P1.1

A variável *size_t* tem 64 bits de tamanho, enquanto que um inteiro apenas tem 32 bits. Sendo assim, o *size_t* poderá alocar até 1,844674407×10¹⁹, com uma máquina de 64 bits, onde um inteiro apenas poderá alocar de [-2147483648, 2147483647].

Após o inteiro chegar a 2147483647, voltará a -2147483648. Como a programação C deixa escrever fora da memória alocada para os arrays, o valor irá ser escrito no endereço matriz[0] - 2147483648.

```
int main() {
  int i = 2147483647;
  printf("Valor que inteiro toma quando chega ao valor 2147483647 -> %d\n", i);
  i++;
  printf("Valor que inteiro toma quando chega ao valor 2147483648 -> %d\n", i);
}
```

A função vulnerável, mesmo com valores como 2147483650 não dá erro, pois a matriz apenas irá escrever num endereço diferente ao esperado. Apenas não irá escrever na posição matriz[2147483648], mas sim noutro endereço (dependendo do intervalo de endereços dado a esse processo, discutido à frente).

Dará erro caso tente ler essa mesma posição, pois a posição matriz[-2147483648] estará do intervalo de endereços dados a esse processo.

P1.2

A variável utilizada para definir o tamanho real a ser copiado está expressa em *size_t*. Esta apresenta problemas no sentido em que não é possível representar números negativos. Desta forma, se for introduzido uma string com comprimento igual a 0, nesta função seria alocado o equivalente a **-1** posições de memória. Como o valor não é representável acaba-se por alocar muito espaço cerca de 2^64.

```
int main(int argc, char *argv[]) {
    char o[MAX_SIZE];
    strcpy(o, argv[1]);
    vulneravel(&o, strlen(o));
}
```

Dada a existência de um limite na quantidade de memória disponível e alocada ao processo, a função falha, apresentando um erro de *Segmentation Fault*. O processo termina sem efetuar a cópia da *String*.

P1.3

A variável utilizada para definir o tamanho está expressa em *size_t*, porém a variável tamanho_real é um *int.* Esta apresenta problemas no sentido em que se o número que for passado no tamanho for um número demasiado grande (superior a 2147483648, visto ser subtraída uma unidade e que o maior número que pode ser guardado numa variável do tipo inteiro é 2147483648), ao realizar a operação que obtém o tamanho_real convertendo o valor anterior num inteiro (e subtraindo uma unidade) iremos obter um valor negativo.

A função malloc no entanto, usa o valor *inteiro* para alocar memória como sendo um *size_t*, assim mesmo tendo obtido um valor negativo na variável tamanho_real, na função malloc este não será negativo uma vez que ao fazer a sua conversão para *size_t* iremos obter o valor pretendido, isto é o valor passado à função subtraindo uma unidade.

Porém quando o mesmo valor é utilizado na função memcpy este será utilizado como sendo um inteiro e os bits que estão em memória na variável irão representar um número negativo pelo que a função irá falhar causando segmentation fault.

```
int main() {
    char* o;
    printf("Call Function”);
    vulneravel(o, 2147483647);
    printf("Call Function");
    vulneravel(o, 2147483648); //última função executada com sucesso
    printf("\Call Function");
    vulneravel(o, 2147483649); //a chamada desta função causa segmentation fault
    printf("Call Function");
    vulneravel(o, 2147483650);
}
```