2009

Proyecto de Criptografía

Esteganografía

Proyecto de una aplicación para ocultar archivos en un código fuente mediante esteganografía, utilizando lenguaje de programación C.

GRUPO 10

Alejandro Avilés del Moral
Antonio Sánchez Perea
Carlos Morera de la Chica

PLANTEAMIENTO:

Mediante el uso de técnicas de esteganografía, desarrollaremos una aplicación que oculte cualquier archivo en una plantilla de código en C que a su vez será compilable.

El problema lo abordaremos mediante lenguaje C, con el que desarrollaremos la aplicación que lleve a cabo el propósito antes citado. En esta aplicación se introducirá un archivo que será comprimido, cifrado y traducido a información que se ocultará en la plantilla. También será capaz de realizar el proceso a la inversa para que a partir del código fuente consiga el archivo original.

RESUMEN DE PROCESOS:

Debido a la doble función que tiene el programa, se implementan dos bloques simétricos de procesos mediante los que ocultar el archivo en el código fuente o extraer del código fuente la información del archivo que originalmente fue ocultado en el.

A continuación se detallan los diferentes subprocesos para cada tipo de proceso.

Procesos para la ocultación

1. Comprimimos para eliminar la redundancia del archivo y la entropía. Se realizara mediante la utilización en el código de las librerías Zlib.
2. Se procede al cifrado del archivo comprimido mediante el algoritmo RIJNDAEL.
3. Una vez cifrado se realiza la traducción del archivo tal que cadenas de bits, usando un diccionario de palabras, serán correspondidas por nombres de variables del código fuente resultante.
4. La introducción de las variables se hará en plantillas de código fuente, tanto en declaraciones de variables como en cabeceras de funciones. Estas plantillas se irán colocando recursivamente según la longitud del archivo inicial

Procesos para la extracción

1. Se leerá el código fuente mientras se van identificando las variables.
2. Las variables identificadas son traducidas a la cadena de bits a la que corresponden.
3. Una vez extraído la información correspondiente al archivo, se procede a descifrar mediante RIJNDAEL.
4. Descomprimir el archivo descifrado con las librerías Zlib para obtener el archivo original.

USO DEL PROGRAMA:

El programa se ejecutara en línea de comandos. Para utilizarlos deberemos ejecutar el comando *exe -[opciones].*

Opciones

-e / -d
Estas opciones serán usadas para: (e)ncrypt: ocultar archivo de entrada.
(d)ecrypt: extrae del archivo de entrada.

-i [archivo de entrada]
Con esta opción decidiremos el nombre del archivo que queremos ocultar, que podrá tener cualquier extensión, o en cambio el nombre del archivo de código fuente del que queremos extraer el archivo.

-o [archivo de salida]
En esta opción deberemos introducir el nombre del archivo de salida tanto en el caso de ocultar como en el de extraer, salvo que en el primer caso será el nombre del código fuente resultante y en el segundo el del archivo original.

-p [passphrase]
Aquí deberos de introducir una passphrase al ocultar el archivo así como al extraerlo, para que solamente pueda mostrarse la información oculta en el código fuente la persona que tenga posesión de ella.
Podrá tener como máximo 32 caracteres repartidos hasta en 16 palabras, por lo que deberá ser la última opción introducida.

Ejemplos de utilización.

1. Para ocultar el archivo e introducirlo en las variables de un código fuente.

   exe -e -i [archivo original] -o [código fuente de salida] -p [passphrase] [passphrase] […] …

2. Para extraer el archivo del código fuente.

   exe -d -i [código fuente de entrada] -o [archivo original] -p [passphrase] [passphrase] […] …

PROCESOS DETALLADOS:

El programa, como ya hemos dicho, se compone de 2 procesos bien diferenciados e inversos. El primero es la ocultación del archivo en un código fuente mediante la función *estega* y el segundo el proceso inverso mediante *desestega*, extraer la información previamente ocultada en un código fuente.

Ocultación:

Hemos usado una sola función llamada ***estega*** que a su vez llama a las demás en el orden necesario. Para comprimir se usa la función *zdef* sobre el archivo origen, generando uno temporal (sal_zlib) con un nivel de compresión por defecto (Z_DEFAULT_COMPRESSION).

Para cifrar el archivo ya comprimido y con menos redundancia, usamos el algoritmo de cifrado RIJNDAEL, más conocido como AES (Advanced Encryption Standard). La función ***encrypt*** se encarga de esto, pasando como parámetros el archivo de salida comprimido (sal_zlib), el archivo de salida del cifrado (sal_rdl) y la passphrase.

En este último proceso, el de generación del código fuente, es en el que nos vamos a centrar. Vamos a usar una estructura dinámica. Un vector de caracteres con unos contadores.

1º Con la función *escribir_reserva_mem* calculamos el tamaño total del archivo y reservamos memoria para usarla a lo largo del programa.

2º Ahora llamamos a *escribir_carga_archivo*, que carga desde el archivo byte a byte a la estructura creada anteriormente.

3º En este tercer punto lo que hacemos es eliminar algunos archivos temporales que se han podido crear

4º En este punto vamos a empezar a generar el código fuente de salida. En primer lugar generamos el main, porque recordemos que el código fuente que genera es compilable.

escribir_genera_main genera, a partir de la estructura, un archivo temporal con el main de la futura función en un archivo temporal llamado main_tmp. Esto devuelve 1 o 0, depediendo si aún queda suficiente archivo para escribir o no.

Ahora llamamos múltiples veces, hasta que se haya escrito todo el archivo origen a la función *escribir_funcion*, que devuelve lo mismo que la funcion que genera el main.

Después de esto, sólo nos queda juntar todos los archivos temporales en uno final. Para ello existe *escribir_empaquetar*, que lo único que hace es generar un único archivo .c con todos los archivos temporales que habíamos creado anteriormente.

Por último liberamos la memoria de la estructura con *escribir_free* y se acaba la ejecución del programa.

Extracción:

Usamos una sola función, ***desestega***, que se encarga de llamar al resto que veremos a continuación. De forma simétrica a al proceso de ocultación, nos centraremos en el subproceso de análisis de variables del código fuente, pues es dónde se halla la información del archivo original. Para ello utilizamos la función *leer*, que se encarga de ir leyendo línea por línea el código.

En ***leer***, el análisis de variables no comenzará hasta encontrar el "int main() {", a partir de dónde cada vez que encuentre una definición de variable "int VAR" ésta será traducida a la cadena de bits que corresponde y guardada en un archivo temporal mediante la función *VarToBit*. Como también se pueden encontrar variables en las cabeceras de funciones, para captarlas usaremos la función *VarCabecera*.

VarCabecera se limita a extraer las variables de la cabecera en forma de "void VAR1(int VAR2,…){" y llamar a la función *VarToBit* para cada una de ellas.

VarToBit tiene la función de separar el nombre de una función en las distintas palabras que la componen, encontrar su posición en el diccionario mediante *buscaPos*, y junto con el número que acompaña a la variable, escribirlos todos en el archivo temporal.

Una vez extraída toda la información de las variables del código fuente en el archivo temporal, ahora se deberá descifrar usando el algoritmo de cifrado RIJNDAEL. La función ***decrypt*** se encarga de esto, pasando como parámetros el archivo temporal, el archivo donde se guardará descifrado y la passphrase. Si ésta última fuera incorrecta se generaría un archivo corrupto que no se podría descomprimir en el siguiente paso.

Por último, para recuperar el archivo original, teniendo el archivo descifrado, lo único que queda es descomprimirlo usando la librería zlib, la misma que hemos usado para comprimir al generarlo. Usamos la función *zinf*, con los descriptores de fichero del archivo descifrado y del archivo de destino.

Main:

En la llamada principal del programa se parsean los argumentos para poder elegir el modo, archivos de entrada y salida y la passphrase a usar.

Si se llama al programa con **-e** se realizarán los subprocesos de la función *estega*.
Si por el contrario se llama con **-d** se realizarán los de *desestega*.