ABSTRACT

The unique features of Voice over Internet Protocol (VoIP) systems introduce various security challenges which entail novel detection approaches. Signature-based detection approaches have been lacking the necessary flexibility to model attacks semantically and to thwart obfuscation. In this paper, we demonstrate the design and implementation of a signature database that complements a specification-based detection module in an Intrusion Detection System (IDS) that targets VoIP environments. Our design models attacks at the session higher level and provides an effective solution against obfuscation attempts. The hit rate and performance evaluation figures shown by the database attest the quality of the design.

1 INTRODUCTION

Intrusion detection systems can be classified based on the detection principle into signature-based, anomaly-based, and specification-based systems. Signature-based systems use stored behavior patterns to identify and detect attacks. This approach can detect known attacks accurately, but is ineffective against previously unseen attacks, as no signatures are available for such attacks. Anomaly-based systems create a normal behavior model for a system using previously seen behaviors in the absence of attacks. Then the model is used to classify any behaviors that violate it as potential attacks. This approach can detect unknown attacks; however it produces false alarms for legitimate but previously unseen behaviors. Specification-based systems are based on manually developed specifications that capture legitimate (rather than previously seen) system behaviors. Therefore, they are capable of detecting new attacks avoiding the high rate of false alarms caused by legitimate-but-unseen-behavior in the anomaly detection approach. However, specification-based approaches have a drawback when it comes to certain attacks such as denial of service and network probing [13].

Protocols in VoIP can be classified into signaling, media transport, and support protocols. Signaling protocols are responsible for call setup, tear down, and modification. Media transport protocols are involved in end-to-end transport of voice and multimedia data. Support protocols are used to enable services and features required for proper network operation. A major source of vulnerabilities in these protocols is that they transmit packet headers and payloads in clear text by default due to the absence of built-in authentication and encryption [1]. 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 this paper, we present the design and implementation of a signature database that assists a specification-based module in a hybrid IDS designed for VoIP systems. The design of the database enables the utilization of semantics-aware signatures alongside traditional syntax-aware ones. 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. 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 the related work. Section 3 discusses State Transition Analysis Techniques which form the base of our design. SIP suite and its threat model are discussed in Section 4. Section 5 sheds some light on the system design. The implemented attacks used to test our system are detailed in Section 6. Section 7 discusses some implementation issues and presents the detection and performance evaluation results. The paper is concluded in Section 8.

2 RELATED WORK

Hybrid IDSs can trace their origins back to systems such as IDES [10], NADIR [11], and W&S [12]. 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 [13] proposed a hybrid intrusion detection system 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 attacks are detectable in our hybrid architecture.

STAT [2] and NetSTAT [14] adopted state transition analysis for host and network-based detection respectively. Being only signature-based limited the ability of these systems to detect new attacks. On the other hand, J. M. Orset, B. Alcalde, and A. Cavalli [15] 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 [16]. The Snort database design defines the lowest level of detail as an event, which is the combination of a collection of packet header and data, and an active Snort rule, called a signature. However, Snort’s approach falls short of providing a basis for effective semantics-aware signatures at the session level. Sommer and Paxon [17] proposed adding connection-level context to signatures to reduce false positives in signature-based detection. However, their aim was to complement the most common form of signature matching, which is low-level string matching, with context. Our signature database combines both types of signatures, byte-level and semantics-aware. Our semantics-awareness is based on describing attacks using state transition diagrams which allow us to represent attacks at the session level rather than lower 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.

3 STATE TRANSITION ANALYSIS

State transition analysis provides a method of representing the sequence of actions that the attacker performs to achieve a security violation. A major advantage of using this approach is its ability to foresee an incoming penetration based on the current system state. This advantage allows IDSs to limit the damage before it occurs. 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 [2]. This level of abstraction allows for higher-level and semantics-aware representation of the attack scenario.

Figure 1 shows a state transition diagram of a certain attack.

![State Transition Diagram of an Attack](image)

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.

Despite the remarkable convenience STDs provide, 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, cannot be modeled easily using STDs. This limitation is the reason why STD-based systems cannot directly detect attacks such as Denial of Service and failed logins [3]. Clearly, more flexibility has to be added to STD-based approaches in order to detect such attacks.

4 SIP-BASED VOIP AND THREAT MODEL

Session Initiation Protocol (SIP) is a standard signaling
protocol for VoIP. It was developed by the Internet Engineering Task Force (IETF) in RFC 2543 which was updated by RFC 3261. SIP was designed to address some important issues in setting up and tearing down sessions, such as user location, user availability, and session management. The simplicity and versatility of SIP make it the choice of instant messaging, video conferencing, and multiplayer game applications among others. SIP uses other protocols to perform various functions during a session such as Session Description Protocol (SDP) to describe the characteristics of end devices, Resource Reservation Protocol (RSVP) for voice quality, and Real-time Transport Protocol (RTP) for real-time transmission.

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

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

The session is initiated using the INVITE method. INVITE requests follow a three-way handshake model, which means that the UA, after receiving a final response to an INVITE request, must send an ACK request. After establishing a session, the users can send and receive data using RTP. If the UA wants to cancel an invitation to a session after it has sent the INVITE request, it sends a CANCEL request. INVITE requests can also be sent within dialogs to renegotiate the session description. A session is terminated with a BYE request.

SIP is susceptible to many attacks such as Denial of Service (which includes scenarios like targeting a certain UA or server and flooding them with requests), tearing down sessions prematurely by sending fake BYE or CANCEL requests, and session hijacking by sending fake Re-INVITE requests [4]. With regard to RTP, Attackers can inject artificial packets with higher sequence numbers, which causes the injected packets to be played in place of the real ones. Flooding with RTP packets deteriorates the perceived Quality of Service (QoS) and may also cause phones dysfunctional and reboot operations [5].

5 SYSTEM ARCHITECTURE

In this section we start by mentioning the components of the hybrid IDS and their interaction with each other before detailing the components of the signature database.

5.1 System Components

The proposed architecture of the hybrid host-based intrusion detection system is shown in figure 2.

The filter receives the incoming traffic and classifies it into signaling (SIP) and media (RTP) packets. The packet verifier receives packets from the filter and examines them in terms of size and structure. Too big or malformed packets are rejected by the packet verifier in order not to deplete the processing power of the endpoint. Individual header fields of the packet are examined to check if they comply with the protocol specifications, and whether mandatory fields are present. Then 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 an Extended Finite State Machine (EFSM) for each of the protocols involved in a session to detect any deviation from proper protocol behavior according to specifications. EFSMs in the behavior observer exchange information which allows for cross-protocol detection. When reaching a certain state in the EFSM, the system retrieves all the records of that state from the state table to perform further checks on semantics violations. Clearly, detecting and reporting attacks take place in real-time.

5.2 Signature Database Components

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 system, and each field defines a high-level attribute of the protocol. This table is meant for organizational purposes and to add some normalization to the design of the signature database. Table I shows an example of the content of two records from the protocol table. The table shows how our database defines SIP and RTP at a high level. The Protocol ID field gives each protocol a unique identifier, and is used to join various tables as will be shown shortly.

### Table I
**EXAMPLE OF PROTOCOL TABLE CONTENTS**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Content 1</th>
<th>Field Content 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol ID</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>Protocol Name</td>
<td>SIP</td>
<td>RTP</td>
</tr>
<tr>
<td>Layer</td>
<td>Application Layer</td>
<td>Application Layer</td>
</tr>
<tr>
<td>Description</td>
<td>A protocol used for session initiation</td>
<td>A protocol used for real-time transmission</td>
</tr>
<tr>
<td>Field Content 1</td>
<td>Field Content 2</td>
<td></td>
</tr>
</tbody>
</table>

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.
- **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 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 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 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.

### Table II
**EXAMPLE OF PROTOCOL TABLE CONTENTS**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Content 1</th>
<th>Field Content 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol ID</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>Field ID</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Field Name</td>
<td>Start Line</td>
<td>From</td>
</tr>
<tr>
<td>Description</td>
<td>To distinguish requests from responses</td>
<td>The sender of the message</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Pattern</td>
<td>INVITE</td>
<td>sip:<a href="mailto:alice@domain.com">alice@domain.com</a></td>
</tr>
<tr>
<td>Stand-alone</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Next Protocol ID</td>
<td>53</td>
<td>Null</td>
</tr>
<tr>
<td>Next Field ID</td>
<td>5</td>
<td>Null</td>
</tr>
<tr>
<td>Impact</td>
<td>INVITE requests from Alice should not be received for administrative reasons</td>
<td>INVITE requests from Alice should not be received for administrative reasons</td>
</tr>
</tbody>
</table>

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 Denial of Service saturation attacks. Furthermore, a record could contain a stored procedure to be executed upon arriving at the certain state. Such a procedure is meant to predict an impending compromise at the current system’s safe state and to limit the damage before it occurs. This strategy stems from the fact that for multi-step attacks, there are benign steps that precede the attack sequence. The administrator can provide the state table with the necessary procedures to be taken at the safe state that precedes the attack. 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 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.

An example of the contents of this table will be shortly shown in the section on the implemented attacks.

5.3 **Signature Database Design Advantages**

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.

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. Our design provides 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 enhances the performance.

3. Our signature database can thwart obfuscation attempts made by attackers to evade detection by representing attacks in the state table using a higher-level and audit record independent representation. Upon reaching the safe state that precedes a compromised one in an attack, the system executes a special procedure that can deal with different types of intermediate states that represent variants of the attack. Furthermore, more than one procedure can be stored in a single record in the state table to add more flexibility and deal with more variants of the attack.

6 **IMPLEMENTED ATTACKS**

We implement four attacks to demonstrate the functionality of the signature database. The attacks are launched exploiting various vulnerabilities in SIP as signaling protocol and RTP as media transport protocol. The rest of this section discusses the attacks and the detection methodology for each.

6.1 **The BYE Attack**

As mentioned earlier, a BYE request can be sent by either the caller or the callee to terminate the session. An attacker can abuse this feature, taking advantage of the lack of authentication, by sending this message to either the caller or the callee to fool them into tearing 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 is considered as a Denial of Service (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 a SIP session upon receiving a BYE message. We set a value to the timer field in the signature. The recommended action field stores 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 III shows the signature. The quasi-code of BYE_Procedure( ) which is the recommended action appears in figure 3.

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

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

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.

6.3 The REGISTER Flooding Attack

A number of SIP clients can launch a REGISTER flooding attack to swamp a single registrar server within a short duration of time. REGISTER requests are accepted by registrar servers to store a binding between a user’s SIP address and the address of the host where the user is currently residing or wishing to receive requests. REGISTER flooding attack can be viewed as a DoS attack.

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

6.4 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.
network comprises two domains each with a Proxy and Registrar Server. Each domain also contains a set of User Agents (endpoints) which are connected to the servers by a 10Base-T Ethernet. We use the Audio/Video profile with minimal control (RTP/AVP), with UDP as the underlying protocol. Our payload type is static with the identification number 32, and the clock rate 8000 Hz. Endpoints in a domain make calls to other endpoints in the other domain randomly and without predefined durations. Our IDS is installed on all endpoints and servers in both domains. The Internet connection between the two domains is assumed to have a delay of 40 ms and a packet loss of 0.2%. We run the experiment for 120 minutes.

7.2 Detection Accuracy

During the experiment, a random number of each of the implemented attacks has been launched. The IDS with the help of the signature database has succeeded to detect all the instances of the launched attacks without missing any.

During the experiment, we have 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. The same rule applies to threshold detection which can be a major source of false alarms. The values of threshold fields are left to system administrators. Hence, system administrators can set these values in a way that reflects the conditions of the underlying network and the security relevance of the event to avoid unwanted false alarms.

7.3 Performance Evaluation

It is vital that any security measure to be implemented in a VoIP network does not impede the performance of the network. The implementation of various security measures in a VoIP network can introduce some complications that can degrade QoS. These complications range from delaying call setups to delaying delivery of data packets. Therefore, we measure the performance of the 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, and processing delay.

End-to-end delay in VoIP refers to the time it takes for a voice transmission to go from its source to its destination. The ITU-T G.114 standard describes that a 150 milliseconds one-way delay is acceptable for high voice quality [8]. Every element along the voice path adds to this delay. This includes switches, routers, and public Internet connections. Figure 5 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 milliseconds on average to the voice transmission delay. The operation of the signature database adds about 0.06 milliseconds on average to this delay. As shown in the figure, the overall delay remains considerably less than the upper bound of 150 milliseconds.

Call setup delay in VoIP environments is the period that starts when a caller dials the last digit of the called number and ends when the caller receives the last bit of the response. VoIP systems are expected to give a performance comparable of that of PSTNs. Users may be annoyed with a setup process that requires more than a few seconds. Figure 6 shows the call setup delay introduced by our IDS at a certain endpoint during the simulation. The hybrid IDS adds about 67 milliseconds to the call setup process, with about 2 milliseconds contributed by the operation of the signature database. Such an increase in the call setup time is tolerable by VoIP users. Furthermore, the overall call setup delay remains within the limit of one or...
attacks against the detection system in a simulator environment. Our database has shown promising efficiency detecting all the attacks and low runtime impact on the operation of hosts.

REFERENCES


