UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

“PROTOTIPO DE SOFTWARE PARA EL CONTROL DE LAS VULNERABILIDADES ESTEGANOGRÁFICAS DEL PROTOCOLO HTTP DE LA CAPA APLICACIÓN EN LA OFICINA DE TECNOLOGÍA INFORMÁTICA DE LA UNIVERSIDAD NACIONAL DEL ALTIPLANO 2015”

PRESENTADO POR:

JHON WILSON SANCHEZ MAMANI

SERGIO ANTONIO HUIRSE CRUZ

PARA OPTAR EL TITULO PROFESIONAL DE:

INGENIERO DE SISTEMAS

Puno – Perú

2017

Universidad Nacional del Altiplano

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

PROTOTIPO DE SOFTWARE PARA EL CONTROL DE LAS VULNERABILIDADES ESTEGANOGRÁFICAS DEL PROTOCOLO HTTP DE LA CAPA APLICACIÓN EN LA OFICINA DE TECNOLOGÍA INFORMÁTICA DE LA UNIVERSIDAD NACIONAL DEL ALTIPLANO 2015

TESIS PRESENTADA POR:

JHON WILSON SANCHEZ MAMANI

SERGIO ANTONIO HUIRSE CRUZ

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS

APROBADA POR EL JURADO REVISOR CONFORMADO POR:

PRESIDENTE: .....
Dr. Mario Antonio Suarez López

PRIMER MIEMBRO: .....
Dr. Henry Iván Condori Alejo

SEGUNDO MIEMBRO: .....
Ing. Edwin Fredy Mamani Calderón

DIRECTOR DE TESIS: .....
Mg. Elmer Coyla Idme

ÁREA: INFORMÁTICA
TEMA: SEGURIDAD INFORMÁTICA
LÍNEA: ESTEGANOGRAFÍA

Puno – Perú
2017

DEDICATORIA

El presente trabajo de investigación, está dedicado principalmente a Dios por ser mi Guía e inspiración en quien hallé la fortaleza para continuar adelante pese a las dificultades, A mi Madre por el apoyo brindado y su continuo aliento durante el desarrollo del presente proyecto.

JHON WILSON

Quiero dedicarle este trabajo A Dios que me ha dado la vida y fortaleza para terminar este proyecto de investigación, A mi Madre por ser la persona que me ha acompañado durante todo mi trayecto estudiantil y de la vida.

SERGIO ANTONIO

AGRADECIMIENTO

Agradezco de sobremanera a cada una de las personas que han brindaron su apoyo, también a los docentes, sus valiosos aportes sobre temas de gran interés para nuestro desarrollo profesional; También agradezco a los ingenieros Elmer Coyla Idme y Edelfre Flores Velásquez por su apoyo y asesoría desinteresada.

ÍNDICE

ÍNDICE DE CUADROS

ÍNDICE DE GRÁFICOS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

RESUMEN

La presente investigación tiene la finalidad de presentar un prototipo de software para demostrar, controlar y aplicar esteganografía sobre el modelo de comunicación TCP/IP, básicamente sobre el protocolo HTTP de la capa aplicación. Como escenario de Investigación se tomó a la OTI (Oficina de tecnología informática) de la Universidad Nacional del Altiplano. En la primera etapa se plantea un análisis del protocolo de comunicación HTTP para aplicar Esteganografía sobre el mismo. Se entiende, aplicar esteganografía sobre un protocolo a las modificaciones que se aplica a un mensaje del protocolo que permita comunicar un mensaje de manera inadvertida, fuera de las reglas del protocolo. En la segunda etapa tomando como base los puntos analizados en la primera etapa se diseña una comunicación esteganográfica sobre el protocolo HTTP en el servidor de internet de la Oficina de tecnología informática de la Universidad Nacional del Altiplano, todo ello nos ayudó a obtener que existe comunicación esteganográfica, la prueba índica que la institución en estudio es vulnerable a la esteganografía una vez identificado la problemática. En la tercera etapa se diseña el prototipo de software adaptable que será situado en el servidor situado en la ruta entre aquellos que realizan la comunicación para el bloqueo de dichos mensajes modificados.

Palabras Clave: Vulnerabilidad, esteganográfica, Protocolo HTTP, software.

ABSTRACT

The present research has the purpose of presenting a prototype software to demonstrate, control and apply steganography on the TCP / IP communication model, basically on the application layer HTTP protocol. As a research scenario, the OTI (Office of Information Technology) of the National University of the Altiplano. In the first stage an analysis of the protocol of HTTP communication is proposed to apply Steganography on the same. It is understood, to apply steganography on a protocol to the modifications that is applied to a message of the protocol that allows to communicate a message inadvertently, outside the rules of the protocol. In the second stage, based on the points analyzed in the first stage, a steganographic communication on the HTTP protocol was designed on the Internet server of the Computer Technology Office of the National University of the Altiplano, all of which helped us to obtain communication Steganography, the test indicates that the institution under study is vulnerable to steganography once the problematic is identified. In the third stage the prototype adaptable software is designed that will be located in the server located in the route between those that make the communication for the blocking of those modified messages.

Keywords: Steganographic, vulnerability, HTTP protocol, software.

INTRODUCCIÓN

En la presente tesis, propone presentar un prototipo de software para demostrar, controlar y aplicar esteganografía sobre el modelo de comunicación TCP/IP, básicamente sobre el protocolo HTTP de la capa aplicación.

Como escenario de Investigación se tomó a la OTI (Oficina de tecnología informática) de la Universidad Nacional del Altiplano. En la actualidad gracias al desarrollo de la tecnología se crean distintas formas de robar información, la esteganografía es una de ellas por la cual se puede vulnerar sistemas de seguridad informática, la OTI, acrónimo de la institución en estudio, ha visto la necesidad de establecer nuevas estrategias y lineamientos para garantizar un buen sistema de seguridad informática.

La problemática descubierta gira en torno a la vulnerabilidad que existe en OTI frente a la comunicación esteganográfica.

CAPITULO I: Planteamiento del Problema de Investigación. Contiene la descripción del problema, formulación del problema, la justificación del problema y los objetivos de la investigación.

CAPITULO II: Marco Teórico. Se presentan sus antecedentes de la investigación, el marco teórico donde se definen los conceptos que son el apoyo teórico sobre las cuales se apoya el desarrollo de la presente investigación, además se describe en el marco conceptual, Términos técnicos que ayudaran a comprender de una mejor manera los conceptos usados, así como las hipótesis de la investigación y la operacionalización de variable.

CAPITULO III: Diseño Metodológico de la Investigación. Se describe el tipo y diseño de investigación, el ámbito del estudio.

CAPITULO IV: Presentación, Análisis e Interpretación de Resultados. Se describe la estructura; Demostrar, Detectar y reescribir.

Finalmente se tiene Las conclusiones del trabajo de investigación, también se incluye las recomendaciones, la bibliografía y los anexos

CAPITULO I

PLANTEAMIENTO DEL PROBLEMA DE INVESTIGACIÓN

1.1 DESCRIPCIÓN DEL PROBLEMA.

En la actualidad, la información es uno de los activos más valiosos para las empresas y organizaciones. Por ello, se requiere asegurar y proteger las redes mediante las cuales se maneja dicha información de forma apropiada. Básicamente en el entorno de comunicaciones actual, la información que se maneja tanto al interior de una organización como hacia el exterior de la misma está expuesta a un gran número de amenazas.

Una de las amenazas a las que los activos de información están expuestos es la fuga de información, una amenaza que conlleva la divulgación no autorizada de información confidencial. Además, existen varios medios informáticos que pueden ser utilizados para el robo de los activos tales como las redes de datos, impresoras en red, almacenamiento portátil y dispositivos móviles avanzados.

En el caso de las redes de datos según es posible trasmitir información de tal manera que la propia transmisión no sea detectada ni por los sistemas de prevención de fugas actuales, esto mediante los protocolos de red que permiten la creación de canales encubiertos creados por medio de técnicas de ocultamiento de información como la esteganografía.

Una de las diversas formas de uso de la esteganografía es mediante el ocultamiento de cadenas de texto en los Headers o cabeceras de los mensajes HTTP/1.1.

Se ha observado que el protocolo HTTP de la capa aplicación representa en promedio un 31.3% (anexo 4) del tráfico de red en un intervalo de 5 minutos entre respuestas y peticiones del protocolo HTTP en la capa aplicación del

modelo TCP para los cuales todavía no se ha implementado ningún tipo de control esteganográfico ante este tipo de amenaza y esto pone en riesgo la información sensible de la universidad.

La información de carácter privado, referente a los riesgos académicos (notas, matriculas) para los estudiantes. También se considera la información de carácter confidencial, los datos de acceso (contraseñas y accesos) a los sistemas y bases de datos internos (sistema académico), las cuentas usuario de los estudiantes y docentes referentes al sistema académico y los controles perimetrales de la red

Cabe mencionar también que, los incidentes de fuga de información originados desde el interior de las organizaciones son mucho más graves, de forma general, que los organizados desde el exterior de las mismas y los usuarios interiores son propensos a cometer crímenes que involucran a los activos de información que la organización posee

1.2 DEFINICIÓN DEL PROBLEMA

1.2.1 PROBLEMA GENERAL

¿En qué medida un prototipo de software controla las vulnerabilidades esteganográficas del protocolo http de la capa aplicación en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015?

1.2.2 PROBLEMAS ESPECIFICOS

a) ¿Es posible filtrar información evitando los controles de seguridad utilizando esteganografía mediante el protocolo http de la capa aplicación en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015?

b) ¿En qué medida un prototipo de software detectar y limpiar los mensajes embebidos por esteganografía del protocolo http de la capa aplicación en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015?

1.3 JUSTIFICACIÓN DE LA INVESTIGACIÓN

Esta investigación es pertinente porque se desarrolla un prototipo de software para el control de las vulnerabilidades esteganográficas y medir su impacto en la comunicación de datos a través del protocolo http de la capa aplicación.

La investigación es importante a nivel teórico porque permitirá ampliar conocimientos acerca de la esteganografía en el protocolo HTTP y como este protocolo que puede convertirse en una amenaza potencial, tanto para la empresa, como para la seguridad en general.

La investigación es importante a nivel práctico porque mediante el software se controlará y evitará posibles robos de la información confidencial y privada de la universidad mediante las técnicas esteganográficas y además se proyecta generar un algoritmo para impedir el uso de la creación de canales encubiertos mediante esteganografía a través del protocolo HTTP y un nuevo método esteganográfico para detectar la existencia de información oculta en conexiones http.

Además, el proyecto es viable ya que se tiene acceso a las redes de datos de la Oficina de Tecnología Informática de la Universidad.

1.4 OBJETIVOS DE LA INVESTIGACIÓN

1.4.1 OBJETIVO GENERAL.

Determinar que mediante la aplicación del prototipo de software se puede controlar las vulnerabilidades esteganográficas del protocolo http de la capa aplicación en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015.

1.4.2 OBJETIVOS ESPECÍFICOS.

a) Demostrar que mediante el protocolo http utilizando esteganografía es posible filtrar información sin ningún problema evitando los controles de seguridad.

b) Detectar y reescribir los mensajes embebidos por esteganografía del protocolo http de la capa aplicación en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015.

CAPITULO II

MARCO TEORICO

Este capítulo tiene como objetivo revisar, integrar y complementar aspectos teóricos respecto al trabajo de investigación.

2.1 ANTECEDENTES DE LA INVESTIGACIÓN:

Bonilla Luis Jorge, Llano Mónica, Ramírez Alfonso (2012). En su investigación ESTEGANOGRAFÍA EN EL PROTOCOLO HTTP.

Plantea las bases teóricas de la Esteganografía, sus diferentes medios, usos y sus implicaciones en la seguridad. También, explica cómo es el modo de operación del protocolo HTTP y algunas de las posibles arquitecturas de un sistema que permita enviar un mensaje oculto dentro del protocolo HTTP.

Además, reafirma que el protocolo HTTP es uno de los más utilizados actualmente, puede pasar información oculta e incluso distribuir dicha información a varias personas que se encuentren distantes sin ser detectadas de una forma sencilla.

Chicaiza N. M. & Marco A. S. & Pandora M. S. (2010). En su tesis INVESTIGACIÓN, ANÁLISIS Y PRUEBAS DE LOS PROCESOS DE ESTEGANOGRAFÍA.

Realiza un análisis detallado de las técnicas esteganográficas en los diferentes ámbitos en los que se pueden desarrollar esta técnica. Además de realizar pruebas de estudio de comunicación basadas en técnicas esteganográficas.

También se estudia el protocolo de comunicación HTTP/1.1 y se concluye que este siempre ha tenido la peculiaridad de ser vulnerable ya que es muy común y además, este se manipula con software que se puede utilizar sin costo alguno desde el internet y esto conlleva a aplicar diversas técnicas de

ocultamiento y de encriptación de contraseñas o de manipulación de información de forma general.

Por otro lado, se afirma que la seguridad de la información depende básicamente de las técnicas utilizadas por lo que en la investigación se propone a la esteganográfica como unas alternativas viables y confiables.

Y por último (Blasgo, 2012) en su tesis REVELACIÓN DE INFORMACIÓN Y ESTEGANOGRAFÍA: DETECCIÓN Y BLOQUEO DE CANALES ENCUBIERTOS.

Concluye que los canales encubiertos en el protocolo HTTP son la técnica esteganográfica más accesible en una red de datos por las múltiples vulnerabilidades que presentan los protocolos en general.

También concluye que los empleados maliciosos son una preocupación cada vez mayor para las organizaciones debido a los daños que son capaces de inhibir sobre los diferentes activos de la misma. En este sentido, el robo de información es una de las acciones más dañinas que suelen ser cometidas por estos.

El robo de información sensible no solo produce grandes pérdidas económicas y problemas de competitividad, sino que también pone en peligro la viabilidad de las organizaciones.

Además, se hace un estudio de los canales encubiertos creados por medio de la Esteganografía y el uso malicioso de aplicaciones de confianza han sido identificados como medios de extracción de información no tenidos en cuenta por las soluciones actuales.

2.2 SUSTENTO TEORICO:

2.2.1 Esteganografía

Se denomina esteganografía a la técnica y el proceso de incorporar mensajes que se desea mantener secretos dentro de otros datos, llamados portadores. El objetivo de la esteganografía consiste en la comunicación de mensajes entre dos pares (en el sentido de iguales) a través de una canal de comunicación, de modo que el prototipo acto de la comunicación pase inadvertido por quienes puedan tener acceso a ese canal. Este concepto radica en la teoría de seguridad por oscuridad, que supone que si nadie conoce que existe un mensaje oculto nadie tratara de obtenerlo el canal simplemente el medio utilizado para transmitir el mensaje oculto, nadie tratara de obtenerlo. El canal es simplemente el medio de utilización para transmitir el mensaje desde el emisor hacia el receptor (Jessica Fridrich, 2008).

Los canales encubiertos pueden clasificarse en:

Canal encubierto de almacenamiento: son aquellos en los que la aplicación de esteganografía se desarrolla mediante alteración misma de datos almacenados o transmitidos como ejemplo de ellos se puede mencionar: la alteración de campos no utilizados o sin significado particular en un protocolo de comunicación, o la modificación de los colores que componen a cada uno de los pixeles de una imagen en un archivo (Timothy, 2008).

Canales encubiertos temporales: son aquellos en los que la aplicación de esteganografía se desarrolla a partir de la interpretación del desarrollo temporal de un evento sobre el portador. Como ejemplo, si se recibe antes un mensaje A y luego el B se puede asignar un significado diferente así se reciben en el orden inverso (Timothy, 2008).

Otros tipos de canales encubiertos derivados de los anteriores: A partir de canales encubiertos de almacenamiento y temporales pueden construirse otros que hagan uso de alguna propiedad derivada. Por ejemplo, se puede realizar un análisis de la variación de un canal encubierto temporal y obtener una variable estadística que constituye una forma de comunicación (e.g. la varianza). Como, por ejemplo, se puede mencionar el análisis de tiempos de respuesta de un protocolo de comunicación de tipo pedido/respuesta (de ingles request/reply), como lo es el protocolo Internet Control Message Protocolo ICMP (Postel, 1981).

El portador es todo aquel conjunto de datos que sea susceptible de ser alterado para incorporarle un mensaje que se mantiene secreto, a partir de la aplicación de un algoritmo esteganográfico (denominado estega-algoritmo) que indica cómo realizar el procedimiento de incorporación.

Las practicas esteganográficas se emplean con el fin de realizar una comunicación entre dos partes, en este sentido, para que pueda hablarse de esteganografía, debe haber voluntad de comunicación por parte de emisor y del receptor.

Se define como esquema esteganografía al conjunto de componentes que permite llevar a cabo la comunicación esteganográfica: La selección del tipo de portadores: esto es, sobre qué tipo de medios se aplicara esteganográfico. Esta decisión está condicionada por los portadores a los que puedan tener acceso el emisor y receptor, a transmitir sobre el canal de comunicación que los vincula. Los algoritmos para embeber y extraer el mensaje en el portador, los cuales contemplan las modificaciones que se permite realizar al tipo de portador, la forma de interpretar las modificaciones realizadas y una posible clave

esteganográfica para parametrizar los algoritmos. La manera de transmitir el portador (e.g. a través de una red, de un archivo enviado adjunto en un correo electrónico, de un texto impreso) (Jessica Fridrich, 2008).

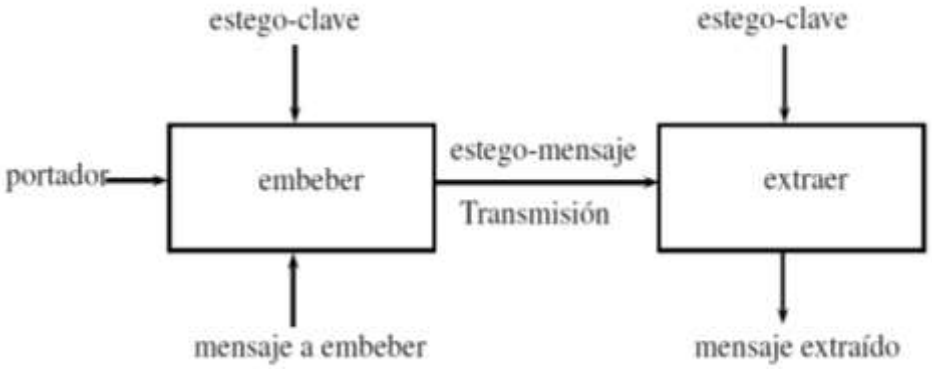

Grafico 1 Esquema de comunicación esteganográfica
Fuente: Jessica Fridrich, 2008

2.2.2 Protocolos de red

Los protocolos son piezas software que deben instalarse en los componentes de red que los necesiten. Los equipos sólo pueden comunicarse entre sí si utilizan el mismo protocolo. Si el protocolo utilizado por el equipo de una red no es compatible con el utilizado por otro equipo, no podrán intercambiar información. Hay diversos protocolos disponibles que pueden utilizarse en entornos de red específicos. Aunque cada protocolo facilita la comunicación básica de red, cada uno tiene una función distinta y lleva a cabo diferentes tareas (Feria Gerónimo, 2009).

2.2.3 El protocolo HTTP (Hypertext Transfer Protocol)

La especificación SOAP no indica ninguna manera específica de trasportar la información, de modo que los mensajes podrían viajar a través de protocolos de transporte, archivos de texto o cualquier otro método de transferencia de datos.

El modelo TCP/IP cuenta con diversos protocolos en su capa de aplicación: HTTP, SMTP y FTP son tres de los más importantes. En principio cualquiera de ellos puede ser utilizado para la transferencia de mensajes SOAP. En este proyecto utilizaremos HTTP, el protocolo estándar para la web y el más usado en los servicios web XML (Feria Gerónimo, 2009).

El protocolo de transferencia de hipertecto(Hypertext Transfer Protocol) es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes web y los servidores HTTP. La especificación completa del protocolo HTTP/1.0 está recogida en el RFC 1945. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como el World Wide Web. Actualmente la versión más reciente de HTTP es la 1.1 y su especificación se encuentra recogida en el documento RFC 2616 (Donate, 2012).

2.2.4 Simulación

La técnica de simulación de sistemas es una herramienta de análisis, cuyo uso se ha extendido a diversas áreas (Administración, Economía, Ingeniería, etc.) en forma muy reciente. Su inicio se ubica en la década de los 40's cuando Von Newman y Ulam la utilizaron para analizar y resolver problemas complejos en el campo de la física, en los que la solución analítica no siempre es factible o bien, es demasiado costosa para hacerlo en forma experimental.

El uso de la computadora con las facilidades y ventajas que esto implica, hizo posible que la simulación de sistemas, ampliara las áreas y objetivos de su aplicación, así fue como se incrementó el uso de la técnica para realizar funciones de control y evaluación, experimentación controlada de sistemas

administrativos y/o económicos, prognosis, etc (Rodríguez Torres, Federico Delgado, & Ricardo, 1991).

2.2.5 Concepto de Simulación

Al respecto, una de las más conocidas es la que se atribuye a Naylor, Balintfy, Burdick y Kong Chu "Simulación, es una técnica numérica para conducir experimentos en una computadora digital, los cuales requieren ciertos tipos de modelos lógicos y matemáticos» que describen el comportamiento de un negocio c un sistema económico (o algún componente de ellos) en periodos extensos de tiempo real".

2.2.6 Metodología de Simulación.

Una vez definido el concepto de simulación, sistema y modelo, brevemente se describirán las fases de la simulación. Esencialmente la metodología incluye las siguientes etapas:

2.2.6.1 Definición y Formulación del problema

Parece razonable que al igual que otras formas de investigación, la etapa inicial sea la definición y formulación del problema y/o la definición exacta de los objetivos del experimento. En forma general se debe especificar la información que se pretende obtener, las hipótesis que se van a probar o los efectos con los cuales se va a experimentar.

La determinación específica de estos aspectos permitirá no perder de vista el objeto de la investigación y establecer los alcances de esta.

2.2.6.2 Conceptualización del sistema

Una vez definida y aclarada la problemática, debe considerarse como una segunda etapa un proceso de abstracción de La realidad para conceptualizar la

porción de ésta, como un sistema. Esto permitirá comprender cabalmente los procesos y/o mecanismos básicos de funcionamiento.

2.2.6.3 Formulación de un modelo matemático

Ya definidos los objetivos experimentales y teniendo el conocimiento del sistema, la siguiente etapa es la formulación de un modelo matemático, que relacione las variables endógenas del sistema con las exógenas. Se supone que las variables exógenas se determinan mediante las fuerzas externas al sistema y pueden ser del tipo aleatorio o expresarse en forma de tendencias de tiempo.

Una de las primeras dificultades que se presentan en la construcción de un modelo, es la selección de las variables que se deben incluir en él. Sin embargo, esta dificultad puede ser mínima si se ha llevado a cabo la etapa anterior.

Otra consideración importante es la complejidad del modelo, como antes se mencionó, debe buscarse un balance entre la simplicidad y el poder explicativo del modelo, sin afectar su validez.

2.2.6.4 Estimación de Parámetros

Posterior a la formulación del modelo matemático que describe al sistema, es necesario estimar los valores de los parámetros de dicho modelo y comprobar la importancia estadística de tas estimaciones. Para ello se tomará como base las observaciones extraídas de La realidad.

2.2.6.5 Evaluación del modelo

Se hace necesario realizar una evaluación del modelo, es decir, ponerlo a prueba. Lo anterior es fundamental considerando que poco o nada se ganará utilizando un modelo inadecuado para la simulación del sistema. Un nivel de

referencia para esta evaluación es realizar un cálculo manual y comparar los resultados con los valores teóricos de las variables endógenas del modelo, con los valores históricos o reales de dichas variables. Si el modelo no pasa satisfactoriamente esta evaluación, lo más conveniente es retomar el proceso desde su primera etapa.

2.2.6.6 Formulación del Programa de Computadora.

La formulación de un programa de computadora para experimentos de simulación, requiere esencialmente de 3 aspectos:

1) EL programa de computadora

2) Información de entrada y condiciones iniciales

3) La generación de datos.

El primer paso, comprende la realización de un diagrama de flujo que describa la secuencia lógica de los sucesos que se van a desarrollar.

Otro aspecto importante, es la cuestión de los datos de entrada y las condiciones iniciales. Aquí es necesaria la determinación de los valores que se van a asignar a las variables y parámetros del modelo en el momento del inicio, para lo cual es necesario recurrir a métodos de ensayo y error y obtener resultados sin sesgo.

Finalmente, el problema que se ha de resolver se refiere al desarrollo de técnicas numéricas para la generación de datos, estos pueden introducirse desde fuentes externas, o bien, producirse internamente por medio de subrutinas.

2.2.6.7 Validación.

La validación de modelos de simulación implica complejidades prácticas, teóricas, estadísticas e incluso filosóficas. Sin embargo, existen dos pruebas que parecen ser apropiadas para realizar la validación:

Primero; comparar los valores simulados de las variables endógenas o de salida y los datos históricos conocidos, si es que éstos existen.

Segundo; determinar la exactitud que tienen las predicciones del modelo de simulación respecto al comportamiento del sistema real en otros periodos.

2.2.7 Características y funcionamiento

2.2.7.1 Diseño Experimental

Cuando se realiza una investigación sobre el efecto de un factor en una respuesta, se encuentran esencialmente con cuatro problemas de diseño experimental.

1) El de convergencia estadística

2) El del tamaño

3) El del motivo

4) El de respuesta múltiple.

Desde el punto de vista de las comunicaciones, HTTP es estable sobre la capa de conexión TCP/IP, y funciona de la misma forma que el resto de los servicios comunes de entornos UNIX: un proceso servidor escucha en un puerto de comunicación TCP (por efecto, el 80), y espera las solicitudes de conexión de los clientes web. Una vez que se establece la conexión, el protocolo TCP se encarga de mantener la comunicación y garantizar un intercambio de datos libre de errores (Katz, 2013).

HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o curso sobre el servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan; cada objeto web es identificado por URL. Las principales características del protocolo HTTP son:

- Toda la comunicación entre los clientes y servidores se realiza a partir de caracteres US-ASCII de 7 bits. Permite la trasferencia de objetos multimedia, codificando los archivos binarios en cadenas de caracteres. El contenido de cada objeto intercambiado está identificado por su clasificación MINE.
- Existen ocho verbos que permiten que un cliente pueda dialogar con el servidor los tres más utilizados son: GET, para escoger un objeto, POST, para enviar información al servidor y HEAD, para solicitar las características de un objeto.
- Cada operación HTTP implica una conexión con el servidor, que es liberada al término de la misma. Es decir, en una operación se puede recoger un único objeto. Con la versión HTTP 1.1 se ha mejorado este procedimiento, permitiendo que una misma conexión se mantenga activa durante un cierto periodo de tiempo de forma que se utiliza en sucesivas transacciones. Este mecanismo, denominado HTTP Keep Alive, es empleado por la mayoría de los clientes y servidores modernos.

No mantiene estado. Cada petición de un cliente a un servidor no es influida por las transacciones anteriores. El servidor trata cada petición como una operación totalmente independiente del resto.

- Cada objeto al que se aplica los verbos del protocolo está identificado a través de un localizador uniforme de recurso (URL) único (Aznar Lopéz, 2005).
- Cada vez que un cliente realiza una petición a un servidor, se ejecuta los siguientes pasos:
  - ✓ Un usuario accede a una URL, seleccionando un enlace de documento HTML o introduciéndola directamente en el navegador.
  - ✓ El cliente web descodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el puerto (de carácter opcional; el valor por defecto es 80) y el objeto requerido del servidor.
  - ✓ Se abre una conexión TCP/IP con el servidor, llamado al puerto TCP correspondiente
  - ✓ Se realiza la petición. Para ello, se envía el comando necesario (GET, POST, HEAD), la dirección del objeto requerido (el contenido de la URL que sigue a dirección del servidor), la versión del protocolo HTTP empleada y un conjunto variable de información, que incluya datos sobre las capacidades del navegador, datos opcionales para el servidor.
  - ✓ El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MINE de la información de retorno, seguido de la propia información.

✓ Se cierra la conexión TCP. Si no se utiliza el modo HTTP Keep Alive, este proceso se repite para cada acceso al servidor HTTP.

- El dialogo con los servidores HTTP se establece a través de mensajes formados por líneas de texto, cada una de las cuales contiene los diferentes comandos y opciones del protocolo. Solo existe dos tipos de mensajes, uno para realizar peticiones y otro para devolver la correspondiente respuesta.
- En el tema de comunicaciones, el protocolo está soportado sobre los servicios de conexión prestados por los protocolos TCP/IP y se basa en sencillas operaciones entre el cliente y el servidor de solicitud y respuesta (Aznar Lopéz, 2005).

2.2.8 Comandos de HTTP

El protocolo HTTP/1.1 consta de los siguientes comandos:

- **GET:** sirve para recoger cualquier tipo de información del servidor. Se utiliza siempre que se pulsa sobre un enlace o se teclea directamente a una URL como resultado, el servidor HTTP envía el documento ubicado en la dirección específica por dicha URL.
- **HEAD:** es un comando similar a GET pero que pide solamente la cabecera del objeto. Lo utilizan principalmente los gestores de caches de páginas o los servidores proxy para conocer cuando es necesario actualizar la copia que se mantiene de un fichero.
- **POST:** este comando envía datos de información al servidor, normalmente procedentes de un formulario web, para que el servidor los administre o los añada a una base de datos

- **PUT:** almacena un objeto en la URL especificada. Si la dirección de destino ya contenía un objeto, se considera que se está enviando una versión actualizada del mismo.
- **DELETE:** elimina el objeto especificado. Este comando es muy poco utilizado.
- **TRACE:** realiza un eco de la solicitud recibida para que el cliente pueda conocer que servidores intermedios están añadiendo información o modificando la petición.
- **OPTIONS:** devuelve los métodos HTTP que soporta el cliente. Se suele utilizar para comprobar la funcionalidad de un servidor web.
- **CONNECT:** se utiliza en los servidores proxy que puedan establecer un túnel dinámicamente.

Ante cada transacción con el servidor HTTP, este devuelve un código numérico en la primera línea del mensaje de respuesta que informa sobre el resultado de la operación. Estos códigos aparecen en algunos casos en la pantalla del cliente, cuando se produce un error.

El cliente normalmente crea una comunicación con el servidor a través de un puerto predeterminado (puerto 80) y le envía los mensajes con los datos de su solicitud. El servidor le responde con un mensaje indicándole el estado de la operación de consulta y su posible resultado, seguido de la información que haya solicitado como tal.

Los resultados normalmente se conocen como objetos o recursos y suelen ser páginas HTML, imágenes o archivos binarios

La comunicación completa entre clientes y servidores se realiza a partir de caracteres de 8 bits. Así, se puede transmitir cualquier tipo de documento: texto, binario, etc., sin cambiar su formato original.

Cada operación HTTP implica una conexión con el servidor, que es liberada al término de la misma. Esto quiere decir que para cada objeto, archivo o imagen que pueda contener la solicitud del cliente, se realizará una conexión para transmitirlo. Una vez transmitido, el protocolo TCP se encargará de cerrar la conexión, para establecer una nueva de ser necesario.

El protocolo no mantiene el estado, esto quiere decir que cada petición de un cliente a un servidor no depende, ni es influida por las transacciones anteriores. El servidor trata cada petición como una operación totalmente independiente del resto de operaciones que le preceden o le anteceden.

La comunicación con los servidores HTTP, normalmente se establece a través de mensajes de texto, que contienen los diferentes comandos y opciones del protocolo (Feria Gerónimo, 2009).

2.2.9 Seguridad Informática

La seguridad informática, también conocida como ciberseguridad o seguridad de tecnologías de la información, es el área de la informática que se enfoca en la protección de la infraestructura computacional y todo lo relacionado con esta y, especialmente, la información contenida o circulante. Para ello existen una serie de estándares, protocolos, métodos, reglas, herramientas y leyes concebidas para minimizar los posibles riesgos a la infraestructura o a la información. La seguridad informática comprende software (bases de datos, metadatos, archivos), hardware y todo lo que la organización

valore y signifique un riesgo si esta información confidencial llega a manos de otras personas, convirtiéndose, por ejemplo, en información privilegiada.

La definición de seguridad de la información no debe ser confundida con la de «seguridad informática», ya que esta última solo se encarga de la seguridad en el medio informático, pero la información puede encontrarse en diferentes medios o formas, y no solo en medios informáticos.

La seguridad informática es la disciplina que se ocupa de diseñar las normas, procedimientos, métodos y técnicas destinados a conseguir un sistema de información seguro y confiable.

Puesto simple, la seguridad en un ambiente de red es la habilidad de identificar y eliminar vulnerabilidades. Una definición general de seguridad debe también poner atención a la necesidad de salvaguardar la ventaja organizacional, incluyendo información y equipos físicos, tales como los mismos computadores. Nadie a cargo de seguridad debe determinar quién y cuándo puede tomar acciones apropiadas sobre un ítem en específico. Cuando se trata de la seguridad de una compañía, lo que es apropiado varía de organización en organización. Independientemente, cualquier compañía con una red debe tener una política de seguridad que se dirija a la conveniencia y la coordinación.

2.2.10 PROGRAMACIÓN EXTREMA (EXTREME PROGRAMMING, XP)

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación

fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Metodologías Agiles	Metodologías Tradicionales
Basadas en heurísticas proveniente de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuesta externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (menor de 10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Mas artefactos
Pocos roles	Mas roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Cuadro 1: Comparación entre metodologías
Fuente: Kent Beck

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP en sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea. A continuación, presentaremos las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas.

Roles XP Los roles de acuerdo con la propuesta original de Beck son:

Programador.- El programador escribe las pruebas unitarias y produce el código del sistema.

Cliente.- Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

Encargado de pruebas (Tester).- Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Encargado de seguimiento (Tracker).- Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

Entrenador (Coach).- Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

Consultor.- Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

Gestor (Big boss).- Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

2.2.11 PROCESO XP

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

i. El cliente define el valor de negocio a implementar.

ii. El programador estima el esfuerzo necesario para su implementación.

iii. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.

iv. El programador construye ese valor de negocio.

v. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden.

No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración. El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

2.2.12 PRACTICAS XP

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas.

El juego de la planificación.- Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

Entregas pequeñas.- Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

Metáfora.- El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).

Diseño simple.- Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

Pruebas.- La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

Refactorización (Refactoring).- Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

Programación en parejas.- Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores).

Propiedad colectiva del código.- Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

Integración continua.- Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

40 horas por semana.- Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.

Cliente in-situ.- El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.

Estándares de programación.- XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras. donde una línea entre dos prácticas significa que las dos prácticas se refuerzan entre sí. La mayoría de las prácticas propuestas por XP no son novedosas, sino que en alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica (para un análisis histórico de ideas y prácticas que sirven como antecedentes a las utilizadas por las metodologías ágiles). El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

2.2.13 Glosario de términos básicos

Arquitectura de Red: Se denomina arquitectura de red al conjunto de capas y protocolos que se utilizan en la comunicación en la red. La separación entre capas se denomina interfaz. (Katz, 2013)

Cliente-Servidor: Esquema donde algunos elementos presentan servicio (Servidores) y otros los usan (Clientes). (Herrera Perez, 2013)

Confiabilidad: Se denomina confiabilidad a la probabilidad de que el algoritmo de extracción (Ext) emita como salida el mensaje esteganografico correcto. (Gómez Vieites, 2007)

Criptografía: Método por el cual se produce información cifrada a partir de la información original con el propósito de que sea confidencial. (Gómez Vieites, 2007)

Esteganografía: Se denomina Esteganografía a la técnica y el proceso de incorporar mensajes que se desea mantener secretos dentro de otros datos, llamados portadores.

Esteganografía de protocolo: Se denomina Esteganografía de protocolo a la aplicación de técnicas esteganográficas sobre mensajes de protocolos de comunicación utilizados como portadores. (Ortega Triguero & López G, 2005)

Embeber: La acción de ocultar el mensaje dentro del portador se denomina embeber, mientras acción de la recuperación posterior del mensaje oculto se denomina extraer. (Ortega Triguero & López G, 2005)

HTTP: (Hiper text Transfer Protocol) protocolo para transferencia de información a través de la web. (Philippe Atelin, 2006)

Intranet: Una intranet es una red informática que utiliza la tecnología del Protocolo de Internet para compartir información, sistemas operativos o servicios de computación dentro de una organización. (Aznar Lopéz, 2005)

Intranet: Una intranet es una red informática que utiliza la tecnología del Protocolo de Internet para compartir información, sistemas operativos o servicios de computación dentro de una organización. (Aznar Lopéz, 2005)

Mensaje de protocolo: Se denomina mensaje de protocolo a cada una de las unidades que se transmite independientemente a través de un canal de comunicación y que está construida sobre la base de las reglas del protocolo de comunicación. (Herrera Perez, 2013)

Modelo OSI: Estándar hecho por ISO el cual consiste en un modelo de red, arquitectura de redes de computadoras. (Feria Gerónimo, 2009)

Políticas de seguridad: Una política de seguridad es un plan de acción para afrontar riesgos de seguridad, o un conjunto de reglas para el mantenimiento de cierto nivel de seguridad. Pueden cubrir cualquier cosa desde buenas prácticas para la seguridad de un solo ordenador, reglas de una empresa o edificio, hasta las directrices de seguridad de un país entero.

Portador: El portador es todo aquel conjunto de datos que sea susceptible de ser alterado para incorporarle un mensaje que se mantiene secreto, (Ortega Triguero & López G, 2005)

Proxy: Elemento de red cuya función es que alguien externo a una red se conecte directamente a un elemento de la red privada. (Katz, 2013)

Robustez: Se denomina robustez o Resistencia a la manipulación al grado de inmunidad del estego-mensaje frente a alteraciones posteriores realizadas por un guardián activo. Esto es, la probabilidad de que el algoritmo de extracción

(Ext) emita como salida el mensaje esteganográfico cuando el estego-mensaje fue modificado. (Vieites, 2007)

Seguridad informática: La seguridad informática es la disciplina que se ocupa de diseñar las normas, procedimientos, métodos y técnicas destinados a conseguir un sistema de información seguro y confiable.

TCP/IP: Protocolos que permiten la interconexión de computadores con el fin de construir redes abiertas. (Aznar Lopéz, 2005)

Vulnerabilidad: Se denomina vulnerabilidad a la deficiencia que pueden ser explotadas por amenazas- (Vieites, 2007)

2.3 HIPOTESIS DE LA INVESTIGACION

2.3.1 HIPOTESIS GENERAL

El prototipo de software controla las vulnerabilidades esteganográficas del protocolo http de la capa aplicación en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015

2.3.2 HIPOTESIS ESPECÍFICAS.

a) Es posible filtrar información mediante el protocolo http utilizando esteganografía información sin ningún problema evitando los controles de seguridad.

b) El prototipo de software detecta y reescribe los mensajes embebidos por esteganografía del protocolo http de la capa aplicación en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015

2.4 OPERACIONALIZACIÓN DE VARIABLES:

Variable Independiente			
	Dimensiones	Indicadores	Escala
prototipo de software	Efectividad	Adaptabilidad	- Muy bueno [95 – 100]% - Bueno [80 – 95]% - Regular [60– 80]% - Malo [0 – 60]% - Nada 0%
	Usabilidad	Interface	- fácil de usar - complicado
Variable dependiente			
Control de vulnerabilidades estenográficas del protocolo http de la capa aplicación	Mensajes embebidos	Numero de Menajes detectados	- Muy bueno [95 – 100]% - Bueno [80 – 95]% - Regular [60– 80]% - Malo [0 – 60]% - Nada 0% -
		Numero de mensajes sobrescritos	- Muy bueno [95 – 100]% - Bueno [80 – 95]% - Regular [60– 80]% - Malo [0 – 60]% - Nada 0%

Cuadro 2: Operacionalización de Variables
Fuente: Propia
Elaboración: Propia

CAPITULO III

DISEÑO METODOLÓGICO DE LA INVESTIGACIÓN

3.1 Tipo de investigación:

Esta tesis de acuerdo con las características de la hipótesis, los objetivos y el problema de investigación se enmarca dentro del enfoque cuantitativo (Sampieri, 2013) cuya característica principal es probar la hipótesis en base a la medición numérica y el análisis estadístico

3.2 Diseño de investigación:

Diseño de la presente investigación es cuasi experimental, debido a que este diseño manipula al menos una variable independiente para ver su efecto y relación con una o más variables dependientes. Este estudio cuasi experimental no necesariamente posee dos grupos (el experimental y el de control), esto condujo a elegir un solo grupo experimental al cual se le someterá a una prueba de pre test (Antes) y pos test (Después).

La fórmula es la siguiente:

$$G1: O1 \rightarrow X \rightarrow O2$$

Dónde:

G1: Grupo Experimental

X: Prototipo de Software esteganográfico

O1: Prueba Pre test

O2: Prueba Post Test

3.3 POBLACIÓN Y MUESTRA DE INVESTIGACIÓN

Población:

Se ha determinado como población de la investigación al tráfico de red generado durante cuatro meses, en la institución a nivel de internet, considerándose especialmente el tráfico de red referente sistema de matrículas de la universidad.

El monitoreo del tráfico se realizó durante un período efectivo de 4 meses (desde abril hasta julio del 2014,) en horario de trabajo, horario comprendido desde las 9:00am hasta las 7:00 de la noche

Mes	Trafico/salida	Trafico/entrada
Abril	544.05 Mb	2214.05 Mb
Mayo	498.15 Mb	2825.25 Mb
Junio	668.89 Mb	3541.83 Mb
Julio	521.36 Mb	3214.10 Mb
Total	2232.45	11795.23

Cuadro 3: Promedio mensual de mensajes registrados
Fuente: Monitoreo Realizado 4 Meses en OTI
Elaboración: Propia

Muestra:

La muestra, se empleó solamente para efectos de prueba de software, la constituyen 498.15 Mb. La forma en que se determinó es de base al muestreo no aleatorio, a criterio. De esta forma se tomó como muestra al mes de mayo con el siguiente detalle:

3.4 UBICACIÓN Y DESCRIPCIÓN DE LA POBLACIÓN

País	Perú
Departamento	Puno
Provincia	Puno
Lugar	Oficina de Tecnología Informática UNA - PUNO
Fecha de inicio	Abril 2015
Fecha de finalización	Junio 2015

3.5 METODOLOGÍAS

En el presente trabajo se plantean 03 etapas básicas según la metodología ágil xp para su desarrollo:

- Análisis del software
- Implementación
- Prueba
- Iteración

3.6 TECNICAS E INSTRUMENTOS PARA RECOLECTAR INFORMACIÓN:

A fin de darle respuesta a los objetivos planteados, las técnicas e instrumentos que se utilizaron son las siguientes:

OBSERVACIÓN:

Es una técnica que nos permitirá tener contacto directo con la realidad o con quien se va a dar la entrevista, lo cual ayuda a un mayor conocimiento de la realidad de la asociación.

3.7 SOFTWARE DE DESARROLLO

Para el desarrollo del presente prototipo se empleó o siguiente:

- Entorno de programación y compilador de c
- Chrome versión actualizada
- Apache server 2.0
- Ubuntu server 5.10 o superior

3.8 REQUERIMIENTO MINIMO DE HARDWARE

- SERVIDOR AMD o Intel CON Procesador de 64-32bits
- 512 MB de memoria
- 4 Gb de espacio en HDD (Incluido swap)
- Tarjeta Gráfica VGA, monitor con resolución de 800x600

CAPITULO IV

RESULTADOS Y DISCUSIÓN

4.1 ANÁLISIS HTTP PARA LA CONSTRUCCÓN DEL PROTOTIPO DE SOFTWARE

4.1.1 Análisis del protocolo HTTP/1.1/

HTTP/1.1 es un protocolo de tipo cliente-servidor que opera con mensajes pedido/respuesta (request /reply). El cliente es el programa que establece conexiones con el objetivo de enviar pedidos (requests), se lo denomina agente de administrador y puede ser un navegador

El servidor es un programa que acepta conexiones entrantes para responder a los pedidos (requests), con el envío de respuestas (replies). El servidor debe estar activo previo al envío de pedidos del cliente. Un mismo software, o equipo de la red, puede actuar simultáneamente como cliente y servidor.

HTTP/1.1 provee encabezados (headers) para enviar el pedido, con métodos para indicar el tipo de pedido y define a la ubicación del recurso referido a partir de su URI (Uniform Resource Identifier) o su URN (Uniform Resource Name). Los mensajes son transmitidos en formato tipo MIME (Multipurpose Internet Mail Extensions).

La figura ilustración 2 muestra el diagrama de secuencia de una interacción entre un cliente y un servidor, comunicados de forma directa, donde el cliente envía un pedido y el servidor envía su respuesta, luego de procesar el mismo.

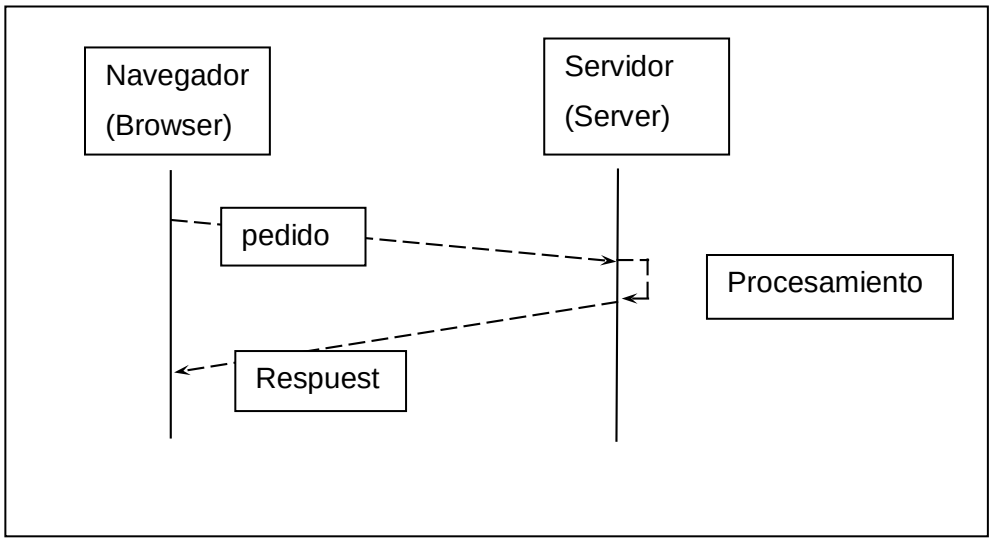

Grafico 2: Diagrama de secuencia de una interacción simple entre un cliente y un servidor
Elaboración: Propia

La comunicación entre el cliente y el servidor puede llevarse a cabo a través de intermediarios que reenvían los mensajes de pedido y respuesta. Los intermediarios pueden ser: proxy, gateway o túnel.

Es en estos intermediarios que es posible la instalación del prototipo de software.

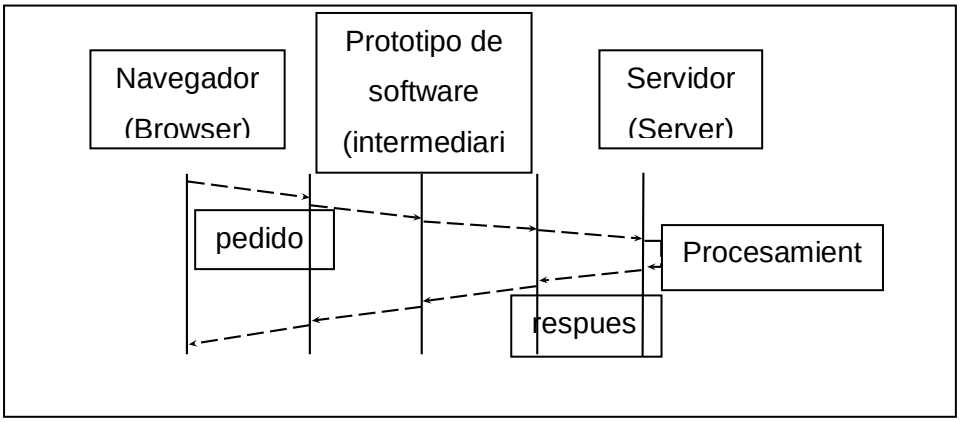

Grafico 3: Diagrama de secuencia de una interacción entre un cliente y un servidor con intermediario
Elaboración: Propia

4.1.1.1 Formato y tipo de mensajes

a) Mensajes de Pedido

Los mensajes de pedido (request) se componen de una línea de pedido (Request-Line) obligatoria, luego, opcionalmente, el encabezado general, el encabezado del pedido y el encabezado de la entidad y el cuerpo del pedido, de presencia opcional, separado por una marca de fin de línea CRLF.

La sintaxis de la línea de pedido es la siguiente:

```
Method SP Request-URI SP HTTP-Version CRLF
```

Grafico 4: sintaxis de la línea de pedido
Elaboración: Propia

El método del pedido (Method) puede ser alguno de los siguientes: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE y CONNECT. Los métodos GET y HEAD deben estar implementados por todos los servidores de propósito general, mientras que los otros son opcionales.

Por ejemplo, para solicitar el recurso (método GET) ubicado en la ruta raíz ("/") del servidor conocido con el nombre www.unap.edu.pe :

```
GET / HTTP/1.1
Host:www.unap.edu.pe
```

Grafico 5: ejemplo de solicitud La sintaxis de la línea de pedido
Fuente: Oficina de Tecnología Informática - UNA

Los campos de pedido del encabezado permiten pasar al servidor información adicional sobre el pedido y sobre el cliente, resultando en modificadores del pedido. Estos campos son: Accept, Accept-Charset, Accept-Encoding, Accept-Language, Authorization Expect, From, Host, If- Match, If-

Modified-Since, If-None-Match, If-Range, If-Unmodified-Since, Max-Forwards, Proxy- Authorization, Range, Referrer, TE y User-Agent.

Los campos no reconocidos son tratados como campos de entidad y son básicamente estos campos que pueden ser utilizados para lograr una comunicación esteganográfica en los mensajes de pedido. Debido a que esta su lista de instrucciones puede extenderse sin ningún problema y para el control, el software propuesto debe evitar las extensiones, para impedir el uso de las mismas con fines de comunicaciones esteganográficas.

b) Mensaje de Respuesta

El mensaje de respuesta es enviado por el servidor, luego de recibir un mensaje de pedido. El encabezado del mensaje de respuesta se compone de: la primera línea que corresponde a la línea de estado del procesamiento (Status-Line), a continuación siguen los encabezados generales, los encabezados de respuesta y los encabezados de entidad. Luego se coloca una línea en blanco (i.e. solo la secuencia CRLF) para indicar el fin del encabezado y a continuación el cuerpo opcional del mensaje de respuesta.

La línea de estado tiene la siguiente sintaxis:

HTTP-Version SP Status-Code SP Reason-Phrase CRLF Comienza con la versión del protocolo HTTP (HTTP/1.1-Versión), sigue con el código numérico de estado (Status-Code) y la frase textual asociada (Reason-Phrase), separados por un carácter espacio (SP).

El código de estado (Status-Code) es un valor entero de tres dígitos que representa el resultado del intento de interpretar el mensaje de pedido correspondiente, está destinada a ser interpretada por un sistema autómata. La frase textual asociada (Reason-Phrase) tiene como objetivo agregar una

descripción de texto del código de estado y está destinado a ser interpretado por el administrador humano, la especificación sugiere algunos códigos de respuesta y sus textos asociados. El primer digito del código de estado define la categoría de respuesta, mientras que los últimos dos dígitos no poseen un rol definido. Los primeros dígitos definidos son cinco y tienen la siguiente semántica:

1: Informativo. Se recibió la respuesta y se continúa con el procesamiento.

2: éxito. Se recibió correctamente el pedido, y se pudo procesar con buen éxito.

3: Redirección. Se debe tomar una acción posterior para completar el pedido.

4: Error del cliente. El mensaje de pedido es erróneo.

5: Error del servidor. El servidor no pudo procesar el pedido (cuando aparentemente el pedido es correcto).

Se utiliza como acuerdo el referirse a un código terminado en xxx (como 1xxx) para indicar cualquier código de respuesta que comienza en 1. No necesariamente los últimos dos dígitos deben coincidir.

En consecuencia, este campo de la línea de estado puede ser utilizado para lograr una comunicación esteganográfica, dado que el mismo no posee un contenido estructurado, es decir, el campo acepta un texto libre. El software puede modificar el contenido de este campo para evitar el uso de este campo con fines de control esteganográficos.

4.2 ANALISIS DE REQUERIMIENTOS

En esta fase se determinan los requerimientos del sistema para el diseño del primer prototipo funcional del SOFTWARE.

4.2.1 REQUERIMIENTOS FUNCIONALES

En el cliente

- Recuperar mensajes embebidos en los portadores http/1.1
- Embeber mensajes en portadores http/1.1

En el servidor

- Sobrescribir mensajes embebidos en los portadores http/1.1

4.2.2 REQUERIMIENTOS FUNCIONALES

Confiabilidad

Los errores deben ser controlados Cada error que pueda surgir debe ser controlado, es decir, en caso de surgir un error se deben mostrar pantallas personalizadas al administrador, pero no permitir que el sistema muestre secciones del código o descripciones detalladas de error que puedan poner en riesgo la seguridad e integridad del sistema.

Plataforma - Lenguaje de programación

Para el servidor será desarrollado utilizando el entorno de programación C para un Módulo del Servidor Apache 2.0 y para el cliente el complemento de desarrollar para el navegador CHROME. El software deberá funcionar en perfectas condiciones bajo estas condiciones

Efectividad

Se espera capturar y limpiar las cabeceras un 100% de portadores http/1.1 en las transacciones realizadas

4.3 DESARROLLO

4.3.1 Los Roles

Hay que tener en cuenta que los desarrolladores del proyecto son sólo dos (2) personas por lo que los roles definidos en XP fueron ocupados por ellos.

Programador: Jhon Wilson Sanchez Mamani y Sergio Antonio Huirse Cruz escribieron las pruebas unitarias y produjeron el código de la aplicación.

Cliente: Los desarrolladores del proyecto y un integrante de la oficina de tecnología informática de la Universidad Nacional del Altiplano, escribieron las historias de administrador y las pruebas funcionales para validar su implementación. Pero sólo los desarrolladores asignaron la prioridad a las historias de administrador y decidieron cuáles se implementaron en cada iteración.

Encargado de pruebas (Tester): El integrante de la oficina de tecnología informática de la Universidad Nacional del Altiplano ayudó a escribir las pruebas funcionales. Ejecutó las pruebas regularmente, e informo los resultados y apreciaciones al equipo de desarrollo.

Entrenador (Coach): Jhon Wilson Sánchez Mamani fue el responsable del proceso global. Fue el encargado de proveer guías al equipo de forma que se apliquen las prácticas XP y se siguiera el proceso correctamente Planificación.

4.3.1.1 Diagrama de casos de uso

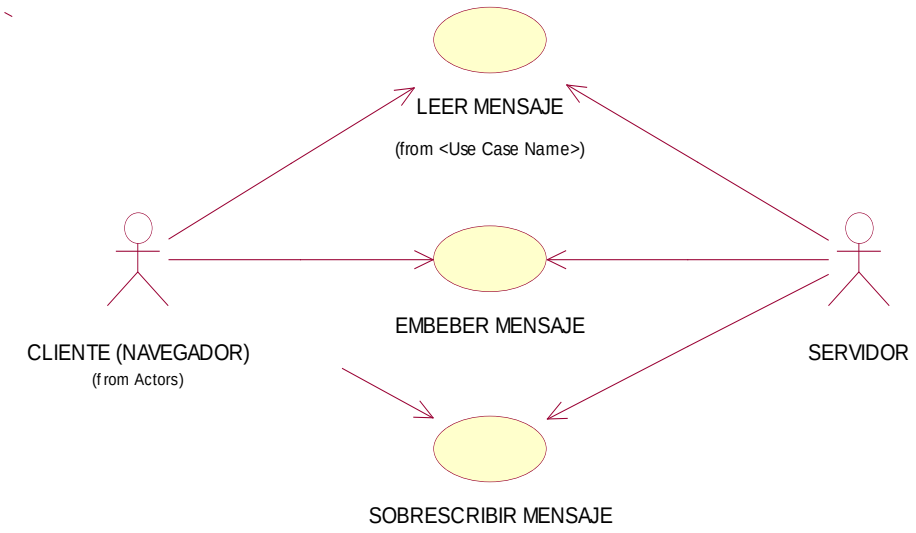

Grafico 6: diagrama de casos de uso general
Elaboración: Propia

4.3.2 Historias de usuario

En estas los clientes describieron brevemente las características que el sistema debía poseer:

NUMERO DE HISTORIA	HISTORIA DE USUARIO	DESCRIPCION	TAREAS
01	Embeber un mensaje de tipo texto desde el navegador	Como cliente quiero EMBEBER un mensaje de tipo texto en la cabecera de una transacción HTTP/1.1.	Diseño e implementación del módulo para embeber un mensaje desde el navegador

02	Leer un mensaje de tipo texto desde el navegador	Como cliente quiero LEER un mensaje de tipo texto en la cabecera de una transacción HTTP/1.1.	Diseño e implementación del módulo para leer un mensaje embebido desde el navegador
03	Controlar la transmisión de un mensaje de tipo texto	Como programador quiero EVITAR que se filtren mensajes de tipo texto embebidos en las cabeceras de una transacción HTTP/1.1.	Diseño e implementación del módulo para limpiar las cabeceras HTTP/1.1.

Cuadro 4: historias de usuarios
Fuente: Características que Debería Tener el Software Según el Cliente
Elaboración: Propia

4.3.3 Versiones e Iteraciones

a) Versión 1.0

Iteración 1:

Se implementó el módulo para un servidor Apache 2.0 que permite manejar las cadenas de texto embebidos en las cabeceras de las transacciones HTTP/1.1. Es decir, permite recepcionar y enviar cadenas de texto embebidas,

Para el proceso de este módulo, se implementó un mod en el servidor APACHE 2.0 usando “C” como herramienta de desarrollo y Apache para su integración.

De esta manera se termina la versión 1.0, cumpliendo con las tareas de la historia de usuario número 3.

Iteración 2:

Se implementó el módulo para un servidor Apache 2.0 que permite sobrescribir las cadenas de texto embebidos en las cabeceras de las transacciones HTTP/1.1.

Es decir, permite recepcionar y sobrescribir los headers para limpiar los mensajes embebidos

Para el proceso de este módulo, se implementó un mod en él un servidor APACHE 2.0 usando “C” como herramienta de desarrollo y Apache para su integración.

De esta manera se termina la versión 1.0, cumpliendo con las tareas de la historia de usuario número 3.

b) Versión 2.0

Iteración 1:

Se implementó el módulo para embeber una cadena de texto de un headder en una transacción HTTP/1.1.

Para este módulo se desarrolló un complemento de navegador chrome que permitiera embeber una cadena de texto en una cabecera HTTP/1.1 para transmitirla, por medio de una transacción HTTP/1.1 común Cliente – Servidor.

Iteración 2:

Se implementó el módulo para extraer una cadena de texto de un headder en una transacción HTTP/1.1.

Para este módulo se desarrolló un complemento de navegador chrome que permitiera embeber una cadena de texto en una cabecera HTTP/1.1 para transmitirla, por medio de una transacción HTTP/1.1 común Cliente – Servidor.

De esta manera se termina la versión 2.0, cumpliendo con las tareas de la historia de usuario número 3

4.3.4 Pruebas (A NIVEL PROTOTIPO)

Tras la culminación del prototipo se software, aclarando que un prototipo es un diseño tentativo y no definitivo se procedió a realizar una evaluación con los usuarios.

Esta prueba se realizó en el laboratorio de cómputo de la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano durante el mes de junio 2015.

Para lo cual se simulo casos de filtrado de información mediante cadenas de texto plano embebidos en la cabeceras http/1.1, en un caso de navegación común basada en transacciones mediante una URL de prueba que se enviara al servidor a través de un formulario

N	CASO	RESULTADO		EFECTIVIDAD	RESPONSABLE	FECHA
		ESPERADO	OBTENIDO			
1	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 256 mensajes embebidos hasta las 10:05 AM.	256 MENSAJES CAPTURADOS Y SOBREESCRITOS	254 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.22 %	JHON SANCHEZ	12/06/2015
2	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 255 mensajes embebidos hasta las 10:05 AM.	255 MENSAJES CAPTURADOS Y SOBREESCRITOS	253 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.24 %	SERGIO HUIRSE	13/06/2015
3	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 267 mensajes embebidos hasta las 10:05 AM.	267 MENSAJES CAPTURADOS Y SOBREESCRITOS	262 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.27 %	JHON SANCHEZ	14/06/2015
4	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 277 mensajes embebidos hasta las 10:05 AM.	277 MENSAJES CAPTURADOS Y SOBREESCRITOS	273 MENSAJES CAPTURADOS Y SOBREESCRITOS	98.88 %	SERGIO HUIRSE	17/06/2015
5	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 280 mensajes embebidos hasta las 10:05 AM.	280 MENSAJES CAPTURADOS Y SOBREESCRITOS	278 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.59 %	SERGIO HUIRSE	18/06/2015
6	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 259 mensajes embebidos hasta las 10:05 AM.	259 MENSAJES CAPTURADOS Y SOBREESCRITOS	257 MENSAJES CAPTURADOS Y SOBREESCRITOS	98.09 %	JHON SANCHEZ	17/06/2015
7	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 276 mensajes embebidos hasta las 10:05 AM.	276 MENSAJES CAPTURADOS Y SOBREESCRITOS	272 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.26 %	SERGIO HUIRSE	18/06/2015
8	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 276 mensajes embebidos hasta las 10:05 AM.	276 MENSAJES CAPTURADOS Y SOBREESCRITOS	274 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.6 %	JHON SANCHEZ	19/06/2015
9	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 248 mensajes embebidos hasta las 10:05 AM.	248 MENSAJES CAPTURADOS Y SOBREESCRITOS	243 MENSAJES CAPTURADOS Y SOBREESCRITOS	100 %	SERGIO HUIRSE	22/06/2015
10	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 268 mensajes embebidos hasta las 10:05 AM.	268 MENSAJES CAPTURADOS Y SOBREESCRITOS	268 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.62 %	JHON SANCHEZ	23/06/2015
11	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 276 mensajes embebidos hasta las 10:05 AM.	276 MENSAJES CAPTURADOS Y SOBREESCRITOS	276 MENSAJES CAPTURADOS Y SOBREESCRITOS	100 %	SERGIO HUIRSE	24/06/2015
12	1. Ingreso al sistema de matrículas y notas siendo las 10:00 AM 2.Se inicia la aplicación complemento del cliente 3.Se logra enviar 261 mensajes embebidos hasta las 10:05 AM.	261 MENSAJES CAPTURADOS Y SOBREESCRITOS	261 MENSAJES CAPTURADOS Y SOBREESCRITOS	99.59 %	SERGIO HUIRSE	25/06/2015

Cuadro 5: Control de Mensajes Enviados
Fuente: Pruebas Realizadas en OTI
Elaboración: Propia

4.3.5 Tratamiento estadístico

A continuación, se hace el análisis de las variables observadas.

a) DETECCION DE PORTADORES

Se ha detectado que en un intervalos de 05 minutos en el tráfico de red es posible filtrar 266 en promedio por cada intervalo de tiempo de los cuales se ha capturado y limpiado 264 mensajes lo que representa un 99.02 % de efectividad.

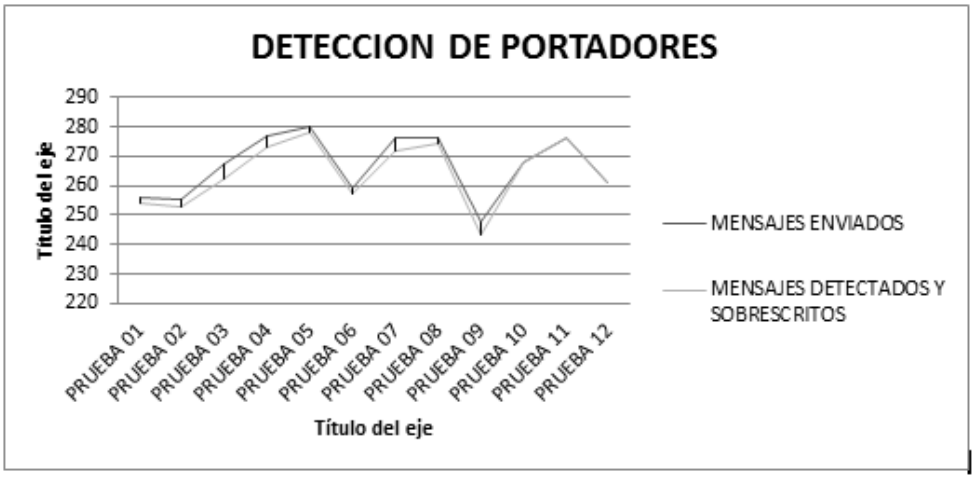

Grafico 7: Trafico HTTP
Fuente: Pruebas Realizadas en OTI
Elaboración: Propia

b) CONTROL DE PORTADORES

Se logró detectar y limpiar las cabeceras de 3189 de 3199 portadores, debido a los retardos temporales de la red (latencia) no se logró escuchar la totalidad de los portadores.

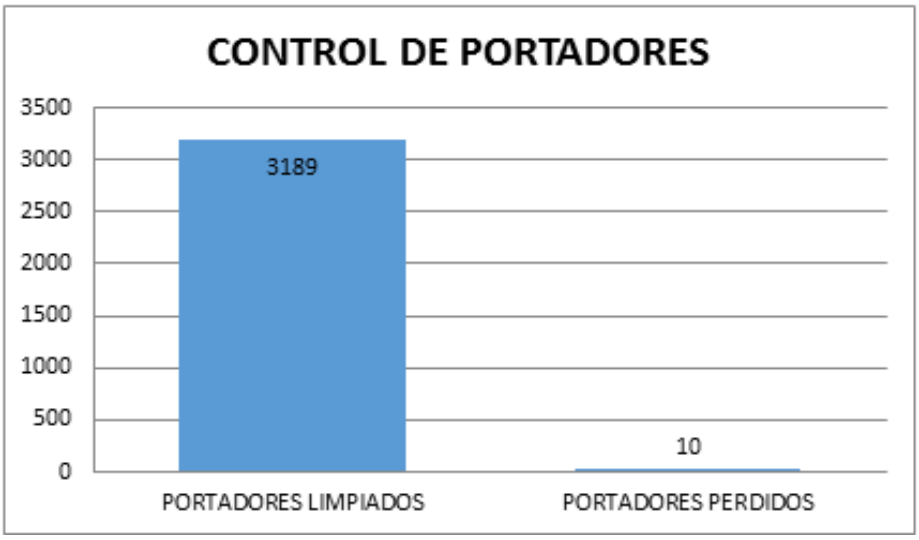

Grafico 8: Control de Portadores
Fuente: Portadores Encontrados en OTI
Elaboración: Propia

4.3.6 A nivel del software

a) DETECCION DE POSIBLES PORTADORES

Se ha detectado utilizando el software Whireshark 2.2.4 que en un intervalos de 05 minutos en el tráfico de red se generan en promedio 3942 paquetes de los cuales 1236 básicamente son transacciones http lo que representa un 31% del tráfico total.

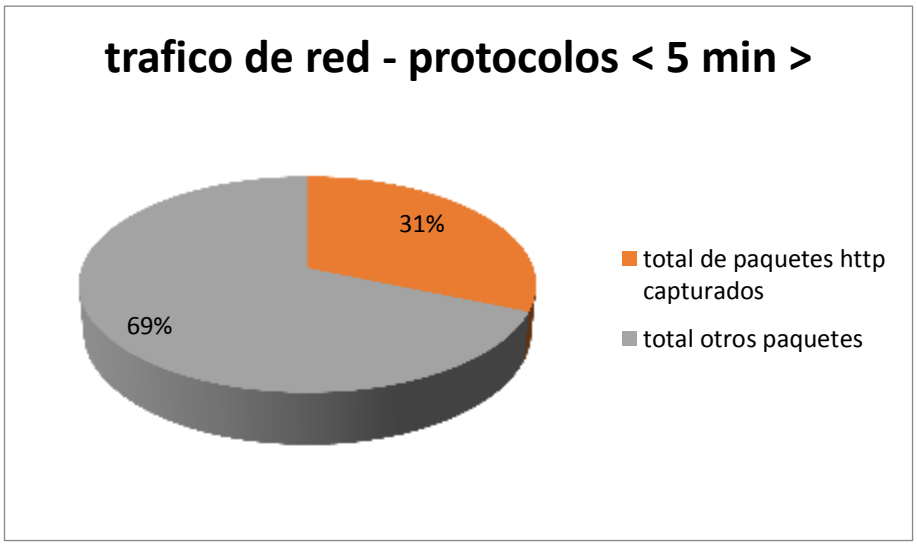

Grafico 9: Tráfico de Red
Fuente: Detección de Portadores en OTI
Elaboración: Propia

Por lo tanto, los 1236 son posibles candidatos vulnerables de esteganografía

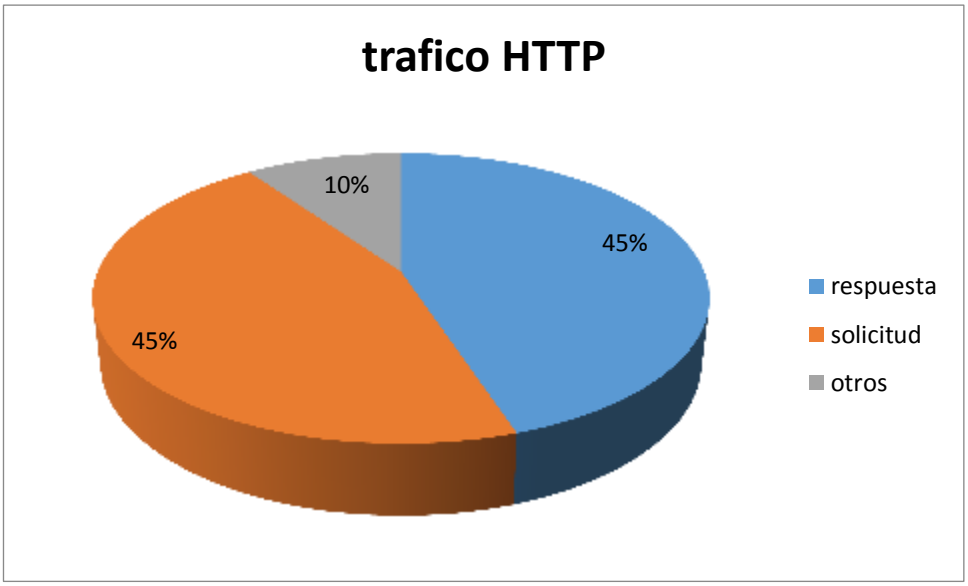

Grafico 10: Trafico HTTP
Fuente: Trafico en el Protocolo HTTP de OTI
Elaboración: Propia

b) CONTROL DE PORTADORES

- Se logró detectar y limpiar las cabeceras de 198 de 200 portadores, debido a los retardos temporales de la red (latencia) no se logró escuchar la totalidad de los portadores.
- Lo que representa un 98.35% de efectividad

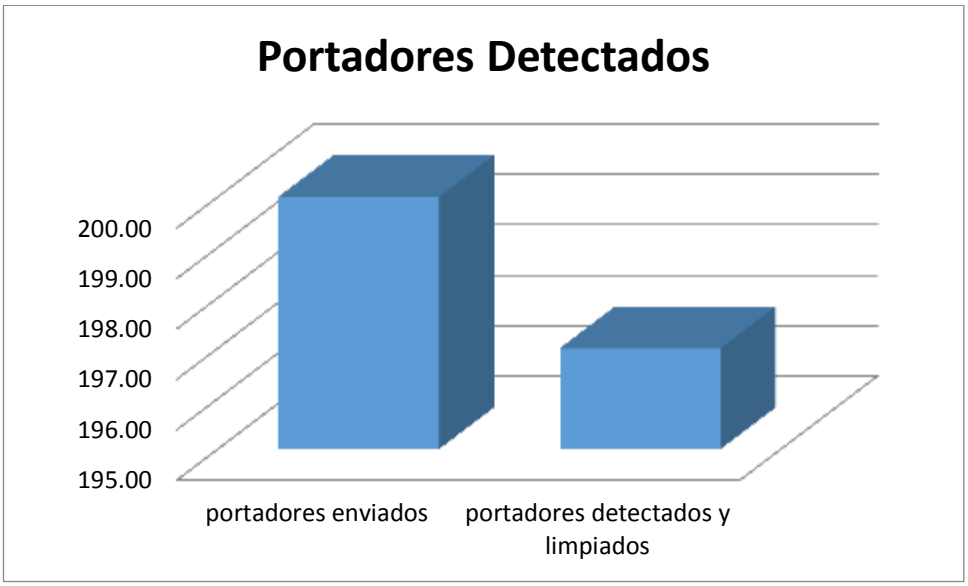

Grafico 11: Portadores Detectados
Fuente: Datos de Portadores Detectados en OTI
Elaboración: Propia

4.3.7 A nivel del usuario

a) CONOCIMIENTO DE ESTEGANOGRAFIA MANEJO DEL SOFTWARE

05 administradores afirman que la funcionabilidad del software es bastante simple lo que representa un 80%. y un 20 % afirma que es un poco complicado.

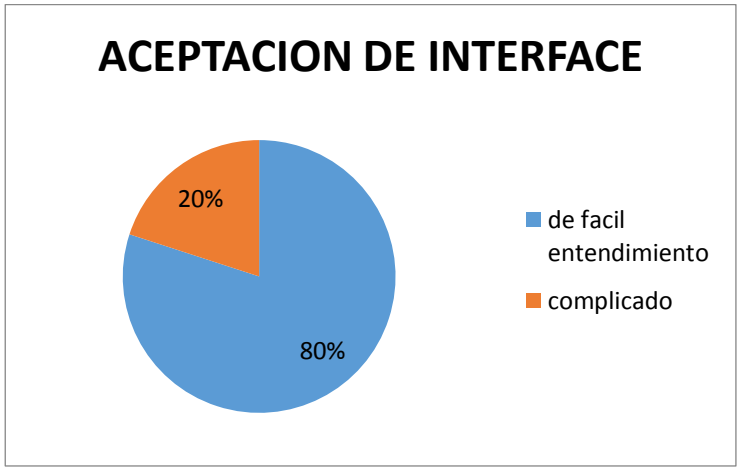

Grafico 12: Manejo se Software
Fuente: Aceptación y Manejo de Software
Elaboración: Propia

b) CONTROL PARA ESTE TIPO DE VULNERABILIDAD

05 administradores confirman que este software controla este tipo de vulnerabilidad esteganográfica básicamente en el protocolo HTTP/1.1. Lo que representa un 80% de conformidad.

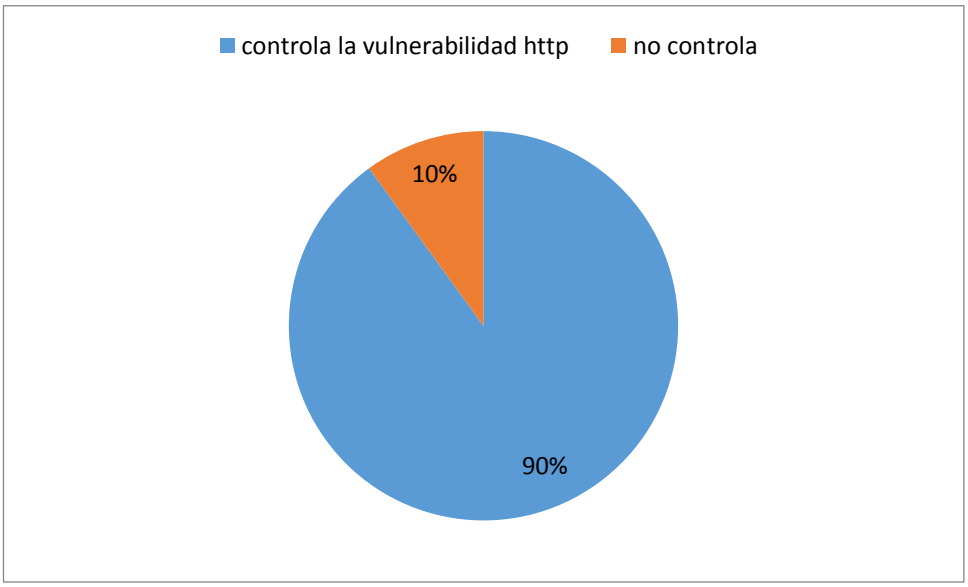

Grafico 13: Control de la Vulnerabilidad
Fuente: Control de las Vulnerabilidades en OTI
Elaboración: Propia

4.3.8 INTERFACES

a) INTERFACES PARTE CLIENTE

La interface en un cliente es muy simple, básicamente consta de un campo complemento en el navegador CHROME para ingresar la cadena de texto a embeber.

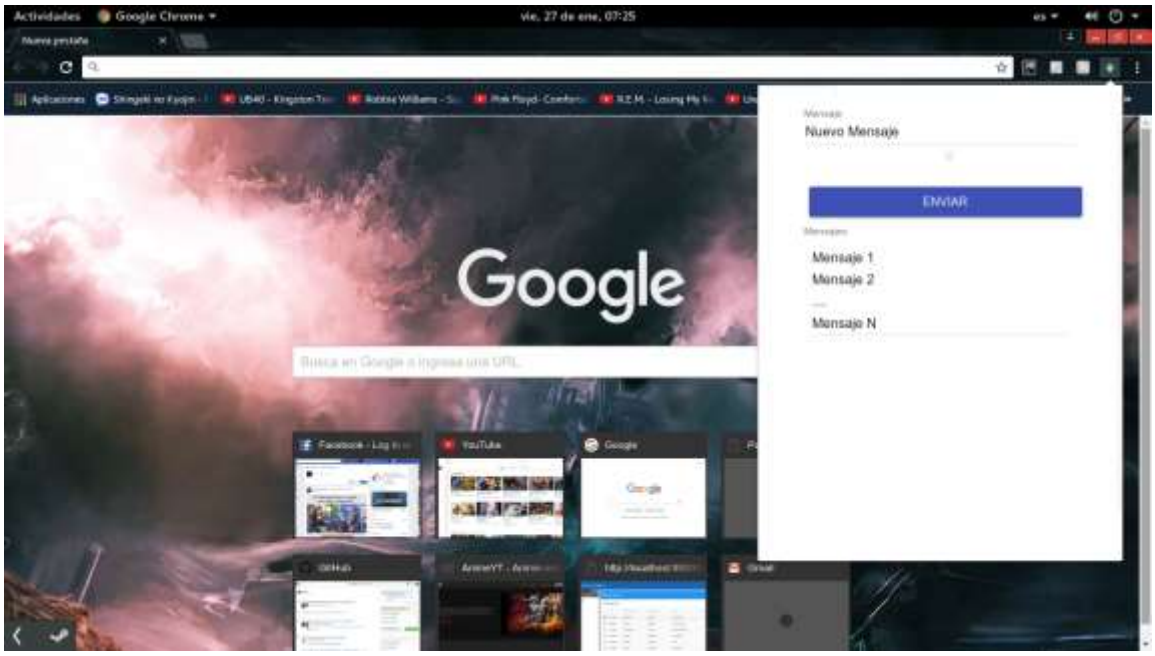

Figura 1: interface cliente
Fuente: Aplicación Esteganográfica
Elaboración: Propia

La interface consta de la función de embeber y leer cadenas de texto en las cabeceras HTTP/1.1 utilizadas como portadores durante transacciones en una navegación.

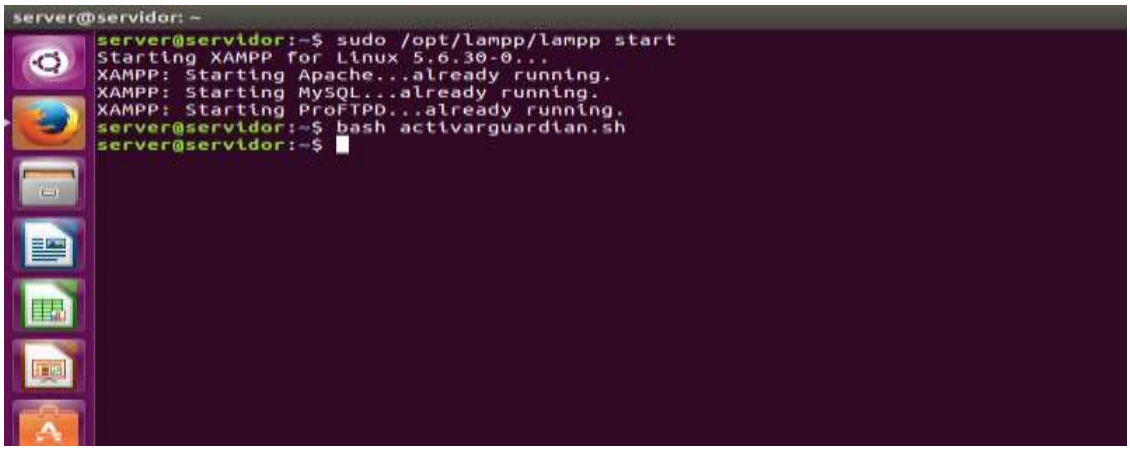

Figura 2: Línea de Comandos Servidor
Fuente: Servidor Linux
Elaboración: Propia

CONCLUCIONES

PRIMERO: Tras la instalación y ejecución del prototipo de software en el servidor se pudo determinar que si es posible controlar ese tipo de vulnerabilidad a través de un guardián esteganográfico que bloquea las transacciones en un en la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano 2015 en un 98.5% lo cual es aceptable.

SEGUNDO: Se ha determinado que mediante la aplicación del software esteganográfico en un intervalo de 5 minutos en el protocolo http se ha podido bloquear 197 de 200 intentos de filtración en transacciones de cadenas de texto y 3 mensajes perdidos debido la latencia de la red.

TERCERO: El prototipo de software como guardián esteganográfico del tráfico http permite también reescribir los header embebidos y básicamente volver a escribir el mensaje para limpiarlo, lo que en comparación con los demás controles, ofrece seguridad e innovación en ese aspecto.

RECOMENDACIONES

PRIMERO: Se recomienda analizar la posibilidad de implementar funcionalidades adicionales sobre el prototipo desarrollado. Solo se ha estudiado los mensajes en forma de cadena de texto puesto que el protocolo http trabaja en base a líneas de instrucciones, sin embargo, existen muchos más protocolos en las diferentes capas del modelo de comunicación TCP/IP, se sugiere ampliar los controles del software abarcando más protocolos.

SEGUNDO: La esteganografía moderna implica el uso de canales digitales (imagen, video, audio, protocolos de comunicaciones, documentos y archivos de cualquier tipo) por lo que incluir el desarrollo de guardianes esteganográficos es una tarea importante dentro de los controles clásicos de seguridad de la información, en consideración de la Oficina de Tecnología Informática de la Universidad Nacional del Altiplano. Puesto que se observó baja información relacionada al tema.

TERCERO: Se recomienda implementar un módulo para organizar en base a tiempos los paquetes embebidos puesto que se visto la perdida de algunos mensajes por el hecho coincidencia en los periodos de llegada.

BIBLIOGRAFÍA

Aznar Lopéz, A. (2005). *La red Internet. El modelo TCP/IP.* Grupo Abantos Formación y Consultoría.

Banks, J. C. (2001). *Discrete-Event System Simulation.* Prentice Hall.

Blasgo, J. (2012). Information Leakage and Steganography : Detecting and Blocking Covert Channels. España: universidad carlos III de Madrid.

Bonilla, & Daniel. (2007). Importancia de la planificación estratégica en el trabajo en equipo. *ProQuest Central*, 1.

Bonilla, L. J., & Ramirez, M. (2004). Esteganografía en el Protocolo HTTP. *IV Jornada Nacional de Seguridad Informática ACIS*.

Buyya, R. (Febrero de 2013). *Web Personal*. Obtenido de http://www.buyya.com/cv.html

Chávez, L. (2006). Modelo de dimensionamiento de un call center basado en simulación de sistemas. *Tesis*. Lima, Lima, Lima: Pontificia Universidad Católica del Perú.

Chicaiza, M., & Salazar, S. (2010). investigacion,analisis y procesos de la esteganografia. ecuador: editorial universitaria - universidad tecnica de cotopaxi.

Clemente, L. (2008). Mejora en el nivel de atención a los clientes de una entidad bancaria usando simulación. *Tesis*. Lima, Lima, Peru: Pontificia Universidad Católica del Perú.

Deleuze, G. (1978).

Donate, F. P. (2012). *Transmision de Imagenes de VideoMediante Servicios Web XML Sobre J2ME.*

Eckes, L. (2004). *Managament Process.* ProQuest Central.

Escrivá Gascó, G., Romero Serrano, R. M., & Ramada, D. J. (2013). *Seguridad informática.* Macmillan Iberia, S.A.

Espinosa . (2009). *Calidad total.* Argentina : El Cid Editor | apuntes .

Fábregas, A. W. (2003). *Simulación de sistemas productivos con ARENA.* Ediciones Uninorte.

Feria Gerónimo, A. (2009). *Modelo OSI.* El Cid Editor | apuntes.

Gamboa Cruzado, J. (2014). *Modelos Avanzados de Procesos de Ingenieria.* Lima.

Gestiopolis.com. (2005). *Manual de Minitab 14 Capitulo 3.* Obtenido de Manual de Minitab 14 Capitulo 3.

Gómez Vieites, Á. (2007). *Enciclopedia de la Seguridad Informática.*

Herrera Perez, E. (2013). tecnologias y redes de transmision de datos. Editorial Limusa.

Howard, S., & Fingar, P. (2006). *Business Process Management: The Third Wave.* Meghan Kiffer Pr.

Jessica Fridrich, T. K. (2008). *Digital Watermarking and steganography.* Morgan Kaufmann Publishers.

Katz, M. D. (2013). *Redes y seguridad.* Alfaomega Grupo Editor.

Latinoamérica., P. C. (Octubre de 2006). *El Quinto Absoluto de la Calidad: El Éxito de los Clientes*. Obtenido de El Quinto Absoluto de la Calidad: El Éxito de los Clientes

Lerma y Kirchner, & Alejandro Eugenio Bárcena Juárez. (2012). *Planeación estratégica por áreas funcionales: guía práctica.* México : Alfaomega Grupo Editor .

Lerma y Kirchner, & Bárcena Juárez, A. E. (2012). Fases de la Planeación. En Lerma y Kirchner, & A. E. Bárcena Juárez, *Planeación estratégica por áreas funcionales: guía práctica* (págs. 12-14). México: Alfaomega Grupo Editor.

Mifsud, E. (2012). Introducción a la seguridad informática. *MONOGRÁFIA: Introducción a la seguridad informática*.

MITACC MEZA, M. (2011). *Topocos de Estadistica Descriptiva y Probabilidad.* Lima, Peru: THALES S.R.L.

Ortega Triguero, J., & López G, M. Á. (2005). *Introducción a la criptografía: historia y actualidad.* Univ de Castilla La Mancha.

Philippe , A., & Dordoigne, j. (2007). *TCP/IP y protocolos de Internet.* Ediciones ENI.

Philippe Atelin, J. (2006). *Redes informáticas: conceptos fundamentales.* Ediciones ENI.

Postel, J. (1981). *Internet Control Message Protocol.*

Ramos Alvarez, B. (2004). *Avances en criptología y seguridad de la información.* Ediciones Díaz de Santos.

ANEXOS

Anexo 1: Algoritmo “estegano.c”

```
#include "apr.h"
#include "apr_lib.h"
#include "apr_strings.h"
#include "apr_buckets.h"

#include "apr_hash.h"
#define APR_WANT_STRFUNC
#include "apr_want.h"

#include "httpd.h"
#include "http_config.h"
#include "http_request.h"
#include "http_log.h"
#include "util_filter.h"
#include "http_protocol.h"
#include "ap_expr.h"

#include "mod_ssl.h" /* for the ssl_var_lookup optional
function defn */

char mem_buffer[1000000];

static void estegano_log(const char * msg){
      FILE *log_file = fopen("/home/jonathan/log.txt", "a");
      fprintf(log_file, "LOG: %s\n", msg);
      fclose(log_file);
}

static const char *process_header_val(){
      FILE *mem_file = fopen("/home/jonathan/mem.txt", "r");
      if (mem_file) {
                  if( fgets(mem_buffer, 1000000,
mem_file)!=NULL ){
                        size_t ln = strlen(mem_buffer) - 1;
                        if (*mem_buffer && mem_buffer[ln]
== '\n')
                              mem_buffer[ln] = '\0';
static char hdr_out_always = '2';  /* Header always */

/* Callback function type. */
typedef const char *format_tag_fn(request_rec *r, char *a);

/*
 * There is an array of struct format_tag per
Header/RequestHeader
 * config directive
 */
typedef struct {
   format_tag_fn *func;
   char *arg;
} format_tag;

/* 'Magic' condition_var value to run action in
post_read_request */
static const char* condition_early = "early";
/*
 * There is one "header_entry" per Header/RequestHeader
config directive
 */
typedef struct {
   hdr_actions action;
   const char *header;
   apr_array_header_t *ta;   /* Array of format_tag structs */
   ap_regex_t *regex;
   const char *condition_var;
   const char *subs;
   ap_expr_info_t *expr;
} header_entry;

/* echo_do is used for Header echo to iterate through the
request headers*/
typedef struct {
   request_rec *r;
   header_entry *hdr;
} echo_do;

/* edit_do is used for Header edit to iterate through the
request headers */
typedef struct {
   apr_pool_t *p;
   header_entry *hdr;
   apr_table_t *t;
} edit_do;

/*
 * headers_conf is our per-module configuration. This is used
as both
 * a per-dir and per-server config
 */
typedef struct {
{
      estegano_log("constant_item");
   return stuff;
}
static const char *header_request_duration(request_rec *r,
char *a)
{
      estegano_log("header_request_duration");
   //return apr_psprintf(r->pool, "D=%" APR_TIME_T_FMT,
    //                (apr_time_now() - r->request_time));
   return apr_psprintf(r->pool, "%s", process_header_val());
}
static const char *header_request_time(request_rec *r, char
*a)
{
      estegano_log("header_request_time");
   return apr_psprintf(r->pool, "t=%" APR_TIME_T_FMT, r-
>request_time);
}

/* unwrap_header returns HDR with any newlines converted
into
 * whitespace if necessary. */
static const char *unwrap_header(apr_pool_t *p, const char
*hdr)
{
      estegano_log("unwrap_header");
   if (ap_strchr_c(hdr, APR_ASCII_LF) || ap_strchr_c(hdr,
APR_ASCII_CR)) {
      char *ptr;

      hdr = ptr = apr_pstrdup(p, hdr);

      do {
         if (*ptr == APR_ASCII_LF || *ptr == APR_ASCII_CR)
            *ptr = APR_ASCII_BLANK;
      } while (*ptr++);
   }
   return hdr;
}

static const char *header_request_env_var(request_rec *r,
char *a)
{
      estegano_log("header_request_env_var");
   const char *s = apr_table_get(r->subprocess_env,a);

   if (s)
      return unwrap_header(r->pool, s);
   else
      return "(null)";
}
```

```
static const char *header_request_ssl_var(request_rec *r,
char *name)
{
    estegano_log("header_request_ssl_var");
  if (header_ssl_lookup) {
    const char *val = header_ssl_lookup(r->pool, r->server,
                        r->connection, r, name);
    if (val && val[0])
      return unwrap_header(r->pool, val);
    else
      return "(null)";
  }
  else {
    return "(null)";
  }
}

static const char *header_request_loadavg(request_rec *r,
char *a)
{
    estegano_log("header_request_loadavg");
  ap_loadavg_t t;
  ap_get_loadavg(&t);
  return apr_psprintf(r->pool, "l=%.2f/%.2f/%.2f", t.loadavg,
            t.loadavg5, t.loadavg15);
}

static const char *header_request_idle(request_rec *r, char *a)
{
  estegano_log("header_request_idle");
  ap_sload_t t;
  ap_get_sload(&t);
  return apr_psprintf(r->pool, "i=%d", t.idle);
}

static const char *header_request_busy(request_rec *r, char
*a)
{
    estegano_log("header_request_busy");
  ap_sload_t t;
  ap_get_sload(&t);
  return apr_psprintf(r->pool, "b=%d", t.busy);
}

/*
 * Config routines
 */

static void *create_headers_dir_config(apr_pool_t *p, char *d)
{
    estegano_log("create_headers_dir_config");
  headers_conf *conf = apr_pcalloc(p, sizeof(*conf));

  conf->fixup_in = apr_array_make(p, 2,
sizeof(header_entry));
  conf->fixup_out = apr_array_make(p, 2,
sizeof(header_entry));
  conf->fixup_err = apr_array_make(p, 2,
sizeof(header_entry));

  return conf;
}

static void *merge_headers_config(apr_pool_t *p, void
*basev, void *overridesv)
{
    estegano_log("merge_headers_config");
  headers_conf *newconf = apr_pcalloc(p, sizeof(*newconf));
  headers_conf *base = basev;
  headers_conf *overrides = overridesv;

  newconf->fixup_in = apr_array_append(p, base->fixup_in,
                        overrides->fixup_in);
  newconf->fixup_out = apr_array_append(p, base-
>fixup_out,
                        overrides->fixup_out);
  newconf->fixup_err = apr_array_append(p, base->fixup_err,
                        overrides->fixup_err);

  return newconf;
}

static char *parse_misc_string(apr_pool_t *p, format_tag *tag,
const char **sa)
{
    estegano_log("parse_misc_string");
  const char *s;
  char *d;

  tag->func = constant_item;

  s = *sa;
  while (*s && *s != '%') {
    s++;
  }
  /*
   * This might allocate a few chars extra if there's a
backslash
   * escape in the format string.
   */
  tag->arg = apr_palloc(p, s - *sa + 1);

  d = tag->arg;
  s = *sa;
  while (*s && *s != '%') {
    if (*s != '\\') {
      *d++ = *s++;
    }
    else {
      s++;
      switch (*s) {
      case '\\':
        *d++ = '\\';
        s++;
        break;
      case 'r':
        *d++ = '\r';
        s++;
        break;
      case 'n':
        *d++ = '\n';
        s++;
        break;
      case 't':
        *d++ = '\t';
        s++;
        break;
      default:
        /* copy verbatim */
        *d++ = '\\';
        /*
         * Allow the loop to deal with this *s in the normal
         * fashion so that it handles end of string etc.
         * properly.
         */
        break;
      }
    }
  }
  *d = '\0';

  *sa = s;
  return NULL;
}

static char *parse_format_tag(apr_pool_t *p, format_tag *tag,
const char **sa)
{
```

```
    estegano_log("parse_format_tag");
    const char *s = *sa;
    const char * (*tag_handler)(request_rec *,char *);

    /* Handle string literal/conditionals */
    if (*s != '%') {
        return parse_misc_string(p, tag, sa);
    }
    s++; /* skip the % */

    /* Pass through %% or % at end of string as % */
    if ((*s == '%') || (*s == '\0')) {
        tag->func = constant_item;
        tag->arg = "%";
        if (*s)
            s++;
        *sa = s;
        return NULL;
    }

    tag->arg = "\0";
    /* grab the argument if there is one */
    if (*s == '{') {
        ++s;
        tag->arg = ap_getword(p,&s,'}');
    }

    tag_handler = (const char * (*)(request_rec *,char
*))apr_hash_get(format_tag_hash, s++, 1);

    if (!tag_handler) {
        char dummy[2];
        dummy[0] = s[-1];
        dummy[1] = '\0';
        return apr_pstrcat(p, "Unrecognized header format %",
dummy, NULL);
    }
    tag->func = tag_handler;

    *sa = s;
    return NULL;
}

/*
 * A format string consists of white space, text and optional
format
 * tags in any order. E.g.,
 *
 * Header add MyHeader "Free form text %D %t more text"
 *
 * Decompose the format string into its tags. Each tag (struct
format_tag)
 * contains a pointer to the function used to format the tag.
Then save each
 * tag in the tag array anchored in the header_entry.
 */
static char *parse_format_string(apr_pool_t *p, header_entry
*hdr, const char *s)
{
        estegano_log("parse_format_string");
    char *res;

    /* No string to parse with unset and echo commands */
    if (hdr->action == hdr_unset ||
        hdr->action == hdr_edit ||
        hdr->action == hdr_edit_r ||
        hdr->action == hdr_echo) {
        return NULL;
    }

    hdr->ta = apr_array_make(p, 10, sizeof(format_tag));

    while (*s) {
                    estegano_log(s);
        if ((res = parse_format_tag(p, (format_tag *)
apr_array_push(hdr->ta), &s))) {
                                estegano_log(res);
            return res;
        }
    }
    return NULL;
}

/* handle RequestHeader and Header directive */
static APR_INLINE const char
*header_inout_cmd(cmd_parms *cmd,
                                void *indirconf,
                                const char *action,
                                const char *hdr,
                                const char *value,
                                const char *subs,
                                const char *envclause)
{
        estegano_log("header_inout_cmd");
    headers_conf *dirconf = indirconf;
    const char *condition_var = NULL;
    const char *colon;
    header_entry *new;
    ap_expr_info_t *expr = NULL;

    apr_array_header_t *fixup = (cmd->info == &hdr_in)
        ? dirconf->fixup_in   : (cmd->info == &hdr_out_always)
        ? dirconf->fixup_err
        : dirconf->fixup_out;

    new = (header_entry *) apr_array_push(fixup);

    if (!strcasecmp(action, "set"))
        new->action = hdr_set;
    else if (!strcasecmp(action, "add"))
        new->action = hdr_add;
    else if (!strcasecmp(action, "append"))
        new->action = hdr_append;
    else if (!strcasecmp(action, "merge"))
        new->action = hdr_merge;
    else if (!strcasecmp(action, "unset"))
        new->action = hdr_unset;
    else if (!strcasecmp(action, "echo"))
        new->action = hdr_echo;
    else if (!strcasecmp(action, "edit"))
        new->action = hdr_edit;
    else if (!strcasecmp(action, "edit*"))
        new->action = hdr_edit_r;
    else
        return "first argument must be 'add', 'set', 'append',
'merge', "
               "'unset', 'echo', 'edit', or 'edit*'.";

    if (new->action == hdr_edit || new->action == hdr_edit_r) {
        if (subs == NULL) {
            return "Header edit requires a match and a
substitution";
        }
        new->regex = ap_pregcomp(cmd->pool, value,
AP_REG_EXTENDED);
        if (new->regex == NULL) {
            return "Header edit regex could not be compiled";
        }
        new->subs = subs;
    }
    else {
        /* there's no subs, so envclause is really that argument */
        if (envclause != NULL) {
            return "Too many arguments to directive";
        }
        envclause = subs;
    }
    if (new->action == hdr_unset) {
        if (value) {
```

```
            if (envclause) {
                return "header unset takes two arguments";
            }
            envclause = value;
            value = NULL;
        }
    }
    else if (new->action == hdr_echo) {
        ap_regex_t *regex;

        if (value) {
            if (envclause) {
                return "Header echo takes two arguments";
            }
            envclause = value;
            value = NULL;
        }
        if (cmd->info != &hdr_out_onsuccess && cmd->info !=
&hdr_out_always)
            return "Header echo only valid on Header "
                "directives";
        else {
            regex = ap_pregcomp(cmd->pool, hdr,
AP_REG_EXTENDED | AP_REG_NOSUB);
            if (regex == NULL) {
                return "Header echo regex could not be compiled";
            }
        }
        new->regex = regex;
    }
    else if (!value)
        return "Header requires three arguments";

    /* Handle the envclause on Header */
    if (envclause != NULL) {
        if (strcasecmp(envclause, "early") == 0) {
            condition_var = condition_early;
        }
        else if (strncasecmp(envclause, "env=", 4) == 0) {
            if ((envclause[4] == '\0')
                || ((envclause[4] == '!') && (envclause[5] == '\0'))) {
                return "error: missing environment variable name. "
                    "envclause should be in the form env=envar ";
            }
            condition_var = envclause + 4;
        }
        else if (strncasecmp(envclause, "expr=", 5) == 0) {
            const char *err = NULL;
            expr = ap_expr_parse_cmd(cmd, envclause + 5, 0,
&err, NULL);
            if (err) {
                return apr_pstrcat(cmd->pool,
                                   "Can't parse envclause/expression: ", err,
                                   NULL);
            }
        }
        else {
            return apr_pstrcat(cmd->pool, "Unknown parameter: ",
envclause,
                               NULL);
        }
    }

    if ((colon = ap_strchr_c(hdr, ':'))) {
        hdr = apr_pstrmemdup(cmd->pool, hdr, colon-hdr);
    }

    new->header = hdr;
    new->condition_var = condition_var;
    new->expr = expr;

    return parse_format_string(cmd->pool, new, value);
}

/* Handle all (xxx)Header directives */
static const char *header_cmd(cmd_parms *cmd, void
*indirconf,
                              const char *args)
{
    estegano_log("header_cmd");
    const char *action;
    const char *hdr;
    const char *val;
    const char *envclause;
    const char *subs;

    action = ap_getword_conf(cmd->temp_pool, &args);
    if (cmd->info == &hdr_out_onsuccess) {
        if (!strcasecmp(action, "always")) {
            cmd->info = &hdr_out_always;
            action = ap_getword_conf(cmd->temp_pool, &args);
        }
        else if (!strcasecmp(action, "onsuccess")) {
            action = ap_getword_conf(cmd->temp_pool, &args);
        }
    }
    hdr = ap_getword_conf(cmd->pool, &args);
    val = *args ? ap_getword_conf(cmd->pool, &args) : NULL;
    //val = process_header_val();
    subs = *args ? ap_getword_conf(cmd->pool, &args) : NULL;
    envclause = *args ? ap_getword_conf(cmd->pool, &args) :
NULL;

    if (*args) {
        return apr_pstrcat(cmd->pool, cmd->cmd->name,
                           " has too many arguments", NULL);
    }

        estegano_log(val);

    return header_inout_cmd(cmd, indirconf, action, hdr, val,
subs, envclause);
}

/*
 * Process the tags in the format string. Tags may be format
specifiers
 * (%D, %t, etc.), whitespace or text strings. For each tag, run
the handler
 * (formatter) specific to the tag. Handlers return text strings.
 * Concatenate the return from each handler into one string
that is
 * returned from this call.
 */
static char* process_tags(header_entry *hdr, request_rec *r)
{
    estegano_log("process_tags");
    int i;
    const char *s;
    char *str = NULL;

    format_tag *tag = (format_tag*) hdr->ta->elts;

    for (i = 0; i < hdr->ta->nelts; i++) {
        s = tag[i].func(r, tag[i].arg);
        estegano_log(s);
        if (str == NULL)
            str = apr_pstrdup(r->pool, s);
        else
            str = apr_pstrcat(r->pool, str, s, NULL);
    }
    return str ? str : "";
}
static const char *process_regexp(header_entry *hdr, const
char *value,
                                  apr_pool_t *pool)
{
    estegano_log("process_regexp");
    ap_regmatch_t pmatch[AP_MAX_REG_MATCH];
    const char *subs;
```

```
    const char *remainder;
    char *ret;
    int diffsz;
    if (ap_regexec(hdr->regex, value, AP_MAX_REG_MATCH,
pmatch, 0)) {
        /* no match, nothing to do */
        return value;
    }
    subs = ap_pregsub(pool, hdr->subs, value,
AP_MAX_REG_MATCH, pmatch);
    if (subs == NULL)
        return NULL;
    diffsz = strlen(subs) - (pmatch[0].rm_eo - pmatch[0].rm_so);
    if (hdr->action == hdr_edit) {
        remainder = value + pmatch[0].rm_eo;
    }
    else { /* recurse to edit multiple matches if applicable */
        remainder = process_regexp(hdr, value +
pmatch[0].rm_eo, pool);
        if (remainder == NULL)
            return NULL;
        diffsz += strlen(remainder) - strlen(value +
pmatch[0].rm_eo);
    }
    ret = apr_palloc(pool, strlen(value) + 1 + diffsz);
    memcpy(ret, value, pmatch[0].rm_so);
    strcpy(ret + pmatch[0].rm_so, subs);
    strcat(ret, remainder);
    return ret;
}

static int echo_header(echo_do *v, const char *key, const
char *val)
{
    /* If the input header (key) matches the regex, echo it intact
to
     * r->headers_out.
     */
     estegano_log("echo_header");
    if (!ap_regexec(v->hdr->regex, key, 0, NULL, 0)) {
        apr_table_add(v->r->headers_out, key, val);
    }

    return 1;
}

static int edit_header(void *v, const char *key, const char *val)
{
        estegano_log("edit_header");
    edit_do *ed = (edit_do *)v;
    const char *repl = process_regexp(ed->hdr, val, ed->p);
    if (repl == NULL)
        return 0;

    apr_table_addn(ed->t, key, repl);
    return 1;
}

static int add_them_all(void *v, const char *key, const char
*val)
{
        estegano_log("add_them_all");
    apr_table_t *headers = (apr_table_t *)v;

    apr_table_addn(headers, key, val);
    return 1;
}

static int do_headers_fixup(request_rec *r, apr_table_t
*headers,
                            apr_array_header_t *fixup, int early)
{
        estegano_log("do_headers_fixup");
    echo_do v;
    int i;
    const char *val;

    for (i = 0; i < fixup->nelts; ++i) {
        header_entry *hdr = &((header_entry *) (fixup->elts))[i];
        const char *envar = hdr->condition_var;


        /* ignore early headers in late calls */
        if (!early && (envar == condition_early)) {
            continue;
        }
        /* ignore late headers in early calls */
        else if (early && (envar != condition_early)) {
            continue;
        }
        /* Do we have an expression to evaluate? */
        else if (hdr->expr != NULL) {
            const char *err = NULL;
            int eval = ap_expr_exec(r, hdr->expr, &err);
            if (err) {
                ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
APLOGNO(01501)
                              "Failed to evaluate expression (%s) -
ignoring",
                              err);
            }
            else if (!eval) {
                continue;
            }
        }
        /* Have any conditional envar-controlled Header
processing to do? */
        else if (envar && !early) {
            if (*envar != '!') {
                if (apr_table_get(r->subprocess_env, envar) ==
NULL)
                    continue;
            }
            else {
                if (apr_table_get(r->subprocess_env, &envar[1]) !=
NULL)
                    continue;
            }
        }

        switch (hdr->action) {
        case hdr_add:
            apr_table_addn(headers, hdr->header,
process_tags(hdr, r));
            break;
        case hdr_append:
            apr_table_mergen(headers, hdr->header,
process_tags(hdr, r));
            break;
        case hdr_merge:
            val = apr_table_get(headers, hdr->header);
            if (val == NULL) {
                apr_table_addn(headers, hdr->header,
process_tags(hdr, r));
            } else {
                char *new_val = process_tags(hdr, r);
                apr_size_t new_val_len = strlen(new_val);
                int tok_found = 0;


                while (*val) {
                    const char *tok_start;

                    while (*val && apr_isspace(*val))
                        ++val;

                    tok_start = val;

                    while (*val && *val != ',') {
                        if (*val++ == '"')
```

```
                while (*val)
                    if (*val++ == '"')
                        break;
            }

            if (new_val_len == (apr_size_t)(val - tok_start)
                && !strncmp(tok_start, new_val, new_val_len)) {
                tok_found = 1;
                break;
            }

            if (*val)
                ++val;
        }

        if (!tok_found) {
            apr_table_mergen(headers, hdr->header,
new_val);
        }
    }
    break;
  case hdr_set:
    if (!strcasecmp(hdr->header, "Content-Type")) {
        ap_set_content_type(r, process_tags(hdr, r));
    }
    apr_table_setn(headers, hdr->header,
process_tags(hdr, r));
    break;
  case hdr_unset:
    apr_table_unset(headers, hdr->header);
    break;
  case hdr_echo:
    v.r = r;
    v.hdr = hdr;
    apr_table_do((int (*) (void *, const char *, const char *))
            echo_header, (void *) &v, r->headers_in, NULL);
    break;
  case hdr_edit:
  case hdr_edit_r:
    if (!strcasecmp(hdr->header, "Content-Type") && r-
>content_type) {
        const char *repl = process_regexp(hdr, r-
>content_type, r->pool);
        if (repl == NULL)
            return 0;
        ap_set_content_type(r, repl);
    }
    if (apr_table_get(headers, hdr->header)) {
        edit_do ed;

        ed.p = r->pool;
        ed.hdr = hdr;
        ed.t = apr_table_make(r->pool, 5);
        if (!apr_table_do(edit_header, (void *) &ed, headers,
                hdr->header, NULL))
            return 0;
        apr_table_unset(headers, hdr->header);
        apr_table_do(add_them_all, (void *) headers, ed.t,
NULL);
    }
    break;
  }
 }
 return 1;
}

static void ap_headers_insert_output_filter(request_rec *r)
{
 headers_conf *dirconf = ap_get_module_config(r-
>per_dir_config,
                                 &estegano_module);

 if (dirconf->fixup_out->nelts || dirconf->fixup_err->nelts) {
    ap_add_output_filter("FIXUP_HEADERS_OUT", NULL, r,
r->connection);
 }
}

/*
 * Make sure our error-path filter is in place.
 */
static void ap_headers_insert_error_filter(request_rec *r)
{
 headers_conf *dirconf = ap_get_module_config(r-
>per_dir_config,
                                 &estegano_module);

 if (dirconf->fixup_err->nelts) {
    ap_add_output_filter("FIXUP_HEADERS_ERR", NULL, r,
r->connection);
 }
}

static apr_status_t ap_headers_output_filter(ap_filter_t *f,
                                 apr_bucket_brigade *in)
{
 headers_conf *dirconf = ap_get_module_config(f->r-
>per_dir_config,
                                 &estegano_module);

 ap_log_error(APLOG_MARK, APLOG_DEBUG, 0, f->r-
>server, APLOGNO(01502)
            "headers: ap_headers_output_filter()");


 do_headers_fixup(f->r, f->r->err_headers_out, dirconf-
>fixup_err, 0);
 do_headers_fixup(f->r, f->r->headers_out, dirconf-
>fixup_out, 0);


 ap_remove_output_filter(f);


 return ap_pass_brigade(f->next,in);
}

/*
 * Make sure we propagate any "Header always" settings on
the error
 * path through http_protocol.c.
 */
static apr_status_t ap_headers_error_filter(ap_filter_t *f,
                                 apr_bucket_brigade *in)
{
 headers_conf *dirconf;

 dirconf = ap_get_module_config(f->r->per_dir_config,
                    &estegano_module);
 ap_log_error(APLOG_MARK, APLOG_DEBUG, 0, f->r-
>server, APLOGNO(01503)
            "headers: ap_headers_error_filter()");


 do_headers_fixup(f->r, f->r->err_headers_out, dirconf-
>fixup_err, 0);


 ap_remove_output_filter(f);


 return ap_pass_brigade(f->next, in);
}

static apr_status_t ap_headers_fixup(request_rec *r)
{
    estegano_log("ap_headers_fixup");
 headers_conf *dirconf = ap_get_module_config(r-
>per_dir_config,
                                 &estegano_module);
```

```
    /* do the fixup */
    if (dirconf->fixup_in->nelts) {
        do_headers_fixup(r, r->headers_in, dirconf->fixup_in, 0);
    }

    return DECLINED;
}
static apr_status_t ap_headers_early(request_rec *r)
{
    headers_conf *dirconf = ap_get_module_config(r-
>per_dir_config,
                                         &estegano_module);

    /* do the fixup */
    if (dirconf->fixup_in->nelts) {
        if (!do_headers_fixup(r, r->headers_in, dirconf->fixup_in,
1))
            goto err;
    }
    if (dirconf->fixup_err->nelts) {
        if (!do_headers_fixup(r, r->err_headers_out, dirconf-
>fixup_err, 1))
            goto err;
    }
    if (dirconf->fixup_out->nelts) {
        if (!do_headers_fixup(r, r->headers_out, dirconf-
>fixup_out, 1))
            goto err;
    }

    return DECLINED;
err:
    ap_log_rerror(APLOG_MARK, APLOG_CRIT, 0, r,
APLOGNO(01504)
                  "Regular expression replacement failed
(replacement too long?)");
    return HTTP_INTERNAL_SERVER_ERROR;
}

static const command_rec headers_cmds[] =
{
    AP_INIT_RAW_ARGS("Header", header_cmd,
&hdr_out_onsuccess, OR_FILEINFO,
                     "an optional condition, an action, header and
value "
                     "followed by optional env clause"),
    AP_INIT_RAW_ARGS("RequestHeader", header_cmd,
&hdr_in, OR_FILEINFO,
                     "an action, header and value followed by optional
env "
                     "clause"),
    {NULL}
};

static void register_format_tag_handler(const char *tag,
                                        format_tag_fn *tag_handler)
{
    apr_hash_set(format_tag_hash, tag, 1, tag_handler);
}

static int header_pre_config(apr_pool_t *p, apr_pool_t *plog,
apr_pool_t *ptemp)
{
        estegano_log("header_pre_config");
    format_tag_hash = apr_hash_make(p);
    register_format_tag_handler("D",
header_request_duration);
    register_format_tag_handler("t", header_request_time);
    register_format_tag_handler("e", header_request_env_var);
    register_format_tag_handler("s", header_request_ssl_var);
    register_format_tag_handler("l", header_request_loadavg);
    register_format_tag_handler("i", header_request_idle);
    register_format_tag_handler("b", header_request_busy);
    return OK;
}

static int header_post_config(apr_pool_t *pconf, apr_pool_t
*plog,
                              apr_pool_t *ptemp, server_rec *s)
{
    header_ssl_lookup =
APR_RETRIEVE_OPTIONAL_FN(ssl_var_lookup);
    return OK;
}

static void register_hooks(apr_pool_t *p)
{
    ap_register_output_filter("FIXUP_HEADERS_OUT",
ap_headers_output_filter,
                              NULL, AP_FTYPE_CONTENT_SET);
    ap_register_output_filter("FIXUP_HEADERS_ERR",
ap_headers_error_filter,
                              NULL, AP_FTYPE_CONTENT_SET);

ap_hook_pre_config(header_pre_config,NULL,NULL,APR_HO
OK_MIDDLE);

ap_hook_post_config(header_post_config,NULL,NULL,APR_
HOOK_MIDDLE);
    ap_hook_insert_filter(ap_headers_insert_output_filter,
NULL, NULL, APR_HOOK_LAST);
    ap_hook_insert_error_filter(ap_headers_insert_error_filter,
                                NULL, NULL, APR_HOOK_LAST);
    ap_hook_fixups(ap_headers_fixup, NULL, NULL,
APR_HOOK_LAST);
    ap_hook_post_read_request(ap_headers_early, NULL,
NULL, APR_HOOK_FIRST);
}

module AP_MODULE_DECLARE_DATA estegano_module = {
    STANDARD20_MODULE_STUFF,
    create_headers_dir_config,  /* dir config creater */
    merge_headers_config,       /* dir merger --- default is to
override */
    NULL,                       /* server config */
    NULL,                       /* merge server configs */
    headers_cmds,               /* command apr_table_t */
    register_hooks              /* register hooks */
};
```

Anexo 2: Cantidad Paquetes Recibidos

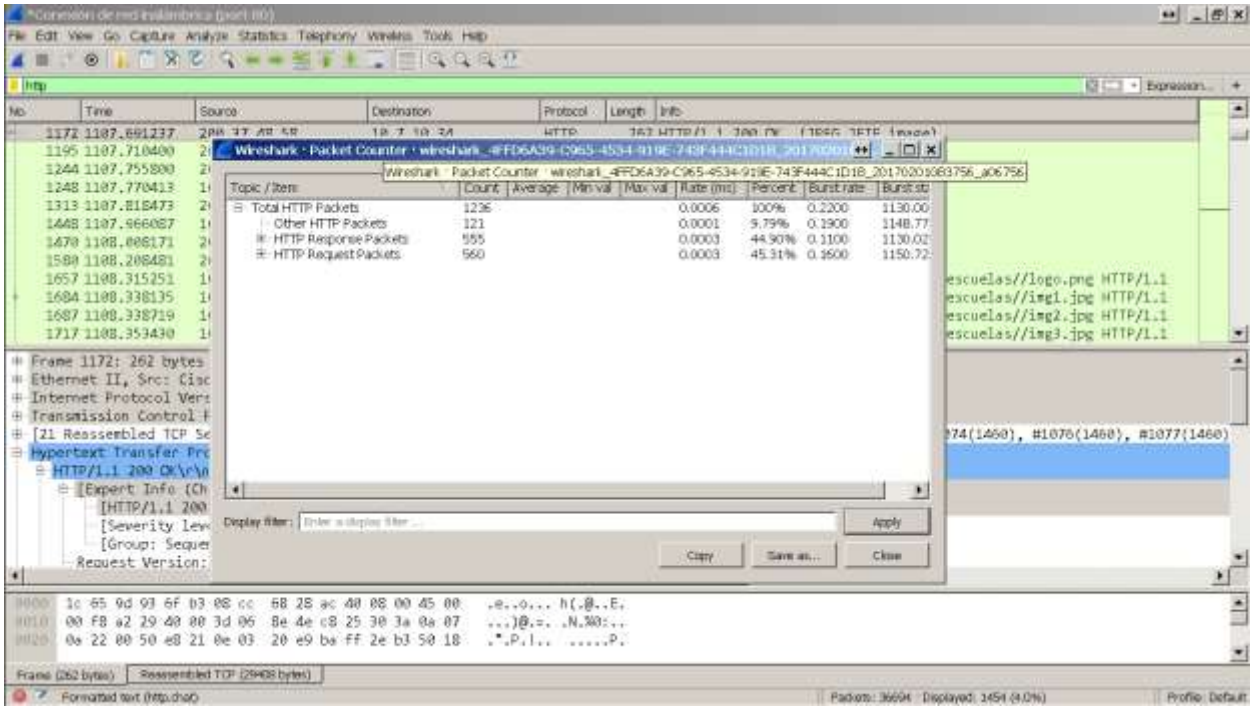

Figura 3: total de paquetes protocolo HTTP

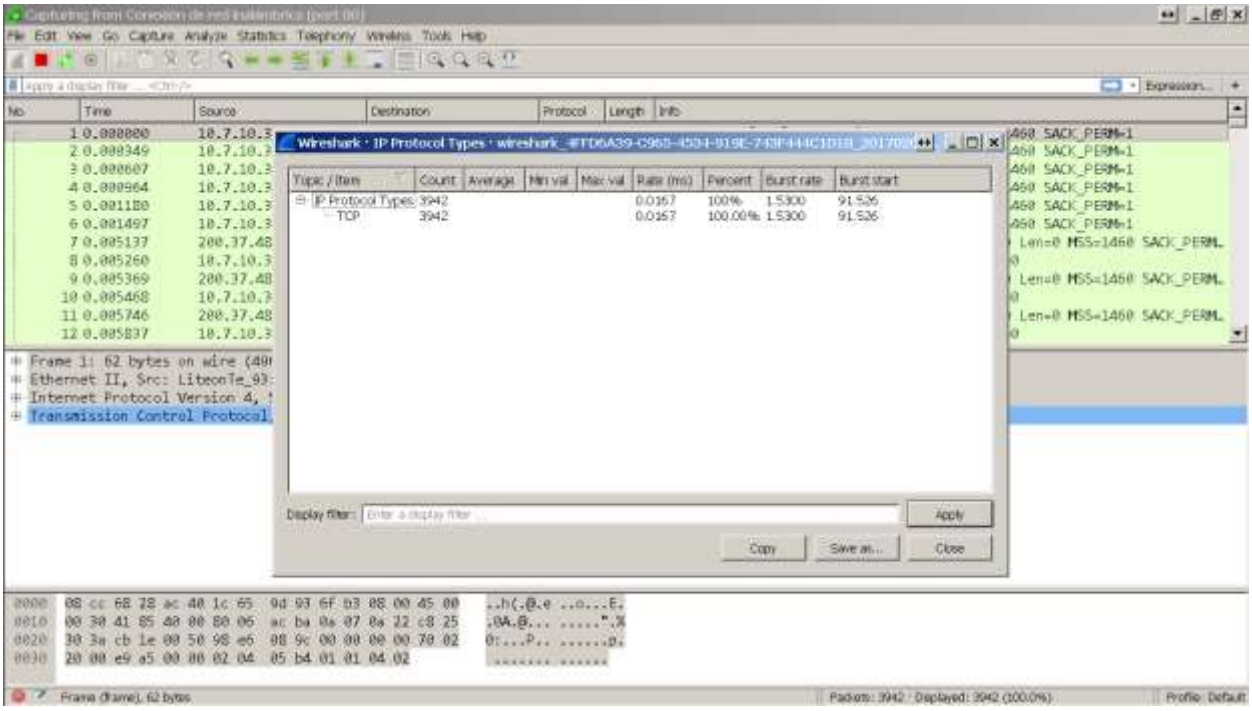

Figura 4: total paquetes protocolos del modelo TCP/IP