CERIAS Tech Report 2006-17

SPACEDIVE: A DISTRIBUTED INTRUSION DETECTION SYSTEM FOR VOICE-OVER-IP ENVIRONMENTS

by Vinita Apte, Yu-Sung Wu, Saurabh Bagchi, Sachin Garg, Navjot Singh

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

SPACEDIVE: A Distributed Intrusion Detection System for Voice-over-IP Environments

Vinita Apte, Yu-Sung Wu, Saurabh Bagchi
Dependable Computing Systems Lab
School of Electrical & Computer Eng
Purdue University
Email: {vapte,yswu,sbagchi}@purdue.edu

Sachin Garg, Navjot Singh
Avaya Labs

Email:{sgarg,singh}@avaya.com

Abstract

Voice over IP (VoIP) systems are gaining in popularity as the technology for transmitting voice traffic over IP networks. As the popularity of VoIP systems increases, they are being subjected to different kinds of intrusions some of which are specific to such systems and some which follow a general pattern of IP attacks. VoIP systems pose several new challenges to Intrusion Detection System (IDS) designers. First, these systems employ multiple protocols for call management (e.g., SIP) and data delivery (e.g., RTP). Second, the systems are distributed in nature and employ distributed clients, servers and proxies. Third, the attacks to such systems span a large class, from denial of service to billing fraud attacks. Finally, the systems are heterogeneous, have soft real time requirements, and are typically under several different administrative domains.

In this paper, we propose the design of an intrusion detection system targeted to VoIP systems, called SPACEDIVE. SPACEDIVE is structured to detect different classes of intrusions, including, masquerading, denial of service, and media stream-based attacks. It can be installed at multiple points – clients, servers, or proxies, and can operate with both classes of protocols that compose VoIP systems – call management protocols, e.g., the Session Initiation Protocol (SIP), and media delivery protocols, e.g., the Real Time Transport Protocol (RTP). SPACEDIVE proposes the abstraction of correlation based IDS and provides a rule language to express correlated rules. The correlation may be of information gathered from peer entities or entities at different levels. SPACEDIVE is demonstrated on a sample VoIP system that comprises SIP clients and SIP servers spread over two domains. Several attack scenarios are created and the accuracy and the efficiency of the system evaluated with rules meant to catch these attacks.

Keywords: Intrusion detection, Voice over IP system, Hierarchical alert correlation, Cross-protocol detection, Stateful detection, SIP, RTP.

1 Introduction

Voice over IP (VoIP) systems are gaining in popularity as the technology for transmitting voice traffic over IP networks. Along with the anticipated widespread adoption of VoIP systems comes the possibility of security attacks targeted against such systems. VoIP systems use a multitude of protocols, primarily *control protocols* for signaling, establishing calls, negotiating call parameters, and monitoring health of the ongoing call and *data protocols* for carrying the voice data over the IP network. The attacks can be thought of as a combination of traditional kinds of security attacks against IP networks and novel attacks enabled by the architecture of VoIP systems.

Let us first identify the key features of VoIP systems and the attacks against such systems that serve as the design motivation for an intrusion detection system (IDS). VoIP applications have soft real time requirements and therefore Denial of Service (DoS) attacks can take the form of overwhelming the system with excess traffic, enough to violate the timing requirements from the system. Thus, simply encrypting and authenticating the traffic

between the end points is not sufficient. Second, the attacks can span multiple protocols between different end points and be spread over arbitrary time periods. The symptoms of an attack are often manifested to an IDS by combining information from multiple sources, e.g., for a mobile client, by combining information from a VoIP location and registrar server with domain specific knowledge (speed of the mobile client). In typical deployments, a VoIP server may be expected to handle peak loads of several tens to hundreds of clients. Scalability with respect to the number of clients and the amount of traffic is thus an important requirement for VoIP servers and by extension, any IDS that operates in the critical path of these servers.

Considering a range of attack scenarios seen in practice, we observe that the attack symptom is often detectable only by correlating information from multiple sources. The correlation is required among information from multiple protocols at multiple end points. The correlation may need to be done from sources that are peers, such as, two communicating clients or across peer levels, such as, the communicating clients and the servers. In another dimension, the correlation may be needed between information across multiple calls – for example, a malicious client launching a toll fraud on multiple calls.

Naturally, one first looks to the sizable work in the area of correlation based intrusion detection systems to come up with the solution for the VoIP domain. Correlation based IDS's have the goal of clustering alerts from multiple detectors (in our case, the detectors resident on the different VoIP components) to come up with a combined alert or the determination of an attack. However looking into the details, we find that there is no solution that suits our purpose completely due to the following factors. We cannot assume that the correlation engine in a VoIP system will work on alerts from multiple detectors *at the same host*. - typically the alerts will come from detectors spread over multiple hosts. The hosts may in fact be located a large network distance apart and may be under different administrative domains. A basic premise in most correlation systems is to use similarity in the alerts to cluster them. Some of the primary attributes used are not applicable in VoIP systems – source IP (the client's IP address may change due to mobility), time (the attack packets may be spread out widely in time), and session ID (transport protocols, such as RTP, are mostly connectionless). Another dimension for correlation is the prerequisite and consequence of alerts, with the idea being that if the prerequisite of an alert matches the consequence of another then the two alerts can be chained. However alerts in our target VoIP system do not have *a priori* information about prerequisite and consequence attached to them. An aspect that works to ease the task of

correlation is that the individual alerts are generated by our system as well and thus we are relieved of the task of correlating alerts from a heterogeneous set of IDS's.

We propose the design of a system called SPACEDIVE to serve as the correlation-based IDS for VoIP systems. The implementation provides an instantiation of the design principles, which are applicable to the design of any IDS for VoIP applications. Our solution centers on three basic principles.

1. *Rule matching engine at the local and remote levels*. The system includes rule matching engines located at the individual VoIP components (local rule matching engine, or RME_L) as well as remote rule matching engines (RME_R). Thus fast matching of local attack patterns can be done together with correlation of attack patterns arising from multiple sources. The placement of the RME's is flexible as is the generation and transfer of information needed for the RME_R's.
2. *Cross protocol and stateful detection*. These principles were present in our earlier version of the system, SCIDIVE [1] and are maintained in the distributed architecture as well. Stateful detection denotes the functionality of assembling state from multiple packets and using the aggregated state in the rule-matching engine. Cross protocol detection denotes the functionality of matching rules that span multiple protocols. However, SCIDIVE did not provide any rule language for specifying these patterns while SPACEDIVE does. Thus the system takes the rulebase as an input easing the task of applying SPACEDIVE to new deployments.
3. *Integration with Snort*. SPACEDIVE needs to examine packets coming in at a host executing a VoIP component. The packet rate may be high and therefore fast matching of the rules locally and fast processing to generate events for correlation are required. The Snort IDS is well known for its efficiency in examining incoming packets and SPACEDIVE leverages the Snort functionality. To avoid performance loss, SPACEDIVE is built *into* Snort using part of its low-level functionality (examining and processing packets) and adding to it (e.g., to build state to support stateful detection) and building completely the high level functionality specific to the VoIP environment.

The typical process flow for the detection of an attack in SPACEDIVE is as follows. At the local rule matching engine (RME_L), an incoming packet is sniffed and passed through local rules. These local rules are specified in a language derived from Snort's and augmented with constructs to create state. The match generates an event for the local event trail and optionally state associated with the event. An event parser next parses the local events to

optionally generate a network event. A network event is synonymous with an event that needs to be aggregated with events from other VoIP components for matching at an RME_R. At the RME_R, there exists a rulebase for network events specified in a high level language introduced in this paper. The network events from the individual RME_L's can be either pushed to or pulled by the RME_R, depending on the nature of the event and the rulebase at the RME_R.

SPACEDIVE is demonstrated on a sample VoIP system that comprises SIP clients, SIP servers (gateway, proxy, and registrar servers), and support servers (DNS, FTP servers, etc.). The system uses SIP Express Router for the servers [12]and X-Lite [13] as the client. The protocols used are SIP for call management and RTP for real-time audio data transfer. The clients and servers are spread over two different administrative domains. A representative set of attacks based on control traffic, data traffic, and cross protocol are developed and injected to test SPACEDIVE. The experiments bring out the performance of SPACEDIVE in terms of processing and matching incoming VoIP packets and compare this to baseline Snort. The use of SPACEDIVE makes the VoIP system more resilient to DoS attack by a rogue client compared to the baseline unprotected VoIP system.

In summary, the contributions of the paper and the advantages of SPACEDIVE can be specified as follows:

1. SPACEDIVE presents the architecture of a hierarchical correlation based IDS that is well suited to detecting attacks in VoIP applications. The ability to match rules remotely makes the system less prone to DoS attacks launched against VoIP components or their hosts.
2. SPACEDIVE provides a language to specify rules for local matching and remote matching. SPACEDIVE's architecture makes the matching of rules, efficient and scalable, both essential features for a VoIP system.
3. SPACEDIVE anticipates the growing trend of peer-to-peer VoIP systems and its architecture is well suited to detecting intrusions on the basis of information exchanged on a peer-to-peer basis.

The rest of the paper is organized as follows. Section 2 gives an overview of VoIP systems and prior work in VoIP security. Section 3 presents the architecture of SPACEDIVE and motivates the stateful, cross-protocol, and hierarchical correlation based detection through running examples. Section 4 presents the algorithms of SPACEDIVE and their complexity. Section 5 presents different attack scenarios and their detection strategy and the experimental results. Section 6 concludes the paper.

2 Background and Related Work

Voice over IP systems provide facilities for setting up and managing voice communications based on one of two main protocols: H.323 [2] and SIP [3]. H.323 is the most widely deployed standard in VoIP communications, but SIP is increasing in popularity due to its simplicity and ease of implementation. With both types of systems, endpoints or terminals send and receive RTP [4] packets that contain encoded voice conversations. Since voice calls may be made between IP phones and phones on the Public-Switched Telephone Network (PSTN), gateways often perform transparent translation between IP and non-IP based networks. Such gateways may implement protocols for media gateway management such as MGCP [5] and MEGACO/H.248 [6].

SIP networks also include additional types of servers. The *proxy server* is an intermediate device that receives SIP requests from a client and then forwards the requests on the client's behalf. The *redirect server* provides the client with information about the next hop or hops that a message should take and then the client contacts the next hop server directly. The *registrar/location server* processes requests from SIP clients for registration of their current location. Figure 1 shows a sample VoIP architecture using SIP.

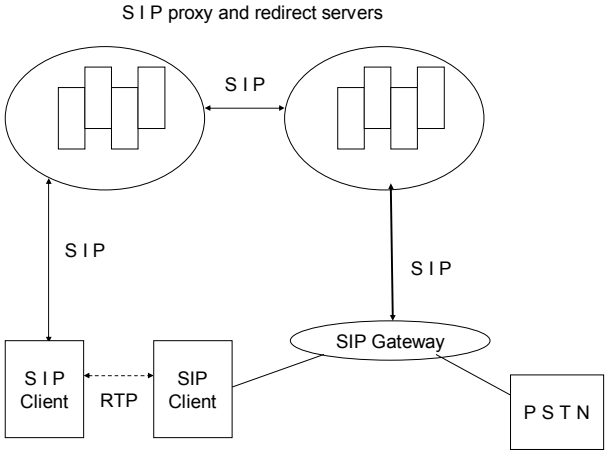


Figure 1:SIP-based VoIP architecture

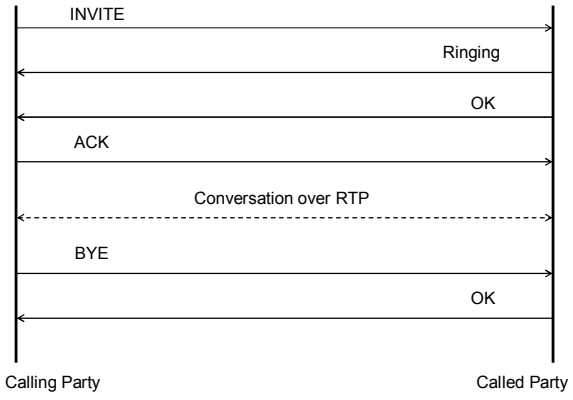


Figure 2: Sample SIP messages for a SIP session

Both H.323 and SIP provide functionality for call setup, management, and media delivery. SIP uses a simple set of request messages: INVITE, ACK, OPTIONS, BYE, CANCEL, and REGISTER. SIP provides a globally reachable address to which callees bind using SIP REGISTER method. The INVITE message is used by a user client agent wishing to initiate a session, which can be responded to with an OK, followed by an ACK. To tear down a connection, a BYE message is sent. Figure 2 illustrates a typical set of SIP messages leading to a session.

The Real Time Transport Protocol (RTP) is the most common media delivery protocol used in VoIP systems. RTP itself does not guarantee real-time delivery of data, but it does provide mechanisms to support streaming data. Typically, RTP runs on top of UDP. The data transport is augmented by a control protocol – the Real Time Control Protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality.

Security of voice networks is gaining considerable attention from VoIP service providers and vendors. This has manifested itself in the form of the VoIP Security Alliance (voipsa) [7] which has as its members many VoIP service providers, researchers and vendors. VOIPSA's mission is to drive adoption of VoIP by promoting the growth of VoIP security research, education and awareness. The Secure Real-time Transport Protocol (SRTP) [8] provides confidentiality, message authentication, and replay protection for the Real-time Transport Protocol (RTP). However, despite using SRTP, RTP remains vulnerable to denial of service attacks. Rosenberg states in [9] that to launch a denial of service attack, the attacker simply obtains the IP address of the attack target and sets up a substantial number of RTP sessions using SIP or RTSP. In the case of SIP, it would do so by sending an INVITE with a Session Description Protocol (SDP) [11] body.

There are very few intrusion detection systems specifically built for voice networks. BLAZE [9] is an intrusion detection architecture which proposes a mobile agent paradigm for intrusion detection, but it has not been implemented and the paper gives no system details. It shares with us the goal of correlating alerts from different components in a VoIP system. However, its only mention of correlation is sending mobile agents to collect traces of data from remote sites suspected in connection with an attack. Our previous work SCIDIVE [1] is an intrusion detection system which makes use of the stateful and cross protocol detection approach. However, it is neither distributed nor correlation-based. There are very few data sources for intrusions in VoIP systems [20][21], though general discussions of attack and solution strategies abound. Therefore our attack scenarios are hypothesized based on conversations with VoIP service providers.

Distributed Intrusion Detection Systems multiply the power of a single IDS by detecting attacks based on events obtained from a large number of geographically dispersed agents. The main idea is that both the components and the decision making are distributed. Our previous work on a system called CIDS [15] falls in this category. *Correlation-based IDSs* propose a system model for correlating alerts from multiple elementary detectors to

perform more accurate intrusion detection [16]-[19]. However existing correlation based IDSs do not readily lend themselves to our problem domain as mentioned in the Introduction.

We build SPACEDIVE over the Snort intrusion detection system [10]. Snort is an open source network intrusion prevention and detection system utilizing a rule-driven language, which combines the benefits of signature, protocol and anomaly based inspection methods. We use the fast packet filtering and low level rule matching functionalities of Snort.

3 SPACEDIVE Design

3.1 SPACEDIVE Design Hierarchy

The SPACEDIVE design can be broken down in two parts – the local-level design and the network-level design. Local-level design involves a single VoIP component (client, proxy, etc.) and has the local Rule Matching Engine (RME_L). Network–level design takes into consideration all the components deployed in one domain or across multiple domains and the interactions between them and provides the remote Rule Matching Engine (RME_R). Providing separate IDSs at the local level and at the network level plays a large part in keeping SPACEDIVE scalable.

3.2 SPACEDIVE Local Level Design

Figure 3 shows the SPACEDIVE components at the local level. The components below the dashed line represent an instance of SPACEDIVE installed on each VoIP component and integrated with Snort. The *sniffing module* makes use of the libpcap library to read packets received over the network. Currently, Snort understands 4 protocols: IP, TCP, UDP, and ICMP. We have modified Snort so that it now identifies SIP and RTP packets too.

The State Repository stores the current state of the system. State comprises the status of an ongoing session – i.e. connecting, established, terminated, etc., the status of a node, e.g., if the node has moved, or the reception of a particular type of packet (e.g. a SIP BYE message). The *Event Trail* keeps track of events, specified using the low level rule language. The event trail contains events ordered by session ID. For events that have no associated session ID, such as RTP packets, the timestamp is used for ordering. The *Processing Engine* determines whether a pre-defined event has occurred and records it in the event trail. It also updates the state of the rule variables in the *State Repository*. The *Event Parser* takes the event trail as input and generates a trail of "Network Events". What constitutes a Network Event is specified in the RME_R, which disseminates the pertinent network event definitions

to the local RMEs. The RME_R uses the Network Event Trail to correlate events across the different components of the network.

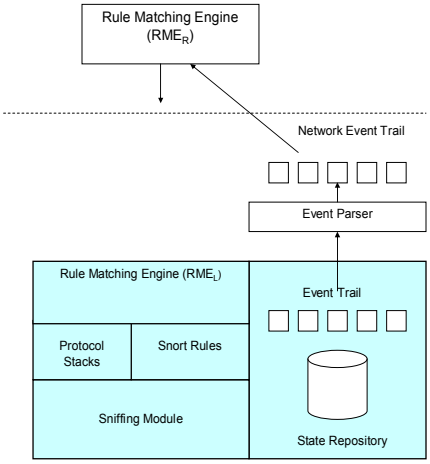


Figure 3: Local-level SPACEDIVE design
(Components below the dotted line are local)

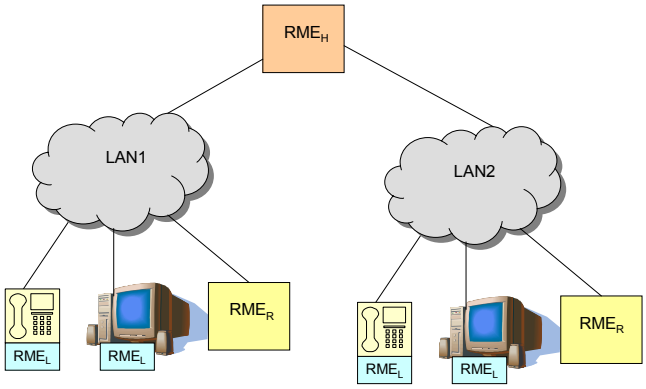


Figure 4: Rule Matching Engine Hierarchy.

3.3 SPACEDIVE Network Level Design

At the network level, SPACEDIVE views the system as composed of multiple VoIP domains, each with its own RME_R (Figure 4).

The RME_R's perform remote rule matching from network events generated by each RME_L in its domain. Each RME_R comes with a configuration script that gives it the following information – the IP address and hostname of all the clients, servers, and proxies in its domain. The RME uses this configuration information for network level rule matching.

We have developed a high-level rule language for specifying network level events in the RMEs. At the network level a rule R can be represented as:

$R = ((where_i{:}what_i)\ conn_i)\ response \quad (i = 1, \ldots, N)$

In R, $where_i$ denotes the location of event i, $what_i$ denotes the network level event i, $conn_i$ is the connective between the events and may be AND, OR, NOT, BEFORE, AFTER, or NULL. The clause *response* indicates the response to be taken in the case of a match and is currently limited to an alert or dropping a packet. The RME extracts the event specifications ($what_i$) from the rule pertaining to a particular node ($where_i$) and disseminates them to that node at start-up. These event specifications are then used by the event parsers in the RME_L's to

generate network events. For example, consider a VoIP call established between clients A and B. An attacker C sends a faked SIP BYE message to A to make it believe that B wants to tear down the connection. Since B does not know of this attack, it will continue sending RTP packets to A. To detect this attack, we look for an orphan RTP flow at A after the BYE message is received. We can frame the high level rule as:

```
(clientA:RTP_Flow) AFTER (clientB:BYE_Sent) alert
```

To make the detection infrastructure scalable, the RMEs may be arranged in a hierarchy as shown in Figure 4 correlating information across domains. In this hierarchy, RME_H is a higher level RME that can look at rules corresponding to multiple administrative domains. Driven by the realization that different parts of a VoIP system may be owned by different organizations, SPACEDIVE has the capability to accommodate multiple peer-level RME_R's. The policies and the resultant rules in each administrative domain may be encapsulated within the RME_L's and the RME_R without the need to share them. A third-party organization (such as, VOIPSA) may own the RME_H which matches for rules that affect multiple organizations.

3.4 Low level rule language

Since the RME_L is built on Snort, we make use of the Snort rule language to match incoming packets at each VoIP component to generate the local events. A rule can be broken down into two basic parts, the rule header and options for the rule. The rule header contains the action to perform, the protocol that the rule applies to, and the source and destination addresses and ports. The rule options is used to create a descriptive message to associate with the rule, as well as check a variety of other packet attributes by making use of Snort's extensive library of plug-ins.

The general form of a local rule is: *action proto src_ip src_port direction dst_ip dst_port* (*options*).

There are four major categories of rule options:

meta-data These options provide information about the rule but do not have any affect during detection

payload These options all look for data inside the packet payload and can be inter-related

non-payload These options look for non-payload data

post-detection These options are rule specific triggers that happen after a rule has "fired."

Some important rule options are 'content' (allows the user to set rules that search for specific content in the packet payload and trigger response based on that data) and 'priority' (assigns a severity level to rules.)

When a packet comes in, its source and destination IP addresses and ports are compared to the rules in the ruleset. If any of them are applicable to the packet, then the options are compared to the packet. If these comparisons return a match, then the specified action is taken.

The native rule language of Snort is not well-suited for stateful or cross-protocol detection. Snort provides very limited capability for remembering state both within a session for a given protocol and across protocols. To make up for this, we add constructs to the existing rule language so that it is better-suited for detecting attacks targeted to VoIP environments that span packets in a session and different protocols. Next, we provide a brief description of the new constructs.

(a) **var**. This construct is used to set the integer value of a variable in case of a rule match. This is used as a way of keeping state. The var construct belongs to the 'options' part of a Snort rule.

(b) **Event.** The event construct is used to create event trails. It tells Snort to record an event when the corresponding rule-match occurs. An event can be triggered on a combination of rule matches according to the following constructs.

(c) And/Or/Not – Logical Constructs. These constructs are used to trigger an event based on logical combinations of rule matches;

(d) **Before/After – Temporal Constructs.** The Before and After constructs are used to trigger events based on a temporal sequence of rule matches.

(e) **Net_Event.** This construct follows the same syntax as 'Event' except that it is used to represent a network event as opposed to a local event.

(f) **Protocol-specific constructs.** To detect certain attacks we need to look into specific fields in the header of a protocol. For example, we may need to know the window of allowable sequence numbers for an RTP packet. This leads us to define a construct called 'seqwin' that represents the in-range sequence numbers of RTP. This means that now SPACEDIVE needs to parse the VoIP protocols (currently SIP and RTP) in addition to the four protocols (IP, ICMP, TCP, UDP) Snort is currently able to parse. This is more difficult especially for RTP, because RTP packets do not have a fixed length and they do not contain any string identifier to identify them as RTP packets. SPACEDIVE uses the port numbers negotiated via SIP to identify an RTP session. Currently, we limit our SIP

parsing to the session id and the RTP port fields contained in the SDP header of SIP since these fields suffice for the range of attacks we consider; while in the case of RTP, we parse the entire header.

Sample Rule: Suppose that an attack pattern consists of packet A containing the string "DESTROY" followed by packet B containing the string "ERASE". Assume that packet A is on port 5000 and packet B is an RTP packet on port 6000. Then we construct the rule as follows, which will log an event in the local event trail if both packets are found.

```
var r1; var r2;
alert udp any 5000 -> any any (content:"DESTROY"; var:r1);
alert rtp any 6000 -> any any (content:"ERASE"; var:r2);
event (r1 AND r2);
```

4 Algorithms in SPACEDIVE

4.1 Local Level Event Generation

At the local level, all rules specified in the rules file are parsed to form an expression tree. For example, consider the rule *event ((rv1 AND rv2) OR rv3 AND rv4)*. The corresponding expression tree for this rule is Figure 5(a).

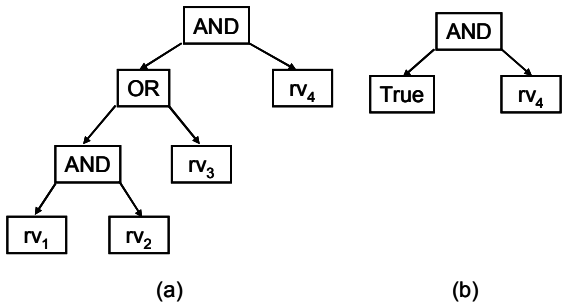


Figure 5. Example rule processing at RME_L

Each variable (rv_1, rv_2, rv_3, rv_4) in the rule will have a pointer to the root of this expression tree stored with it. Note that a variable may be involved in multiple rules. In that case, we store a list of root pointers with each variable.

Whenever a variable is set to 1 after a rule match, we need to evaluate the expression tree. For example, in the above case, when rv_3 is set to 1, we find that the value of the OR expression (the parent of rv_3) becomes TRUE, irrespective of the values of rv_1 and rv_2. So we can simplify the tree structure by rolling up the sub-tree rooted at the OR as shown in Figure 5(b).

Assuming a total of r rules and an average of v variables in each rule, the total number of variables at RME_L is $V=r.v$. The search operation is $O(\log V)$ time using binary search. Let H be the height of the original expression tree before any roll-up. The worst case for locating the variable in the tree is when the tree is unbalanced with height $H = v-1$. To locate a variable in the expression tree of height i, we need $O(i)$ operations. In the worst case, we assume that setting each variable in the tree to 1 decreases the height of the tree by just one, i.e. no tree roll-up

is possible at any stage. This would happen for example if all the connectives are AND. Thus, to evaluate the entire expression we get a complexity

$$O(\sum_{i=1}^{H}(\log V + i)) = O(H.\log V + H^2) \quad = O(v.\log V + v^2) = O[v.\log v + v.\log r + (\log v)^2]$$

If $r < v$, then the operation is $O(v^2)$.

4.2 Network level event generation

The event parser is loaded with a rules file at start-up by pushing this information from the RME_R that tells it what events to look for in generating the network events. It then matches these rule definitions with the local event trail to generate network events. The search through the event trail is linear, not logarithmic, since the events are generated at runtime and the sorting to enable binary search is not possible. Consider that the RME_R manages M RME_L's. The number of remote events for each RME_L follows a uniform distribution U(LR, UR). Let the buffer size for the local event trail be L. In the worst case, there will be UR rules, the local event trail buffer is full, and each rule will have to be checked against the entire buffer. This gives a worst case complexity for the remote event generation as $O(UR \times L)$.

4.3 Processing at RME_R

The RME_R also generates an expression tree from the high level rules. The rules are pointed to by a hash table of size M, each entry corresponding to an RME_L. Assuming r rules at the RME_R, the number of what clauses (equal to the number of nodes in the expression tree for one rule) is $h = kM/r$, where $k = ½(LR+UR)$. Therefore the cost of matching one rule is $O(\sum_{i=1}^{h}(c + i)) = O(h^2)$.

5 Demonstration and Experiments

5.1 Experimental Testbed

Figure 6 shows the layout of our testbed. To realistically simulate a VoIP environment, we have built a testbed with two domains. This enables us to demonstrate intra domain calls as well as inter-domain calls. Each domain has a SIP gateway, a proxy server, a registrar server, clients and support servers like FTP, DNS etc.

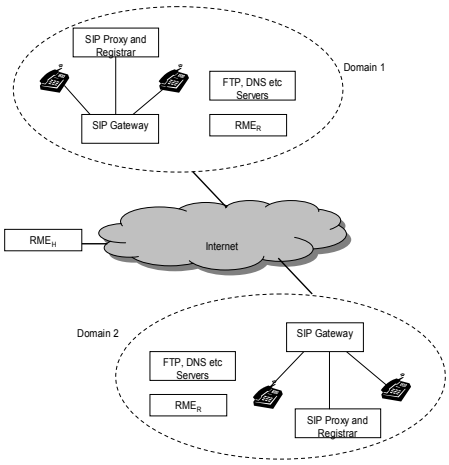


Figure 6: SPACEDIVE Testbed

The SIP clients and servers are equipped with the SPACEDIVE IDS. We use the SIP Express Router (ser) [12] for the SIP servers. Ser can be configured as a SIP registrar or proxy server. Our SIP clients are Windows based and use X-Lite [13]. In the testbed, we have the gateway, registrar and proxy server running on the same machine. We deploy RME_H in domain 1 though in practice it can belong to either domain or be in a separate domain altogether.

5.2 Workload

The normal workload consists of the scenario where client 1 from domain 1 makes a call to client 2 in domain 2. When client 1 initiates a call, a SIP request is sent to the SIP server (either a proxy or a redirect server) in domain 1 with the addresses of the caller and the callee. If a proxy server is used, client 1 sends an INVITE request to the proxy server, the proxy server determines the path, and then forwards the request to client 2. Client 2 responds to the proxy server, which in turn, forwards the response to client 1. The proxy server forwards the acknowledgments of both parties. A session is then established between the two clients. The communication between the caller and the callee happens through RTP packets. The SIP Gateways provide call control.

5.3 Attack Scenarios

This section describes several possible attack scenarios on a VoIP system. In all the attack scenarios, S1, S2 are two SIP proxies overseeing two different domains, A and B are legitimate clients, while H is a malicious client. To compare and contrast the power of a local IDS (SCIDIVE) and SPACEDIVE, we give the steps in detecting each attack scenario by the two systems. Fragments of the relevant rules from SPACEDIVE are also given.

At the end of the section, we classify the attack scenarios in two dimensions and summarize the detection in the two systems.

5.3.1 Call Hijacking

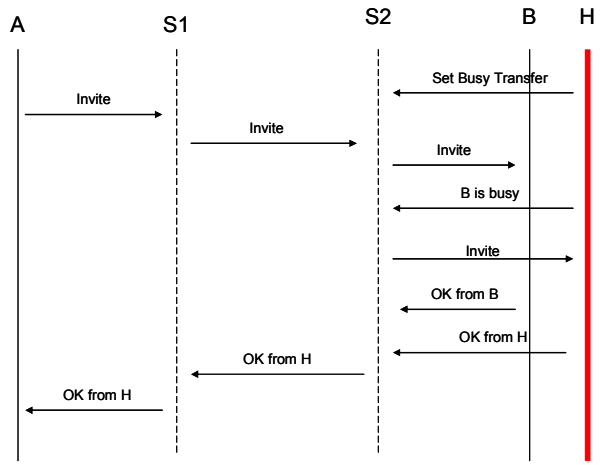


Figure 7: Call Hijacking

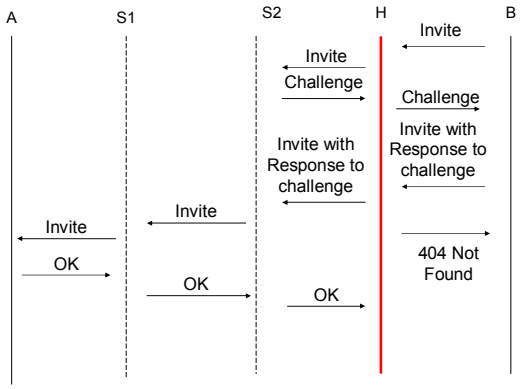
In the Call Hijacking attack, we assume that H is at a place on the network where she can sniff the traffic from B. A possible case is that B and H happen to be using the same network hub. In the attack, H first sends a busy transfer request to server S2 such that any call to B will be transferred to H when B is busy. Now A places a call to B. H is able to sniff the Invite message from S2 to B and responds with a 'B is busy' message back to S2 before B is able to reply to S2. S2 sends a new invite message to H, thinking that B is busy and the call should be transferred to H. Although B's ok message will eventually go to S2, many server implementations will regard this as a noisy reply. On Openser, with the default configuration, this ok message will also be forwarded to A, which is then ignored by the X-Lite client. Thus H hijacks the conversation and can collect confidential information that A wants to pass to B.

SCIDIVE Detection: This can be detected by a SCIDIVE component placed at S2 which detects noisy OK replies. Specifically, when SCIDIVE sees a 'B is busy' message at S2, then an 'Ok' message from B should be regarded as a sign of a call hijacking attack.

SPACEDIVE Detection: A SPACEDIVE rule checks if the OK reply from B goes correctly all the way from B to A. The correlation is done across events at B, S2, S1, and A. This falls in the general class of rule called *end-to-end matching* in SPACEDIVE, where the correlation is done across events at each component in the path. This is a powerful rule class and can detect many different kinds of attacks.

Component	Rule Snippet
A	`alert sip any any -> any any (var:rv1; content:INVITE;); net_event (rv1;)`
B	`alert sip any any -> any any (var:rv2; content:OK;); net_event (rv2;)`
S2	`alert sip any any -> any any (var:rv3; content:OK;); net_event (rv3;)`
RME	`(A:SIP_SESSION_ESTB) AND (B:SIP_OK) AND (NOT(S2:SIP_OK)) alert`

5.3.2 Man in the middle attack : intercepting outgoing calls



In this attack, we assume H is on the route between S2 and B. The goal for H is to intercept outgoing calls from B. The INVITE messages are authenticated through a challenge-response mechanism. As B places an outgoing call, the attacker H forwards the INVITE messages and the challenge-responses between S2 and B until the authentication phase is completed.

Figure 8: Man in the middle attack

Then H fakes a '404 Not Found' message back to B such that B thinks A is not present. In effect a call is established between H and A with H representing itself as B.

SCIDIVE Detection: SCIDIVE is not able to detect this attack, since the message exchanges that happen at A, S1, S2, and B can all be part of a legitimate call signaling process. Furthermore, since H is the malicious client, SCIDIVE cannot be placed on H.

SPACEDIVE Detection: This can be detected by SPACEDIVE with an end-to-end matching rule for the OK message going correctly all the way from A to B through S1 and S2.

Component	Rule Snippet
S1	`alert sip A any -> any any (var:rv1; content:OK;); net_event (rv1;)`
S2	`alert sip S1 any -> any any (var:rv2; content:OK;) net_event (rv2;)`
B	`alert sip S2 any -> any any (var:rv3; content:OK;) net_event (rv3;)`
RME	`(S1:SIP_OK) AND (S2:SIP_OK) AND (NOT(B:SIP_OK)) alert`

5.3.3 BYE Attack

In this attack, H's goal is to prematurely tear down a current call session between A and B. For the attack, H sends a BYE to A, which will trick A into tearing down the dialog with B. Here, either the session is not secure, or if it is secure, H is able to masquerade as B through some vulnerabilities in the system.

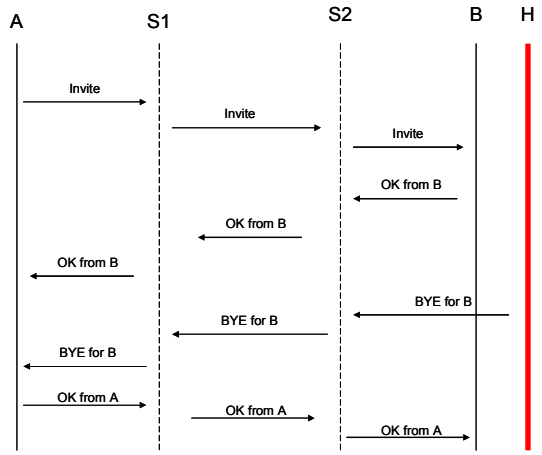


Figure 9: BYE Attack

SCIDIVE Detection: Check that the RTP stream from B to A has ceased after the BYE message is seen at A. However the RTP stream may happen to be nonexistent during the detection window possibly due to silent suppression at B leading to a missed alarm. An alternative rule is by checking spurious OK message from A at B. The OK message from A carries the information that it is in response to a BYE request. However, by the time SCIDIVE detects the attack, it has been launched and has potentially caused some damage.

SPACEDIVE Detection: This can be detected by SPACEDIVE with an end-to-end matching rule for the BYE message from B→S2→S1→A. In this attack scenario, the part of B→S2 is missing.

Component	Rule Snippet
S2	`alert sip B any -> any any (var:rv1; content:BYE;) net_event (rv1;)`
S1	`alert sip S2 any -> any any (var:rv2; content:BYE;) net_event (rv2;)`
A	`alert sip S1 any -> any any (var:rv3; content:BYE;) net_event (rv3;)`
RME	`(NOT(S2:SIP_BYE)) AND (S1:SIP_BYE) AND (A:SIP_BYE) alert`

5.3.4 Compromised SIP Proxy

In this attack (Figure 10), the server S2 it is compromised by H. H impersonates B by sending from S2 an OK reply purportedly from B as the reply to A's invitation for initiating a dialog.

SCIDIVE Detection: This attack cannot be detected in SCIDIVE since with a compromised S2, it's likely that the SCIDIVE IDS at S2 may also be compromised or disabled. This points to a fundamental advantage of a correlation based IDS over a single host one –more eyeballs are better for detection, *if* properly harnessed.

SPACEDIVE Detection: This can be detected by SPACEDIVE with an end-to-end matching rule for the INVITE message from A→S1→S2→B. In this attack scenario, the part of S2→B is missing.

Component	Rule Snippet
S1	`alert sip A any -> any any (var:rv1; content:INVITE;) net_event (rv1;)`
S2	`alert sip S1 any -> any any (var:rv2; content:INVITE;) net_event (rv2;)`
B	`alert sip S2 any -> any any (var:rv3; content:INVITE;) net_event (rv3;)`
RME	`(S1:SIP_INVITE) AND (S2:SIP_INVITE) AND (NOT(B:SIP_INVITE)) alert`

5.3.5 Billing Fraud

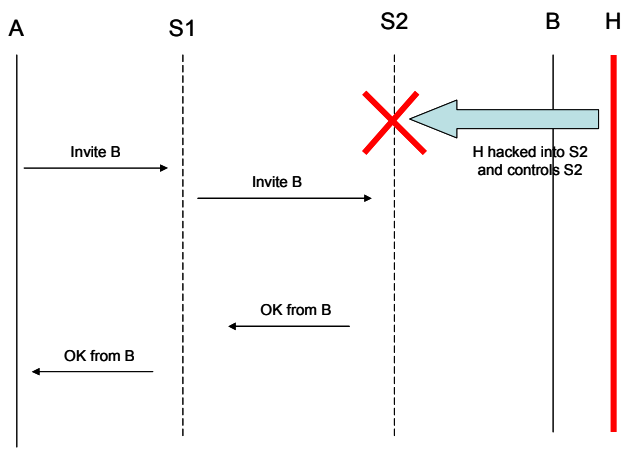


Figure 10: Compromised SIP Proxy

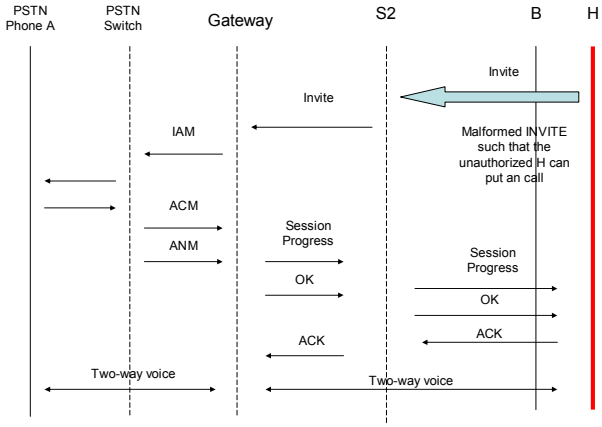


Figure 11: Billing Fraud

Assume a hybrid system with SIP based VoIP telephony and traditional PSTN telephony. The accounting of the relaying calls is done at the SIP proxy S2. Therefore, only authorized users from the VoIP side are able to make calls to a PSTN phone. Here, we assume that the attacker H exploits a vulnerability at S2, such that by sending a malformed INVITE message, she is able to impersonate B and place a call to the PSTN phone. The messages in the PSTN system used in the call setup are IAM: initial address message, ACM: session progress, ANM: ok message [14].

SCIDIVE Detection: A SCIDIVE IDS at S2 may use signature based rule checking to decide whether the incoming INVITE message contains malformed data. A weakness of this approach is that the signature is very specific and there may be a large number of ways of exploiting vulnerabilities in S2 through malformed INVITE messages.

SPACEDIVE Detection: In SPACEDIVE, detection components installed on both S2 and B check whether B did send out the INVITE message.

Component	Rule Snippet
B	`alert sip A any -> any any (var:rv1; content:INVITE;) net_event (rv1;)`
S1	`alert sip B any -> any any (var:rv2; content:INVITE;) net_event (rv2;)`
RME	`(NOT(B:SIP_INVITE)) AND (S1:SIP_INVITE) alert`

5.3.6 Denial of Service (DoS) Attack

An attacker can launch a denial of service attack by flooding the servers signaling port, flooding the media proxy's listening port or by flooding the clients media or signaling port.

SPACEDIVE Detection: SPACEDIVE can detect a DoS attack targeted at a client's media port. To detect a DoS attack launched on host A we have the rule: `alert rtp any any → A any (seqwin:50; var:rv;); net_event (rv;).` This rule will generate an alert if the RTP sequence numbers of two consecutive packets in the same session are off by +50 or -50 units. The detection in SCIDIVE follows the same principle though the rule has to be hard coded in the system.

In Table 1, we classify the attack scenarios in two dimensions – whether secure VoIP protocols can nullify the attack and whether the attack is dependent on position of H vis-à-vis A and B. A "Yes" indicator in a column indicates the attack is more difficult to execute. Finally, the table gives whether the attack scenario can be detected in SPACEDIVE and SCIDIVE.

Table 1: Classification of attack scenarios

Attack Scenario	Mitigated by secure VoIP protocol; Dependent on position of malicious client	Detected by SCIDIVE; Detected by SPACEDIVE
Call hijacking	YES; YES	YES; YES
Man in the middle	NO; YES	NO; YES
Bye Attack	YES; NO	YES; YES
Compromised SIP proxy	NO; NO	NO; YES
Billing fraud	YES; NO	MAYBE; YES

Note that all the above attacks are active attacks. SPACEDIVE cannot detect passive attacks like eavesdropping. This is an inherent limitation of the current IDS technology in general.

5.4 Experimental Results

5.4.1 Timeline for correlated detection

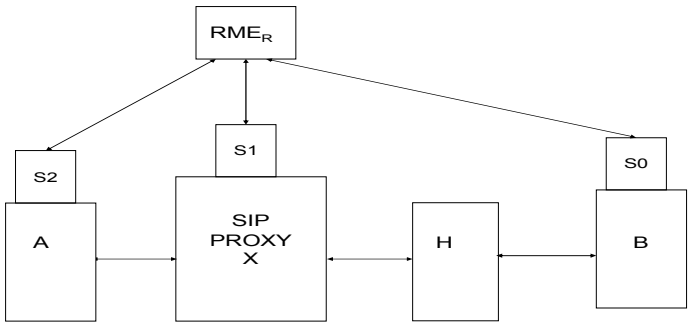


Figure 12: Experimental Setup

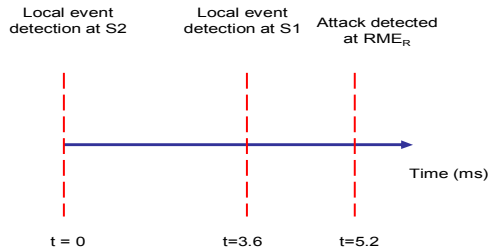


Figure 13: Timeline for Remote Rule Matching
(times are not drawn to scale)

For this experiment, we use the Man in the Middle Attack [Section 5.3.2] to demonstrate the timeline of the correlated detections among SPACEDIVE components. The layout of the testbed is shown in Figure 12. We use an Openser SIP proxy (say, X) and two X-Lite SIP softphones on A and B. The attacker H, which is a simple home made proxy, relays the traffic between B and X. We have SPACEDIVE RME_L's deployed on A, B, and X, and the

RME_R on a separate host. B first tries to make an outbound call to A and begins by authenticating with X. The authentication process ends with an INVITE message from X to B. Until this point, H faithfully relays all messages exchanged between X and B. After this, H immediately sends a '404 Not Found' message back to B and stops relaying any message between B and X. Once A gets the INVITE message, she will reply with an OK message, which should go all the way back to B for establishing the call. However, since H has sent the '404 Not Found' message instead of the OK message to B, B will never see an OK message and will assume that A is not present on the other side. The way we detect this attack is by putting local rules at *S0, S1, and S2* such that when an OK message is seen, the local detector will send an event to the RME_R. The rule at RME_R is then to check whether there are three OK messages from *S2*, *S1*, and *S0* in sequence within a bounded time window – we use a configurable window of 4 ms from the time the first event is received. In this case, *S0* does not see the OK message, which will then trigger the man in the middle detection alert. The timeline for these detections is shown in Figure 13.

5.4.2 Performance of rule matching

In this experiment we compared the rule matching overhead of SPACEDIVE for different rule types. We tested four categories of rules, each of which involves content matching for a string in the payload.

Type 0 is rule matching in Snort, while Type 1 is a vanilla Snort rule matched in SPACEDIVE. Type 2 rules use the `var` construct to set the value of a variable in the state repository. Type 3 rules are ones that use the event construct to create a local event in the event trail. We show cases of using 1, 2, 4, and 8 variables to construct the event. Figure 14 shows the comparative performance of the four categories of rules.

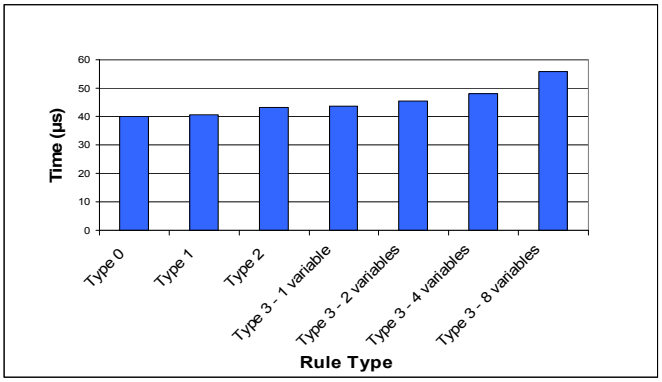


Figure 14: Processing time for classes of SPACEDIVE rules.

As seen from the figure, rule matching for Type 1 rules incurs negligible overhead. Type 2 rules involve about 8% overhead over Type 0. Type 3 rules with 1 variable fare almost the same as Type 2 rules. This is quite intuitive, since the expression tree for one variable is just a single node (the root of the tree). As expected, the performance cost increases with the number of variables. For a Type 3 rule with 8 variables, the overhead is, on

an average about 40%. This results from the fact that an expression tree with 8 variables has a depth of at least 3. This increases the time it takes to perform the tree rollup.

5.4.3 Resilience to DoS

DoS attacks are crucial ones in VoIP systems since secure VoIP protocols such as sRTP do not protect against them and the traffic is delay-sensitive. The objective of this experiment is to test the resilience of a VoIP client to a DoS attack in the baseline system and equipped with SPACEDIVE. A malicious client M sends garbage RTP packets to a receiver A concurrently with a legitimate sender client B. Client A requires service at the rate of 64 kbps for acceptable voice quality. This is used to normalize the quality of service in the presence of the client M.

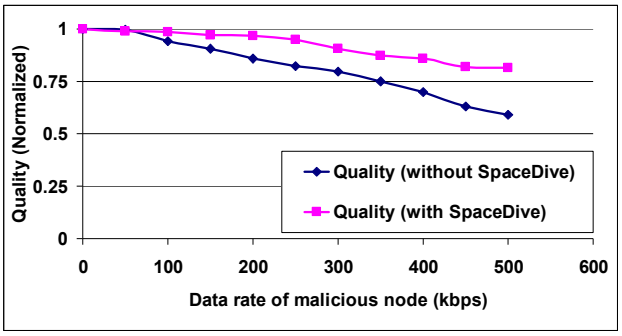


Figure 15: Resilience to DoS Attacks

Figure 15 shows the degradation in the quality as M is able to pump more DoS traffic to A. Since SPACEDIVE is configured to drop garbage RTP packets (from M), A's degradation in quality is much less steep. However, the processing to match the rule for RTP packets causes some degradation.

6 Conclusions

We have presented the design and implementation of a correlation based IDS for VoIP systems called SPACEDIVE. SPACEDIVE places local rule matching engines (RMEL) at individual VoIP components and a remote rule matching engine (RMER) for each domain to correlate events across components. The RMEL leverages the fast packet matching capability of Snort and augments Snort's rule language to perform stateful and cross-protocol detection. SPACEDIVE presents a flexible, easy to parse, high level rule language to match network events. Several attack scenarios are presented and classified to bring out the power of a correlation based IDS for VoIP. The design is implemented in a testbed and demonstrated to increase the resilience of a VoIP client to DoS attacks. The cost of the rule matching is also quantified.

In ongoing work, we are looking at techniques to reduce false alarms from the RME_L's through correlation, use of the RME_H to have rules private to an organization, and negotiation protocol between the RME's automatically generated from the rulebase.

References

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

[2] ITU-T, "Packet-based multimedia communications systems," Recommendation H.323, February 1998.

[3] M. Handley et. al., "SIP: Session Initiation Protocol," RFC 2543, March 1999.

[4] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications,"IETF, RFC 3550, July 2003,

[5] M. Arango *et. al.*, "Media Gateway Control Protocol (MGCP) Version 1.0," RFC 2705, October 1999.

[6] F. Cuervo *et. al.*, "Megaco Protocol Version 1.0," RFC 3015, November 2000.

[7] VoIP Security Alliance, www.voipsa.org.

[8] M. Baugher *et. al*."The Secure Real-time Transport Protocol (SRTP)," RFC 3711, March 2004

[9] K. Singh and S. Vuong, " BLAZE: A Mobile Agent Paradigm for VoIP Intrusion Detection Systems," Proceedings of ICETE 2004 - First International Conference on E-Business and Telecommunication Networks, Setubal, Portugal, August 2004.

[10] The Snort Intrusion Detection System, www.snort.org

[11] M. Handley and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998

[12] SIP Express Router (ser), http://www.iptel.org/ser/

[13] X-Lite, http://xten.com/index.php?menu=X-Series

[14] Carrier Grade Voice Over IP 2nd Edition by Daniel Collins, McGraw Hill

[15] Y. Wu, B. Foo, Y. Mei, and S. Bagchi, "Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS," In Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03), pp. 234-244, December 2003

[16] Fredrik Valeur, Giovanni Vigna, Christopher Krügel, Richard A. Kemmerer: A Comprehensive Approach to Intrusion Detection Alert Correlation. IEEE Trans. Dependable Sec. Comput. 1(3): 146-169 (2004).

[17] P. Porras, M. Fong, and A. Valdes, "A Mission-Impact-Based Approach to INFOSEC Alarm Correlation," Proc. Int'l Symp. the Recent Advances in Intrusion Detection, pp. 95-114, Oct. 2002.

[18] P. Ning, Y. Cui, and D.S. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," Proc. ACM Conf. Computer and Comm. Security, pp. 245-254, Nov. 2002.

[19] D. Andersson, M. Fong, and A. Valdes, "Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis," Proc. Third Ann. IEEE Information Assurance Workshop, June 2002.

[20] Son Vuong; Yan Bai, "A survey of VoIP intrusions and intrusion detection systems," Advanced Communication Technology, 2004. The 6th International Conference on , vol.1, no.pp. 317- 322, 2004.

[21] Andon Batchvarov, "Security Issues and Solutions for Voice over IP compared to Circuit Switched Networks," INFOTECH Seminar Advanced Communication Services (ACS), 2004.