On The Performance of a Hybrid Intrusion Detection Architecture for Voice over IP Systems

Bazara I. A. Barry
University of Cape Town
Department of Electrical Engineering
Rondebosch, 7701
+27216502813
barry@crg.ee.uct.ac.za

H. Anthony Chan
University of Cape Town
Department of Electrical Engineering
Rondebosch, 7701
+27216502813
h.a.chan@ieee.org

ABSTRACT
Voice over IP (VoIP) environments pose challenging threats to Intrusion Detection Systems (IDSs). Services over VoIP systems are provided by multiple interacting protocols, each with its own vulnerabilities. This scheme could result in novel and more complex attacks, and requires cross-protocol aware IDSs. Furthermore, VoIP devices may suffer a full or partial service loss if the syntax or semantics of the aforementioned protocols are violated. Usually, a single detection approach is suited to identify a subset of the security violations to which a system is subject in VoIP environments. Therefore, a hybrid approach that combines the strengths and avoids the weaknesses of various approaches is needed. In this paper, we discuss the performance and the detection accuracy of a hybrid, host-based intrusion detection system suitable for VoIP environments. Our system has two combined detection modules, namely, a specification-based and a signature-based module. Both modules use State Machines and State Transition Analysis Techniques to model proper protocols' behaviors and potential attacks. Both modules address the issues related to syntax and semantics anomaly detection for the monitored protocols. In addition, our architecture provides a cross-protocol framework for various protocols to exchange useful detection information in real time. We implement our proposed architecture in a network simulator, alongside implementing a variety of attacks to test the credibility of the design. The implemented IDS shows an excellent detection accuracy, and low runtime impact on the performance of the VoIP system.

Categories and Subject Descriptors
K.6.5 [**Security and Protection**]: Unauthorized access

General Terms
Performance, Security

Keywords
intrusion detection, VoIP, hybrid detection, performance evaluation.

Conference name: SecureComm 2008, September 22 - 25, 2008, Istanbul, Turkey.

1. INTRODUCTION
Anderson, who introduced the concept of intrusion detection in 1980, defined an intrusion attempt or a threat to be the potential possibility of a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable or unusable [1]. Intrusion detection systems can be classified based on the detection techniques into three main types. The first one, anomaly detection, explores issues in intrusion detection associated with deviation from normal system or user behavior. Anomaly detection's philosophy is that, if we could establish a normal profile for a system, we could in principle report all system states varying from the normal profile as intrusion attempts. Anomaly detection has the advantage of detecting previously-unknown attacks but at the cost of relatively high false alarm rate, which is due to the fact that systems often exhibit legitimate but previously-unseen behavior. A main issue in anomaly detection is to select the threshold upon which we measure the deviation from the normal profile. The second detection technique is signature detection which uses attack signatures to discriminate between normal and anomalous behavior. Signature detection is capable of detecting all known attack patterns, but is of little use for unknown ones [12].

A third technique that has been overtaking both previous approaches is intrusion detection by static program analysis which was first proposed by Wagner and Dean [22]. This technique performs a static analysis of the program to create an abstract automata model of the functions and system calls. The program's behavioral specifications are used to create the model and as a basis to detect attacks. If the program executes a function or invokes a system call which violates the model, the IDS assumes that an intruder has corrupted the program. This approach has become more mature with the inputs of Balepin et al [2], and has had the name specification-based intrusion detection. Specification-based technique has the capacity to detect previously-unseen attacks with the lowest false alarm rate. This advantage is due to the programmatic nature of the IDS, which contains a model that represents all possible legal paths through the program or the protocol session, ensuring that any detected deviation from the model is not caused by the program's code but by code inserted by a bug or an attacker.

In a narrow sense, specification-based detection means looking for behavior in network traffic that is peculiar in terms of the specification for the protocol the traffic is using. In this case, detection is interested in syntax violation. In a broader sense, the

term could mean applying anomaly detection on the semantics of traffic as expressed using the protocol. In this approach, traffic is not peculiar due to a particular protocol element it is using, but rather what in aggregate it is trying to achieve with the protocol. Semantics violations are the main concern here.

Considering the resources they monitor, IDSs can also be differentiated popularly as: (a) Network IDS (NIDS) which is a dedicated monitoring component on a network, and (b) Host IDS (HIDS) which monitors a host computer only.

IP forms the decisive difference between traditional circuit-switched networks and VoIP networks. It is used in VoIP networks to carry voice alongside ordinary data. VoIP networks break voice and data into packets that are routed to a certain destination using multiple independent paths. This feature can benefit the network in terms of self-recovery with failed link paths, but has some security implications as will be shown in the implementation section. Clearly, such a difference entails changes in the infrastructure and protocols used.

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

H.323 and SIP are the most dominant VoIP multimedia suites. Both protocols are used for signaling, and with them come other protocols that cater for functions other than signaling. Signaling protocols are responsible for call setup, tear down, and modification. Media transport protocols are involved in end-to-end transport of voice and multimedia data. In addition to that, support protocols are used to enable services and features required for proper network operation. A major source of vulnerabilities in these protocols is that they transmit packet headers and payloads in clear text by default due to the absence of built-in authentication and encryption [23]. It is therefore easy for attackers to cause call termination and call flooding as well as to spoof caller ID or to exercise other attacks.

Our approach takes advantage of a combination of technologies to enhance the efficiency of intrusion detection in VoIP environments. It starts with developing host-based specifications for the protocols involved in SIP-based IP telephony, namely Session Initiation Protocol (SIP) and Real-time Transport Protocol (RTP). The tendency towards a host-based architecture is encouraged by the current shift of interest in VoIP environments from the center to the edges of the network. The specifications are derived from Request For Comment documents (RFCs) which describe the protocols. The developed specifications form the basis of our specification-based detection modules which are aided by signature-based ones to store more complex, and specific attack patterns. This hybrid approach helps to improve the efficiency of the system and lower the false alarm rate.

Both specification-based and signature-based modules combine protocol syntax and protocol semantics anomaly detection techniques. Such a feature is vital in the detection process to cover all aspects of the protocols being monitored. It allows us to report any violations of the standards in the protocol packets, alongside reporting any deviation from the expected protocol behavior during a session.

Our specification-based detection modules are developed in part based on the Communicating Extended Finite State Machines (CEFSMs) model which allows us to represent each of the involved protocols as an Extended Finite State Machine (EFSM). An EFSM is a suitable structure to model the control flow of the protocol and its data, and to spot any deviation from the expected behavior. The Communicating Extended Finite State Machines model enables a combination of stateful and cross-protocol intrusion detection. Stateful detection implies building up relevant state within a session and across sessions and using the state in matching for possible attacks. Cross-protocol detection allows the IDS to access packets from multiple protocols in a system to perform its detection. Both types of detection are important to keep the state of the session and to guard against anomalies of involved protocols.

On the other hand, our signature-based detection modules are based in part on describing an attack at the session level as a sequence of actions that the attacker performs to compromise the security of a computer system. This sequence of actions is best modeled by state transition analysis techniques which represent an attack as well defined states and transitions between these states. This representation allows signatures to complement the EFSM-based module in the specification-based component, and to overcome some of its shortcomings. It also allows the creation of semantics-aware signatures.

The rest of the paper is organized as follows. Section 2 discusses State Transition Analysis Techniques alongside State Machine models, and their use in intrusion detection. SIP suite and its security concerns are discussed in Section 3. Section 4 sheds some light on the system design. The implemented attacks used to test our system are detailed in Section 5 followed by discussing some simulation issues in Section 6. The detection and performance evaluation results are presented in Section 7 before mentioning the related works in Section 8. The paper is concluded in Section 9.

2. STATE TRANSITION ANALYSIS AND STATE MACHINES

State Transition Analysis was developed by the Reliable Software Group at University of California, Santa Barbara to represent computer penetrations. State transition analysis provides a method of representing the sequence of actions that the attacker performs to achieve a security violation. A major advantage of using this approach is its ability to foresee an incoming penetration based on the current system state. This advantage allows IDSs to limit the damage before it occurs. This important feature is shown in figure 1. The figure shows a state transition diagram of a certain attack. To successfully execute the attack, the attacker needs to reach the system state $State_0$, have Assertion 1 and Assertion 2 hold, and perform a sequence of actions represented by $State_1$ and $State_2$. If the system reaches $State_0$ and finds the specified assertions hold, the signature-based detection mechanism can raise an alarm warning about the potential impending *Compromised State*. The intermediate states $State_1$ and $State_2$ can be used to represent

variants of the penetration. A State Transition Diagram (STD) is the graphical representation that the state transition analysis uses to represent a penetration. This representation is useful for describing attacks in that it provides an interesting level of abstraction to the analyst: just above the system call and below English description [4]. This level of abstraction allows for higher-level and semantics-aware representation of the attack scenario. A state diagram can take the form of a directed graph which consists of: labeled circles to represent system states, input symbols to represent the input at each state, output symbols to represent the output from each state, edges to represent transitions between states, a start state, and a final state. State diagrams are the common factor between State Transition Analysis and State Machines.

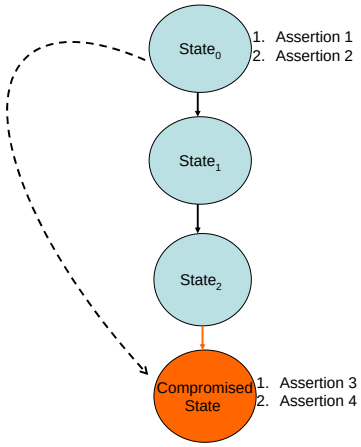

Figure 1. State Transition Diagram of an Attack.

2.1 Extended Finite State Machines

A communication protocol can be modeled as a Finite State Machine (FSM) that produces outputs on its state transitions after receiving inputs. However, FSMs are not powerful enough to model variables and operations based on variable values which form an important part of any protocol design. These limitations led to extending FSMs into Extended Finite State Machines (EFSMs).

An Extended finite state machine (EFSM) M is a quintuple:

$$M = (I, O, S, x, T)$$

where I, O, S, x, and T are finite sets of input symbols, output symbols, states, variables, and transitions respectively. Each transition t in the set T is a 6-tuple:

$$t = (s_t, q_t, a_t, o_t, P_t, A_t)$$

where s_t, q_t, a_t, and o_t are the start (current) state, end (next) state, input, and output, respectively. $P_t(x)$ is a predicate on the current variable values and $A_t(x)$ gives an action on variable values. Initially, the machine is in an initial state $s \in S$ with initial variable values: x_{init}. Suppose that at a state s the current variable values are x. Upon input a, the machine follows a transition $t = (s, q, a, o, P, A)$ if x is valid for P: $P(x)$ = TRUE. In this case, the machine outputs o, updates the current variable values by action $x := A(x)$, and moves to state q [8].

We can build elaborate systems of interacting machines by connecting the output signals of one machine to the input signals of another to form communicating extended finite state machines [3].

2.2 EFSMs in Specification-based Detection

Internet protocols can be easily modeled as EFSMs. A protocol can be viewed as a sequence of processes (states) chained by a set of events (transitions). A running protocol EFSM receives packets (input signals) through one of the available ports. Packets usually contain header fields with values (input parameters). Upon receiving a packet, a check is performed to identify the packet type (predicate), and to determine the appropriate event (transition). Some transitions represent unexpected packets, which usually occur due to network failures or an attack. Similarly, absence of expected packets, and the consequent transition on a timeout event, suggests a failure or an attack. Another source of input to a protocol state machine could be a signal sent by another protocol state machine (synchronization signals). The execution of the chosen event (transition) could result in producing and sending a packet with its header values (parameterized output signal) by a dedicated function (output parameter function). The protocol then updates its state information (context variables) by a pre-defined set of instructions (context update function).

Figure 2 shows a state transition diagram (STD) for a protocol that has three states and two transitions based on its specifications. When the state machine is in state 1, and upon receiving input signal (inp_1), a predicate is computed to choose the appropriate transition which leads to state 2. The dotted transition which leads to the attack state represents an unexpected input received at state 1. The unexpected input results in the predicate failing to enable a legitimate transition, and the machine raising a protocol violation flag.

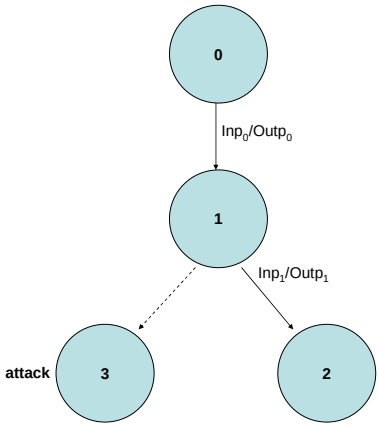

Figure 2. A state transition diagram that shows normal and potential abnormal protocol behavior.

Despite the remarkable convenience STDs provide for the EFSM model in specification-based IDSs, there are also problems with this approach in this type of detection:

1. It can model abnormal behavior as a simple and straightforward sequence of events, rather than more complex forms. This limitation will become clear when we discuss some of the complex cross-protocol attacks in the implementation section such as BYE and Re-INVITE attacks.

2. Some attacks, which are launched by abusing legitimate features of the system, can pass EFSM-based anomaly

detection without being detected. This limitation is the reason why STD-based EFSMs cannot directly detect attacks such as Denial of Service and failed logins in anomaly mode [19].

Clearly, a signature-based module that is based on a flexible state transition analysis is needed to assist specification-based module to detect such attacks.

3. SIP-BASED VOIP AND THREAT MODEL

Session Initiation Protocol (SIP) is a standard signaling protocol for VoIP, 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. Its simplicity and versatility 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 RTP for real-time transmission.

3.1 SIP Session

Figure 3 shows the establishment of a SIP session between two users in the same domain. 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.

3.2 SIP and RTP Threat Model

SIP is susceptible to the following threats and attacks:

- Denial of service: The consequence of a DoS attack is that the entity attacked becomes unavailable. DoS attacks include scenarios like targeting a certain UA or proxy and flooding them with requests.
- Eavesdropping: If messages are sent in clear text, any malicious user can eavesdrop and get session information, making it easy for them to launch a variety of hijacking-style attacks.
- Tearing down sessions: An attacker can insert messages like a CANCEL request to stop a caller from communicating with someone else. It can also send a BYE request to terminate the session.
- Session hijacking: An attacker can send an INVITE request within dialog requests to modify requests en route to change session descriptions and direct media elsewhere.
- Man in the middle: This attack is where attackers tamper with a message on its way to a recipient [14].

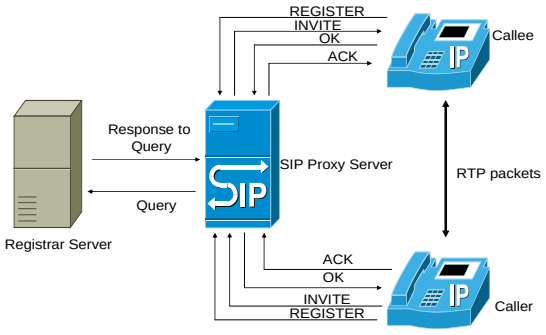

Figure 3. Establishment of a typical SIP session.

With regard to RTP, attackers can inject artificial packets with higher sequence numbers that will cause the injected packets to be played in place of the real ones [23]. Flooding with RTP packets not only deteriorates the perceived quality of service (QoS) but also may cause phones dysfunctional and reboot operations [16].

4. THE PROPOSED ARCHITECTURE

The proposed architecture of our host-based intrusion detection system is shown in figure 4. The following is a detailed description of the architecture components.

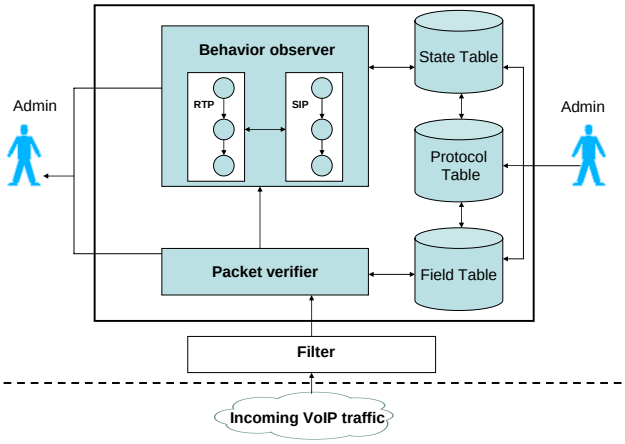

Figure 4. System Architecture.

1. *The Filter*: It classifies the incoming VoIP traffic into signaling and media packets. Currently, the filter supports SIP for signaling and RTP for media delivery.
2. *The Packet Verifier*: Its purpose is to validate compliance with protocol syntax according to standards. It checks the length of the fields, validates in terms of mandatory fields, and examines the structure of the message. This way, many unknown attacks can be detected such as attacks aiming at exploiting a vulnerability in the endpoint implementation by sending invalid protocol fields, which can lead to inadvertent leakage of sensitive network topology information, call hijacking, or Denial of Service (DoS) attacks.

3. *The Behavior Observer*: The main duty of the *behavior observer* is to guard against semantics anomalies. It performs stateful detection by keeping the Extended Finite State Machines of the protocols involved in a call. Protocol EFSMs are designed based on protocol specifications, so they can detect any deviation from normal protocol behavior. This way, the *behavior observer* can detect unknown attacks. Each protocol EFSM is provided with getter functions, so that other protocol EFSMs can get values of header fields and protocol state, which benefits detection accuracy.

4. *The Protocol Table*: This table is responsible for defining protocols at a high-level of abstraction. Each record in this table defines a specific protocol supported by the system, and each field defines a high-level attribute of the protocol. This table is meant for organizational purposes and to add some normalization to the design of the signature database. Table 1 shows an example of the content of two records from the *protocol table*. The table shows how our database defines SIP and RTP at a high level. The Protocol ID field gives each protocol a unique identifier, and is used to join various tables as will be shown shortly.

Table 1. Example of Protocol Table Contents

Field Name	Field Content 1	Field Content 2
Protocol ID	53	54
Protocol Name	SIP	RTP
Layer	Application Layer	Application Layer
Description	A protocol used for session initiation	A protocol used for real-time transmission

5. *The Field Table*: Each record in this table represents a certain field in the protocol's header and a suspicious pattern associated with it. Multiple records in this table can be used to form a signature that spans across many fields and protocols. Table 2 depicts an example of the content of the *field table*. It shows two records representing a signature that includes two SIP's header fields, namely, the *start line* and *from* fields. The signature indicates that the system should raise an alert whenever an INVITE request is received from *sip:alice@domain.com*. A field worth noting in the field table is the *Stand-Alone* field which determines whether the pattern associated with the header field forms a signature on its own, or as part of other header fields. A false in this field instructs the retrieval system to retrieve the record with the *Next Protocol ID* and *Next Field ID* to form the full signature with the current field. Null values in *Next Protocol ID* and *Next Field ID* denote the end of the retrieval process.

6. *The State Table*: Each record in this table represents a state in the protocol's EFSM. When a session reaches a certain protocol state, the IDS retrieves all the records associated with that state from the *state table*. A record could contain various values suitable for threshold detection such as the upper limit for the number of requests allowed within a specific amount of time at that state, to avoid Denial of Service saturation attacks. Furthermore, a record could contain a stored procedure to be executed upon arriving at the certain state. Such a procedure is meant to predict an impending compromise at the current system's safe state, and to limit the damage before it occurs. This strategy stems from the fact that for multi-step attacks, there are benign steps that precede the attack sequence. The administrator can provide the *state table* with the necessary procedures to be taken at the safe state that precedes the attack. It should be obvious from the aforementioned description that this table deals with input that has the perfect syntax, but is trying to achieve something that violates the semantics of the protocol. Hence, it is the semantics-based component of the database. Some examples of the contents of this table will be shown in the section on implemented attacks.

Table 2. Example of Field Table Contents

Field Name	Field Content 1	Field Content 2
Protocol ID	53	53
Field ID	1	5
Field Name	Start Line	From
Description	To distinguish requests from responses	The sender of the message
Type	String	String
Pattern	INVITE	sip:alice@domain.com
Stand-alone	False	False
Next Protocol ID	53	Null
Next Field ID	5	Null
Impact	INVITE requests from *Alice* should not be received for administrative reasons	INVITE requests from *Alice* should not be received for administrative reasons

4.1 Architecture Components Interaction

The *filter* is the first component to receive the incoming VoIP traffic. It helps classify the traffic into signaling and media packets, and forward packets to the right verifier. The *packet verifier* receives packets from the *filter* and parses them. The parsing process examines the packet in terms of its size and structure. Too big and malformed packets are rejected by the *packet verifier* in order not to deplete the processing power of the endpoint. After examining the general structure of the packet the verifier starts checking the header fields individually. It checks whether mandatory fields are present, and if their values are within the limits defined by the protocol specifications. After checking compliance with specifications for a certain field, the system retrieves all the records of the field from the *field table* to perform signature detection. If approved, packets are sent to the *behavior observer*. The *behavior observer* keeps track of the session and whether it progresses according to specifications. This session awareness is achieved by keeping an EFSM for the protocols involved to guard against any unacceptable behavior that violates proper protocol semantics. When reaching a certain state in the EFSM, the system retrieves all the records of that state from the *state table* to perform further checks on semantics violations. Clearly, detecting and reporting attacks take place in real-time.

5. IMPLEMENTED ATTACKS

We implement six attacks to demonstrate the functionality of the intrusion detection system. The attacks are launched exploiting

various vulnerabilities in SIP as a signaling protocol and RTP as media transport protocol. The implemented attacks can be classified either as flooding attacks, message flow attacks, or parser attacks. Such attacks are common in VoIP environments since current SIP specifications do not mandate authentication for all types of requests used by the protocol. Furthermore, existing security mechanisms that guarantee message integrity, confidentiality, and origin authentication can only protect against outsiders and not against insiders who abuse their privileges. The rest of this section discusses the attacks and the detection methodology for each.

5.1 The BYE Attack

As mentioned earlier, a BYE request can be sent by either the caller or the callee to terminate the session. An attacker can abuse this feature by sending this message to either the caller or the callee to fool them into tearing down the session prematurely. The User Agent that receives the faked BYE message will immediately stop sending RTP packets, whereas the other User Agent will continue sending its RTP packets. BYE attack is common in VoIP environments and can be accomplished either by sniffing the network or performing a man-in-the-middle attack to insert a BYE request into the session. Wherever there is no authentication mechanism in place, and considering the attacker's ability to discover the current session parameters, this attack can be launched successfully. BYE attack is considered a Denial of Service (DoS) attack.

Table 3. BYE Attack Signature

Field Name	Field Content 1
Protocol ID	53
State ID	30
State Name	BYE Received
Description	The system state after receiving BYE
Threshold	Null
Time Unit	Null
Timer	20 MSEC
Recommended Action	BYE_Procedure( )
Impact	Such action causes Denial of Service (DoS) at the endpoint

Although BYE attack occurs within the signaling protocol (SIP), checking the status of RTP flow in the endpoint is vital in the detection process. A genuine BYE sender will stop sending RTP packets immediately after sending a BYE message. Receiving RTP packets from the original sender on the original port after seeing the BYE message is an indicator of a BYE attack. To detect such an attack, we store a signature in the *state table* of our database. The stored signature represents the state of a SIP session upon receiving a BYE message. We set a value to the *Timer* field in the signature. The *Recommended Action* includes a cross-protocol detection procedure that checks RTP status after receiving the BYE message. If the system receives any RTP packets before the timer expires, it is an indication a BYE attack is taking place. Table 3 shows the signature. The quasi-code of BYE_Procedure( ) which is the recommended action appears in figure 5.

```
Procedure BYE_Procedure ( )
  while (Timer > 0)
    { if (RTP packets are received from original address)
        Raise_Alarm (BYE_attack)
      else
        Timer = Timer -1   }
```

Figure 5. Quasi-code for BYE attack detection.

A point worth noting is that network conditions could scupper the aforementioned strategy. If RTP packets are delayed beyond the average time after receiving a legitimate BYE request due to network congestion, our database will generate a false positive.

5.2 The Re-INVITE Attack

Another name for this attack is Call Hijacking. SIP clients use Re-INVITE message if they want to move the phone call from one device to another without tearing down the session. This feature is called call migrating. An attacker can abuse this feature by sending a Re-INVITE message to one of the parties involved in a session to fool it into believing that the other party is going to change its IP address to a new address. The new address is controlled by the attacker. This attack can be seen as a DoS attack. Furthermore, it breaches the privacy of the call since the attacker will be able to receive voice that is not meant for it. Lack of authentication enables attackers to launch this attack against endpoints.

To detect Re-INVITE attacks we use an approach similar to the one used to detect BYE attacks. Clearly, continuing to receive RTP packets from the original address on the original port after receiving a Re-INVITE denotes a call hijacking attempt. We create a signature in the *state table* denoting the system state upon receiving a Re-INVITE. Similar to the approach used in BYE attack, we set a value to the *Timer* field in the signature.

Similar to BYE attack, if a benign Re-INVITE arrives before RTP packets due to taking a different path or any other network conditions, the system will raise a false flag. Packets between two endpoints in an IP-based network are not confined to a certain route. Such a scenario is rare although it is possible.

5.3 The CANCEL Attack

CANCEL message is sent if the caller decides not to proceed with the call attempt. It asks the callee to cease processing the previous request and generate an error response designating that request. It is sent usually after receiving a provisional response from the callee. Provisional responses indicate that the request has been received, and is being processed by the callee. Without proper authentication, the receiving UA cannot differentiate a faked CANCEL message from a genuine one, which leads to a denial of communication between user agents.

Our system detects this attempt by carefully monitoring the signaling protocol behavior in the *behavior observer*. Sending a CANCEL after receiving OK response or not receiving a provisional response would be incorrect protocol behavior. Deploying our IDS prototype on all components of the network guarantees that CANCEL is sent only if a provisional response is received and any OK response is not received. This way the attack is detected early on the attacker side. This detection

methodology shows the statefulness and compliance to specifications of our system.

5.4 The REGISTER Flooding Attack

Overwhelming victim resources by flooding it with malicious traffic is the most basic and probably the most difficult to defend against DoS attack. A number of SIP clients can launch a REGISTER flooding attack to swamp a single registrar server within a short duration of time. REGISTER requests are accepted by registrar servers to store a binding between a user's SIP address and the address of the host where the user is currently residing or wishing to receive requests. REGISTER flooding attack can be viewed as a DoS attack. Even proper authentication would not stop such an attack if the attackers are insiders with bad intentions.

To detect this attack, we create a signature in the *state table* denoting the system state upon receiving a REGISTER request. Two values are set to the *Threshold* and *Time Unit* fields respectively. Whenever the number of REGISTER requests exceeds the threshold within the specified time unit, the system raises a REGISTER flooding attack flag.

5.5 Malformed Messages Attack

Attackers can create extra-long messages with fields of increased length, or huge message body. They can also omit some of the mandatory fields in the messages being sent. Such an attack targets the protocol parser at the endpoint, and aims at depleting its processing power and increasing its network utilization. Different implementations of the protocol could respond to such messages in different ways. It is likely that attackers try various malformed message combinations to discover a flaw in the end system. In addition, such malformed messages could lead some endpoints to crash which is considered a DoS situation. Malformed messages attack targets both signaling and transport protocols.

To detect such attacks the *packet verifier* provides input validation for the incoming packets before they are passed to the parser. It checks every incoming packet for adherence to the protocol specifications in terms of field presence, length, and other criteria.

5.6 Voice Injection Attack

This attack targets RTP which is used to carry call data such as voice and video. Lack of integrity checking could allow an attacker to inject an alternative RTP stream to one of the parties involved in a session. An attacker can send artificial RTP packets with higher sequence numbers than the original ones, which causes the receiver to play the artificial ones instead.

To detect such an attack we can store a signature in the *state table* to denote the system state upon receiving an RTP packet. A special procedure in the *Recommended Action* field should compare the sequence number of the packet to that of the previous one. Whenever there is an increase that exceeds the number in the *Threshold* field, an alarm is raised. Table 4 shows the signature.

A similar signature can be created with the Threshold targeting the timestamp value in the RTP packet.

6. SIMULATION ISSUES

We use OMNeT++ [11] simulator as the platform for our design. OMNeT++ is a discrete event simulation tool that uses a modular structure. It may be used to simulate nearly any kind of communication networks and distributed systems. Several research groups at the University of Karlsruhe developed MMSim [10] which is a model to simulate multimedia protocols using OMNeT++. We use MMSim to implement our design. OMNeT++ uses two programming languages, namely NED and C++. NED language is used to describe the topology of a network and its modules, whereas C++ is used for the actual implementation of the modules. In addition, OMNeT++ provides a high degree of parameterization through the use of NED and initialization files and a solid support for Finite State Machines in the form of ready-to-use classes and functions.

Table 4. Voice Injection Signature

Field Name	Field Content 1
Protocol ID	54
State ID	7
State Name	RTP Received
Description	The system state after receiving an RTP packet
Threshold	50
Time Unit	Null
Timer	Null
Recommended Action	RTP_inj_Procedure( )
Impact	Receivers play artificial stream instead of real one

6.1 Topology and Configuration

Figure 6 shows the simulated network topology. Our network comprises two domains each with a Proxy and Registrar Server. Each domain also contains a set of User Agents (endpoints) which are connected to the servers by a 10Base-T Ethernet. We use the Audio/Video profile with minimal control (RTP/AVP), with UDP as the underlying protocol. Our payload type is static with the identification number 32, and the clock rate 8000 Hz. Endpoints in a domain make calls to other endpoints in the other domain randomly and without predefined durations. Our IDS is installed on all endpoints and servers in both domains. The Internet connection between the two domains is assumed to have a delay of 40 ms and a packet loss of 0.2%. We run the experiment for 60 minutes.

6.2 Attack Implementation and Detection Using the Simulator

OMNeT++ enjoys the support of several Random Number Generators that can be configured in the initialization files. All attacks are given identification numbers, which are stored in an array-like structure. The code that launches attacks chooses a number randomly from the range of the identification numbers, and launches the associated attack accordingly. Furthermore, the attack launching code itself is activated in the endpoints based on a randomly selected number that should exceed a certain threshold. This technique guarantees that the majority of the simulated network traffic remains benign.

Message manipulation functions provided by protocol modules allow for creating malformed packets easily. The simulator library

contains various functions to set the value of different fields, and the length of the entire message.

Events in the simulator environment can be controlled to occur at a specific time. Message/event related functions can be used to send messages to other modules, schedule an event, or delete a scheduled event. This feature facilitates launching attacks that require accurate timing.

MMSim module provides interaction between SIP and RTP which makes cross-protocol detection possible. RTP protocol attributes can be captured by SIP through a specialized function that can be called from SIP module. On the other hand, C++ streams which are associated with files are used to emulate our signature database with links to specific functions that perform the recommended actions.

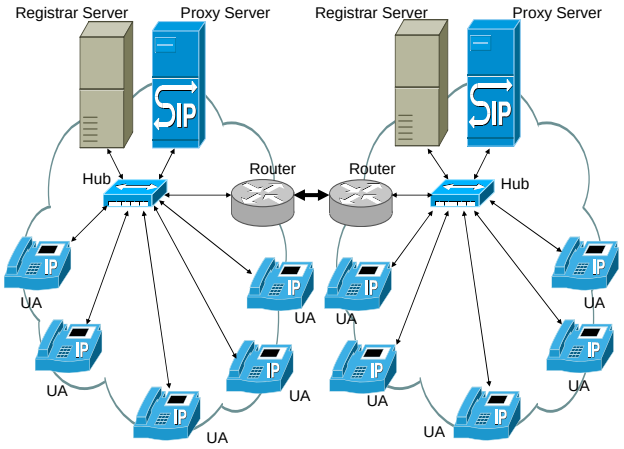

Figure 6. Simulated Network Topology.

7. RESULTS AND ANALYSIS

7.1 Detection Accuracy

Table 5 summarizes the detection accuracy results. The table shows how various components in the proposed architecture contributed to detecting *all* attacks launched during the experiment. Some of the attacks such as CANCEL and Malformed packets were unknown to the IDS prior to the experiment. We believe the more details we put in our implementation following protocol specifications, the more unknown attacks we detect, since unknown attacks are mostly protocol violations.

Table 5. Detection Accuracy Results

Attack Name	Instances	Attacks Detected	Detecting Module
BYE	7	7	State Table
Re-INVITE	4	4	State Table
CANCEL	2	2	Behavior Observer
REGISTER flooding	6	6	State Table
Malformed packets	4	4	Packet Verifier
Voice Injection	3	3	State Table

During the experiment we simulated false BYE and Re-INVITE attacks by delaying RTP packets in both after receiving a BYE message, and a Re-INVITE request respectively. Our IDS raised false flags on both occasions. We believe abnormal network conditions are to blame for these false positives, and not our detection mechanism. Delay as a result of propagation, handling, or queuing is a major issue in packet-based VoIP environments. However, our parameterized *State Table* can be used to overcome such situations. The choice of the values for timers and thresholds is left to the discretion of the system administrator. Hence, system administrators can set these values in a way that reflects the conditions of the underlying network to avoid unwanted false alarms.

7.2 Performance Evaluation

It is vital that any security measure to be implemented in a VoIP network does not impede the performance of the network or affect it badly. Quality of service (QoS) is very important to the operation of VoIP networks. The implementation of various security measures in a VoIP network can introduce some complications that can degrade QoS. These complications range from delaying call setups to delaying delivery of data packets.

In this section we evaluate the performance of the proposed architecture, and show its effect on the network. Our discussion will focus on three main issues, namely, end-to-end delay, call setup delay, and packet loss. Before we start discussing these three issues in detail, we show in figure 7 the number of call requests as captured at the Proxy Server of one of the domains.

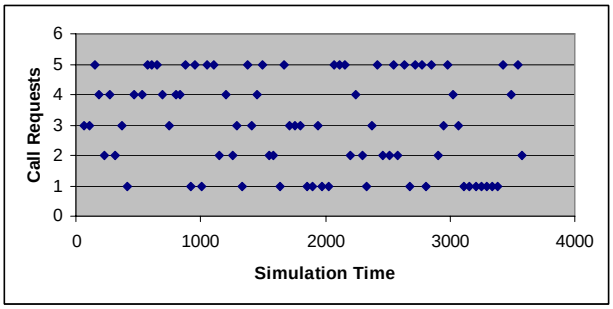

Figure 7. Call Requests at a Proxy Server.

End-to-end delay in VoIP refers to the time it takes for a voice transmission to go from its source to its destination. The ITU-T G.114 standard describes that a 150 milliseconds one-way delay is acceptable for high voice quality [5]. Every element along the voice path adds to this delay. This includes switches, routers, and public Internet connections. Figure 8 shows the end-to-end delay experienced by an endpoint in the network with and without our IDS installed. The figure shows end-to-end delay for individual RTP voice packets. Our IDS added about 2.6 milliseconds on average to the voice transmission delay. As shown in the figure, the overall delay remains considerably less than the upper bound of 150 milliseconds. The delay variation (jitter) remains at 2 milliseconds with a slight addition of $3 * 10^{-5}$ seconds by our IDS. Therefore, our IDS has a trifling impact on end-to-end delay.

Call setup delay in VoIP 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. VoIP systems are expected to give a performance comparable of that of PSTNs. Users may be annoyed with a setup process that requires more

than a few seconds. Figure 9 shows the call setup delay introduced by our IDS at a certain endpoint during the simulation. The operation of the IDS adds about 68 milliseconds to the call setup process. Such an increase in the call setup time is tolerable by VoIP users. Furthermore, the overall call setup delay remains within the limit of one or two seconds mentioned in [6].

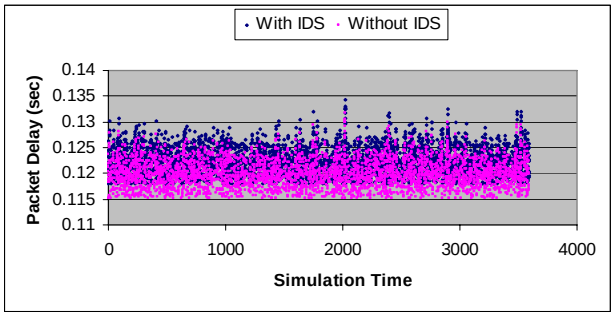

Figure 8. End-to-end delay.

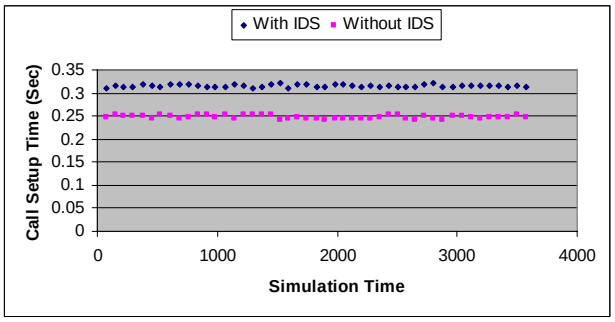

Figure 9. Call Setup Delay at an Endpoint.

Ideally, there should be no packet loss for VoIP. Losses of 3 and 4 percent could place the quality in VoIP networks encoded by certain codecs at a level below the quality of service level of PSTNs. Part of the problem in VoIP environments is their reliance on RTP which uses the unreliable UDP for transport with its unguaranteed packet delivery. Figure 10 shows the packet loss rate at servers and endpoints buffers with and without our IDS for various amounts of traffic. The packet loss rate with our IDS is only 0.02 % higher than the rate without it on average. The overall packet loss remains at 0.04% on average, which is considerably less than the 1 percent level specified by many codecs as the upper limit.

The good performance figures shown by our architecture can be attributed to two main factors: Firstly, the *Behavior Observer* implements finite state machines in switch-like statements, which makes memory management efficient. Considering the expensiveness of creating objects, there is no need in this scheme to create a new object for each transition or state in the finite state machine. Information that identifies calls uniquely can be stored at the cost of a few hundred bytes per call. This low cost allows servers to accommodate hundreds of calls simultaneously without degrading the performance of the system. Secondly, retrieving from the database requires going through only one level of hierarchy. Specification-based modules directly retrieve from *Field* and *State tables* which contain the actual signatures. In addition, Administrators can store more than one procedure in the *Recommended Action Field* of the database for a single record. Therefore, the database can store in one record multiple signatures with slight variations.

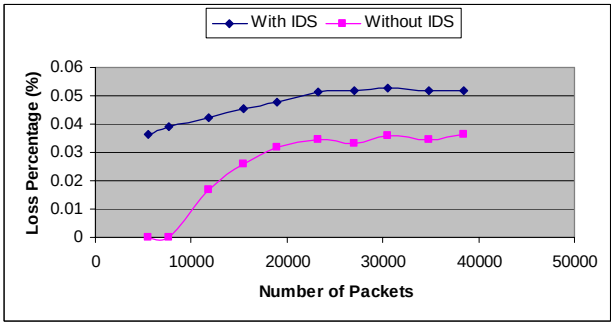

Figure 10. Packet loss.

8. RELATED WORK

Hybrid IDSs can trace their origins back to systems such as IDES [9], NADIR [7], and W&S [20]. These systems combined anomaly detection with penetration identification. However, it was difficult to establish proper behavior patterns, resulting in a relatively large number of false alarms. Using system specifications as the detection baseline in our architecture reduces the false alarm rate significantly.

STAT [4] and NetSTAT [21] adopted state transition analysis for host and network-based detection respectively. Being only signature-based limited the ability of these systems to detect new attacks. On the other hand, J. M. Orset, B. Alcalde, and A. Cavalli [13] proposed an EFSM-based IDS that uses specifications of routing protocol OLSR to detect anomalies in Ad Hoc networks. However, their solution was not complemented by a signature-based component, which made it difficult to detect attacks such as DoS attacks. These shortcomings are addressed in our architecture which has a specification-based module working in conjunction with a signature-based one.

Our signature database can be compared to systems such as Snort [17]. The Snort database design defines the lowest level of detail as an event, which is the combination of a collection of packet header and data, and an active Snort rule, called a signature. However, Snort's approach falls short of providing a basis for semantics-aware signatures at the session level. Sommer and Paxon [18] proposed adding connection-level context to signatures to reduce false positives in misuse detection. However, their aim was to complement the most common form of signature matching, which is low-level string matching, with context. Our signature database combines both types of signatures, byte-level and semantics-aware. Our semantics-awareness is based on describing attacks using state transition diagrams which allow us to represent attacks at the session level rather than lower and semantics-less levels. The lowest level of detail in our semantics-based module is the state instead of the traditional event. This feature enables our database to store a higher-level abstraction of attacks than previous works, and support more general signatures.

Several IDSs have been proposed to meet the special needs of VoIP environments. SCIDIVE [23] is a stateful, and cross-protocol IDS for VoIP. SCIDIVE can be considered a signature based detection system rather than an anomaly based system. This limitation is addressed by vIDS [16]. Instead of relying entirely on a rule database, vIDS is based on interacting protocol state

machines. This design covers the issues relating to semantics anomaly detection, while not addressing syntax anomaly detection properly. vFDS [15] is an online statistical detection mechanism designed for VoIP systems. vFDS relies on pure statistical anomaly approaches which affect its sensitivity negatively. In addition, vFDS is limited to detecting flooding attacks. Our design provides a combination of specification-based and signature-based detection techniques to bring the false alarm rate to its lowest level. It also addresses syntax and semantics-related issues to cover a wider range of attacks.

9. CONCLUSION AND FUTURE WORK

We have proposed a hybrid, host-based intrusion detection architecture that combines specification and signature-based detection techniques for VoIP systems. Our architecture caters for stateful detection, and allows protocols to exchange useful information to improve detection efficiency. Our signature-based module supports a high-level of abstraction for attacks which helps bring semantics awareness into attack description. Various experimental results show excellent detection capabilities, and low runtime impact on VoIP endpoints and servers.

VoIP environments share the same infrastructure with IP-based networks, and consequently they inherit all the security weaknesses of IP. Our future work involves applying the same techniques to lower layers such as transport and network layer.

10. REFERENCES

[1] Anderson, J. P. 1980 Computer Security Threat Monitoring and Surveillance. James P. Anderson Co. Fort Washington, PA, (April 1980).

[2] Balepin, I., Maltsev, S., Rowe, J., and Levitt, K. 2003 Using Specification-based Intrusion Detection for Automated Response. In *Proceedings of the Sixth International Symposium, Recent Advances in Intrusion Detection (RAID'03)* (Pittsburg, PA, 2003).

[3] Holzmann, J. G. 1991 *Design and Validation of Computer Protocols*. Prentice Hall, New Jersey, 166.

[4] Ilgun, K., Kemmerer, R. A., and Porras, P. A. 1995 State Transition Analysis: A Rule-Based Intrusion Detection Approach. *IEEE Transactions on Software Engineering, 21* (3). 181-199.

[5] International Telecommunication Union – Telecommunication Standardization Section Recommendation G.114: One-way Transmission Time. May 2003. Retrieved March 2008, from ITU web site: http://www.itu.int.

[6] Internet Engineering Task Force – Internet Draft: VoIP Signaling Performance Requirements and Expectations. June 1999. Retrieved March 2008, from IETF web site: http://tools.ietf.org.

[7] Jackson, K. A., DuBios, D. H., and Stalling, C. A. 1991 An Expert System Application For Network Intrusion Detection. In *Proceedings of the 14th National Computer Security Conference*, (Baltimore, MD, October 1991).

[8] Lee, D. and Yannakakis, M. 1996 Principles and Methods of Testing Finite State Machines. *Proceedings of The IEEE, 84* (8). 1090-1123.

[9] Lunt, T. F., Tamaru, A., Gilham, F., Jagannathan, R., Jalali, C., Neumann, P. G., Javitz, H. S., Valdes, A., and Garvey, T. D. 1992 A Real-time Intrusion Detection Expert System (IDES). Final Technical Report. Computer Science Laboratory. SRI International. Menlo Park. CA.

[10] MMSim – Simulation of Multimedia Protocols using OMNeT++. Retrieved January 2008, from http://www.ibr.cs.tu-bs.de/projects/mmsim.

[11] OMNeT++ Simulator. Retrieved January 2008, from OMNeT++ web site: http://www.omnetpp.org.

[12] Oppliger, R. 2002 *Internet and Intranet Security*. 2nd edition. Artech House, Norwood, MA, 374.

[13] Orset, J. M., Alcalde, B., and Cavalli, A. 2005 An EFSM-Based Intrusion Detection System for Ad Hoc Networks. In *Proceedings of The Third International Symposium, Automated Technology for Verification and Analysis (ATVA)*, (Taipei, Taiwan, October 2005).

[14] Poikselka, M., Mayer, G., Khartabil, H., and Niemi, A. 2004 *The IMS: IP Multimedia Concepts and Services in the Mobile Domain*. Wiley, Sussex, 278.

[15] Sengar, H., Wijesekera, D., Wang, H., and Jajodia, S. 2006 Fast Detection of Denial-of-Service Attacks on IP Telephony. In *Proceedings of IEEE Fourteenth International Workshop on Quality of Service*, (New Haven, CT, 2006).

[16] Sengar, H., Wijesekera, D., Wang, H., and Jajodia, S. 2006 VoIP Intrusion Detection Through Interacting Protocol State Machines. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, (Philadelphia, USA, 2006).

[17] Snort – The de facto Standard for Intrusion Detection/Prevention. Retrieved March 2008, from Snort web site: http://www.snort.org.

[18] Sommer, R. and Paxon, V. 2003 Enhancing Byte-level Network Intrusion Detection Signatures with Context. In *Proceedings of Tenth ACM Conference on Computer and Communication Security*, (Washington DC, 2003).

[19] Sundaram, A. 1996 An Introduction to Intrusion Detection. *ACM Crossroads Magazine, Special Issue on Computer Security, 2* (4), 3-7.

[20] Vaccaro, H. S. and Liepins, G. E. 1989 Detection of Anomalous Computer Session Activity. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, (Oakland, CA, May 1989).

[21] Vigna, G. and Kemmerer, R. 1998 NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the 14th Annual Computer Security Application Conference (ACSAC)*, (Scottsdale, Arizona, 1998)

[22] Wagner, D. and Dean, R. 2001. Intrusion Detection via Static Analysis. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, CA, May 2001).

[23] Wu, Y., Bagchi, S., Garg, S., Singh, N. and Tsai, T. 2004 SCIDIVE: A Stateful and Cross Protocol Intrusion Detection Architecture for Voice-over-IP Environments. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)* (Florence, Italy, 2004).