Intrusion Detection in Voice-over-IP Environments

YU-SUNG WU, VINITA APTE[1], SAURABH BAGCHI

Dependable Computing Systems Lab

School of Electrical & Computer Engineering. -Purdue University

465 Northwestern Avenue

West Lafayette, IN 47907

USA

Email: {vapte,ywsu,sbagchi}@purdue.edu
Telephone: +1 765-494-3362
Fax: +1 765-494-2706

SACHIN GARG

Yahoo Labs, India

Email: gsachin@yahoo-inc.com

NAVJOT SINGH

Avaya Labs

Email: singh@avaya.com

[1] The first two authors contributed equally to the paper. The order of these two author names is not significant.

Abstract

In this article, we present the design of an intrusion detection system for VoIP networks. The first part of our work consists of a simple single-component intrusion detection system called SCIDIVE.

In the second part, we extend the design of SCIDIVE and build a *distributed and correlation-based* intrusion detection system called SPACEDIVE. We create several attack scenarios and evaluate the accuracy and efficiency of the system in the face of these attacks.

To the best of our knowledge, this is the first comprehensive look at the problem of intrusion detection in VoIP systems. It includes treatment of the challenges faced due to the distributed nature of the system, the nature of the VoIP traffic, and the specific kinds of attacks at such systems.

Keywords*: Intrusion detection, Voice over IP system, Cross-protocol detection, Stateful detection, Correlation-based IDS, SIP, RTP.*

Introduction

Voice over IP (VoIP) systems are gaining in popularity as the technology for transmitting voice traffic over IP networks. While VoIP technology is set to revolutionize communications, and is already being used by a number of traditional telephone companies to connect their regional offices, on a smaller scale it can also be a useful solution for businesses looking to trim their telephone expenses. 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 of which follow a general pattern of attacks against an IP infrastructure. There have been enormous strides made in the field of intrusion detection systems (IDS) for different components of the information technology infrastructure. Some of the IDSs are generic in nature and can be customized with detection rules specific to the environment in which they are deployed (e.g., Snort [7] and Prelude [10]), and some are tools specifically targeted to an environment or to specific classes of intrusions, such as IBM Tivoli Intrusion Manager for MQSeries products [11]. VoIP systems pose several new challenges to IDS designers. First, these systems employ multiple protocols for call management and data delivery. Second, the systems are distributed in nature and employ distributed clients, servers, and proxies. Third, the attacks against such systems span a large class, from denial of service to billing fraud. Finally, the systems are heterogeneous and typically under several different administrative domains, e.g., the proxy server may be provided by the service provider and the client managed by the home organization.
In this article, we present our work in the design and deployment of an intrusion detection system for VoIP networks. The initial goal of our work was to build a local, single-component VoIP IDS. We call this system SCIDIVE (pronounced as *Skydive*). SCIDIVE 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, without substantial system customization, be extended for detecting new classes of attacks. The IDS can handle client mobility, an important design goal of VoIP protocols such as SIP, and does not flag false alarms for such situations. SCIDIVE can operate with both classes of protocols that compose VoIP systems – call management protocols (*CMP*), e.g., the Session Initiation Protocol (SIP) [2], and media delivery protocols (*MDP*), e.g., the Real Time Transport Protocol (RTP) [3]. SCIDIVE proposes two critical abstractions for VoIP IDS – *stateful detection* and *cross-protocol detection*. *Stateful detection* denotes the functionality of assembling state from multiple packets and using the aggregated state in the rule-matching engine. The reassembly functionality is applicable to packets of both CMP and MDP and can be configured to handle packets spread out arbitrarily far apart in time. Some existing IDSs provide support for reassembly, but they are restrictive and applicable only to specific protocols. For example, Snort's stream4 module can reassemble TCP packets that belong to the same session. *Cross protocol detection* denotes the functionality of matching rules that span multiple protocols, e.g., detecting a pattern in a SIP packet followed by one in a succeeding RTP packet followed by one in an RTCP packet. The aggregation across protocols can be chained in an arbitrarily long manner and spread out in time. This abstraction is powerful for VoIP systems because they involve multiple protocols and several attacks are based on sequences that cross protocol boundaries. There are very few systems today that support cross-protocol detection. One of the notable ones is WebSTAT [12] for detecting attacks against web servers by correlating protocols in a vertical stack, e.g., application level (web server) and operating system log. Since VoIP systems use multiple application layer protocols, horizontal cross-protocol correlation is required.

The architecture of SCIDIVE uses a *Distiller*, through which all incoming network traffic passes and which translates packets into protocol dependent information units called *Footprints*. The Footprints that belong to the same session are grouped into *Trails*. The *Event Generator* maps Footprints into *Events* which are matched by the *Rule Matching Engine* against a *Ruleset*. According to the stateful and cross-protocol philosophies, the Events can potentially have state information and encapsulate information from multiple packets. The rules considered are misuse-based, as opposed to the complementary philosophy of anomaly-based rules. The classic tradeoffs between the two approaches also hold here—misuse-based detection cannot deal with novel types of attacks, while anomaly-based rules can have large amount of false positives.

SCIDIVE is demonstrated on a sample VoIP system that comprises SIP clients and SIP proxy servers. The system uses SIP Express Router for the proxy and three different kinds of clients – KDE's KPhone [13], Microsoft Windows Messenger [14], and XTen's X-Lite IP Telephony client [9]. The protocols used are SIP for call management and RTP for real-time audio data transfer. In our experiments, an instance of SCIDIVE is associated with each client. Four different types of attacks are simulated on the system and the effectiveness and efficiency of SCIDIVE analyzed.

VoIP systems involve multiple components and are distributed in nature. These components are widely spread and often fall under different administrative domains. Since SCIDIVE uses local, single-component based detection, it cannot detect an attack that manifests itself at multiple components. Therefore, in the second part of our work, we extend the design of SCIDIVE and build a *distributed and correlation-based* intrusion detection system called SPACEDIVE. SPACEDIVE also uses the concepts of stateful and cross-protocol detection. It further proposes the abstraction of correlation based IDS and provides a rule language to express correlated rules. 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. 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. Moreover, our testbed is in the form of a hierarchy of components that respects administrative domains and is thus suitable for deployment in real-world settings. Several attack scenarios are created and the accuracy and the efficiency of the system evaluated with rules meant to catch these attacks.

In summary, the contributions 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.

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. Additionally, we present a taxonomy of attacks against VoIP systems. Some notable attacks from this taxonomy are redirect attacks, toll frauds, call interception attacks, etc. We propose the *end-to-end matching* paradigm as a powerful approach for detecting a large class of these attacks. Yet another contribution of this work is our rule language that extends Snort's rule language and customizes it for a VoIP IDS.

The taxonomy of attacks and the rule language can be the starting point for much needed research and development in VoIP IDS.

VoIP Overview

Voice over IP (VoIP) systems provide facilities for setting up and managing voice communications based on one of two main call management protocols: H.323 [1] and SIP. H.323 is the most widely deployed standard in VoIP communications, but SIP is increasing in popularity due to its simplicity and corresponding ease of implementation. With both protocols, endpoints or terminals, which may be physical phones (hardphones) or software programs executing on a general-purpose computer (softphones), send and receive RTP 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 [4] and MEGACO/H.248 [5]. Within an H.323 network, an optional gatekeeper may be present. The gatekeeper performs several functions including authorizing network access, assisting in managing quality of service, and providing address-translation services. Also, multipoint controllers may be present to manage multipoint conferences between three or more terminals or gateways.

SIP networks also include additional types of servers.

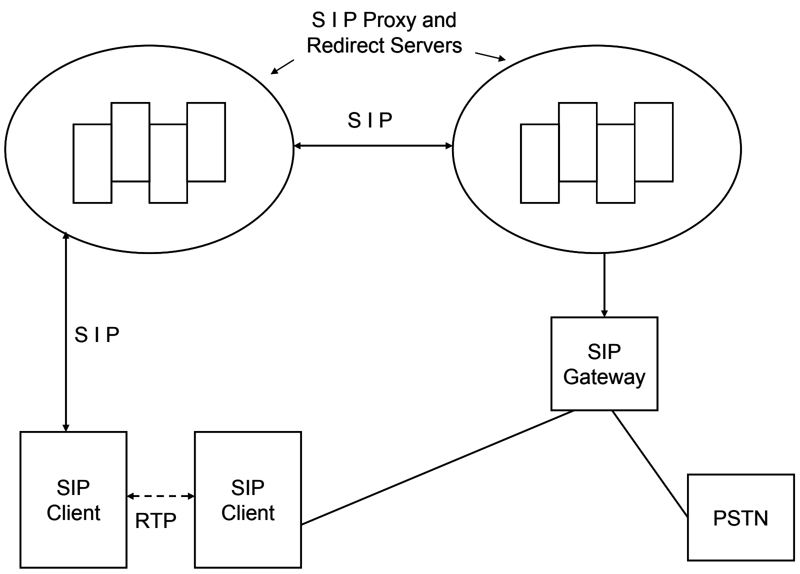

Figure 1: SIP-based VoIP Environment

A proxy server forwards requests, possibly after performing some processing or translation. A redirect server is used to support mobile clients and performs address translation for an accepted request and returns the new address to the originator of the request. Both proxy and redirect servers may be used to implement call forwarding and other similar services. User agent clients send requests to user agent servers to initiate calls. The user notifies a registrar of his current location to allow others to contact him. The registrar is often combined with a proxy or redirect server.

Suppose that a user Bob wants to call another user Alice. Bob begins by sending a SIP INVITE message to its own proxy server that contains Alice's SIP address in the destination field. Bob's proxy server then does a lookup (this could be a DNS lookup, a simple table lookup etc) to locate Alice's proxy server and then forwards the INVITE to Alice. When Alice receives the INVITE and is willing to accept the call from Bob, she sends a SIP OK message back to Bob using the same path. Once this call setup is complete, Alice and Bob can exchange voice data encoded in the form of RTP packets directly.

Both H.323 and SIP provide protocols for call setup, management, and media delivery. Voice is encoded using a negotiated codec and delivered using RTP over UDP/IP for both protocols. However, call setup and management are handled quite differently. H.323 relies on the H.225.0 [15] and H.245 protocols [16], whereas SIP uses a much simpler 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. CANCEL cancels a pending INVITE. The OPTIONS message is used to query or change optional parameters of the session, such as, encryption. Some common SIP error messages are described in Table 1.

Table 1: Sample SIP Error Messages

301 Moved permanently - Redirect
302 Moved temporarily - Redirect
403 Forbidden

404 Not Found
408 Request Timeout
480 Temporarily Unavailable
500 Server Internal Error

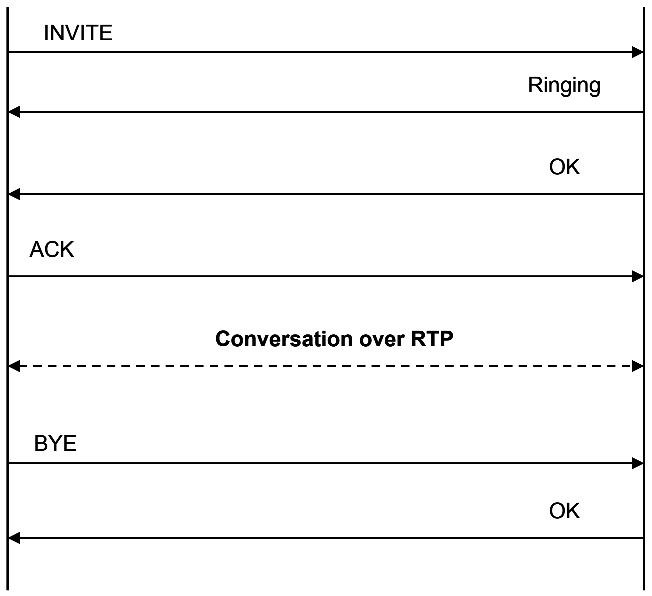

Figure 2: Sample SIP message exchange

VoIP Vulnerabilities

One of the main advantages of a VoIP system is the convergence of voice and data networks with voice being conveyed over a data network. While this offers advantages in cost and ease of management, the use of the data network in a converged system makes the voice network vulnerable to the same vulnerabilities suffered by the data network. This includes well-known attacks such as denial of service attacks as well as authentication attacks. In addition, a voice network introduces potential vulnerabilities related to toll fraud, privacy, and denial of service attacks based on degrading the quality of service of the voice conversation.
A major source of vulnerabilities lies in the protocols used to set up and manage calls. Both H.323 and SIP transmit packet headers and payload in clear text without a per-message integrity check (e.g. digitally signed), which allows an attacker to forge packets that manipulate device and call states. For example, such forged packets can prematurely terminate calls, redirect calls, or facilitate toll fraud. Some efforts are currently underway to develop encrypted signaling, but no solution has found widespread adoption. Some of the security features are built into SIP, such as, end-to-end or hop-by-hop encryption of the SIP traffic. Some other solutions rely on lower layer security primitives, such as, IPSec. However, a concern regarding use of any of the security primitives is the computation load on the end points, some of which are very resource constrained. It also relies on the deployment and use of keys, which requires the adoption of a key management infrastructure to which the different VoIP service providers must agree. This has turned out to be a challenging deployment issue so far. .
In addition to vulnerabilities present in the signaling protocols, the RTP protocol for media delivery also introduces several vulnerabilities due to the absence of authentication and encryption. Even if Secure RTP (SRTP) is used, the system

would still be vulnerable to smart DoS attacks, as the extra encryption-decryption time required for each SRTP packet only worsens the situation. Each RTP packet header contains a sequence number that allows the recipient to play back voice packets in the proper order. However, an attacker can easily inject artificial packets with higher sequence numbers that will cause the injected packets to be played in place of the real packets. Also SPIT (Spam over Internet Telephony) is predicted to become a problem as VoIP systems become more widespread.

Attacks Taxonomy

We can classify the possible attacks into two broad categories:
(i) Generic Attacks, (ii) VoIP-Specific Attacks

Generic Attacks

We classify the generic attacks into four categories: Denial of Service (DoS) attacks, Buffer Overflow, Unauthorized access, and Attack on the operating system. These kinds of attacks are well-known in traditional computing environments, but also affect VoIP environments, since they share several protocols, and hence their vulnerabilities.

VoIP-specific Attacks

Signal Protocol Attack: Such attacks exploit vulnerabilities in the signaling protocol. For example, the SIP protocol contains holes in the subset related to *invite* messages.
Redirect Attacks: A redirect attack might change a voicemail address or a call forwarding address to the address of a hacker, thereby opening a channel to abuse.
Call Interception: An unauthorized person could monitor and intercept voice packets, perhaps reading and stealing or corrupting them.
This is an example of a passive attack and is outside the purview of SPACEDIVE.
Toll Fraud: An unauthorized person could monitor and intercept call setup packets, gaining sufficient information to allow her to masquerade as a legitimate user and make fraudulent calls.

SCIDIVE Architecture

VoIP applications involve multiple protocols and each of these protocols has its own protocol states that need to be well-synchronized in order to keep the VoIP applications running correctly. As the end goal all attacks against VoIP is to cause disruptions to these applications, we found that it would be handy for our IDS to have the capability to detect the disruptions from the protocols and the corresponding states. To achieve this, the IDS's rules must possess the ability to express the combination of the protocol and the states of interest. Most existing IDS solutions, such as Snort, provide only a generic architecture in which the IDS itself does not pay much attention to the application level protocols and the corresponding state information. Consequently, the user has to construct detection rules from the ground to capture these application level contexts for performing detection. This creates problems in directly applying these IDS solutions effectively to a VoIP system.

The design of the SCIDIVE architecture and its corresponding components is meant to facilitate the process of detecting VoIP related attacks by utilizing both the protocols and the state information. We propose a new IDS methodology in SCIDIVE in which each rule could harvest the stateful information of VoIP sessions for matching malicious patterns – *Stateful Detection*. It could also harvest packets from different protocols involved in a VoIP protocol for detection – *Cross Protocol Detection*. Note that the SCIDIVE architecture is not specific to the signaling protocol or the media protocol; it just assumes that there are two distinct protocols for these two functions. Thus, either H.323 or SIP can be used for the signaling protocol. However, in our current implementation, the parser for the rules has been created for SIP and our discussion hereon will assume SIP as the signaling protocol.

SCIDIVE Components: Footprints, Trails, Events, Rules

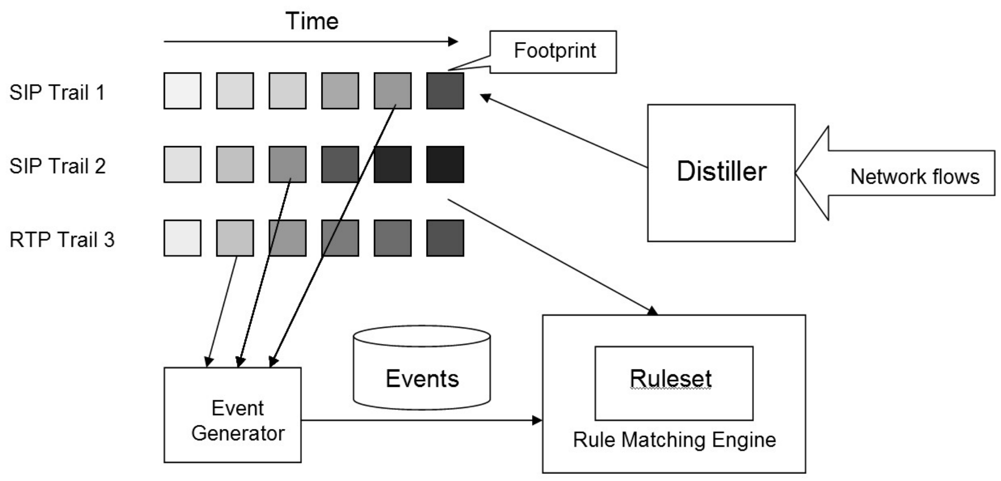

Figure 3: SCIDIVE Architecture

Figure 3 presents an overview of the SCIDIVE architecture. In SCIDIVE, incoming network flows first pass through the ***Distiller***, which translates packets into protocol dependent information units called ***Footprints***. The Distiller is responsible for doing IP fragmentation, reassembly, decoding protocols, and finally generating the corresponding Footprints. A ***Footprint*** is a protocol dependent information unit, which, for example, could be composed of a SIP message or an RTP packet. Footprints that belong to the same session are grouped into ***Trails***. In Figure 2 we have three Trails that correspond to two SIP sessions and one RTP session. The ***Event Generator*** maps footprints into a single event. For example**,** we can map two out of order RTP Footprints into an event called *RtpJitter*. Event Generator is hard-coded and seamlessly coupled with internal structures for best possible performance. In general, it is just a layer of abstraction, which correlates the information in footprints and concentrates the information into a single event. It helps performance by hiding some computationally expensive matching, e.g., by triggering the ruleset at the moment of interest instead of triggering it upon each incoming RTP Footprint.
Ruleset is triggered by a sequence of Events, e.g., we can define a rule for detecting RTP flow [event 1] after a session is torn down [event 2]. The matching in the Ruleset is based on Events that can potentially encapsulate information from multiple packets and can bear state information. Besides the information that Events provide, the Ruleset can also perform the matching based on crude

information directly from the Trails in case no suitable Event is available. For example, we might be interested in knowing who prematurely tears down the session. To achieve this, we probably need to have a look at the corresponding SIP Footprint to identify the ID and IP address of the originator. This direct access however will cause some degree of inefficiency.

Cross-protocol Methodology for Detection

We propose a powerful abstraction for intrusion detection systems in general, and VoIP IDSs in particular, namely, cross-protocol detection. An IDS that uses cross-protocol detection accesses packets from multiple protocols in a system to perform its detection. This methodology is suitable to systems that use multiple protocols and where attacks spanning these multiple protocols are possible. There is the important design consideration that such access to information across protocols must be made efficiently.

A VoIP system incorporates multiple protocols. A typical example is the use of SIP to establish a connection, followed by use of RTP to transfer voice data. Also, RTCP and ICMP are used to monitor the health of the connection. VoIP systems typically have application level software for billing purposes and therefore may have accounting software and a database.

To motivate the need for cross-protocol detection, we introduce a synthetic example of a billing fraud attack. Since VoIP systems have been gaining in popularity only of late, there are very few instances of actual attacks in databases such as CERT [15] and Bugtraq [16]. In our synthetic scenario, the attack is launched by the attacker exploiting a vulnerability in the SIP proxy. She sends a carefully crafted SIP message to fool the proxy into believing the call is initiated by someone else. The proxy initiates the accounting software with the information about the incorrect source for the call. This allows the attacker to make calls without being charged. Using the cross-protocol methodology for detection, one can create a cross-protocol rule to look at the SIP messages, the transaction messages between the accounting software and the database, and the RTP flows later on. Specifically, each of the following three conditions must hold.

1. The SIP message should follow the correct format.
2. When the accounting software sends out a transaction to denote a call from user A to user B, check if user A has sent a SIP Call Initialization message to user B. If user A has not set up the call with a legitimate SIP Call Initialization message, then this condition will be violated.
3. Check the source/destination IP addresses of the subsequent RTP flows. Together with information from DNS and SIP Location Servers, we can reconfirm that each RTP flow has a corresponding legitimate call setup.

In SCIDIVE, cross-protocol detection is achieved through keeping multiple trails for different sessions. In our example, we can have (i) a 'SIP trail' which tracks all the SIP messages in the session between user A and user B; (ii) an 'RTP trail' which tracks all the RTP packets in the session between A and B; and (iii) an 'Accounting trail' which tracks relevant accounting transactions in this session between A and B. Then, we can define three events based on the three trails corresponding to the three conditions above. The first event is "an incorrectly formatted SIP message in the SIP trail", which could be an indication of an attempt to exploit the vulnerability in the SIP proxy. The second event is "a transaction in the Accounting trail that has no matching call initialization message in the SIP trail". The third event is "either the source or destination IP addresses of the RTP packet without a matching address in the SIP packet". The third event is specialized to take mobility into account, which will be indicated by a SIP REINVITE message with an update of state at the SIP Registrar that maintains location information. In the Ruleset, we can put a rule called *Billing Fraud,* which is triggered by a combination of these three events. An advantage of creating a rule based on a sequence of three events is improving the accuracy of the alarm because the rule is based on three facets of the attack. It is perceivable that relying solely on Event 1 or Event 3 to signal 'Billing Fraud' alarm will result in false alarms. Also, bugs or temporary system failures might cause Event 2. Therefore, relying solely on Event 2 will possibly give us false alarms.

Stateful Methodology for Detection

A second abstraction useful for VoIP systems in particular is stateful detection. Stateful detection implies building up *relevant* state *within* a session and *across* sessions and using the state in matching for possible attacks. It is important that the state aggregation be done efficiently so that the technique is applicable in high throughput systems, such as VoIP systems.

A VoIP system maintains considerable amount of system state. The client side maintains state about all the active connections – when the connection was initiated, when it can be torn down, and what the properties of the connection are. The server side also maintains state relevant to billing, such as the duration of the call. To motivate the need for stateful detection, we introduce a synthetic example of a DoS attack and a password guessing attack. An unauthorized user client keeps sending unauthenticated REGISTER requests to bombard the SIP proxy and ignores the 401 UNAUTHORIZED reply error message from the SIP proxy. If the user client keeps sending the same request to the server, it can be seen as a type of DoS attack on the SIP proxy. Along with the UNAUTHORIZED reply message, the proxy sends a challenge phrase to the client. If the client keeps sending requests with different values in the challenge response field, this could be seen as a type of attack that is trying to break the authentication key by brute force. In either case, it would be helpful for detection if the system can look at the series of user client requests and the subsequent server responses. Since 4XX responses are not uncommon in a normal session, a traditional IDS like Snort with a rule to detect multiple 4XX responses may flag a large number of false alarms. For example, most user clients send an unauthenticated REGISTER request to the server, presuming that the SIP Proxy does not require authentication. Later, the server sends a 401 response along with a challenge phrase to the client to indicate that authentication is required. The client should then send a new REGISTER request to the server along with the correct response phrase to continue the registration process. If the IDS does not isolate 4XX error messages from different sessions and doesn't correlate 4XX error messages with requests, it is likely it will make false verdicts based on unrelated 4XX error messages. In SCIDIVE, Footprints that belong to a session are structured and kept in a single trail. Therefore, the history of all the state transitions of each session can be easily tracked. To handle the two attack scenarios above, we can set up the following two events – (i) an event "DoS via repeated SIP requests", which represents continuous, alternating SIP requests and 4XX error messages in a particular session; (ii) an event "Password guessing" which represents continuous, alternating SIP requests with different challenge responses and 401 Unauthorized reply error messages in a particular session. Flagging of the two events indicates two different kinds of attacks that may have different responses. In the first case, the response may be to identify the source of the attacker and block her IP address while in the second case, it may be important to take more stringent measures to ensure the security of the system, such as changing the authentication password to a longer one.

Placement of SCIDIVE Components

The SCIDIVE architecture has a great deal of flexibility in terms of the placement of its components. For example, it is possible to deploy the SCIDIVE IDS only on the SIP client side for detecting anomalies in the VoIP traffic in and out of the client. A more aggressive approach would be to deploy the SCIDIVE IDS on all the components – Clients, SIP Proxy, and Registrar server – in a VoIP system. For example, in the Billing attack scenario outlined earlier in the section "Cross-protocol Methodology for Detection", we need to deploy the IDS on the SIP Proxy and the Accounting Server to detect Event 1 and Event 2. Also we need to deploy the IDS close to both clients to monitor RTP flows to detect Event 3. It seems a logical path to use multiple SCIDIVE IDS and an alert correlation engine to give better detection accuracy. The actual gains, impacts on the system performance, and costs for doing so are explored in the next part in the system called SPACEDIVE.

Prototype and Experiments

An IDS prototype is built to instantiate the SCIDIVE architecture for VoIP systems. We implement four attacks against the VoIP system, instantiate rules in SCIDIVE for detecting the attacks, and perform analysis of the detection efficiency. For simplicity, the IDS is placed at each client for the experiments here. This configuration is shown in Figure 4 and is referred to as an *End-point based IDS architecture*.

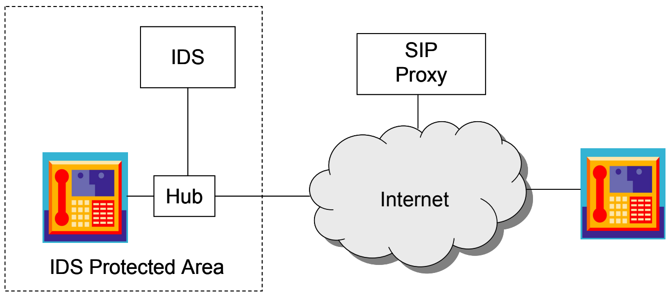

Figure 4: IDS Engine sits on or close to the end-point

Testbed

Our testbed comprises the following:
• Proxy : SIP Express Router from www.iptel.org
• Clients : Kphone from www.wirlab.net; Windows Messenger from Microsoft; X-Lite from www.xten.com
Figure 5 depicts the topology of our testbed. The IDS is connected to a hub through which the traffic of Client A can be seen. Although the prototype IDS can also see the traffic of Client B and the SIP Proxy, it does not look into those traffic for any purpose, thus mimicking an end-point based IDS.

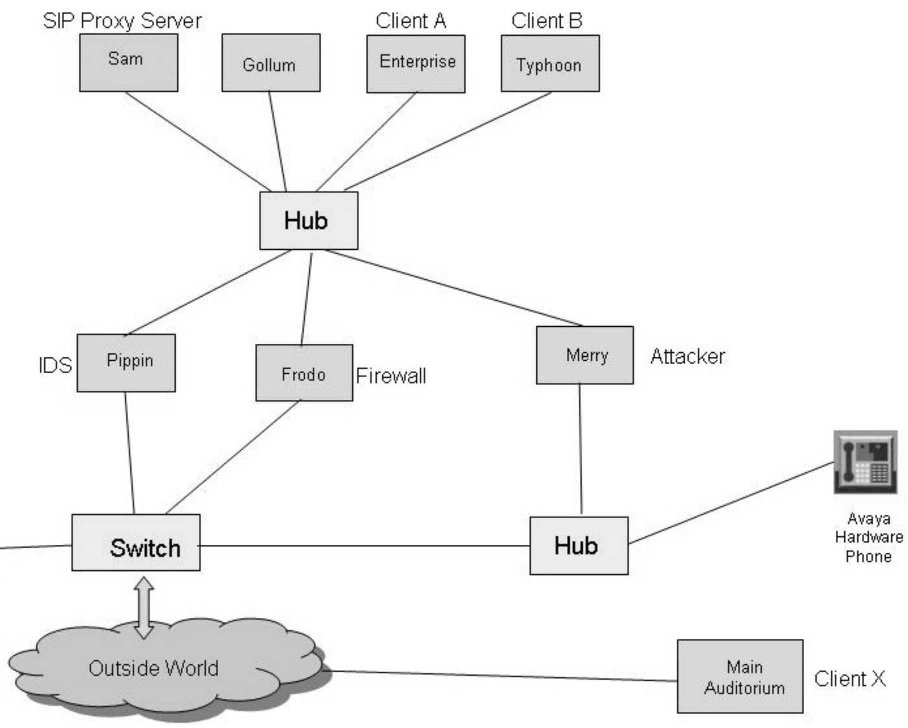

Figure 5: Testbed with client phones, proxy servers, network elements, attacker host and the IDS

Attacks and Demonstration of the IDS

We implement four attacks to demonstrate the functionality of the IDS. Three of the four attacks are based on the vulnerabilities in the signaling protocol, i.e., SIP. They are BYE attack, Fake Instant Messaging, and Call Hijacking. The fourth is based on the vulnerabilities in the media transport protocol, which is the RTP attack. A summary of the attacks is presented in Table 2. Details of the attacks are presented in the following sections.

Table 2: Summary of Attacks

Name of Attack	**Protocols Involved**	**Cross-protocol or not?; If yes, how?**	**Stateful or not?; If yes, how?**	**Rule Snippet**
BYE Attack	SIP, RTP	Yes; Detect no RTP traffic once SIP BYE has been seen.	Yes; Monitor session to determine when it has been torn down.	No RTP traffic should be seen after a SIP BYE from a particular user agent.
Fake Instant Messaging	SIP, IP	Yes; Check the source IP addresses of incoming IM messages (SIP Message)	No;	Check the IP addresses of all the incoming messages. The address should stay the same within a time period.
Call	SIP, RTP	Yes; Detect no	Yes;	No RTP traffic

Hijacking		RTP traffic once SIP REINVITE has been seen.	Monitor session to determine when it has been redirected.	should be seen after a SIP BYE from a particular user agent.
RTP Attack	RTP, IP	Yes; Check the source IP address of RTP packets.	Yes. Check if the sequence numbers in consecutive packets increase as expected.	Check if RTP packets come from legitimate IP address and if the sequence number increases appropriately.

BYE Attack

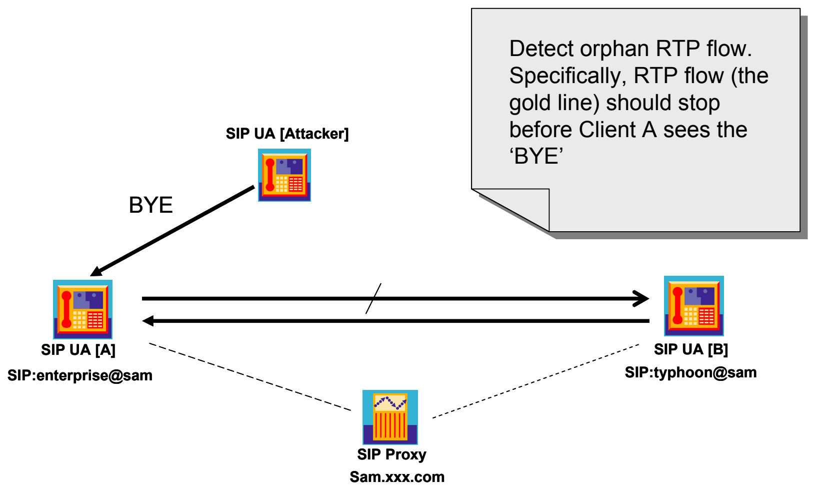
Detect orphan RTP flow. Specifically, RTP flow (the gold line) should stop before Client A sees the 'BYE'
SIP UA [Attacker]
BYE
SIP UA [A]
SIP:enterprise@sam
SIP UA [B]
SIP:typhoon@sam
SIP Proxy
Sam.xxx.com

Figure 6: Schematic of BYE Attack

In this attack scenario, we have three SIP User Agents {A, B, and Attacker}. We also have a SIP Proxy for setting up the connections. The goal of the BYE attack is to prematurely tear down an existing dialog session, which can be viewed as a kind of Denial-of-Service attack. In Figure 6, SIP UA A and SIP UA B have an ongoing dialog session. Later, Attacker sends a faked BYE message to A. After that, A will believe that it is B who wants to tear down the connection by sending the BYE message. A will stop its outward RTP flow immediately, while B will continue to send RTP packets to A, since B has no notion that the connection should be terminated.
In order to detect this attack, we create a rule that detects orphan RTP flow. Specifically, if it is indeed B who wants to stop the connection, then A should not see the RTP flow from B after getting the BYE message. Thus, in the IDS, we create a rule which signals an alarm when seeing new RTP Footprints in the RTP Trail that corresponds to the flow from B after seeing a BYE SIP Footprint. Although the attack itself occurs only within the signaling protocol (SIP), our detection rule spans SIP and RTP and provides an illustration of the importance of cross-protocol detection.

Fake Instant Messaging

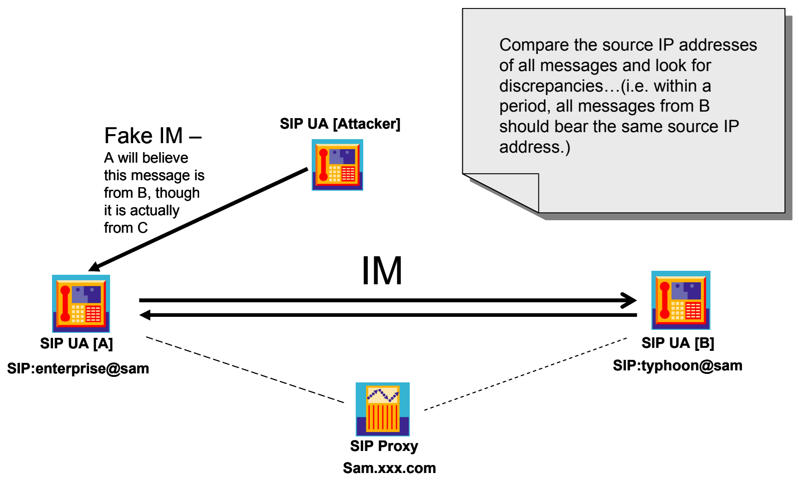
Compare the source IP addresses of all messages and look for discrepancies…(i.e. within a period, all messages from B should bear the same source IP address.)
Fake IM – A will believe this message is from B, though it is actually from C
SIP UA [Attacker]
IM
SIP UA [A]
SIP:enterprise@sam
SIP UA [B]
SIP:typhoon@sam
SIP Proxy
Sam.xxx.com

Figure 7: Schematic of a Fake Instant Messaging Attack

In addition to IP Phone Call setup, SIP also supports Instant Messaging. By faking the header of an instant message appropriately, the attacker can forge a message to A and mislead it into believing the message is from B. Our rule for detecting this attack looks at the IP addresses of the messages. Under most circumstances, within a period, messages from B should bear the same source IP address. Therefore, once we find a message from B which carries a different IP address, it would be an indication that this message is a fake one. The rule takes rate of user mobility into account and allows for changes in the IP address according to the maximum rate of user motion. Indeed, this rule is not perfect. If the attacker is able to spoof its IP address, then this rule will not work. However, based on the Host-based architecture, this is probably the best we can do. This leads to interests in research on a more ambitious architecture like deploying IDS on both client ends.

Call Hijacking

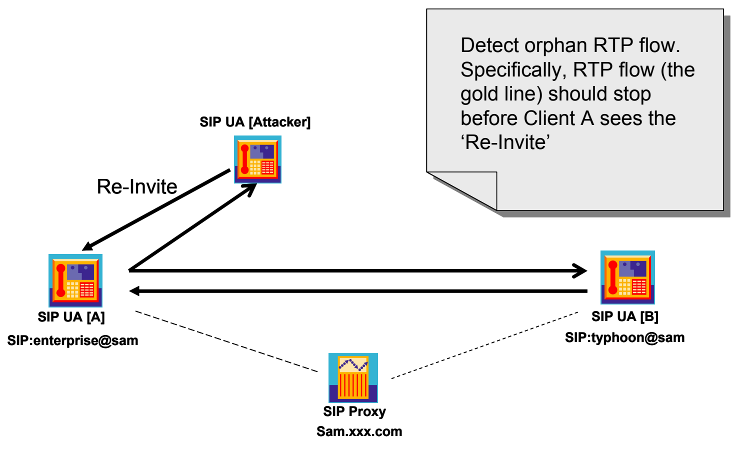
Detect orphan RTP flow. Specifically, RTP flow (the gold line) should stop before Client A sees the 'Re-Invite'
SIP UA [Attacker]
Re-Invite
SIP UA [A]
SIP:enterprise@sam
SIP UA [B]
SIP:typhoon@sam
SIP Proxy
Sam.xxx.com

Figure 8: Schematic of Call Hijacking Attack

A direct impact of this attack is that B will experience a continued silence since A is no longer sending its voice data to B. For this part, the Call Hijacking attack can be seen as a kind of Denial-of-Service attack. A more subtle impact is that if the attacker were able to emulate the voice data of B, then after successfully redirecting the call, A would not be able to distinguish between B and the attacker. This could lead to issues of confidentiality and breach of privacy since the attacker will be able to listen to what A is saying.
To detect this attack, we use a similar approach as for the BYE attack. Intuitively, if the REINVITE message is indeed from B, then A should not see any RTP flow from B after that. By detecting orphan flows, we are able to determine whether it's a legitimate REINVITE message or a malicious one.

RTP Attack

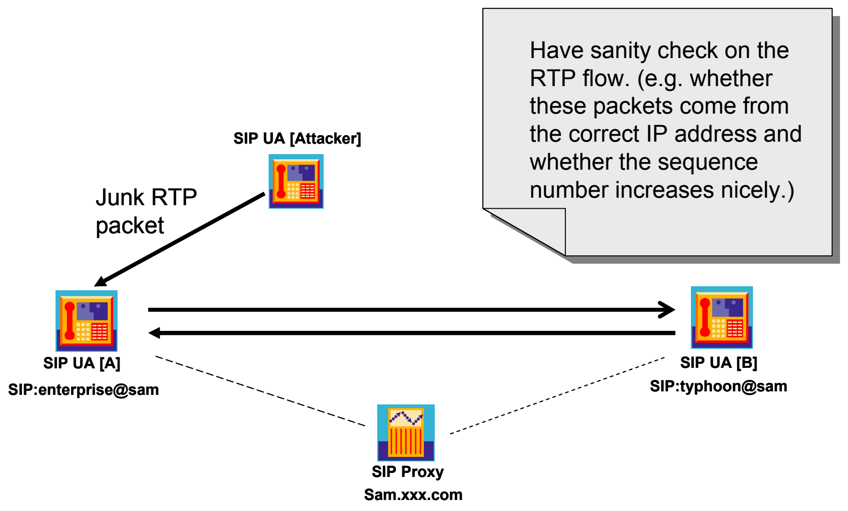
Have sanity check on the RTP flow. (e.g. whether these packets come from the correct IP address and whether the sequence number increases nicely.)
SIP UA [Attacker]
Junk RTP packet
SIP UA [A]
SIP:enterprise@sam
SIP UA [B]
SIP:typhoon@sam
SIP Proxy
Sam.xxx.com

Figure 9: Schematic of an RTP Attack

All the attacks mentioned earlier are signaling based attacks. The RTP attack on the contrary is based on the vulnerabilities in media transport. In this attack, the attacker sends RTP packets whose contents are garbage (both the header and the payload are filled with random bytes) to one of the persons in a dialog. In this example, the attacker sends garbage bytes to A. Since these garbage packets will corrupt the jitter buffer in the IP Phone client, depending on different implementations, this attack could result in intermittent voice conversation or in crashing the client. For example, in our experiment, X-Lite will crash after this attack while the effect on Microsoft Messenger is intermittent voice conversation. This attack also leads to degradation in QoS (jitter) and in the voice quality. The rule we use for detecting this attack is that the sequence number in succeeding RTP packets should increase regularly. Specifically, if we see two consecutive packets whose sequence numbers have a difference greater than 50, the IDS will signal an alarm. The gap between sequence numbers can also be seen in a DoS attack free environment, which can be caused by abrupt network traffic congestion or high CPU utilization on a phone. For example, we have seen cases where the sequence number jumped by hundreds due to crashed phone clients. However, these are not so common as seen in our testbed, and the number 50 is empirically chosen such that the false positive rate is kept low enough (e.g. < 4%

in our experiment). A larger difference value will further decrease the false positive rate but can decrease the sensitivity in detecting DoS attacks as well.

Performance Evaluation

In this section, we comment on the performance of the IDS system with respect to three metrics: (1) the detection delay, *D*, which is defined as the time from when an attack/intrusion is made to the time it is detected, (2) the probability of false alarm, P_f, which is defined as the probability that the IDS flags an intrusion when none has occurred, and (3) the probability of missed alarm, P_m, which is defined as the probability that the IDS system does not flag an intrusion when one occurs. The goal is to make the reader aware of the variables that affect these metrics and give an idea of practical values. Detailed performance evaluation with numerical computation is the subject of ongoing work.

BYE and Call Hijacking attack

In both these attacks, the IDS rule looks at the SIP signaling event (BYE or REINVITE) and monitors the media stream following this event to detect an intrusion. Specifically, the arrival of an RTP packet at the original RTP port from the original sender flags an intrusion.

Detection Delay

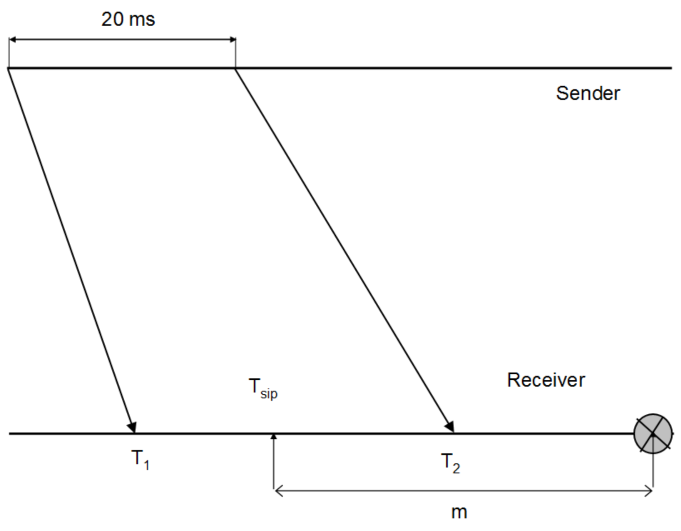

Figure 10: Detection Delay

Measuring from the time the SIP message is received, it includes the time up to the arrival of the RTP packet from the original sender. Obviously, the time depends on the frequency of RTP packets. A typical period employed is 20 milliseconds. However, the RTP packet arrival depends also on the network conditions. Specifically, the delay distribution of packets from the sender to the receiver would cause this quantity to change. Figure 10 outlines the various timing variables involved. Let the time of the last RTP packet arrival be T_1 (before the fake BYE/REINVITE message arrival). Also, without loss of generality, let this message originate from the sender at time 0. Then T_1 is the transport delay of this RTP packet. Further, let T_{sip} be the time of arrival of the SIP message. Let T_2

be the time of arrival of the RTP packet, which originated 20 ms after the previous RTP packet. We assume that the fake SIP message was generated after the 1st RTP packet but before the 2nd RTP packet. When the SIP message is received, the IDS system starts looking for RTP packets for a total duration of “m” milliseconds. Obviously, T_1, T_2 and T_{sip} are random variables and m is a fixed value. The detection delay D is a function of these four parameters as $D = T_2 - T_{sip}$, where $T_2 = 20 + N_{rtp}$ and $T_{sip} = G_{sip} + N_{sip}$. N_{rtp} and N_{sip} are the random variables associated with the network delay for each packet. While the second RTP packet is generated 20 milliseconds after the origin, the fake SIP message is generated between the two RTP packets with a distribution given by G_{sip}. Therefore, $D = 20 + N_{rtp} - (G_{sip} - N_{sip})$. Given the distributions of these random variables, it is possible to compute the detection delay distribution. Under the simplest of assumptions, where the fake SIP message is generated with a uniform distribution in (0,20), and the network delay is assumed to be independent and identical for all packets, the expected detection delay is 10 milliseconds, which is half of the RTP packet generation period.

Probability of Missed Alarm

Since the detection depends on monitoring after a SIP message arrival and since this monitoring interval is necessary finite (‘m’ milliseconds), there is a probability that the IDS system may not detect the intrusion. For instance, if the subsequent RTP packet(s) from the original sender are lost (somewhere in the network) and no RTP packet arrives within the monitoring interval, then no alarm will be raised. Referring to Figure 10, the probability is given by :
$P_f = \Pr\{T_2 > T_{sip} + m\} = \Pr\{N_{rtp} - G_{sip} - N_{sip} > m - 20\}$.

Probability of False Alarm

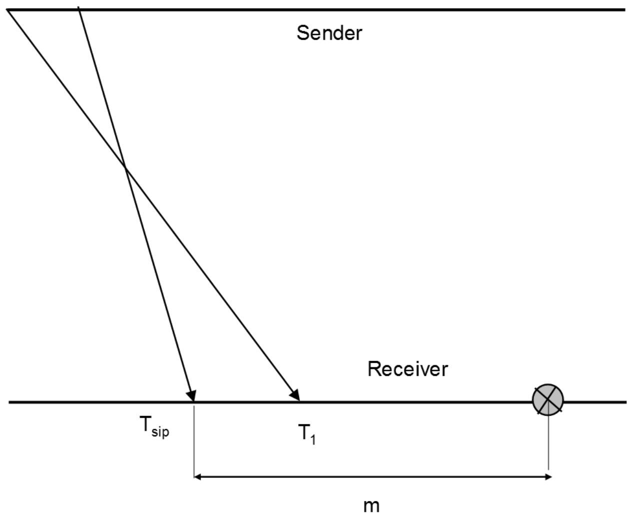

Figure 11: Probability of False Alarm

Although rare, it is possible for a valid BYE message to arrive before the RTP packet if, for instance, they take a different route, as shown in Figure 11. In this case, the IDS system will raise a false alarm. In order to compute this probability, we assume that the sender generated the valid SIP BYE/REINVITE immediately (with zero delay) after it has sent out the last RTP packet. In that case, the false alarm probability is given as $P_f = \Pr\{N_{sip} < N_{rtp}\}$. If the density function and distribution function of N_{rtp} and N_{sip} is assumed to be identical and independent denoted by $f_N(t)$ and $F_N(t)$ respectively, then the false alarm probability is $\int_0^m F_N(t) f_N(t)\, dt$.

SPACEDIVE

As an extension of SCIDIVE, we propose the design of a system called SPACEDIVE to serve as a 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 design principles.

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.

Cross protocol and stateful detection. These principles were present in our earlier version of the system, SCIDIVE and are maintained in the distributed architecture as well. To recap, 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.

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 Design

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.

SPACEDIVE Local Level Design

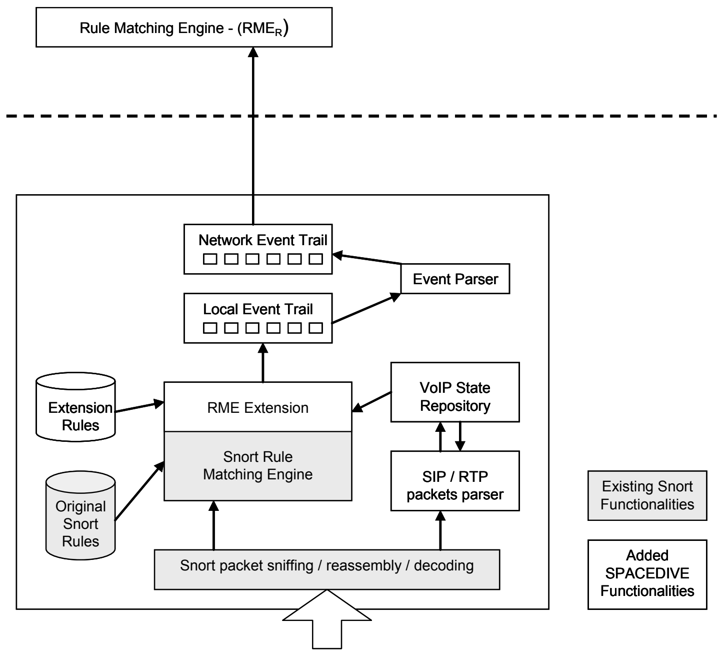
Rule Matching Engine - (RME_R)
Network Event Trail
Event Parser
Local Event Trail
Extension Rules
RME Extension
VoIP State Repository
Snort Rule Matching Engine
Original Snort Rules
SIP / RTP packets parser
Snort packet sniffing / reassembly / decoding
Existing Snort Functionalities
Added SPACEDIVE Functionalities

Figure 12: Local Level Design

Figure 12 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 detailed later in the section titled "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.

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 effect 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 VoIP stateful or cross-protocol detection. Snort provides limited capability for remembering state both within a VoIP session for a given protocol (e.g. SIP) and across protocols (e.g. SIP & RTP). 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.

1. **sip**. This construct is used to identify a SIP call session, which includes all the SIP messages used in a phone call between two parties.
2. **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.
3. **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.
    I. **And/Or/Not – Logical Constructs**. These constructs are used to trigger an event based on logical combinations of rule matches;

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

4. **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.
5. **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);
```

SPACEDIVE Network Level Design

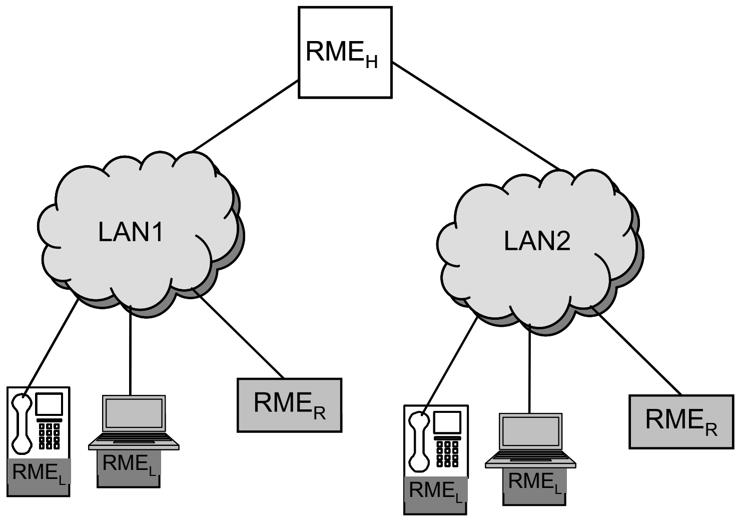

Figure 13: Rule Matching Engine Hierarchy

At the network level, SPACEDIVE views the system as composed of multiple VoIP domains, each with its own RME_R (Figure 13).

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:

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

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 13 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 may own the RME$_H$ which matches for rules that affect multiple organizations.

Algorithms in SPACEDIVE

Local Level Event Generation – Efficient Matching

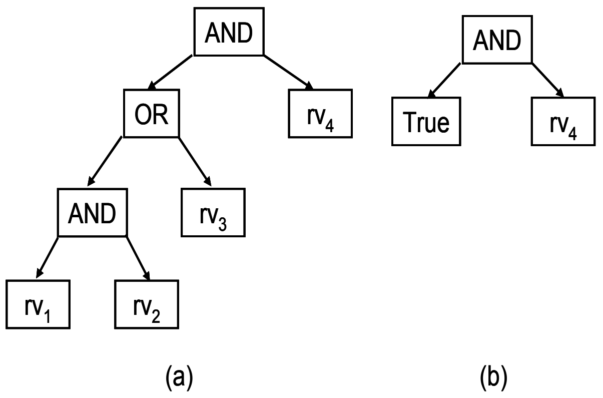

Figure 14: Example rule processing at RME$_L$

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 shown in Figure 14.

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 14.
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
$O(\sum_{i=1}^{H}(\log V + i)) = O(H.\log V + H^2) = O(v.\log V + v^2) = O[v.\log v + v.\log r + v^2]$.
If $r < v$, then the operation is $O(v^2)$.

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. Let us assume 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 $O(UR \times L)$.

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 $O(\sum_{i=1}^{h}(c+i)) = O(h^2)$.

Demonstration and Experiments

Experimental Testbed

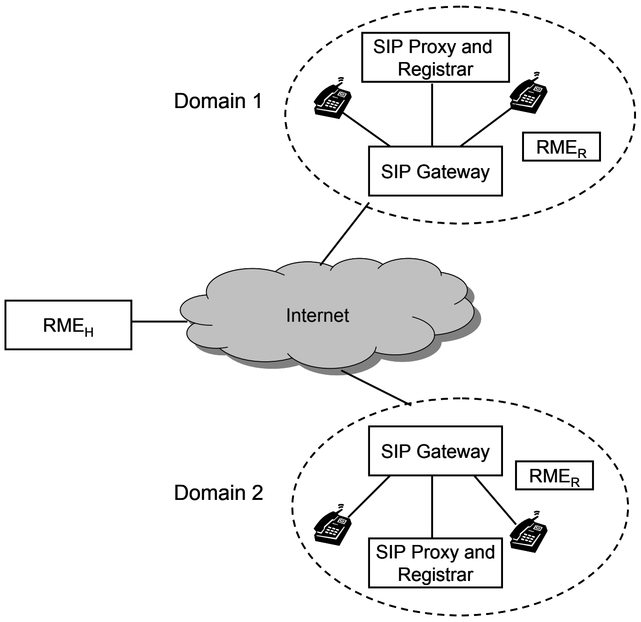

Figure 15: SPACEDIVE Testbed

Figure 15 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. The SIP clients and servers are equipped with the SPACEDIVE IDS. We use the SIP Express Router (ser) [8] 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 [9]. 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.

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.

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.

Call Hijacking

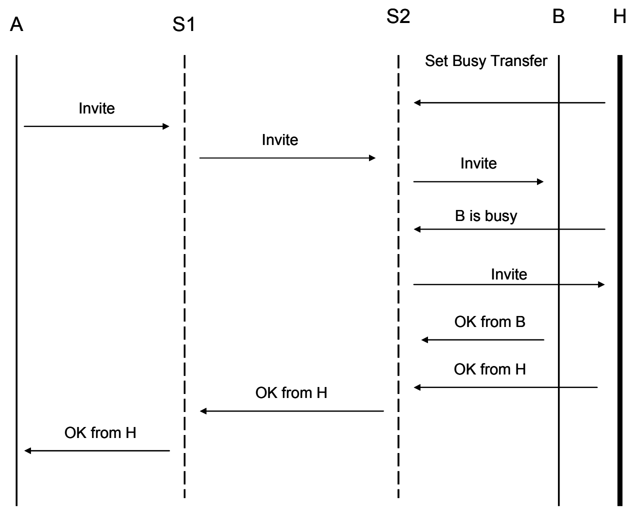

Figure 16: Call Hijacking

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.
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.(Rules in Table 3)

Table 3: Rules to Detect the Call Hijacking Attack

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

Man in the middle attack: intercepting outgoing calls

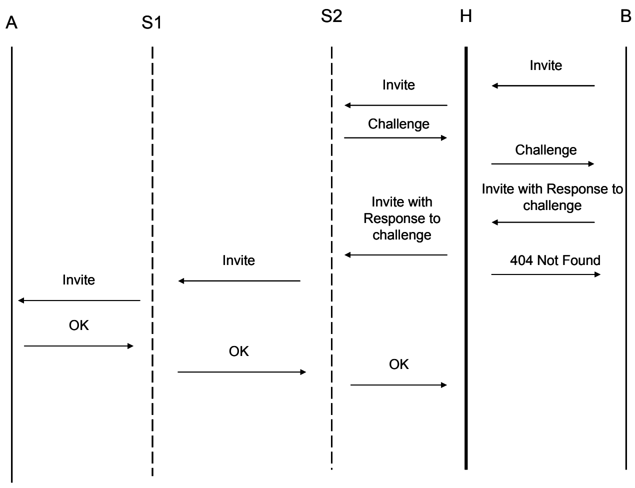

Figure 17: Man in the middle attack

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.
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. (Rules in Table 4)

Table 4: Rules to Detect the Man in the Middle Attack

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

BYE Attack

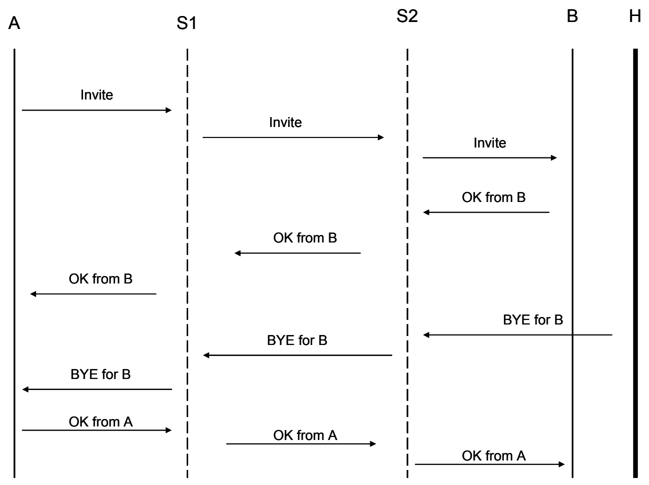

Figure 18: BYE Attack

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 vulnerability in the system.
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. (Rules in Table 5)

Table 5: Rules to Detect the BYE Attack

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`

Compromised SIP Proxy

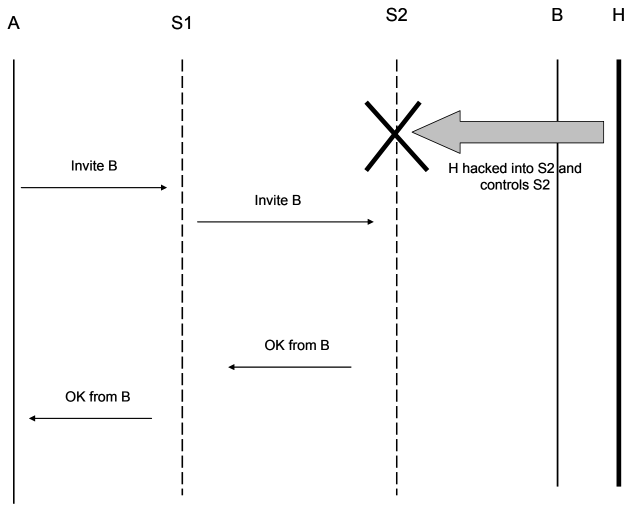

Figure 19: Compromised SIP Proxy

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.

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. (Rules in Table 6)

Table 6: Rules to Detect a Compromised SIP Proxy

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`

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 as shown in Table 7.

Table 7: Rule to Detect a DoS Attack

Rule Snippet
`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. Thus, if one packet has sequence number of 400 and the next one has anything greater than 450 or less than 350, SCIDIVE will flag an alarm. The detection in SCIDIVE follows the same principle though the rule has to be hard coded in the system.

Table 8: Classification of attack scenarios

Attack Scenario	**Mitigated by secure VoIP protocol; Dependent on**	**Detected by SCIDIVE;**

	position of malicious client	**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

In Table 8, 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.
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.

Experimental Results

Timeline for correlated detection

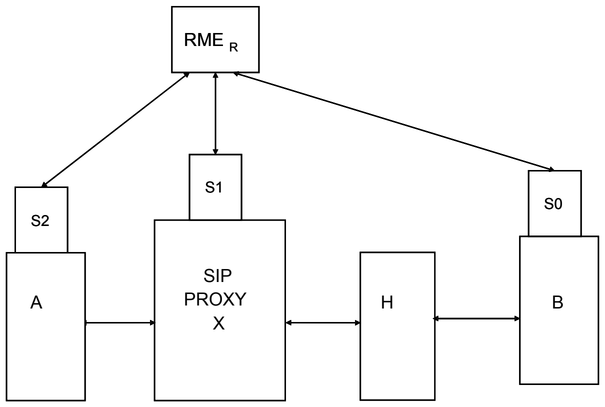

Figure 20: Experimental Setup

For this experiment, we use the Man in the Middle Attack to demonstrate the timeline of the correlated detections among SPACEDIVE components. The layout of the testbed is shown in Figure 20. 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 21.

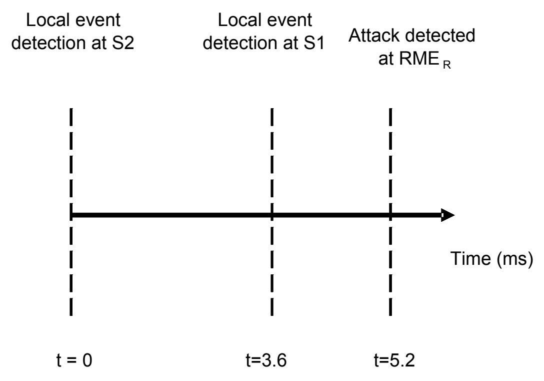

Figure 21: Timeline for Remote Rule Matching (times are not drawn to scale)

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 22 shows the comparative performance of the four categories of rules.

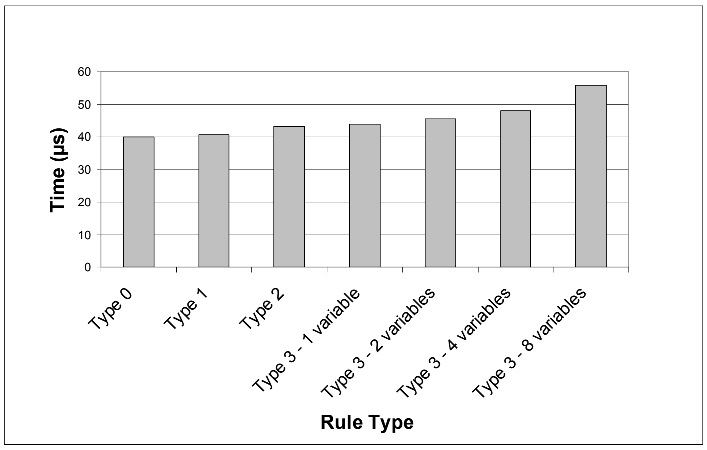

Figure 22: Processing time for classes of SPACEDIVE rules.

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.

Resilience to DoS

DoS attacks are crucial ones in VoIP systems since secure VoIP protocols such as SRTP [6] 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.

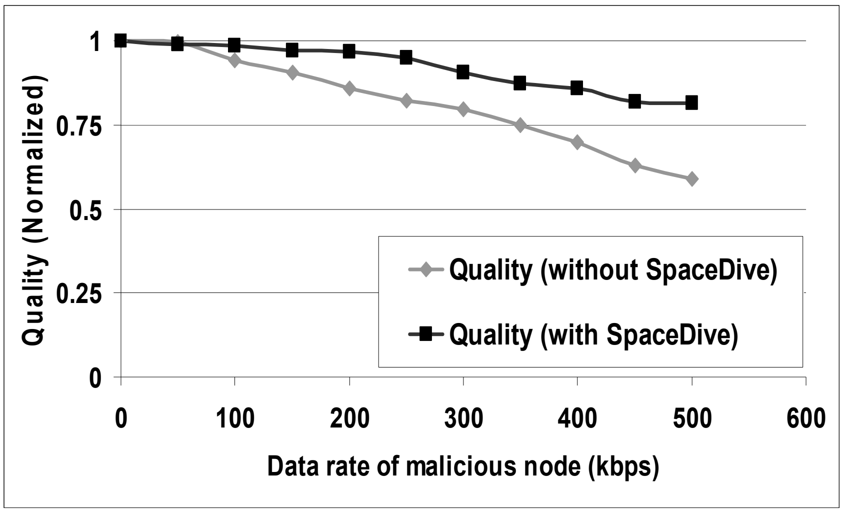

Figure 23: Resilience to DoS Attacks

Figure 23 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

Detection False Positive Rates

Table 9: False Positive Rates from each SPACEDIVE detection rule with respect to four different legitimate call traces.

	v4			v5			v6			v7		
	f.p.	a.t.	r.i.	**f.p.**	a.t.	r.i.	**f.p.**	a.t.	r.i.	**f.p.**	a.t.	r.i.
Call Hijacking	**0.00%**	0	450	**0.00%**	0	450	**0.00%**	0	461	**0.00%**	0	452
Man in the middle	**0.00%**	0	450	**0.00%**	0	450	**0.00%**	0	461	**0.00%**	0	452
BYE	**0.23%**	1	426	**0.24%**	1	423	**0.00%**	0	442	**0.23%**	1	437
Compromised SIP Proxy	**0.00%**	0	526	**0.00%**	0	524	**0.00%**	0	517	**0.00%**	0	515
DoS in RTP Streams	**3.18%**	31	976	**3.90%**	38	974	**1.94%**	19	978	**2.79%**	27	967

a.t. : Number of alerts triggered
r.i. : Number of rule instantiations
f.p. : False Positive rate =(a.t.)/(r.i)

Here we test the five detection rules in SPACEDIVE against 4 call traces collected over different time periods from a closed attack-free environment consisting of two SIP proxy servers and 192 phone clients. These call traces are numbered *v4-v7*. In terms of the average call duration, the respective values for the call traces are {*v4:2.79, v5:4.53, v6:4.02, v7:5*} (unit: minutes). In terms of the average inter-arrival time between consecutive phone calls, the values are {*v4:3.6, v5:3.71, v6:4.79, v7:5.48*} (unit: minutes). The call traces include the network packets collected by tcpdump [17] from the system. The detection result is presented in Table 9. The false alarm rate is defined as the number of alarms triggered over the number of rule instantiations.
Here a detection rule is instantiated when any of the rule components in the corresponding detection rules (Table 3, Table 4, Table 5, Table 6) is generated by a RME_L. For the DoS Attack Detection rule (Table 7), the rule is instantiated every time a RTP packet is sniffed and inspected by SPACEDIVE. The number of rule instantiations from “Compromised SIP Proxy” is the number of calls placed, as the rule is instantiated by the very first INVITE message from a phone call. The number of rule instantiations from “Call Hijacking” and “Man in the middle” is the number of connected calls, as the rule is instantiated by the OK message or the call established state. Since a callee can be busy or unavailable, the number of rule instantiations from “Call Hijacking” and “Man in the middle” is always less than or equal to the number from “Compromised SIP Proxy”. The rule for detecting “BYE Attack” is instantiated by the BYE message and can be seen as the number of connected phone calls that were torn down gracefully. For the “DoS in RTP Streams” rule, the number of rule instantiations corresponds to the number of RTP streams found in the call trace. Since RTP streams are bidirectional, we found roughly 1000 RTP streams out of the call traces each of which consists of approximately 500 calls.
Overall, we found that the false positive rates from SPACEDIVE rules used in detecting the three SIP based attacks are very low (< 0.5%). The false positive rates from the “DoS in RTP Streams” are slightly higher but still low (< 4%). The higher false positive rates from the DoS detection rule in Table 7 are understandable since RTP streams carry huge traffic and jitters can occur normally even when DoS attacks are not present. The false positive rate can be decreased by assigning a higher RTP sequence number gap threshold to flag an alarm. However, a tradeoff is that the number of false positives can be undesirably brought down as well.

Table 10: Number of false alarms from default Snort rules for the four different legitimate call traces

	v4	v5	v6	v7
[1:402:7] ICMP Destination Unreachable Port Unreachable	3823	5535	8413	5535
[1:1417:9] SNMP request udp	51	61	37	61
[1:1419:9] SNMP trap udp	32	54	73	54
[1:1384:8] MISC UPnP malformed advertisement	1222	1768	2691	1768
[1:527:8] BAD-TRAFFIC same SRC/DST	124	224	242	224
[1:553:7] POLICY FTP anonymous login attempt	0	1	1	1

Table 10 shows the number of alerts with respect to the four legitimate VoIP call traces from the default Snort rules (v 2.2.0 Build 30). These alerts are all false positives. Here [x:y:z] is the corresponding Snort rule ID and revision number. Compared to the number of alerts seen from SPACEDIVE rules (Table 9), the false positives from Snort rules are significantly higher. Thus, unmodified Snort is not suitable for VoIP attack detection since the high alert rates can overwhelm a system administrator.

Conclusion and Future Work

We have presented the design and implementation of a stateful and cross-protocol IDS for VoIP systems called SCIDIVE. We further extend the design principles of SCIDIVE to build a distributed and correlation based IDS for VoIP systems - SPACEDIVE. SPACEDIVE places local rule matching engines (RME_L) at individual VoIP components and a remote rule matching engine (RME_R) for each domain to correlate events across components. The RME_L 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. Another important area of our ongoing research is that of peer-to-peer VoIP systems. Peer-to-peer VoIP systems have a great potential for wide-spread deployment – an example is the popular Skype system. We are also looking at ways to detect Spam in VoIP systems. Finally, we are investigating the effectiveness of our system in presence of secure protocols – for example, secure RTP (SRTP).

References

[1] ITU-T, “Packet-based multimedia communications systems,” Recommendation H.323, February 1998.

[2] M. Handley et. al., “SIP: Session Initiation Protocol,” RFC 2543, March 1999.

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

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

[5] F. Cuervo *et. al.*, “Megaco Protocol Version 1.0,” RFC 3015, November 2000.

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

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

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

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

[10] “Prelude Hybrid IDS,” Available at: http://www.prelude-ids.org

[11] IBM Software, “IBM Tivoli Intrusion Manager,” Available at: http://www.ibm.com/software/tivoli/products/intrusionmgr/

[12] Giovanni Vigna, William Robertson, Vishal Kher, Richard A. Kemmerer, “A Stateful Intrusion Detection System for World-Wide Web Servers,” Proceedings of the 19th Annual Computer Security Applications Conference, December 8-12, 2003, Las Vegas, Nevada.

[13] Debian GNU/Linux, “KDE K-Phone,” Available at: http://www.wirlab.net/kphone/

[14] Microsoft, “MSN Messenger v. 6.1,” Available at: http://messenger.msn.com/

[15] ITU-T, “Call Signaling protocols and media stream packetization for packet-based multimedia communication systems,” Recommendation H.225.0, February 1988.

[16] ITU-T, “Control protocol for multimedia communication,” Recommendation H.245, September 1988.

[17] tcpdump/libpcap, Available at: http://www.tcpdump.org/