TEC | Tecnológico de Costa Rica

Seguridad del Software - IC8071

Proyecto 1 - Identificación de Debilidades

Profesor:

Herson Esquivel Vargas

Integrantes:

Leonardo Céspedes Tenorio - 2022080602

Kevin Chang Chang - 2022039050

Fecha de Entrega: 4 de Septiembre

II Semestre 2024

Índice

Software a Analizar

El código que decidimos utilizar y analizar es la versión 4.2.2 de Wireshark. Una herramienta de análisis utilizada para la captura y análisis detallado del tráfico de red. Este código proviene de los repositorios de Ubuntu y está escrito en C.

El sistema operativo utilizado para analizar el código es Ubuntu 24.04 LTS de 64 bits. Para obtener la aplicación se utilizó el comando: *sudo apt-get source tshark.*

Al instalar la aplicación y obtener el código fuente se genera un folder llamado 'wireshark-4.2.2', dentro de este folder utilizamos el comando *find . -name "*.c" -type f -exec wc -l {} + | awk -v x=1000 '$1 > x {print $2}* para encontrar todos los archivos '.c'. Así encontramos el archivo que decidimos analizar (En la dirección *./tools/lemon/lemon.c* dentro del folder principal), llamado 'lemon.c'. Este archivo es un parser generator utilizado por Wireshark.

Herramientas Usadas y las Opciones Utilizadas

Flawfinder

flawfinder --minlevel=0 --neverignore --context --html lemon.c > results.html
flawfinder --neverignore --context --html --inputs lemon.c > resultsInputs.html

Explicación de las opciones

- **--minlevel=0:** Esta opción configura el nivel mínimo de severidad de las vulnerabilidades que se mostrarán en el informe. El nivel de severidad va de 0 (más crítico) a 5 (menos crítico). Al usar --minlevel=0, se solicita a Flawfinder que incluya todas las vulnerabilidades, independientemente de su nivel de severidad.
- **--neverignore:** Esta opción indica que Flawfinder no debe ignorar ningún tipo de vulnerabilidad, incluso si están listadas en una lista de ignorados o configuraciones predeterminadas. Es útil para asegurarte de que se detecten todas las posibles vulnerabilidades, sin excluir ninguna.
- **--context:** Al incluir esta opción, Flawfinder incluirá el contexto del código en el que se encuentra cada vulnerabilidad, proporcionando información adicional sobre el entorno y el contexto de la línea donde se encuentra la vulnerabilidad.

- **--html:** Esta opción especifica que el informe debe generarse en formato HTML, lo cual es más legible y permite una visualización más amigable de los resultados en un navegador web.
- **--inputs:** Esta opción le dice a Flawfinder que también debe buscar vulnerabilidades en el código relacionado con entradas, como entradas del usuario que podrían ser peligrosas si no se validan correctamente.
- **lemon.c:** Este es el archivo de código fuente que Flawfinder analizará en busca de vulnerabilidades.
- **> results.html/resultsInputs.html:** Redirige la salida del comando a un archivo llamado results.html, donde se guardará el informe en formato HTML.

¿Por qué se usaron dos comandos diferentes?

El primer comando proporciona una vista completa de todas las vulnerabilidades, mientras que el segundo se enfoca específicamente en problemas relacionados con entradas, lo cual puede ser crítico para la seguridad de las aplicaciones que manejan datos de entrada del usuario.

Cppcheck

cppcheck --enable=all --template="{line}:{severity}:{id}:{cwe}:{message}:{code}"
--library=std.cfg --inconclusive --force --check-level=exhaustive
--suppress=missingIncludeSystem lemon.c > results.txt 2>&1

Explicación de las opciones

- **--enable=all:** Esta opción habilita todos los tipos de comprobaciones disponibles en Cppcheck, incluyendo todos los niveles de severidad y tipos de problemas que puede detectar. Esto asegura que no se pase por alto ninguna posible advertencia o error.
- **--template="{line}:{severity}:{id}:{cwe}:{message}:{code}":** Configura el formato de salida del informe. El template especifica que la salida debe incluir el número de línea, la severidad del problema, el identificador del problema, el CWE (Common Weakness Enumeration) asociado, un mensaje descriptivo y el código fuente relevante. Esto proporciona un informe detallado y bien estructurado.
- **--inconclusive:** Permite que Cppcheck informe advertencias inconclusas, es decir, problemas para los cuales el análisis no puede proporcionar una certeza absoluta. Esto puede ser útil para una revisión más exhaustiva del código.

- **--force:** Obliga a Cppcheck a realizar el análisis, incluso si detecta problemas menores que normalmente se ignoran. Esto asegura que se realice un análisis completo sin omitir nada.
- **--check-level=exhaustive:** Configura el nivel de exhaustividad del análisis a exhaustive, lo que significa que Cppcheck realizará una revisión detallada y minuciosa del código para encontrar problemas potenciales.
- **--suppress=missingIncludeSystem:** Suprime advertencias relacionadas con archivos de inclusión del sistema que no se encuentran. Esto puede evitar que aparezcan errores relacionados con archivos de encabezado del sistema que no están disponibles en el entorno actual.
- **lemon.c:** El archivo de código fuente que será analizado por Cppcheck.
- **> results.txt 2>&1:** Redirige tanto la salida estándar como los errores estándar a un archivo llamado results.txt. Esto asegura que toda la información generada por el comando, incluyendo advertencias y errores, se guarde en un solo archivo para su revisión.

Splint

splint -DPACKAGE_STRING= -mustfreefresh -exportlocal -nullassign -nullret -nullpass -usedef -compdef -boolops -unrecog -mustfreeonly -bounds -type -branchstate -mem-trans -casebreak -nullstate -compdestroy -usereleased -compmempass -globstate -predboolint -bufferoverflowhigh -formatconst -mayaliasunique -shiftimplementation -retvalint -predboolothers lemon.c > results.txt 2>&1

Explicación de las opciones

- **DPACKAGE_STRING=:** Define un macro del preprocesador vacío para evitar errores de análisis en caso de que no esté definido en el código.
- **mustfreefresh:** Asegura que toda memoria asignada dinámicamente sea liberada, ayudando a prevenir fugas de memoria.
- **exportlocal:** Genera advertencias si una función exportada accede a variables locales, lo que puede llevar a comportamientos indeseados.
- **nullassign:** Advierte sobre la asignación de punteros a NULL, lo cual es crucial para evitar errores al desreferenciar punteros nulos.
- **nullret:** Ayuda a evitar advertencias falsas cuando una función retorna un puntero nulo de manera legítima.

- **nullpass:** Verifica que las funciones reciban los punteros correctos, incluyendo el manejo adecuado de punteros nulos.
- **usedef:** Identifica el uso de variables antes de que sean definidas, evitando problemas con datos no inicializados.
- **compdef:** Asegura que las variables compuestas (como estructuras y uniones) estén correctamente definidas antes de su uso.
- **boolops:** Advierte sobre operaciones booleanas que no cumplen explícitamente con las reglas esperadas de operaciones booleanas.
- **unrecog:** Señala el uso de constructores o sintaxis no reconocidos por el analizador.
- mustfreeonly: Verifica que solo se libere memoria asignada dinámicamente, evitando la liberación de memoria que no ha sido asignada dinámicamente.
- **bounds:** Realiza comprobaciones en los límites de arrays para prevenir errores de desbordamiento de búfer.
- **type:** Realiza verificaciones estrictas de tipos para prevenir errores de incompatibilidad de tipos.
- **branchstate:** Asegura que el estado de las variables sea coherente en todas las ramas de estructuras condicionales.
- **mem-trans:** Revisa la transferencia de memoria para asegurar que no haya problemas con la asignación y liberación de memoria.
- **casebreak:** Advierte si falta un break en una declaración case dentro de una estructura switch, lo cual puede causar errores en el flujo de control.
- **nullstate:** Verifica que el código maneje correctamente los punteros nulos para evitar problemas relacionados con estados nulos.
- **compdestroy:** Asegura que todos los componentes de las estructuras sean liberados correctamente para evitar fugas de memoria.
- **usereleased:** Genera advertencias si se usa memoria después de haber sido liberada, lo cual puede causar comportamientos indeseados.
- **compmempass:** Analiza la transferencia de estructuras compuestas entre funciones para asegurar una gestión adecuada de la memoria.
- **globstate:** Verifica el uso consistente de variables globales a lo largo del código.
- **predboolint:** Permite cierta flexibilidad en la interpretación de expresiones booleanas y operaciones enteras.

- **bufferoverflowhigh:** Incrementa la sensibilidad hacia posibles desbordamientos de búfer, identificando riesgos potenciales con mayor precisión.
- **formatconst:** Exige que las cadenas de formato en funciones como printf sean constantes en tiempo de compilación para prevenir errores de formato.
- **immediatetrans:** Verifica la transferencia inmediata de memoria para garantizar una correcta asignación y liberación sin fugas intermedias.
- **mayaliasunique:** Permite algunas prácticas de programación mientras asegura que los alias sean únicos cuando sea necesario para mantener la seguridad.
- **shiftimplementation:** Advierte sobre operaciones de desplazamiento de bits que podrían comportarse de manera diferente en distintos compiladores.
- **retvalint:** Evita advertencias relacionadas con valores de retorno enteros que no son verificados o utilizados adecuadamente.
- **predboolothers:** Facilita la interpretación de valores booleanos en el código, considerando casos donde false es 0 y true es 1.
- **paramuse:** Genera advertencias para parámetros que están declarados en una función pero no se utilizan dentro de ella.

Frama-C

frama-c -eva-verbose 2 -eva-log a:all.txt,r:results.txt -time all.txt -time results.txt -eva-precision 0 -safe-arrays -warn-unsigned-overflow -eva-flamegraph grafico.log -eva-auto-loop-unroll 20 -eva-plevel 30 -eva lemon.c

Explicación de las opciones

- **eva-verbose 2:** Incrementa el nivel de verbosidad para mostrar una mayor cantidad de mensajes durante el análisis, proporcionando detalles adicionales que pueden ser relevantes para comprender el proceso.
- **eva-log a:all.txt,r:results.txt:** Especifica que todo el registro del análisis se debe guardar en all.txt, mientras que los resultados específicos se guardarán en results.txt.
- **time all.txt -time results.txt:** Incluye información de tiempo de ejecución en los archivos all.txt y results.txt, ayudando a medir el tiempo total y específico de cada parte del análisis.

- **eva-precision 0:** Configura la precisión del análisis a nivel 0, lo que reduce la precisión del análisis pero acelera el proceso. Ideal para una revisión rápida cuando no se requiere una precisión completa.
- **safe-arrays:** Verifica que los accesos a arrays se mantengan dentro de los límites válidos (n-1), ayudando a prevenir errores de desbordamiento de búfer.
- **warn-unsigned-overflow:** Genera advertencias si se detecta un desbordamiento en operaciones con números sin signo, lo que puede ayudar a identificar posibles errores en el manejo de datos numéricos.
- **eva-flamegraph grafico.log:** Genera un flame graph que muestra el tiempo de ejecución de las funciones, permitiendo visualizar cuáles funciones tardaron más en analizarse.
- **eva-auto-loop-unroll 20:** Realiza un desenrollado automático de bucles con menos de 20 iteraciones, lo que permite un análisis más completo de los bucles pequeños.
- **eva-plevel 30:** Limita el número de elementos en un array que se consideran precisos durante el análisis a 30, equilibrando precisión y rendimiento.
- **eva:** Activa el modo de análisis de evaluación en Frama-C para el archivo lemon.c, aplicando las configuraciones y opciones especificadas.

Análisis de CWE para WireShark

Lemon.c (5893 LoC)

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
34	CWE-367		Se considera un falso positivo porque en dicha línea de código únicamente se declara la función access, no realiza ninguna operación ni verificación por sí misma.	✓			
3556	CWE-367		Se considera un falso positivo, pues en el código no se está realizando ninguna operación adicional en el archivo basado en la verificación de access(), entonces realmente no existe esta ventana de tiempo donde el archivo puede cambiar para favorecer a un ataque.	✓			
3650	CWE-367		Se considera un verdadero	✓			

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			positivo. Primeramente se realiza la llamada a access(), donde se verifica si el archivo es accesible para lectura. Si esto no falla, el programa puede abrir el archivo en modo binario. Entre la llamada de access y fopen hay una ventana de tiempo donde el archivo puede ser modificado o cambiado. Esto quiere decir que se verificaría un archivo y se abriría otro, lo cual podría traer consecuencias inesperadas.				
3672	CWE-367		Se considera un falso positivo, pues a pesar de utilizar la función access, no existe un uso crítico posterior a la verificación ni ningún tipo de operación importante que se base en el resultado de dicha función.	✓			
3674	CWE-367		Se considera un falso positivo porque no existe ninguna acción crítica sobre el archivo luego de la llamada de access(). No hay ninguna acción crítica como lectura o escritura sobre el archivo que se está verificando el acceso, por lo cual no hay una brecha real para realizar un ataque.	✓			
2971	CWE-362		Se considera un verdadero positivo porque se podría llegar a presentar el *race condition* al intentar leer un archivo, lo cual implica una posible vulnerabilidad a manipulaciones externas o cambios en el sistema de archivos.	✓			
3222	CWE-362		Se considera un verdadero positivo porque se podría llegar a presentar el *race condition* al intentar leer un archivo, lo cual implica una posible vulnerabilidad a manipulaciones externas o cambios en el sistema de archivos.	✓			

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
3656	CWE-362		Se considera un verdadero positivo porque se podría llegar a presentar el *race condition* al intentar leer un archivo, lo cual implica una posible vulnerabilidad a manipulaciones externas o cambios en el sistema de archivos.	✓			
3685	CWE-362		Se considera un verdadero positivo porque se podría llegar a presentar el *race condition* al intentar leer un archivo, lo cual implica una posible vulnerabilidad a manipulaciones externas o cambios en el sistema de archivos.	✓			
1502	CWE-134		Se considera un falso positivo porque, aunque la función de ErrorMsg recibe como parámetro el format, este se encuentra hardcoded en las partes donde se realiza la llamada de dicha función	✓			✓
137	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘illegal format \n’). No se están utilizando datos del usuario que puedan ser manipulados de alguna manera, es solo un mensaje de error.	✓			✓
499	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘Unable to allocate memory for a new parser action.’). No se están utilizando datos del usuario que puedan ser manipulados de alguna manera, es solo un mensaje de error.	✓			✓
625	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘Unable to allocate memory for a new acttab.’). No se están utilizando datos del usuario que puedan ser manipulados	✓			✓

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			de alguna manera, es solo un mensaje de error.				
645	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘malloc failed\n’). No se están utilizando datos del usuario que puedan ser manipulados de alguna manera, es solo un mensaje de error.	✓			✓
1182	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘internal error on source line…’). No se están utilizando datos del usuario que puedan ser manipulados de alguna manera, es solo un mensaje de error.	✓			✓
1504	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘\n’). No se están utilizando datos del usuario que puedan ser manipulados de alguna manera, es solo un print de un salto de línea.	✓			✓
1550	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘out of memory\n’). No se están utilizando datos del usuario que puedan ser manipulados de alguna manera, es solo un mensaje de error.	✓			✓
1620	CWE-134		Se considera un falso positivo, pues, aunque zLabel llegará a ser alterada de alguna manera, esta solo se está usando como argumento de formato y no como parte del contenido, por lo cual se tiene un control del contenido.	✓			✓
3237	CWE-134		Se considera un falso positivo porque se está utilizando una cadena de formato literal fija (‘::=’). No se están utilizando datos del usuario que puedan ser manipulados de alguna	✓			✓

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			manera, es solo un print de una cadena de texto fija.				
3540	CWE-20		Se considera un verdadero positivo, pues no se realizan validaciones posteriormente para validar que ‘pathlist’ únicamente contenga direcciones seguras.	✓			✓
4964	CWE-20		Se considera un falso positivo, pues posteriormente se realiza la validación por EOF y no hay evidencia de que un input pueda causar un problema.	✓			
2988	CWE-20		Se considera un falso positivo, pues el filesize es manejado de manera adecuada y se le asigna suficiente memoria. Además, se realizan las validaciones correspondientes a la lectura del archivo.	✓			
89	CWE-120	✓	Se considera un verdadero positivo, ya que no hay validaciones en ‘lemon_addtext’ para asegurar que el ‘zBuf’ tenga suficiente espacio para acomodar el texto agregado en ‘zIn’.	✓			
2661	CWE-120		Se considera falso positivo, pues el valor de ‘n’ se calcula de manera precisa antes de la llamada al ‘realloc’ y no hay riesgo de que los datos excedan el tamaño del buffer	✓			
2676	CWE-120		Se considera un falso positivo ya que en el ‘memcpy’ se asegura que el ‘nNew’ no sea más grande que ‘zBuf’ y que ‘zBuf siempre tenga suficiente espacio para manejar lo copiado de ‘nNew’’	✓			
3914	CWE-120		Se considera un falso positivo, ‘sprintf’ genera un string basado en los valores de ‘1-rp->nrhs’ y ‘rp->lhs->dtnum’ donde no existe prueba de que pueda ser mayor al tamaño de ‘zLhs’	✓			
3917	CWE-120		Se considera un falso positivo,	✓			

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			‘sprintf’ genera un string basado en el valor de ‘rp->lhs->dtnum’ donde no existe prueba de que pueda ser mayor al tamaño de ‘zLhs’				
98	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
1528	CWE-119		Se considera falso positivo, pues el tamaño de ‘azDefine’ crece dinámicamente para adaptarse al tamaño de ‘nDefinde’ basado en un calcula alambrado a ‘nDefine’ quitando la posibilidad de un buffer overflow. También se asegura que ‘realloc’ no retorna NULL con validaciones.	✓			
1901	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
2244	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
2632	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
3436	CWE-119		Se considera falso positivo, pues solo se está declarando el	✓			

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer				
3607	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
3631	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
3642	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
3799	CWE-119		Es falso positivo, pues cuando se declara el buffer ‘empty’ ahí mismo se llena el buffer con una cantidad definida de datos y no se vuelve a utilizar el buffer	✓			
3804	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
3854	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que	✓			

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			se exceda del tamaño del buffer				
3855	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
3856	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
3862	CWE-119		Es falso positivo, pues cuando se declara el buffer 'newlinestr' ahí mismo se llena el buffer con una cantidad definida de datos y no se vuelve a utilizar el buffer	✓			
4950	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
4951	CWE-119		Se considera falso positivo, pues solo se está declarando el buffer de memoria y no hay prueba de que se escriba afuera de los límites del buffer o se trate de insertar un valor que se exceda del tamaño del buffer	✓			
60	CWE-126		Se considera un falso positivo, pues en esta línea de código únicamente se define la macro 'lemonStrlen(X)', la cual por sí sola no implica un *buffer over-read*.	✓			
3489	CWE-126		Se considera un verdadero positivo, pues existe una pequeña posibilidad de que el	✓			

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			string no tenga caracter de terminación, lo cual no está validado.				
1261	CWE-571		Se considera un falso positivo, pues *spx* y *spy* son variables cuyos valores pueden cambiar y la condición no debe ser necesariamente verdadera.		✓		
1263	CWE-571		Se considera un falso positivo, pues *spx* y *spy* son variables cuyos valores pueden cambiar y la condición no debe ser necesariamente verdadera.		✓		
1278	CWE-571		Se considera un falso positivo, pues *spx* y *spy* son variables cuyos valores pueden cambiar y la condición no debe ser necesariamente verdadera.		✓		
3750	CWE-570		Se considera un falso positivo, pues la condición no debe ser necesariamente falsa, ya que se está evaluando si lemp->vardest == 0 y basado en otras secciones del código, este valor puede ser diferente.		✓		
497	CWE-401	✓	Se considera un falso negativo, ya que no se están haciendo las validaciones necesarias para verificar si el calloc devuelve NULL lo cual podría llevar a que la memoria no se libere y haya un memory leak.				
1534	CWE-401	✓	Se considera un falso negativo, ya que no se están haciendo las validaciones necesarias para verificar si el malloc devuelve NULL lo cual podría llevar a que la memoria no se libere y haya un memory leak.				
3819	CWE-401		Se considera un verdadero positivo, pues no se están haciendo las validaciones necesarias para verificar si el realloc devuelve NULL, lo cual podría llevar a que la memoria no se libere y haya un memory leak.		✓		
2953	CWE-628		Se considera un falso positivo,		✓		

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			pues en esta línea de código tenemos una definición de función y no una llamada en sí, por lo cual no existe esta debilidad donde se aplican de manera inadecuada los parámetros en la llamada de una función.				
3524	CWE-563		Se considera un falso positivo, pues la variable '*pathbuf'* es utilizada en las líneas posteriores, entonces no aplica el *dead store*.		✓		
2144	CWE-561	✓	Se considera un verdadero positivo, pues la función *OptErr(int n)* es declarada en esta línea, pero no se llama en ningún lugar, por lo que no puede llegar a ser ejecutada.		✓		
5547	CWE-561	✓	Se considera un verdadero positivo, pues la función *Symbol_Nth(int n)* es definida, pero nunca es invocada en el programa		✓		
944	CWE-758		Se considera un falso positivo, pues en esta línea únicamente hay un comentario.			✓	
138	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
500	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
626	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			ningún comportamiento indefinido.				
646	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
696	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
924	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
1184	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
1516	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
1531	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
1537	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
1551	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
1677	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
1698	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
2119	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
2879	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
2944	CWE-758		Se considera un falso positivo porque la función exit() está			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.				
3125	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
3196	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
4112	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
4129	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
4178	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.			✓	
4517	CWE-758		Se considera un falso positivo porque la función exit() está definida como estándar en C para terminar de manera			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			inmediata la ejecución del programa, no depende de ningún comportamiento indefinido.				
5256	CWE-758		Se considera un verdadero positivo pues en caso de que *malloc* falle, 'z' conservará un valor de 0 (NULL) y MemoryCheck no hace las validaciones necesarias para dicho escenario, lo cual podría llevar a un crash o comportamiento inesperado.			✓	
5385	CWE-758		Se considera un verdadero positivo porque no se está validando que sp no sea NULL luego de hacer el *calloc*, lo cual podría fallar si el sistema no tiene suficiente memoria disponible. Además de que el MemoryCheck puede llevar a algún comportamiento inesperado, se les asignan valores a los campos de sp, el cual podría ser nulo y causar un crash.			✓	
5622	CWE-758		Se considera un verdadero positivo, pues no se está validando el escenario en el que falle el *calloc* y la variable *newstate* siga con valor NULL luego de dicha línea de código, lo cual podría llegar a causar comportamientos inesperados al utilizarse en *MemoryCheck(newstate)*.			✓	
994	CWE-710		Se considera un verdadero positivo pues los estándares recomiendan verificar el return de las funciones que tienen que ver con *Memory Allocation* como lo es *'State_new()'* en este caso.			✓	
239	CWE-691		Se considera un falso positivo, pues todas las funciones que retornan un valor 'boolean' retorna explícitamente 'LEMON_TRUE' o 'LEMON_FALSE' y no hay pruebas de que las funciones que hacen condicionales con			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			estas variables utilicen valores diferentes a 1 y 0, además el enum 'Boolean' siempre es declarado				
995	CWE-476		Se considera un verdadero positivo, pues no se está validando el resultado de *State_new()*, la cual retorna un puntero y podría ser nulo y en esta línea se están asignando valores a los campos de dicha variable.			✓	
5622	CWE-476		Se considera un verdadero positivo, pues no se está validando el estado de la variable *newstate* antes de utilizarla en *'MemoryCheck(newstate)'* y esto podría traer resultados inesperados si tiene un valor nulo.			✓	
5385	CWE-476		Se considera un verdadero positivo pues si el *calloc* llegara a fallar, se estaría utilizando la variable sp con valor NULL en la función '*MemoryCheck(sp)'* y esto podría involucrar comportamientos inesperados.			✓	
5256	CWE-476		Se considera un verdadero positivo pues los estándares recomiendan realizar las verificaciones necesarias cuando se trabaja con *Memory Allocation*, lo cual no se realiza en este caso y puede que z entre como null a *'MemoryCheck(z)'*.			✓	
2437	CWE-476		Se considera un verdadero positivo pues no se está validando si la variable msp es nula y se le están asignando valores a sus campos, lo cual causaría un error en caso de que 'msp' sea NULL, pues el campo *type* no existiría.			✓	
2440	CWE-476		Se considera un verdadero positivo pues no se está validando si la variable msp es nula y se le están asignando			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			valores a sus campos, lo cual causaría un error en caso de que 'msp' sea NULL, pues el campo *subsym[0]* no existiría.				
2447	CWE-476		Se considera un verdadero positivo pues no se está validando si la variable msp es nula y se le están asignando valores a sus campos, lo cual causaría un error en caso de que msp sea NULL, pues el campo *subsym[msp->nsubsym-1]* no existiría.			✓	
2772	CWE-476		Se considera un verdadero positivo pues no se está validando si la variable msp es nula y se le están asignando valores a sus campos, lo cual causaría un error en caso de que msp sea NULL, pues el campo *subsym[msp->nsubsum-1]* no existiría.			✓	✓
3755	CWE-476		Se considera un falso positivo, pues en la línea se está validando si cp es NULL en distintos casos y también se está inicializando de manera adecuada.			✓	
4152	CWE-476		Se considera un falso positivo, pues en la línea se está validando si cp es NULL en distintos casos y también se está inicializando de manera adecuada.			✓	✓
5297	CWE-476		Se considera un falso positivo, pues se están haciendo las validaciones necesarias para comprobar que x1a no sea NULL, lo cual permite evitar dicho error.			✓	
5386	CWE-476		Se considera un verdadero positivo porque se están asignando valores a los campos de sp sin haber validado si es NULL o no. En caso de serlo, el campo *name* no existiría.			✓	✓

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
5465	CWE-476		Se considera un falso positivo, pues se están haciendo las validaciones necesarias para comprobar que x2a no sea NULL, lo cual permite evitar dicho error.			✓	
5664	CWE-476		Se considera un falso positivo, pues se están haciendo las validaciones necesarias para comprobar que x3a no sea NULL, lo cual permite evitar dicho error.			✓	
5804	CWE-476		Se considera un falso positivo, pues se están haciendo las validaciones necesarias para comprobar que x4a no sea NULL, lo cual permite evitar dicho error.			✓	
2078	CWE-704		Se considera un verdadero positivo, pues en el caso de que op[j].arg no apunte a un dato de tipo ‘double’, esto resultaría en una conversión de datos errónea. Este comportamiento podría resultar en un comportamiento indefinido o corrupción de la memoria.		✓		
1642	CWE-704		Se considera un falso positivo, pues en esta línea de código no se realiza ningún tipo de conversión o *casting* entre tipos de datos.			✓	
1997	CWE-704		Se considera un verdadero positivo, pues en el caso de que op[j].arg no apunte a un dato de tipo int, esto resultaría en una conversión de datos errónea. Este comportamiento podría resultar en un comportamiento indefinido o corrupción de la memoria.			✓	
1999	CWE-704		Se considera un verdadero positivo, pues en el caso de que op[j].arg no apunte a un dato de tipo char *, esto resultaría en una conversión de datos errónea. Este comportamiento podría resultar en un comportamiento			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
			indefinido o corrupción de la memoria.				
2081	CWE-704		Se considera un verdadero positivo, pues en el caso de que op[j].arg no apunte a un dato de tipo 'double', esto resultaría en una conversión de datos errónea. Este comportamiento podría resultar en un comportamiento indefinido o corrupción de la memoria.			✓	
2087	CWE-704		Se considera un verdadero positivo, pues en el caso de que op[j].arg no apunte a un dato de tipo int, esto resultaría en una conversión de datos errónea. Este comportamiento podría resultar en un comportamiento indefinido o corrupción de la memoria.			✓	
2093	CWE-704		Se considera un verdadero positivo, pues en el caso de que op[j].arg no apunte a un dato de tipo char *, esto resultaría en una conversión de datos errónea. Este comportamiento podría resultar en un comportamiento indefinido o corrupción de la memoria.			✓	
1703	CWE-252		Se consideran falso positivo, ya que 'Symbol_new' siempre retorna un valor, este siendo 'sp' que siempre está garantizado de tener un valor no nulo.			✓	
1691	CWE-252		Se consideran falso positivo, ya que 'Symbol_new' siempre retorna un valor, este siendo 'sp' que siempre está garantizado de tener un valor no nulo.			✓	
3879	CWE-252		Se consideran falso positivo, ya que 'append_str' siempre retorna un valor, este siendo 'z' que siempre está garantizado de tener un valor no nulo y no 0.			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
3880	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
3920	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
3925	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
3936	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
3950	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
3959	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
3969	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
3973	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
4016	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
4024	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
4025	CWE-252		Se consideran falso positivo, ya que ‘append_str’ siempre retorna un valor, este siendo ‘z’ que siempre está garantizado de tener un valor no nulo y no 0.			✓	
1669	CWE-1164		Es un verdadero positivo, ya que ‘int main()’ recibe como parámetro ‘argc’ y a través del código no se utiliza para cambiar el flujo o contenido de los datos. Solo se usa en ‘(void)argc’ que funciona solo para evitar una advertencia en el compilador y no para la funcionalidad del código			✓	
1796	CWE-561		Es un falso positivo, pues se encuentra en ‘int main()’ así que su función sí es llamada y el código no se encuentra detrás de una condición específica ni nada por el estilo por lo que no se considera código muerto o inalcanzable.			✓	
1528	CWE-131		Se considera falso positivo, pues no hay problemas con calcular el valor de ‘nDefine’ ya que siempre coincide con el tamaño de ‘azDefine’ y realloc no retorna NULL.				✓
1548	CWE-131		Se considera falso positivo, pues no hay problemas con calcular el valor de ‘lemonStrlen(z)’ ya que siempre coincide con el tamaño de ‘outputDir’ y malloc no retorna NULL.				✓

# Línea	Debilidad	Manual	Explicación	Flawfinder	Cppcheck	Splint	Frama-C
1534	CWE-131		Se considera falso positivo, pues no hay problemas con calcular el valor de ‘lemonStrlen(z)’ ya que siempre coincide con el tamaño de ‘paz’ y malloc no retorna NULL.				✓
5287	CWE-131		Se considera falso positivo, pues no hay problemas con calcular el valor de ‘s_x1’ ya que siempre coincide con el tamaño de ‘x1a’ y malloc no retorna NULL.				✓
5291	CWE-131		Se considera falso positivo, pues no hay problemas con calcular el valor de ‘x1node’ ya que siempre coincide con el tamaño de ‘x1a->tbl’ y calloc no retorna NULL.				✓
5455	CWE-131		Se considera falso positivo, pues no hay problemas con calcular el valor de ‘x2node’ ya que siempre coincide con el tamaño de ‘x2a’ y malloc no retorna NULL.				✓
2445	CWE-416		Es falso positivo ya que a través del código sólo se libera (‘free()’) a ‘filebuff’, y cuando se hace ‘realloc’ no se está usando ningúna memoria referenciada liberada previamente liberada.				✓
2769	CWE-416		Es falso positivo ya que a través del código sólo se libera (‘free()’) a ‘filebuff’, y cuando se hace ‘realloc’ no se está usando ningúna memoria referenciada liberada previamente liberada.				✓
642	CWE-416		Es falso positivo ya que a través del código sólo se libera (‘free()’) a ‘filebuff’, y cuando se hace ‘realloc’ no se está usando ningúna memoria referenciada liberada previamente liberada.				✓

Reflexión Sobre Herramientas

Para poder elaborar este trabajo, se utilizaron herramientas de análisis automático de código fuente. Funcionaban en base a buscar patrones generales donde era más probable encontrar debilidades. Las herramientas no tenían la capacidad de analizar una línea de código en contexto de las otras o en conjunto, pero aun así nos permitieron agilizar el proceso de análisis, funcionando como una pista inicial sobre dónde empezar a buscar debilidades en el código. Uno de los mayores retos fue encontrar y aprender sobre cada una de las opciones de cada herramienta y cómo utilizar estas opciones para optimizar el proceso.

Para iniciar, Frama-C, que se destaca por su enfoque en garantizar que el código se ajuste a ciertas especificaciones, se presentó como la herramienta que nos trajo más problemas, donde el desorden y redundancia de los archivos que retornaba la herramienta fue la principal. Había muchas líneas que no mostraban debilidades sino información arbitraria, forzandonos a utilizar el comando *grep "^lemon.c:" results.txt > alerts.txt* para tratar de eliminar información innecesaria. Además, las debilidades eran mencionadas de forma muy general, repetía mucho los mismo tipos de debilidades y no mostraba posibles CWE haciendo que esta herramienta fuera muy tediosa de utilizar.

A continuación, con Splint, la herramienta nos pareció muy útil para garantizar que el código siguiera ciertos estándares y presentando errores comunes detectados. Sin embargo, al ser una herramienta más antigua, no estaba tan actualizada con las últimas técnicas y estándares. Al igual que Frama-c, no incluía recomendaciones de posibles CWE y repetía las mismas debilidades varias veces. No obstante, la herramienta sí presentaba dichas debilidades de forma más concreta y explicadas, facilitando nuestro entendimiento de ellas .

Finalmente, las herramientas que nos parecieron más útiles fueron Cppcheck y Flawfinder que tenían un enfoque en detectar problemas de seguridad comunes como el manejo inadecuado de buffers. No obstante, aunque ambas herramientas no realizaban un análisis exhaustivo del código, fueron capaces de mostrar varias debilidades comunes además de mostrar su CWE y presentar información más completa de por qué era una debilidad.

Recomendaciones

1. Para evitar problemas durante el análisis automatizado de código, es recomendable usar archivos fuente que no incluyan archivos personalizados o no estándar. Las herramientas de análisis estático pueden tener dificultades con estas inclusiones, lo que puede resultar en errores de resolución de rutas o interpretación. Al seleccionar un archivo sin dependencias externas o ajustar el código para eliminar estas inclusiones, se facilita un análisis más preciso y se minimizan los riesgos de fallos durante el proceso.
2. No recomendamos la utilización de la herramienta frama-c por múltiples motivos. Primeramente, resulta complejo lograr que se ejecute adecuadamente, ya que sin ciertas opciones da error a la hora de correrlo. Además, cuando se logra ejecutar adecuadamente, da archivos de gran extensión que hay que filtrar para identificar únicamente las debilidades detectadas. Finalmente, las alertas que brinda resultan ambiguas para alguien con relativamente poca experiencia en el área; entonces complica bastante el proceso de identificación de los CWEs.
3. Es importante analizar el código antes de ejecutar las herramientas de detección automáticas. Primeramente, porque comprender el propósito del archivo puede facilitar identificar debilidades manualmente o descartar algunas de las que den las herramientas y segundo, porque es de suma importancia ver que en el contenido del archivo sí puedan detectarse debilidades y no sea como un archivo de configuraciones o algo similar.
4. Es de suma importancia probar distintas opciones de las herramientas porque basándose en los resultados obtenidos con cada una de estas, se puede determinar cuáles son las mejores para completar el objetivo que se busca y cuáles se adaptan mejor al archivo que se está analizando.
5. Analizar con detalle cada uno de los resultados obtenidos e intentar comprender el código para lograr determinar con certeza si se trata de un verdadero positivo o un falso positivo, pues al final eso es el principal objetivo de este ejercicio.