SIP Robustness Testing for Large-Scale Use

Christian Wieser, Marko Laakso

Department of Electrical and Information Engineering
University of Oulu
Oulu, FIN-90014 University of Oulu
ouspg@ee.oulu.fi

Henning Schulzrinne

Department of Computer Science
Columbia University
1214 Amsterdam Avenue, New York, NY - 10027-7003
hgs@cs.columbia.edu

Abstract: The Session Initiation Protocol (SIP) is a signaling protocol for Internet telephony, multimedia conferencing and instant messaging. We describe a method for assessing the robustness of SIP implementation by means of a tool that detects vulnerabilities. We prepared the test material and carried out the tests against a sample set of existing implementations. Many of the implementations available failed to perform in a robust manner under the test. Some failures had information security implications and should hence be considered as vulnerabilities. The results were reported to the respective vendors and, after a grace period, the test suite is now publicly available. By releasing the test material to the public, we hope to contribute to more robust SIP implementations.

1 Introduction

Vulnerabilities in software that endanger confidentiality, integrity and accessibility (information security) of information systems are discovered daily [SAN1 , CER02]. Some vulnerabilities are due to non-robust handling of input data. Such robustness problems can be identified by blackbox syntax testing [KLA00]. New software is still being developed in a fashion that allows such bugs to be deployed to the marketplace. In this paper, we report tests of the robustness of Session Initiation Protocol (SIP) implementations. SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences), such as Internet telephony calls [RSC$^+$02]. We chose to study SIP because it has matured from an academic interest into a commercially relevant protocol with potential for wide deployment. Furthermore, SIP is being adopted by the Third Generation Partnership Project (3GPP) as part of the third generation mobile architecture [3GP03]. SIP will be a link between rather closed telephony systems and the more open Internet Protocol-based environment. The SIP family of specifications is expanding, and some aspects are still under development.

A fair amount of work has been done in the area of conformance and functional testing of SIP implementations. A list of available tools can be found at [SIP03b]. NIST [RDM03] (using XML) and ETSI [ETS03] (using TTCN-3) have also released material for conformance and functional testing. Performance and stress testing tools are also available, e.g. SIPstone [SNLD02]. SIP interoperability testing events [SIP03a] have been arranged since April 1999. During these biannual events, messages that were causing failures were noted down. Some of these messages are valid, some are not. They are available as an IETF draft [SHJS04]. Fuzz [MFS90] was the first widely known robustness testing tool that injected random data via interfaces. Further work in this area is described in [Kak01].

No compherensive effort to produce robustness testing (in the spirit of Fuzz) material for SIP implementations has been made. Moreover, the HTTP-like ASCII presentation of SIP messages may also attract more script-kiddie level hostility than rival protocols - such as H.323. H.323 uses binary (ASN.1) encodings.

The purpose of this paper is to describe a method for assessing the robustness of SIP implementations. A test suite that aims to set a baseline for determining the robustness of SIP products was developed.

First, we describe the design of the SIP robustness test suite. The results of applying the test suite during in-house testing are also given. Second, we discuss the process of reporting our findings. Finally, the paper is concluded by a discussion of the findings.

2 Test Suite: c07-sip

The framework for creating robustness test suites was developed in the PROTOS project [PROc]. One result was a mini-simulation method [Kak01] and its implementation in a test design and generation tool. This tool has proven to be efficient in testing for vulnerabilities by using a black-box testing method based on syntax testing. In syntax testing, test cases are generated from the input specification and injected via interfaces [Bei90, p.284].

In contrast to conformance or functional testing, we are not evaluating the reply to an input, i.e. we do not have a test oracle. We monitor the implementation when it is fed with exceptional input.

The rest of this section presents the steps to create the test suite and the tests we conducted against the available SIP implementations.

```
INVITE sip:UserB@biloxi.com SIP/2.0
Via:  SIP/2.0/UDP client.atlanta.com:5060;branch=z9hG4bK74bf9
Max-Forwards:  70
Expires:  3600
From: BigGuy  <sip:UserA@atlanta.com>;tag=9fxced76sl
To:  LittleGuy <sip:UserB@biloxi.com>
Call-ID: 3848276298220188511@atlanta.com
CSeq:  1 INVITE
Contact:  BigGuy <sip:UserA@client.atlanta.com>
Content-Type:application/sdp
Content-Length:  143

v=0
o=UserA 2890844526 2890844526 IN  IP4  client.atlanta.com
s=-
c=IN  IP4  192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Figure 1: SIP-INVITE(based on [JDS+03]) Request with SDP Content

2.1 Standard Survey

RFC 3261 specifies the Session Initiation Protocol [RSC+02] and serves as protocol specification. Due to our interest in the voice-over-IP usage of SIP, we use SDP [HJ98] to describe a multimedia session, although no actual multimedia content is transmitted during the test.

We chose a subset of SIP, namely SIP-INVITE requests, as the topic of our test suite. A SIP-INVITE request is shown in Figure 2.1. All highlighted items are replaced by anomalies during the execution of the test suite.

2.2 Subject Survey

A survey of available implementations was conducted. This included a diverse selection of implementations to gain better insight into the applications implementing the protocol and to get some idea of the impact of potential vulnerabilities.

RFC 3261 identifies several types of SIP entities: the user agent, e.g. a SIP-enabled voice-over-IP phone; the redirect server, i.e. a user agent server to redirect requests; the proxy server, which acts on behalf of other entities; and the registrar, which provides location services.

A subset of implementations was chosen as a sample set to be tested during the test suite creation and pre-release phases. Altogether, we were able to acquire nine implementations, six user agents and three proxy servers.

2.3 Injection Vector Survey

In the injection vector survey, different methods of delivering the test cases to the implementation are shown in Table 2.3.

Application Protocol	Transport Protocol	Port
SIP	UDP	5060
SIP	TCP	5060
SIP-over-TLS	TCP	5061

Table 1: Injection Vector Survey

All RFC 3261 compliant SIP elements must implement UDP and TCP transports. This test suite was developed for SIP-over-UDP, although delivery over TCP would simplify the session teardown after each test case. Preference was given to UDP, however, since earlier versions of SIP (RFC 2543) required only UDP support and listed TCP support as optional. Consequently, not all implementations available to us supported SIP over TCP.

The most commonly used SIP implementations are user agent and proxy server. Both must implement the handling of SIP-INVITE messages, justifying our focus on SIP-INVITE messages in the design of the test material.

2.4 Specifications Design

Protocol data unit (PDU) specifications are used as a basis for generating the test cases. The starting point is a machine-readable representation of the protocol specification. Our test tool utilizes a custom dialect of the Backus-Naur Form (BNF). An excerpt from the PROTOS BNF to model SIP is presented in Figure 2.4.

2.5 Design of Exceptional Elements

The exceptional element is an input designed to provoke undesirable behavior in the implementation. It can violate the protocol specification, but is sometimes legal or exploits ambiguities in the specification.

Exceptional elements are also modeled as BNF. An example is given in Figure 2.5. Their design is based on the experience gathered from previous test suites and publicly reported vulnerabilities. Furthermore, we used the SIP Torture Test Messages draft [SHJS04] to

```
<SIP-Message> = {
  <Request-Line>
  <Request-Headers>
  <CRLF>
  <Message-Body> #set to SDP-format
}
<Request-Line> = {
  <Method>
  <Sp>
  <Request-URI>
  <Sp>
  <SIP-Version>
  <CRLF>
}
<Method> = "INVITE"
<Sp> = 0x20
<SIP-Version> = "2.0"
...
```

Figure 2: SIP BNF Excerpt

learn from previously discovered issues. Thus, we now focus on how well the test cases represent the external pressure against robustness rather than on the complete coverage of the input space. From the completeness point of view, we consider the test material sufficient when we detect bugs in most available implementations. A test case generally contains one anomaly, with the other elements unaltered.

Table 2.5 lists the used exceptional elements in the test material.

2.6 Design of Test Material

The test material consists of test cases simulating hostile input to the implementation. The cases are arranged into 54 test groups, each of which covers a certain part of the PDU.

2.7 Injection

Test cases were injected by a simple UDP injector. For automation of the test, connection teardown to cancel the previous INVITE requests had to be implemented. Otherwise, test execution against terminals would have required manual intervention to terminate incoming calls in each test case. Therefore, we send a CANCEL and ACK message after each test case [PROb, Appendix A]. Depending on the test case, we cannot rely on the reaction of the tested implementation. We decided to simply send a CANCEL and ACK at a fixed

```
data <ee-overflow-null> {
    () |
    0x00 #null in front
    0x00 (9x 0x61) |
    0x00 (17x 0x61) |
    0x00 (33x 0x61) |
    0x00 (63x 0x61) |
    0x00 (127x 0x61) |
...
    0x61 0x00 0x61  | #null in middle
    (9x 0x61) 0x00 (9x 0x61)  |
    (17x 0x61) 0x00 (17x 0x61)  |
...
    (1024x 0x61) 0x00  |
    (16383x 0x61) 0x00 |
    (33000x 0x61) 0x00
}
```

Figure 3: Excerpt of an Exceptional Element

Name	Description
empty	Omitted (empty) element content
ipv4-ascii	Malformed IPv4 addresses in ASCII and special purpose addresses
overflow-general	a (0x61) character overflows up to 128KB
overflow-slash	Overflows of $/$ up to 128KB
overflow-colon	Overflows of : up to 128KB
overflow-space	Overflows of $SPACE$ characters (0x20) up to 128KB
overflow-null	Overflows of a and NULs (0x00) mixed
overflow-leftbracket	Overflows of $<$ up to 128KB
overflow-rightbracket	Overflows of $>$ up to 128KB
overflow-at	Overflows of @ up to 128KB
overflow-equal	Overflows of $=$ up to 128KB
fmtstring	Format strings
utf-8	Malformed UTF-8 sequences
integer-ascii	Pos/Neg ASCII encoded integers
ansi-escape	ANSI escape sequences
sip-version	Malformed $SIP/2.0$
content-type	Malformed $application/sdp$
sip-URI	Malformed $SIP-URI$
sip-tag	Malformed $SIP-tags$
crlf	Arrangements of CR (0x0d) and LF (0x0a)

Table 2: Exceptional element categories

time point after the test case injection.

By reducing the delay, we could simulate a behavior similar to a TCP SYN flooding attack [CER96]. Furthermore, when we send a CANCEL or ACK before the tested implementation can generate a response to the initial request, we also stress the SIP state-machine. This was not our intention. We adjusted the delays for the implementation in such a way that we did not flood it with requests.

2.8 Instrumentation

The implementation is monitored for undesirable behavior that might have security implications. Instrumentation methods can be roughly divided into two categories, out-of-band and in-band.

Out-of-band instrumentation on the target platform includes debuggers, resource monitoring or custom-made tools used to extract information from the implementation. This is often the preferred form of instrumentation. For in-band instrumentation, the implementation is monitored via the injection vector, i.e., the same interface used to deliver the test cases.

For our tests, we use a variant of in-band instrumentation, called valid-case instrumentation. A valid request (valid-case), which must result in a valid reply, is sent to the subject between the test cases until a response is received. Hence, if no response from the subject is detected, it has failed.

Figure 2.8 shows an implementation passing a test case. Normally, replies are discarded. There is one exception: after sending the valid-case, the injector requires a reply from the implementation before continuing to the next test case. In Figure 2.8, the test case 'crashes' the implementation. The subsequent valid-case does not cause a response, and the injector resends the valid-case.

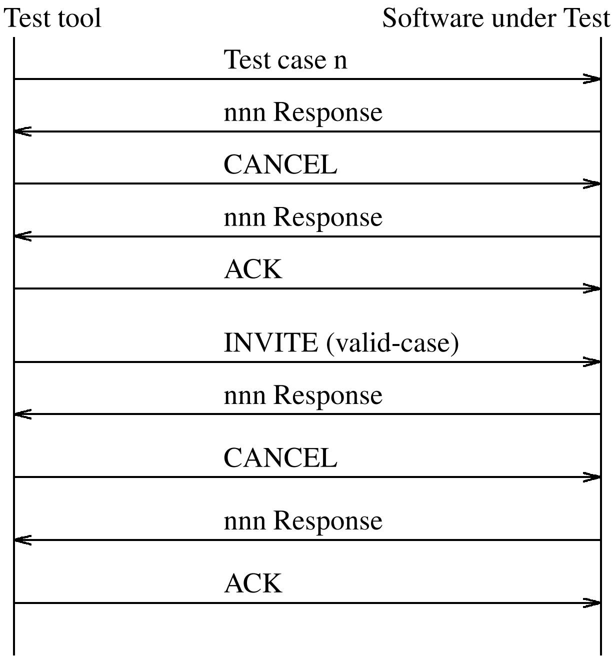
2.9 Results

The results from the test runs are given in Table 1[1]. The results are presented in a tabular form with the test cases divided into test groups. Each failed test case represents, at minimum, a denial-of-service mechanism of exploiting the vulnerability.

In Table 1, the **failed** status is given if any of the following criteria are met, and at least one test case in the test group can be identified as being responsible for one of the following failure modes.

- Software under test suffers a fatal failure and stops functioning normally.
- Software crashes or hangs and needs to be restarted manually.

[1] Product names and respective vendors of the actual implementations are - in accordance to our policy - omitted.



Legend:

- **nnn Response:** any response(s) sent by the implementation

Figure 4: Passed Test Case with Teardown

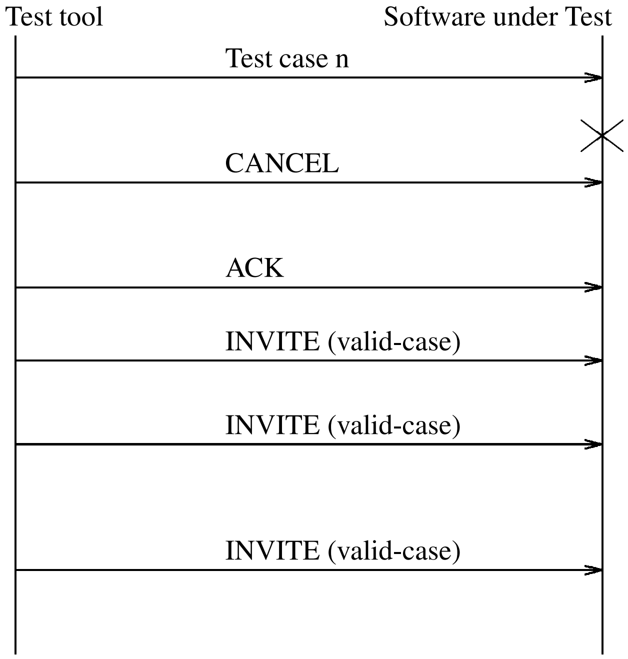


Figure 5: Failed Test Case

	Implementation								
test group	**001**	**002**	**003**	**004**	**005**	**006**	**007**	**008**	**009**
valid	-	-	-	-	-	-	-	-	-
SIP-Method	X	-	-	-	-	X	-	-	-
SIP-Request-URI	X	-	-	-	-	X	-	X	-
SIP-Version	X	-	-	-	-	X	-	-	-
SIP-Via-Host	X	-	X	-	X	X	X	X	-
SIP-Via-Hostcolon	X	-	X	-	-	-	-	-	-
SIP-Via-Hostport	X	X	X	-	-	X	-	-	-
SIP-Via-Version	X	-	X	-	-	X	-	-	-
SIP-Via-Tag	X	?	?	-	-	X	-	-	-
SIP-From-Displayname	X	X	X	-	-	X	-	-	X
SIP-From-Tag	X	X	-	-	-	X	-	-	-
SIP-From-Colon	X	X	-	-	-	X	-	X	X
SIP-From-URI	X	X	X	-	X	X	-	X	-
SIP-Contact-Displayname	X	-	-	-	-	X	-	-	-
SIP-Contact-URI	X	-	-	-	-	X	-	X	-
SIP-Contact-Left-Paranthesis	X	-	-	-	-	X	-	-	-
SIP-Contact-Right-Paranthesis	X	-	-	-	-	X	-	-	-
SIP-To	X	-	-	-	-	X	X	X	-
SIP-To-Left-Paranthesis	X	-	-	-	-	X	-	-	X
SIP-To-Right-Paranthesis	X	-	-	-	-	X	-	-	X
SIP-Call-Id-Value	X	X	?	-	-	X	-	X	-
SIP-Call-Id-At	X	X	-	-	-	X	-	-	-
SIP-Call-Id-Ip	X	X	-	-	-	X	X	X	-
SIP-Expires	X	-	-	-	-	X	-	-	-
SIP-Max-Forwards	X	X	?	-	-	X	-	-	-
SIP-Cseq-Integer	X	-	-	-	-	X	-	-	-
SIP-Cseq-String	X	X	X	-	-	X	-	-	-
SIP-Content-Type	X	-	-	-	-	X	-	-	-
SIP-Content-Length	X	X	-	-	-	X	-	X	-
SIP-Request-CRLF	X	-	-	-	X	-	-	X	-
CRLF-Request	X	-	-	-	-	-	-	-	-
SDP-Attribute-CRLF	X	-	-	-	-	-	-	-	-
SDP-Proto-v-Identifier	X	-	X	-	-	X	-	-	-
SDP-Proto-v-Equal	X	-	-	-	-	-	-	-	-
SDP-Proto-v-Integer	X	-	-	-	-	X	-	-	-
SDP-Origin-Username	X	X	-	-	-	X	-	-	-
SDP-Origin-Sessionid	X	-	-	-	-	X	-	-	-
SDP-Origin-Networktype	X	-	-	-	-	X	-	-	-
SDP-Origin-Ip	X	-	X	-	-	X	X	-	-
SDP-Session	X	-	-	-	-	X	-	-	-
SDP-Connection-Networktype	X	-	-	-	-	X	-	-	-
SDP-Connection-Ip	X	-	-	-	-	X	X	-	-
SDP-Time-Start	X	-	-	-	-	X	-	-	-
SDP-Time-Stop	-	-	-	-	-	-	-	-	-
SDP-Media-Media	X	-	-	-	-	X	-	-	-
SDP-Media-Port	X	-	-	-	-	X	-	-	-
SDP-Media-Transport	X	-	-	-	-	X	-	-	-
SDP-Media-Type	X	-	-	-	-	X	-	-	-
SDP-Attribute-Rtpmap	X	-	-	-	-	X	-	-	-
SDP-Attribute-Colon	X	-	-	-	-	-	-	-	-
SDP-Attribute-Payloadtype	X	-	-	-	-	X	-	-	-
SDP-Attribute-Encodingname	X	-	-	-	-	X	-	-	-
SDP-Attribute-Slash	X	-	-	-	-	-	-	-	-
SDP-Attribute-Clockrate	X	-	-	-	-	X	-	-	-

Legend:

- **X**: failed
- **-**: passed
- **?**: unknown

Table 3: Observed Failures for Test Groups

Category	**Test run**	**Failed test cases**	**Failed groups**
User Agent	tr-001	N	52
User Agent	tr-002	N	12
User Agent	tr-003	129	9
User Agent	tr-004	0	0
User Agent	tr-005	N	3
User Agent	tr-006	N	45
Proxy	tr-007	N	49
Proxy	tr-008	N	10
Proxy	tr-009	33	4

Legend:

- **N**: we were unable to determine the exact number of failures

Table 4: Results Summary

- Software spontaneously restarts itself.
- Software is not responding to a valid message after 16 seconds.

An implementation may occasionally become so badly corrupted that there is no way to collect accurate test results for the whole test run. This constitutes an instance of permanent denial-of-service. Untested regions are marked as **unknown**.

Otherwise, the status is **passed**. The results are further summarized in Table 1.

If the implementation fails in the majority of test groups, there is likely to be a bug in input parsing common to all header fields rather than separate bugs for handling each field. We also found that none of the embedded devices passed the test runs.

In most cases, failures represent memory corruption, stack Corruption, or other fatal error conditions. Some of these may lead to exposure to typical buffer overflow exploits, allowing running of arbitrary code [Ale96] or modification of the target system.

2.10 Limitations

Our approach has some limitations. There is the principal limitation of software testing, as pointed out by Dijkstra, that ‘Program testing can be used to show the presence of bugs, but never their absence’ [Dij69]. Our tool covers merely a miniscule portion of the input space. The provided instrumentation leaves open questions about the actual bugs behind the observed failure modes.

We are using UDP as an injection vector. If another transport protocol is used (e.g. TCP, SIP over TLS), our test results should not be invalidated, because the underlying transport protocol has to deliver the test case to the SIP implementation.

3 Reporting Process

We applied the *constructive vulnerability disclosure* model as proposed in [LTR01] to handle this multi-vendor, multi-vulnerability case. We aimed to distribute the test suite and to inform vendors on this issue. Communication with all parties was done electronically. We describe the different phases during the release of the test suite and discuss the process.

3.1 Development Phase

We described this phase in the previous section. During this phase, internal testing was conducted against the available products, and the test material and documentation were prepared.

3.2 Pre-release Phase

Next, the Computer Emergency Response Team / Coordination Center (CERT/CC), acting as the coordinator in handling this vulnerability case, was informed of our test results. The coordinator acts as a neutral third party between us, the originator, and the affected vendors. The coordinator distributed the material to individual vendors. On our part, access to the material was limited to these parties.

3.3 Release Phase

After a grace period of 121 days, the suite was released to the public [PROb]. The release followed the advisory CA-2003-06 [CER03a] by CERT/CC on such vulnerabilities.

4 Discussion

The feedback on the test suite and the release was positive, and the test suite achieved a fair amount of public and community interest. During our communication with the vendors, we got the impression that the methodology used was not widely known. While this paper describes the use of robustness testing for SIP, the method applies to all network protocols. Some vendors had not been contacted by CERT/CC before, which added extra value to the communication process as a preparatory exercise.

CERT/CC lists 92 vendors in their vulnerability note [CER03b]. 51 (55%) did not provide a vendor statement, 32 vendors (35%) said that they were not vulnerable and 9 vendors (10%) claimed to be vulnerable to the test suite. The high rate of non-responders was in line with similar cases, including SNMP [PROa] (53%). The high rate of invulnerable vendors appears to be mostly due to the CERT/CC practice of listing all contacted vendors, not just the vendors with SIP products. Therefore, we see no contradiction compared to our vulnerability rate of 88% recorded during the development phase.

Some of the vendors prepared patches during the pre-release phase. Some reacted late or not at all. By mid-February 2004, we have registered around 2,500 downloads of the material. Also, our tool was integrated into freely and commercially available testing tools.

5 Conclusion

We applied the PROTOS approach to SIP. Systematic testing for implementation level vulnerabilities showed the presence of these vulnerabilities. Only one implementation out of the sample of nine survived the test material without modifications. Vendors were informed during the pre-release phase. After a grace period, the material was made publicly available. The test suite can be used during in-house testing or as part of regression testing. It also gives consumers a possibility to compare products during and can be used as a criterion in acceptance testing. A easy-to-use tool designed for robustness testing should give a baseline for implementation level robustness, allowing implementors and their customers to find vulnerabilities pro-actively.

Acknowledgment

We wish to express our gratitude to the individual vendors who worked with us to protect their customers. We are grateful to CERT/CC for their patient help, advice, and active role during the vulnerability testing process.

References

[3GP03] 3GPP Home Page. http://www.3gpp.org/, 2003.

[Ale96] Aleph One. smashing the stack for fun and profit. http://www.shmoo.com/phrack/Phrack49/p49-14, November 1996.

[Bei90] B. Beizer. *Software Testing Techniques*. International Thomson Computer Press, 20 Park Plaza, Suite 1001, Boston, MA, 1990.

[CER96] CERT/CC advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks. http://www.cert.org/advisories/CA-1996-21.html, 1996.

[CER02] CERT/CC Statistics 1988-2002. http://www.cert.org/stats/, 2002.

[CER03a] CERT/CC. CERT Advisory CA-2003-06 Multiple vulnerabilities in implementations of the Session Initiation Protocol (SIP). http://www.cert.org/advisories/CA-2003-06.html, February 2003.

[CER03b] CERT/CC. Vulnerability Note VU#528719 - Multiple implementations of the Session Initiation Protocol (SIP) contain vulnerabilities. http://www.kb.cert.org/vuls/id/528719, February 2003.

[Dij69] Edsger W. Dijkstra. Structured programming. circulated privately, August 1969.

[ETS03] ETSI. Using TTCN to test VoIP. http://www.etsi.org/ptcc/, 2003.

[HJ98] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327, April 1998.

[JDS+03] Alan Johnston, Steve Donovan, Robert Sparks, Chris Cunningham, and Summers. Session Initiation Protocol (SIP) Basic Call Flow Examples. RFC 3665, December 2003.

[Kak01] Rauli Kaksonen. *A Functional Method for Assessing Protocol Implementation Security*. VTT Publication series, 2001.

[KLA00] Rauli Kaksonen, Marko Laakso, and AriTakanen. Vulnerability Analysis of Software Through Syntax Testing. http://www.ee.oulu.fi/research/ouspg/protos/analysis/WP2000-robustness/, 2000.

[LTR01] Marko Laakso, Ari Takanen, and Juha Röning. Introducing Constructive Vulnerability Disclosures. 2001.

[MFS90] Barton Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of UNIX Utilities. Technical report, 1990.

[PROa] PROTOS Test-Suite: c06-snmpv1. http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/snmpv1/.

[PROb] PROTOS Test-Suite: c07-sip. http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/.

[PROc] PROTOS - Security Testing of Protocol Implementations. http://www.ee.oulu.fi/research/ouspg/protos/.

[RDM03] M. Ranganathan, Olivier Deruelle, and Doug Montgomery. *Testing SIP Call Flows Using XML Protocol Templates*, chapter pp. 33-48. Lecture Notes in Computer Science. Springer-Verlag Heidelberg, Tiergartenstr. 17, D-69121 Heidelberg, Germany, April 2003.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[SAN1] SANS Security Alert Consensus - Archive. http://www.sans.org/newsletters/sac/, 2001-.

[SHJS04] R. Sparks, A. Hawrylyshen, A. Johnston, and H. Schulzrinne. Session Initiation Protocol Torture Test Messages. IETF Draft, Jannuary 2004.

[SIP03a] SiPit - SIP interoperability test event. http://www.sipit.net/, 2003.

[SIP03b] The SIP Center - Testing and Measurement. http://www.sipcenter.com/vsts/vsts_testing.html, 2003.

[SNLD02] Henning Schulzrinne, Sankaran Narayanan, Jonathan Lennox, and Michael Doyle. SIPstone - Benchmarking SIP Server Performance. http://www.sipstone.org/files/sipstone_0402.pdf, April 2002.