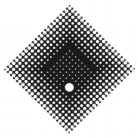

VICTORIA UNIVERSITY
MELBOURNE AUSTRALIA

The Devil is in the Detail: SDP-Driven Malformed Message Attacks and Mitigation in SIP Ecosystems

This is the Published version of the following publication

Tsiatsikas, Z, Kambourakis, G, Geneiatakis, D and Wang, Hua (2019) The Devil is in the Detail: SDP-Driven Malformed Message Attacks and Mitigation in SIP Ecosystems. IEEE Access, 7. pp. 2401-2417. ISSN 2169-3536

The publisher's official version can be found at
https://ieeexplore.ieee.org/document/8575131
Note that access to this version may require subscription.

Downloaded from VU Research Repository https://vuir.vu.edu.au/38782/

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 4, 2018, accepted November 25, 2018, date of publication December 13, 2018,
date of current version January 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2886356

The Devil is in the Detail: SDP-Driven Malformed Message Attacks and Mitigation in SIP Ecosystems

ZISIS TSIATSIKAS[1], GEORGIOS KAMBOURAKIS[1], DIMITRIS GENEIATAKIS[2], AND HUA WANG[3]

[1]Department of Information and Communication Systems Engineering, University of the Aegean, 83200 Karlovasi, Greece
[2]Digital Citizen Security Unit, Joint Research Centre, European Commission, 21023 Ispra, Italy
[3]Centre for Applied Informatics, Victoria University, Melbourne, VIC 8001, Australia

Corresponding author: Dimitris Geneiatakis (dimitrios.geneiatakis@ec.europa.eu)

ABSTRACT VoIP services, in general, and session initiation protocol (SIP) ones, in particular, continue to grow at a fast pace and have already become a key component of next-generation networks. Despite this proliferation, SIP-based services expose a large attack surface for perpetrators, especially those who seek to cause denial of service (DoS). While so far, a plethora of works in the literature have been devoted to the detection of DoS attacks in SIP ecosystems, the focus is on those which exploit SIP headers neglecting the message body. In an effort to fill this gap, this paper concentrates on the detection of DoS attacks, which, instead, capitalize on the session description protocol (SDP) part of SIP requests. To this end, we not only scrutinize this ilk of attacks and demonstrate their effect against the end-user but also develop an open-source extensible SDP parser module capable of detecting intentionally or unintentionally crafted SDP segments parasitizing in SIP requests. Following a firewall-based logic, currently, the parser incorporates 100 different rules organized in four categories (policies) based on the corresponding RFC 4566. Through extensive experimentation, we show that our scheme induces negligible overhead in terms of processing time when working as a software module in either the SIP proxy or a separate machine in front of the latter.

INDEX TERMS Covert channel, DoS, malformed messages, session description protocol, session initiation protocol.

I. INTRODUCTION

Over the last decade, Voice over IP (VoIP) services have dominated the market due to the low cost and flexibility they offer compared to the legacy Public Switched Telephony Network (PSTN) ones. Current reports indicate that until 2020 the global VoIP mobile market share will present a Compound Annual Growth Rate (CAGR) of around 28% [2]. VoIP relies on a set of core operations for delivering services. The most important of them deal with the signaling and session management including the creation, modification, and termination of a multimedia session. A handful of protocols have been proposed to handle session management in VoIP. Session Initiation Protocol (SIP) [3] seems the most prevalent due to its open nature and the flexibility and scalability it offers.

On the downside, as reported in various research works [4]–[6] and software flaw databases,[1] SIP is well-known to be susceptible to a plethora of attacks ranging from Denial of Service (DoS), SQL injection, and signaling manipulation. In a typical DoS attack, the attacker tries to paralyze the victim by either sending against it a surge of SIP requests or a number of malformed messages. In the former case, the victim is unable to serve the voluminous number of incoming requests, while in the latter the sufferer is incapable of parsing or handling properly the incoming request, and the service crashes.

The common denominator of the latter kind of attacks is the manipulation of SIP message headers by the attacker so as to hamper or preferably paralyze the parsing process at the SIP server or client. To cope with this threat, the great mass of works [7]–[10] in the literature propose some way for the SIP server to detect malformed SIP headers, and thus discard the corresponding messages outright. However, what is largely neglected is that similar attacks may take advantage of the Session Description Protocol (SDP) [1] part of a SIP message. Recall that SDP is responsible for negotiating the media information among the communicating peers, and as a result, SDP information is present in a diverse type of SIP requests

[1]Common Vulnerabilities and Exposures https://cve.mitre.org

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

and replies. Moreover, according to the literature [11], SDP can be exploited to build covert communication channels in ignorance of the SIP network. For instance, such hidden channels are typically exploited by botnets for realizing a command and control (C&C) infrastructure.

Bearing the above in mind, the goal of the work at hand is dual; first to provide proofs of the pernicious nature of this type of attacks, on real-life SIP clients and servers, and second to introduce a lightweight and flexible filtering mechanism for effectively copying with them. Our defensive solution comes in the form of a SDP parser software module either embedded in the SIP server or running in a separate machine in front of the former. In this way, the defender is able to timely detect and silently drop messages that do not fully comply with the standard [1]. Also, as a side advantage, the parser is capable of rejecting SIP requests that are found suspicious to carry information that may be part of a covert communication channel.

We evaluated our solution in terms of service time overhead using a custom made architecture, in which we simulate several scenarios involving a mix of normal and attack traffic. The results indicate that the proposed mechanism introduces negligible overhead to the processing of incoming and outgoing SIP messages. To the best of our knowledge, this is the first work that specifically focuses on SDP message manipulations. As already mentioned, this is in contrary to other works in the literature [9], [12], which solely deal with deliberate malicious manipulations in SIP message headers. The main contributions of this work can be summarized as follows:

- We study the impact of SDP malformed messages on a variety of SIP software and hardware phones, and servers.
- We offer a publicly available open-source software module capable of detecting malformed SDP messages lurking in SIP requests. Message parsing is done based on RFC [1], while the implemented software can work alongside the SIP server or independently in a separate machine.
- We extensively assess the performance of the proposed scheme in terms of service time.

The remainder of the paper is organized as follows. The next section provides an overview of SIP architecture. Section III details on the adversary model, while section IV elaborates on SDP-oriented attacks. The internal workings of the implemented SDP parser are given in section V. Section VI deals with the performance evaluation of the proposed SDP parser. The related work is given in Section VII. The last section concludes the paper and outlines future work.

II. SIP ARCHITECTURE

SIP is an application layer, text-based protocol similar to Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP). It is based on a client-server architecture responsible for delivering the necessary messages to establish and manage a multimedia session. Figure 1 depicts a typical SIP architecture. As observed from the figure, the main entities are the SIP proxy, the Registrar, and the communicating end-points, namely User Agent Client (UAC) and User Agent Server (UAS). The SIP Registrar receives REGISTER requests from the User Agents (UA) and stores their location information for routing the incoming requests to the appropriate network domain. The SIP proxy receives requests, say, for establishing a call, and forwards them to the registered UAs, assuming the latter are registered within the same domain. If not, the SIP request is forwarded to the corresponding authoritative SIP proxy.

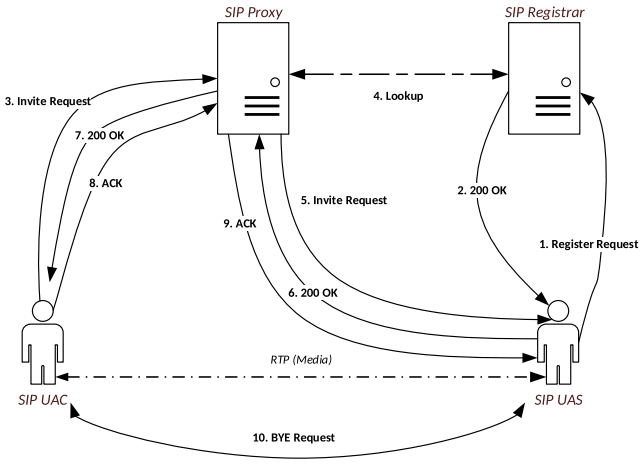

FIGURE 1. Overview of SIP architecture and a typical message flow.

The media session establishment process is initiated by the UAC (caller) sending a SIP INVITE request to the UAS (callee) via one or more SIP proxies. Upon receiving the request, the SIP proxy extracts the callee's username and queries the Registrar for obtaining the corresponding location information. Then, the SIP proxy forwards the request to the callee. After that, the session is considered established and a media protocol takes over the management of the audio and video packets between the endpoints. Real-Time Tranport Protocol (RTP) [13] has been acclaimed as the most appropriate for this purpose. At any time, either the caller or callee may terminate the media session by sending a BYE request toward the other endpoint.

SIP architecture offers a total of 88 messages for session management. Among them, 14 are used as requests and the rest as responses. A SIP message may consist of two main parts, namely the SIP message headers (the upper part of the message) and the message body carrying SDP information. Specifically, the upper part carries information regarding the method, the sender, the recipient of the message, and the communication path. The SDP part, which is the focus of this work, conveys information concerning the media of the session and it may be present in specific SIP requests and responses, including INVITE, ACK, 180 RINGING, 183 Session Progress, 200 OK. That is, with the help of SDP, SIP employs the offer/answer model [14] to establish and manage

TABLE 1. SDP descriptors. Some of the optional descriptors may be repeated across the SDP message segment.

Descriptor	Mandatory	Meaning	Descriptor format	Example of a valid value
<*v*=>	yes	Carries SDP version	v= (Protocol version)	v=0
<*o*=>	yes	Corresponds to the originator of the session	o=<username> <sess-id> <sess-version> <nettype> <addrtype> <unicast-address>	o=tzisis 2233445566 2233445577 IN IP4 192.168.1.1
<*s*=>	yes	Conveys the textual session name	s=<session name>	s=SDP seminar
<*m*=>	yes	Media description	m=<media> <port> <proto> <fmt> ...	m=audio 49170 RTP/AVP 0 8 97
<*t*=>	yes	Start and end times of a session	t=<start-time> <stop-time>	t= 1280656800 1290938400
<*i*=>	no	Provides textual information regarding the session	i=<session description>	i=Department of Information & Communication Systems Engineering
<*u*=>	no	Provides additional info about the session using a URI	u=<uri>	u=http://www.icsd.aegean.gr
<*e*=>	no	Provides a contact email address	e=<email-address>	e=tzisis at aegean.gr
<*p*=>	no	Provides a contact telephone number	p=<phone-number>	p=+30 22730 82XXX
<*c*=>	no	Provides data needed for the connection	c=<nettype> <addrtype> <connection-address>	IN IP4 192.168.1.1
<*b*=>	no	Carries information for the aggregate bandwith limit for the session	b=<bwtype>:<bandwidth>	b=AS:64
<*r*=>	no	Provides information regarding the repeat times of a session	r=<repeat interval> <active duration> <offsets from start-time>	r=7d 1h 0
<*z*=>	no	Carries information for scheduling a session	z=<adjustment time> <offset> <adjustment time> <offset>	z=2882844526 -1h 2898848070 0
<*k*=>	no	Offers information about encryption keys	k=<method>:<encryption key>	k=(base64)gAe1111fQzo4jeIdfrtsY197kV
<*a*=>	no	Used for extending SDP	a=<attribute>:<value>	a=rtpmap: 0 PCMU/8000/1

Session level

```
v=0.
o=meps 234567 234567 IN IP4 45.78.98.9.
s=-.
c=IN IP4 145.178.98.93.
t= 234567 2.
```

Media level

```
m=audio 15000 RTP/AVP 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000.
a=rtpmap:1 opus/3333333/555555555.
a=fmtp:1 usedtx=1000.
a=rtpmap:1 SILK/1 .
a=rtpmap:1 SILK/8.
a=rtpmap:1 G722/1.
a=rtpmap:1 speex/678.
a=rtpmap:1 speex/43.
a=rtpmap:33 PCMU/2.
a=rtpmap:33 PCMA/8.
a=rtpmap:33 iLBC/80.
a=rtpmap:333 GSM/90000000000.
a=rtpmap:222 speex/750.
a=rtpmap:222 telephone-event/4567.
a=extmap:11 urn:ietf:params:rtp-hdrext:csrc-audio-level.
m=video 222 RTP/AVP 4444 2222.
a=recvonly.
a=rtpmap:222 H264/4567.
a=fmtp:222 profile-level-id=11111;packetization-mode=177777.
a=imageattr:222 send * recv [x=[44444-55555],y=[66666-55555]].
a=rtpmap:222 H264/4567.
a=fmtp:222 profile-level-id=7777777.
a=imageattr:222 send * recv [x=[0-3500],y=[0-8888]].
```

FIGURE 2. The SDP part of a typical SIP INVITE request.

multimedia sessions. Figure 2 presents the SDP part of a SIP INVITE request generated by the open source SIP UA Jitsi [15]. According to the corresponding RFC [1], every SDP message can contain up to 5 mandatory and 15 optional different descriptors. Table 1 provides a succinct overview per descriptor along with a typical example.

III. ADVERSARY MODEL

For analyzing DoS attacks caused by malformed SDP segments, we consider an adversary model which includes two main types of opponents; the ones who register with the SIP service (insiders), and those who remain unregistered (outsiders). The former category typically refers to honest-but-curious parties that establish and maintain some bond of trust with the VoIP service, while the latter to malicious external adversaries, i.e., aggressors who try to attack the service from the network perimeter. Specifically, insiders are normally considered to be trusted or semi-trusted, say, they possess a SIP Uniform Resource Identifier (URI) and the matching credentials to authenticate to the local SIP registrar. On the downside, as explained in [16], trust may be the key element for launching more silent type of attacks, without violating the communication protocol or causing any obvious damage. On the other hand, external evil-doers reside outside the local network, may muster an army of attack-bots (e.g., an IP stresser and/or zombie machines), and employ techniques like IP spoofing to minimize the footprint of their attack. This is straightforward for protocols like SIP which support both TCP and UDP. Both these types of adversaries are capable of eavesdropping on SIP traffic, creating SIP messages that contain a SDP segment, injecting data into the exchanged SIP messages, and ultimately launching two basic types of attacks as illustrated in figure 3:

- DoS attacks against specific or random UAs. That is, by manipulating SDP messages, the attacker attempts to cause DoS to the corresponding SIP entity. These messages can be crafted and sent at will to specific or randomly selected users or be part of an ongoing session between a caller and a callee, where the attacker acts as a man-in-the-middle. Additionally, the attacker may send such specially crafted malformed messages to a SIP server with the aim of paralyzing its message parser.
- The creation of *"hide-in-plain-sight"* type of communication channels used to secretly convey information via SDP.

In the first case, the SDP part of the message have been intentionally crafted with the aim to crash the SDP parser at the SIP phone. This is usually accomplished using a packet

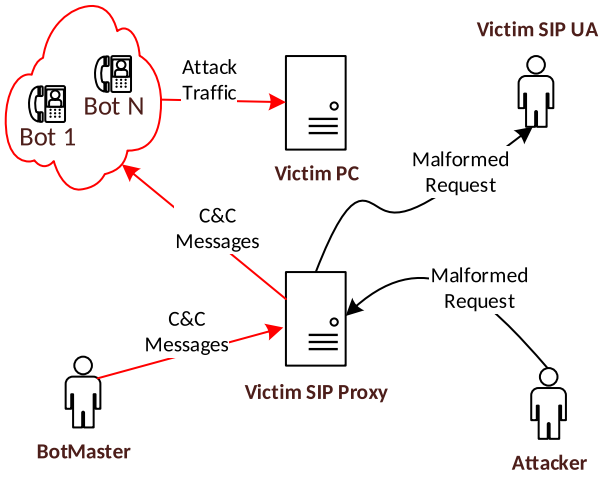

FIGURE 3. The malicious entity sends either malformed requests or (in red) communicates with bots.

manipulation program [17]. Such a tool allows for the capturing and crafting of SIP messages. However, keep in mind that a malformed message may be also unintentionally generated due to a software/hardware error. When the malformed SDP data are received by the target phone, three outcomes are possible. First, the phone may not be able to decode the SDP segment, and it will display an error message. Second, the phone will abnormally crash because for example its parser enters to an infinite loop. Lastly, the malformed data will go unnoticed. This typically happens because the malformed data lie in non-critical fields, i.e., those that do not directly affect the session establishment process. Overall, the effectiveness of this type of attack depends on which specific part (or parts) of the SDP segment have been manipulated. It is also to be noted that the same attack strategy can be used against SIP servers. That is, the attacker sends different SDP malformed messages to the SIP server in hopes of driving its parser to paralysis. Nevertheless, as discussed in the next section, at least two major SIP servers are found immune to this attack because they simply do not check the SDP part of the message (while they should). This means that the SIP server will simply forward the malformed SDP information toward its final destination without bothering to check its soundness. Of course, this negligence renders SIP clients highly prone to the same kind of attack.

As already pointed out, the second type of adversaries are assumed to use the SDP part of SIP messages to create a C&C communication channel as described in detail in [11]. Briefly, this scenario assumes that the SDP part of SIP messages are used for secretly exchanging commands with the sole purpose of coordinating a botnet. Therefore, in such a case, the attacker's goal, acting as a botmaster, is to realize a C&C without affecting the underlying infrastructure and running services. This is accomplished by malformed SDP messages that go unnoticed, i.e., do not induce an error condition on the target device.

IV. SDP-DRIVEN ATTACKS

This section details on SDP attacks targeted at both SIP phones and servers. It also offers a concrete example of a botmaster-to-bots covert communication channel realized via SDP.

A. INTRODUCTION AND DATASET

During the last few years, the industry and research communities came across different kinds of vulnerabilities pertaining to SDP [18]–[20]. These vulnerabilities have been collected and categorized in databases [21], [22], and are mostly due to software bugs in the respected products, either VoIP servers or UAs. Such weaknesses may lead to attacks ranging from DoS to the execution of malicious code. It is therefore obvious that SDP-based attacks can cause unpredictable behaviors on real-life software and/or hardware systems of everyday use.

Attacks which exploit SDP are typically performed in the context of an offer/answer media negotiation. Based on RFC 6337 [14], which describes the offer/answer model [1], [23], the SIP request messages that can contain a SDP segment are INVITE, ACK, PRACK, UPDATE. Moreover, the following messages can be used to convey SDP data in the context of a response: 2xx INVITE, 1xx-rel INVITE, 200 PRACK, 2xx UPDATE. Also, there are some messages which may bear a SDP segment, but they are not part of the offer/answer model. Such messages are for example the OPTIONS request and the corresponding 200 OK answer.

A malformed SDP segment is likely to cause DoS to the SIP server or the UA at different phases of the media negotiation process. That is, a malformed SDP segment which appears in the "offer" phase of media negotiation, may go unnoticed by the SDP parser in the SIP server. These messages target directly the end-user or they are used as vehicles to convey information in the context of a hidden communication channel. On the other hand, an attack against a SIP proxy usually happens in the context of an "answer" phase of media negotiation. This typically occurs because as detailed further down, the message parser in several SIP servers is stateful regarding the SDP parameters. This means that it tries to match the SDP parameters contained in the "offer" phase against those carried by the "answer". In case of inconsistencies, and assuming a software bug, the parser may crash. Bearing the above into mind and following the discussion from section II, Table 2 summarizes the different facets of this ilk of attack, the attacker's goals, and the attack-affected victims.

To test both the SDP data hiding and attack capacity, we created a seemingly legitimate SDP segment using only the G.711 PCMU codec. This selection appears in all messages depicted in figures 4 and 5. That is, the corresponding codec is presented in the <*m*=> line with code "0". Keep in mind that this static payload does necessarily need additional information to be decoded. In this way, we guarantee that the phones will not reject the message with a *"488 not*

TABLE 2. Summary of attack types and impact on the victim.

No	Attack Type	Attacker's Goal	Victim
1	Flooding with malformed messages	Annoy the user, crash or paralyze the device	SIP Proxy/Client
2	Malformed messages	Crash, freeze the device or execute malicious code	SIP Proxy/Client
3	Creation of covert communication channels	Convey hidden commands	Internal/External entities

MalSDP 1

```
o=tzisis 2345679993 2345679990 IN IP4 192.168.2.3
s=-
c=IN IP4 145.178.98.93
t=3303564100 3035677043
u=0 0
e=tzisis@aegean.gr
p=1111111111111
m=video 1024 RTP/AVP 97 98 5 0
a=rtpmap:97 opus/48000/2
a=rtpmap:98 H261/90000
a=rtpmap:2 H261/8000
```

MalSDP 2

```
v=0
o=tzisis 2345679990 2345679993 IN IP4 192.168.2.3
s=-
c=IN IP4 145.178.98.93
t=3303564100 3303564101
u=0 0
e=tzisis@aegean.gr
p=1111111111111
m=video 1024 RANDOM/AVP 97 98 5 0
a=rtpmap:97 opus/48000/2
a=rtpmap:98 H261/8000
m=video 1024 RTP/AVP 97 98 5 0
a=rtpmap:97 opus/48000/2
a=rtpmap:98 H261/90000
```

MalSDP 3

```
v=0
o=tzisis 2345679990 2345679993 IN IP4 192.168.2.3
s=-
c=IN IP4 145.178.98.93
t=3303564100 33035641013
u=0 0
e=tzisis@aegean.gr
p=1111111111111
m=video 1024 RTP/AVP 1 2 5 0
a=rtpmap:97 opus/48000/2
a=rtpmap:98 H261/90000
```

MalSDP 4

```
v=0
o=tzisis 2345679990 23456 IN IP4 192.168.2.3
s=-
c=IN IP4 145.178.98.93
t=3303564100 3303564101
u=0 0
e=tzisis@aegean.gr
p=1111111111111
m=video 1024 RTP/AVP 97 98 5 0
a=rtpmap:97 opus/48000/2
a=rtpmap:98 H261/90000
```

MalSDP 5

```
v=0
o=tzisis 2345679990 2345679993 IN IP4 192.168.2.3
s=-
c=IN IP4 145.178.98.93
t=3303564100 3303564101
u=0 0
e=tzisis@aegean.gr
p=1111111111111
k=(base64)gAe>=?#fQzo4jel.:](:-)97kV
m=video 1024 RTP/AVP 97 98 5 0
a=rtpmap:97 opus/48000/2
a=rtpmap:98 H261/90000
```

MalSDP 6

```
v=0111111111111111111111111111
o=TZISIS- 131479891221829251 IN IP4
192.168.10.98.

s=@#$%$#@$$%%#@#$%$#$%$#$%$##%%$#$$#@
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
X-Lite release 4.9.8 stamp 84253
c=IN IP4 192.168.10.98
t=0
1234560000000000000000000000000000000000000
0000000000000000000000000000000000000000000
m=audio 11111111111 RTP/AVP 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 4 9 8 1200 0 84 101
```

FIGURE 4. Malformed SDP bodies relayed from Kamailio (The malformed part is shown in red font).

acceptable here" response. After that, as in [11], we crafted specific parts of the SDP segment at will. Using this approach, we have created a dataset of 12 representative malformed SDP message bodies depicted in figures 4 and 5. Note that in the course of our experiments, we also tested many more SDP malformed messages and cherry-picked those included in the aforementioned figures. To our knowledge, no such dataset containing a rich set of SDP malformed messages exists, therefore the only option was to create one by crafting SDP messages and performing an error-trial approach to observe the consequences on the various phones and SIP servers.

All these 12 SDP bodies either miss or contain specific pieces of information, which directly or indirectly violate RFCs 4566 and 3551. More specifically, *MalSDP1* lacks the mandatory *<v=>* descriptor. Moreover, it is inconsistent regarding the *<t=>* descriptor. That is, the start time is greater than the end time. Finally, the $a = rtpmap$ attribute does not correspond to an already declared payload. *MalSDP2* contains the word "RANDOM" in the first *<m=>* line. This value is completely irrelevant to the specification. Also, it contains the same media part twice. The *<t=>* descriptor of *MalSDP3* carries an end value which has a length of 11 digits. This number exceeds the maximum allowed limit of 10 digit. Additionally, numbers 1 and 2 cannot be used as a payload because they are reserved. Finally, the numbers 97, 98 correspond to dynamic payloads [23], but they have not been declared in the media level. Keep in mind that codecs which have been defined in "RTP Audio/Video

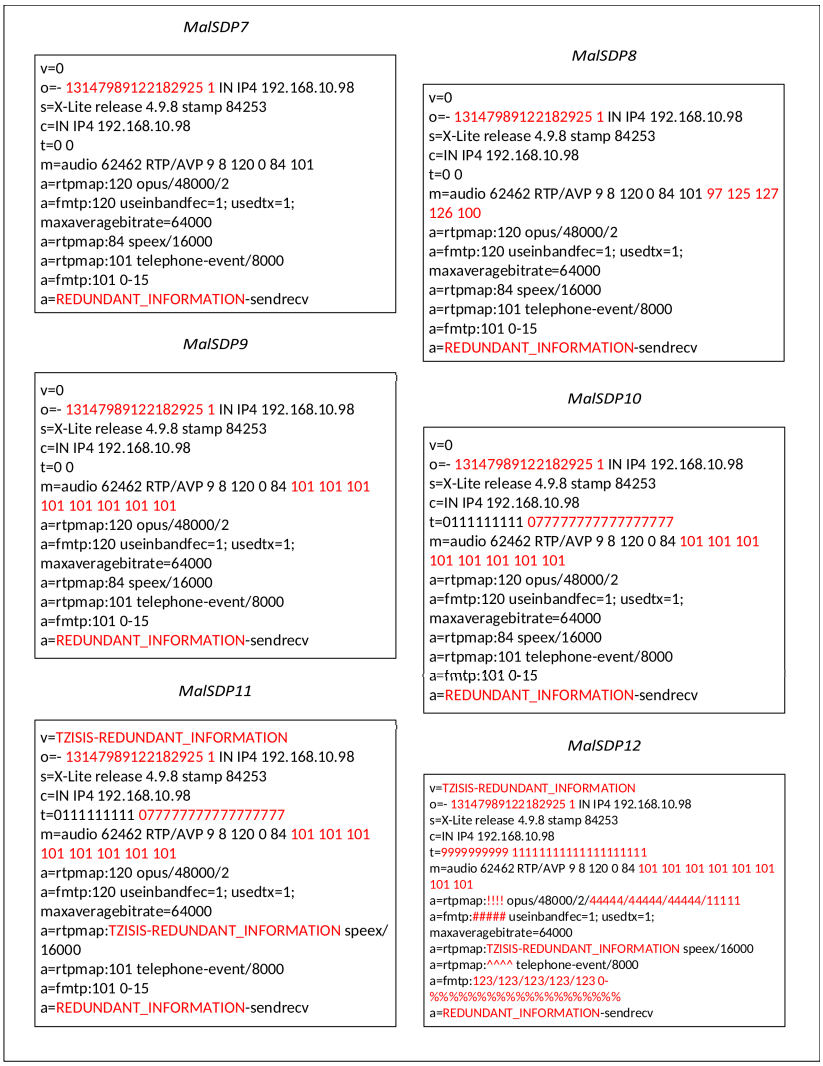

FIGURE 5. Malformed SDP bodies relayed from Kamailio and Asterisk (The malformed part is shown in red font).

profile'', do not require additional information to be decoded. Nevertheless, the aforementioned session attribute lines (used for dynamic codecs), may also be employed for the static ones.

The $<o=>$ descriptor in *MalSDP4* carries a session ID which violates the Network Time Protocol (NTP) format as specified in RFC 4566. *MalSDP5*, conveys a cryptographic key via the $<k=>$ descriptor. However, the use of cryptographic keys in favor of supporting older implementations are not recommended by RFC 4566. Finally, *MalSDP6* makes use of a very big number equal to $34,567,999 * 10^{33}$, as a session-id in the $<o=>$ descriptor. Additionally, it conveys several voluminous strings. Moreover, all the 6 SDP bodies depicted in figure 5 present significant fraudulent alterations in either the session or the media part of their SDP segment. Last but not least, the media level part of these bodies carry the numbers 5, 0, which are nevertheless used only with audio codecs.

B. EXPERIMENTATION WITH SIP EQUIPMENT

The 6 malformed SDP bodies of figure 4 were used to test the robustness of 9 different SIP softphones and 1 hardware phone. For selecting the softphones, we searched the Google play and Apple store, and cherry-picked some of the most popular ones [24], namely Sipdroid v.4.1 beta, EVA Sip phone v.2.1, CSipSimple v.1.02.03, MizuDroid v.2.4.0, Media5-fone v.4.25.4.13060, Linphone v.3.3.2, SessionChat v.6.0, VaxPhone v.8.6.0.2, and rDialer v.1.1. Panasonic KX-HDV130 (with firmware HDV130/06.101) has been used as the hardware SIP phone. This device is amongst the most popular in the Unified Communications (UC) market [25]. All the above mentioned phones' name, model and type are summarized in Table 3.

SIP phones employ a set of audio and video codecs for establishing a multimedia session with peer SIP components. Normally, the phone selects the codecs based on a configurable prioritization list. For instance, ITU-T G.711 [26] is

TABLE 3. SIP Phones used and results (Sw/Hw stands for softphone/hardware phone).

Attack Type (in Table 2)	Phone Name	Attacked with MalSDP#	Behavior	Type/OS
1	Sipdroid	All *MalSDPs*	Error: Codecs incompatible, 403 Forbidden	Sw/Android
1, 2, 3	EVA Sip Phone	*MalSDP1**, *MalSDP4 to 5*$^{+}$	*Continuous Ringing, $^{+}$Error: 400 Bad SDP	Sw/Android
1	CSipSimple	All *MalSDPs*	488 Not acceptable here	Sw/Android
1, 2, 3	MizuDroid	All *MalSDPs*	Continuous Ringing	Sw/Android
1	Media5-fone	All *MalSDPs*	488 Not acceptable here, Missed call	Sw/Android
1, 2, 3	Linphone	*MalSDP1** and *MalSDP3 to 4**, *MalSDP2*$^{+}$ and *MalSDP5*$^{+}$	*Responded normally, $^{+}$Responded but call time counter froze	Sw/Android
1	SessionChat	*MalSDP1**, *MalSDP2 to 5*$^{+}$	*400 Parse Error, $^{+}$480 Temporarily not available	Sw/iOS
1	VaxPhone	All *MalSDPs*	No Response	Sw/iOS
1	rDialer	All *MalSDPs*	No Response	Sw/iOS
1, 2, 3	Panasonic	*MalSDP1 to 2**, *MalSDP3 to 5*$^{+}$	*488 Not acceptable here, $^{+}$The phone rings and the call is answered	Hw/HDV130

one of the most commonly used in VoIP realm. In this context, when the phone receives a SDP segment containing codecs which are incompatible, it will respond with a *"488 not acceptable here"* SIP response. This means that the call is not established. As observed from Table 3, this error message was generated by phones 3, 5 after sending all *MalSDPs* depicted in figure 4. Finally, the same error code was generated by the Panasonic phone, after receiving *MalSDP1* and *MalSDP2*. Obviously, this is an indication that the phone produces a generic - and in this case misleading - codec error message instead of a more precise and detailed one pertaining to the specific SDP error. The rest of the phones responded with different messages. Sipdroid produced a *"403 forbidden"* message. The EVA Sip phone encountered a bug as it kept ringing continuously even after pressing the response button. However, in this case, the corresponding logs showed that the rest of the protocol messages (*180 RINGING* and *200 OK*) have been exchanged successfully. A similar behavior to the EVA SIP phone has been observed with the MizuDroid softphone. Finally, the Linphone ringed normally and the call was answered, but for *MalSDP2* and *MalSDP5* the call time counter had frozen.

We also sent 5 consecutive instances of *MalSDP6* message to all the phones.The EVA Sip phone, Sipdroid, CSipSimple, and Media5-fone showed the same behavior as previously. On the other hand, Linphone and MizuDroid crashed, and additionally for MizuDroid, the smartphone rebooted without a warning.

Regarding the softphones on the iOS platform, SessionChat responded with a *"480 Temporarily not available"* for *MalSDP2* to *MalSDP5*. For *MalSDP1*, the same softphone displayed a *"400 SDP parse error"* message. VaxPhone and rDialer did not respond at all. Finally, the Panasonic phone responded with a *"488 Not acceptable here"* error message for *MalSDP1* and *MalSDP2*. For the rest, the phone ringed normally and the call was answered. Table 3 summarizes the behaviour of the tested software and hardware phones. Generally, a SIP phone which receives a manipulated SDP request and it responds normally, provides the attacker with beneficial information regarding an inherent weakness of the SDP parsing process at the UA side. Then, using this information gathered during a reconnaissance phase, the attacker knows which UAs are suitable for hosting their bot(s). After infection (which is outside the scope of this paper), the botmaster can uneventfully convey hidden commands toward their bots. An example of this situation is the Panasonic device listed in Table 3, which does not produce an error code for messages *MalSDP3* to *MalSDP5*. Details on the creation of such a communication channel are given further down in this section.

As the reader realizes, the root cause of this kind of attacks is that the SIP proxies forwarding SIP messages between the peers do not inspect their SDP bodies for possible inconsistencies. In our tests, we employed two of the most popular SIP servers in the VoIP realm, namely Kamailio ver. 5.0.2 (former OpenSER) [27] and Asterisk ver. 14.6.0 [28]. Both these servers relayed the majority, if not all, of the crafted messages. Precisely, Kamailio relayed all the messages depicted in figures 4 and 5, while Asterisk only those included in figure 5. Generally, Asterisk presented a higher resiliency in crafted SDP bodies, especially when the manipulation concerns the <*m*=> descriptor. This is the main reason that the messages depicted in figure 4 were rejected by Asterisk.

In any case, this parsing negligence leaves plenty of room for creating covert communication channels. That is, as already mentioned, the attacker is able to uneventfully convey ''in plain sight'' whatever information they wish, by simply altering the SDP segment of certain SIP messages. Let's say that a botmaster wishes to construct a simple communication channel consisting of 3 commands pertaining to the type, the parameters, and the initiation/termination of an attack. To do so, they rely on, say, the <*a*=> and <*t*=> descriptors, and exploit their respective fields included in red rectangles in Table 1. For the former descriptor, the possible values are chosen to be 20 (notifies the receiving UA-bot to extract the attack parameters and stand by for further instructions), and 30, 40 (correspondingly signify the initiation/termination of the attack. On the other hand, the first and the second half of the victim's IP are included in the second and third fields of the <*t*=> descriptor. Say for example that the victim's IPv4 is 212.120.83.153. Then, the second and third fields of the <*t*=> descriptor would be 3|3|212|120|00 and 2|3|83|153|110 respectively. Note that the vertical bar character has been included in the numbers to ease the comprehension of the example. That is, assuming a quad-dotted notation, the UA-bot will interpret the aforementioned numbers as follows: the first 2 digits of each number (3, 3 for the first number and 2, 3 for the second) carry the

number of digits that comprise this half of the IPv4 address, which immediately follows (i.e., 212.120 for the first half and 83.153 for the second). Also, the type of the attack is included in the first 2 of the last 3 digits (11) of the second number. Note that all the selected values in both these descriptors seem perfectly legitimate. For instance, the length of the 2 numbers, correspondingly denoting the session id and version, is 10 digits, so as to fully comply with RFC 4566. Also, the channel remains functional because the selected $<a=>$, $<t=>$ descriptors receive values that correspond to fields which do not affect the session establishment. Naturally, one has to also consider that each optional descriptor adds $\approx$10 extra bytes per message, which however is insignificant and hardly perceivable by the underlying defense mechanisms. Another point to consider is that the above mentioned channel will be blocked by the SDP protection module detailed in the next section. This is simply because the syntax of the $<t=>$ descriptor infringes rule no. 62 as referred in the Appendix.

Lastly, more worrisome is the fact that apart from crafted messages, we managed to relay a huge SDP segment using Kamailio. That is, we created a SDP segment with size equal to 12,116 bytes. This segment passed through the SIP proxy intact without returning an error code to the sender. This means that the SIP proxy could potentially relay enormous SDP bodies without hesitation. This naturally renders both the SIP server and registered UAs prone to simple volumetric DDoS attacks.

V. SDP PROTECTION MODULE

This section elaborates on our solution to bolster defense capabilities against SDP-driven attacks.

A. HIGH-LEVEL DESCRIPTION AND NETWORK ARCHITECTURE

A SDP parser is a software module capable of analyzing SDP bodies. Normally, such a module is present in every SIP server and UA. However, as already explained in section 4, both these (real-life) entities largely fail when it comes to malformed SDP bodies. So, from an attacker's viewpoint, virtually all major SIP components, either servers or end-user equipment are sitting targets. Also, more advanced attackers, say, botmasters perceive the SIP infrastructure as an alluring means of building their C&C protocol. Once more, as in the case of many other protocols or systems, it seems that security is not the first priority of the designers/implementors of SIP products and security-by-design is still a far-fetched goal. The solution however is rather straightforward; the SDP parser must be able to tell between a well-formed and malformed message in terms of RFC 4566. Preferably, this must be done for both the incoming and outgoing traffic to/from the SIP server.

To this end, this work introduces an autonomous open-source software module for SDP messages that can be either physically co-located with the SIP proxy or reside in a different machine, say, in the perimeter of the network. In the context of this work, we opt to select the latter configuration as it is entirely transparent to any SIP compliant component. This option also enables one to integrate the SDP parser with, say, a next generation firewall. The overall architecture of such a network configuration is illustrated in figure 6.

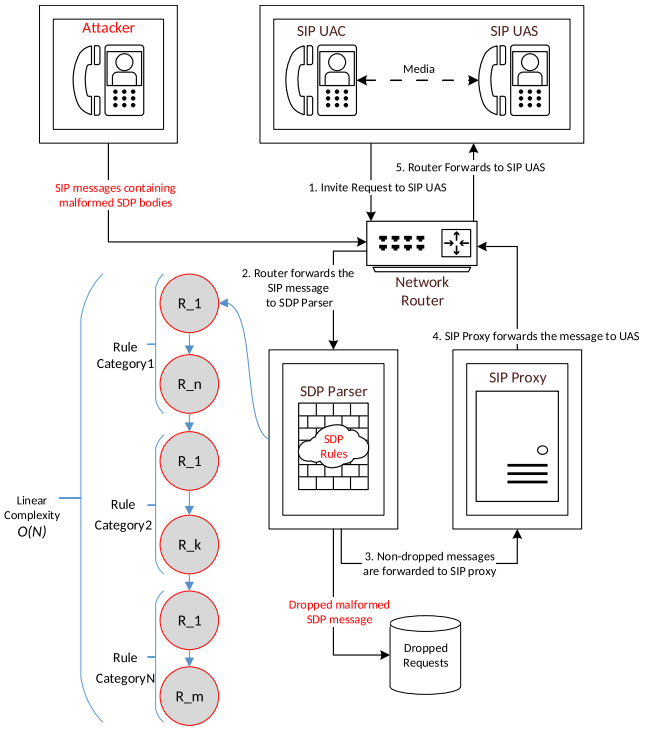

FIGURE 6. Overview of the deployed testbed. The letters n, k, m represent the number of rules per rule category.

Very similar to a firewall, the SDP parser protection module inspects each message based on a parsing policy reflected to one or more sets of rules, and decides if it can be forwarded to the SIP server or silently dropped. That is, the filtering operation starts by extracting the message body from any incoming SIP request. Next, the SIP headers of the message are stripped away and the remaining part containing the SDP message (descriptors) is kept for further processing. Algorithm 8 provides an overview of the process flow and discrete operations of the SDP parser.

B. FILTERING POLICY AND PARSER RULES

As per RFC 4566, the implemented rules have been divided into 4 major categories (policies). Namely, we compiled a set of SDP parsing rules and we grouped them into the following categories: "MUST", "NOT RECOMMENDED", "SHOULD NOT". For the remaining rules, i.e., those which do not fall within any of the above mentioned categories, we created a fourth category, namely "GENERAL INCONSISTENCIES". The "MUST" policy can be further categorized into 3 sub-policies, namely "MUST APPEAR", "MUST NOT APPEAR", and "MUST HAVE". Until now, altogether the 4 chief categories contain a total of 100 rules, which tackle the vast majority of inconsistencies that may exist in a SDP message. Of course, one is able to add more

Algorithm 1 SDP Segment Classification

```
for RuleSet i ← 1 k do
    if RuleSet i is enabled then
        for Rule j ← 1 N do
            if Rule j applies on SDPsegment then Drop message; return;
            end if
        end for
    end if
end for
```

rules at any time. The full list of the implemented rules is given in the Appendix.

Note that all the rules have been created after grindingly extracting every single piece of information which pertains to the standard syntax of SDP. For example, with reference to figure 2, rule 63 describes the length of the <*t*=> descriptor. Based on RFC 4566, the length of this descriptor must be equal to 2, meaning that if this descriptor appears in a SDP session level part, it must have the form $t = <start\text{-}time><stop\text{-}time>$. The same logic has been followed for the rest of the implemented rules. Up to now, our implementation covers all the mandatory descriptors, plus many of the optional ones. Full coverage is expected in a future version of the parser, which is publicly available at [29].

The implemented rules can be categorized either by using the 4 above mentioned categories, or according to their relevance to the different SDP regions (i.e., with reference to figure 2, *session level* and *media level*). This three-fold subcategorization is described under the ''Filtering sub-policies'' header in the Appendix. For example, the mandatory descriptors reside only in the session level of the SDP segment. So, the first filtering sub-policy in the Appendix dictates that all the mandatory descriptors must be present in an SDP message. Otherwise, the message is dropped. Overall, for every defined rule, each time an inconsistency is found, the parser silently drops (and preferably logs) the SIP message. On the contrary, if the message is found to be sound it is forwarded to the SIP proxy.

C. DESIGN CONSIDERATIONS

Misuse detection systems (also known as signature-based detection) rely on known signatures, that is, detection rules aiming to distinguish legitimate traffic instances from the malicious ones. However, while these systems are able to detect known attacks and have a high degree of portability between systems that face the same ilk of attacks, they miss to recognize novel attacks or variations of known ones. Thus, the detection ability of a misuse detection system, as the one proposed in this work, primarily depends on the newness of the detection rules the system has been configured with. In this context, we selected the specific rules based on the fact that, in the normal case, altogether the mandatory descriptors, namely <*v*=>, <*o*=>, <*s*=>, <*m*=>, <*t*=> offer a limited number of ''variables'' in contrast to the optional ones. The term ''variable'' refers to the part of every descriptor which can be altered by the sending entity, either for benign or malicious purposes. For example, with reference to the fourth column of Table 1, and for the <*o*=> mandatory descriptor, the number of ''variables'' is equal to 6. In general, altogether the mandatory descriptors but <*m*=> provide 13 ''variables'', which can be malignantly altered. In the normal case, <*m*=> carries 3 ''variables'' plus a number of payloads, which ranges from 1 to n. Also, in certain cases, e.g., in an audio/video session, there may be more than one instances of <*m*=>.

The 10 optional descriptors on the other hand provide a much larger ''variable'' space for the assailant, simply because their number may be triple the quantum of the mandatory ones (some of them may even be repeated in the SDP segment), and pieces of information conveyed by certain optional descriptors may not be critical or simply ignored by the SIP proxy. Therefore, as shown in the example of section IV, the optional descriptors are low hanging fruits for attackers who seek to hide information in a SDP body toward creating a covert communication channel. Even worse, considering the offer/answer model (see sections II and IV) the UA is capable of renegotiating the SDP parameters at any time and each time present a different SDP segment. For instance, if the caller wishes to put the callee on hold, it needs to send them a re-INVITE carrying the SDP <a=inactive> attribute (along with any other crafty SDP alterations).

All in all, the greater the number of ''variables'' the larger the attack vector the malformed message may yield, because the aggressor is able to arm the message with several different variations of malformed fields in an effort to inflict damage to the victim or pass through a richer set of hidden information. Of course, as already touched upon in section IV, the inclusion of many optional descriptors in the message is sure to increase its size (note that any SIP implementation must be capable of handling messages up to the maximum datagram packet size, e.g., for UDP this size is 65,535 bytes, including IP and UDP headers). Also, it is to be noted that in order to avoid the fragmentation of messages over UDP and offer congestion control for larger messages, RFC 3261 states that any request within 200 bytes of the path Maximum Transmission Unit (MTU) or larger than 1,300 bytes if the path MTU is unknown, must be dispatched using TCP. This, on the other hand, complicates the creation of large malformed SDP messages along with IP spoofing at the attacker's side.

Bearing the above in mind, we designed the SDP parser in such a way so the mandatory descriptors - which are critical and cumulatively have the lesser number of ''variables'' - are always filtered first. After that, the selection of the rules gradually covers the case of voluminous SDP segments, where the parser possibly needs to go through a much greater number of rules. More details regarding the time complexity of the parser are given in the next section.

D. IMPLEMENTATION

The SDP message parser has been implemented as a standalone multi-threaded Java application. Its first task is to intercept SIP requests and responses carrying a SDP segment (see section IV) and destined to the SIP proxy. To do so, we employed the well-known JAIN-SIP stack [30]. The JAIN architecture comprises 3 different layers. The first layer corresponds to the protocol stack. The second refers to the JAIN layer, and the last one to the actual application. Precisely, the latter layer provides the necessary methods to format and send out SIP messages. Furthermore, it provides interfaces capable of extracting and parsing specific message headers. As shown in figure 6, we employed the well-known iptables linux utility program to redirect the traffic as follows: for every incoming packet with destination TCP or UDP port equal to 5060 (the standard SIP proxy port used for non-encrypted signaling traffic), we forwarded the packet to the parser, listening on the corresponding TCP or UDP port 6090. If the message is found to be well-formed, it is forwarded to the SIP proxy as normal. This is done with the help of the *SendRequest* method of the JAIN SIP API.

The parser allows the network administrator to select the descriptors to be filtered based on a graphical user interface (GUI). Namely, the user is capable of selecting a specific policy reflected to one or more categories of rules, as illustrated in figure 6 and described in the Appendix. A screenshot of the parser's GUI is presented in figure 7. As observed from the figure, the various filtering policies (categories of rules) reside in the upper left section of the GUI. Recall that the parser application is freely available for further development and experimentation [29].

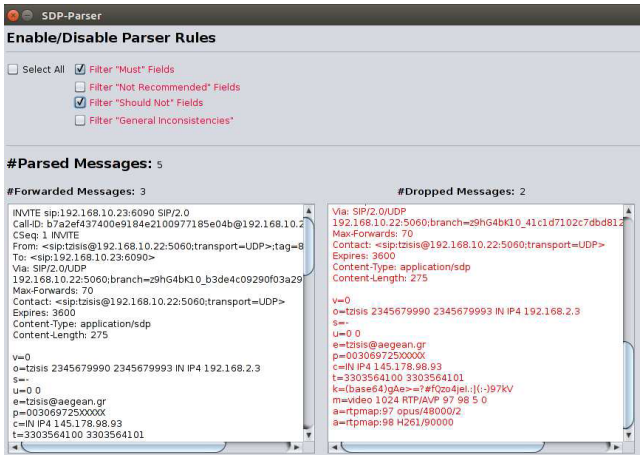

FIGURE 7. The front-end of the SDP parser.

We also developed a Java attack tool capable of generating malformed SDP INVITE SIP messages. The application produces a variety of malformed SDP messages, including the ones described in section 4, as well as others with random values in specific SDP fields. Also, the same tool allows one for sending a single or a surge of malformed SDP requests towards a UA or SIP proxy. While our experiments make use of only INVITE requests, the exact same methodology can be followed for any other SIP request/response having a SDP segment, as detailed in section IV.

VI. PERFORMANCE EVALUATION

This section reports on the performance evaluation of the proposed scheme. To this end, we employed a SIP testbed architecture in which we launched various attack scenarios to estimate the effectiveness of our SDP parser under different traffic conditions. We assess our solution in terms of the introduced overhead for service provision. The following subsections elaborate on the testbed architecture, the deployed scenarios, and the obtained results.

A. TESTBED

The employed testbed is depicted in figure 6. Specifically, 3 different Virtual Machines (VM) have been used to host the UAC/UAS, the SDP malformed message attack tool described in subsection V-D, the SDP parser, and the SIP proxy. The physical machine hosting the VMs is equipped with an Intel i5-4310m processor clocked at 2.7 GHz and 8 GB of RAM. The Kamailio SIP server in ver. 5.0.2 has been used as a SIP registrar and proxy. We utilized the well-known Sipp tool [31] to test the performance of the parser under stress, that is, by simulating legitimate SIP calls between a caller and a callee. The UAC and the UAS operate on the same VM using different port numbers, namely 6040 and 8040 correspondingly. Keep in mind that the conveyed traffic must always be filtered by a firewall.

As already pointed out, for launching the attacks, we used the Java tool already described in subsection V-D. This attack tool dispatches a number of SDP malformed requests (those of figures 4, 5, plus the one with the huge SDP segment) depending on the needs of each scenario. A detailed description for the employed scenarios can be found in the next subsection. The SDP parser has been logically placed in front of the SIP proxy, but both run on the same VM, sharing a dual-core CPU and 2 GB of RAM. Having the SDP parser hosted by a separate physical machine is estimated to introduce negligible time penalty in terms of communication time, given that the SIP proxy and the parser will normally reside in the same subnetwork.

B. SCENARIOS

For evaluating the performance of the parser, and consequently estimate the overhead inflicted by our solution on the provided service, we created a set of 16 scenarios (Sn) divided in 4 categories as shown in Table 4. That is, as indicated in the third column of the table, for each category of scenarios, we flooded the target proxy with 500, 1500, 2500, and 4000 INVITE SDP malformed requests, using random time intervals of 3 to 10 sec between the flood bursts.

We selected random time intervals because the strategy of the attack (i.e., the use of a certain distribution in the attack pattern) does not affect the detection accuracy. This happens because a syntactically wrong message will be dropped

TABLE 4. Parameters of attack scenarios.

Scenario	Legitimate traffic (CPS)	Number of malicious INVITE sent
Sn1.1	10	500
Sn1.2	10	1500
Sn1.3	10	2500
Sn1.4	10	4000
Sn2.1	20	500
Sn2.2	20	1500
Sn2.3	20	2500
Sn2.4	20	4000
Sn3.1	40	500
Sn3.2	40	1500
Sn3.3	40	2500
Sn3.4	40	4000
Sn4.1	80	1500
Sn4.2	80	1500
Sn4.3	80	1500
Sn4.4	80	1500

independently of the call distribution. Namely, in the worst case scenario, the system may loose some packets due to lack of CPU resources, but statistically the SDP parser detection accuracy is not affected as the arriving INVITE packets are mirrored to statistically independent events.

Moreover, as shown in the second column of Table 4, during the execution of the attacks, the SDP and SIP proxies were stressed with legitimate traffic following a pace of 10, 20, 40 and 80 calls per second (CPS).

C. PERFORMANCE EVALUATION

Given that the detection accuracy of the SDP parser depends solely on the implemented rules (i.e., the parser had 100% detection accuracy against the malformed messages of figures 4 and 5), the focus of this section is on service time provision. To this end, we assess the overhead on both the SDP parser and SIP proxy in terms of message processing time. This means that for each group of scenarios in Table 4 (Sn1.1 to Sn1.4, Sn2.1 to Sn2.4, and so on), we measured the processing time per incoming message in both the SDP parser and the SIP proxy. Next, we present the results per group of scenarios using box-and-whisker plots in figure 8, where the lower hand edge of the blue box corresponds to the 1st quartile, the upper hand edge of the red box to the 3rd quartile, and the line between the boxes is the median. For instance, in the upper half of the figure, the distribution of message processing time for all scenarios (Sn1.x) in group Sn1 for the SDP parser is marked as "Sn1" in the horizontal axis.

For the last 4 sub-scenarios (Sn4.x), we used the same number of SDP malformed INVITE packets. As explained in subsection VI-B, this has been done because the detection accuracy of the SDP parser is not affected by the number of packets. Nevertheless, we used a different number of packets for the first 3 groups of scenarios (Sn1 to Sn3) for the purpose of demonstrating that the SDP parser behavior remains stable independently of the deviation in the rate of the incoming malformed packets. As expected, during the experiments, we witnessed network performance issues expressed as packet loss. This was due to the excessive attack traffic on top of the legitimate calls. Keep in mind that this behavior is

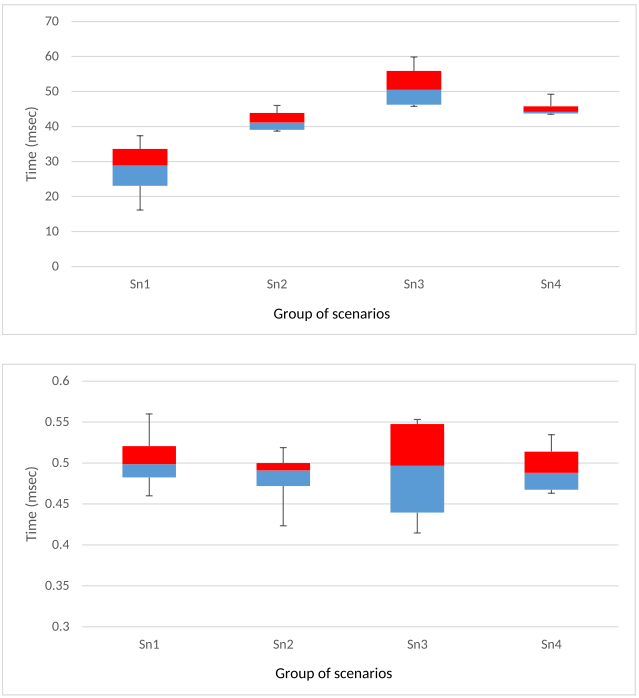

FIGURE 8. Time overhead per category of scenarios for the parser module (upper half) and the SIP proxy Kamailio (bottom half).

expected because of the connectionless nature of UDP, which is often preferred over TCP, especially when the number of devices connecting to the SIP server grows.

Focusing on the upper half of the figure, i.e., the SDP parser, we easily observe that the average message processing (parsing) time per incoming message for all the scenarios fluctuates between 30 to 50 msec. Also, the minimum and maximum values for all the 4 groups of scenarios are between 17 and 60 msec. Overall, we can safely argue that the SDP parser adds a negligible time of the order of tenths of milliseconds. Even in the most stressing group of scenarios (Sn3, Sn4) where the CPS is 40 to 80, the average parsing time does not exceed 60 msec. However, bear in mind that the aforementioned parsing times are only indicative because they do not only depend on the volume and type of SIP traffic, but also on the computing resources available at the parser side (i.e., processor type, available memory, etc).

When comparing the first 3 groups of scenarios, one can also perceive an increment of $\approx$ 10 msec in the average message parsing time proportionally to the CPS parameter. That is, from $\approx$ 30 msec for Sn1 to $\approx$ 40 for Sn2 and $\approx$ 50 for Sn3. Sn4 on the other hand, showed faster message parsing times because the volume of the SDP malformed messages sent in all Sn4.x was constant. This is verified by the corresponding Interquartile Range (IQR = Q3-Q1), which is the smallest amongst all four group of scenarios.

In a nutshell, and as explained further in the next subsection, the parser introduces a delay proportional to the number of rules. That is, in terms of asymptotic notation, the parser's

upper bound is *O(N)*, where *N* is the number of rules enabled in the parser. Of course, this number is reflected to the currently activated SDP filtering policies as the case may be. A second factor which affects the parser complexity pertains to the number of selected categories. When this number is constant (as in our case where k = 4), then the complexity converges to *O(N)*. On the other hand, if k is variable, then the asymptotic worst-case reaches the quadratic complexity, namely *O(N*k)*.

To acquire a better understanding of the aforementioned results in terms of SDP parsing penalization, we compare the previous times against those produced by Kamailio for exactly the same 4 groups of scenarios. The results are depicted in the bottom half of figure 8. From the figure, it is easily perceivable that Kamailio produces very fast and stable message processing times in the order of millisecond even under excessive stress. This is however highly expected as modern VoIP servers are developed for handling thousands of CPS [27]. Specifically, for all the 4 scenarios, the average processing time per message fluctuates between 0.48 and 0.52 msec. This means that even with the addition of the SDP parser in front of the SIP server, one would expect overall message processing times in the order of 100 msec. Also, in the case of a volumetric attack leveraging SDP, the SDP parser is expected to ease the burden of the SIP server because the malformed messages will be dropped.

D. DISCUSSION

As already pointed out, the SDP parser has been developed with a firewall-based logic. That is, as shown in Algorithm 8, upon the arrival of a SIP message carrying SDP information, the parser extracts the SDP segment and performs a linear search against the list of the enabled rules, with the aim of detecting inconsistencies. Obviously, a SDP malformed message will pass through the parser undetected in case it contains an inconsistency that has not been implemented as a rule (or due to the current policy, the corresponding set of rules is not enabled). As already mentioned, the SDP parser performs a linear search, thus, assuming N active rules, the worst case will require the examining of N rules, while the best only involves the first rule in the current set. Note that the worst case applies also when the incoming message is well-formed. To put it another way, the overall time required to detect a malformed message is always proportional to the number of the enabled rules and does not depend on the specific set of rules or the combination of the enabled set of rules as the case may be. Therefore, the time complexity of the parser is *O(N)*. On the other hand, the detection accuracy of the parser will always reach 100%, assuming that the presented inconsistency has been addressed in an already implemented rule, which is additionally enabled.

A separate note should be done regarding the selection of the Java programming language for implementing the parser. Note that this was mainly done for the sake of portability. In this respect however, the parser's performance may be negatively affected by a) the Java Virtual Machine translation cost, and b) Java garbage collection process. Therefore, if portability is not a priority, a C-based implementation of the parser may perform faster given that (a) C code is directly compiled into native code, thus is expected to surpass Java bytecode even in the case of Just-in-Time compilation [32], and (b) memory management optimizations are possible when programming with C (but this may on the other hand increase the software bug surface).

VII. RELATED WORK

Single source or distributed DoS attacks comprise one of the cardinal issues in the cybersecurity domain. Up to now, a great mass of researches have been presented in the literature to cope with this ilk of attacks in diverse network ecosystems [33]–[38]. In this context, one of the biggest challenges at the defender's side pertains to the filtering mechanisms as a part of, say, an Intrusion Detection System (IDS), which are employed for telling the normal and attack traffic apart [39]–[41]. This section elaborates on the related works that have been presented in the literature so far regarding VoIP security in general and malformed-message-driven DoS in particular.

To date, several research works have been devoted to the domain of malformed messages in SIP ecosystems. Their common ground, however, is that they all consider malformed SIP headers, not SDP ones, which therefore is the focus of this work. The very first of them, namely the PROTOS test suite [42], evaluated the robustness of SIP entities against different types of malformed messages. The outcomes of this project indicated that, at least at that time, most of the components of the SIP architecture were unable to successfully parse malformed messages. Actually, this type of flaw is directly inherited to any SIP-based service. For instance, Vrakas [43], [44] demonstrated the impact of malformed messages in the IP Multimedia Subsystem (IMS), which is an integral part of modern 4G and beyond mobile networks. In [7], the same authors employed a bloom filter mechanism for dealing with this kind of DoS attacks, and they reported a ≈ 0.95 probability of detection.

Concentrating on SIP proxies, Geneiatakis *et al.* [9] introduced a protection solution, which can be used to strengthen the robustness of SIP server's message parser against these attacks. Their results showed that SIP parser enhancements inflict negligible delays, and thus do not affect the end-user in terms of service times. Some very similar approaches has been followed by the works in [45]–[47]. Also, Chen and Itoh [12] introduced a methodology for creating appropriate rules for SIP messages based on Augmented Backus-Naur Form (ABNF) metalanguage, while Lahmadi and Festor [48] presented a specification language to describe SIP vulnerabilities, including malformed message ones as well as the associated countermeasures.

Rieck *et al.* [49] contributed a machine learning based solution for detecting both known and unknown attacks against SIP based services. This is orthogonal to existing protection approaches as it can function complementary

TABLE 5. Summary and comparison of related work.

Reference	SIP/SDP	Protection against	Type of solution	Realtime	Min/Max overhead (msec)	Accuracy (%)
Proposed	SDP	Malformed Msgs / Covert channel	Rule-based	Yes	17 / 60	100
[9]	SIP	Malformed Msgs.	Rule-based	Yes	0.016 / 0.719	N/A
[45]	SIP	Malformed Msgs.	Rule-based	Yes	0 / 926	N/A
[46]	SIP	Malformed Msgs.	Rule-based / Hybrid	No	N/A	N/A
[47]	SIP	Malformed Msgs.	Rule-based	No	N/A	N/A
[12]	SIP	Malformed Msgs.	Rule-based	No	N/A	94
[48]	SIP	Malformed Msgs. / DoS	Rule-based	Yes	2000	N/A
[49]	SIP	Novel attacks	Machine Learning	No	N/A	>99
[50]	SIP	Malformed Msgs.	Machine Learning	No	0.45	99.9
[53]	SIP	(D)DoS / Malformed Msgs.	Machine Learning	No	N/A	99.89

to them for improving the security level of the provided services. In the same direction, Ferdus *et al.* [50] developed a two-stage solution based on lexical analysis and machine learning techniques for detecting sophisticated malformed SIP message attacks. Following a similar approach, Tsiatsikas *et al.* [51], [52], evaluated the performance of several machine learning classifiers in the detection of (D)DoS attacks. Also, Akbar *et al.* [53] developed a SIP message parser with the purpose of counteracting (D)DoS attacks. The parser utilizes Support Vector Machine (SVM) for message classification, and they report a 99.89% accuracy under different scenarios involving malformed messages residing in SIP message headers.

Excluding attacks that exploit malformed messages, Sisalem *et al.* [54] demonstrated the consequences of well-formed SIP messages that incorporate in their headers irresolvable addresses. The authors showed that this type of attack can paralyze the provided service. Later on, as a mitigation strategy against this type of attack, Zhang *et al.* [55] introduced a non-blocking DNS caching solution.

Lastly, the still limited number of SDP-related vulnerabilities included in public Common Vulnerabilities and Exposures (CVE) databases [21], [22] are the most relevant to our work. Actually, these flaws pertain to simple, sporadic bugs discovered in random during either the phase of media negotiation at the SIP proxy, or upon message parsing at the SIP phone. Usually, the fixes given in such cases are proprietary or ad-hoc, say, patching the software bug, or correcting the SIP flow from a design point of view. On the contrary, the solution proposed by this work is holistic and purely proactive, meaning that the malformed messages are reaped before having the chance to interact with the SIP entities.

Given the above, the literature so far largely misses the exploration of SIP malformed message attacks, which specifically leverage SDP information conveyed by certain SIP messages. Even the RFC 4475 [56] which is dedicated to SIP torture test messages does not explicitly elaborate on SDP malformed message attacks. To this end, the work at hand not only contributes proofs of the feasibility and impact of this ilk of attack against real-life diverse SIP entities, but additionally offers a publicly available parsing mechanism that can remedy it. Also, the same solution can help into deterring evildoers from exploiting the SIP infrastructure for building secret communication channels. To ease the reading of this section, Table 5 summarizes the works dealing with malformed messages in SIP, and offers a side-by-side comparison based on 6 criteria, where available. Note however that the last two criteria, namely ''overhead'' and ''accuracy'', in the table can only be considered as indicative because:

- They are concerned with different parts of a SIP message. That is, all the works but ours examine the SIP header, not the SDP segment. The latter is far more complex than the former because, as detailed in subsection V-C, it may contain several optional descriptors, and each of them may carry several ''variables'' as shown in the fourth column of the revised Table 1.
- The overall setup, including the testbed, the implemented scenarios, and the dataset in the various works are not the same. With reference to subsections V-D and VI-A, the SDP parser is implemented in Java, and runs on the same VM with Kamailio having a dual-core CPU and 2GB of RAM. On the other hand, for instance, the work in [9] employed a machine equipped with a Pentium 4 processor clocked at 2.5 GHz and 256 MB of RAM, while that in [48], uses an Intel Core i7 2.0 GHz Quad-core CPU and 8 GB RAM.
- The nature of the solutions are different. That is, some are rule-based, while others are machine-learning driven. Therefore, for the former, the detection accuracy depends on the quality and freshness of the implemented rules, while for the latter on factors including the quality of the dataset (labeled/unlabeled data), the classifiers used, and the pre-processing and training processes.

A last point to consider is that SDP is not specific to SIP. In fact, as per RFC 4566, SDP is designed to offer a general purpose format for conveying multimedia session description metadata to the peers. Hence, it can be used in a variety of network domains and applications, and over several transport protocols, including Session Announcement Protocol (SAP), SIP, Real Time Streaming Protocol (RTSP), electronic mail via the Multipurpose Internet Mail Extensions (MIME), and the Hypertext Transport Protocol (HTTP). This means that the contributions of this work directly apply to also any application that uses one of the aforementioned transport protocols to enable multimedia communications. Another positive quality of the proposed rule-based scheme is that

is more generic and it can be more easily incorporated in low computing power client-side equipment, say, a hardware phone in the case of SIP or another end-device utilizing one of the aforementioned protocols. This does not normally apply to machine learning solutions, which are more computationally intensive.

VIII. CONCLUSIONS AND FUTURE WORK

Media negotiation comprises a fundamental operation in SIP ecosystems. However, the text nature of SDP creates a fertile ground for attackers aiming to launch DoS attacks by crafting the message bodies. So far, research has concentrated on malformed SIP message headers, largely neglecting SDP bodies. To fill this literature gap, we not only demonstrate the magnitude of SDP-powered attacks to real-life SIP equipment and the user, but also propose a straightforward to implement and fully compatible with SIP proxy-based method to solve it.

The obtained results show that our solution comes at a negligible cost in terms of service time provision while, as with any misuse detection system, its detection accuracy depends solely on the freshness and quality of the implemented rules. Also, it is directly extensible by simply contributing additional rules to one or more of its categories pertaining to SDP message body filtering policies. These policies can be enforced or disabled by the administrator at will. Also, as explained in Section VI, each new rule added to the list, affects linearly the parser performance. That is, in terms of algorithmic time complexity, the search operation will always be equal to *O(N)* across the number of rules included in the currently enabled policies.

Given that SDP may be used on top of different transport protocols, our future work aims at (a) calibrating the parser and populating it with more rules, (b) testing the parser in other application domains, including Web Real-Time Communication (WebRTC) environments that leverage SIP as their signaling protocol, and (c) implementing a two-layer solution, which in addition to rule-based detection, will employ machine learning techniques to cope with previously unseen instances of SDP malicious alterations.

APPENDIX

This appendix contains the full list of the rules that have been implemented in our SDP parser until now.

Filtering Categories (policies):

- Filter MUST descriptors.
- Filter NOT RECOMMENDED descriptors.
- Filter SHOULD NOT descriptors.
- Filter GENERAL INCONSISTENCIES.

Filtering Sub-policies:

- Examine the presence and location of mandatory descriptors in the session level of SDP message. The included rules can be further categorized into 3 sub-policies, namely "MUST APPEAR", "MUST NOT APPEAR", and "MUST HAVE".
- Examine the number of elements per descriptor.
- Examine the length and/or the type of the elements per descriptor.

Set of currently implemented rules in the SDP parser per policy and sub-policy

MUST

1) "=" A space Must Not appear in the left side of the equal sign.
2) "= " A space Must Not appear in the right side of the equal sign.
3) " = " A space Must Not appear in the left and the right side of the equal sign.
4) The "c=" descriptor Must appear.
5) The "s= " descriptor Must appear at most once in the session level.
6) The "i= " descriptor Must appear at most once in the session level.
7) The "u= " descriptor Must appear at most once in the session level.
8) The "H261" codec Must appear for video media.
9) The "H264" codec Must appear for video media.
10) The "H264-RCD0" codec Must appear for video media.
11) The "H264-SVC" codec Must appear for video media.
12) The "DV" codec Must appear for video media.
13) The "u= " descriptor Must appear before the first "m" one.
14) The "e= " descriptor Must appear before the first "m" one.
15) The "H261" codec Must have a clock rate equal to 90000.
16) The "H264" codec Must have a clock rate equal to 90000.
17) The "H264-RCD0" codec Must have a clock rate equal to 90000.
18) The "H264-SVC" codec Must have a clock rate equal to 90000.
19) The "DV" codec Must have a clock rate equal to 90000.

SHOULD NOT & NOT RECOMMENDED

1) Permanent sessions Should Not be used.
2) Unbounded sessions Should Not be used.
3) "k=" descriptor is Not Recommended.
4) "b=X-" descriptor is Not Recommended.

GENERAL INCONSISTENCIES

1) RTP payload number "0" is used for audio.
2) No need for an "a=rtpmap:0" attribute.
3) No need for an "a=fmtp:0" attribute.
4) RTP payload number "1" is RESERVED. It cannot be used.
5) RTP payload number "2" is RESERVED. It cannot be used.
6) RTP payload number "3 to 18" is used only for audio.
7) No need for an "a=rtpmap:XXX" attribute for RTP payloads with numbers "3-18".

8) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "3-18".
9) RTP payload numbers "19 to 24" are either RESERVED or UNASSIGNED. They cannot be used.
10) RTP payload numbers "25 to 26" are used for video.
11) No need for an "a=rtpmap:XXX" attribute for RTP payloads with numbers "25-26".
12) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "25-26".
13) RTP payload number "27" is either RESERVED or UNASSIGNED. It cannot be used.
14) RTP payload numbers "28" is used for video.
15) No need for an "a=rtpmap:XXX" attribute for RTP payloads with number "28".
16) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "28".
17) RTP payload numbers "29" is used for video.
18) No need for an "a=rtpmap:XXX" attribute for RTP payloads with number "29".
19) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "29".
20) RTP payload numbers "30" is used for video.
21) No need for an "a=rtpmap:XXX" attribute for RTP payloads with number "30".
22) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "30".
23) RTP payload numbers "33" is used for audio/video.
24) No need for an "a=rtpmap:XXX" attribute for RTP payloads with number "33".
25) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "33".
26) RTP payload numbers "31" is used for audio/video.
27) No need for an "a=rtpmap:XXX" attribute for RTP payloads with number "31".
28) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "31".
29) RTP payload numbers "32" is used for audio/video.
30) No need for an "a=rtpmap:XXX" attribute for RTP payloads with number "32".
31) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "32".
32) RTP payload numbers "34" is used for audio/video.
33) No need for an "a=rtpmap:XXX" attribute for RTP payloads with number "34".
34) No need for an "a=fmtp:XXX" attribute for RTP payloads with numbers "34".
35) Codec "G7291" must be used for audio.
36) Codec "VC1" must be used for video.
37) RTP payload numbers "35 to 71" are either RESERVED or UNASSIGNED. They cannot be used.
38) RTP payload numbers "72 to 76" are either RESERVED or UNASSIGNED. They cannot be used.
39) RTP payload numbers "77 to 95" are either RESERVED or UNASSIGNED. They cannot be used.
40) RTP payload numbers "96 to 127" a=rtpmap is not contained in the m line.
41) RTP payload numbers "96 to 127" fmtp is not contained in the m line.
42) The sixth argument in the "o" must exist in the SIP headers.
43) Codec "VC1" must have 2 arguments.
44) Codec "G7221" must have 2 arguments.
45) Check if "o" line contains 6 arguments.
46) RTP payload numbers ">127" are out of range.
47) If the "VC1 fmtp" attribute is present, it Must contain one of the following profile values: "profile=0", "profile=1", "profile=2".
48) If the "VC1 fmtp" attribute, is present, it Must contain one of the following level values: "level=1", "level=2", "level=3".
49) Codec "VC1" must have a clock rate equal to 90000.
50) Codec "G7221" clock rate is either equal to "16000" or "32000".
51) Codec "G7221" must have a clock rate equal to "24000" "32000" assuming a sample rate equal to "16000".
52) Codec "G7221" must have a clock rate equal to "32000" "48000" assuming a sample rate equal to "32000".
53) Codec "G7221" must have a bit rate which is multiple of 400.
54) Codec "G7221" must have one of the values mentioned in rules 50, 51, 52 for the clock rate. Otherwise the value is out of range.
55) The clock rate for codec "G7291" must be "8000", "12000", "14000","16000", "18000", "20000", "22000", "24000", "26000", "28000", "30000" or "32000".
56) The clock rate for codec "L20" must be "8000", "11025", "16000","22050", "24000", "32000", "44100" or "48000".
57) The clock rate for codec "L24" must be "8000", "11025", "16000","22050", "24000", "32000", "44100" or "48000".
58) A bigger number of "rtpmap" attributes than RTP payloads exists in the SIP body.
59) The "a=orient" attribute can receive the following values: "portrait", "landscape", "seascape".
60) Check the transport protocol in the m line. It should be equal to one of the following values: "udp", "RTP/AVP", "RTP/SAVP".
61) An m line should contain one of the following values: "audio", "video", "application", "text", "message".
62) Problem in the start and or end time in the "t" line.
63) The length of the "t" line must be equal to 2.
64) The length of the "c" line must be equal to 3.
65) The length of the "v" line must be equal to 1.
66) The length of the "s" line must be equal to 1.
67) The first argument in the "c" line must be equal to "IN".
68) The second argument in the "c" line must be equal to "IP4".

69) The second argument in the "v" line must be equal to "0".
70) The first argument in the "o" line must be equal to the username or equal to "-".
71) The second argument in the "o" line must be 10 digits.
72) The third argument in the "o" line must be 10 digits.
73) The fourth argument in the "o" line must be equal to "IN".
74) The fifth argument in the "o" line must be equal to "IP4".
75) The first argument in the "t" line must be 10 digits.
76) The second argument in the "t" line must be 10 digits.
77) The port number must be between "1024" and "65535".

REFERENCES

[1] M. Handley, V. Jacobson, and C. Perkins, *SDP: Session Description Protocol*, document RFC 4566, IETF, Fremont, CA, USA, Jul. 2006.
[2] Technavio. *Global Mobile VoIP Market 2016–2020*. Accessed: Sep. 7, 2017. [Online]. Available: http://www.technavio.com/report/global-machine-machine-m2m-and-connected-devices-mobile-voip-market
[3] J. Rosenberg *et al.*, *SIP: Session Initiation Protocol*, document RFC 3261, 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3261.txt
[4] A. D. Keromytis, "Voice-over-IP security: Research and practice," *IEEE Security Privacy*, vol. 8, no. 2, pp. 76–78, Mar./Apr. 2010.
[5] D. Geneiatakis *et al.*, "Survey of security vulnerabilities in session initiation protocol," *IEEE Commun. Surveys Tuts.*, vol. 8, no. 3, pp. 68–81, 3rd Quart., 2006.
[6] S. Ehlert, D. Geneiatakis, and T. Magedanz, "Survey of network security systems to counter SIP-based denial-of-service attacks," *Comput. Secur.*, vol. 29, no. 1, pp. 225–243, 2010.
[7] D. Geneiatakis, N. Vrakas, and C. Lambrinoudakis, "Utilizing bloom filters for detecting flooding attacks against SIP based services," *Comput. Secur.*, vol. 28, no. 7, pp. 578–591, 2009.
[8] H. Sengar, H. Wang, D. Wijesekera, and S. Jajodia, "Detecting VoIP floods using the Hellinger distance," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 6, pp. 794–805, Jun. 2008.
[9] D. Geneiatakis, G. Kambourakis, C. Lambrinoudakis, T. Dagiuklas, and S. Gritzalis, "A framework for protecting a SIP-based infrastructure against malformed message attacks," *Comput. Netw.*, vol. 51, no. 10, pp. 2580–2593, 2006.
[10] J. Tang, Y. Cheng, Y. Hao, and W. Song, "SIP flooding attack detection with a multi-dimensional sketch design," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 6, pp. 582–595, Nov./Dec. 2014.
[11] Z. Tsiatsikas, M. Anagnostopoulos, G. Kambourakis, S. Lambrou, and D. Geneiatakis, "Hidden in plain sight. SDP-based covert channel for botnet communication," in *Proc. TrustBus*, 2015, pp. 48–59.
[12] E. Y. Chen and M. Itoh, "Scalable detection of SIP fuzzing attacks," in *Proc. 2nd Int. Conf. Emerg. Secur. Inf., Syst. Technol.*, Aug. 2008, pp. 114–119.
[13] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, document RFC 3550, 2003.
[14] T. Sawada and P. Kyzivat, *Session Initiation Protocol (SIP) Usage of the Offer/Answer Model*, document RFC 6337, Aug. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6337.txt
[15] E. Ivov. *Jitsi*. Accessed: Sep. 7, 2017. [Online]. Available: https://jitsi.org/
[16] E. Vasilomanolakis, J. H. Wolf, L. Böck, S. Karuppayah, and M. Mühlhäuser. (2017). "I trust my zombies: A trust-enabled botnet." [Online]. Available: https://arxiv.org/abs/1712.03713
[17] P. Biondi. *Scapy*. Accessed: Sep. 7, 2017. [Online]. Available: http://www.secdev.org/projects/scapy/
[18] Cisco Security Advisory. *Cisco Meeting Server Session Description Protocol Media Lines Buffer Overflow Vulnerability*. Accessed: Sep. 7, 2017. [Online]. Available: https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20161102-cms1
[19] Cisco Security Advisory. *Cisco Unified IP Phone 8900/9900 Series Crafted SDP Packet Vulnerability*. Accessed: Sep. 7, 2017. [Online]. Available: https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/Cisco-SA-20131010-CVE-2013-5526
[20] Asterisk Project Security Advisory. *Buffer Overflow Exploit Through SIP SDP Header*. Accessed: Sep. 7, 2017. [Online]. Available: http://downloads.asterisk.org/pub/security/AST-2013-001.html
[21] CVE Details—The Ultimate Security Vulnerability Datasource. *Known SDP-Driven Vulnerabilities*. Accessed: May 31, 2017. [Online]. Available: https://www.cvedetails.com/google-search-results.php?q=session+description+protocol+%28SDP%29
[22] NIST. *Computer Security Resource Center—National Vulnerability Database*. Accessed: May 31, 2017. [Online]. Available: https://nvd.nist.gov/vuln/search/results?adv_search=false&form_type=basic&results_type=overview&search_type=all&query=session+description+protocol
[23] H. Schulzrinne and S. Casner, *SIP: Session Initiation Protocol*, document RFC 3551, 2003. [Online]. Available: https://tools.ietf.org/html/rfc3551
[24] *How Does Mobile VoIP Work?*. Accessed: Sep. 7, 2017. [Online]. Available: https://www.voip-info.org/wiki/view/Mobile+VoIP
[25] M. Grech. *10 Simple But Powerful Small Business VoIP Phones for Under $100*. Accessed: May 21, 2018. [Online]. Available: https://getvoip.com/blog/2016/06/16/small-business-voip-phones/
[26] *Pulse Code Modulation (PCM) of Voice Frequencies*, document G.711, 1988. Accessed: Jun. 3, 2017. [Online]. Available: https://www.itu.int/rec/T-REC-G.711/en
[27] Kamailio. *The Open Source SIP Server*. Accessed: Dec. 20, 2018. [Online]. Available: http://www.kamailio.org/w/
[28] Asterisk. *Open Source Communications Software | Asterisk Official Site*. Accessed: Sep. 7, 2017. [Online]. Available: http://www.asterisk.org/
[29] Z. Tsiatsikas, D. Geneiatakis, and G. Kambourakis. *Research Project SCYPE: Software Modules*. Accessed: Dec. 20, 2018. [Online]. Available: https://scype.samos.aegean.gr/tzisis/scype_5179_software/
[30] E. Proulx. *An Introduction to the Jain SIP API*. Accessed: Sep. 7, 2017. [Online]. Available: http://www.oracle.com/technetwork/java/introduction-jain-sip-090386.html
[31] SIPp. *Free Open Source Test Tool / Traffic Generator for the SIP Protocol*. Accessed: Dec. 20, 2018. [Online]. Available: http://sipp.sourceforge.net/index.html
[32] D. Lion, A. Chiu, H. Sun, X. Zhuang, N. Grcevski, and D. Yuan, "Don't get caught in the cold, warm-up your JVM: Understand and eliminate JVM warm-up overhead in data-parallel systems," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*. Savannah, GA. USA: USENIX Assoc., 2016, pp. 383–400. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/lion
[33] X. Li, X. Liang, R. Lu, H. Zhu, X. Lin, and X. Shen, "Securing smart grid: Cyber attacks, countermeasures, and challenges," *IEEE Commun. Mag.*, vol. 50, no. 8, pp. 38–45, Aug. 2012.
[34] Z. Zhang, P.-H. Ho, X. Lin, and H. Shen, "Janus: A two-sided analytical model for multi-stage coordinated attacks," in *Information Security and Cryptology—ICISC*, M. S. Rhee and B. Lee, Eds. Berlin, Germany: Springer, 2006, pp. 136–154.
[35] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "DDoS attacks in cloud computing: Issues, taxonomy, and future directions," *Comput. Commun.*, vol. 107, pp. 30–48, Jul. 2017, doi: 10.1016/j.comcom.2017.03.010.
[36] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and software-defined networking," *Comput. Netw.*, vol. 81, pp. 308–319, Mar. 2015.
[37] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore, "Game-theoretic analysis of DDoS attacks against Bitcoin mining pools," in *Financial Cryptography and Data Security*, R. Böhme, M. Brenner, T. Moore, and M. Smith, Eds. Berlin, Germany: Springer, 2014, pp. 72–86.
[38] G. Kambourakis, C. Kolias, and A. Stavrou, "The mirai botnet and the IoT zombie armies," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2017, pp. 267–272.
[39] S. Behal, K. Kumar, and M. Sachdeva, "Characterizing DDoS attacks and flash events: Review, research gaps and future directions," *Comput. Sci. Rev.*, vol. 25, pp. 101–114, Aug. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574013717300941

[40] M. Sachdeva, K. Kumar, and G. Singh, "A comprehensive approach to discriminate DDoS attacks from flash events," *J. Inf. Secur. Appl.*, vol. 26, pp. 8–22, Feb. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214212615000472

[41] P. Nespoli, D. Papamartzivanos, F. G. Mármol, and G. Kambourakis, "Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1361–1396, 2nd Quart., 2018.

[42] University of OULU. *PROTOS Test-Suite: C07-SIP*. Accessed: Sep. 7, 2017. [Online]. Available: https://www.ee.oulu.fi/research/ouspg/PROTOS_Test-Suite_c07-sip

[43] N. Vrakas, D. Geneiatakis, and C. Lambrinoudakis, "Is IP Multimedia Subsystem affected by 'malformed message' attacks?—An evaluation of OpenIMS," in *Proc. Int. Conf. Secur. Cryptogr.*, Jul. 2011, pp. 275–280.

[44] N. Vrakas, D. Geneiatakis, and C. Lambrinoudakis, "Evaluating the security and privacy protection level of IP multimedia subsystem environments," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 803–819, 2nd Quart., 2013.

[45] A. K. Shrestha, "Security of SIP-based infrastructure against malicious message attacks," in *Proc. 8th Int. Conf. Softw., Knowl., Inf. Manage. Appl. (SKIMA)*, Dec. 2014, pp. 1–8.

[46] H. Li, H. Lin, H. Hou, and X. Yang, "An efficient intrusion detection and prevention system against SIP malformed messages attacks," in *Proc. Int. Conf. Comput. Aspects Social Netw. (CASoN)*, Sep. 2010, pp. 69–73.

[47] H. Li, H. Lin, X. Yang, and F. Liu, "A rules-based intrusion detection and prevention framework against SIP malformed messages attacks," in *Proc. 3rd IEEE Int. Conf. Broadband Netw. Multimedia Technol. (IC-BNMT)*, Oct. 2010, pp. 700–705.

[48] A. Lahmadi and O. Festor, "Veto: An exploit prevention language from known vulnerabilities in SIP services," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2010, pp. 216–223.

[49] K. Rieck, S. Wahl, P. Laskov, P. Domschitz, and K.-R. Müller, *A Self-Learning System for Detection of Anomalous SIP Messages*. Berlin, Germany: Springer, 2008, pp. 90–106.

[50] R. Ferdous, R. Lo Cigno, and A. Zorat, "Classification of SIP messages by a syntax filter and SVMs," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2012, pp. 2714–2719.

[51] Z. Tsiatsikas, A. Fakis, D. Papamartzivanos, D. Geneiatakis, G. Kambourakis, and C. Kolias, "Battling against DDoS in SIP: Is Machine Learning-based detection an effective weapon?" in *Proc. 12th Int. Conf. Secur. Cryptogr.*, Jul. 2015, pp. 301–308.

[52] Z. Tsiatsikas, D. Geneiatakis, G. Kambourakis, and S. Gritzalis, "Realtime DDoS detection in SIP ecosystems: Machine learning tools of the trade," in *Proc. 10th Int. Conf. Netw. Syst. Secur. (NSS)*, Taipei, Taiwan, Sep. 2016, pp. 126–139.

[53] A. Akbar, S. M. Basha, S. A. Sattar, and S. Raziuddin, "An intelligent SIP message parser for detecting and mitigating DDoS attacks," *Int. J. Innov. Eng. Technol.*, vol. 7, no. 2, pp. 1–7, Aug. 2016.

[54] D. Sisalem, J. Kuthan, and S. Ehlert, "Denial of service attacks targeting a SIP VoIP infrastructure: Attack scenarios and prevention mechanisms," *IEEE Netw.*, vol. 20, no. 5, pp. 26–31, Sep. 2006.

[55] G. Zhang, S. Ehlert, T. Magedanz, and D. Sisalem, "Denial of service attack and prevention on SIP VoIP infrastructures using DNS flooding," in *Proc. 1st Int. Conf. Princ., Syst. Appl. IP Telecommun. (IPTComm)*, New York, NY, USA, Jul. 2007, pp. 57–66.

[56] R. Sparks, A. Hawrylyshen, A. Johnston, and J. R. H. Schulzrinne, *Session Initiation Protocol (SIP) Torture Test Messages*, document RFC 4475, May 2006. [Online]. Available: https://www.ietf.org/rfc/rfc4475.txt

ZISIS TSIATSIKAS received the five-year Diploma degree in information and communication systems engineering and the M.Sc. degree in security of information and communication systems from the Department of Information and Communications Systems Engineering, University of the Aegean, Greece, in 2011 and 2013, respectively, where he is currently pursuing the Ph.D. degree with the Department of Information and Communications Systems Engineering, under the supervision of Assoc. Prof. G. Kambourakis. His research interests lie in the field of SIP security and privacy.

GEORGIOS KAMBOURAKIS received the Ph.D. degree in information and communication systems engineering from the Department of Information and Communications Systems Engineering, University of the Aegean, Greece. He is currently an Associate Professor and the Head of the Department of Information and Communications Systems Engineering, University of the Aegean. His research interests are in the fields of mobile and wireless networks security and privacy, and he has more than 120 refereed publications in the aforementioned areas. He is a reviewer for a plethora of IEEE and other international journals and has served as a technical program committee member for more than 240 international conferences in security and networking. More information at http://www.icsd.aegean.gr/gkamb.

DIMITRIS GENEIATAKIS received the five-year Diploma degree in information and communication systems engineering, the M.Sc. degree in security of information and communication systems, and the Ph.D. degree in information and communication systems security from the Department of Information and Communications Systems Engineering, University of Aegean, Greece, in 2003, 2005, and 2008, respectively. He was a Post-Doctoral Researcher with Columbia University, and was a Visiting Lecturer with the Departments of Telecommunications Science and Technology and Digital Systems, University of Peloponnese and Piraeus, Greece. He was a Lecturer with the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Greece. He is currently the Scientific Project Officer with the Joint Research Center, European Commission. He has participated in various research programs projects in the area of information systems security. He has authored more than 50 refereed papers in international scientific journals and conference proceedings. His current research interests are in the areas of security mechanisms in Internet telephony, the Internet of Things, intrusion detection systems, network security, and lately in software security.

HUA WANG received the Ph.D. degree from the University of Southern Queensland, Australia. He was a Professor with the University of Southern Queensland. He is currently a full-time Professor with Victoria University. He has more than 10 years of teaching and working experience in applied informatics at both enterprise and university. He has expertise in electronic commerce, business process modeling, and enterprise architecture. As a Chief Investigator, three Australian Research Council Discovery grants have been awarded to him, since 2006, and 200 peer-reviewed scholar papers have been published. Six Ph.D. students have already graduated under his principal supervision.