Ejercicio 3 - Seginf

Manuel Panichelli LU 072/18

Enunciado

```
int func(char* str) {
  char buffer[100];
  unsigned short len = strlen(str);

  if (len >= 100) {
    return (-1);
  }

  strncpy(buffer,str,strlen(str));
  return 0;
}
```

1. Explique cuál es la vulnerabilidad que posee.
2. Explique cómo se puede evitar. Se solicita dar dos explicaciones: por un lado, sugerencias de cambios en el código, y por otro, soluciones externas, suponiendo que no se puede modificar el código.

Resolucion

1. Vuln

Me imagino que la idea del programador fue evitar el *clasico* buffer overflow, con el uso de un string de longitud mayor a 100, que terminaria escribiendo por fuera del buffer potencialmente todas las posiciones del stack que queden debajo de el, como por ejemplo la direccion de retorno.

Pero el problema es que lo chequea, y luego hace el `strncpy` calculando el `strlen` nuevamente. Como la funcion no se ejecuta de forma atomica, en el medio podria cambiar a lo que apunta `*str`, y de esa forma lograr un buffer overflow.

Una forma de vulnerarlo seria probar muchas veces este cambio, y el que no devuelva -1 logro el buffer overflow.

Ejemplo de scheduling e input malicioso que romperia

```
func                  user malicioso

                                  char* str = "hola"
                                  func(&str)

char buffer[100];
len = strlen(str);
// len = 4

if (len >= 100) {
  return (-1);
}
                                  *str = "algun str malicioso ..." (len > 100)

// strlen(str) > 100
strncpy(buffer,str,strlen(str));
return 0;
```

2. Soluciones

La sugerencia mas facil de cambio en el codigo seria calcular una sola vez la longitud, y de esa forma, aunque cambie a lo que apunta, siempre se escribirian los primeros 100 bytes. (a lo sumo, no quedaria null-terminated).

```
int func(char* str) {
    char buffer[100];
    unsigned short len = strlen(str);

    if (len >= 100) {
```

```
        return (-1);
    }

-   strncpy(buffer,str,strlen(str));
+   strncpy(buffer,str,len);
    return 0;
}
```

Si no se puede modificar este codigo, se podrian implementar uno de los mecanismos de proteccion contra buffer overflows que vimos. Lo mas robusto seria implementarlas todas, ya que para lograr vulnerarlo habria que romper todas, lo cual lo hace mas dificil y menos probable.

- ASLR (Address Space Layout Randomization) y DEP (Data Execution Prevention)

  Si bien no previenen el buffer overflow, ayudaria a que no se pueda saltar (ASLR, no sabes donde esta) a una direccion del stack para ejecutar codigo (DEP, no se puede ejecutar) alli.

- Stack Canaries

  Habria que recompilar el programa, y el compilador pondria un *canary* en el stack, que chequearia antes de retornar de la funcion, con lo que se sabria si se sobreescribio el stack y se puede matar al proceso.

  A lo sumo terminaria en denegacion de servicio, pero no habria buffer overflow.