

# SEGURIDAD EN SISTEMAS OPERATIVOS

4º Grado en Informática – Complementos de Ing. del Software  
Curso 2019-20

---

**Práctica [2].** Ingeniería inversa y vulnerabilidades.

**Sesión [2].** Explotaciones y protecciones del formato ELF.

**Autor<sup>1</sup>:** Nazaret Román Guerrero

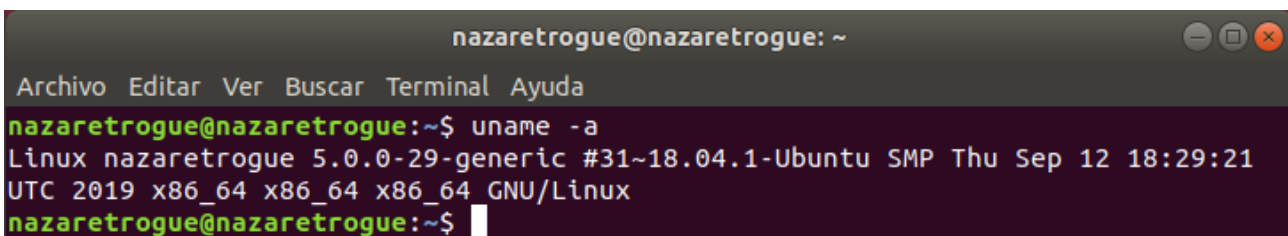
---

## Ejercicio 1.

---

Para el sistema que utilizas, indica la arquitectura, distribución y compilador que utilizas e indica qué protecciones se utilizan de cara a proteger un binario ELF.

Para saber cuál es la arquitectura del sistema que tenemos, se utiliza la orden `uname -a`. La salida de dicho comando es:



```
nazaretroque@nazaretroque: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
nazaretroque@nazaretroque:~$ uname -a  
Linux nazaretroque 5.0.0-29-generic #31~18.04.1-Ubuntu SMP Thu Sep 12 18:29:21  
UTC 2019 x86_64 x86_64 x86_64 GNU/Linux  
nazaretroque@nazaretroque:~$
```

La opción `-a` muestra la misma información que:

- `-s`: nombre del kernel (en este caso `Linux`).
- `-r`: información sobre el lanzamiento del kernel (en este caso `5.0.0-29-generic`).
- `-v`: versión del kernel (en este caso `#31~18.04.1-Ubuntu SMP Thu Sep 12 18:29:21 UTC 2019`).
- `-m`: arquitectura del hardware (en este caso `x86_64`).
- `-p`: tipo de procesador (en este caso, `x86_64`, igual que la opción `-m`).
- `-o`: sistema operativo que se está usando (en este caso, `GNU/Linux`).

Para saber la distribución de nuestro sistema, se utiliza la orden `lsb_release -a`, que muestra la misma información que muestra el comando anterior, ya que nos dice la versión

---

<sup>1</sup> Como autor declaro que los contenidos del presente documento son originales y elaborados por mi. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la “[Normativa de evaluación y de calificaciones de los estudiantes de la Universidad de Granada](#)” esto “conllevará la calificación numérica de cero ... independientemente del resto de calificaciones que el estudiante hubiera obtenido ...”

del sistema operativo. La salida es la que sigue:

```
nazaretroque@nazaretroque:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.2 LTS
Release:        18.04
Codename:       bionic
nazaretroque@nazaretroque:~$
```

Para saber el compilador que tiene nuestro sistema, primero hay que listar todos los que haya instalados. Para ello utilizamos la orden `dpkg --get-architecture | grep compiler`, cuya salida es la siguiente:

```
nazaretroque@nazaretroque:~$ dpkg --get-architecture | grep compiler
ii g++              4:7.4.0-1ubuntu2.3      amd64      GNU C++ compiler
ii g++-7            7.4.0-1ubuntu1-18.04.1 amd64      GNU C++ compiler
ii gcc              4:7.4.0-1ubuntu2.3      amd64      GNU C compiler
ii gcc-7            7.4.0-1ubuntu1-18.04.1 amd64      GNU C compiler
ii libllvm6.0:amd64 1:6.0-1ubuntu2          amd64      Modular compiler and toolchain technologies, runtime library
ii libllvm7:amd64   1:7.3-ubuntu0.18.04.1  amd64      Modular compiler and toolchain technologies, runtime library
ii libllvm8:amd64   1:8.3-ubuntu18.04.1    amd64      Modular compiler and toolchain technologies, runtime library
ii libxkbcommon0:amd64 0.8.0-1ubuntu0.1      amd64      Library interface to the XKB compiler - shared library
nazaretroque@nazaretroque:~$
```

Vamos a comprobar la información detallada de uno de ellos, el primero que aparece en la lista por ejemplo. Para mostrar la información de la versión simplemente utilizamos la opción `--version`:

```
nazaretroque@nazaretroque:~$ g++ --version
g++ (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
nazaretroque@nazaretroque:~$
```

Donde podemos comprobar que corresponde con la versión 7.4.0 de g++ para Ubuntu 18.04.1.

Respecto a las protecciones que ofrece el sistema a los archivos ELF, podemos clasificarlas en las que son del kernel y las que son del compilador.

- Kernel.
  - La distribución 18.04 implementa protección de pila, de heap, ofuscación de punteros, ASLR (pila, mmap, ejecución, VDSO), ejecutables independientes de la posición (PIE) y construcción de programas fortificados, tal y como nos indica la página oficial de Ubuntu, <https://wiki.ubuntu.com/Security/Features>.

- **Compilador.**
  - Protección de pila: utilizado cuando se tiene un programa con un bug que el atacante puede explotar por ejemplo. Se utiliza la opción `-fstack-protector`.
  - Protección de rotura de pila: utilizada cuando puede haber problemas con desbordamientos de búffer que implican memoria y strings. Se utiliza una macro, `FORTIFY_SOURCE`, que chequea el desbordamiento en funciones como `memcpy`, `memmove`, `strcpy`... entre otras.
  - Ejecutables independientes de la posición: el ejecutable se localiza en una posición aleatoria de memoria, que no tiene por qué estar junto al segmento de memoria que contiene el código de variables, librerías... Se utiliza la opción `-fPIC`.

## Ejercicio 2.

---

Podemos mejorar el siguiente virus:

- a) Escribiendo un mejor escáner/limpiador para él.
- b) Añadir `#ifdef` para comprobar la arquitectura de forma que cambie de forma automática el valor `EM_386` en tiempo de compilación para adaptarlo al sistema donde estemos.
- a) Si lo que deseamos es escribir un limpiador mejor para el virus, debemos modificar el script `1x2k2_cleaner`, donde podemos añadir o eliminar instrucciones para adaptarlo a lo que deseamos.

Si lo que deseamos es complicar la búsqueda del virus en los archivos ejecutables, un buen método sería integrar el código malicioso en distintas partes en cada ejecutable, de manera que, si queremos eliminar los fragmentos de código de cada archivo infectado no sería suficiente utilizando el código del cleaner dado, sino que habría que examinar cada línea del archivo para saber dónde comienza y donde finaliza el virus. Eso complica el cleaner, ya que habría que modificarlo de manera que analizara línea a línea los archivos ELF y actuara sobre la parte del fichero en la que está el virus. Esto sería una técnica básica de ofuscación, ya que dificulta la búsqueda de archivos infectados por el simple hecho de que en cada archivo el virus se encuentra situado en una parte diferente del código.

- b) El valor por defecto asignado es para máquinas del tipo 386, así que debemos adaptarlo para otras arquitecturas. Para ello, modificamos el archivo `1x2k2.c` y añadimos directivas `#ifdef` para poder adaptar el virus a diferentes arquitecturas.

Para saber las macros de arquitecturas a las que adaptar el código, debemos mirar el archivo `elf.h`, tal y como indican los comentarios del propio archivo `1x2k2.c`.

El campo `e_machine` del `struct elf32_hdr` y del `struct elf64_hdr` son los que contienen la arquitectura del sistema. Los valores que puede tomar dicho campo se pueden ver en <https://www.sco.com/developers/gabi/2000-07-17/ch4.eheader.html>.

Por ejemplo, en nuestro caso vamos a adaptar el virus para arquitecturas como PowerPC, 64-bits PowerPC, Itanium64 y SPARCV9, cuyo valor para el campo `e_machine` es `EM_PPC`, `EM_PPC64`, `EM_IA_64` y `EM_SPARCV9` respectivamente.

El código añadido quedaría como se muestra en la imagen:

```
#ifdef __POWERPC__
    if (ehdr.e_machine != EM_PPC || ehdr.e_machine != EM_PPC64) return 1;

#elif __ia64__
    if (ehdr.e_machine != EM_IA_64) return 1;

#elif __sparc__
    if (ehdr.e_machine != EM_SPARCV9) return 1;

#else
    if (ehdr.e_machine != EM_386) return 1;

#endif
```

De forma que nuestro virus se adaptaría en tiempo de compilación a la arquitectura que sea. En el caso de ser cualquier otra arquitectura, tomaría el valor por defecto de una máquina 386.