Removing Ambiguities of IP Telephony Traffic Using Protocol Scrubbers

Bazara I. A. Barry
Department of Computer Science – University of Khartoum
Khartoum, Sudan

ABSTRACT

Network intrusion detection systems (NIDSs) face the serious challenge of attacks such as insertion and evasion attacks that are caused by ambiguous network traffic. Such ambiguity comes as a result of the nature of network traffic which includes protocol implementation variations and errors alongside legitimate network traffic. Moreover, attackers can intentionally introduce further ambiguities in the traffic. Consequently, NIDSs need to be aware of these ambiguities when detection is performed and make sure to differentiate between true attacks and protocol implementation variations or errors; otherwise, detection accuracy can be affected negatively. In this paper we present the design and implementation of tools that are called protocol scrubbers whose main functionality is to remove ambiguities from network traffic before it is presented to the NIDS. The proposed protocol scrubbers are designed for session initiation and data transfer protocols in IP telephony systems. They guarantee that the traffic presented to NIDSs is unambiguous by eliminating ambiguous behaviors of protocols using well-designed protocol state machines, and walking through packet headers of protocols to make sure packets will be interpreted in the desired way by the NIDS. The experimental results shown in this paper demonstrate the good quality and applicability of the introduced scrubbers.

Keywords: Protocols, Intrusion detection systems, Security, IP telephony, Protocol scrubbers.

1. INTRODUCTION

IP forms the decisive difference between circuit-switched networks and IP telephony networks. It is being used to carry voice alongside data. IP networks, which are packet-switched, break voice and data into packets that are routed to a certain destination. Upon arrival at the destination, the packets are reassembled into their original format. Contrary to circuit-switched networks, packets in packet-switched networks can travel across multiple independent paths to the final destination. This feature can benefit the network in terms of self-recovery with failed link paths because paths can be allocated dynamically. These differences between traditional circuit-switched and IP telephony networks entail changes in the infrastructure and protocols used.

Components in IP telephony infrastructure can be generally classified into servers, endpoints, and routing nodes. IP telephony servers are the components responsible for various duties aiming at maintaining the service and enhancing it such as address resolution and registration. Endpoints are the devices capable of initiating and terminating a call. Routing nodes have the capacity to connect IP networks to either other IP networks or circuit-switched networks.

The most dominant multimedia suites in IP telephony are H.323 and SIP. Both protocols are used for signaling and with them come other protocols that cater for functions other than signaling in IP telephony environments. In this paper we focus on SIP suite. Session Initiation Protocol (SIP) is a standard signaling protocol for IP telephony, and is appropriately coined as the "SS7 of future telephony." It was developed by the Internet Engineering Task Force (IETF) in RFC 2543 which was updated by RFC 3261. SIP was designed to address some important issues in setting up and tearing down sessions such as user location, user availability, and session management. The simplicity and versatility of SIP make it the choice of instant messaging, video conferencing, and multiplayer game applications among others. SIP uses other protocols to perform various functions during a session such as Session Description Protocol (SDP) to describe the characteristics of end devices, Resource Reservation Setup Protocol (RSVP) for voice quality, and Real-time Transport Protocol (RTP) for real-time transmission.

IP telephony protocols have a tendency towards openness and simplicity which gives attackers the opportunity to manipulate the protocols to their advantage and use their very features to launch attacks. Attackers can introduce ambiguous network traffic that may not be interpreted in the same way at different endpoints. Sophisticated attackers can leverage subtle differences in protocol implementations to wedge attacks past the NIDS's detection mechanism by purposefully creating ambiguous flows. In such attacks, the NIDS treats the traffic as benign, whereas the destination endpoint reconstructs a malicious interpretation. Such attacks can eventually defeat the purpose of NIDSs and turn them unusable.

In this paper we present the design and implementation of an application layer network scrubbing tool that targets the protocols SIP and RTP in IP telephony environments. Our network scrubber examines incoming traffic before it gets examined by the NIDS and removes any potential ambiguities that may hinder the NIDS's detection capabilities. The application layer scrubber picks one interpretation of the protocols and converts incoming flows into a single interpretation that is interpreted universally by all endpoints.

The rest of the paper is organized as follows: Section II discusses the related work. Section III sheds some light on Session Initiation Protocol (SIP) and Real-time Transfer Protocol (RTP) internals and message structure. Section VI shows the design of our protocol scrubber in terms of the state machines and normalization process. Section V demonstrates the implementation and experimental results. Section VI concludes the paper.

2. RELATED WORK

We start in this section with discussing insertion and evasion attacks which were mentioned briefly previously. Insertion and evasion attacks are often associated with ambiguous network traffic. Network traffic is called *ambiguous* if it is treated differently by different nodes in the network. In other words, a secondary source of information is needed by the node to interpret the traffic correctly.

A common problem that a NIDS faces, and is related to ambiguity, is discerning whether a certain packet in the traffic is acceptable to an end-system the NIDS is monitoring or not. Some of the causes of such a problem are the NIDS's lack of knowledge about the network topology, the end system's configuration, and the end system's operating system [1]. Such a problem can render a NIDS unreliable due to the misleading and less information it provides and the false sense of security it gives to security officers [1].

Insertion attacks involve the NIDS accepting packets that are rejected by end-systems. A good example to be given here is related to IP fragmentation and reassembly. A receiver reassembles incoming fragmented packets using sequence numbers and offset values. If an attacker manages to insert packets in the incoming stream, the packets will be reassembled in a way different from the expected by the end-system which should always reject such packets. Therefore, a NIDS should never accept such packets.

Evasion attacks on the other hand involve the NIDS rejecting packets that are acceptable by the end-system. Continuing with the same example above, an attacker can disrupt a NIDS causing it to miss part of the incoming traffic. Therefore, a NIDS will not be able to reassemble incoming traffic in the same way end-systems do.

Most of insertion and evasion attacks can be attributed to attackers taking advantage of wrong behaviors in the monitored protocols such as sending packets with bad header fields.

In the following, we shed some light on the related works in the area of protocol scrubbing.

Protocol Scrubbing to Counter TCP/IP Fingerprinting
TCP/IP stack fingerprinting is the process of determining the identity of a remote host's operating system by analyzing packets from that host. This process is called fingerprinting because it is similar to identifying an unknown person by taking his or her unique fingerprints and finding a match in a database of known fingerprints. Attackers can use fingerprinting to quickly create a list of targets with known vulnerabilities based on the operating system.

M. Smart, G. Malan, and F. Jahanian developed a tool called a fingerprint scrubber to remove ambiguities from TCP/IP traffic that give clues to a host's operating system. The tool was designed to be placed between a trusted network of heterogeneous systems and an untrusted connection (i.e. the Internet). It operated at the IP and TCP layers to cover a wide range of known and potential fingerprinting scans [2].

Unlike our proposed scrubber, the fingerprint scrubber is confined to counter fingerprinting at the IP and TCP layers. It neither addresses application layer protocol issues nor other insertion and evasion attacks.

Transport and Application Protocol Scrubbing
The abovementioned work was taken a step further by developing a transport scrubber that addressed the problem of transport attacks by removing protocol ambiguities, enabling downstream passive network-based intrusion detection systems to operate with high assurance.

The authors implemented an application scrubbing mechanism that allowed the creation of active, interposed intrusion detection systems that can be used to elide or modify important network protocols in real-time; effectively enabling an immediate response upon detection of severe misuse.

The application-level scrubbing mechanism was based on the FreeBSD kernel, and involved modifications to the kernel to include additions to the socket API to allow a user-level application scrubber to bind a local socket to a set of remote network addresses. This simple primitive allowed the easy creation of transparently interposed application scrubbers [3].

The above application-level scrubbing mechanism provides a general purpose platform to create application scrubbers, whereas our proposed application layer scrubber is focused on IP telephony protocols and their ambiguities.

Protocol Scrubbing Through Transparent Flow Modification
The work mentioned in the previous section was taken to the next level by implementing scrubbers for various protocols such as IP, TCP, ICMP. This variety allowed the scrubbers to cover a wider range of attacks at an acceptable level of performance [4].

However, the same approaches to implementing the scrubbers mentioned in the previous section were followed, and the same argument regarding the differences from our model applies.

Traffic Normalization That Maintains End-to-end Protocol Semantics
Mark Handley, Vern Paxson and Christian Keibrich introduced *Norm*, a traffic normalizer that sits directly in the path of traffic into a site and patches up or normalizes the packet stream to remove potential ambiguities. The result is that a NIDS monitoring the normalized traffic stream no longer needs to consider potential ambiguities in interpreting the stream.

Compared to the TCP/IP scrubbers mentioned previously, *Norm* has the distinction of attempting to develop a systematic approach to identifying all potential normalizations, and emphasizing the implications of various normalizations with regard to maintaining or eroding the end-to-end transport semantics defined by the TCP/IP protocol suite. Furthermore, it attempted to defend against attacks on the normalizer itself, both through state exhaustion, and through state loss if the attacker can cause the normalizer or NIDS to restart [5].

Our proposed scrubber has the advantage of following a similar methodology to systematically examine the ambiguities of *application layer IP telephony protocols*.

3. SIP SUITE RELATED PROTOCOLS

As mentioned in the introduction, we consider SIP suite for our discussion on the related IP telephony protocols. Specifically, we concentrate on SIP and RTP for the vital role they play in establishing, tearing down, and carrying the data of the session.

SIP Message Format

The SIP message is made up of three parts: the start line, message headers, and body. The start line contents vary depending on whether the SIP message is a request or a response. For requests it is referred to as a request line and for responses it is referred to as a status line. Figure 1 shows SIP message format.

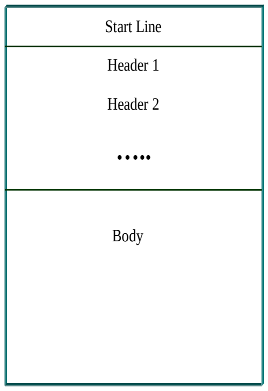

Fig. 1. SIP Message Format.

The base SIP specifications define six types of request: the INVITE request, CANCEL request, ACK request and BYE request are used for session creation, modification, establishment, and termination; the REGISTER request is used to register a certain user's contact information; and the OPTIONS request is used as a poll for querying servers and their capabilities.

Response types or codes are also classified into six classes. *1xx* for provisional/informational responses, *2xx* for success responses, *3xx* for redirection responses, *4xx* for client error responses, *5xx* for server error responses, and *6xx* for global failure responses. The *"xx"* are two digits that indicate the exact nature of the response: for example, a *"180"* provisional response indicates ringing by the remote end, while a *"181"* provisional response indicates that a call is being forwarded.

Header fields contain information related to the request like the initiator of the request, the recipient, and call identification. Some headers are mandatory in every SIP request and response. These are: To (carries the recipient of the request), From (carries the initiator of the request), Call ID (carries the unique identifier of the call), CSeq (used to identify the order of transactions), Via (contains the transport protocol and the address where the response is to be sent), Max-Forwards (used to limit the number of hops a request traverses and to avoid loops), and Contact (contains the address of the host where the request originated).

Message bodies can carry any text-based information whose interpretation is determined by request and response codes.

SIP Architecture

Elements in SIP can be classified into user agents (UAs) and intermediaries (servers). In an ideal world, communications between two endpoints (or UAs) happen without the need for servers. However, this is not always the case as network administrators and service providers would like to keep track of traffic in their network.

A SIP UA or terminal is the endpoint of dialogs: it sends and receives SIP requests and responses, it is the endpoint of multimedia streams, and it is usually the user equipment (UE) which is an application in a terminal or a dedicated hardware appliance.

SIP servers are logical entities where SIP messages pass through on their way to their final destination. These servers are used to route and redirect requests. These servers include:

1) Proxy server—receives and forwards SIP requests.
2) Redirect server—maps the address of requests into new addresses.
3) Location server—keeps track of the location of users.
4) Registrar—a server that accepts REGISTER requests.
5) Application server—an Application Server (AS) is an entity in the network that provides end users with a service.

SIP Session

Figure 2 shows the establishment of a SIP session between two users in the same domain.

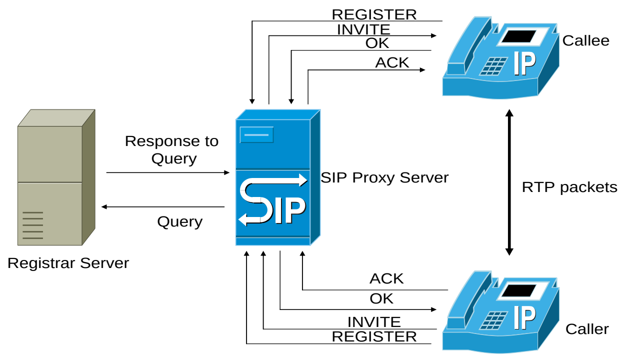

Fig. 2. Establishment of A Typical SIP Session.

When turning on their devices, both users register their availability and their IP addresses with the SIP proxy server using REGISTER request. The proxy server then sends this information to the relevant Registrar server. The caller tells the proxy server that he/she wants to contact a certain callee using INVITE request. The SIP proxy server relays the caller's invitation to the callee. The callee informs the proxy server that the caller's invitation is acceptable with OK response. The SIP proxy server communicates this response to the caller who sends ACK response establishing a session. The users then create a point-to-point RTP connection enabling them to interact. Any of the parties involved in a session can end it by sending a BYE request.

RTP Message Format

Real-time Transport Protocol (RTP) is an application layer protocol that provides end-to-end delivery services for real-time audio and video. It was developed by the Internet

Engineering Task Force (IETF) in RFC 1889 which was updated by RFC 3550. Figure 3 shows RTP message format.

Version	Padding	Extension	Contributing Source	Marker	Payload Type	Sequence Number
Timestamp						
Synchronization Source (SSRC) identifier						
Contributing Source (CSRC) identifier						

Fig. 3. RTP Message Format.

The message fields are: Version (contains the version of RTP), Padding (indicates whether the message contains padding octets or not, and may be needed by some encryption algorithms), Extension (indicates if there is an RTP header extension), Contributing Source Count (contains the number of contributing source (CSRC) IDs that follow the fixed header), Marker (interpreted by an application profile), Payload Type (identifies the payload format), Sequence Number (increments by one with each packet and is used by the receiver to reorder the packets), Timestamp (indicates the time when the first octet in the payload was sampled), Synchronization Source Identifier (identifies the source of RTP packets), and Contributing Source Identifier (if a mixer has been used, this field carries a list of sources that have contributed to the mixed media stream).

4. SIP SUITE SCRUBBER

Our scrubber enforces protocol invariants on the incoming traffic, which allows for the elimination of insertion and evasion attacks that target NIDSs. We utilize a combination of protocol state machines and packet normalizers to eliminate ambiguities in protocol behaviors and header values respectively. The result is that a NIDS monitoring the scrubbed traffic stream no longer needs to consider potential ambiguities in interpreting the stream.

Our specifications for SIP and RTP Finite State Machines (FSMs) and packet normalizers are based on RFCs 3261 [8] and 1889 [9] respectively. Request for Comment (RFC) documents provide designers and programmers with rich information regarding the operation and message flow of a certain protocol. However, RFC documents usually contain very detailed descriptions that could be time-consuming if implemented precisely. Furthermore, precise implementation of RFCs may be undesirable due to the inevitable discrepancies among different implementations of a protocol FSM. Such discrepancies could make the same traffic be classified differently by different FSMs of the same protocol. Therefore, we implement the essential details that describe a protocol in a more abstract way.

Session Initiation Protocol

For SIP, our state machine implementation is based on the base types of requests defined in RFC 3261. A certain client starts at the initial state INIT where no connection is established. An INVITE request is sent by the client if it wishes to start a call. If the client does not want to proceed with the call attempt, it can send a CANCEL request setting the state machine back to the initial state. Otherwise, an ACK message is sent following the callee's acceptance to start a call and change the state to Call Established. After call establishment, a client can send a Re-INVITE request if it wishes to move the call to another device without tearing down the session. A client can terminate a call by sending a BYE request. Figure 4 shows the above described state machine of a SIP client. For the sake of simplicity, we have not included the remaining two requests, namely, REGISTER and OPTIONS nor have we included the various types of SIP responses and message codes in the figure

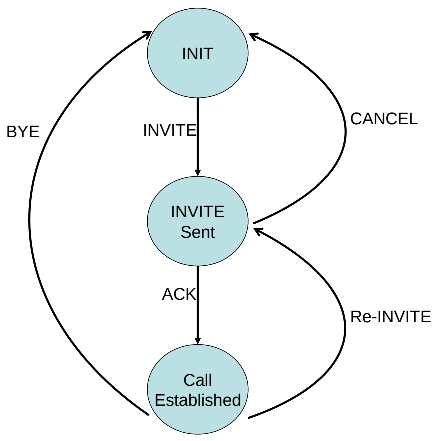

Fig. 4. SIP Simplified State Machine.

SIP packet verifier is designed to accept messages that are conformant to SIP specifications. A SIP message consists of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body. The start-line, each message-header line, and the empty line must be terminated by a carriage-return line-feed sequence (CRLF). The empty line must be present even if the message-body is not.

For SIP requests, the start line, which is referred to as the request line in this context, contains a method name, a Request-URI, and the protocol version separated by a single space (SP) character. Request-URI, which indicates the user or service to which this request is being addressed, must not contain non-escaped spaces or control characters and must not be enclosed in "<>". The SIP-Version string is case-insensitive, and includes the version of SIP in use.

For SIP responses, the start line, which is referred to as the status line in this context, consists of the protocol version followed by a numeric Status-Code and its associated textual phrase, with each element separated by an SP character. The Status-Code is a 3-digit integer result code that indicates the outcome of an attempt to understand and satisfy a request.
Each header field consists of a field name followed by a colon (":") and the field value. Table I shows the mandatory headers in every SIP request and response and their format.

It is important to note that the brackets around parameters indicate that they are optional and are not part of the header syntax. Whenever (;parameters) appears it indicates that multiple parameters can appear in a header and that semicolons separate the parameters. For the sake of simplicity, we do not mention the different requirements for messages inside or outside a dialog although they have been implemented.

Table I: Format of Mandatory SIP Headers

Header Name	Header Format	Examples and Comments
To (carries the recipient of the request)	To: SIP-URI(;parameters)	To: Carol <sip:carol@chicago.com>. The display name Carol is optional
From (carries the initiator of the request)	From: SIP-URI(;parameters)	From: Alice <sip:alice@atlanta.com>;tag=1928301774. The tag parameter contains a random string that is used for identification purposes.
Call-ID (carries the unique identifier of the call)	Call-ID: unique-id	Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com. Call-IDs are case-sensitive and are simply compared byte-by-byte.
CSeq (used to identify the order of transactions)	CSeq: digit method	CSeq: 4711 INVITE. The method must match that of the request. The sequence number value must be expressible as a 32-bit unsigned-integer and must be less than 231.
Via (contains the transport protocol and the address where the response is to be sent)	Via: SIP/2.0/[transport-protocol] sent-by(;parameters)	Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds. The protocol name and protocol version in the header field must be SIP and 2.0, respectively. The Via header field value must contain a branch parameter that is used to identify the transaction created by that request and is used by both the client and the server.
Max-Forwards (used to limit the number of hops a request traverses and to avoid loops)	Max-Forwards: digit	The value of this header field should always be 70.
Contact (contains the address of the host where the request originated)	Contact: SIP-URI(;parameters)	Contact: <sip:alice@pc33.atlanta.com>. This header field is mandatory for requests that create dialog.

It is important to note that the brackets around parameters indicate that they are optional and are not part of the header syntax. Whenever (;parameters) appears it indicates that multiple parameters can appear in a header and that semicolons separate the parameters. For the sake of simplicity, we do not mention the different requirements for messages inside or outside a dialog although they have been implemented.

Run-time Transport Protocol

For RTP, the implementation of the state machine is simpler due to the lesser number of states in the protocol state machine. A client starts at the initial state INIT where it can either receive or send packets. Upon receiving a packet, the state changes to Packet Received. Whilst at that state, a machine can either send a packet changing the state to Packet Sent, or remain at the same state receiving more packets. Similarly, at the Packet Sent state a machine can either receive packets changing the state to Packet Received, or send more packets staying at Packet Sent. Figure 5 shows the simplified RTP state machine.

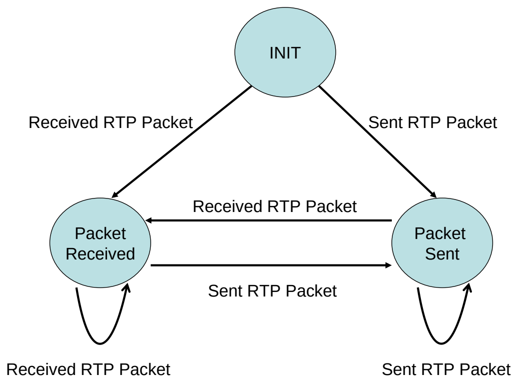

Fig. 5. Simplified RTP State Machine

RTP packet verifier follows the protocol specifications when examining packets. Table II shows the fixed header fields of RTP packets and some constraints on their lengths and values.

Table II. RTP Header Fields Sizes and Requirements

Header Name	Header Format
Version	2 bits. The version identified by RFC 1889 is 2
Padding	1 bit. If set, the packet contains one or more additional padding octets at the end, which are not part of the payload.
Extension	1 bit. If set, the RTP fixed header is followed by exactly one header extension
CSRC count (CC)	4 bits
Marker	1 bit
Payload type	7 bits
Sequence number	16 bits
Timestamp	32 bits
SSRC	32 bits
CSRC	0 to 15 items, 32 bits each. The number of items is given by the CC field. If there are more than 15 contributing sources, only 15 may be identified.

Figure 6 shows the components and a typical placement of the IP telephony protocol scrubber. The scrubber sits in front of the NIDS to normalize incoming traffic by removing its ambiguities. The scrubber has two main components, namely, the packet normalizing engine and the FSM engine. The packet normalizing engine runs normalizers that walk through packet headers to make sure that their values do not confuse NIDS. The FSM engine controls the protocol finite state machines which ensure that the packet flow does not deviate from protocol specifications which may cause ambiguity.

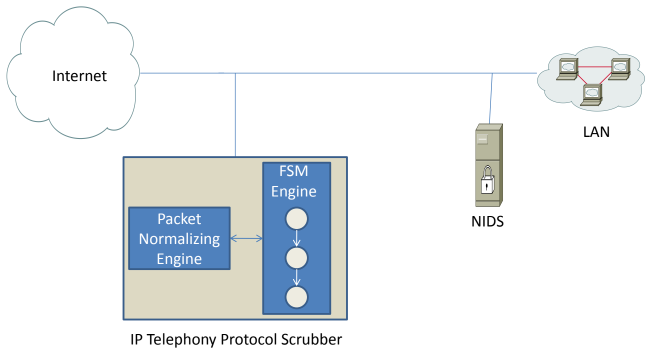

Fig. 6. Components and Placement of Scrubber.

5. IMPLEMENTATION AND RESULTS

We use OMNeT++ to implement our IP telephony scrubber. OMNeT++ is an object-oriented discrete event simulation tool that uses a modular structure. It may be used for traffic modeling of telecommunication networks, protocol modeling, and evaluating performance aspects of complex software systems among other things [6]. We use MMSim [7] which was developed by several research groups at the University of Karlsruhe to simulate multimedia protocols using OMNeT++. The MMSim model provides support for SIP, RTP, and Real-Time Streaming Protocol (RTSP).

Implementation and Testing Approach

OMNeT++ uses two programming languages, namely NED (Network Description) language and C++. NED language is used to describe the model structure and the topology of a network and its modules. A network description may consist of a number of component descriptions that can be reused in another network description, which facilitates the modular description of a network. On the other hand, C++ is used for the actual implementation of simple modules such as messages (packets) and queues. C++ is also used to implement the actual details of each protocol, where every major operation of the protocol is implemented as a member function in the class files that represent the protocol.

Ambiguity is often associated with anomalous network traffic. Our aim is to generate such traffic and present it to the scrubber which is supposed to normalize it. OMNeT++ libraries provide various functions to manipulate traffic, editing values of packet header fields, changing the order of packets, or even deleting packets. In addition, OMNeT++ provides solid support for Finite State Machines in the form of ready-to-use classes and functions. Our testing approach is based on the following steps:

1) Creating normalized traffic and capturing it in the file *f_1*.
2) Recreating the normalized traffic using the same parameters, and introducing anomalies in it.
3) Presenting anomalous traffic to the scrubber.
4) Capturing traffic normalized by the scrubber in the file *f_2*.
5) Comparing *f_1* and *f_2*.

Obviously, having similar traffic files (*f_1* and *f_2*) marks the success of our normalization process.

Experimental Setup

As can be seen from Figure 6, our simulated environment comprises an Internet and a Local Area Network (LAN). The link that connects the two networks has a 40 milliseconds delay and 0.02% packet loss.

We use the Audio/Video profile with minimal control (RTP/AVP), with UDP as the underlying protocol. An application profile describes how audio and video data may be carried within RTP. Our payload type is static with the identification number 10, and has the encoding L16. The payload type defines how a particular payload is carried in RTP. The clock rate, which is used to generate RTP timestamps, is 44100 Hz and the number of transmission channels is 2.

Endpoints on the Internet make calls to other endpoints in the LAN at specified rates that can be easily configured. We specifically use two types of rate, namely, high (10 calls per second) and low (1 call per second). Each type of load is run five times, and each run lasts for 60 minutes. The results which will be shown shortly are averaged across the different runs and taken with and without the operation of the scrubber to observe the difference.

Experimental Results

Our IP telephony protocol scrubber succeeded in normalizing ambiguous flows of traffic presented to it. The efficient implementation of the packet normalizing and FSM engines allowed the scrubber to cope with the varying amounts of traffic and provide the desired results.

We used two metrics to measure the performance of the scrubber, namely, end-to-end delay and call setup delay. End-to-end delay in IP telephony refers to the time it takes for a voice transmission to go from its source to its destination. Every element along the voice path adds to this delay. This includes switches, routers, and public Internet connections. Figure 7 shows the end-to-end delay at an endpoint in the LAN with and without the operation of the scrubber. From the figure, the average end-to-end delay with the scrubber's effect is 125 milliseconds, whereas disabling the scrubber brings the average end-to-end delay down to 123.5 milliseconds. It is obvious that the operation of the scrubber does not increase end-to-end delay beyond IP telephony acceptable levels.

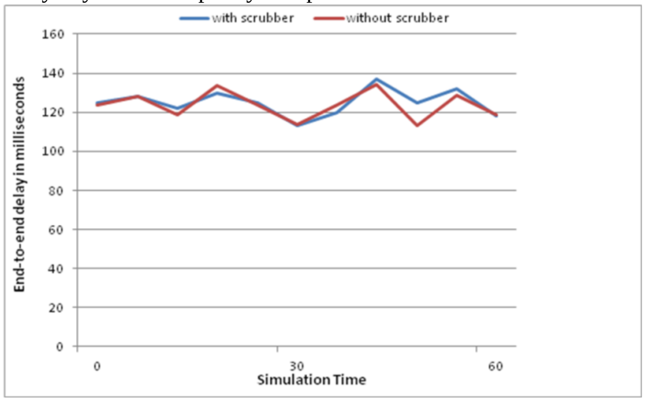

Fig. 7. End-to-end Delay.

Call setup delay in IP telephony environments is the period that starts when a caller dials the last digit of the called number and ends when the caller receives the last bit of the response. Figure 8 shows measured call setup delay for calls initiated by endpoints on the Internet to others in the LAN with and without the effect of the IP telephony scrubber. The average call setup delay without the scrubber is 252 milliseconds, whereas it is 287 milliseconds with the scrubber. Clearly, the overall call setup delay remains within IP telephony acceptable limits.

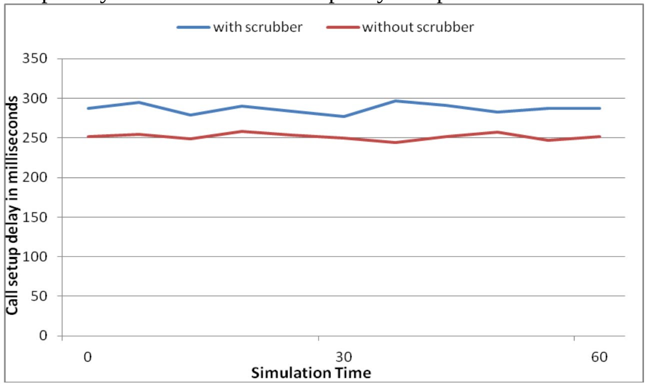

Fig. 8. Call setup delay.

6. CONCLUSION

In this paper, we have presented the design and implementation of an application layer protocol scrubber for IP telephony. The scrubber targets eliminating ambiguities in two major application layer protocols, namely, SIP and RTP. Such functionality allows network IDSs to receive and analyze unambiguous network traffic and avoid many evasion and insertion attacks among others. The scrubber has two main components, namely, a packet normalizing engine which eliminates ambiguity and ensures conformance with protocol standards in the packet header values, and a finite state machine engine that does the same to the protocol's flow of messages. The experimental results demonstrate the minor impact of the scrubber in terms of performance and the success of the traffic normalization process.

7. REFERENCES

[1] T. H. Ptacek and T. N. Newsham, "Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection", Secure Networks, Inc., Jan. 1998. Available: http://insecure.org/stf/secnet_ids/secnet_ids.html, October 2010.

[2] M. Smart, G. R. Malan, and F. Jahanian, "Defeating TCP/IP Stack Fingerprinting," Proceedings of 9th USENIX Security Symposium, Denver, Colorado, August 2000.

[3] G. R. Malan, D. Watson, F. Jahanian, and P. Howell, "Transport and Application Protocol Scrubbing," Proceedings of INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Tel Aviv, Israel, Mar 2000.

[4] D. Watson, M. Smart, G. R. Malan, and F. Jahanian, "Protocol Scrubbing: Network Security Through Transparent Flow Modification," IEEE/ACM TRANSACTIONS ON NETWORKING, vol. 12, issue. 2, pp. 261-273, April 2004

[5] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," Proceedings of the 10th conference on USENIX Security Symposium, Washington, D. C., 2001.

[6] OMNeT++ Simulator. Available: http://www.omnetpp.org, February 2010.

[7] MMSim – Simulation of Multimedia Protocols using OMNeT++. Available: http://www.ibr.cs.tu-bs.de/projects/mmsim, January 2010.

[8] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, IETF Network Working Group, June 2002.

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport Protocol for Real-Time Applications," RFC 1889, IETF Network Working Group, January 1996.