VoIP Intrusion Detection Through Interacting Protocol State Machines

Hemant Sengar† Duminda Wijesekera† Haining Wang ‡ Sushil Jajodia†

†Center for Secure Information Systems
George Mason University
Fairfax, VA 22030, USA
{hsengar,dwijesek,jajodia}@gmu.edu

‡Department of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
hnw@cs.wm.edu

Abstract

Being a fast-growing Internet application, Voice over Internet Protocol (VoIP) shares the network resources with the regular Internet traffic, and is susceptible to the existing security holes of the Internet. Moreover, given that voice communication is time sensitive and uses a suite of interacting protocols, VoIP exposes new forms of vulnerabilities to malicious attacks. In this paper, we propose a highly-needed VoIP intrusion detection system. Our approach is novel in that, it utilizes not only the state machines of network protocols but also the interaction among them for intrusion detection. This detection approach is particularly suited for protecting VoIP applications, in which a melange of protocols are involved to provide IP telephony services. Based on tracking deviations from interacting protocol state machines, our solution shows promising detection characteristics and low runtime impact on the perceived quality of voice streams.

1 Introduction

IP telephony, commonly known as *Voice over IP* (VoIP), is emerging as a viable alternative to traditional telephone systems. As the popularity of VoIP and its deployment grows, it will become the target of hackers and crackers. VoIP suffers threats from many different protocol layers. Malicious attackers may exploit the misconfiguration of devices, the vulnerability of the underlying operating systems, and protocol implementation flaws to break in. Well-known attacks of data networks such as worms, viruses, Trojan horse, denial-of-service (DoS) attacks can also plague VoIP network devices [16].

An Intrusion Detection System (IDS) helps administrators to monitor and defend against security breaches. Intrusion detection techniques are generally divided into two paradigms, *anomaly detection* and *misuse detection*. In anomaly detection techniques, the deviation from normal system behaviors is detected, whereas misuse detection is based on the matching of attack signatures. Unlike signature-based intrusion detection, anomaly detection has the advantage of detecting previously-unknown attacks but at the cost of relatively high false alarm rate. Sekar et al. [15] introduced a third category of *specification-based* intrusion detection. Specification-based approach takes the manual development of a specification that captures legitimate system behavior and detects any deviation thereof. This approach can detect unseen attacks with low false alarm rate. Based on the state transition analysis, Vigna et al. proposed NetSTAT [18] and WebSTAT [19] intrusion detection systems. As a popular network-based intrusion detection system, Snort [11] monitors network traffic between trusted devices and the untrusted Internet, and inspects packets by signature matching. However, these previous approaches fall short of defending VoIP applications, because of the cross-protocol interaction and distributed nature of VoIP.

VoIP systems use multiple protocols for call control and data delivery. For example, in SIP-based IP telephony, Session Initiation Protocol (SIP) [12] is used to control call setup and teardown, while Real-time Transport Protocol (RTP) [14] is for media delivery. A VoIP system is distributed in nature, consisting of IP phones, SIP proxies, and many other servers. Defending against malicious attacks on such a heterogeneous and distributed environment is far from trivial. Recently, Wu et al. [20] proposed a stateful and cross-protocol intrusion detection architecture for VoIP, in which protocol dependent information is assembled from multiple packets and aggregated states are used in the rule-matching engine. Different from their approach, in this paper, we propose a VoIP IDS based on protocol state machines. Specifically, instead of collecting and deriving the call state and protocol dependent information from the packets, our approach utilizes the state transitions made in the protocol state machines for intrusion detection. These

transitions are triggered either by the arrival of packets or the internal interaction between protocol state machines. Our approach of incorporating the interaction between protocol state machines is particularly suited for intrusion detection in VoIP. Call control and media delivery protocols are synchronized by exchanging synchronization messages for critical events throughout the established sessions.

A protocol state machine based IDS can be considered as a variant of anomaly detection mechanism, which classifies a deviation from normal behavior as a suspicious attack. The protocol state machine is built from specification, and is used to derive legitimate states and transitions. After the protocol state machine is constructed and the related attribute features are identified, this protocol state machine based approach not only lowers the number of false alarms but also is capable of detecting previously unseen attacks. To validate its effectiveness, we evaluate the proposed VoIP Intrusion Detection System (vIDS) using our VoIP network testbed and OPNET network simulator. Our experimental results demonstrate that on average the online placement of vIDS induces the additional delay of $\simeq$ 100 ms to call setup time. The average increase of CPU overhead induced by vIDS is only $\simeq 3.6\%$. Although the associated memory cost is proportional to the number of calls to be monitored, as with each call, only one instance of a protocol state machine is maintained at the memory. Once the calls have successfully reached the *final state*, the corresponding protocol state machines will be deleted from the memory. More importantly, the memory cost per call is insignificant. Therefore, vIDS can monitor thousands of calls at the same time and its memory cost can be easily afforded by a modern computer.

The remainder of this paper is structured as follows. Section 2 briefly describes SIP protocol and its usage in enterprise networks. Section 3 presents the threat models of SIP and RTP. Section 4 presents the extended finite state machines and its application in developing vIDS. Section 5 details the architecture of vIDS. Section 6 describes how to detect the SIP and RTP related attacks using protocol state machines. Section 7 conducts the performance evaluation of vIDS. Section 8 surveys related work. Finally, Section 9 concludes the paper.

2 Background

We first present the SIP-based IP telephony. Then, we describe its usage in enterprise networks and illustrate the placement of vIDS in such a network.

2.1 SIP-based IP Telephony

SIP is a standard signaling protocol for VoIP, and is appropriately coined as the "SS7 of future telephony" [6]. SIP is a text-based application level protocol to setup, modify, and teardown multimedia sessions between one or more participants. SIP messages can be transmitted over UDP or TCP, but UDP is preferred over TCP because of its simplicity and lower transmission delays. SIP architecture identifies two basic types of components, *user agents* (i.e. phones) and *SIP servers* (i.e. Location, Redirect, Registrar and Proxy servers). Each UA is a combination of two entities, the *user agent client* (UAC) and the *user agent server* (UAS). The UA switches back and forth between being an UAC and an UAS.

The SIP messages are classified into two groups: *requests* and *responses*. The SIP requests are also called methods, and there are six of them (`INVITE`, `ACK`, `BYE`, `CANCEL`, `REGISTER` and `OPTIONS`) described in [12]. Other methods are proposed as the extensions of the original six methods. For each request of an UAC, UAS or SIP server generates a SIP response. Each response message is identified by a numeric status code (similar to HTTP response messages).

Now, we use an example of a typical call setup flow to highlight the usage of SIP request and response messages between user agents UA-A and UA-B. Suppose that the two UAs belong to different domains that have their own proxy servers. UA-A calls UA-B using its SIP phone over the Internet. The outbound proxy server uses the Domain Name System (DNS) to locate the inbound proxy server at the other domain. After obtaining the IP address of inbound proxy server, the outbound proxy server of UA-A sends the `INVITE` request to the domain of UA-B. The inbound proxy server consults a location service database to find out the current location of UA-B and forwards the `INVITE` request to UA-B's SIP phone. Exchanging `INVITE/200 OK/ACK` messages completes the three-way handshake and establishes a SIP session. A set of parameters are exchanged via SIP messages (in the message body using Session Description Protocol (SDP) [2]) between the two end points before a RTP-based voice channel is established. In general, the path of media packets is independent of that of the SIP signaling messages. The signaling protocol remains oblivious to the changes made to the media path during the life span of the call, unless it is explicitly requested through a `re-INVITE` message. At the end of the call, UA-B (or UA-A) hangs up by sending a `BYE` message. Subsequently, UA-A (or UA-B) terminates the session and sends back a `200 OK` response. This example shows the basic functionality of SIP, detailed description of the SIP operations is in RFC 3261 [12].

2.2 Enterprise IP Telephony

Figure 1 illustrates a SIP-based IP telephony enabled enterprise network and the placement of vIDS inside the

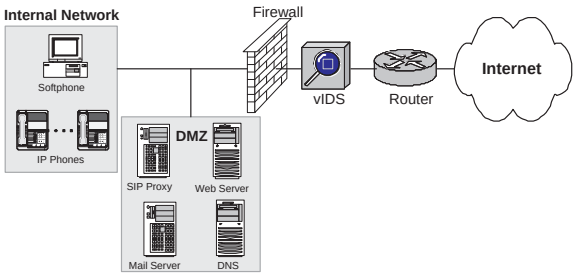


Figure 1. Enterprise IP Telephony Network

network. The enterprise network consists of an internal network and a de-militarized zone (DMZ). DMZ contains many servers, including the SIP proxy server that is accessed by the internal network and the public Internet. Under the assumption that most VoIP related security threats are from the outside of the enterprise network, the online vIDS is located strategically between the edge router and the firewall, monitoring all traffic traveling to and from both DMZ and the internal network to the Internet. Note that SIP proxy server has no media capability and only facilitates the two end points (i.e. IP telephones) to discover and contact each other through SIP signaling. After the end points have been located, the media is directly delivered from end-to-end, without going through the proxy. Thus, it is necessary for vIDS to see the signaling messages and the media flows for all SIP clients. In Figure 1, the firewall and vIDS devices are shown as separate components. In practical deployment, vIDS can be installed at the firewall, and is capable of analyzing traffic streams at the application layer.

3 The Threat Model

In this section, we describe various malicious attacks targeting at different protocol layers of IP telephony services. Our focus is mainly in detecting SIP and RTP based attacks.

3.1 SIP-based attacks

SIP stacks can be found on desktops, laptops, VoIP phones, mobile phones, and wireless devices. SIP is a simple but efficient call control protocol involving many intermediaries and multi-faceted trust relationships among them. Some devices trust each other, whereas others do not. Attackers have a range of target devices, starting from end devices to Routers, Switches, Signaling Gateways, Media Gateway Controllers, and SIP Proxies. Any device in the path from caller to callee can be an attacker's target. Several DoS and fraud attacks against SIP-based VoIP systems have been described in [1, 8, 13]. A great deal of the discussion of possible attacks centers around an assumption of lack of proper authentication. However, many attacks are still possible to be launched by an authenticated but misbehaving UA. In the following, we describe some of these attacks, which require proper analysis of attack patterns for detection.

`CANCEL` Denial of Service Attack: The `CANCEL` method is used to terminate pending searches or call attempts. Specifically, it asks the UAS to cease the request processing and generate an error response to the request. A `CANCEL` is for an outstanding `INVITE`, since `INVITE` is the only request taking some time (few seconds) to complete. All other SIP requests are responded immediately. It can be generated either by a UA or the proxy servers that have received 1xx responses but still wait for the final 2xx responses. Note that without proper authentication, the receiving UA cannot differentiate the spoofed `CANCEL` message from the genuine one, leading to the denial of the communication between UAs.

`BYE` Denial of Service Attack: When a caller has received the response message `200 OK`, the session is considered as established. An established media session is terminated upon receiving a `BYE` message. It is an end-to-end message sent by UAs participating in the session. The `BYE` attack aborts an established call between UAs. For example, UA-A and UA-B have established a call (i.e. `200 OK` and `ACK` messages are already exchanged), suddenly malicious UA-C sends a `BYE` message to either UAs, A or B. The receiving UA will prematurely teardown the established call assuming that it is requested by the partner UA.

`INVITE` Request Flooding Attack: IP phones have the capability of generating multiple calls at the same time but can only support a few. A number of IP phones together may launch an `INVITE` flooding attack to overwhelm a single telephone terminal within a short duration of time. Proxy servers are also susceptible to `INVITE` flooding attacks.

There are various other attacks, which can also exploit the SIP signaling messages. For example, in a call hijacking attack, a new `INVITE` request could be send within a pre-existing dialog. Billing and toll fraud can be realized if one end sends a `BYE` message to stop billing but continues sending RTP packets. Distributed Reflection DoS (DRDoS) attack on a victim is also possible. If spoofed requests are sent to a large number of SIP proxy servers (i.e. reflectors) on the Internet with the victim's IP address as the source of the requester, the victim will be swamped with the subsequent response messages, thereby causing a DRDoS attack.

3.2 RTP-based attacks

The threats resulting from the vulnerabilities of the media path are described as follows:

Media spamming: A SDP description conveys media at-

tributes preferred by the caller for call setup. The caller lists the preferred media capabilities in SDP and sends in either `INVITE` or `ACK` messages. In response to `INVITE`, the called party lists its own media capability in the `200 OK` message. A third party knowing the SDP information (i.e. IP address, port number, media type and its encoding scheme etc.) and the RTP synchronization source (SSRC) identifier could fabricate RTP packets. The SSRC identifier is used to identify the corresponding participant within an RTP session. By having the same SSRC identifier with higher sequence number or timestamp in the spoofed RTP packets, the third party can play unauthorized media.

RTP packets flooding: During the setup of a media session, information such as media transport protocol, media encoding, sampling rate, and port number are exchanged. The calling party should transmit the media stream according to the negotiated media encoding scheme. Changing the encoding scheme or flooding with RTP packets not only deteriorates the perceived quality of service (QoS) but also may cause phones dysfunctional and reboot operations.

4 A Formal Model

A state machine provides a low level abstraction of a protocol. It can express the protocol design in terms of desirable or undesirable protocol states and state transitions. The formal model of a communicating finite state machine plays an important role in the formal validation of protocol, protocol synthesis, and its conformance testing [4]. In this section, we present a formal definition of extended finite state machine and its application for intrusion detection in VoIP systems. We construct communicating finite state machines by connecting the output of one machine to the input of another machine. This presents a powerful representation for describing various interacting protocols involved in IP telephony services.

4.1 Extended Finite State Machine

A Mealy (finite state) machine extended with input and output parameters, context variables, operations and predicates[1], is referred as an extended finite state machine (EFSM) [10]. EFSMs are often used to model communication system behaviors. Parameters, variables, predicates and operations are used to describe the data flow and the context of the communication. The underlying FSM describes the control flow of the system. For detailed description of EFSM, see [7, 10, 15].

Definition 1. An extended finite state machine (EFSM) M is a quintuple $M = (\Sigma, S, \overrightarrow{v}, D, T)$, where:

- Σ is an event alphabet of the EFSM, each event does have name and its argument.
- S is a finite set of states, including *start state* and *final state*.
- $\overrightarrow{v}$ is a vector of finite set of state variables, $\overrightarrow{v} = (v_1, v_2, ..., v_i, .., v_k)$.
- D is a set of domain values for state variables, $D = (d_1, d_2, ..., d_i, ..d_k)$, where d_i denotes the domain value for the variable v_i.
- T is a transition relation: $S \times D \times \Sigma \longrightarrow (S, D)$.

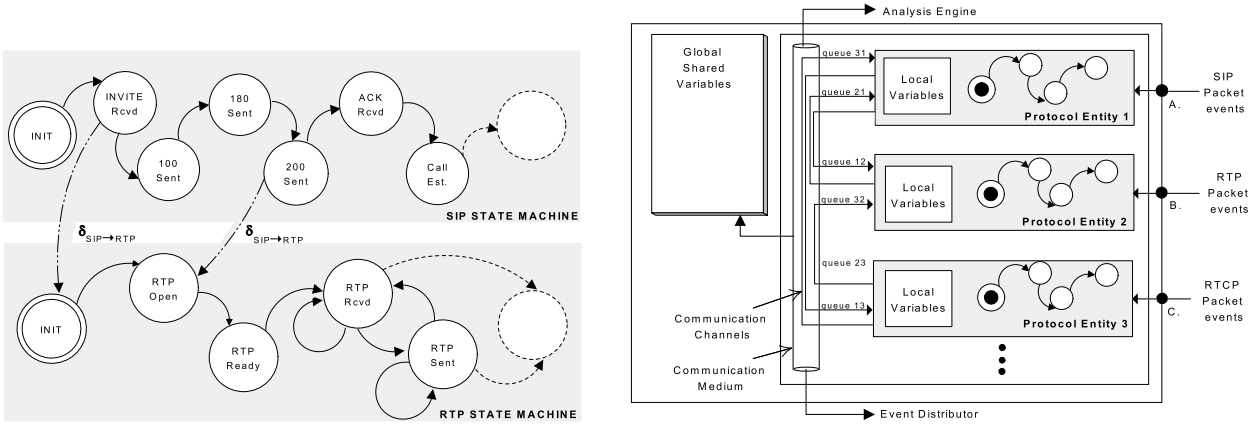
Each transition t in the transition relation set T (i.e. t $\in T$) is a tuple $< s_t, event, P_t, A_t, q_t >$, where $s_t, q_t \in S$ are the beginning and ending states of the transition respectively, *event* $\in \Sigma$ is an event identifier (i.e. name) for the transition with input vector $\overrightarrow{x}$ (i.e. arguments), $P_t(\overrightarrow{x} \times \overrightarrow{v})$ is a predicate on the valuation of input vector and current state variables, and $A_t(\overrightarrow{v})$ defines an action on state variables. Predicate P on the values of state variables $\overrightarrow{v}$ and input parameter values $\overrightarrow{x}$, should return either $FALSE$ or $TRUE$, i.e. $P(\overrightarrow{x} \times \overrightarrow{v}) \rightarrow \{TRUE, FALSE\}$. A context update function $A(\overrightarrow{v})$ is an assignment: $\overrightarrow{v} := A(\overrightarrow{v})$, which changes the current state variable values of vector $\overrightarrow{v}$ before moving to a new state.

To distinguish between the input and output events of the transitions, we use CSP notations [3]. For example, $c?event(\overrightarrow{x})$ denotes an input event with an identifier *event* and $\overrightarrow{x} = (x_1, x_2, ..., x_n)$ as event parameters. Similarly, an output event is denoted by $c!event(\overrightarrow{x})$. Here c is a particular type of channel through which data values are passed along. It may represent SIP or RTP channels, where packet events are received and an internal buffer between protocol state machines (see Figure 2(b)), where synchronization message events are queued. The transition edges between the nodes of the EFSM directed graph can be represented as $< s_t, c?event(\overrightarrow{x}), P_t, A_t, q_t) >$ or $< s_t, c!event(\overrightarrow{x}), P_t, A_t, q_t >$. Now suppose that all the transitions $< s_t, c?event(\overrightarrow{x}), P_t, A_t, q_t >$, where $1 \leq t \leq r$, begin with the same start state and event, then nondeterminism may arise. However, if we assume that the predicates are mutually disjoint, i.e. $P_i(\overrightarrow{x} \times \overrightarrow{v}) \cap P_j(\overrightarrow{x} \times \overrightarrow{v}) = \emptyset$, $1 \leq i \neq j \leq r$, then it will be a deterministic EFSM. In a deterministic EFSM, there is at most one transition to follow at any moment.

4.2 VoIP IDS Specification

Protocol processes can often be modeled as a collection of communicating finite state machines. In order to maintain the temporal order of the events, the protocol entities

[1]The predicates are defined over context variables and input parameters.



(a) Interaction between SIP and RTP (Call setup request)

(b) Communicating EFSM (per call basis)

Figure 2. VoIP IDS Specification

must exchange some data values and synchronization messages [21]. To model security violations, out of all available attributes, only a fraction of attributes are needed to represent state transitions. The selection of attributes not only depends upon the protocol entities but also the security violation we want to capture. In this paper, the selected attributes are represented as a vector $\overrightarrow{v}$. For an EFSM M, a state s_i and a valuation of the state variable vector $\overrightarrow{v}$ constitute a so called *configuration* of the EFSM. Assume that the corresponding vector values are implicitly defined, without loss of generality, the configurations can be viewed as states. We are interested in the configurations that are reachable from the initial or intermediate configuration to the attack configuration through zero or more intermediate states. The paths along the transitions from s_i to s_{attack} constitute *attack patterns*. The memory cost of characterizing an instance of a protocol state machine for a particular call depends upon *configuration* (i.e. $(s_i, \overrightarrow{v})$). In the following, we describe the relevant features of EFSM and its application to vIDS.

A transition relation t (i.e $t \in T$) between states s_i and s_j is annotated as an attack signature, if s_j corresponds to s_{attack}. The predicate P_t operates over current state variables and input parameters, which are carried in the message header or body, and consequently returns a boolean value. In vIDS, P_t describes some security aspects of a call, e.g. the identification of communicating users (*To:*, *From:*), their locations (*source IP*, *destination IP*), and the traffic characteristics (*encoding scheme, call ID, sequence numbers*). P_t captures the security aspects by putting constraints on the allowed event parameter values ($\overrightarrow{x}$) and comparing them with corresponding state variable values ($\overrightarrow{v}$). The *event* is either an arrival of a data packet or a synchronization message from co-operating protocol entities. Update function A_t modifies the current values of state variables before moving to the next state s_j.

Each protocol entity maintains a vector $\overrightarrow{v}$ of locally and globally accessible variables. Local variables are related to one particular protocol state machine, whereas global variables contain the values that are relevant to other protocol machines as well. For example, Figure 2(a) shows how SIP and RTP protocol state machines undergo through normal transitions, after receiving an `INVITE` request for the call setup from a remote end. At the (INIT) state, the arrival of an `INVITE` message brings input vector $\overrightarrow{x}$, composed of relevant header fields and message body values. The SIP protocol state machine, after receiving the `INVITE` request, parses the SIP header and SDP message body. The header field values, such as *Call-ID* and *branch* parameters in the `Via` header field and *tag* parameter values in the `From` and `To` fields are stored in the local state variables indicated by $v.l\_variable\_name$. The media information contained in the SDP message body, such as IP address, port number of the source, and offered media encoding schemes, are available to RTP protocol machine by writing them into the global shared variables that are represented as $v.g\_variable\_name$.

The (INIT) state of a SIP protocol state machine makes a transition t, $\langle$`INIT`, `SIP_Packet`, $true$, A_t, `INVITE Rcvd`$\rangle$ to the (INVITE Rcvd) state, and sends a synchronization message (i.e. c!$\delta_{SIP \rightarrow RTP}$) to the RTP state machine. The transition's update function A_t initializes vector $\overrightarrow{v}$ (both local and global state variables) with the corresponding values of input vector $\overrightarrow{x}$. The synchronization messages are transmitted through the communication channels between protocol entities (see Figure 2(b)). We assume that these communication channels are reliable and function as FIFO queues. The FIFO queue associated with the communication channel between protocol entity `1` and protocol entity `2` is represented as $queue_{12}$. The synchronization events waiting in a FIFO queue have higher priority

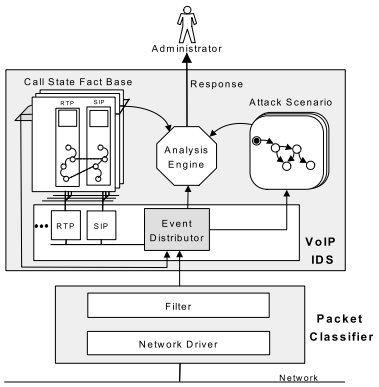


Figure 3. VoIP IDS Architecture

than the data packet events.

On receiving a synchronization event from the communication channel, the RTP machine makes a transition from the (INIT) state to the (RTP Open) state. At this state, the RTP state variables are initialized with the media information contained in the `INVITE` message of SDP. Figure 2(b) illustrates the communication channels between the protocol entities and the associated queues. Without loss of generality, we only present the states and transitions of both SIP and RTP protocol state machines after receiving an `INVITE` message from the remote caller's end. For other SIP messages or RTP packets involved during the call setup phase, the dynamics of protocol state machines are similar to those shown in Figure 2(a).

5 vIDS Architecture

VoIP services involve many different protocols. Therefore, to achieve a holistic view of an ongoing session, it is important to pay special attention to the cross-protocol interactions among protocol state machines. Figure 3 shows the proposed vIDS architecture and its components.

As shown in Figure 3, vIDS sits on top of *Packet Classifier*. VoIP intrusion detection is performed through state transition analysis of the state machines. It can detect both known and unknown attacks. The protocol specification-based state machines allow us to detect any deviation from normal system behaviors, and hence, capture unknown attacks. The vIDS component, *Call State Fact Base*, stores the control state and its state variables and keeps track of the progress of state machines for each ongoing call. The state information is updated by the arrival of packets from the *Event Distributor* component. The *Attack Scenario* component is a collection of known attack patterns, including the intermediate states and transitions that lead to attack states. vIDS conducts the state transition analysis of packet streams on call by call basis. All the packets belonging to one particular call are assigned to one group. In the group, packets are further classified into subgroups based on the specific protocols. While RTP packets trigger the transitions in RTP state machine, SIP state machine transitions are caused by the arrival of SIP packets. Although both state machines run in parallel, they are synchronized through the shared global variables and the internal synchronizing message events between protocol state machines. The *Event Distributor* component further classifies the received packets into the session and protocol dependent groups with the help of *Call State Fact Base*, and then distributes to the corresponding protocol state machine. The *Analysis Engine* component receives packets from *Event Distributor* and state information from *Call State Fact Base* or *Attack Scenario*. When protocol misbehavior (e.g. deviation from protocol specification based state machines) or attack scenario match (i.e. a transition leading to an attack state) happens, vIDS raises an alert flag and notifies administrators for further analysis.

6 Attack Detection Patterns

People may have a concern that the development of detailed protocol state machines could be a complex and time consuming process, therefore the proposed approach may not be a practical solution for intrusion detection. Fortunately, SIP-based call setup and teardown process can be easily captured in a protocol state machine. Even if it is not trivial to derive a protocol state machine, it is straightforward to develop attack scenarios for known attacks. In this section, we are particularly interested in developing attack patterns (or signatures) for some VoIP attacks discussed in Section 3.

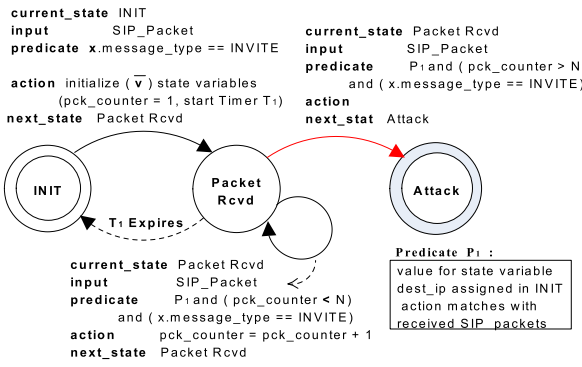


Figure 4. `INVITE` flooding attack

`INVITE` Request Flooding Attack: Figure 4 shows the state transition diagram for the detection of `INVITE` request flooding attacks. At the (`INIT`) state, the predicate checks the *message_type* of the received message. Each SIP message brings along with an input vector of values

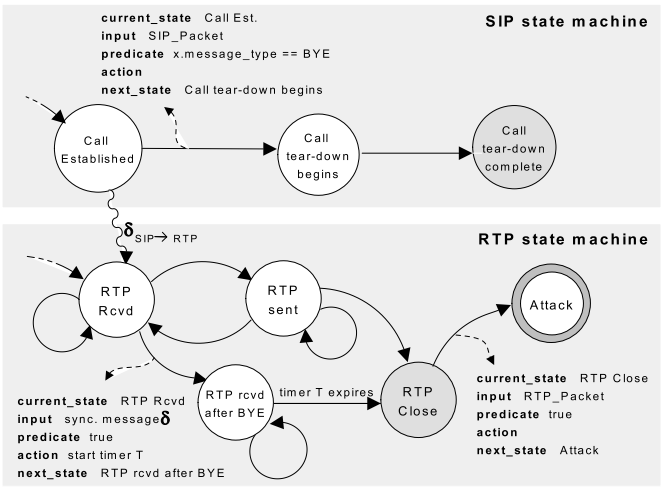


Figure 5. `BYE` **DoS attack**

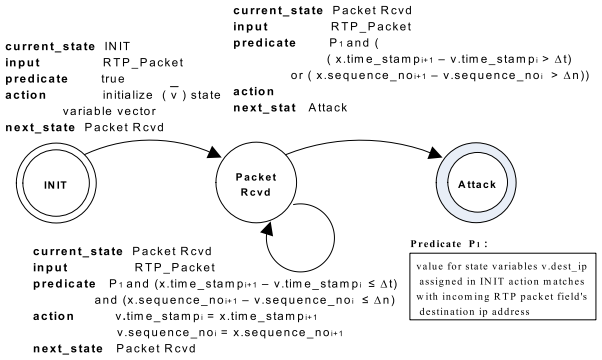


Figure 6. Media Spamming Attack

$\overrightarrow{x}$. On sniffing the first `INVITE` request received from x.src_ip (S) and destined for x.dest_ip (D), the state machine makes a transition from the (`INIT`) state to the intermediate state (`Packet Rcvd`). During this transition, state variable vector $\overrightarrow{v} = (v_1, v_2, ..., v_i, .., v_k)$ is initialized and assigned by update function A_t to the corresponding values of input vector $\overrightarrow{x}$, i.e. $A_t(v_i) \Rightarrow v_i := x_i$. It also starts a counter ($pck\_counter$) to count the received `INVITE` messages for the same destination within a certain amount of time (T_1). Timer T_1 sets the time window, under which `N` received `INVITE` requests are considered as normal. The setting of threshold `N` depends upon the up-limit that a particular type of a phone can handle. If there is a sudden surge of `INVITE` requests that exceeds the threshold `N`, it is a strong indication of a flooding attack.

`BYE` Denial of Service Attack: The proposed vIDS can detect this kind of attacks by checking cross-protocol interaction between SIP and RTP. Figure 5 shows the partial SIP and RTP state machines and their cross-protocol interactions. At the (`Call Established`) state after receiving a `BYE` message, SIP state machine makes a transition to (`Call tear-down begins`) state. Before this transition occurs, a synchronization message $\delta_{SIP \rightarrow RTP}$ is sent to RTP state machine. On receiving δ synchronization message, (`RTP Rcvd`) state makes a transition to the intermediate (`RTP rcvd after BYE`) state. At this state, timer T is also started for all in-flight RTP packets to arrive. The value of T should be small enough, since the genuine UA will stop sending RTP packets as soon as the `BYE` request is passed to the client transaction. After the expiration of T, (`RTP rcvd after BYE`) state makes a transition to (`RTP Close`) state. At this state, if there are still incoming RTP packets, it is an indication of a `BYE` DoS attack.

Media spamming: Figure 6 shows the state transition diagram for a media spamming attack. On receiving the first RTP packet (with packet field vector $\overrightarrow{x}$) from source x.src_ip(S) and destination x.dest_ip(D), the state machine makes a transition t, $\langle$`INIT`, `RTP_Packet`, $true$, A_t, `Packet Rcvd`$\rangle$ from the initial state (`INIT`) to the intermediate state (`Packet Rcvd`). During this transition, state variable vector $\overrightarrow{v}$ is initialized and assigned by update function A_t to the corresponding values of input vector $\overrightarrow{x}$, i.e. $A_t(v_i) \Rightarrow v_i := x_k$. At the (`Packet Rcvd`) state, each incoming RTP packet for the same destination `D` in the enterprise network is allowed to make a transition either to itself or to the (`Attack`) state depending upon the predicate outcome. During the transition to itself, function $A(v_i)$ updates state variable vector $\overrightarrow{v}$ with the recent values of the $(i+1)^{th}$ incoming packet (e.g. $v.time\_stamp_i := x.time\_stamp_{i+1}$). Media spamming attack is detected by observing the *sequence number* and *timestamp* of the incoming RTP packets. If the *timestamp* or the *sequence number* of the incoming packet has a sudden gap larger than Δt or Δn respectively, compared to the earlier received packet, then the fabricated message being injected into the media stream is detected. These rules are expressed as a part of predicates in (`Packet Rcvd`) state. The more formal definition of the rule for attack detection is given as follows: $((x.time\_stamp_{i+1} - v.time\_stamp_i > \Delta t)$ or $(x.sequence\_number_{i+1} - v.sequence\_number_i > \Delta n))$, where Δt and Δn are adjustable threshold variables.

7 Performance Evaluation

In this section, we evaluate the performance of vIDS in terms of additional delay induced to call setup time, CPU cost, and memory consumption. Since VoIP is a time-sensitive service, the effect of online placement of vIDS upon the QoS of voice stream is also studied. Finally, we assess the detection accuracy and sensitivity of vIDS.

7.1 VoIP Network Testbed

The topology of our VoIP network testbed is shown in Figure 7. The VoIP system consists of SIP proxy servers, IP softphones, routers and other data networking elements available in the network simulator OPNET [9]. Each enterprise network (A and B) is simulated by 10 generic Windows PCs (733 MHz Pentium III with 128 Mbytes RAM) acting as SIP UAs and one Sun Ultra 10 (333 MHz with 128 Mbytes RAM) machine acting as a SIP proxy server. vIDS is implemented on the Sun Ultra 10 machine and strategically located between the edge router and the hub of network B, allowing the visibility of all traffic. The VoIP system emulates the scenario of many UAs of network A making calls to UAs of network B. We assume that enterprise networks are based on $100BaseT$ Ethernet links and are connected to Internet clouds by $DS1$ link. The Internet delay between A and B is assumed to be 50 ms with 0.42% packet loss rate. The voice codec algorithm used is G.729 with the setting of ⟨Frame Size = 10 ms, Lookahead Size = 5 ms, DSP Processing Ratio = 1, Coding Rate = 8Kbps, Speech Activity Detection = Enabled⟩. The average SIP message size is assumed to be constant and is set to 500 bytes.

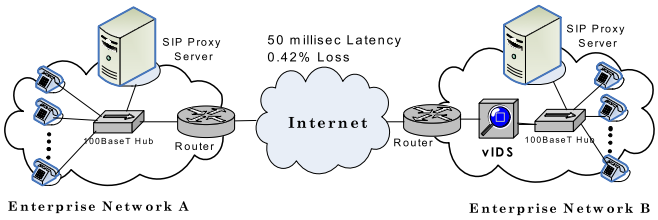


Figure 7. Simulated Network Topology

To emulate the realistic call behaviors, in our experiments, the UAs of network A generate call requests randomly and independently of each other. The call duration and calling interval between calls are also assumed to be randomly distributed. The experiment runs for 120 minutes. Figure 8 plots number of calls arrival and duration observed at enterprise network B's SIP proxy server.

7.2 Call Setup Delay

In the telephony world, performance requirements are generally expressed as cross-switching or message transfer times but not as call setup delays. Nevertheless, since IP telephony is used by Internet users, if they encounter long connection delays, the proposed security mechanism may not be adopted by the Internet service providers. Therefore, the extra delays induced to call setup times by vIDS is an important metric. In general, call setup delay is defined as

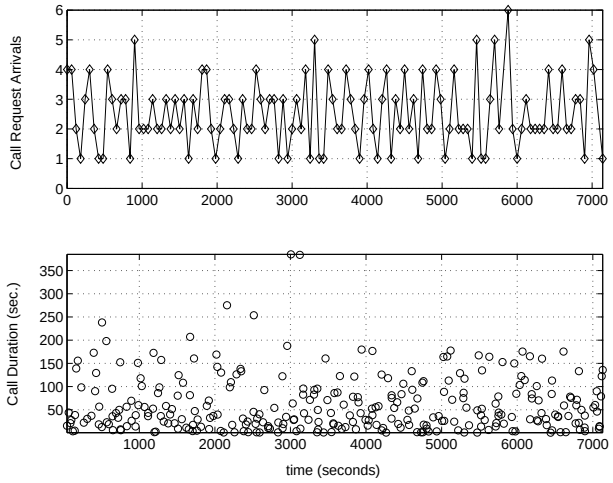


Figure 8. Call arrivals and duration

the interval between entering the last dialed digit and receiving ringback. In SIP-based VoIP systems, the call setup time can be defined as the time interval between a caller sending an `INVITE` message and receiving a `180 ringing` message back from the callee. Out of all 20 UAs available in network A, we have shown the call setup delays with and without vIDS for two representative UAs 3 and 4 in Figure 9. As it is evident from Figure 9, the average delay induced by vIDS to call setup is $\simeq$ 100 ms. Such an additional delay of 100 ms will be hardly noticeable by VoIP service subscribers (i.e. UAs).

7.3 Overhead Introduced by vIDS

During the call monitoring process, with the arrival of an `INVITE` request message, one instance of each protocol state machine is initiated starting from `(INIT)` state. As the call progresses, states make transitions to other states. At the end of the call, the associated instances of protocol machines are removed from the memory. The memory cost of maintaining the attack patterns is in the order of few KBytes. SIP messages are text based with varied length header fields. All mandatory fields, including source, destination, port numbers, and media information, consume about 450 bytes. Similarly, the RTP state information such as source, destination, ports, sequence number, timestamp, synchronization source (SSRC) identifier, and other relevant variable values, requires only 40 bytes of memory space. Although the memory requirement grows linearly with the number of calls, the very low memory cost per call allows us to easily monitor thousands of calls at the same time. In the absence of vIDS, the vIDS host (e.g. see Figure 7) simply forwards the received packets, whereas in the presence of vIDS, packets are logged at the granular-

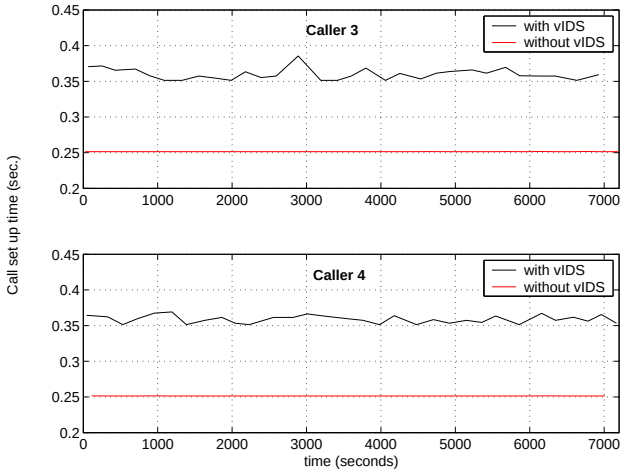


Figure 9. Call Setup Delay

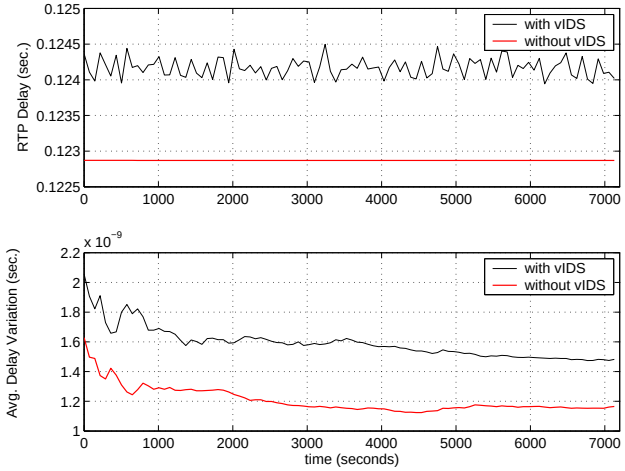


Figure 10. Impact on RTP streams

ity of a millisecond. The increase of CPU overhead due to running vIDS is $\simeq 3.6\%$.

7.4 Impact on QoS of RTP streams

IP telephony places stringent requirements on RTP streams to meet the specified QoS. The latency upper-bound is 150 ms for one way traffic, and jitter should be bounded as well. In our experiments, we study the effect of online placement of vIDS upon the QoS of voice streams. We evaluate the impact of vIDS with respect to two metrics : (1) end-to-end delay of RTP packets and (2) RTP jitter behaviors. Figure 10 shows the impact of vIDS on the QoS of RTP streams. On average, vIDS adds $\simeq$ 1.5 ms of additional delay to RTP based voice streams, while the delay variations are 0.3×10^{-9} seconds higher than those without the vIDS. Therefore, vIDS has a negligible effect upon RTP delay and jitter, which will not be perceived by VoIP service subscribers.

7.5 Detection Accuracy and Sensitivity

Note that vIDS is based on protocol state machines and the attack signatures of the known attacks. In our preliminary experiments with a few known attack scenarios, vIDS successfully detects these attacks without false alarms. For those attacks which have already been identified and recorded with attack patterns in the *attack signature* database, vIDS demonstrates 100% detection accuracy with zero false positive. However, the detection of unknown attacks (i.e., the attacks which do not have the corresponding signatures in the database) is largely dependent upon the development of protocol state machines. We postulate that the detailed and accurate representation of protocol state machines should be capable of detecting unknown attacks. The effectiveness of vIDS in detecting unknown attacks will be our future work.

The detection sensitivity of vIDS is defined as the earliest possible time to detect an intrusion since its commencement. The major strength of specification-based intrusion detection lies in its accurate and early detection capability, provided that the protocol state machine is detailed enough. The intrusion detection delay is mainly determined by the various *timer* in attack patterns, for example, timer T_1 in `INVITE` flooding detection and timer T in `BYE` DoS attack detection. T_1 depends upon the required detection granularity and the computational resources available at vIDS, whereas timer T depends upon the network conditions. After receiving a `BYE` message, setting timer T to one round trip time (RTT) should be long enough to receive all in-flight RTP packets, consequently, there would be less chance of false alarms. Seeking the optimized values of timers and their relationship with the probability of false alarms is our ongoing work.

8 Related Work

The work done by Sekar et al. [15] and Vigna et al. [17, 18, 19] are the closest to our work, in which state transition analysis tool (STAT) is used for intrusion detection. Sekar's specification based anomaly detection method [15] utilizes extended finite state automata to model network protocols. Vigna et al. proposed NetSTAT tool [18], an approach extending the STAT to network based intrusion detection, and WebSTAT tool [19] for detecting web-based attacks. WebSTAT operates on multiple event streams and correlates both network and operating system level events with the entries contained in the server log. Porras et al. [5] employed the similar technique to model computer penetrations as a

series of state changes from an initial secure state to a target compromised state. AODVSTAT [17] is also a STAT based tool for network-based real-time intrusion detection in the context of wireless networks, which are based on Ad hoc On-Demand Distance Vector (AODV) routing protocol.

Wu et al. [20] proposed SCIDIVE, a stateful cross-protocol intrusion detection architecture for VoIP. The architecture of SCIDIVE translates all incoming network packets into protocol dependent information. Packets are grouped according to sessions. The aggregated state from the multiple packets of a session are matched by the *Rule Matching Engine* against the *ruleset*. This approach has the same disadvantages as that of misuse intrusion detection system. Our proposed scheme is based on these previous approaches with significant enhancements via communicating extended finite state machines. Our approach is particularly suitable for VoIP applications because of its multi-protocol awareness.

9 Conclusions

In this paper, we formally described the *extended finite state machine* and utilized it for VoIP intrusion detection. We presented the potential security threats to the emerging SIP-based VoIP services, and detailed the stateful intrusion detection mechanism that is based on the communicating extended finite state machines. The proposed vIDS is particularly suitable for defending VoIP applications, because of its holistic consideration of multi-protocols and cross-protocol interactions. We have evaluated the performance of vIDS through our VoIP network testbed. Our experimental results show that the online placement of vIDS induces $\simeq$ 100 ms delay to call setup and an additional $\simeq$ 3.6% overhead to CPU cost. Due to the low memory cost per call, vIDS can easily monitor thousands of calls at the same time. Moreover, vIDS has negligible impact upon the perceived quality of voice streams. Finally, we demonstrated the high detection accuracy of vIDS and discussed its detection sensitivity.

References

[1] O. Arkin. Why E.T. Can't Phone Home? - Security Risk Factors with IP Telephony. Presentation, AusCERT Australia, 2004.

[2] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327, IETF Network Working Group, 1998.

[3] C. Hoare. Communicating Sequential Processes. In *Communications of the ACM, 21(8)*, pages 666–677, 1978.

[4] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1st edition, 1991.

[5] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[6] A. Johnston. *SIP Understanding the Session Initiation Protocol*. Artech House, 2nd edition, 2004.

[7] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE, Vol 84*, pages 1089–1123, August 1996.

[8] A. Niemi. Authentication of SIP calls. In *Tik-110.501 Seminar on Network Security*, 2000.

[9] OPNET. Optimum Network Performance, Modeler Tool Version 9.1. Network Simulation Tool, http://www.opnet.com/, 2003.

[10] A. Petrenko, S. Boroday, and R. Groz. Confirming Configurations in EFSM Testing. In *IEEE Transactions on Software Engineering (TSE)*, January 2004.

[11] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the* 13^{th} *System Administration Conference (LISA), USENIX Association*, pages 229–238, November 1999.

[12] 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, 2002.

[13] S. Salsano, L. Veltri, and D. Papalilo. SIP Security Issues: The SIP Authentication Procedure and its Processing Load. In *IEEE Networks*, pages 38–44, November 2002.

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

[15] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection : A new approach for detecting network intrusions. In *ACM Computer and Communication Security Conference (CCS)*, Washington DC, November 2002.

[16] Tipping Point. Intrusion Prevention : The Future of VoIP Security. White paper, http://www.tippingpoint.com/solutions_voip.html, 2005.

[17] G. Vigna, S. Gwalani, K. Srinivasan, E. Belding-Royer, and R. Kemmerer. An Intrusion Detection Tool for AODV-based Ad Hoc Wireless Networks. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 16–27, Tucson, AZ, December 2004.

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

[19] G. Vigna, W. Robertson, V. Kher, and R. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, December 2003.

[20] Y. Wu, S. Bagchi, S. Garg, N. Singh, and T. Tsai. SCIDIVE: A Stateful and Cross Protocol Intrusion Detection Architecture for Voice-over-IP Environments. In *IEEE Dependable Systems and Networks Conference (DSN 2004)*, June 2004.

[21] H. Yamaguchi, K. Okano, T. Higashino, and K. Taniguchi. Synthesis of Protocol Entities Specifications from Service Specifications in a Petri Net Model with Registers. In *15th International Conference on Distributed Computing Systems (ICDCS'95)*, May 1995.