# Buffer overflows y vulnerabilidades de strings de formato

## Buffer overflow

Corrupción de la pila de ejecución provocada por la escritura de una dirección de memoria fuera de rango pero perteneciente al espacio de memoria del proceso. Sobreescribiendo la dirección de retorno de una rutina, el usuario puede ganar el control del flujo de ejecución y adquirir los permisos del proceso vulnerable.

## Strings de formato

Cadena que indica el formato con el que ciertos datos se tratan por parte de funciones como printf. Si se le da control al usuario sobre esta cadena, éste la puede emplear para sobreescribir direcciones de memoria que le den control sobre el proceso y por tanto adquiera sus permisos

# Buffer overflows vs Format Strings

| | **Buffer overflow** | **Format String** |
|---|---|---|
| **Público desde** | Mediados de los ’80 | Junio de 1999 |
| **Peligro reconocido** | Década de los ’90 | Junio de 2000 |
| **Número de exploits** | Varios miles | Varias docenas |
| **Considerado como** | Problema de seguridad | Error de programación |
| **Técnicas** | Avanzadas | Básicas |
| **Visibilidad** | Difíciles de reconocer | Fáciles de hallar |

# Buffer overflows

En la pila de ejecución se almacenan variables dinámicas (por ejemplo las asociadas a una función).

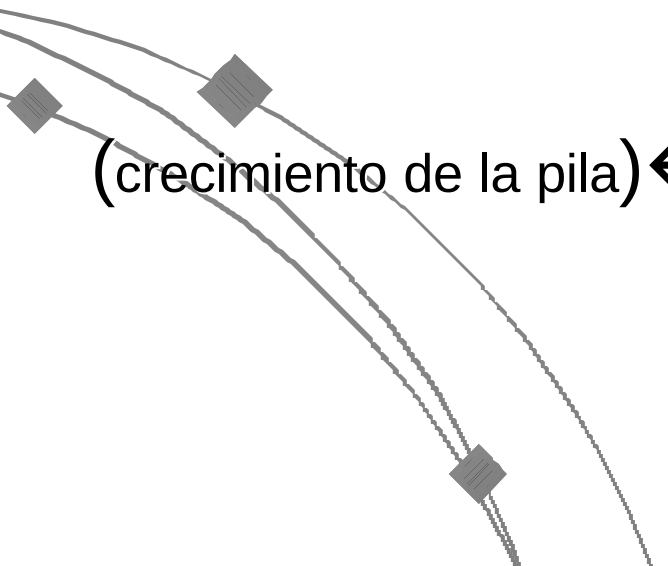
Debido a la falta de comprobación de límites en la escritura sobre un array, puede accederse a la dirección de retorno de una rutina y conseguir que apunte a código suministrado por el usuario.

## Shell de root – Ingredientes

- Un lenguaje sin chequeo de límites
- Código vulnerable (que dé por sentado el tamaño de la entrada y no haga comprobaciones)
- Flag de suid en el ejecutable vulnerable

# Buffer overflows: Estructura de la pila

```
...
char buffer1[256];
funcion(1,2,3);
...
```

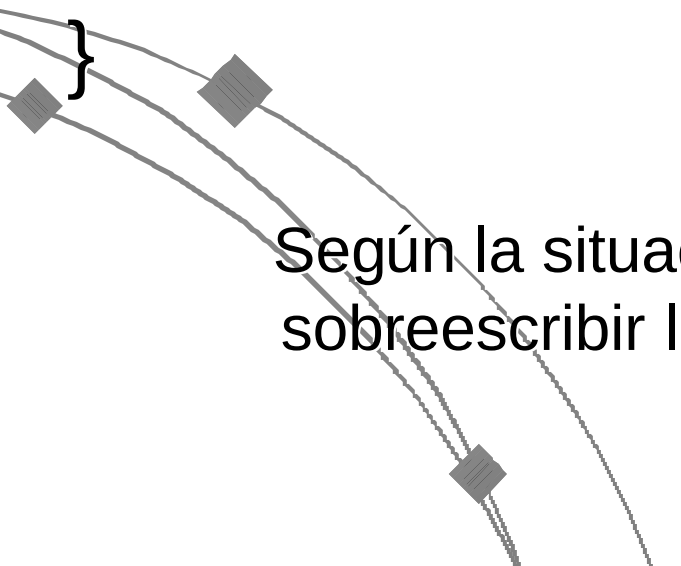


(crecimiento de la pila)←[buffer1][sfp][@retorno][1][2][3]

# Buffer overflows: Código vulnerable

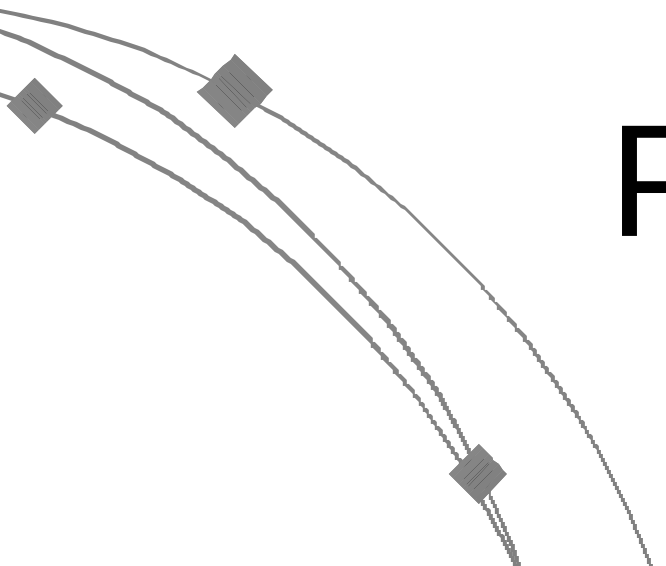
```
void funcion(char *str){
   char buffer[16];

   strcpy(buffer, str);
}
```



Según la situación de buffer en la pila podemos sobreescribir la dirección de retorno de funcion

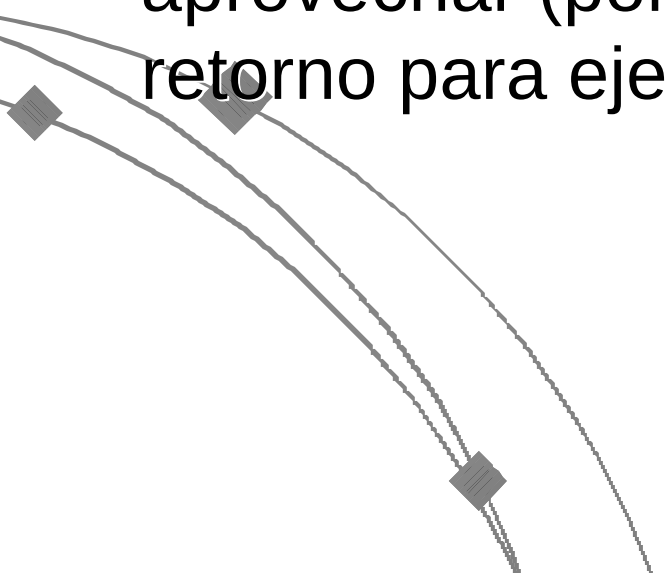
# Buffer overflows:



# Práctica

# Format strings: Teoría

Si el usuario gana el control de una cadena de formato en una llamada como printf, puede escribir cualquier dirección de memoria y aprovechar (por ejemplo) una dirección de retorno para ejecutar su código.



# Format strings: Código vulnerable

```
int funcion(char *usuario){
   printf(usuario);
}

/* Versión correcta */
int funcion(char *usuario){
   printf("%s", usuario);
}
```

# Format strings: Parámetros de formato

| parámetro | salida | pasado por |
|---|---|---|
| **%d** | decimal(int) | valor |
| **%u** | decimal sin signo (unsigned int) | valor |
| **%x** | hexadecimal (unsigned int) | valor |
| **%s** | string((const) (unsigned) char * ) | referencia |
| **%n** | número de bytes escritos hasta ahora, (*int) | referencia |

# String format

# Práctica