SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks. 2009; **2**:457–475
Published online 2 December 2008 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/sec.77

Syntax, and semantics-based signature database for hybrid intrusion detection systems

Bazara I. A. Barry[1*,†] and H. Anthony Chan[1,2]

[1]*Department of Electrical Engineering, University of Cape Town, South Africa*
[2]*Huawei Technologies, Plano, TX, U.S.A.*

Summary

Signature-based intrusion detection systems (IDSs) have the advantages of producing a lower false alarm rate and using less system resources compared to anomaly based systems. However, they are susceptible to obfuscation used by attackers to introduce new variants of the attacks stored in the database. Some of the disadvantages of signature-based IDSs can be attributed to the fact that they are mostly purely syntactic and ignore the semantics of the monitored systems. In this paper, we present the design and implementation of a signature database that assists a Specification-based IDS in a converged environment. Our design is novel in terms of considering the semantics of the monitored protocols alongside their syntax. Our protocol semantics awareness is based on the state transition analysis technique which models intrusions at a high level using state transition diagrams. The signature database is hierarchically designed to insure a balance between ease of use and fast retrieval in real time. The database prototype is tested against some implemented attacks and shows promising efficiency. Copyright © 2008 John Wiley & Sons, Ltd.

KEY WORDS: intrusion detection; convergence; VoIP; signature-based detection; cross-protocol detection; hybrid detection

1. Introduction

Primarily, an intrusion detection system (IDS) is concerned with the detection of hostile actions. IDSs have been classified according to the detection approach to anomaly and signature-based systems [1]. Signature-based detection systems try to match computer activity to stored signatures of known exploits or attacks. In other words, signature-based detection systems use a priori knowledge on attacks to look for attack traces. Anomaly detection on the other hand is an approach to detect intrusions by first learning the characteristics of normal activity. Then systems are designed to detect anything that deviates from normal activity [2].

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 [3]. 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

*Correspondence to: Bazara I. A. Barry, Department of Electrical Engineering, University of Cape Town, South Africa.
†E-mail: barry@crg.ee.uct.ac.za

and also used 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 Sekar et al. [4] and Balepin et al. [5] 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. However, specification-based approach has a downside when it comes to detecting attacks such as network probing and denial of service (DoS) [4].

In a narrow sense, specification-based anomaly 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.

According to the resources they monitor, IDSs are divided into two main categories: host-based IDSs and network-based IDSs [6]. Host-based IDSs are installed locally on host machines. Host-based IDSs evaluate the activities and access to key servers upon which a host-based IDS agent has been placed [7]. On the other hand, network-based IDSs inspect the packets passing through the network [8].

Convergence in networks refers to the structures and processes that result from design and implementation of a common networking infrastructure that accommodates data, voice, and multimedia communications [9]. Network convergence is the first step toward application convergence which happens above the network layer. Convergence in applications refers to the building of applications that span over different protocols/specifications [10]. Putting many technologies on the same network is going to combine the potential attack targets and multiply them. Voice over IP (VoIP) is emerging as a standard that reflects convergence and replaces older Public Switched Telephone Network (PSTN) systems. Standard VoIP protocols such as SIP for signaling and Real-time Transmission Protocol (RTP) for data delivery do not provide adequate or standardized call party authentication or end-to-end call confidentiality and integrity [11]. 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. In addition, the use of low-level protocols such as IP, UDP, and TCP could propagate more threats to VoIP applications. In most network architectures and corresponding communications protocol stacks, network layer protocol data units are transmitted in the clear meaning that they are not cryptographically protected during their transmission. Consequently, it is relatively simple to do malicious things, such as inspecting the contents of the data units, forging the source or destination addresses, modifying the contents, or even replying old data units. IP as a network layer protocol is no exception. The lack of built-in security for IP packets makes it relatively easy to launch attacks such as malformed packets, flooding, denial of service (DoS), and buffer overflow, which can result in a full or partial service loss.

Considering the strengths and weaknesses each of the detection approaches has and bearing in mind the variety of threats VoIP converged systems are exposed to, we present the design and implementation of a signature database that is meant to accompany a specification-based IDS which uses the extended finite state machine (EFSM) model in attack detection. The signature database is designed for a host-based IDS. The design of the database enables the utilization of semantics-aware signatures alongside traditional syntax-aware ones. Our semantics-awareness is based on describing an attack at a high 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. We introduce additions to the traditional state transition analysis techniques that enable them to deal with attacks that were difficult to deal with previously such as DoS attacks. Furthermore, the introduced additions allow the database to model more complex penetrations. Our signatures have the capacity to foresee an impending compromise at a system's safe state and warn administrators beforehand. In addition, many variations of the same penetration can be stored efficiently in our signature database to scupper obfuscation attempts by attackers. Administrators in this scheme can flexibly adjust detection parameters in a way that reflects the security relevance of events and produces meaningful alarms. Our approach overtakes other approaches designed for VoIP converged environments in that it covers the threats of lower stack

DOI: 10.1002/sec

layers alongside covering application layer threats using the same principles. Our design is implemented and tested using a network simulator and shows promising detection accuracy and performance.

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, its threat model, and the threat model of lower TCP/IP protocols 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 and implementation 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. System Formal Model

In this section, we discuss the basics of state transition analysis techniques and EFSM model and how they are used in intrusion detection. The section draws attention to some of the limitations associated with EFSM model in specification-based detection which encourages the use of a more flexible and advanced state transition analysis as an accompanying signature-based detection mechanism.

2.1. State Transition Analysis

State transition analysis was developed by the Reliable Software Group at University of California, Santa Barbara to represent computer penetrations. State transition analysis forms a method for representing the sequence of actions that the attacker performs to achieve a security violation. In state transition analysis, a penetration is viewed as a sequence of actions performed by an attacker, that leads from some initial state on a system to a target compromised state, where a state is a snapshot of the system representing the values of all volatile, semi-permanent, and permanent memory locations on the system. 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 [12]. This level of abstraction allows for higher-level, semantics-aware representation of the attack scenario. Furthermore,

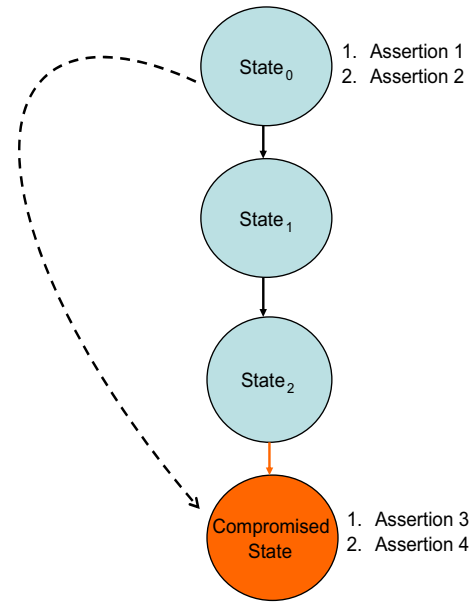

Fig. 1. A State Transition Diagram of an Attack.

state transition analysis has the advantage of the ability to foresee an impending compromise and preempt or limit the damage before it occurs. State transition analysis can anticipate an impending compromise at the state prior to the compromised one and forewarn administrators accordingly.

Figure 1 sheds more light on this important feature that is missed in other signature-based IDSs. The figure shows a STD 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. State diagrams are the common factor between state transition analysis and state machines.

2.2. Extended Finite State Machines

A communication protocol can be modeled as an 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 EFSMs.

DOI: 10.1002/sec

In the following definition of the EFSM, let X and Y be finite sets of inputs and outputs, R and V be finite disjoint sets of parameter and variable names. For $x \in X$, we note $R_x \subseteq R$ the set of input parameters and D_{Rx} the set of valuations of parameters in the set R_x. For $y \in Y$, we similarly define R_y and D_{Ry}. Finally, D_V is a set of context variable valuations v.

An EFSM M over X, Y, R, V, and the associated valuation domains is a pair (S, T) of a finite set of states S and a finite set of transitions T between states in S, such that each transition $t \in T$ is a tuple (s, x, P, op, y, up, s'), where

- $s, s' \in S$ are the initial and final states of the transition, respectively;
- $x \in X$ is the input of the transition;
- $y \in Y$ is the output of the transition;
- P, op, and up are functions, defined over input parameters and context variables V, namely,
  - P: $D_{Rx} \times D_V \rightarrow \{\text{True, False}\}$ is the predicate of the transition.
  - op: $D_{Rx} \times D_V \rightarrow \mathrm{D}_{Ry}$ is the output parameter function of the transition.
  - up: $\mathrm{D}_{Rx} \times D_V \rightarrow D_V$ is the context update function of the transition.

The EFSM operates as follows: The machine receives input along with input parameters (if any) and computes the predicates that are satisfied for the current configuration. The predicates identify enabled transitions. A single transition, among those enabled fires. Executing the chosen transition, the machine produces output along with output parameters, which, if they exist, are computed from the current context and input parameters by the use of the output parameter function. The machine updates the current context according to the context update function and moves from the initial to the final state of the transition. Transitions are atomic and cannot be interrupted. The machine usually starts from a designated configuration, called the initial configuration. A pair of an EFSM M and an initial configuration is called a strongly initialized EFSM [13].

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 EFSMs (CEFSMs) [14].

2.3. The Use of CEFSMs in Intrusion 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 suggest 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 predefined set of instructions (context update function).

Figure 2 shows an 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.

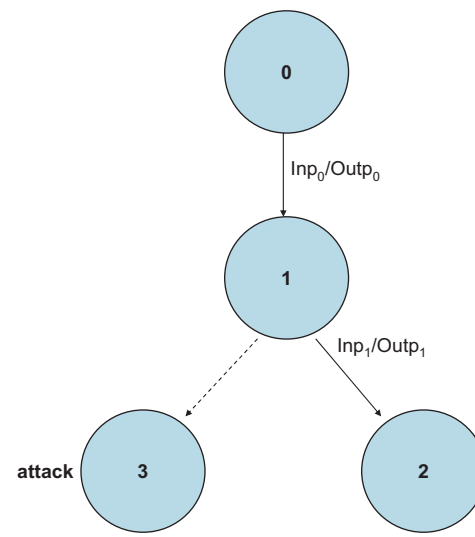

Fig. 2. A state transition diagram that shows normal and potential abnormal protocol behavior.

DOI: 10.1002/sec

2.4. Limitations of EFSM Model in Intrusion Detection

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 DoS and failed logins in anomaly mode [15].

Clearly, a signature-based module that is based on a flexible and more advanced state transition analysis is needed to assist specification-based module to detect such attacks. The suggestion of an accompanying signature detection mechanism based on state transition analysis is a natural product of the remarkable similarities between EFSM-based and state transition analysis-based approaches.

3. Related Protocols and Threat Model

In today's VoIP converged networks, two major application level signaling protocols dominate: SIP and H.323. Our discussion will focus on the issues related to SIP-based VoIP. This section will shed some light on the architecture, elements, and message flow of SIP, alongside the threats surrounding it and its accompanying data transmission protocol RTP. In addition, the discussion will continue to touch on threats of lower layer protocols.

Session Initiation Protocol (SIP) is a standard signaling protocol for VoIP, which was developed by the Internet Engineering Task Force (IETF) and is fully covered in RFC 3261. SIP was designed to address some important issues in setting up and tearing down sessions, such as user location, user availability, and session management. The simplicity and versatility of SIP make it the choice of instant messaging, video conferencing, and multiplayer game applications among others. SIP uses other protocols to perform various functions during a session such as Session Description Protocol (SDP) to describe the characteristics of end devices, Resource Reservation Protocol (RSVP) for voice quality, and RTP for real-time transmission. Elements in SIP can be classified into endpoints (also known as User Agents UAs) and intermediaries (also known as servers). The SIP message is made up of three parts: the start line (which tells whether the message is a request or a response), message headers, and body. SIP is designed to be independent of the underlying transport protocol, and hence SIP clients can use TCP or UDP to connect to SIP servers and other SIP endpoints.

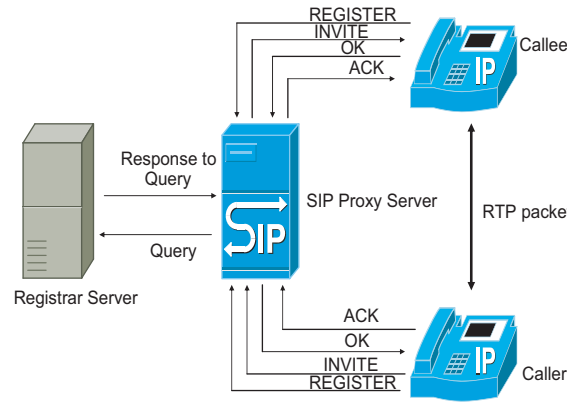

Fig. 3. Establishment of a Typical SIP Session.

3.1. SIP Session

Let's consider a situation where two SIP users want to establish a session. When turning on their devices, both users register their availability and their IP addresses with the SIP proxy server using REGISTER request. The proxy server then sends this information to the relevant Registrar server. The caller tells the proxy server 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. Figure 3 shows the establishment of an SIP session between two users in the same domain.

3.2. SIP Threat Model

The following are some of the threats SIP are susceptible to:

DOI: 10.1002/sec

- DoS: The consequence of a DoS attack is that the entity attacked becomes unavailable. This includes 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 [16].

3.3. RTP Threat Model

Attackers can inject artificial packets with higher sequence numbers that will cause the injected packets to be played in place of the real ones [11]. Flooding with RTP packets may cause phones dysfunctional and reboot operations [17].

3.4. Lower Layers Threat Model

Ever since Bellovin's paper on the security problems in the TCP/IP protocol suite [18], dozens of papers have been published on the same issue. The way TCP accepts new connections allows attackers to launch denial-of-service attacks to prevent anyone from using a particular host. Generally, attackers can take advantage of the stateful nature of TCP to cripple the protocol whilst in a certain state. The lack of built-in authentication with IP packets makes it easy for attackers to spoof packet addresses and get unauthorized privileges. Many could argue that lower layer attacks are probably not effective in today's environments for they have been around for a long time. However, due to software reuse and potential coding errors in future systems, there is always the possibility that these attacks, like any other, may become effective once again against some future systems [19]. Some recent reports on the performance of a newly released version of Microsoft Windows Vista proved the effectiveness of some of these attacks against the protocol stack of the operating system [20]. That was despite the fact that Microsoft removed a large body of tried and tested code and replaced it with freshly written one.

4. System Architecture

In this section, we start by discussing the various components in our hybrid IDS and how they interact with each other. This discussion will form a prelude to a detailed description for the components of the database. Furthermore, we show how the design of the database reflects the system formal model that was discussed in Section 2. Finally, we mention the advantages of our signature database.

4.1. Hybrid Intrusion Detection System Components

The proposed architecture of our host-based IDSs is shown in Figure 4. The *filter* is the first component to receive the incoming traffic. It classifies the traffic based on the active protocols at the receiving layer, and forwards packets to the right verifier. For instance, it classifies the packets coming from the network layer to the transport layer into TCP and UDP packets. It also classifies the packets coming to the application layer into signaling (SIP) and media (RTP) packets. 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 *packet 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 for potential suspicious patterns associated with the field. 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 EFSMs for the protocols involved to guard against any unacceptable behavior that violates proper protocol semantics. This way, unknown attacks can be detected by the *behavior observer*. Each protocol EFSM is provided with *state variables* to hold the values of header fields in incoming packets. A protocol EFSM is also provided with *getter* functions, so that other protocol EFSMs can get values of header fields and protocol state, which benefits the system in terms of detection accuracy. When reaching a certain state in

DOI: 10.1002/sec

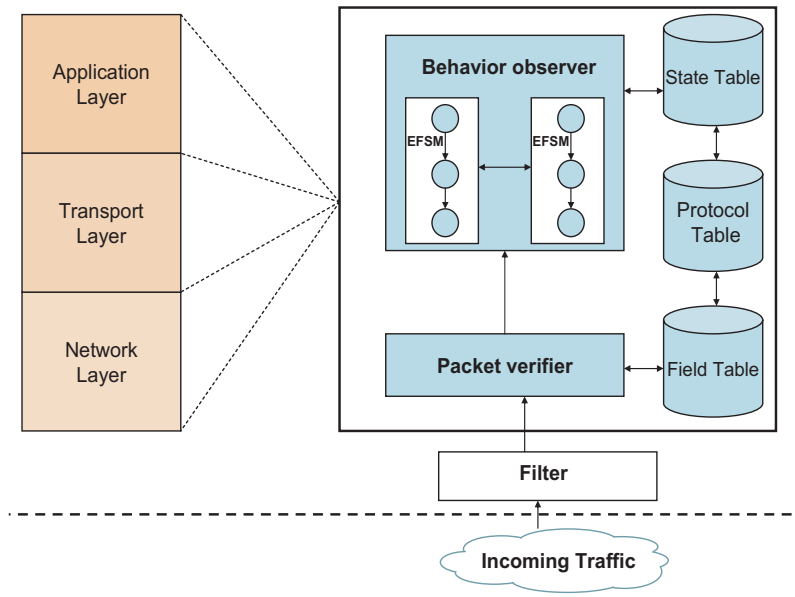

Fig. 4. Hybrid intrusion detection system architecture.

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.

4.2. Signature Database Components

Figure 5 shows the hierarchy of our signature database that contains two levels. The rest of this subsection details each of the three tables in the database.

(1) *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 IDS, 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. The following is a list of the major fields in the Protocol Table and their functionality:

- *Protocol ID*: A unique identifier that identifies the protocol supported by the system.
- *Protocol Name*: A name given to the protocol.
- *Layer*: The layer on which the protocol operates such as transport or application layer.
- *Description*: As its name suggests, this field describes the protocol and its role in the message exchange.

Table I shows an example of the content of two records from the protocol table. The table shows how our database defines SIP, RTP, and UDP 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.

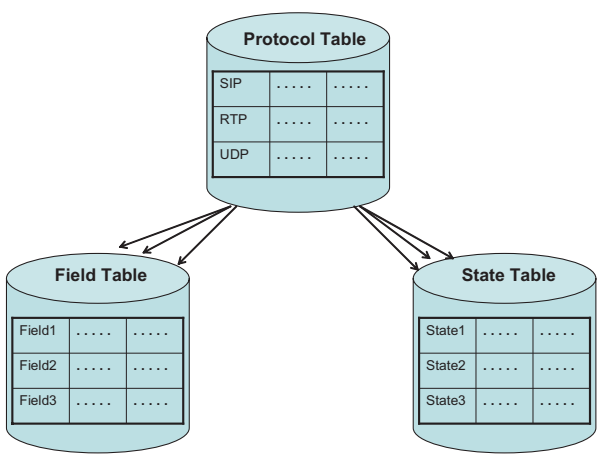

Fig. 5. A High-level hierarchy of the database.

(2) *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. Below, is a list of the main fields and their descriptions.

- *Protocol ID*: The same as in the Protocol Table, and the joiner of the two relations.
- *Field ID*: A unique identifier that identifies the field of the protocol header.

DOI: 10.1002/sec

Table I. Example of protocol table content.

Field name	Field content 1	Field content 2	Field content 3
Protocol ID	53	54	25
Protocol name	SIP	RTP	UDP
Layer	Application	Application	Transport
Description	A protocol used for session initiation	A protocol used for real-time transmission	A protocol used for unreliable transport

- *Field Name*: A name given to the field.
- *Description*: A description that shows the function of the field.
- *Type*: The data type of the field.
- *Pattern*: This field usually contains suspicious patterns the administrator is interested in detecting.
- *Stand-Alone Pattern*: A Boolean field to identify whether the above-described pattern forms an attack on its own, or as part of other fields. This feature enables the database to hold signatures, which span across multiple fields and multiple protocols.
- *Next Protocol ID*: If the Stand-Alone Pattern field contains False, this field points to the protocol ID of the next field in the multi-field signature.
- *Next Field ID*: If the Stand-Alone Pattern Boolean field contains False, this field points to the field ID of the next field in the multi-field signature.
- *Impact*: The effect of the attack on the system.

Table II 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 False in Stand-alone Pattern 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.

(3) *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 DoS 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. By providing this feature, the *State Table* reflects the design philosophy adopted by state transition analysis. 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

Table II. Example of field table content.

Field name	Field content 1	Field content 2
Protocol ID	53	53
Field ID	1	5
Field name	Start line	From
Description	Distinguishes SIP requests from responses	Contains 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

DOI: 10.1002/sec

Table III. Example of state table content.

Field name	Field content
Protocol ID	24
State ID	4
State name	SYN Received
Description	The system state after receiving SYN request
Threshold	200
Time unit	1 s
Timer	Null
Recommended action	SYN_Procedure()
Impact	Such action causes Denial of Service (DoS) at the server

of the protocol. Hence, it is the semantics-based component of the database. The following is a list of the main fields in the table.

- *Protocol ID*: The same as in the Protocol Table, and the joiner of the two tables.
- *State ID*: A unique identifier that identifies a state in the protocol EFSM.
- *State Name*: A name given to the state.
- *Description*: A description of the state and the system upon reaching it.
- *Threshold*: Identifies the upper limit for the number of requests that can be received at this state.
- *Time Unit*: Denotes the period of time during which the threshold is measured.
- *Timer*: Denotes a value for a timer that can be used at the state.
- *Recommended Action*: The procedure that should be executed by the system upon reaching the state to detect potential attacks.
- *Impact*: The effect of the attack on the system.

Table III shows an example of a signature from the state table. The signature shown in Table III indicates that the IDS should raise an alert whenever the number of SYN messages exceeds 200 within 1 s at the SYN Received state. The use of the Timer and Recommended action fields will be clarified in the section on specific implemented attacks.

4.3. How the Signature Database Reflects State Transition Analysis

Figure 6 delineates the relationship between our design represented by the *State Table*, and state transition analysis techniques. In the figure, a *State Table* lies on the right side, whereas attacks represented by state transition analysis lie on the left. Each record in the *State Table* represents a *safe state* in state transition analysis. As mentioned earlier, a *safe state* usually precedes a *compromised* one. A *compromised* state can be predicted and dealt with using the *stored procedure* that forms a part in every record in the *State Table*. Stored procedures use *Threshold* and *Timer* fields associated with records to provide more flexible detection capabilities. Furthermore, *Thresholds* and the associated *Time Units* in the *State Table* can be used to extend the capabilities of state transition analysis to detect DoS attacks and the likes. Therefore, the *State Table* adopts the design approach of state transition analysis bringing semantics awareness to attack modeling and improves the same approach to model attacks that were difficult to model previously.

4.4. Advantages of Signature Database Design

(1) The design of the database tables is simple and clean. This advantage is achieved by separating the anomalies in protocol traffic from specific attacks. The packet verifier and behavior observer eliminate anomalies according to protocol specifications. They also remove ambiguities in the incoming traffic which lets the field table and the state table focus on the modeling of specific attacks, rather than all anomalous behaviors. For example, if the packet verifier were not present in the design, we would have—somehow—to store signatures for all malformed SIP messages. Bearing in mind that there are different types of SIP messages, and for each message there are multiple headers with different combinations, we would end up storing a huge number of signatures. Since signatures are stored in the external memory which has a considerably longer access time than the random access memory, simplifying the design and limiting the number of signatures should be a goal sought by IDS designers.

(2) Our design maintains a reasonable balance between database normalization and performance. A normalized database has many one-to-one relationships and many tables to reflect these relationships. Such a design suggests small tables with a relatively few attributes for each. Although high normalization provides a clearer overview on the database for designers, it usually comes at the expense of retrieval time. For example, the primary type of information in Snort system [21] which is called the *event*, is represented in no

DOI: 10.1002/sec

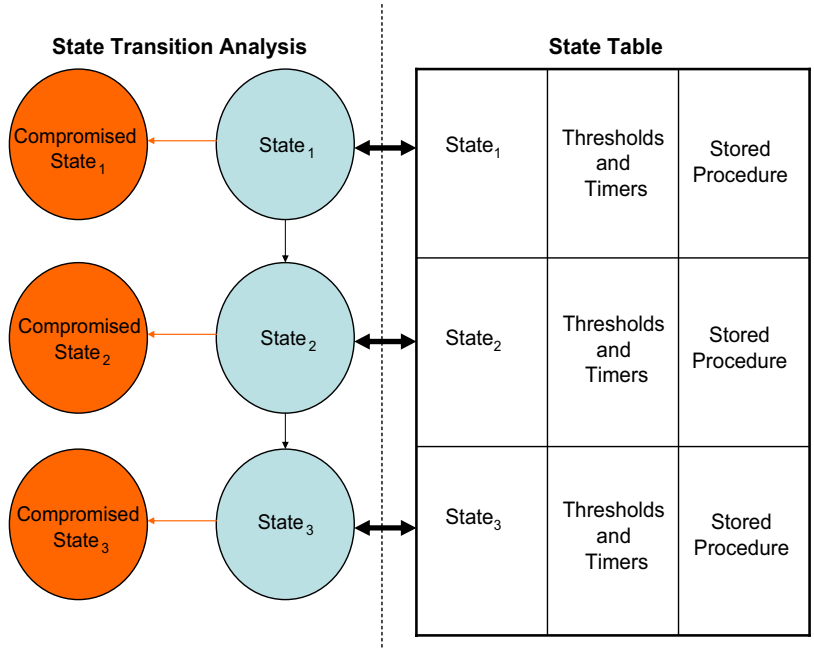

Fig. 6. The relationship between state transition analysis and state table.

less than six tables. Accessing the information of a single event may require joining six or more tables which could affect performance negatively [22]. The number of tables to be accessed grows directly with the number of protocols involved in a signature. On the other hand, we provide a less normalized database (two levels of hierarchy) with more attributes per table. A signature in our database is entirely stored in a single table (either field or state table). For reporting purposes, another table (the protocol table) is accessed, which puts the cost of accessing information in our database at no more than two tables. Furthermore, unlike other comparable signature databases, a cross-protocol signature can be stored in a single table (either field or state table), which benefits the system in terms of performance.

(3) Our system can thwart obfuscation attempts made by attackers to evade detection. Obfuscation is a technique used by attackers to introduce slight modifications in a certain penetration hoping to do the same damage without being detected by the signature database that stores a signature for the unmodified penetration. Thus, even if the attack scenario is represented in the signature database, a minor variation of the attack can go unnoticed. Our system tackles this problem by representing attacks in the state table using a higher-level representation. Upon reaching the safe state that precedes a compromised one in an attack, the system executes the procedure in the Recommended Action field. Such a procedure can deal with different types of intermediate states that represent variants of the attack. Furthermore, more than one procedure can be stored in the Recommended Action field to add more flexibility and deal with more variants of the attack.

5. Implemented Attacks and Detection Methodology

We implement six attacks to demonstrate the functionality of the signature database. The attacks are launched exploiting various vulnerabilities in protocols at different layers of the protocol stack. At the application layer we target SIP as a signaling protocol and RTP as media transport protocol. The implemented attacks at this layer are meant to be diverse and reflect different attack classes such as DoS, session hijacking, and session tearing down. These attacks can be found in classifications such as the one released by VoIPSA for threats that IP telephony is vulnerable to [23]. Such attacks are common in converged 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

DOI: 10.1002/sec

integrity, confidentiality, and origin authentication can only protect against outsiders and not against insiders who abuse their privileges.

For lower layers we implement some of the attacks that have been widely used to test IDSs. These attacks target protocols such as TCP, UDP, ICMP, and IP, and can be found in data sets such as 1999 DARPA Intrusion Detection Evaluation [24]. The rest of this section discusses the attacks and the detection methodology for each.

5.1. 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 DoS attack.

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 an 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 IV shows the signature. The pseudo-code of BYE_Procedure() which is the recommended action appears in Figure 7.

In this example we choose the Timer value to be 20 ms, which is the time each voice packet represents on average [9]. A point worth noting is that network conditions could scupper the aforementioned strategy.

Table IV. BYE attack signature.

Field name	Field content
Protocol ID	53
State ID	30
State name	BYE Received
Description	The system state after receiving BYE request
Threshold	Null
Time unit	Null
Timer	20 ms
Recommended action	BYE_Procedure()
Impact	Such action causes DoS at the endpoint

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. 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

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

Fig. 7. Pseudo-code of BYE attack detection.

```
Procedure Smurf_Procedure ( )
  while (Time_Unit > 0)
  {
    if (echo_request is received)
        number_of_requests = number_of_requests + 1
        if (number_of_requests >= Threshold)
            Raise_Alarm (Smurf_attack)
        else
            Time_Unit = Time_Unit - 1
    else
        Time_Unit = Time_Unit - 1
  }
```

Fig. 8. Pseudo-code of Smurf attack detection.

DOI: 10.1002/sec

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

Table V. Voice injection attack signature.

Field name	Field content
Protocol ID	54
State ID	7
State name	RTP Packet 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

5.4. UDP Storm

This attack causes two machines to attack each other. The idea is that there are a number of ports that will respond with another packet if a packet is sent. For example, Echo (port 7) will echo a packet back, while chargen (port 19) will generate a stream of characters. Consider a UDP packet with source port 7 and destination port 19. The packet generates some characters from the destination machine, headed for the echo port of the source machine. The source machine echoes these packets back, generating even more packets, and so on. Eventually, both machines are spending all their time sending packets back and forth which could result in a DoS.

To detect such an attack we create a signature in the field table for the source and destination port fields. Whenever an incoming UDP packet has the number 7 in its Source port and the number 19 in its Destination port, a UDP storm flag is raised by the system. Table VI shows the signature.

5.5. Land Attack

In this attack, a TCP SYN packet is constructed with the source and destination IP addresses the same and both set to the target machine. A direct impact of this attack is causing the target to reply to itself which could result in the machine being locked up or unresponsive. Attackers launching this attack take advantage of the lack of built-in authentication in IP which makes it easy to spoof packet addresses. Land attack can be considered a DoS attack.

To detect this attack a signature is created in the State table representing TCP state upon receiving a SYN packet. The recommended action field includes a cross-protocol procedure that checks the IP addresses of the packet, and raises an alarm if they are equal. *State variables* in the behavior observer can be used to store and compare IP addresses. Table VII shows the signature.

5.6. Smurf Attack

Smurf attack is a way of generating a lot of traffic targeting a victim host. The attack floods the target system via spoofed broadcast echo request (ping) messages. The attacker constructs echo requests (ping) with the target as the source IP and broadcast them to an intermediary network to maximize the number of machines responding. The machines at the intermediary network all respond to the echo request with packets destined for the target machine. The target machine

DOI: 10.1002/sec

Table VI. UDP storm signature.

Field name	Field content 1	Field content 2
Protocol ID	25	25
Field ID	1	2
Field name	Source port	Destination port
Description	The source port of the sender of the packet	The destination port of the receiver of the packet
Type	Number	Number
Pattern	7	19
Stand-alone	False	False
Next protocol ID	25	Null
Next field ID	2	Null
Impact	Such action causes Denial of Service (DoS) at the endpoint	Such action causes Denial of Service (DoS) at the endpoint

Table VII. LAND attack signature.

Field name	Field content
Protocol ID	26
State ID	5
State name	SYN packet received
Description	The system state after receiving SYN packet
Threshold	Null
Time unit	Null
Timer	Null
Recommended action	SYN_Rec_Procedure()
Impact	Such action causes Denial of Service (DoS) at the endpoint

Table VIII. Smurf attack signature.

Field name	Field content
Protocol ID	21
State ID	2
State name	Echo request received
Description	The system state after receiving an echo request
Threshold	100
Time unit	500 ms
Timer	Null
Recommended action	Smurf_Procedure()
Impact	Such action causes Denial of Service (DoS) at the endpoint

cannot process the large number of packets received and goes down under the load. Two main factors contribute to the success of Smurf attack, namely, the absence of cryptographic authentication at the network layer which allows attackers to spoof addresses, and poor network configuration by administrators which allows attackers to use networks as intermediaries.

A signature can be stored in the State Table representing the state of ICMP upon receiving an echo request. Two values are set to the Threshold and Time Unit, respectively. Whenever the number of arriving echo requests exceeds the Threshold within the Time unit, a Smurf attack flag is raised. Table VIII shows the signature. The pseudo-code of the recommended action is shown in Figure 8.

6. Implementation and Simulation

We use OMNeT++ [25] 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. OMNeT++ support for TCP/IP protocols such as IP, ICMP, UDP, and TCP started with the Internet Protocol Suite (IPSuite) and has culminated in the more recent INET Framework. Several research groups at the University of Karlsruhe developed MMSim [26] which is a model to simulate multimedia protocols using OMNeT++. The MMSim model provides support for SIP, RTP, and Real-Time Streaming Protocol (RTSP).

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. Therefore, the design of a topology and the implementation of the modules that exist in the topology are separated. In addition, OMNeT++ provides a high degree of parameterization through the use of NED and initialization files and a solid support for FSMs in the form of ready-to-use classes and functions.

6.1. Network Topology and Configuration

Figure 9 shows the simulated network topology. Our network comprises two domains each with a Proxy

DOI: 10.1002/sec

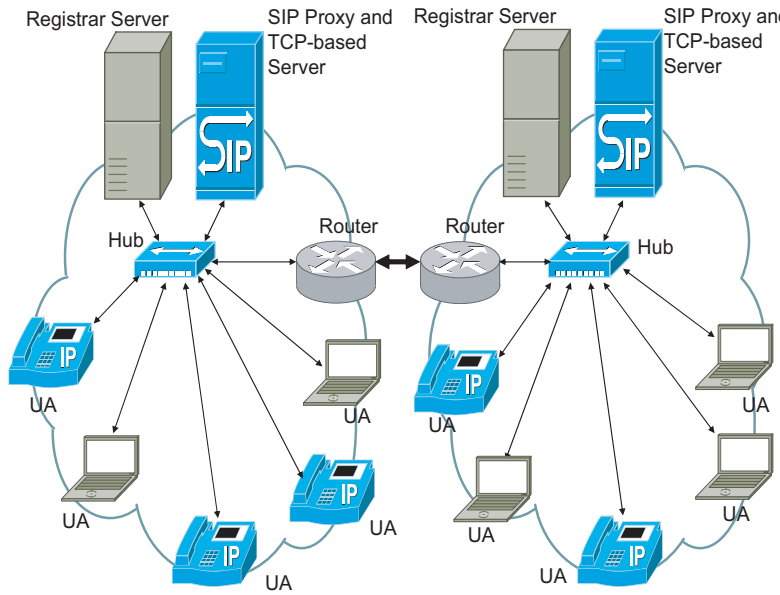

Fig. 9. Simulated network topology.

and Registrar Server. Each domain also contains a set of User Agents (endpoints) which are connected to the servers by a 10Base-T Ethernet. Proxy servers and endpoints in each domain also run TCP-based server and client applications. TCP-based server and client applications simulate web users issuing HTTP requests. This setting of client and server applications is meant to emulate the operation and functionality of converged services. A converged service is an application that spans communications over multiple protocols to provide higher level function, and better user experience.

For the VoIP application, 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 10, and has the encoding L16. The clock rate used is 44 100 Hz and the number of channels is 2. Endpoints in a domain make calls to other endpoints in the other domain randomly and without predefined durations.

On the other hand, the TCP-based server and client applications use TCP Reno algorithm. We use a maximum segment size (MSS) of 1024 bytes per segment, and an advertised window of 14 336 bytes. Similar to voice-based applications, TCP-based applications establish sessions with servers randomly, and sessions last for random 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 per cent. The experiment runs for five different runs, and each run lasts for 120 min. The results which will be shown shortly in the next section are averaged across the different runs.

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 background 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

DOI: 10.1002/sec

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 attributes can be captured by SIP through a specialized function that can be called from SIP module. In addition, messages in OMNeT++ have a field called *control info* that carries auxiliary information to facilitate communication between protocol layers which makes cross-layer detection possible.

On the other hand, C++ streams which are associated with files are used to emulate our signature database. Functions that perform the recommended actions are given names that reflect the associated protocol and state. This way, functions are linked to records in the state table. When an administrator defines a function to perform a certain recommended action, he/she should store the function's name in a system text file. When the detection process reaches a certain state of a protocol during a session, the IDS searches the system text file for the function's name using a combination of the protocol and state ID. If the function's name is found, it means that there is a function defined for that state of the protocol. Therefore, the IDS calls the function accordingly to perform the recommended action. More than one function can be defined for a single state of a protocol by adding different suffixes to the end of the function's name.

7. Results and Analysis

7.1. Detection Accuracy

Table IX summarizes the detection accuracy results. The table shows how our signature database has detected *all* attacks launched during the experiment.

Whenever the detection of an attack relies upon a certain order of packet arrivals within a predefined amount of time, there is a possibility of a false alarm. This phenomenon is due to network conditions, and non-guaranteed delivery within a specific time window.

Table IX. Detection accuracy results.

Attack name	Instances	Attacks detected	Detecting module
BYE	6	6	State table
Re-INVITE	5	5	State table
Voice injection	4	4	State table
UDP storm	2	2	Field table
Land	7	7	State table
Smurf	4	4	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 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.

A major source of false alarms in intrusion detection is threshold detection. The goal of threshold detection is to record each occurrence of a specific event and detect when the number of occurrences of that event surpasses a reasonable amount that one might expect to occur within a specified time period. An unnaturally high number of occurrences within a short period of time may indicate an attack. Once the threshold number of occurrences is surpassed, the threshold detector can notify the administrator. The relatively high rate of false alarms stems from the difficulty to identify the threshold number and the time frame for the specific event. These two parameters are highly dependent upon the security relevance of the event and the historical number of occurrences. Therefore, the choice of these values is often left to administrators [12], which is the policy adopted by our system to reduce false alarms originating from threshold detection. The assumption here is that in a well-designed system, any alarm contains information. For example, one may see a few packets that look like a probe for vulnerable systems. The administrator may want to know about this, even though it is not yet a problem and even though in reality it may not be a prelude to an attack at all. In this scheme the system reports only alarms for events that are meaningful to administrators, and hence reduces the amount of false alarms significantly. The origins of this school of thought are discussed in detail in Reference [27].

7.2. Performance Evaluation

It is vital that any security measure to be implemented in a VoIP converged network does not impede the performance of the network. Quality of service (QoS) is very important to the operation of VoIP networks. The implementation of various security measures in

DOI: 10.1002/sec

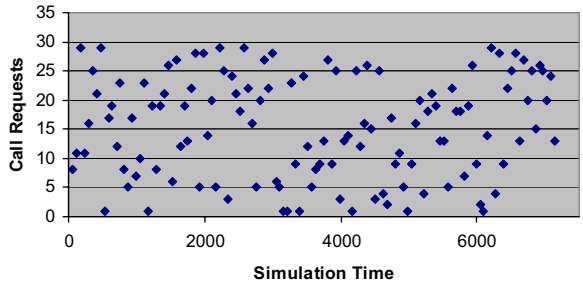

Fig. 10. Call requests at a proxy server.

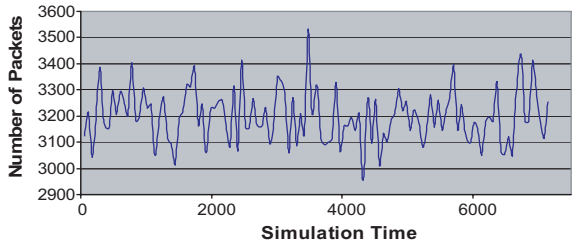

Fig. 11. Amount of traffic at a TCP-based Ssrver.

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 signature database, and show its effect on the hosts. We measure the performance of IDS on hosts with and without the signature database, and compare it to the performance of hosts with no IDS installed. Our discussion will focus on end-to-end delay, call setup delay, processing delay, and packet loss at endpoints and servers. Before we start discussing these issues in detail, we show in Figure 10 the number of call requests as captured at the Proxy Server of one of the domains.

In addition, we show in Figure 11 the amount of traffic received by one of the TCP-based servers.

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 ms one-way delay is acceptable for high voice quality [28]. Every element along the voice path adds to this delay. This includes switches, routers, and public Internet connections. Figure 12 shows the end-to-end delay experienced by an endpoint in the network with and without the operation of the signature database, and without the IDS installed. Our hybrid IDS adds about 2.7 ms on average to the voice transmission delay. The operation of the signature database adds about 0.06 ms on average to this delay. As shown in the figure, the overall delay remains considerably less than the upper

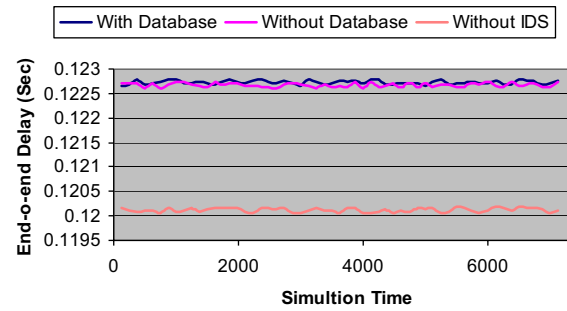

Fig. 12. End-to-end delay.

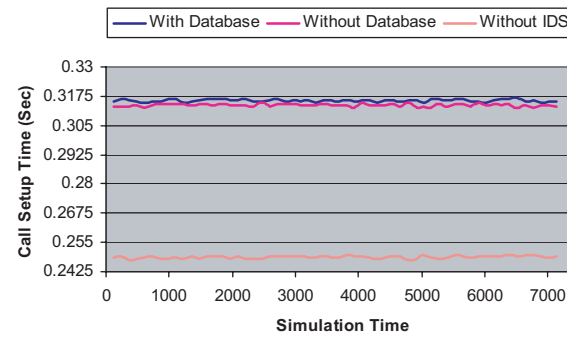

Fig. 13. Call setup delay.

bound of 150 ms. The delay variation (jitter) remains around 2 ms with a slight addition of 1.6×10^{-5} s by our IDS. Therefore, our hybrid 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 to that of PSTNs. Users may be annoyed with a setup process that requires more than a few seconds. Figure 13 shows the call setup delay introduced by our IDS at a certain endpoint during the simulation. The hybrid IDS adds about 67 ms to the call setup process, with about 2 ms contributed by the operation of the signature database. Such an increase inthe call setup time is tolerable by VoIP users. Furthermore, the overall call setup delay remains within the limit of 1–2 s mentioned in Reference [29].

End-to-end and call setup delays can be affected by another important factor which is processing delay. Processing delay is the time required by an endpoint or a server to process a message. We show in Figure 14 the processing delay at a host with and without our IDS. Our IDS adds about 4.4 per cent increase to the processing delay on average, and the processing delay per message remains around 2429 μm. The operation

DOI: 10.1002/sec

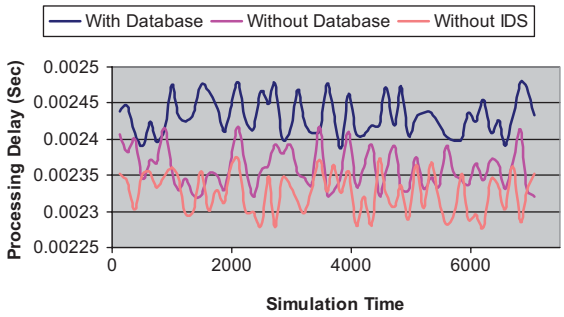

Fig. 14. Processing delay.

of the signature database adds about 73.2 μm on average to this delay. An important observation that can be taken from Figure 14 is the processing spikes indicating extra processing time needed for certain packets.

We observe that high packet loads have no remarkable impact on end-to-end, call setup, and processing delay. The averaged results under high loads remain close to the abovementioned figures. However, the impact is noticeable when it comes to packet loss. High sending rates can lead to packet drops especially in UDP-based transmissions. Unlike TCP, UDP lacks built-in transmission control mechanism that makes senders adapt themselves to the buffer capacity of receivers. The absence of such a mechanism in UDP could lead to a situation where the receiver is unable to keep up with the high sending rate of the sender, which results in some packet drops from the receiver's buffer. Another contributing factor to packet loss is processing spikes. Processing spikes mean that the CPU is spending too much time on some packets which has the consequence of missing subsequent ones. Ideally, there should be no packet loss for VoIP. Losses of 3 and 4% could place the quality in VoIP networks encoded by certain codecs at a level below the QoS level of PSTNs.

Figure 15 shows the packet loss rate at servers and endpoints buffers with and without our IDS for various amounts of traffic. The figure does not show a separate curve for the operation of the signature database because its effect could not be distinguished from the effect of the hybrid IDS during the experiment. The packet loss rate with our hybrid IDS is only 0.02% higher than the rate without it on average. The overall packet loss remains at 0.05% on average, which is considerably less than the 1% level specified by many codecs as the upper limit.

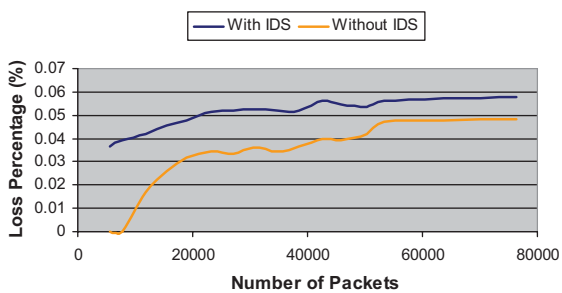

Fig. 15. Packet loss rate.

The good performance figures shown by our signature database can be attributed to the fact that 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.

8. Related Work

Hybrid IDSs can trace their origins back to systems such as IDES [30], NADIR [31], and W&S [32]. 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. Sekar et al. [4] proposed a hybrid IDS that combined specification-based and anomaly based detection. Their reliance on protocol specifications helped to simplify the process of feature selection which plays a major role in anomaly detection approaches. However, their reliance on anomaly based detection hindered the system ability to detect attacks that were not based on repetition such as Land attack.

STAT [12] and NetSTAT [33] 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, Orset, *et al.* [34] 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 [21]. The Snort database design defines

DOI: 10.1002/sec

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 [35] 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 STDs 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 [11] 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 [17]. 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 [36] 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

In this paper, we have proposed a highly demanded syntax and semantics-based signature database for hybrid IDSs in VoIP converged environments. Our database supports traditional byte-level signatures at the field level in the protocol header. In addition, it introduces the state-based model, which enables administrators to create semantics-aware signatures. Our database improves state transition analysis techniques, which are used to represent attacks, to depict the semantics of the protocols in signatures. Our improvement comes in the form of flexible detection parameter setting to help detect attacks that have been challenging to detect such as DoS and complex cross-protocol attacks. We have presented the efficiency of the signatures by implementing and launching six different attacks against the detection system in a simulator environment. Our database has showed promising efficiency detecting all the attacks, and low runtime impact on the operation of hosts.

References

1. Rhodes B, Mahaffey J, Cannady J. Multiple self-organizing maps for intrusion detection, *Proceedings of the 23rd National Information Systems Security Conference*, Baltimore, MD, 2000.
2. Kemmerer RA, Vigna G. Intrusion detection: a brief history and overview. *IEEE Computer–Special Issue on Security and Privacy* 2002; **35**(4): 27–30.
3. Wagner D, Dean R. Intrusion detection via static analysis, *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.
4. Sekar R, Gupta A, Frullo J, Shanbhag T, Tiwari A, Yang H, Zhou S. Specification-based anomaly detection: a new approach for detecting network intrusions, In *ACM Computer and Communication Security Conference (CCS)*, Washington DC, November 2002.
5. Balepin I, Maltsev S, Rowe J, Levitt K. 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.
6. Depren O, Topallar M, Anarim E, Ciliz MK. An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications* 2005; **4**(29): 713–722.
7. Lichodzijewski P, Zincir-Heywood A, Heywood M. Host-based intrusion detection using self-organizing maps, In *Proceedings of the IEEE International Joint Conference on Neural Networks*, Hawaii, May 2002.
8. Lichodzijewski P. Network-based anomaly detection using self-organizing maps, *Technical Report*, Nova Scotia, Dalhousie University, Halifax, 2002.
9. Porter T. Practical VoIP Security. Syngress: Rockland, MA, 2006; Chapter 1.
10. Khan N. The SIP servlet programming model, Technology White Paper (Oct. 2007). Available http://dev2dev.bea.com
11. Wu Y, Bagchi S, Garg S, Singh N, Tsai T. 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.
12. Ilgun K, Kemmerer RA, Porras PA. State transition analysis: a rule-based intrusion detection approach. *IEEE Transactions on Software Engineering* 1995; **21**(3): 181–199.
13. Petrenko A, Boroday S, Groz R. Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering* 2004; **30**(1): 29–42.
14. Gerald Holzmann J. Design and Validation of Computer Protocols, Prentice Hall: New Jersey, 1991; Chapter 8.
15. Sundaram A. An introduction to intrusion detection. *ACM Crossroads Magazine, Special Issue on Computer Security* 1996; **2**(4): 3–7.

DOI: 10.1002/sec

16. Poikselka M, Mayer G, Khartabil H, Niemi A. The IMS: IP Multimedia Concepts and Services in the Mobile Domain, Wiley: Sussex, 2004; 278.
17. Sengar H, Wijesekera D, Wang H, Jajodia S. VoIP intrusion detection through interacting protocol state machines, In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, Philadelphia, USA, 2006.
18. Bellovin SM. Security problems in the TCP/IP protocol suite. *ACM SIGCOMM Computer Communication Review* 1989; **19**(2): 32–348.
19. Marchette DJ. Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint, Springer-Verlag: York, PA, 2001; Chapter 4.
20. Newsham T, Hoagland J. Windows vista network attack surface analysis: a broad overview, Technical Report, C SYMANTEC Advanced Threat Research, Published July 2006. Available: http://www.symantec.com
21. Snort–The de facto Standard for Intrusion Detection/ Prevention. Available: http://www.snort.org
22. Wasniowski RA. Network intrusion detection system with data mart, *Proceedings of The 2006 International Conference on Security and Management*, Las Vegas, USA, 2006.
23. VOIPSA. VoIP Security and Privacy Threat Taxonomy, October 2005. Available http://www.voipsa.org
24. Lippmann R, Haines JW, Fried DJ, Korba J, Das K. The 1999 DARPA off-line intrusion detection evaluation, *In Proceedings of Recent Advances in Intrusion Detection (RAID)*, Toulouse, France, October 2000.
25. OMNeT++ Simulator. Available: http://www.omnetpp.org
26. MMSim–Simulation of Multimedia Protocols using OMNeT++. Available: http://www.ibr.cs.tu-bs.de/projects/ mmsim
27. Proctor PE. The Practical Intrusion Detection Handbook, Prentice-Hall: Englewood Cliffs, New Jersey, 2001; 108–111.
28. International Telecommunication Union–Telecommunication Standardization Section Recommendation G.114 (2003, May). *One-way Transmission Time*. Available: http://www.itu.int
29. Internet Engineering Task Force–Internet Draft (1999. June). *VoIP Signaling Performance Requirements and Expectations*. Available: http://tools.ietf.org
30. Lunt TF, Tamaru A, Gilham F, Jagannathan R, Jalali C, Neumann PG, Javitz HS, Valdes A, Garvey TD. A real-time intrusion detection expert system (IDES), OT Final Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1992.
31. Jackson KA, DuBios DH, Stalling CA. An expert system application for network intrusion detection, *Proceedings of the 14th National Computer Security Conference*, Baltimore, MD, pp. 215–225, October 1991.
32. Vaccaro HS, Liepins GE. Detection of anomalous computer session activity, *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, pp. 280–289, May 1989.
33. Vigna G, Kemmerer R. NetSTAT: a network-based intrusion detection approach, In *Proceedings of the 14th Annual Computer Security Application Conference (ACSAC)*, Scottsdale, Arizona, 1998.
34. Orset JM, Alcalde B, Cavalli A. An EFSM-based intrusion detection system for Ad Hoc networks, *Proceedings of The Third International Symposium, Automated Technology for Verification and Analysis (ATVA)*, Taipei, Taiwan, October 2005.
35. Sommer R, Paxon V. Enhancing byte-level network intrusion detection signatures with context, *Proceedings of Tenth ACM Conference on Computer and Communication Security*, Washington DC, 2003.
36. Sengar H, Wijesekera D, Wang H, Jajodia S. Fast Detection of Denial-of-Service Attacks on IP Telephony, *Proceedings of IEEE Fourteenth International Workshop on Quality of Service*, New Haven, CT, 2006.

DOI: 10.1002/sec