Actividad autocorregible: Análisis estático de código de una aplicación

Recuerda que esta actividad deberá ser completada en la plataforma. Solo tendrás un intento. Una vez que la envíes, se dará por cerrada. Tu puntuación y las respuestas correctas se mostrarán cuando finalice el periodo de entrega de la tarea.

Objetivos

* Analizar el código fuente de una aplicación para poder determinar el nivel de riesgo de las vulnerabilidades encontradas.
* Conocer el tipo de defectos de seguridad que se pueden cometer en lenguajes como C y Java.
* Prepararse para poder analizar el código en base al conocimiento de los defectos de programación que se pueden cometer.

Pautas de elaboración

Esta actividad profundiza en el estudio de la práctica de seguridad del *software* más importante que implantar en un ciclo de vida de desarrollo seguro de un *software* (S‑SDLC): revisión estática de código.

**Planteamiento de preguntas**

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

#define BUFSIZE 256

int main(int argc, char \*\*argv) {

char \*buf;

buf = (char \*)malloc(sizeof(char)\*BUFSIZE);

strcpy(buf, argv[1]);

*}*

A. Stack overflow.

B. Heap overflow.

C. Format string.

D. No existe vulnerabilidad.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

#include <stdio.h>

void printWrapper(char \*string) {

printf(string);

}

int main(int argc, char \*\*argv) {

char buf[5012];

memcpy(buf, argv[1], 5012);

printWrapper(argv[1]);

return (0);

}…

A. Stack overflow.

B. Heap overflow.

C. Format string.

D. No existe vulnerabilidad.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

char\* ptr = (char\*)malloc (SIZE);

if (err) {

abrt = 1;

free(ptr);

}

...

if (abrt) {

logError("operation aborted before commit", ptr);

}

A. Use after free.

B. Heap overflow.

C. Format string.

D. No existe vulnerabilidad.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

bar connection(){

foo = malloc(1024);

return foo;

}

endConnection(bar foo) {

free(foo);

}

int main() {

while(1) //thread 1

//On a connection

foo=connection(); //thread 2

//When the connection ends

endConnection(foo)

}

A. Use after free.

B. Heap overflow.

C. Format string.

D. Memory leak.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

srand(time());

int randNum = rand();

A. Use after free.

B. Heap overflow.

C. Format string.

D. Uso de un generador de números pseudoaleatorios (PRNG) criptográficamente débil.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

private void buildList ( int untrustedListSize ){

if ( 0 > untrustedListSize ){

die("Negative value supplied for list size, die evil hacker!");

}

Widget[] list = new Widget [ untrustedListSize ];

list[0] = new Widget();

}

A. Validación incorrecta del índice del array.

B. Uso de claves criptográficas codificadas.

C. Format string.

D. Uso de un generador de números pseudoaleatorios (PRNG) criptográficamente débil.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

struct hostent \*clienthp;

char hostname[MAX\_LEN];

// create server socket, bind to server address and listen on socket

...

// accept client connections and process requests

int count = 0;

for (count = 0; count < MAX\_CONNECTIONS; count++) {

int clientlen = sizeof(struct sockaddr\_in);

int clientsocket = accept(serversocket, (struct sockaddr \*)&clientaddr, &clientlen);

if (clientsocket >= 0) {

clienthp = gethostbyaddr((char\*) &clientaddr.sin\_addr.s\_addr, sizeof(clientaddr.sin\_addr.s\_addr), AF\_INET);

strcpy(hostname, clienthp->h\_name);

logOutput("Accepted client connection from host ", hostname);

// process client request

...

close(clientsocket);

}

}

close(serversocket);

A. Use after free.

B. Integer overflow.

C. Format string.

D. Buffer Overflow.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

int a = 5, b = 6;

size\_t len = a - b;

char buf[len]; // Just blows up the stack

}

A. Use after free.

B. Integer overflow.

C. Format string.

D. Buffer Overflow.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

void host\_lookup(char \*user\_supplied\_addr){

struct hostent \*hp;

in\_addr\_t \*addr;

char hostname[64];

in\_addr\_t inet\_addr(const char \*cp);

/\*routine that ensures user\_supplied\_addr is in the right format for conversion \*/

validate\_addr\_form(user\_supplied\_addr);

addr = inet\_addr(user\_supplied\_addr);

hp = gethostbyaddr( addr, sizeof(struct in\_addr), AF\_INET);

strcpy(hostname, hp->h\_name);

}

A. Use after free.

B. Integer overflow.

C. Format string.

D. NULL Pointer Dereference.

1. Dado el siguiente código, ¿qué vulnerabilidad contiene?

my $dataPath = "/users/cwe/profiles";

my $username = param("user");

my $profilePath = $dataPath . "/" . $username;

open(my $fh, "<", $profilePath) || ExitError("profile read error: $profilePath");

print "<ul>\n";

while (<$fh>) {

print "<li>$\_</li>\n";

}

print "</ul>\n";

A. Use after free.

B. Manipulación de rutas.

C. Format string.

D. Buffer Overflow.