



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTEGANOGRAFÍA SOBRE PROTOCOLOS DE RED/INTERNET

Alumno: Jose Antonio López Rufián

Tutor: Prof. D. Manuel José Lucena López
Dpto: Informática

Septiembre, 2017



Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Manuel José Lucena López, tutor del Proyecto Fin de Carrera titulado:
Esteganografía sobre protocolos de red/Internet, que presenta Jose Antonio López
Rufián, autoriza su presentación para defensa y evaluación en la Escuela Politécnica
Superior de Jaén.

Jaén, Septiembre de 2017

El alumno:

**LOPEZ RUFIAN,
JOSE ANTONIO
(FIRMA)**

Firmado digitalmente
por LOPEZ RUFIAN,
JOSE ANTONIO (FIRMA)
Fecha: 2017.09.04
18:10:10 +02'00'

Jose Antonio López Rufián

Los tutores:

Digitally signed by LUCENA
LOPEZ MANUEL JOSE -
26477356D
Date: 2017.09.04 11:20:53
CEST

Manuel José Lucena López

Índice

1. Introducción	6
1.1. Motivación	6
1.1.1. Situación actual.....	6
1.1.2. Bloqueos en redes sociales y aplicaciones de mensajería y llamadas ...	7
1.1.3. Bloqueo de servicios cifrados	8
1.2. Objetivos	9
2. Antecedentes	10
2.1. Esteganografía	10
2.2. Historia	10
2.3. Sistemas esteganográficos	13
2.4. Covert Channels sobre TCP/IP	16
2.5. Herramientas existentes.....	17
3. Metodología	18
3.1. Planificación temporal	18
3.2. Tecnologías utilizadas	18
3.2.1. Node.js.....	20
3.2.2. Electron.....	21
3.2.3. Angular JS y Bootstrap	22
3.2.4. Raw-Socket.....	23
3.2.5. Express.js	23
3.2.6. AES-js.....	23
3.2.7. Typescript	24
4. Diseño del estego-sistema.....	25
4.1. Cifrado	25
4.2. Advanced Encryption System.....	28
4.3. Estructura del estego-mensaje	28
4.4. Compresión	30
5. Estego-medios	30
5.1. ICMP	31
5.2. VoIP.....	32
5.3. HTTP Requests	35
5.4. HTTP Video Streaming.....	39

6.	Prueba de concepto: StegoNet	41
1.1.	Estructura de la aplicación	41
6.1.1.	Consideraciones previas.....	41
6.1.2.	Main Process	41
6.1.3.	Renderer Process	43
6.1.4.	Stego-chunk.....	45
6.1.5.	Construcción del estego-mensaje	45
6.1.6.	Obtención del mensaje	46
6.2.	ICMP	47
6.2.1.	Emisor.....	47
6.2.2.	Receptor	47
6.2.3.	Ejemplo	48
6.3.	HTTP Requests	51
6.3.1.	Emisor.....	51
6.3.2.	Receptor	51
6.3.3.	Ejemplo	52
6.4.	Despliegue de la aplicación	54
6.5.	Manual de Usuario	55
6.5.1.	Ventana principal	55
6.5.2.	Ventana de mensajería.....	56
6.5.3.	Ejemplo práctico	57
7.	Resultados	59
7.1.	Limitaciones	60
7.2.	Detección y contramedidas	61
8.	Conclusiones	62
8.1.	Dificultades encontradas	64
9.	Líneas de futuro	66
10.	Bibliografía.....	67
11.	Anexo I. Contenido Suministrado en CD	68

Índice de Ilustraciones y tablas

Ilustración 1. Número de países donde aplicaciones han sido bloqueadas o usuarios arrestados.	8
Ilustración 2. Patrón "Bow Tie".	11
Ilustración 3. Los ojos de "La Gioconda" de Leonardo Da Vinci.....	11
Ilustración 4. La Última Cena de Leonardo Da Vinci.	12
Ilustración 5. Esquema de un estego-sistema.....	13
Ilustración 6. Características de un sistema esteganográfico.	15
Ilustración 7. Estructura de un datagrama UDP.....	16
Ilustración 8. Logo Node.js.	20
Ilustración 9. Logo V8 Engine by Google.	20
Ilustración 10. Logo Electron Framework.....	21
Ilustración 11. Logos de Bootstrap 4 y Angular 4.....	22
Ilustración 12. Conjuntos JavaScript y Typescript.	24
Ilustración 13. Algoritmo de cifrado CTR.....	26
Ilustración 14. Algoritmo de cifrado OFB.	26
Ilustración 15. Modo de cifrado ECB.....	26
Ilustración 16. Algoritmo de cifrado AES.....	28
Ilustración 17. Estructura estego-mensaje.....	29
Ilustración 18. Estructura de un datagrama ICMP.	31
Ilustración 19. Diagrama de una llamada con SIP.	33
Ilustración 20. Pila de protocolos VoIP.....	33
Ilustración 21. HTTP Request.	36
Ilustración 22. Primera ventana de la aplicación.	43
Ilustración 23. Ventana de envío y recepción de mensajes.	44
Ilustración 24. Datagrama IP.	48
Ilustración 25. Ejemplo de envío de mensaje usando ICMP.	49
Ilustración 26. Tráfico capturado con Wireshark del mensaje del ejemplo.....	50
Ilustración 27. Ejemplo de mensajes recibidos usando HTTP Request.	52
Ilustración 28. Captura de tráfico con Wireshark de tráfico HTTP.....	53
Ilustración 29. Captura de tráfico con WireShark de un estego-mensaje HTTP.	53
Ilustración 22. Primera ventana de la aplicación.	56
Ilustración 30. Ventana de mensajería con zonas marcadas.	56
Ilustración 31. Ejemplo de envío de mensaje usando HTTP Request.	58
Ilustración 32. Progreso del mensaje enviado. Los estego-mensajes llegan desordenados.	58
Ilustración 33. Mensaje enviado.	59
Ilustración 34. Balance final del estego-sistema.	63
Tabla 1. Versión de Electron.....	22
Tabla 2. Versiones de Angular y Bootstrap.	22
Tabla 3. Mensajes recibidos por Main Process usando IPC..	42
Tabla 4. Mensajes recibidos por Renderer Process usando IPC.....	44

1. Introducción

En el presente trabajo se pretende proporcionar una herramienta basada en la Esteganografía y en su capacidad para crear canales ocultos de información que dificulten a un atacante espiar nuestras comunicaciones y robar nuestros datos. Lógicamente, la Esteganografía, al igual que sucede con la Criptografía, es una técnica de doble uso y en la práctica puede ser utilizada con fines maliciosos.

Este documento seguirá un orden cronológico del trabajo realizado, exponiendo las motivaciones principales de este proyecto y la solución final, el diseño de un estego-sistema y una aplicación que hará uso del mismo, StegoNet.

1.1. Motivación

Este proyecto surge para explorar la posibilidad de establecer canales de comunicación capaces de pasar inadvertidos frente a observadores maliciosos. Como veremos a continuación, estamos expuestos a numerosas amenazas a nuestras comunicaciones, lo que justifica el interés de este trabajo.

1.1.1. Situación actual

En su esfuerzo por aumentar la seguridad nacional y reforzar el poder de la ley, un gran número de gobiernos han aprobado nuevas leyes que limitan la privacidad y autorizan la vigilancia generalizada. Esta tendencia está presente tanto en países democráticos como no democráticos y da lugar a debates sobre hasta qué punto los gobiernos pueden tener acceso a comunicaciones cifradas utilizando puertas traseras.

Los peores ejemplos los encontramos en países autoritarios donde los gobiernos utilizan las leyes antiterroristas para perseguir a los ciudadanos simplemente por escribir sobre democracia, religión o derechos humanos.

Estas son las conclusiones que se pueden extraer del último informe [1] publicado en Freedom on the Net, un proyecto de Freedom House en el cual publican un reporte anual analizando la libertad y censura de las comunicaciones a través de internet en 65 países cubriendo así el 88% del total de los usuarios de internet.

Las medidas recogidas en dichos reportes se dividen en las siguientes categorías:

- **Obstáculos de acceso:** detalles sobre las barreras económicas y de infraestructuras, control de los proveedores de acceso e independencia de los cuerpos regulatorios.

- **Limitaciones de contenido:** analizan regulaciones en el contenido, filtrado y bloqueo de sitios web, censura, diversidad de nuevas noticias en medios online y el uso de herramientas para movilizaciones ciudadanas.

- **Violaciones de los derechos de los usuarios:** aborda la vigilancia, libertad de expresión, privacidad y repercusiones por la actividad de los usuarios en internet como persecuciones o encarcelamientos.

1.1.2. Bloqueos en redes sociales y aplicaciones de mensajería y llamadas

Gobiernos en 15 países cortaron temporalmente el acceso a todo internet o redes móviles, algunas veces simplemente para evitar que usuarios publicaran información en redes sociales.

La represión de los usuarios por sus actividad en redes sociales o aplicaciones de mensajería aumentó al mismo tiempo que se intensificaban los arrestos y castigos.

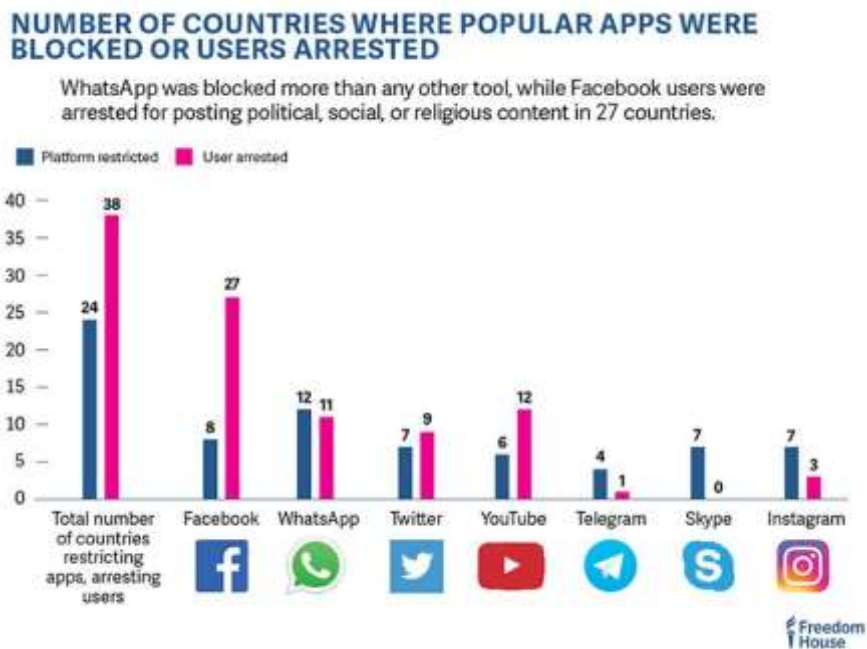


Ilustración 1. Número de países donde aplicaciones han sido bloqueadas o usuarios arrestados.

WhatsApp es la aplicación que más restricciones suma, un total de 12 de los 65 países estudiados bloquearon el servicio completo o algunas de sus características. Telegram, Facebook Messenger, Line y Hangouts también fueron parcialmente bloqueadas.

Diez países bloquearon el acceso a plataformas de llamadas de voz y vídeo sobre internet como Skype o Facetime.

1.1.3. Bloqueo de servicios cifrados

Los gobiernos han incrementado las restricciones a servicios de mensajería y llamadas que utilizan protocolos de cifrado seguros ya que dificultan el acceso a las comunicaciones de los usuarios.

Nuevas aplicaciones que proporcionan un alto nivel de anonimato, comparado con los SMS convencionales, o aquellas que utilizan fuertes cifrados punto a punto que evitan la escucha de comunicaciones, son directamente bloqueadas.

1.2. Objetivos

Tal y como se ha expuesto en el apartado anterior, necesitamos nuevas técnicas que permitan la comunicación por un canal inseguro, partiendo de la premisa de que la comunicación está siendo monitorizada. Añadiendo un hándicap, no basta con cifrar la comunicación puesto que si se detecta la misma sería bloqueada.

En este proyecto se hará uso de técnicas de Esteganografía en protocolos de red donde la cantidad de tráfico generado dificulte la distinción de un paquete de red legítimo de uno que contenga información oculta.

Para ello se estudiarán los avances conseguidos en este ámbito hasta la fecha y se elegirá un protocolo para realizar una aplicación como prueba de concepto que sea extensible a cualquier otro protocolo de red con poco esfuerzo de desarrollo.

Así pues, el objetivo principal del proyecto se puede desglosar en los siguientes apartados:

1. Diseñar un estego-sistema que permita dividir un mensaje a enviar en trozos y su reensamblado en destino.
2. Estudiar protocolos de red con gran cantidad de tráfico donde poder ocultar los mensajes como protocolos de streaming de vídeo, VoIP, etc...
3. Desarrollar una aplicación como prueba de concepto donde se ponga en práctica los dos puntos anteriores.

Estos objetivos principales pueden ser complementados con los siguientes objetivos secundarios o deseables:

1. El estego-sistema diseñado para dividir, enviar y recuperar el mensaje pueda hacerse extensible a cualquier protocolo de red.
2. La aplicación desarrollada pueda ser utilizada en varios sistemas operativos.

2. Antecedentes

En esta sección se pretende introducir al mundo de la Esteganografía detallando su historia y su uso en la actualidad, amén de introducir los covert-channels y el uso que tendrán en el presente proyecto.

2.1. Esteganografía

El término Esteganografía, proviene de las palabras griegas Steganos (oculto) y Graphein (escribir) se puede traducir como escritura oculta, por lo tanto podemos definir la Esteganografía como la ciencia y el arte de ocultar una información dentro de otra, que haría la función de tapadera o cubierta, con la intención de que no se perciba ni siquiera la existencia de dicha información.

A menudo la utilización de esta tapadera o cubierta con fines esteganográficos recibe el nombre de “estego-medio”.

En teoría, sólo quienes conozcan cierta información acerca de la ocultación, estarían en condiciones de descubrirla.

2.2. Historia

Las técnicas esteganográficas se inventaron hace miles de años, en la antigua China ya se empleaban para enviar mensajes ocultos entre personas. Se empleaban diversos métodos como tatuajes, tintas invisibles, rejillas o cambios de letras.

La historia nos da ejemplos, a cual más llamativo e interesante, de técnicas esteganográficas como la de John Wilkins (1614-1672), que desarrolló procedimientos para ocultar información utilizando dibujos geométricos, puntos, líneas o triángulos de un dibujo para enmascarar información.

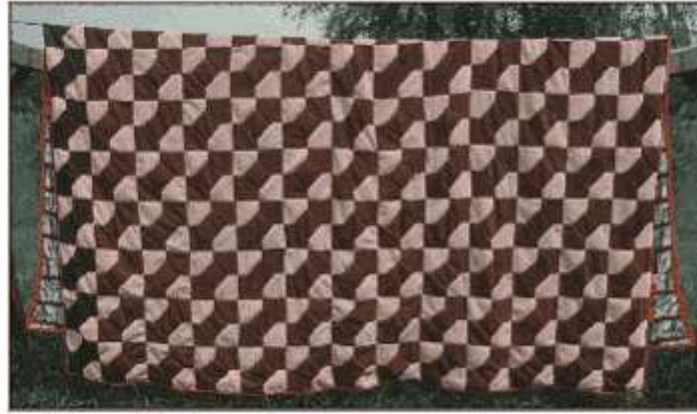


Ilustración 2. Patrón "Bow Tie".

Uno de los grandes maestros de la historia, el artista e inventor Leonardo Da Vinci utilizó técnicas esteganográficas para ocultar mensajes en algunos de sus cuadros. El ejemplo más conocido son iniciales y números en las pupilas de la más famosa de sus obras, La Gioconda.

En el año 2010 [\[12\]](#), gracias a unas reprografías digitales realizadas al cuadro, los expertos del Comité Nacional para el Patrimonio Cultural de Italia descubrieron una serie de iniciales y números ocultos en el iris de Mona Lisa: así, en el ojo derecho se encontraron las letras LV (aludiendo al nombre del autor) y en el izquierdo lo que podría parecer una CE o B. También en el arco del puente encontraron un 72 o la letra L y el número 2.



Ilustración 3. Los ojos de "La Gioconda" de Leonardo Da Vinci

Otro ejemplo lo podemos encontrar en el fresco pintado por el maestro italiano en el año 1498, La Última Cena. Son muchas las teorías que envuelven a esta pintura acentuadas desde la publicación por Dan Brown de su libro *“Davinci’s Code”*. Una de las más fáciles de visualizar ya que Leonardo no pintó aureolas a ninguno de los presentes, sugiriendo que los apóstoles eran personas normales y que incluso Jesús era mortal.



Ilustración 4. La Última Cena de Leonardo Da Vinci.

Esta y otras teorías son posibles mensajes ocultos que utilizó el inventor para transmitir información aunque es algo que nunca sabremos con certeza.

Las técnicas han ido evolucionando hasta llegar a nuestros días, en los que la tecnología digital ha hecho cambiar radicalmente todas estas técnicas y utilizar los contenidos digitales para ocultar los mensajes.

En la II Guerra Mundial, los alemanes emplearon documentos gráficos reducidos a un punto de menos de un milímetro de diámetro que se imprimían sobre cualquier carta al final de una línea, periódicos o debajo de sellos.

En nuestros días esta técnica ha evolucionado y se usa en impresoras que imprimen puntos microscópicos invisibles al ojo humano y que permiten, entre otras cosas, rastrear un documento una vez impreso.

En el año 2001, Phil Carmody en su intento por evadir las jurisprudencia de los EEUU al querer publicar el código fuente del programa DeCSS (Decoder Content Scramblins System), un software que anulaba el sistema de protección de los DVD y facilitaba su reproducción, pensó en buscar un número primo cuya representación binaria se correspondiese con la versión comprimida (algoritmo gzip) del programa escrito en lenguaje C. La jurisprudencia de los EEUU no regula la publicación de un número primo.

2.3. Sistemas esteganográficos

Para construir un estego-sistema vamos a diferenciar claramente los elementos que intervienen:

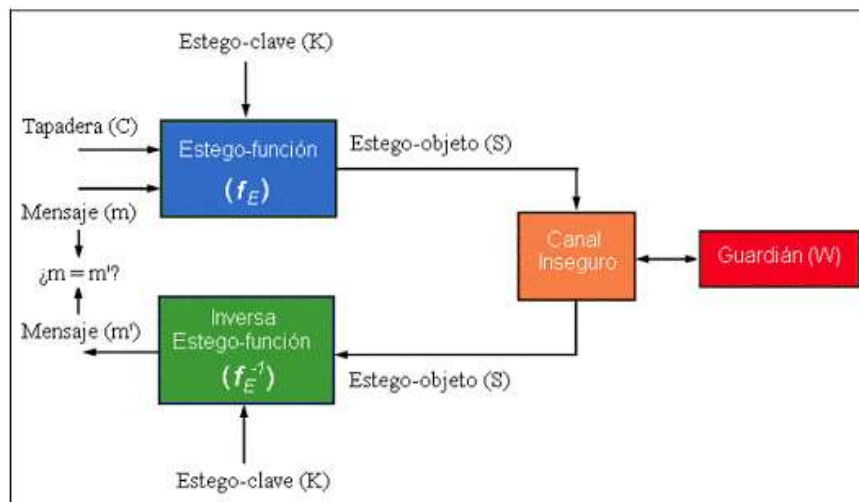


Ilustración 5. Esquema de un estego-sistema.

- **Mensaje:** Es el mensaje que se desea transmitir.
- **Portador o estego-medio:** Es el medio u objeto susceptible de ser alterado en el cual vamos a esconder la información.
- **Estego-objeto o estego-mensaje:** El objeto resultante de ocultar el mensaje en la tapadera.
- **Estego-clave:** Clave compartida por emisor y receptor con la que se cifrará el mensaje.
- **Estego-función:** Es la función o algoritmo encargado de ocultar/extraer el mensaje en/de la tapadera.

- **Canal:** El canal dependerá del entorno en el que nos encontremos, pudiendo ser una persona que transporta una carta con un mensaje oculto en ella o un canal de comunicación de red a través de internet.
- **Guardián:** En la Ilustración 5, este rol identifica al agente que vigila el canal y el cual tratamos de evadir con el sistema esteganográfico.

En la Esteganografía moderna la seguridad de los procedimientos esteganográficos no depende de mantener en secreto el algoritmo de ocultación e incluso tampoco del tipo de la tapadera/cubierta utilizada. La seguridad recae exclusivamente en mantener una pequeña información secreta entre los intervinientes de la comunicación enmascarada, lo más común es una clave.

En la actualidad podemos distinguir tres tendencias en el diseño de estos sistemas:

1. **Estego-sistemas de clave simétrica:** Emisor y receptor comparten una clave secreta (estego-clave) y toda la seguridad del sistema se aposenta en ella. Lo normal es que el algoritmo esteganográfico y el tipo de tapadera utilizado sean públicos.
2. **Estego-sistemas de clave pública:** Requieren el uso de dos claves. Una clave pública para el proceso de ocultación y una clave privada para obtener el mensaje oculto.
3. **Estego-sistemas cuánticos:** Aprovechan los conocimientos sobre física cuántica para diseñar sistemas que faciliten la ocultación de información.

Cuando se diseña un sistema esteganográfico debemos buscar un equilibrio entre 3 características: capacidad (cantidad de información que puede ser ocultada), seguridad/invisibilidad (probabilidad de detección por un estego-analista) y robustez (cantidad de alteraciones dañinas que el medio puede soportar antes de que se pierda la información oculta).

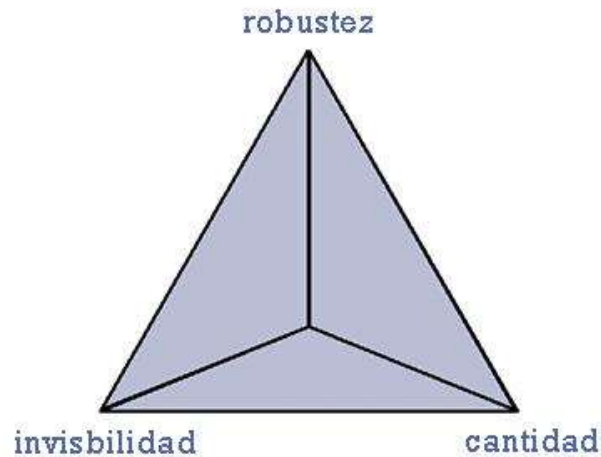


Ilustración 6. Características de un sistema esteganográfico.

Para poder mantener centrada la pirámide de características debemos atender a las siguientes cuestiones cuando diseñemos un estego-sistema:

1. **Introducir los ataques estegoanalíticos** (visuales, estadísticos, caracterización del portador, etc.) en el diseño del algoritmo esteganográfico. Debemos probar si el algoritmo desarrollado es seguro a los ataques y tendencias conocidas.
2. **Elección del portador.** La tendencia de la comunidad científica es recomendar la utilización de cubiertas únicas por cada transmisión encubierta, es decir, utilizar distintas técnicas o protocolos para dificultar ataques de comparación y dificultar la tarea del estego-analista.
3. **Caracterización del portador.** Una vez elegido un tipo de portador, por ejemplo protocolos de red, es importante destacar que no todos los portadores de este tipo son igual de seguros. Por ejemplo, no es lo mismo incluir mensajes ocultos en el campo payload del protocolo ICMP que un protocolo que genere gran cantidad de tráfico.
4. **Limitaciones de los algoritmos estegoanalíticos.** Este hecho se puede resumir coloquialmente de la siguiente manera: a menor información oculta, la probabilidad de detección será más baja.

2.4. Covert Channels sobre TCP/IP

Un Covert Channel [2] es un canal de comunicación oculto que permite a dos procesos de un sistema la transferencia de información, de forma que el proceso de comunicación no es parte del diseño original del sistema.

En 1973 Butler W. Lampson definiría la idea de canal oculto o covert-channel, como aquel canal de comunicación que viola una o más políticas de seguridad del sistema donde se aplica.

Desde un punto de vista esteganográfico, en la mayoría de protocolos existen ciertas debilidades o consideraciones no estrictamente definidas en diseño, que nos permitirán crear canales ocultos de información en una conexión de datos más o menos común.

La creación de canales ocultos, es viable mediante la manipulación del formato de los paquetes a enviar mediante el uso de campos reservados o que no se validan, la reordenación de los paquetes a enviar, la creación de canales en función del tiempo que se tarda en enviar una serie de informaciones [3], etc.

Se han documentado técnicas de ocultación para multitud de protocolos: HTTP, TCP, UDP, Ipv4, IPv6, ICMP, IPSEC, IGMP, FTP, DNS, protocolos de redes de área local (802.2, 802.3), redes inalámbricas, etc.

Como ejemplo se va a ilustrar un datagrama del protocolo UDP (User Datagram Protocol) definido en la [RFC 768](#) y los lugares en los que se puede ocultar información.

UDP es un protocolo no orientado a conexión, es decir para transferir datos entre dos entidades del nivel UDP no es necesario que, de forma previa, se establezca una conexión entre ellas. UDP es un protocolo no fiable porque no garantiza la entrega de datos al destino, los datos pueden perderse, duplicarse o entregarse desordenados.

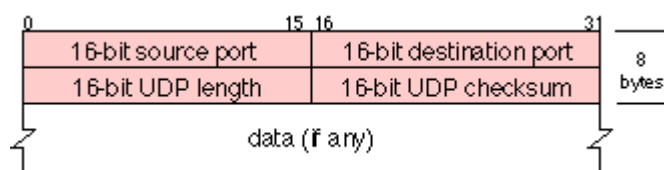


Ilustración 7. Estructura de un datagrama UDP.

Escenarios de ocultación:

- **Source port:** Permite indicar al receptor el puerto por el que está transmitiendo el emisor. En este campo se pueden ocultar 16 bits de información por cada paquete enviado.
- **Checksum:** Este campo se utiliza para que el receptor pueda detectar errores en el paquete recibido. Si utilizamos este campo para enviar información, al llegar a destino y comprobar la suma de verificación daría resultado erróneo, sin embargo si aceptamos el paquete en lugar de descartarlo tendríamos la información oculta. Para comprobar que los datos que se envían son correctos se podría **añadir un checksum a los propios datos enviados**.

2.5. Herramientas existentes

Existen multitud de herramientas diseñadas para el mismo propósito que se trata en este proyecto, sin embargo la mayoría de ellas se escribieron hace bastantes años como pruebas de concepto y además de ser difícil (o imposible) de encontrar, algunas sólo funcionan bajo Windows XP.

No obstante, está bien mencionar algunas de ellas para tener una referencia de aplicaciones que se diseñaron y funcionaron en su momento:

- **Covert_tcp.** Este programa permite manipular la cabecera de los paquetes TCP/IP para transferir información oculta a una máquina destino. Este programa puede actuar como servidor y cliente, y oculta los datos en la cabecera de los datagramas IP.
- **StegTunnel.** Permite establecer canales ocultos de información utilizando el campo de identificación de los datagramas IP y los campos de número de secuencia de cualquier conexión TCP. Tiene una arquitectura cliente-servidor que deben compartir previamente una clave secreta para poder descifrar la información intercambiada.

3. Metodología

3.1. Planificación temporal

El proyecto se inició el 14 de Noviembre de 2016 y se finaliza el 1 de Septiembre de 2017.

La planificación inicial de este proyecto se divide en 3 partes y cada parte debía completarse en 1 mes y medio dedicando 3 horas y media al día:

- Parte 1 - Documentación sobre estudios referentes a la Esteganografía en protocolos de red
- Parte 2 - Elegir protocolo de red y tecnología para desarrollo de prueba de concepto
- Parte 3 - Desarrollar prueba de concepto, analizar resultados y redacción de la memoria

La primera parte se llevaría a cabo durante los meses de Noviembre de 2016 y Enero de 2017 dejando el mes de Diciembre con menos carga de trabajo para poder entregar prácticas y preparar exámenes de las asignaturas matriculadas.

La segunda parte se ejecutaría durante el mes de Febrero y Marzo de 2017 pasados los exámenes.

La tercera parte se realizaría durante los meses de Abril y Mayo de 2017.

3.2. Tecnologías utilizadas

Tras concluir la fase de búsqueda de información realicé un análisis de las posibles tecnologías que podría utilizar para la realización del proyecto.

El lenguaje de programación que se eligiese debía cumplir unos requisitos previos como:

- **Requisito 1:** Debe poder crear sockets de red.
- **Requisito 2:** Ejecutable para cualquier sistema operativo.

- **Requisito 3:** Puesto que el fin del proyecto es desarrollar una aplicación que pueda utilizarse en software existente, es deseable que haya abundante software libre desarrollado en el que sea sencillo incorporar esta técnica.

Dado que recientemente había estado trabajando con Python decidí comenzar mi búsqueda por este lenguaje de programación.

Parecía muy prometedor ya que cumplía los requisitos 1 y 2 por lo que profundicé y descubrí Twisted, un framework de red para python muy completo y potente.

Comencé a realizar algunas pruebas con un servidor Asterisk para tratar de escribir una pequeña aplicación con Twisted que utilizase el protocolo SIP para establecer una llamada VoIP, donde posteriormente se aplicaría algún protocolo esteganográfico.

Tras dos semanas probando el framework descubro que su integración con una interfaz gráfica es más compleja de lo esperado y pronunciaba más si cabe su curva de aprendizaje.

En este momento decido cambiar de tecnología aún sabiendo que me exponía a retrasarme en los plazos establecidos. De no haber cambiado de tecnología el aprendizaje del framework me hubiese retrasado más que el cambio.

Puesto que tengo experiencia en tecnologías web, la segunda alternativa tras desechar python fue utilizar Node.js.

3.2.1. Node.js



Ilustración 8. Logo Node.js.

Node.js es un entorno de ejecución de código JavaScript open-source, construido sobre el motor de JavaScript V8 de Chrome diseñado para ejecutar código del lado del servidor. Node.js utiliza un modelo de E/S no bloqueante orientado a eventos, lo que le hace ligero y eficiente. El ecosistema de paquetes de Node.js, NPM, es actualmente el repositorio de librerías open-source más grande del mundo.

La comunidad de Node.js es muy amplia contando con alrededor de 7 millones de desarrolladores y en su repositorio NPM podemos encontrar paquetes casi para cualquier propósito. Si esto no fuese suficiente, es posible desarrollar librerías propias en C que después serán utilizadas por nuestros paquetes JavaScript. Esto hace cumplir el requisito 3.

Node.js permite la creación de servidores web y herramientas de red usando JavaScript y una colección de “módulos” que gestionan varias funcionalidades como E/S en el sistema de ficheros, acceso a la red (DNS, HTTP, TCP, TLS/SSL o UDP), datos binarios (buffers) o funciones criptográficas entre otras.

Funciona mediante un único hilo de ejecución, utilizando llamadas de E/S no bloqueantes, permite soportar decenas de miles de conexiones simultáneas sin el coste del cambio de contexto entre múltiples hilos.



Ilustración 9. Logo V8 Engine by Google.

V8 es el motor de ejecución de JavaScript creado para Google Chrome. Escrito en C++, compila el código JavaScript a código máquina nativo en lugar de interpretarlo en tiempo real.

Node.js utiliza libuv para gestionar eventos asíncronos. Libuv es una capa de abstracción para funcionalidades de red y sistema de ficheros para sistemas operativos Windows, Linux y macOS

3.2.2. Electron



Ilustración 10. Logo Electron Framework.

Electron es un framework open source desarrollada por GitHub para construir aplicaciones de escritorio con HTML, CSS y JavaScript. Para conseguir esto combina el navegador Chromium y Node.js en un único entorno de ejecución.

Las aplicaciones desarrolladas con Electron pueden ser compiladas para Linux, Mac OS y Windows.

La idea es usar el navegador web Chromium como interfaz gráfica y Node.js como un servidor que se ejecuta en segundo plano y será el que dará acceso a cumplir el requisito 2.

Tal y como se ha explicado antes, Electron se compone de 2 partes, el navegador web y el servidor Node.js, es decir, la parte encargada de renderizar las vistas que muestran al usuario se ejecutan en el proceso Renderer y el servidor Node en el proceso Main.

Para poder comunicar estos dos procesos se usará IPC o Inter Process Communication.

Este sencillo mecanismo nos va a permitir establecer una comunicación basada en mensajes entre los dos procesos.

Paquete	Versión
Electrón	1.6.11
Node.js	7.4.0
Chromium	56.0.2924.87

Tabla 1. Versión de Electron.

3.2.3. Angular JS y Bootstrap

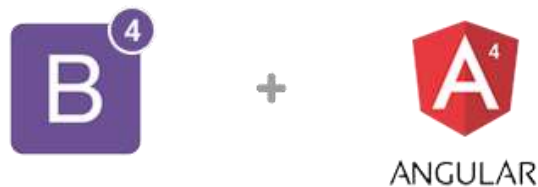


Ilustración 11. Logos de Bootstrap 4 y Angular 4.

Electron nos facilita el uso de HTML y JavaScript para el desarrollo de la interfaz pero para facilitar el desarrollo y utilizar un paradigma de programación asíncrona para comunicar los dos procesos de Electron, es aconsejable el uso de un framework que nos ayude en esta tarea. Es por ello que para el desarrollo de la interfaz he utilizado Angular JS en su versión 4.0.8.

Paquete	Versión
Angular	4.0.0
Bootstrap	4.0.0-alpha.6

Tabla 2. Versiones de Angular y Bootstrap.

3.2.4. Raw-Socket

Es un paquete para Node.js el cual nos permite crear sockets de red en cualquier puerto en un servidor Node.js.

Los sockets creados por este paquete soportan IPv4 e IPv6 y son creados utilizando la función del sistema operativo `socket()`.

Algunos sistemas operativos (como es el caso de Linux) restringen el uso de determinados puertos a superusuarios o administradores.

Este es un módulo nativo de Node.js lo que significa que incluye código escrito en C++ y que será necesario compilar durante el proceso de instalación

3.2.5. Express.js

Express es un framework de aplicaciones web para Node.js con licencia del MIT. Está diseñado para construir aplicaciones web y APIs.

Express nos permitirá manejar todos los “verbos” de HTTP (GET, POST, PUT, DELETE...), establecer diferentes rutas en URLs específicas o servir ficheros estáticos (HTML, JavaScript, CSS...) todo ello manualmente, aunque también se podría utilizar algún framework basado en express que facilite la tarea, aunque no es ese el objetivo de este proyecto.

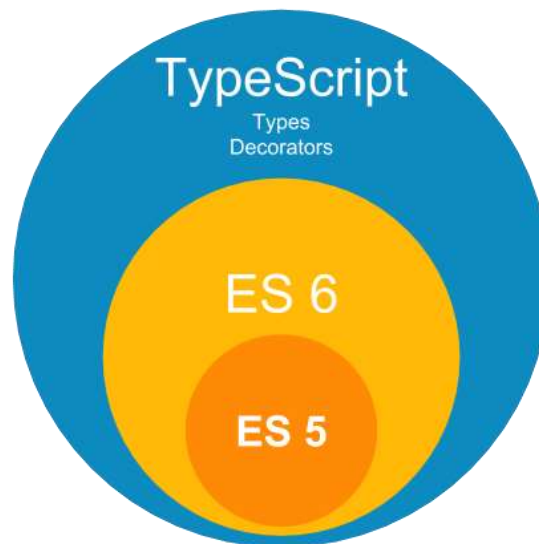
3.2.6. AES-js

Este módulo de Node.js es una implementación en JavaScript del algoritmo de cifrado por bloques AES y sus modos de operación (CBC, CFB, CTR, ECB y OFB).

Soporta tamaños de clave de 128, 192 y 256 bits.

3.2.7. Typescript

Typescript es lenguaje de programación open source desarrollado y mantenido por Microsoft. Es un superconjunto de ECMAScript 6 (Javascript), es decir, incluye todas las funcionalidades de ES6, y además incorpora una capa por encima con funcionalidades extra.



TypeScript is a javascript superset

Ilustración 12. Conjuntos JavaScript y Typescript.

La principal característica de TypeScript por encima de Javascript es que permite tipado estático de variables lo que aportará mayor robustez al código, pero por contra puede hacer perder flexibilidad.

Otra característica que TypeScript mejora sobre JavaScript es el uso de clases de objetos y encapsulamiento de los miembros de las clases.

Utilizar este lenguaje de programación en el proyecto aportará los beneficios del tipado estático que permitirá identificar posibles errores en el código al compilar y no en tiempo de ejecución como sucede con JavaScript. Además brinda la posibilidad de trabajar con clases y todos sus beneficios como herencia y encapsulamiento.

4. Diseño del estego-sistema

En este apartado se realizará un análisis de las alternativas y decisiones tomadas en el proceso de diseño de cada uno de los apartados implicados en el estego-sistema.

El objetivo es diseñar un protocolo que permita dividir un mensaje en trozos y ocultarlos utilizando algún protocolo de red. En destino, se estará ejecutando una función por cada paquete de red recibido correspondiente al protocolo en cuestión la cual comprobará si se trata de un paquete legítimo o de uno con información oculta, en este caso la extraerá y mostrará al usuario.

4.1. Cifrado

La idea de utilizar Esteganografía para enviar los mensajes ocultos, no justifica que estos no puedan ser descubiertos por alguien que esté monitorizando el tráfico de red, por lo tanto deben ser cifrados haciendo que sean indistinguibles de datos aleatorios.

Para llevar a cabo esta tarea se propone el uso de un algoritmo de cifrado por bloques o de flujo pues estas técnicas son las que más se ajustan al problema que se le está dando cobertura en el presente proyecto.

Lo ideal sería utilizar un modo de cifrado como CTR, OFB o ECB los cuales se exponen a continuación:

- **CTR - Counter Mode:** Este modo simula un cifrado de flujo. Usa un cifrado de bloque para producir un flujo pseudo aleatorio conocido como keystream. Este flujo se combina con el texto plano mediante XOR dando lugar al cifrado. Para generar el keystream se cifra un contador combinado con un número aleatorio (nonce) mediante ECB y se va incrementando.

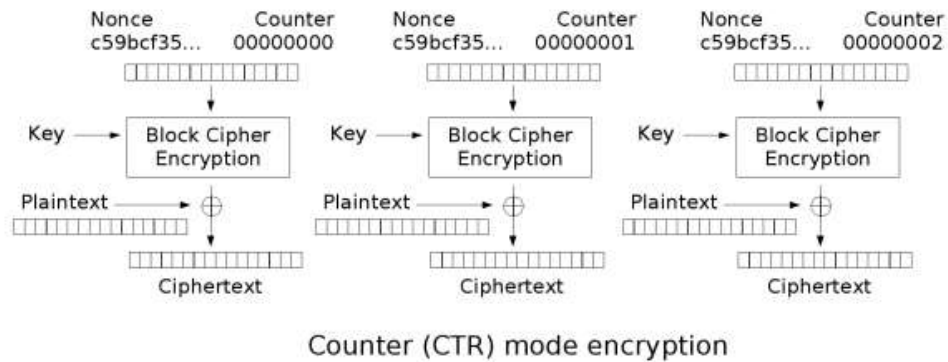


Ilustración 13. Algoritmo de cifrado CTR.

- **OFB - Output Feedback Mode:** Modo de cifrado de flujo. El flujo se genera cifrando el bloque anterior, dando lugar al siguiente bloque. El primer bloque de keystream se crea cifrando un vector de inicialización.

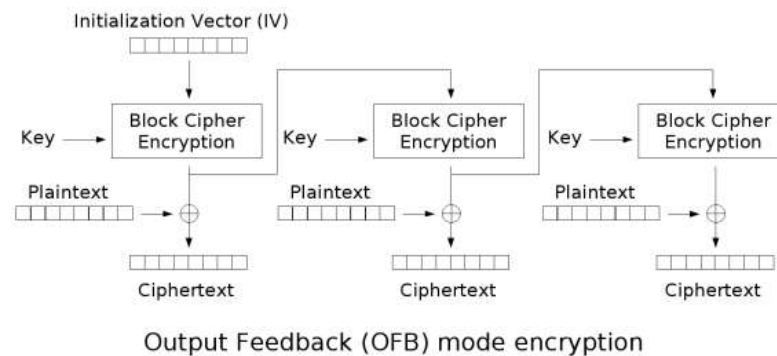


Ilustración 14. Algoritmo de cifrado OFB.

- **ECB - Electronic Code Book:** Se limita a dividir el mensaje en bloques y cifrarlos por separado.

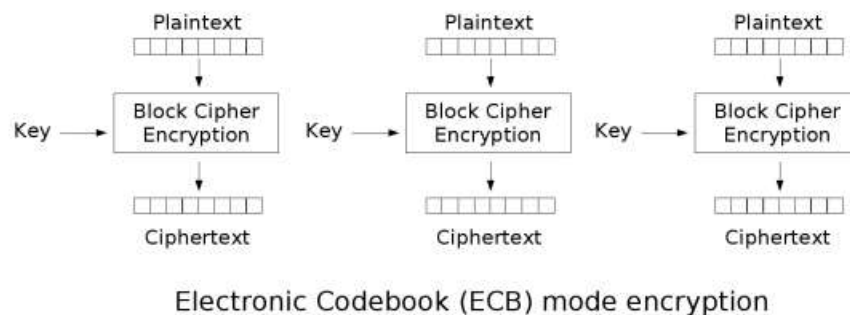


Ilustración 15. Modo de cifrado ECB

Llegados a este punto se han de analizar y sopesar las consecuencias de utilizar cada modo de cifrado.

Por un lado, tenemos los cifrados de flujo, OFB y CTR los cuales aportan una mayor seguridad y dificultad para que un atacante pueda descifrar el mensaje.

También son más robustos en términos de integridad de la información que se enviaría pues si un atacante consiguiera modificar tan solo uno de los mensajes, el mensaje completo sería invalidado y descartado en destino.

Otro punto a favor es el tamaño del bloque pues puede ser variable, esta es una gran ventaja pues nos permitiría poder tener mensajes pequeños y aumentaría el número de lugares donde poder esconder el mensaje.

Sin embargo, no tenemos garantía de que los paquetes en los que se va a ocultar el mensaje lleguen a destino, tómese como ejemplo el protocolo UDP, por lo tanto si se utiliza un modo de cifrado de flujo, con la pérdida de una parte del mensaje sería imposible recuperar el mensaje.

Esto también afectaría a técnicas esteganográficas basadas en tiempo, en las cuales se juega con retrasos en los paquetes que contienen el mensaje oculto para que el paquete se de como perdido aunque finalmente llegue a destino. Si alguno de estos paquetes retrasados es descartado, impediría el uso de esta técnica.

Por otra parte el modo de cifrado por bloques evitaría los problemas anteriores puesto que la pérdida de un mensaje por el uso de un protocolo como UDP no imposibilita la recepción del mensaje de manera parcial, aunque llegaría con huecos por cada mensaje perdido.

El hecho de cifrar los bloques por separado implica que cuando se cifre un bloque con cierto valor, siempre se obtendrá el mismo resultado. Esto hace posible los ataques de diccionario y si un atacante consigue romper el cifrado, podría modificar el mensaje..

4.2. Advanced Encryption System

La decisión tomada pasa por utilizar un modo de cifrado ECB y para evitar que un atacante pudiese romper el cifrado se utilizará el algoritmo AES con clave de 256 bits.

Utilizando este tamaño de clave evitamos que un atacante pueda descifrar los mensajes pues las posibles combinaciones es de 1.1×10^{77} un coste computacional que es imposible de asumir a día de hoy.

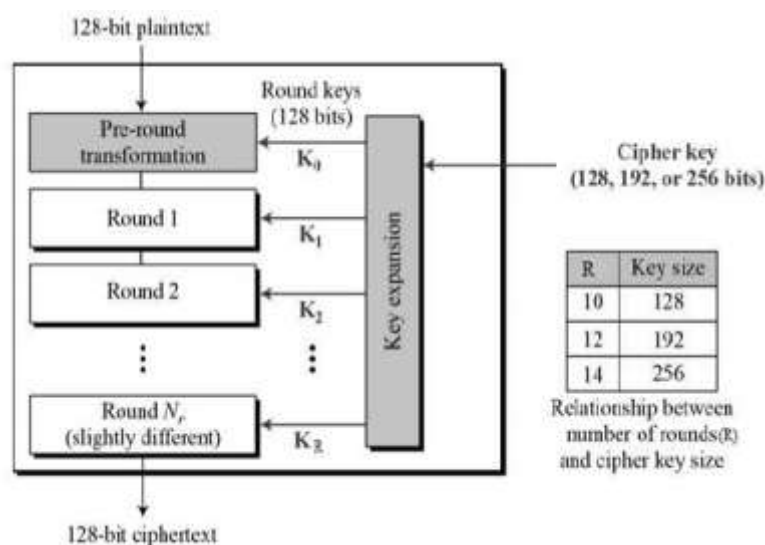


Ilustración 16. Algoritmo de cifrado AES.

AES tiene un tamaño de bloque fijo de 16 Bytes independientemente del tamaño de clave que se utilice. Si la información a cifrar no es múltiplo de 16 se rellenará un nuevo bloque hasta llegar a 16 Bytes.

4.3. Estructura del estego-mensaje

La aplicación debe ser capaz de enviar y recibir múltiples mensajes por lo que cada mensaje deberá ser marcado con un identificador único.

Dada la naturaleza de la solución esteganográfica escogida, deberemos trocear los mensajes para poder encapsularlos en los paquetes de red, por lo que cada trozo de mensaje llevará asociado un número de secuencia.

Para distinguir un paquete legítimo de un estego-mensaje, añadiremos un campo de comprobación de modo que cuando se reciba un paquete y se descifre, si el campo de comprobación coincide con el mensaje, es un paquete que contiene un estego-mensaje.

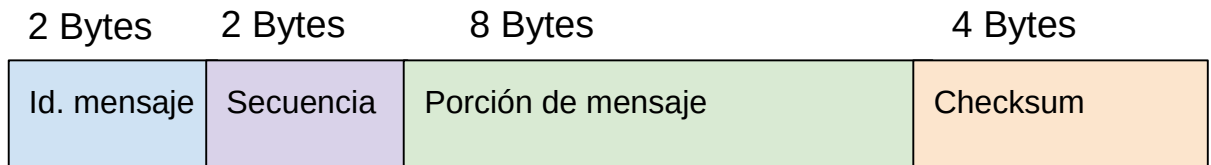


Ilustración 17. Estructura estego-mensaje.

- **Id. mensaje:** Este identificador único se contendrá los dos primeros bytes de un hash generado por el algoritmo SHA3 del mensaje completo. Utilizando 2 bytes podemos distinguir un mensaje de entre 65536.
- **Secuencia:** Será un número que se incrementará en cada trozo de mensaje que enviamos. Con 2 bytes podremos trocear un mensaje en hasta 65536 porciones.
- **Checksum:** Contendrá los cuatro primeros bytes de un hash generado por el algoritmo SHA3 de la porción que estamos enviando del mensaje. Utilizando 4 bytes podemos reducir la tasa de fallo en la que un paquete legítimo sea tratado como un estego-mensaje a una probabilidad de 1 entre 4294967296.
- **Porción de mensaje:** Porción de 8 bytes del mensaje en texto plano. Se usa este tamaño mínimo ya que el algoritmo de cifrado devolverá un bloque de 16 Bytes y teniendo en cuenta el tamaño de los campos anteriores, 8 Bytes es el restante hasta completar los 16 Bytes. Si el campo contiene menos de 8 bytes, se rellenará con el carácter ASCII ".".

Una vez se haya completado todos los campos del mensaje, éste se cifra utilizando un hash de 256 bits de la clave compartida por emisor y receptor.

El tamaño mínimo de un estego-mensaje será de 16 Bytes, que es el tamaño del bloque generado por el algoritmo de cifrado AES.

Dada la estructura del estego-mensaje podemos calcular la longitud máxima de un mensaje de la siguiente forma:

- El campo secuencia se podrá incrementar hasta el índice 65536, esto implica que cada mensaje se podrá dividir en 65536 trozos o estego-mensajes.
- Cada estego-mensaje portará 8 Bytes del mensaje original.
- El resultado indica que el mensaje original deberá tener un máximo de **524288** caracteres ASCII.

Para ejemplificar el resultado anterior, con **524288** caracteres podríamos utilizar esta estructura de estego-mensaje para enviar los 73 capítulos de “El Quijote” a una media de 4 páginas por capítulo y 10300 caracteres por página.

4.4. Compresión

Teniendo en cuenta el resultado del apartado anterior surge la idea de comprimir el mensaje original antes de trocearlo. Esto aumentaría considerablemente la cantidad de información que se puede incluir en un estego-mensaje, sin embargo, utilizar compresión iría contra algunas decisiones tomadas durante la fase de diseño del estego-sistema.

Hasta el momento, se han tomado decisiones que permitan que aunque se pierda algún estego-mensaje, por ejemplo por problemas de red, el mensaje en parte llegue a su destino y sea legible. Utilizar compresión implica que si se pierde un estego-mensaje es posible que no se pueda recuperar al menos parte de él.

5. Estego-medios

En este apartado vamos a identificar qué procedimientos y estego-medios son los más adecuados en función de los intereses perseguidos. Esta elección no será sencilla y depende de factores como: la "invisibilidad" de la técnica esteganográfica, la cantidad de información a ocultar, la tolerancia a ataques activos o las limitaciones del canal de transmisión.

Para dificultar la labor de un estegoanálisis, se pretende buscar protocolos de red que generen mucho tráfico como Streaming de vídeo, videollamadas, etc... y que permitan la ocultación de los bloques generados por el algoritmo de cifrado AES con un tamaño de 16 Bytes.

5.1. ICMP

Hace años era común que los cortafuegos habilitarían el tráfico de ciertos mensajes ICMP, como el tráfico generado por la herramienta ping, que permite determinar si un equipo está funcionando.

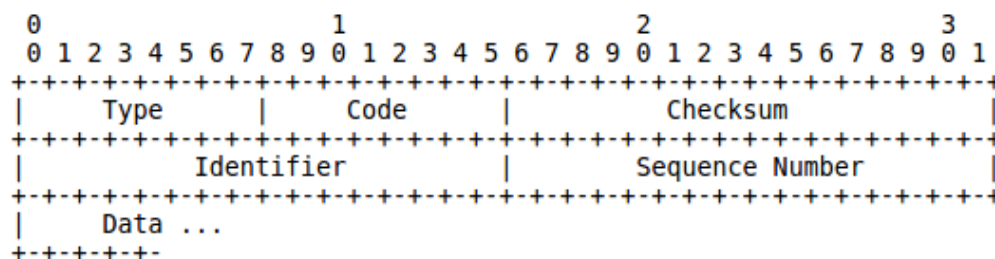


Ilustración 18. Estructura de un datagrama ICMP.

El campo Type indica si se trata de un paquete "Request" o "Reply" y el campo Code indica si existe algún error como "Unreachable" por lo tanto son necesarios y no podemos modificarlos.

Cualquiera de los demás campos podemos modificarlo para introducir información oculta, aunque esta técnica se utilizó por primera vez en el año 1996 y a día de hoy es fácil de detectar.

En el campo Data se suelen escribir datos aleatorios, como una lista de los caracteres ASCII, dependiendo del sistema operativo.

Este protocolo no cumple la condición de generar gran cantidad de tráfico pero puede ser de utilidad durante el desarrollo dada la estructura de un paquete ICMP y el funcionamiento del protocolo.

5.2. VoIP

La telefonía IP o Voz sobre IP (VoIP) es un servicio en tiempo real que permite a los usuarios realizar llamadas telefónicas a través de redes IP. En los últimos años esta tecnología ha adquirido una mayor importancia y crecimiento implantandose en grandes empresas permitiendo la interconexión de sedes y abaratando los costes de las llamadas telefónicas.

Existen muchas técnicas esteganográficas referentes al campo de la VoIP pero todas ellas se pueden agrupar en dos categorías:

- Técnicas que modifican PDU (Protocol Data Unit), como las cabeceras o el cuerpo del mensaje.
- Relación de tiempo entre PDUs. Estas técnicas modifican el tiempo entre cada datagrama del protocolo.

Una llamada IP consta de dos fases: la llamada o signaling y la conversación, ambas usan distintos protocolos de red.

Uno de los protocolos más usados para establecer la conexión es el protocolo SIP (Session Initialization Protocol). Este protocolo es usado para inicializar la llamada y establecer los parámetros de la conexión, así como finalizar la misma.

Los mensajes SIP describen la identidad de los participantes en una llamada y cómo los participantes pueden ser alcanzados sobre una red IP. Encapsulado dentro de los mensajes SIP, algunas veces también podemos ver la declaración SDP (Session Description Protocol) que definirá el tipo de canales de comunicación que pueden ser establecidos para la sesión, esto declarará qué codecs están disponibles.

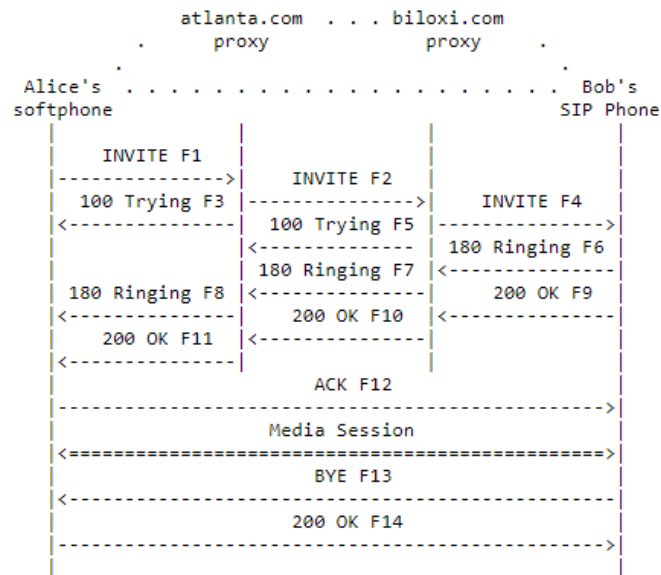


Ilustración 19. Diagrama de una llamada con SIP.

La ilustración anterior muestra un ejemplo de llamada (signaling) mediante el protocolo SIP, en la que podemos ver como el protocolo se basa en el mecanismo petición-respuesta.

Una vez que el intercambio de los mensajes de configuración son completados, la comunicación puede ser intercambiada utilizando otro protocolo, típicamente RTP (Real-Time Transmission Protocol).

Durante la fase de llamada, dos streams de audio o vídeo son enviados bidireccionalmente entre las partes de la llamada. RTP (Real-time Transport Protocol) es el más usado para transporte de voz y vídeo.

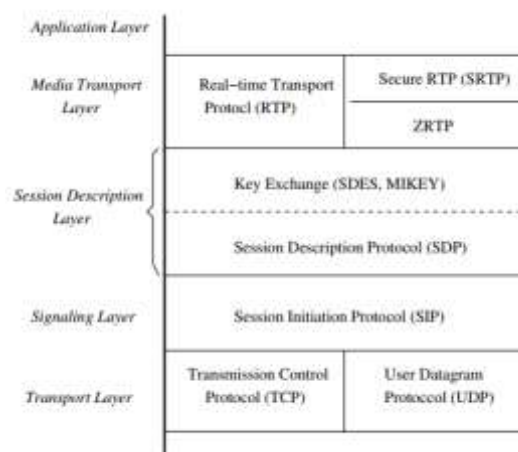


Ilustración 20. Pila de protocolos VoIP.

Las ventajas de usar un protocolo de VoIP como estego-medio son las siguientes:

- El extendido uso de esta tecnología no llamará la atención en un análisis de tráfico de red.
- Gran volumen de datos de conversaciones de VoIP en la red. La comunicación oculta puede pasar desapercibida dado el gran número de llamadas que se producen.
- Gran ancho de banda. En una conversación basada en el códec de audio G.711, cada paquete RTP contiene 20 ms de voz, con una tasa de 50 paquetes por segundo. Con tan sólo ocultar 1 bit en cada paquete obtenemos un ancho de banda de 50 bits por segundo.
- Es un servicio en tiempo real, lo que añade dificultad para un estegoanálisis. Además añade algunas posibilidades para ocultar información como paquetes retrasados que son descartados en destino porque no tendría sentido decodificar voz que ya ha pasado en el tiempo.

Estas ventajas hacen que este protocolo sea un buen candidato a estego-medio, sin embargo, el protocolo diseñado nos obliga a buscar un medio que nos permita transportar 16 Bytes por cada paquete lo cual descarta el uso de técnicas como LSB (Least Significant Bit).

Los autores de [\[7\]](#) proponen el uso del protocolo SIP como estego-medio y facilitan estadísticas del número de mensajes SIP que se pueden enviar durante una llamada sin hacer saltar las alarmas del IDS SNORT, dicho número de mensajes asciende a como máximo 4 mensajes SIP por segundo..

El RFC 3261 define los campos obligatorios que deben aparecer en las cabeceras de los mensajes SIP que son To, From, CSeq, Call-ID, Max-Forwards y Via. SIP proporciona un total de 115 campos para usar en la cabecera de los mensajes de los cuales 109 son opcionales, además SIP puede contener información de SDP para establecer los parámetros de la llamada. SDP utiliza 5 campos obligatorios y 15 opcionales.

Si sumamos lo anterior contamos con 124 campos opcionales en los cuales ocultar información.

Puesto que son muchas las posibilidades no se van a incluir en el presente documento para no hacerlo innecesariamente extenso, en su lugar se destaca el hecho de que si bien no en todos los campos se podría incluir nuestro estego-mensaje de 16 Bytes, una alternativa sería dividir el estego-mensaje en porciones más pequeñas e incluirlas en algunas de las cabeceras opcionales del mensaje SIP. Aún dividiendolo en 16 trozos de 1 Byte, seguiríamos teniendo otros 108 campos opcionales del protocolo.

Otra alternativa basada es la técnica LACK [\[3\]](#) (Lost Audio paCKet) la cual consiste en retener paquetes de audio con contenido oculto en el payload de RTP para que cuando lleguen a destino estos sean descartados para la comunicación pero previamente procesados para extraer el mensaje oculto.

La implementación de LACK es compleja y su efectividad dependerá de la duración de la llamada. Por otro lado, si se retrasan muchos paquetes la calidad de la llamada se deteriorará y esto puede provocar que el covert channel quede expuesto.

Para utilizar un protocolo de VoIP como estego-medio se propone el uso del protocolo SIP y la técnica descrita anteriormente además de la técnica LACK ya que a pesar de su compleja implementación, su detección por parte de un IDS es más compleja.

5.3. HTTP Requests

Ya que el protocolo HTTP es uno de los más usados hoy en día en todo el mundo, no debemos pasarlo por alto como posible estego-medio y es que son muchos los lugares y las técnicas que podemos usar para ocultar mensajes utilizando este protocolo.

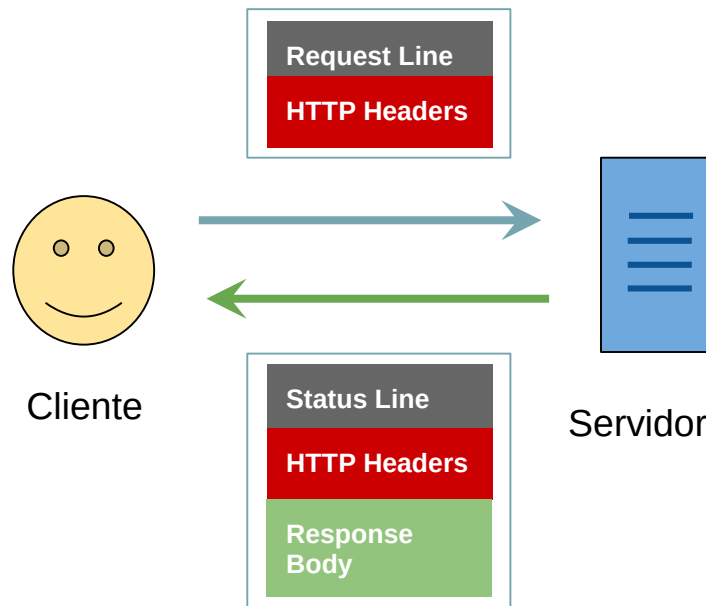


Ilustración 21. HTTP Request.

HTTP es un protocolo cliente-servidor sin estado, que trabaja en la capa de aplicación del modelo OSI. El cliente envía un mensaje “request” y espera a que el servidor conteste con un mensaje “response”.

Un mensaje HTTP tiene la siguiente estructura:

- Línea inicial
 - Peticiones: Método o verbo seguido de la URL del recurso y la versión HTTP
 - Respuestas: Versión HTTP seguida del código de respuesta
- Cabeceras del mensaje que terminan con una línea en blanco
- Cuerpo del mensaje. Es opcional, depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL. Típicamente tiene los datos que se intercambian cliente y servidor.

En el protocolo HTTP se definen una serie de métodos también conocidos como verbos que representan la acción que se solicita, a continuación se detallan los métodos más interesantes para el presente análisis:

- **GET:** Pide una representación del recurso especificado por la URI y sus parámetros. Este método sólo debe devolver datos, no debe tener otro efecto.
- **POST:** Envía los datos que se incluyen en el cuerpo de la petición para que sean procesados por el recurso especificado. El resultado esperado puede ser la creación de un nuevo recurso o actualización de los recursos existentes.
- **DELETE:** Solicita que el servidor elimine el recurso especificado en la URI.
- **PUT:** La intención de este verbo es que el objeto o entidad incluida en la request sea almacenado bajo el recurso especificado por la URI.

Después de recibir e interpretar un mensaje request, el servidor responde con un mensaje HTTP response.

La primera línea de un mensaje de respuesta HTTP contiene la línea de estado compuesta por la versión del protocolo seguida de un código numérico y un texto asociado a dicho código.

El primer dígito del código define el tipo de respuesta. Los dos últimos dígitos se utilizan para especificar la respuesta. Existen cinco valores para el primer dígito:

- 1xx: Informational - Indica que la petición ha sido recibida y se está procesando
- 2xx: Success - La petición ha sido correctamente recibida, entendida y aceptada.
- 3xx: Redirection - Otra acción es requerida para completar la petición
- 4xx: Client Error - La petición contiene algún error y no ha podido ser procesada.
- 5xx: Server Error - El servidor ha fallado al procesar una petición aparentemente válida.

A continuación se muestra un ejemplo de petición GET y una posible respuesta por parte del servidor:

Petición GET

```
GET /index.php?param1=some-value&param2=hello-world-value HTTP/1.1
Host: www.example.com
```

Respuesta

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

HTTP no tiene restricción ni en el tamaño de la URL ni en el cuerpo de los mensajes, esto nos da una posible vía para ocultar mensajes.

Actualmente podemos ver como los enlaces de páginas o aplicaciones web contienen largas cadenas de caracteres aparentemente aleatorios. Este tipo de URLs en las que se incluyen Tokens de identificación del usuario autenticado o identificadores de recurso pueden servir como medio para ocultar un mensaje:

Ejemplo URL Google Docs

```
https://docs.google.com/document/d/1UWYd4W2EbQyVsWcLxC_fXfxvPQn4pN0Us5vwwE_9ZhU/edit#
```

Ejemplo URL con estego-mensaje como parámetro

```
http://192.168.7.26?recurso=54asd78uhnkj3787&token=56tfvyb76yguhb68
```

En este ejemplo se han incluido en una URL dos estego-mensajes de 16 Bytes como parámetros GET de la URL aunque se podría conseguir el mismo efecto usando parámetros POST.

Para llevar a cabo esta técnica sin hacer saltar las alarmas, hay que tener en cuenta que no es habitual realizar muchas peticiones GET o POST seguidas a un mismo recurso simplemente cambiando el contenido de los parámetros.

Para confundir a un posible Guardian podemos realizar pausas entre las peticiones simulando un comportamiento natural de navegación web, cambiando en cada petición el contenido de los parámetros.

Otra posible técnica para confundir sería añadir ruido, es decir, enviar otras peticiones con contenido aleatorio que dificulten el análisis.

De esta forma y en teoría, en una misma petición podemos pasar 2 o 3 mensajes de 16 Bytes cada uno y dejando 2 o 3 segundos entre cada petición conseguimos un ancho de banda de 16 Bytes por segundo.

Si optamos por añadir ruido y no esperar entre las peticiones, conseguiremos un ancho de banda de hasta 64 Bytes por segundo.

5.4. HTTP Video Streaming

Partiendo del análisis anterior encontramos nuevas ideas y posibles lugares donde ocultar los mensajes. En este caso se ha pensado en un streaming de vídeo sobre el protocolo HTTP ya que la gran cantidad de tráfico que genera ayudaría a evitar que alguien detectase el covert-channel.

Para utilizar esta técnica podemos usar dos fuentes para los streams, la cámara web o un archivo de vídeo. Independientemente de el origen del vídeo, el servidor enviará al cliente mensajes HTTP con código de respuesta “206 Partial-content”, un ejemplo lo podemos ver a continuación:

```
HTTP/1.1 206 Partial Content
Date: Mon, 05 May 2015 00:36:57 GMT
Last-Modified: Wed, 15 Nov 2015 04:58:08 GMT
Server: Apache/2.0.52 (Red Hat)
Accept-Ranges: bytes
Content-Length: 1000
Content-Range: bytes 0-999/3980
Content-Type: video/mp4

{binary data}
```

Observando las cabeceras de la respuesta que envía el servidor para cada stream destacan dos lugares donde podríamos ocultar mensajes.

Uno de los lugares es la cabecera Server, la cual indica qué tipo de servidor web es y el sistema operativo que utiliza, de esta forma podríamos enviar un estego-mensaje en cada stream enviado al cliente.

Aunque parezca demasiado obvio el lugar y fácil de detectar, esta cabecera está siendo eliminada o alterada en muchos servidores web puesto que indicar qué tipo de servidor es y el sistema operativo que usa, es una fuga de información que puede aprovechar un atacante.

De esta forma podemos aprovechar el que muchos servidores oculten esta cabecera para pasar desapercibidos.

Otro lugar es la cabecera Last-Modified la cual indica la última vez que se modificó el recurso. Esta cabecera no es necesaria y podríamos ocultar un estego-mensaje en ella aunque sería fácilmente detectable dado el formato de fecha que suele llevar esta cabecera.

6. Prueba de concepto: StegoNet

Para poner en práctica lo estudiado hasta el momento, se va a desarrollar una aplicación capaz de trocear y esconder un mensaje dado en un protocolo de red y recuperarlos en destino.

Se analizarán las alternativas y detallarán las decisiones tomadas en cada apartado antes de llegar al desarrollo de la aplicación.

1.1. Estructura de la aplicación

6.1.1. Consideraciones previas

A continuación se exponen algunas consideraciones previas a utilizar la aplicación:

- Dado que la aplicación necesita crear sockets de red, algunos de ellos en puertos protegidos por el sistema (ejemplo ICMP) será necesario ejecutar la aplicación como superusuario o administrador.
- Ambos usuarios deben conocer la clave compartida para el cifrado para lo cual deberán utilizar otro medio.
- Ambos usuarios deberán usar la misma técnica esteganográfica para comunicarse.

6.1.2. Main Process

Este proceso es el que ejecuta la aplicación electron propiamente dicha, que después mediante una instrucción mostrará como interfaz en el Rendered Process la aplicación Angular.

Este proceso es el encargado de iniciar la aplicación y, mediante Inter Process Communication, escuchar los mensajes que se envíen desde Renderer Process.

Los mensajes que se recibirá este proceso son los siguientes:

Mensaje recibido	Acción
<i>createPingServer</i>	Crea un nuevo socket que utiliza el protocolo ICMP y espera a recibir paquetes. Por cada paquete que llegue se ejecutará la estego-función que comprueba si contiene algún mensaje oculto.
<i>destroyPingServer</i>	Destruye el socket.
<i>sendPingMessage</i>	Recibe un mensaje y lo pasa a la estego-función, la cual lo trocea en estego-mensajes y después envía por la red.
<i>createHttpServer</i>	Crea un servidor HTTP
<i>stopHTTPserver</i>	Para el servidor HTTP
<i>sendHTTPmessage</i>	Recibe un mensaje y lo pasa a la estego-función, la cual lo trocea en estego-mensajes y envía por la red.

Tabla 3.Mensajes recibidos por Main Process usando IPC..

Toda el peso de las funciones esteganográficas recae en este proceso, de forma que cuando reciba los mensajes anteriores realizará las tareas que en ellos se indican.

Cuando se crea un nuevo cliente o servidor, independientemente de la técnica, se instancia una nueva clase la cual implementa el método para enviar mensajes o crear el servidor.

6.1.3. Renderer Process

Este proceso es el que recibe la interacción del usuario.

Cuando se inicia la aplicación se mostrará una ventana en la que el usuario debe elegir qué técnica desea usar para enviar su mensaje.

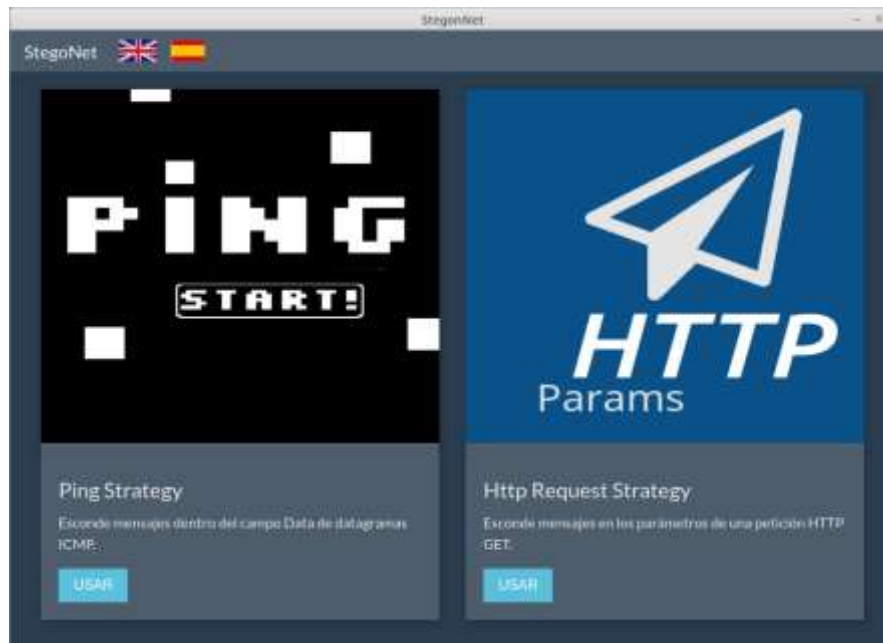


Ilustración 22. Primera ventana de la aplicación.

Cada técnica lleva asociadas unas restricciones específicas como puede ser el tamaño del mensaje a enviar. Esta Información aparece en la ficha de cada técnica.

Una vez elegida la técnica, la ventana aparecerá dividida en dos partes, Enviar y Recibir. En la parte de la izquierda encontraremos los ajustes necesarios para establecer una conexión con alguien que quiera enviarnos un mensaje.

Por otra parte, en el lado opuesto encontraremos los ajustes necesarios para enviar un mensaje. Todas las técnicas comparten dos campos, Contraseña con la que se cifrará el mensaje y Dirección IP del destino del mensaje.

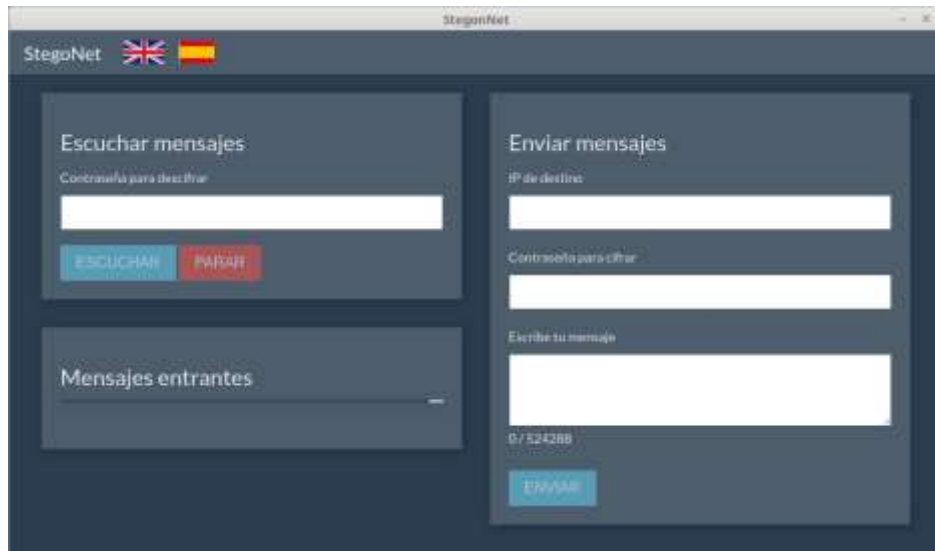


Ilustración 23. Ventana de envío y recepción de mensajes.

Cuando el usuario interactúa con la aplicación pidiendo enviar un mensaje o escuchar por mensajes de otros, el proceso `Renderer` se comunica con el proceso principal el cual contesta utilizando IPC cuando tiene un nuevo mensaje:

Mensaje recibido	Acción
<i>messageReceived</i>	Recibe un objeto mensaje: <pre> { id_message: id_message, seq: seq, text: message } </pre>
<i>messageChunked</i>	Contiene un objeto indicando el número de trozos en los que se ha dividido el mensaje. Esto será utilizado para mostrar al usuario el progreso de envío.
<i>chunkSend</i>	Este mensaje llegará por cada estego-mensaje enviado y se utilizará para incrementar el progreso de envío mostrado al usuario.
<i>targetUnreachable</i>	No se ha podido conectar con la IP especificada.

Tabla 4. Mensajes recibidos por `Renderer Process` usando IPC.

El proceso `Renderer` irá mostrando los trozos de mensaje conforme llegan pero siempre ordenados por el número de secuencia, es decir, si primero llega el trozo de mensaje con número de secuencia 4 lo mostrará y cuando llegue un mensaje con número de secuencia inferior, ordenará el mensaje en función de ésta.

6.1.4. Stego-chunk

Esta clase es la encargada de trocear el mensaje para poder ocultarlo en el estego-medio. Consta de dos funciones principales “*paquetize*” y “*unpack*”.

Packetize es la función encargada de crear la estructura del estego-mensaje y unpack realiza el proceso inverso, comprobando si el objeto que se le pasa como parámetro contiene un estego-mensaje o no.

6.1.5. Construcción del estego-mensaje

Cuando se instancia un objeto de la clase StegoChunk es necesario pasarle como parámetro la contraseña que usará para cifrar/descifrar un mensaje.

Para que los usuarios puedan utilizar contraseñas alfanuméricas de tamaño variable que les sean fáciles de recordar, es necesario transformar esa contraseña para que tenga un tamaño de 256 bits antes de cifrar, para ello se genera un hash de tal longitud utilizando el algoritmo SHA 3.

En las técnicas implementadas en la aplicación el mensaje utilizará codificación ASCII de 8 bits lo que significa que para representar cada carácter será necesario 1 Byte. Utilizar ASCII limita el rango de caracteres que podemos enviar en el mensaje como tildes o la letra Ñ. El motivo de dicha decisión radica en el hecho de que emplear una codificación como UTF-8 que sí nos permitiría utilizar estos caracteres, implicaría utilizar 16 bits por carácter, lo que reduciría el ancho de banda del estego-medio a la mitad.

1. El primer paso es convertir el texto recibido a codificación ASCII utilizando el módulo de Node transliteration.
2. Dividimos el mensaje en trozos del tamaño del campo mensaje de nuestro estego-mensaje, en este caso 8 Bytes.
3. Calculamos el identificador de mensaje mediante un hash SHA3-256 del mensaje completo y una cadena de texto aleatoria, de la cadena de texto obtenida cogemos los dos primeros bytes. El motivo de concatenar una cadena

aleatoria al mensaje para calcular el hash distinguir entre dos mensajes que tengan igual contenido. Este identificador lo compartirán todos los estego-mensajes del mensaje actual.

4. Dado que vamos a trabajar con datos de un tamaño fijado en Bytes, usaremos el objeto Buffer de Node.js para crear las distintas partes del estego-mensaje.
5. Por cada trozo de mensaje:
 - i. Incrementamos un contador utilizado como valor del campo secuencia y creamos un Buffer de 2 Bytes en el que escribimos el valor del contador.
 - ii. Generamos un hash SHA3-256 del trozo del mensaje y guardamos los 4 primeros bytes en un objeto Buffer de dicho tamaño. Este será el campo checksum del estego-mensaje.
 - iii. Concatenamos los dos buffers anteriores junto con el identificador del mensaje y el trozo del mensaje en un Buffer.
 - iv. Convertimos el buffer anterior a hexadecimal y ciframos usando AES

6.1.6. Obtención del mensaje

Este es el proceso inverso al apartado anterior en el que se descifra y recupera cada una de las partes del estego-mensaje.

Cabe destacar en esta parte la comprobación del campo checksum ya que el contenido de este campo es el que indicará si lo que hemos recibido es un estego-mensaje o cualquier otro tipo de datos.

Después de descifrar los datos recibidos se obtienen los bytes que corresponderían a cada uno de los campos del estego-mensaje, **sin tener la certeza aún de que realmente sea un estego-mensaje.**

Calculamos el hash SHA3-256 del campo mensaje y comparamos los 4 primeros bytes con el campo checksum para determinar si se trata de un estego-mensaje o no.

6.2. ICMP

Para comenzar a desarrollar la prueba de concepto se ha optado por utilizar el protocolo ICMP en una primera fase hasta comprobar que la forma de estructurar la información a ocultar funciona correctamente.

El motivo es que el protocolo ICMP permite añadir datos arbitrarios en el campo Data a partir del bit 80 del datagrama ICMP, lo cual va a facilitar tanto el desarrollo como las pruebas que se realicen con herramientas de análisis de tráfico de red como Wireshark.

6.2.1. Emisor

Para construir el emisor en esta técnica se ha usado la librería net-ping escrita en JavaScript para Node.js.

El objetivo de esta librería es realizar un ping a una IP dada. Para poder esconder en el campo DATA del paquete ICMP el estego-mensaje ha sido necesario modificar dicha librería para sustituir los datos que incluía en dicho campo ya que por defecto enviaba una cadena binaria de ceros.

6.2.2. Receptor

Dado que estamos escuchando paquetes enviados utilizando el protocolo ICMP, los datos que recibe la aplicación contienen las cabeceras IP e ICMP.

Para poder comprobar si un paquete contiene un estego-mensaje es necesario eliminar dichas cabeceras antes de pasar el Buffer de datos a la clase stegoChunk.

IP Datagram				
	Bits 0–7	Bits 8–15	Bits 16–23	Bits 24–31
IP Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		flags and offset	
	Time To Live (TTL)	Protocol	Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (optional)	Payload Data			

Ilustración 24. Datagrama IP.

La cabecera IP tiene un tamaño de 20 Bytes y la cabecera ICMP contiene 8 Bytes por lo tanto al Buffer de datos recibido de la red debemos pasarle un offset de 28 Bytes para recuperar el contenido del campo Data o Payload que es el lugar donde ocultamos el estego-mensaje en esta técnica.

6.2.3. Ejemplo

Para demostrar el funcionamiento de esta técnica a continuación se detallan los pasos necesarios y el proceso de transformación del mensaje, así como una captura de tráfico de red.

El primer paso es elegir la técnica Ping en la ventana principal de la aplicación y completar los datos de servidor o cliente. En este caso se ha creado un servidor y un cliente en la dirección 127.0.0.1.

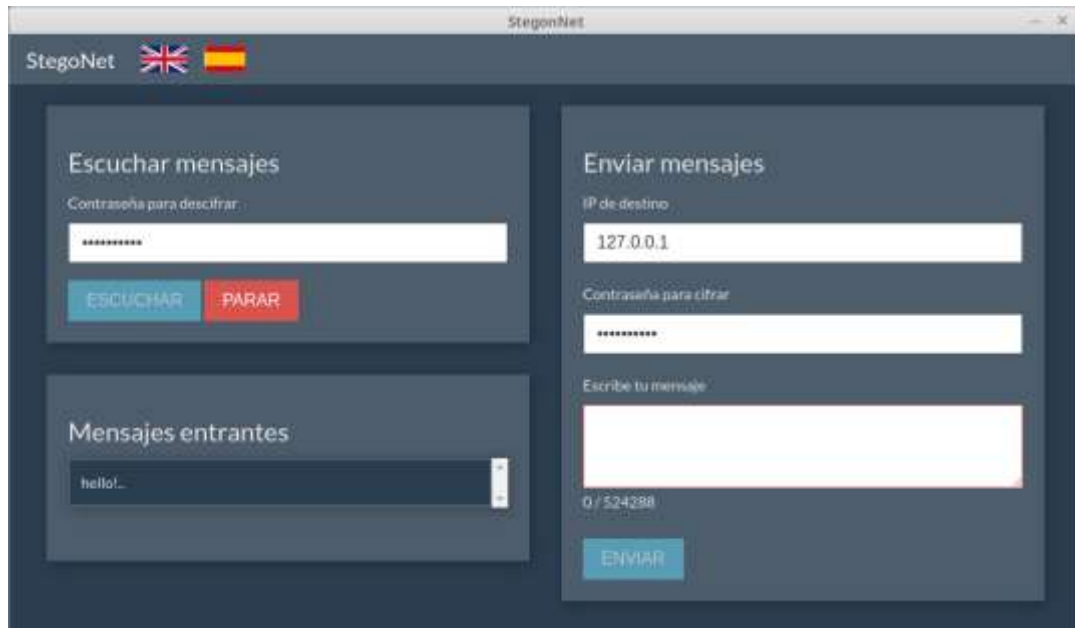


Ilustración 25. Ejemplo de envío de mensaje usando ICMP.

Una vez enviado el mensaje, es el Main Process el encargado de crear una instancia de la clase StegoChunk, la cual dividirá el mensaje en trozos para enviarlo por la red.

Para poder trabajar a nivel de bytes, se utiliza el tipo de dato Buffer de Node.js.

1. La clase StegoChunk ejecuta el método `packetize` recibiendo como parámetro el mensaje "hello!".
2. Se divide el mensaje en trozos. En este caso como el tamaño del mensaje es inferior a 8 Bytes se completa con "." quedando "hello!..".
3. Se genera el campo `id_mensaje` a partir de los 2 primeros bytes de un hash generado con SHA3 de 256 bits.
4. Por cada trozo del mensaje obtenidos en el paso 2:
 - i. Obtenemos la secuencia incrementando una variable y creando un buffer de 2 bytes en el cual se escribe el valor.
 - ii. Obtenemos el hash sha3-256 del trozo de mensaje. **"Dee6f2523c8a6c0504078fd661d324dd1ef163048ed6965fbb3149e74d0edb41"**
 - iii. Guardamos en un buffer los 4 primeros bytes del hash anterior. **<Buffer 64 65 65 36>**

- iv. Se concatenan todos los campos obteniendo un Buffer de 16 Bytes.
<Buffer 35 36 00 00 68 65 6c 6c 6f 21 2e 2e 64 65 65 36>
 - v. Se convierte el buffer anterior a Hexadecimal y se cifra.
“65063fe4770e7fd43ced0a76f4ded309”.
 - vi. En este momento ya tenemos el estego-mensaje cifrado. Se guarda en un array con todos los demás estego-mensajes.
5. Cuando el paso anterior termina tenemos un array con todos los estego-mensajes, dependiendo de la técnica empleada será necesario convertir cada mensaje a Buffer para poder enviarlo por un socket de red.

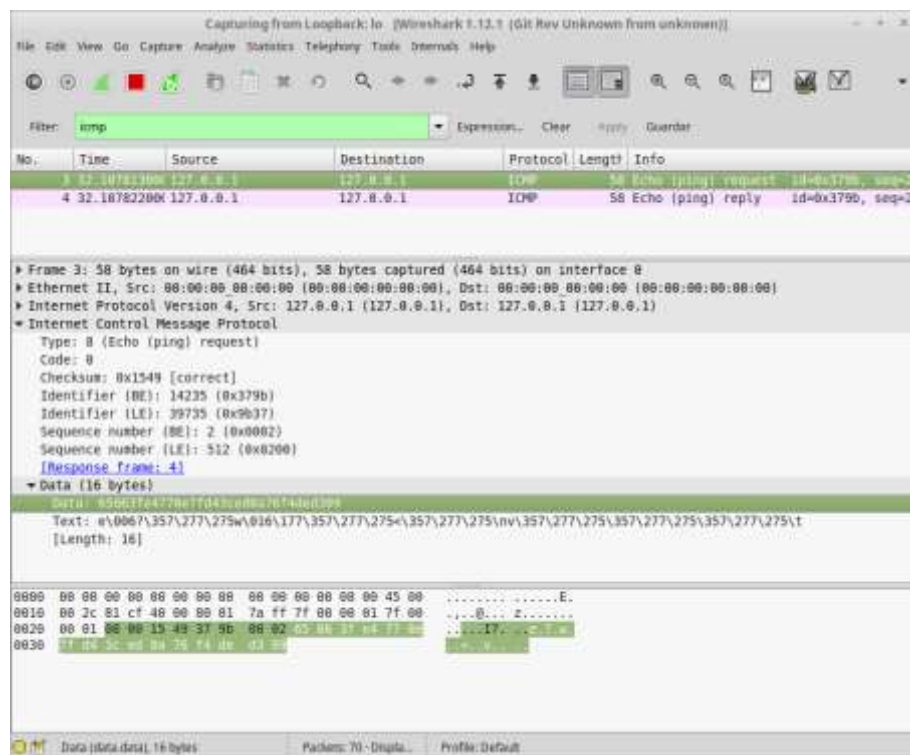


Ilustración 26. Tráfico capturado con Wireshark del mensaje del ejemplo.

6.3. HTTP Requests

Esta técnica se basa en el análisis realizado sobre el protocolo HTTP y cómo podríamos insertar estego-mensajes en los parámetros de una URL al hacer una petición GET a un servidor web.

Para esta prueba de concepto, se utilizará el protocolo HTTP en lugar de HTTPS para poder analizar el tráfico en texto plano de igual forma que podría hacerlo un tercero que desee espiar la comunicación.

Utilizar el protocolo HTTPS no es una mala idea ya que cada día es más común dicho tráfico y nuestros estego-mensajes pasarían más inadvertidos, además de añadir una segunda barrera de cifrado.

6.3.1. Emisor

El servidor Node.js que se ejecuta en segundo plano junto con la aplicación será el encargado de realizar las HTTP requests necesarias, una por cada estego-mensaje generado al dividir el mensaje original.

Para simular un tráfico web manual de un usuario, debemos evitar enviar seguidas muchas peticiones GET distintas ya que esto podría alertar a quien monitorice la red y descubrir el cover-channel, por lo tanto el emisor esperará un tiempo aleatorio de entre 3 y 15 segundos entre cada petición que oculte un estego-mensaje.

No tomaremos en cuenta la respuesta del servidor, lo único que nos interesa saber es si se ha producido un error al tratar de hacer la petición GET, si ese fuera el caso informaremos al proceso Renderer para que muestre un error al usuario.

6.3.2. Receptor

En este otro lado hacemos uso del framework Express.js para crear un servidor web que escuche en el puerto 80 y en la URL http://DIRECCION_IP/resource/:resource donde **:resource** es el valor del parámetro en el que vamos a insertar los estego-mensajes.

La respuesta para alguien que haga una petición GET a dicha URL siempre será la misma *“Hello World!”* aunque aquí es donde se podría profundizar más para engañar a un tercero que se encuentre monitorizando el tráfico de red por ejemplo montando un completo servicio web que sirva una web con contenido dinámico.

6.3.3. Ejemplo

Para mostrar el funcionamiento de esta técnica se ha preparado un entorno de red local con dos equipos conectados a la misma red Wi-Fi con direcciones IP:

- Equipo 1: 192.168.1.129
- Equipo 2: 192.168.1.133

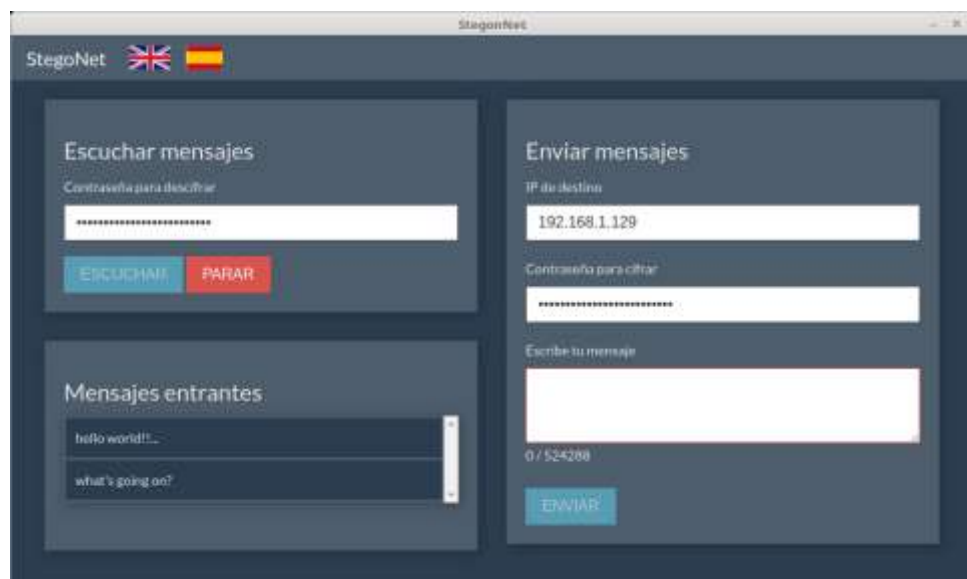


Ilustración 27. Ejemplo de mensajes recibidos usando HTTP Request.

Mientras se realiza la prueba se realiza una captura de tráfico con la herramienta Wireshark en la cual podemos ver un ejemplo de tráfico de otras webs comparado con el tráfico de la aplicación desarrollada.

The image shows a Wireshark packet capture of an HTTP GET request. The packet list on the left shows packets 8279 through 8686. Packet 8302 is selected, showing details for Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The HTTP request is for 'photo.jpg' from 'http://www.elladodelmal.com/2017/08/nuevo-libo-hacking-web-applications.html?...'.

No.	Time	Source	Destination	Protocol	Length	Info
8279	123.41086900	192.168.1.129	216.58.210.211	HTTP	609	GET /2017/08/nuevo-libo-hacking-web-applications.html?action=ba
8284	123.42959900	192.168.1.129	37.187.86.201	HTTP	1894	GET /subscribe-widget/index.html?id=c2279732-2fad-4653-b130-b67
8294	123.50527200	37.187.86.201	192.168.1.129	HTTP	976	HTTP/1.1 200 OK (text/html)
8298	123.51925200	192.168.1.129	216.58.210.179	HTTP	640	GET /2017/08/nuevo-libo-hacking-web-applications.html?action=ge
8302	123.53400000	192.168.1.129	216.58.210.161	HTTP	307	GET /.../ID: F0T000/AAAAAAAAAA/AAAAAAAAAA/DGagLrUMmg/s35-c/photo.jpg HTTP/1.1
8306	123.50067200	216.58.210.161	192.168.1.129	HTTP	557	HTTP/1.1 200 OK (JPEG JFIF image)
8312	123.65266500	216.58.210.211	192.168.1.129	HTTP	767	HTTP/1.1 200 OK (text/javascript)
8316	123.65722100	216.58.210.211	192.168.1.129	HTTP	788	HTTP/1.1 200 OK (text/javascript)
8329	123.66424000	216.58.210.211	192.168.1.129	HTTP	374	HTTP/1.1 200 OK (text/javascript)
8340	123.78502200	216.58.210.179	192.168.1.129	HTTP	682	HTTP/1.1 200 OK (text/javascript)
8504	136.51602400	192.168.1.133	192.168.1.129	HTTP	165	GET /resource/7161b498f6b92b54d55f7ac05ef4444c HTTP/1.1
8507	136.51900500	192.168.1.133	192.168.1.129	HTTP	165	GET /resource/1b0f4be05c73b9b172bcb8760094db37 HTTP/1.1
8510	136.52790500	192.168.1.129	192.168.1.133	HTTP	277	HTTP/1.1 200 OK (text/html)
8514	136.53114400	192.168.1.129	192.168.1.133	HTTP	277	HTTP/1.1 200 OK (text/html)
8649	179.30614000	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1
8672	180.30735400	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1
8680	181.30760400	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1
8686	182.30805500	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1

Frame 8302: 607 bytes on wire (5496 bits), 607 bytes captured (5496 bits) on interface 0
 Ethernet II, Src: 2c:56:dc:47:1d:a6 (2c:56:dc:47:1d:a6), Dst: Contrend_e6:fa:20 (38:72:c0:e6:fa:20)
 Internet Protocol Version 4, Src: 192.168.1.129 (192.168.1.129), Dst: 216.58.210.161 (216.58.210.161)
 Transmission Control Protocol, Src Port: 59815 (59815), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 621
 Hypertext Transfer Protocol
 GET /.../ID: F0T000/AAAAAAAAAA/AAAAAAAAAA/DGagLrUMmg/s35-c/photo.jpg HTTP/1.1
 Host: lth-googleusercontent.com/r/n
 Connection: keep-alive/r/n
 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/60.0.3112.78 Chrome/60.0.3112.78 Safari/537.36
 Accept: image/webp,image/apng,image/*,*/*;q=0.8/r/n
 Referer: http://www.elladodelmal.com/2017/08/nuevo-libo-hacking-web-applications.html/r/n
 Accept-Encoding: gzip, deflate/r/n
 Accept-Language: es/r/n
 Cookie: NID=110=1LKCOJm28uvdnDj4GP9G50ehvFQBPgy-WyueqB1jArydRa0_xRus5vgKxfZMmPPMPyELbLbX723FOX-50Yuz9c3b09g8ooFsd-fw1fB4CnspHzydQX3IAquD...

Ilustración 28. Captura de tráfico con Wireshark de tráfico HTTP.

En esta primera captura se destaca el tráfico generado al conectarse a la web <http://elladodelmal.com> donde la URL completa que se solicita al servidor es la siguiente:

```
http://www.elladodelmal.com/2017/08/nuevo-libo-hacking-web-applications.html?action=backlinks&widgetId=Blog1&widgetType=Blog&responseType=js&postId=4774165360224760541&xssi_token=AOuZoY4tI3YUssrbMVzTerL1QKYJBMAOTw%3A1503479901333
```

Si comparamos esta captura con el tráfico que contiene estego-mensajes, podemos comprobar como el contenido del parámetro podría pasar perfectamente desapercibido:

The image shows a Wireshark packet capture of an HTTP GET request. The packet list on the left shows packets 8340 through 8686. Packet 8504 is selected, showing details for Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The HTTP request is for a resource from 'http://192.168.1.129/resource/7161b498f6b92b54d55f7ac05ef4444c'.

No.	Time	Source	Destination	Protocol	Length	Info
8340	123.78502200	216.58.210.179	192.168.1.129	HTTP	682	HTTP/1.1 200 OK (text/javascript)
8504	136.51602400	192.168.1.133	192.168.1.129	HTTP	165	GET /resource/7161b498f6b92b54d55f7ac05ef4444c HTTP/1.1
8507	136.51900500	192.168.1.133	192.168.1.129	HTTP	165	GET /resource/1b0f4be05c73b9b172bcb8760094db37 HTTP/1.1
8510	136.52790500	192.168.1.129	192.168.1.133	HTTP	277	HTTP/1.1 200 OK (text/html)
8514	136.53114400	192.168.1.129	192.168.1.133	HTTP	277	HTTP/1.1 200 OK (text/html)
8649	179.30614000	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1
8672	180.30735400	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1
8680	181.30760400	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1
8686	182.30805500	192.168.1.129	239.255.255.250	SSDP	208	M-SEARCH * HTTP/1.1

Frame 8504: 165 bytes on wire (1320 bits), 165 bytes captured (1320 bits) on interface 0
 Ethernet II, Src: IntelCor 38:46:42 (78:92:9c:38:46:42), Dst: 2c:56:dc:47:1d:a6 (2c:56:dc:47:1d:a6)
 Internet Protocol Version 4, Src: 192.168.1.133 (192.168.1.133), Dst: 192.168.1.129 (192.168.1.129)
 Transmission Control Protocol, Src Port: 41220 (41220), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 99
 Hypertext Transfer Protocol
 GET /resource/7161b498f6b92b54d55f7ac05ef4444c HTTP/1.1
 Host: 192.168.1.129/r/n
 Connection: close/r/n
 /r/n
 [Full request URI: http://192.168.1.129/resource/7161b498f6b92b54d55f7ac05ef4444c]
 [HTTP request 1/1]
 [Response in frame: 8510]

Ilustración 29. Captura de tráfico con WireShark de un estego-mensaje HTTP.

Lo único que restaría añadir es el resto de cabeceras y un cuerpo de respuesta similar al de una página web normal para poder así engañar a quien pueda ver el tráfico de red y que no se descubra el covert-channel.

6.4. Despliegue de la aplicación

Una vez todo terminado y preparado para el despliegue existen varias alternativas para el despliegue de la aplicación a distintos sistemas operativos.

Podemos generar instalables para los distintos sistemas operativos utilizando para ello módulos como electron-winstaller para windows o electron-installer-debian que genera un archivo DEB. Sin embargo, generar instaladores presenta algunas desventajas como que para algunas plataformas como Windows debe estar firmado.

Como alternativa rápida podemos generar un paquete para cada sistema operativo, portable que no necesita de instalación y que se puede distribuir comprimido en un fichero ZIP.

Antes de nada, para generar un paquete para cada sistema operativo será necesario generarlo en dicho sistema operativo. Esto es así debido a que en la aplicación se usan módulos nativos de Node escritos en C++ y es necesario compilar dichos módulos.

Para ello se clonará el repositorio del proyecto en el sistema operativo del cual se quiere obtener una versión y se ejecutará en una terminal, dentro de la carpeta del proyecto:

```
npm install
```

Esto descargará todas las dependencias del proyecto, incluidos los módulos nativos de Node.js y se compilarán automáticamente durante el proceso de instalación.

Una vez descargadas las dependencias hay que comilar el código Typescript a JavaScript, para ello podemos ejecutar el siguiente script incluido en el fichero package.json, el cual compilará tanto los ficheros Typescript de Angular como los de Electron en la carpeta dist:

```
$ npm run build.prod
```

Una vez hecho esto ya podemos generar el paquete que se distribuirá. Utilizando las instrucciones de la web de Electron o automatizando el proceso con electron-packager.

Para usar la segunda opción se han incluido varios scripts en package.json que automatizan el proceso:

```
$ npm run bundle.linux  
$ npm run bundle.windows  
$ npm run bundle.mac
```

Esto generará una carpeta bajo el directorio bundles, dicha carpeta contiene un archivo ejecutable llamado StegoNet el cual abrirá la aplicación.

6.5. Manual de Usuario

La interfaz que se ha utilizado para la aplicación es muy simple e intuitiva utilizando componentes y estilos como Bootstrap que muchos usuarios ya están acostumbrados a ver en páginas web, por lo que su uso no debería implicar un mayor esfuerzo.

En el apartado anterior se explicó cómo se puede generar ejecutables e instalables para distintos sistemas operativos. En este apartado se explicará el uso de la aplicación suponiendo que ya esté instalada o se utilice la versión ejecutable.

6.5.1. Ventana principal

En esta ventana podemos elegir la estrategia a usar pulsando sobre el botón azul de cada estrategia.

En la parte superior de la ventana encontramos dos banderas de Idioma, pulsando sobre alguna de ellas se cambiará inmediatamente los textos empleados en la aplicación.

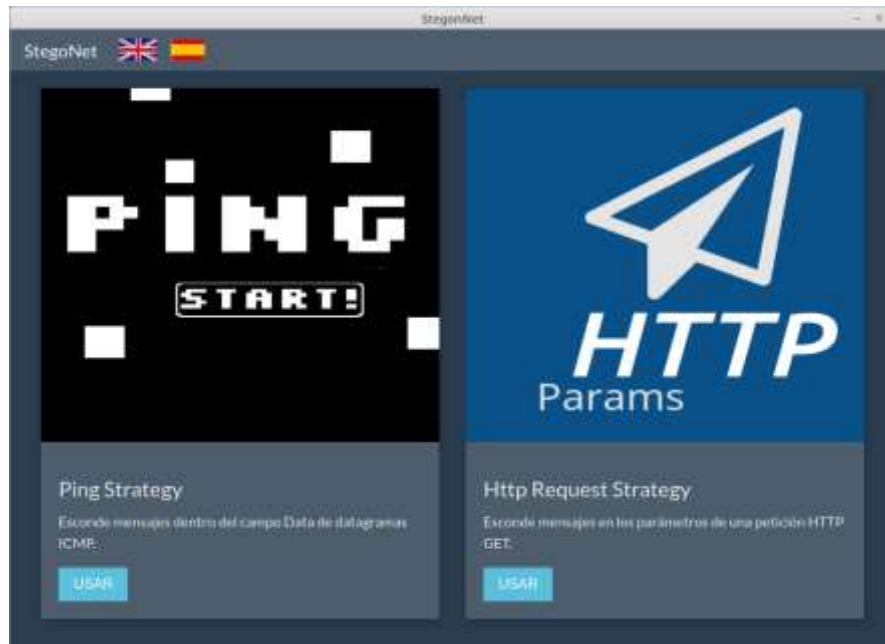


Ilustración 22. Primera ventana de la aplicación.

6.5.2. Ventana de mensajería

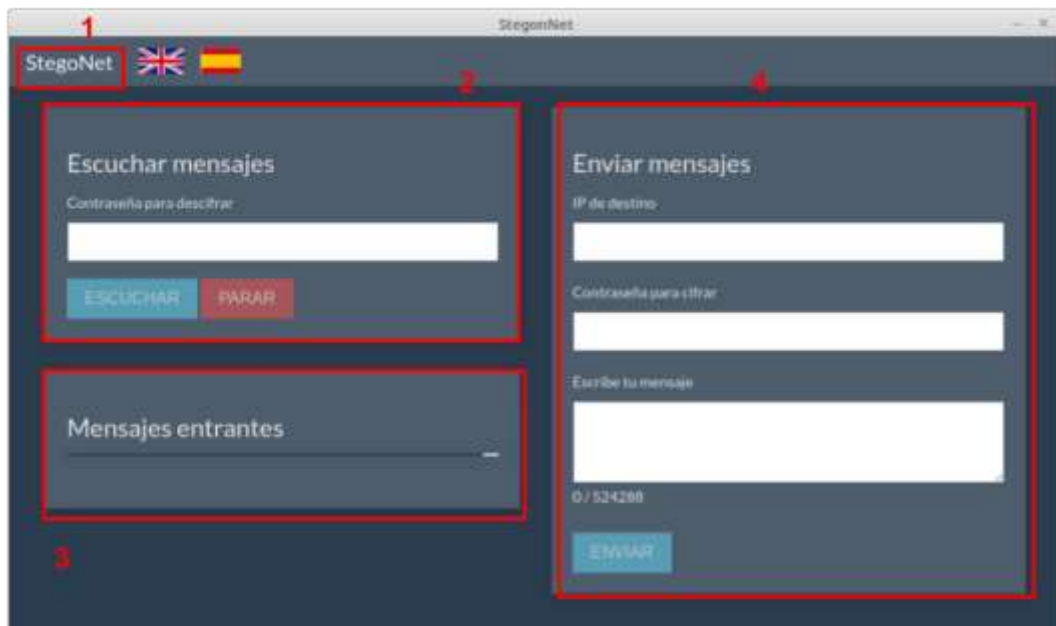


Ilustración 30. Ventana de mensajería con zonas marcadas.

En la ilustración anterior se han resaltado 4 zonas las cuales se explican a continuación:

1. Nombre de la aplicación, si pulsamos sobre él nos redirigirá a la ventana principal.

2. **Escuchar mensajes.** Por defecto los dos botones estarán desactivados. El botón escuchar se activará cuando se introduzca una contraseña, al pulsar sobre él se desactivará y activará el botón parar, el cual hará que deje de recibir mensajes.
3. **Mensajes entrantes.** En este apartado se mostrarán los mensajes a medida que lleguen. Si los esteo-mensajes llegan desordenados estos se irán ordenando en tiempo real a medida que llegan.
4. **Enviar mensajes.** Rellenaremos los 3 campos para que se active el botón “enviar”. Cuando pulsemos en enviar se mostrará una barra de progreso hasta que se envíe el mensaje.

6.5.3. Ejemplo práctico

Para mostrar el uso de la aplicación se va a proceder al envío mediante la técnica “HTTP Request” el siguiente texto:

“En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda.”

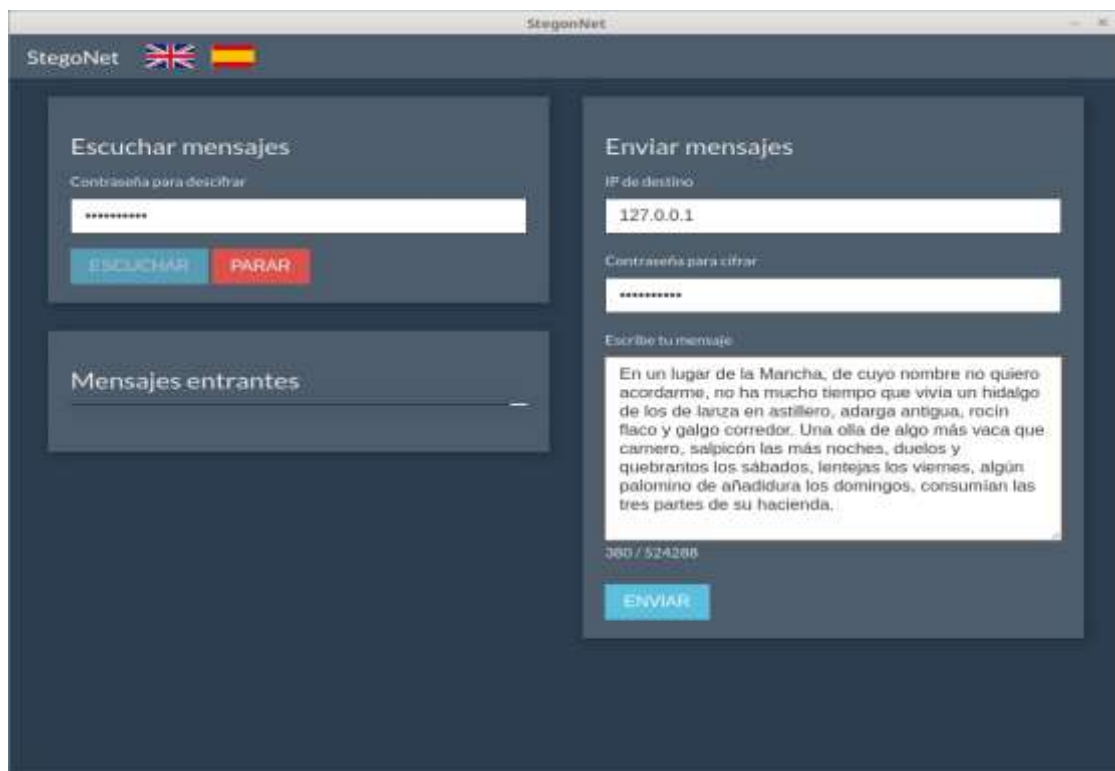
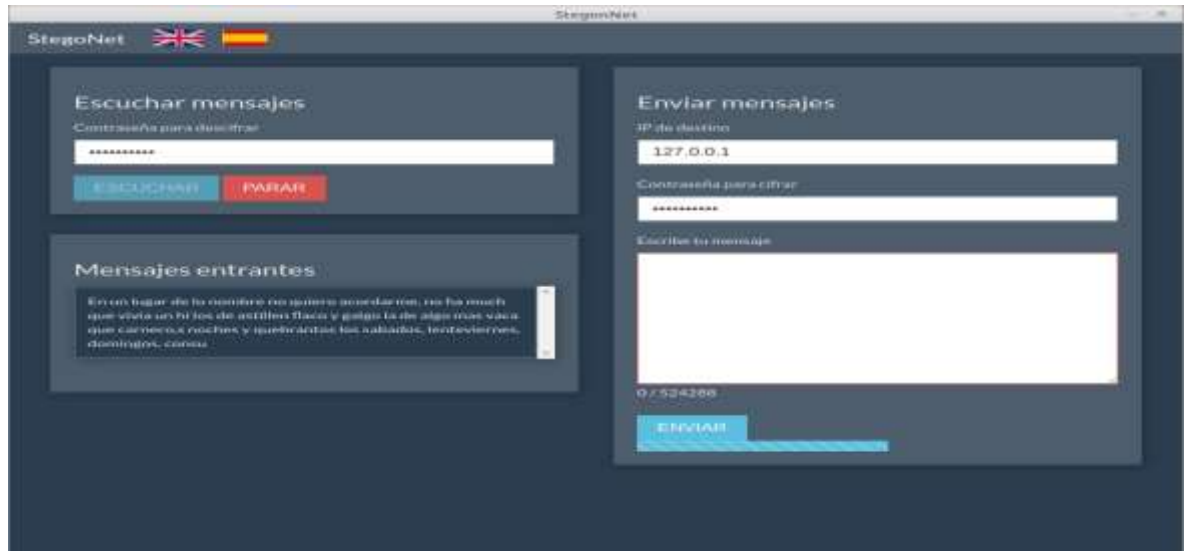


Ilustración 31. Ejemplo de envío de mensaje usando HTTP Request.

Para este ejemplo tanto receptor como emisor son el mismo equipo pero el funcionamiento sería el mismo para cualquier otro destinatario con la diferencia de que el mensaje que se envía no aparecería en la sección de “mensajes entrantes” del emisor:

**Ilustración 32. Progreso del mensaje enviado. Los estego-mensajes llegan desordenados.**

La ilustración anterior ha sido recortada y ampliada para que se pueda apreciar cómo a pesar de que los estego-mensajes lleguen al destino desordenados, estos se van ordenando y mostrando a medida que llegan.

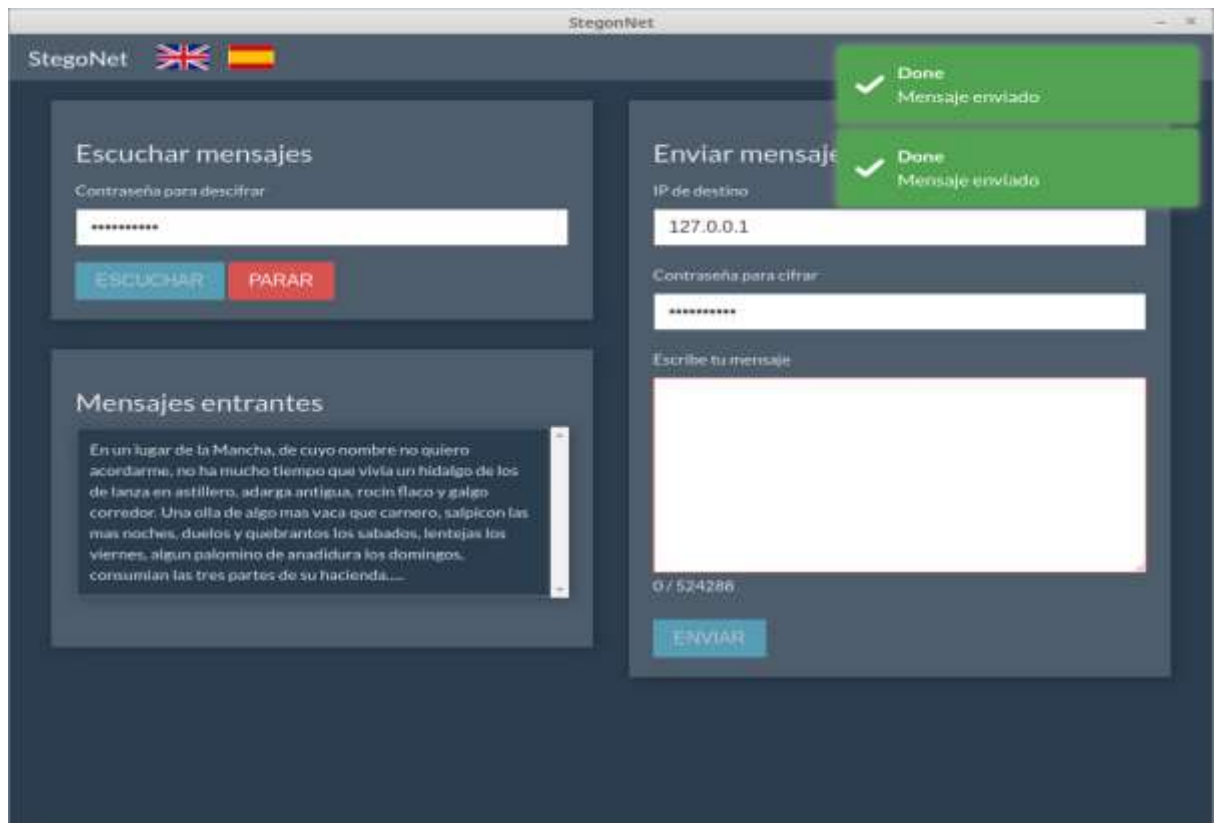


Ilustración 33. Mensaje enviado.

Finalmente el mensaje completo llega mostrándose un mensaje de aviso en la parte superior derecha y el destinatario puede leerlo perfectamente.

7. Resultados

Los resultados obtenidos son muy prometedores ya que se ha conseguido diseñar un estego-sistema que se puede aplicar en múltiples escenarios.

Los protocolos analizados son los más comunes en el tráfico de red ya que la navegación web junto con servicios de VoIP es el tipo de uso más común que un usuario puede llevar a cabo en su día a día.

7.1. Limitaciones

La principal limitación la encontramos en el ancho de banda que dependerá del protocolo elegido para transportar los estego-mensajes. Aunque todos ellos deben permitir insertar el estego-mensaje de 16 Bytes, algunos requerirán pausas o intervalos de tiempo entre cada estego-mensaje enviado para no descubrir el covert-channel y otros como en HTTP Request podremos incluir más de un estego-mensaje por cada request HTTP.

Con respecto a la pérdida de paquetes, si se utiliza el protocolo UDP, puesto que es un protocolo no orientado a conexión, es posible que se produzcan pérdidas.

Ante esta situación, dada la estructura propuesta de los estego-mensajes podemos conocer cuando se producen estas pérdidas ya que cada estego-mensaje contiene un identificador y un número de secuencia.

El utilizar un algoritmo de cifrado ECB presenta algunas desventajas. El hecho de cifrar los bloques por separado implica que cuando se cifre un bloque con cierto valor, siempre se obtendrá el mismo resultado. Esto hace posible los ataques de diccionario.

Por último, la aplicación ha sido desarrollada para utilizar el protocolo IPv4 y si se utiliza a través de internet en lugar de una red local, el destinatario de los mensajes deberá tener activada la redirección de puertos en su router.

Uno de los objetivos era que la aplicación pudiese usar el protocolo IPv6 sin embargo por falta de tiempo y dado que su uso en internet aún no está extendido a todos los usuarios se ha dedicado más tiempo de desarrollo a otras características como la presentación de mensajes en tiempo real.

7.2. Detección y contramedidas

En este apartado vamos a analizar los posibles vectores de ataque al estego-sistema para descubrirlo e intentar recuperar la información oculta.

Para ello se considerará una comunicación oculta y un guardián o estego-analista el cual dispone de las siguientes capacidades:

1. **Ataque pasivo.** El guardián puede analizar la información intercambiada, dejando que la comunicación tenga lugar si no advierte información oculta.
2. **Ataque activo.** El guardián podría modificar la información, accidentalmente o a propósito, e incluso dañarla significativamente.
3. **Ataque malicioso.** El guardián puede modificar a su antojo la información ocultada en una cubierta con la intención de provocar una acción determinada en las entidades receptoras de dicha información.

Estos tres tipos de ataques están relacionados y cada uno de ellos implica primero haber realizado el ataque anterior.

Un ataque pasivo puede ser conseguido utilizando un Sistema de Detección de Intrusos, también conocido por sus siglas en inglés IDS (Intrusion Detection System), con complejas reglas que definen límites al tráfico de red o usar detección de anomalías lo que constantemente monitorizará el tráfico de red comparándolo con muestras de tráfico de red previas. La segunda solución presenta un mayor riesgo de falsos positivos para el estego-analista ya que comparar todo el tráfico de red en protocolos que generan gran cantidad de datos como RTP es una tarea compleja.

Las medidas tomadas para evitar un **ataque pasivo**, o lo que es igual, pasar inadvertidos, han sido utilizar protocolos de red que generen gran cantidad de tráfico o en su defecto, protocolos con un uso extendido y que su presencia pueda pasar inadvertida, además de introducir tiempos pseudo-aleatorios entre el envío de cada mensaje para simular un tráfico de red normal o “manual”.

Los ataques activos o maliciosos los evitamos mediante el cifrado de los estego-mensajes con un algoritmo de cifrado seguro.

Si un atacante consigue descubrir el covert-channel, seguiría sin poder tener acceso a la conversación dado que cada estego-mensaje está cifrado con AES-256.

Un ataque posible sería si un atacante conociese la contraseña que se usó para cifrar los mensajes, de este modo podría no sólo acceder a la conversación si no también modificarla.

El otro y último posible ataque sería que el guardián descartase paquetes aleatorios de la comunicación lo cual podría hacer perder parte del mensaje. Este último podría solucionarse enviando más de una copia de cada estego-mensaje, ya que implementar un mecanismo de solicitud de paquetes perdidos podría destapar el covert-channel.

8. Conclusiones

Teniendo en cuenta los objetivos propuestos para este proyecto, cabe destacar que los principales se han completado obteniendo como resultado el análisis de los protocolos de red descrito en este documento y una prueba de concepto en la que se utiliza el estego-sistema diseñado.

Los objetivos secundarios han sido cubiertos parcialmente, dado que el estego-sistema genera bloques de 16 Bytes y esto hará que no puedan ocultarse en cualquier protocolo.

Atendiendo a la pirámide de diseño de un estego-sistema, en la cual se valoran las tres características básicas que un estego-sistema debe tener, a continuación se explican las decisiones tomadas en cada aspecto:

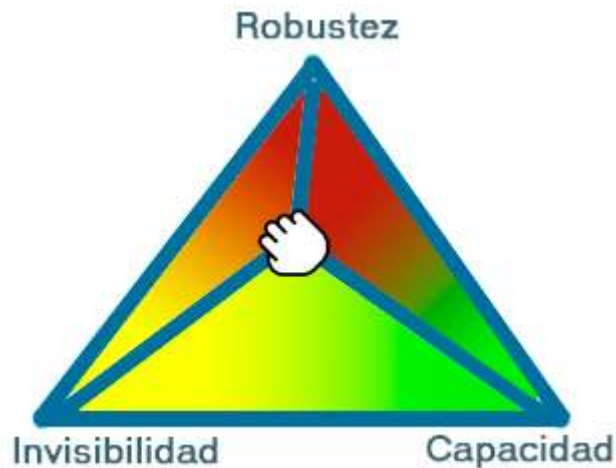


Ilustración 34. Balance final del estego-sistema.

En la ilustración anterior se pretende ilustrar el compromiso alcanzado entre las tres características, dejando ver como la Robustez es la que mayor peso tiene.

Para proporcionar **robustez** al estego-sistema se ha tenido en cuenta durante el diseño las siguientes cuestiones:

- Posibilidad de pérdida de mensajes y cómo afectaría a la comunicación, incluyendo un campo de secuencia en el mensaje para identificar cada porción del mensaje original y usar un cifrado por bloques para que no se pierda el mensaje original entero en caso de que alguna porción se pierda.
- Se ha diseñado un mecanismo para descartar mensajes que no contengan información oculta utilizando el campo checksum en el que se guardan los 4 primeros bytes del hash de la porción de mensaje.
- Añadimos un identificador de mensaje para conocer a qué mensaje pertenece cada porción o estego-mensaje y así poder mantener varias conversaciones simultáneamente.
- Para asegurar la integridad del mensaje, cada estego-mensaje está cifrado con AES y clave de 256 bits

La **cantidad** de información contenida en cada estego-mensaje es de 8 Bytes lo cual nos permitirá encapsular 8 caracteres ASCII por mensaje.

El ancho de banda proporcionado por los covert-channels no es muy grande y serán necesarios varios segundos de espera para poder transmitir un mensaje. Parte de culpa de este problema lo tiene el diseño del estego-sistema ya que obliga a reducir los posibles protocolos utilizados a aquellos que permitan enviar 16 Bytes por mensaje.

La **invisibilidad** dependerá del estego-medio, ya que en el presente trabajo se proponen varios, en cada protocolo o técnica utilizados la invisibilidad variará atendiendo a la cantidad de tráfico de red generado por dicho protocolo, el lugar de ocultación como cabeceras o payload y lo extendido que esté el protocolo de red elegido.

Dado que los protocolos propuestos son punto a punto, un problema latente es que si se descubre el covert-channel, aunque la información sea indescifrable por un tercero, sí que se puede identificar a los participantes por sus direcciones IP.

Es cierto, al igual que sucede con otras tecnologías, que la Esteganografía, entre sus múltiples usos, podría facilitar la comunicación de grupos para realizar tareas delictivas. Quizás por este motivo en la primera década del siglo XXI se ha dedicado un esfuerzo enorme en avanzar en algoritmos y herramientas de estegoanálisis. En la actualidad, no se conocen estudios públicos significativos de estegoanálisis masivo, quizás por la complejidad de realizar estos.

8.1. Dificultades encontradas

Con respecto a la planificación temporal, la segunda parte se ejecutaría durante el mes de Febrero y Marzo de 2017 pasados los exámenes pero se estimó mal la duración de esta parte y el aprendizaje necesario para empezar a manejar el framework de desarrollo. Debido a esto, esta parte finalizó en Junio de 2017.

La tercera parte se realizaría durante los meses de Abril y Mayo de 2017, aunque debido al retraso acumulado por la parte anterior y la merma en las horas diarias que podía dedicar, finalmente esta parte comenzó en el mes de Junio de 2017 terminando en Agosto del mismo año.

Desde el punto de vista del desarrollo de la aplicación se han encontrado algunas dificultades relacionadas con un problema de compatibilidad entre las zonas de ejecución de Electron y Angular.

Estas zonas son contextos de ejecución que manejarán cambios en el estado de la aplicación que pueden ser provocados por:

- Eventos del usuario como click, change, input, submit,...
- XMLHttpRequest - Por ejemplo solicitando datos de un servicio remoto.
- Timers - setTimeout(), setInterval()

Electron utiliza su propio contexto de ejecución o zona proporcionada por zone-node para registrar y manejar los eventos del menú de ventana o las alertas nativas del sistema lo cual hace que cuando registramos un evento en Angular no se dispare o un cambio en los datos que se muestran al usuario no se produzca.

La solución consiste en indicar expresamente en código que cierta función debe ser ejecutada dentro de la zona de Angular.

Por otra parte, existe algún tipo de restricción por parte de sistemas operativos Windows que no permite abrir un socket de red para escuchar en el protocolo ICMP usando el módulo RAW-SOCKET, esta restricción, sin embargo, sí que permite realizar ping o *Requests* por lo que en Windows podemos enviar mensajes pero no recibirlos usando esta técnica. No se ha detectado ningún error en la aplicación dado que no se produce ningún error en la ejecución. Utilizando la herramienta Wireshark se ha comprobado como el tráfico es correcto en ambos equipos (emisor y receptor) aunque la aplicación no es capaz de detectar que ha recibido el mensaje.

9. Líneas de futuro

La aplicación desarrollada como prueba de concepto carece de ciertas características deseables que por falta de tiempo no se han añadido pero sería interesante para una futura versión 2.0.

Algunas de estas características se enumeran a continuación:

1. Permitir que dos usuarios utilicen distintas técnicas esteganográficas para una comunicación bidireccional.
2. Añadir más técnicas esteganográficas como LACK o ocultación en SIP para protocolos VoIP.
3. Permitir que en la técnica HTTP Request el usuario pueda establecer la ruta en la que escuchará el servidor web y en qué parámetros se puede ocultar los estego-mensajes.
4. En la misma técnica anterior, añadir un servicio web completo, que responda con contenido web para que en caso de que alguien decida conectarse a dicha web, reciba contenido que despiste y mantenga la invisibilidad del covert-channel.
5. Rediseño de la interfaz, ya que en el presente proyecto no se le ha dado gran importancia, dedicando la mayor parte del esfuerzo de desarrollo al diseño del estego-sistema.
6. Aunque en la prueba de concepto desarrollada sólo se utiliza texto en los estego-mensajes, en estos podemos incluir cualquier tipo de información por lo que si se amplía para poder enviar imágenes o cualquier otro tipo de archivo sería necesario eliminar los caracteres de relleno que se usan para completar los estego-mensajes que no llegan a 8 Bytes en el campo “mensaje”.

10. Bibliografía

[1] Freedom on the Net 2016, Silencing the Messenger: Communication Apps under Pressure. Freedom House. Sanja Kelly, Mai Truong, Adrian Shahbaz, and Madeline Earp.

[2] Covert Channels in Internet Protocols: A Survey. D Llamas, C Allison and A Miller

[3] ReLACK: A Reliable VoIP Steganography Approach. Mohammad Hamdaqa, Ladan Tahvildari

[4] Exploitation of data streams authorized by a network access control system for arbitrary data transfers: tunneling and covert channels over the HTTP protocol. Alex Dyatlov and Simon Castro.

[5] Curso de privacidad y protección de comunicaciones digitales. Cryptored Crypt4you. Universidad Politécnica de Madrid.

[6] Mazurczyk, W. 2013. VoIP steganography and its detection - A survey. ACM Comput. Surv. 46, 2, Article 20 (November 2013) 21 pages.
DOI: <http://dx.doi.org/10.1145/2543581.2543587>.

[7] Mehić M., Mikulec M., Voznak M., Kapicak L. (2014) Creating Covert Channel Using SIP. In: Dziech A., Czyżewski A. (eds) Multimedia Communications, Services and Security. MCSS 2014. Communications in Computer and Information Science, vol 429. Springer, Cham

[8] Cabuk, C. Brodley, and C. Shields, "IP Covert Timing Channels: Design and Detection." Purdue University, Tufts University, Georgetown University, USA,
<http://www.cs.jhu.edu/~fabian/courses/CS600.624/covert.pdf> 2004.

[9] Ildefonso Ruano Ruano "Transparencias de la asignatura 'Protocolos de soporte para las aplicaciones multimedia' del curso 2016/17", Universidad de Jaén, 2017.

[10] RFC3261. SIP: Session Initialization Protocol. Internet Engineering Task Force.
<https://tools.ietf.org/html/rfc3261>

[11] RFC2616. Hypertext Transfer Protocol -- HTTP/1.1. Internet Engineering Task Force
<https://tools.ietf.org/html/rfc2616>

[12] Mona Lisa painting 'contains hidden code'. The Telegraph, 12 Dec 2010.
<http://www.telegraph.co.uk/culture/art/art-news/8197896/Mona-Lisa-painting-contains-hidden-code.html>

11. Anexo I. Contenido Suministrado en CD

Todo el contenido relacionado con el presente trabajo se entrega en un CD el cual tiene la siguiente estructura de directorios y archivos:

- Source_code. Esta carpeta contiene el código fuente de la aplicación. Dentro de ella existe un fichero Readme.md el cual contiene instrucciones para poder continuar con el desarrollo.
- Memoria.pdf. Este fichero en formato PDF contiene la memoria del proyecto.
- StegoNet-Linux-x64. Esta carpeta contiene la aplicación lista para ser ejecutada en un sistema operativo Linux basado en Debian. Para abrir la aplicación es necesario ejecutar como “root” el archivo StegoNet.
- StegoNet-Win32-x64. Esta carpeta contiene la aplicación lista para ser ejecutada en un sistema operativo Windows (probado en W10). Para abrir la aplicación es necesario ejecutar como “administrador” el archivo StegoNet.exe.