

CHECKSEC

First of all I install checksec

```
└─$ sudo apt install checksec
The following packages were automatically installed and are no longer required:
icu-devtools libglapi-mesa libpython3.12-minimal python3.12-tk
libflac12t64 libicu-dev libpython3.12-stdlib ruby-zeitwerk
libfuse3-3 liblbfgsb0 libpython3.12t64 strongswan state UP g
libgeos3.13.0 libpoppler145 python3-setproctitle
Use 'sudo apt autoremove' to remove them.

Installing:
checksec 0:11763sec preferred 11763sec

Summary:
Upgrading: 0, Installing: 1, Removing: 0, Not Upgrading: 7
Download size: 23.6 kB
Space needed: 96.3 kB / 62.4 GB available

Get:1 http://kali.download/kali kali-rolling/main amd64 checksec all 2.6.0-2
[23.6 kB]
Fetched 23.6 kB in 7s (3,602 B/s)
Selecting previously unselected package checksec.
(Reading database ... 417539 files and directories currently installed.)
Preparing to unpack .../checksec_2.6.0-2_all.deb ...
Unpacking checksec (2.6.0-2) ...
```

Then we analyze the comando ls:

```
(kali@kali)-[~]
└─$ sudo checksec --file=/usr/bin/ls
RELRO      STACK CANARY      NX      Fortify      PIE      RPATH      RU
NPAT      Symbols      FORTIFY Fortified      Fortifiable      FILE
Partial RELRO      Canary found      NX enabled      PIE enabled      No RPATH      No
RUNPATH      No Symbols      Yes      5      14      /usr/bin/ls

(kali@kali)-[~]
└─$
```

Now we create the file hello.c, then we compile it:

```

(kali@kali)-[~]
└─$ sudo nano hello.c
(kali@kali)-[~]
└─$ ls
2.7  connection_component.py  Desktop  Documents  Downloads  hello.c  leerWebSocket.py  Music  Pictures  Public  __pycache__  Templates  venv  Videos  WebSocketServer.py
(kali@kali)-[~]
└─$ sudo gcc hello.c -o hello
(kali@kali)-[~]
└─$ ll
total 76
-rw-rw-r-- 1 kali kali 1219 Apr 22 11:56 2.7
-rw-rw-r-- 1 kali kali 1027 Apr 23 12:19 connection_component.py
drwxr-xr-x 3 kali kali 4096 Apr 23 11:50 Desktop
drwxr-xr-x 2 kali kali 4096 Apr 22 11:03 Documents
drwxr-xr-x 2 kali kali 4096 Apr 22 11:03 Downloads
-rwxr-xr-x 1 root root 15952 May 19 09:58 hello
-rw-rw-r-- 1 root root 71 May 19 09:58 hello.c
-rw-rw-r-- 1 kali kali 517 Apr 23 12:20 leerWebSocket.py
drwxr-xr-x 2 kali kali 4096 Apr 22 11:03 Music
drwxr-xr-x 2 kali kali 4096 Apr 22 11:03 Pictures
drwxr-xr-x 2 kali kali 4096 Apr 22 11:03 Public
drwxr-xr-x 2 kali kali 4096 Apr 23 12:19 __pycache__
drwxr-xr-x 2 kali kali 4096 Apr 22 11:03 Templates
drwxrwxr-x 5 kali kali 4096 Apr 23 11:54 venv
drwxr-xr-x 2 kali kali 4096 Apr 22 11:03 Videos
-rw-rw-r-- 1 kali kali 2147 Apr 23 12:23 WebSocketServer.py

```

We analyze it with file and checksec

```

(kali@kali)-[~]
└─$ file hello
hello: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=59ede49dcb30147f96252b6f2384a209c1e55162, for GNU/Linux 3.2.0, not strip
ped
(kali@kali)-[~]
└─$ checksec --file=./hello
RELRO      STACK CANARY      NX enabled      PIE enabled      RPATH      RUNPATH      Symbols      FORTIFY Fortified      Fortifiable      FILE
Partial RELRO      No canary found      NX enabled      PIE enabled      No RPATH      No RUNPATH      No Symbols      No      0      0      ./hello

```

The next step is installing jq and try these checksec functions:

```

(kali@kali)-[~]
└─$ checksec --file=./hello --output=json | jq | grep symbols
  "symbols": "yes",
(kali@kali)-[~]
└─$ sudo checksec --file=/usr/bin/ls --output=json | jq
{
  "/usr/bin/ls": {
    "relro": "partial",
    "canary": "yes",
    "nx": "yes",
    "pie": "yes",
    "rpath": "no",
    "runpath": "no",
    "symbols": "no",
    "fortify_source": "yes",
    "fortified": "5",
    "fortify-able": "14"
  }
}

```

To avoid security problems and data loses, we run these command:

```
(kali@kali)-[~]
$ checksec --file=./hello --output=json | jq | grep symbols

"symbols": "yes",

(kali@kali)-[~]
$ gcc -s hello.c -o hello

(kali@kali)-[~]
$ checksec --file=./hello --output=json | jq | grep symbols

"symbols": "no",
```

To try stack and buffer overflow we create a file canary.c, but in the moment of compilation it gives us this issue:

```
(kali@kali)-[~]
$ sudo gcc canary.c -o canary

canary.c: In function 'vuln':
canary.c:9:5: error: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   9 |     gets (buffer);
     |     ^~~~~
     |     fgets
```

I asked ChatGPT to remake the code not using the function gets:

```
#include <stdio.h>
#include <string.h>

void hacked() {
    puts("¡Función que no llama nadie!");
}

void vuln() {
    char buffer[64];

    puts("Stack protection?");
    fgets(buffer, 512, stdin); // <-- MAL USO: permite meter más de 64 bytes

    printf("%s", buffer);
    puts("Ataque buffer overflow");

    fgets(buffer, 512, stdin); // <-- Aquí vuelve a sobrescribirse
}

int main() {
    vuln();
    return 0;
}
```

```
(kali@kali)-[~]
$ ./canary
Stack protection?
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Ataque buffer overflow

zsh: segmentation fault ./canary

(kali@kali)-[~]
$
```

To protect the program when compiling, we add this code in the compilation process:

```
(kali@kali)-[~]
$ gcc canary.c -o canary -fstack-protector-all

canary.c: In function 'vuln':
canary.c:12:5: warning: 'fgets' writing 512 bytes into a region of size 64 overflows the destination [-Wstringop-overflow=]
 12 |     fgets(buffer, 512, stdin); // ← MAL USO: permite meter más de 64 bytes
    |     ^~~~~~
canary.c:9:10: note: destination object 'buffer' of size 64
   9 |     char buffer[64];
     |           ^~~~~~
In file included from canary.c:1:
/usr/include/stdio.h:654:14: note: in a call to function 'fgets' declared with attribute 'access(write_only, 1, 2)'
 654 | extern char *fgets (char *__restrict _s, int _n, FILE *__restrict __stream)
     |                  ^~~~~~
canary.c:17:5: warning: 'fgets' writing 512 bytes into a region of size 64 overflows the destination [-Wstringop-overflow=]
 17 |     fgets(buffer, 512, stdin); // ← Aquí vuelve a sobrescribirse
     |     ^~~~~~
canary.c:9:10: note: destination object 'buffer' of size 64
   9 |     char buffer[64];
     |           ^~~~~~
/usr/include/stdio.h:654:14: note: in a call to function 'fgets' declared with attribute 'access(write_only, 1, 2)'
 654 | extern char *fgets (char *__restrict _s, int _n, FILE *__restrict __stream)
     |                  ^~~~~~
```

Now, when we run the program, it appears this:

[illegible]

And if we prove it with checksec:

```
(kali㉿kali)-[~]  
$ checksec --file=./canary --output=json | jq | grep can  
  
"./canary": {  
  "canary": "yes",
```

Now we run the following commands, these commands use checksec to verify whether the binary has PIE (Position Independent Executable) enabled, and show that compiling with -no-pie disables it.

```

(kali@kali)-[~]
$ checksec --file=./hello --output=json | jq | grep pie

  "pie": "yes",

(kali@kali)-[~]
$ checksec --file=/bin/ls --output=json | jq | grep pie

  "pie": "yes",

(kali@kali)-[~]
$ gcc -no-pie hello.c -o hello

(kali@kali)-[~]
$ checksec --file=./hello --output=json | jq | grep pie

  "pie": "no",

```

Now we run this comando that checks if the binary hello has NX (Non-eXecutable stack) protection enabled, and confirms it with "nx": "yes".

```

(kali@kali)-[~]
$ checksec --file=./hello --output=json | jq | grep nx

  "nx": "yes",

```

We run this commands that when compiling with -Wl,-z,relro,-z,now enables Full RELRO (Relocation Read-Only), improving binary security compared to the default Partial RELR

```

(kali@kali)-[~]
$ gcc hello.c -o hello

(kali@kali)-[~]
$ checksec --file=./hello
RELRO      STACK CANARY      NX      PIE      RPATH      RUNPATH      Symbols      FORTIFY Fortified      Fortifiable      FILE
Partial RELRO  No canary found  NX enabled  PIE enabled  No RPATH  No RUNPATH  36 Symbols  No      0      0      ./hello

(kali@kali)-[~]
$ gcc -Wl,-z,relro,-z,now hello.c -o hello

(kali@kali)-[~]
$ checksec --file=./hello
RELRO      STACK CANARY      NX      PIE      RPATH      RUNPATH      Symbols      FORTIFY Fortified      Fortifiable      FILE
Full RELRO  No canary found  NX enabled  PIE enabled  No RPATH  No RUNPATH  36 Symbols  No      0      0      ./hello

```

Now this comand compiles with _FORTIFY_SOURCE=2 and optimization, but since no fortifiable functions are used, checksec shows no FORTIFY protection applied.

```

(kali@kali)-[~]
$ gcc -D_FORTIFY_SOURCE=2 -O2 hello.c -o hello

(kali@kali)-[~]
$ checksec --file=./hello
RELRO      STACK CANARY      NX      PIE      RPATH      RUNPATH      Symbols      FORTIFY Fortified      Fortifiable      FILE
Partial RELRO  No canary found  NX enabled  PIE enabled  No RPATH  No RUNPATH  36 Symbols  No      0      0      ./hello

```


We upload the canary.c file using fgets instead of gets and deleting the line jump.

```
#include <stdio.h>
#include <string.h>
void hacked () { puts ("Funcion que no llama nadie"); }

void vuln () {
    char bufer[64];
    puts ("stack protection");
    // Reemplazar gets con fgets
    fgets (bufer, sizeof(bufer), stdin);
    // Eliminar el salto de linea del final de bufer
    bufer[strcspn(bufer, "\n")] = '\0';
    printf("\n%s" , bufer);
    puts("\nAtaque buffer overflow");
}

int main(){ vuln(); }
```

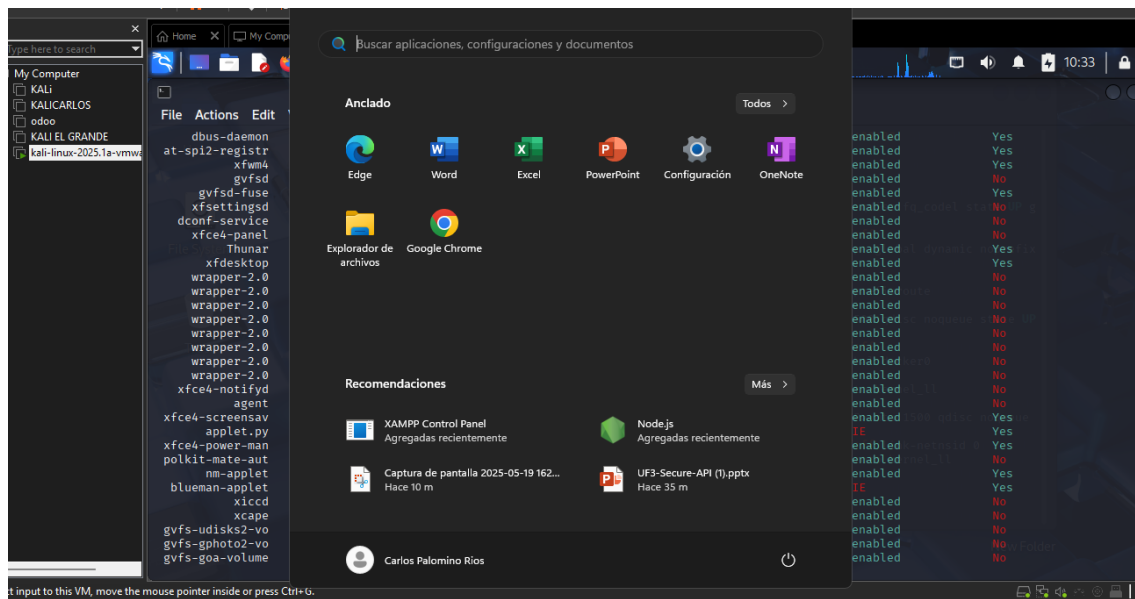
Finally we run the following comands with checksec to see all thats its executing:

```
(kali@kali)-[~]
└─$ checksec --proc=bash
* System-wide ASLR (kernel.randomize_va_space): Full (Setting: 2)
Description - Make the addresses of mmap base, heap, stack and VDSO page randomized.
This, among other things, implies that shared libraries will be loaded to random addresses. Also for PIE-linked binaries, the location of code start is randomized.
See the kernel file 'Documentation/sysctl/kernel.txt' for more details.
* Does the CPU support NX: Yes

COMMAND    PID RELRO    STACK CANARY    SECCOMP    NX/PaX    PIE    FORTIFY
bash 158556 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes

(kali@kali)-[~]
└─$ checksec --proc-all
* System-wide ASLR (kernel.randomize_va_space): Full (Setting: 2)
Description - Make the addresses of mmap base, heap, stack and VDSO page randomized.
This, among other things, implies that shared libraries will be loaded to random addresses. Also for PIE-linked binaries, the location of code start is randomized.
See the kernel file 'Documentation/sysctl/kernel.txt' for more details.
* Does the CPU support NX: Yes
* Core-Dumps access to all users: Not Restricted

COMMAND    PID RELRO    STACK CANARY    SECCOMP    NX/PaX    PIE    FORTIFY
exe 135484 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
zsh 135491 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
exe 139546 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
zsh 139553 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
mpiris-proxy 1409 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
pipewire 1410 Full RELRO    Canary found    Seccomp-bpf    NX enabled    PIE enabled    Yes
pipewire 1412 Full RELRO    Canary found    Seccomp-bpf    NX enabled    PIE enabled    Yes
wireplumber 1413 Full RELRO    Canary found    Seccomp-bpf    NX enabled    PIE enabled    Yes
pipewire-pulse 1414 Full RELRO    Canary found    Seccomp-bpf    NX enabled    PIE enabled    Yes
dbus-daemon 1415 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
gnome-keyring-d 1416 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
xfce4-session 1438 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
at-spi-bus-launcher 1505 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
dbus-daemon 1512 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
at-spi2-registr 1523 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
xfwm4 1540 Full RELRO    Canary found    No Seccomp    NX enabled    PIE enabled    Yes
```



To prove that this is my pc, because the virtual machine is literally a Kali prebuilt VM downloaded from the official website, in this screenshot I show my name Carlos Palomino Rios with the VM behind running the comands of the practice.