS test tool has a very simple design to assess vulnerabilities in a SIP implementation. But it has a limited scope i.e. it only tests INVITE messages. Identifying the test cases where the server crashed is difficult. Test cases have to be run multiple times to identify the location where server actually crashed. Test cases, which Java program did not send to SIP proxy, are also not explained in the PROTOS documentation.

Given the PROTOS software, it is useful for an initial assessment to proactively alert developers and vendors about the vulnerabilities existed in the SIP implementation. However, based on our study, additional work is needed on PROTOS test software if it is expected for use to conduct a thorough vulnerability assessment.

Finding optimum wait time to run each group is a time consuming process but application of Gaussian estimation to derive overall group wait time significantly improved the quality of the experimental test. Even though the confidence level was set to 5%, only 1.74% of the total cases fell out of range.

While the results are showing that our SIP implementation successful passed PROTOS test criteria, we believe that more work should be done to further analyze the results. In particular, we need to establish a ground reference detailing the desired expected outcome/behavior of the SIP proxy for comparison purpose. This is necessary if the goodness of the SIP implementation is to be evaluated in terms of standard figure of merits such as false positive and false negative rates. This will be the focus of our future research.

Acknowledgement: This research is supported in part under the PSC-CUNY Research Award.

8. References

- [1] "Two attacks against VoIP," Peter Thermos, April 4, 2006 (<http://www.securityfocus.com/infocus/1862/1>).
- [2] "SIP, Security, and Session Border Controller," Newport Networks Ltd., 2005 (<http://www.newport-networks.com/cust-docs/38-SIP-Security.pdf>).
- [3] PROTOS Test-Suite: C07-sip source paper <http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/>
- [4] SIP RFC 3261 documentation <http://www.faqs.org/rfcs/rfc3261.html>
- [5] Test suit for analysis of Internet system components, Voice-over IP, and SIP, the 3G signaling protocol. http://www.testingtech.de/products/voip_sip.php
- [6] Session Initiation Protocol Torture Test Messages draft-ietf-sipping-torture-tests-09 <http://www.ietf.org/internet-drafts/draft-ietf-sipping-torture-tests-09.txt>
- [7] Experiences of Using TTCN-3 for Testing SIP & OSP <http://portal.etsi.org/ptcc/downloads/TTCN3SIPOSP.pdf>
- [8] SiVuS - The VoIP Vulnerability Scanner <http://www.vopsecurity.org/html/tools.html>
- [9] SIP Forum Test Framework (SFTF)- a testing software for SIP <http://www.sipfoundry.org/sftf/index.html>
- [10] CERT Coordination Center <http://www.cert.org/>.
- [11] Internet Protocol Telephone <http://www.iptel.org/>