



Facultad de Ciencias Exactas, Ingeniería y Agrimensura
UNIVERSIDAD NACIONAL DE ROSARIO

# Seguridad Informática

# Trabajo Práctico 1

## Alumna:

**Cipullo, Inés**

2023

## Ejercicio 3

Las politicas de seguridad, y en particular su implementación, suelen agregarle restricciones al software. Esto significa, por lo general, que la experiencia del usuario se ve negativamente afectada, lo cual agrega fricción al proceso de venta. Además, se necesita cierto entrenamiento tanto del equipo comercial del producto como de los clientes sobre las politicas.

## Ejercicio 9

Por un lado, el obfuscamiento de código que se encuentra al acceso del público general, una técnica bastante conocida de ocultamiento de la información, se considera como que no agrega apreciable seguridad al sistema. Esto es principalmente porque no es usual que las vulnerabilidades se descrubran y exploten leyendo el código, pero también porque esto no cambia el significado del programa, y eventualmente se lograra deducir, aunque se tarde un poco más porque el código esta obfuscado.

Por otro lado, una práctica de ocultamiento de la información que si agrega seguridad es encriptar los datos, ya sea porque están en tránsito o son archivos que contienen secretos. La encriptación es de gran importancia dentro de la protección de la información.

## Ejercicio 13

### a.

El modelo de Biba para integridad, denominado *Biba Integrity Model* es un sistema formal de estados/transaciones sobre politicas de seguridad computacional que describe un conjunto de reglas de control de acceso diseñadas para asegurar la integridad de los datos (correctitud y completitud de los datos). Los datos y los usuarios son agrupados en niveles de integridad ordenados. El modelo está armado como para que los sujetos no puedan atentar contra la integridad de datos en un nivel mayor al del sujeto, o que la integridad del sujeto no se vea comprometida por datos de un nivel menor que el sujeto.

La idea del modelo es evitar que los datos fluyan de niveles bajos hacia niveles altos, para que los datos de alto nivel no modifiquen su integridad basandose en datos de bajo nivel.

### b.

Se podrían implementar los modelos de Bell-LaPadula y Biba en simultaneo para modelar confidencialidad e integridad a la par, suponiendo la misma cantidad de niveles en ambos. Depende de como se decidan mapear los niveles de seguridad, el sistema permitirá mayor o menor circulación de datos.

Si se mapea el mayor nivel de confidencialidad con el mayor nivel de integridad (y sucesivamente en los menores niveles), resulta en un modelo donde la única circulación de datos habilitada es entre objetos del mismo nivel. No podrá haber circulación de datos niveles distintos porque BLP prohíbe un

sentido, y Biba el otro. En cambio, si se mapea el mayor nivel de confidencialidad con el menor nivel de integridad (y el menor nivel de confidencialidad con el mayor de integridad, y así sucesivamente), resulta en un modelo con circulación de datos entre objetos del mismo nivel y también desde objetos de menor nivel de confidencialidad hacia objetos de mayor nivel de confidencialidad, unidireccionalmente. Esto es porque ese sentido de circulación de los datos está permitido por ambos modelos, según como se acomodaron los niveles.

## Ejercicio 21

Tenemos el siguiente programa:

```
-- Prog 3
x = readH();
writeL("Loading");
while (x-- > 100)
    writeL(".");
writeH(x);
```

Notamos que hay flujo indebido de información de H a L, esto es porque la guarda del bucle `while` depende de la variable `x`, que es de nivel H, pero luego dentro del bucle se escribe sobre variables L. Más explicitamente, la condición del bucle va decreciendo el valor de `x` en cada iteración y luego dentro del bucle se imprime un punto `"."` en un canal de nivel L. Esto significa que si hay un proceso L del otro lado del canal leyendo todos los puntos que se escriben,

podrá saber el valor de `x`, que es de nivel H y cuyo valor es potencialmente un secreto.

## Ejercicio 22

Un Sistema Operativo que implementa el mecanismo de *multi-ejecución segura* (SME) basicamente hará un `fork` de un proceso cada vez que accede a datos con una clase de acceso más alta que la del proceso actual, y le asignará al nuevo proceso la clase de acceso de los datos de mayor nivel.

En el ejemplo presentado en el Ejercicio 21, una ejecución bajo SME suponiendo que hay dos niveles de acceso (L y H) funcionaría como sigue:

- Apenas se lanza el proceso es clasificado como L.
- Al intentar asignarle a `x` un valor H, el OS hace un `fork` y crea un nuevo proceso a nivel H, resultando en dos procesos, $P_L$ (que ya existía) y uno nuevo, $P_H$.
- $P_L$ no tiene permiso para acceder a información H, por lo que el OS va a asignarle un valor constante a la variable `x`, dependiendo de como esté implementado. Luego se ejecutará el bucle sin restricciones, dependiendo del valor de `x`, que podiblemente sea nulo. Finalmente, se esribirá el valor resultante en `x` en un canal H, para lo cual $P_L$ tiene permisos.
- Por su lado, en $P_H$ la variable `x` podrá tomar su valor real ya que habrá permisos para leer información H y se ejecutará el bucle pero sin poder escribir en L, ya que no contará permisos para eso. Finalmente se escribirá el nuevo valor de `x` en H.

En resumen, va a haber dos procesos desde el inicio del programa, $P_L$ y $P_H$ con permisos L y H respectivamente. $P_L$ ejecutará todo el programa pero sin el valor real de `x`, mientras que $P_H$ si tendrá el valor real de `x` pero no podrá ejecutar las sentencias de escritura en L.

## Ejercicio 23

El paso 5 del desafío *Information Flow Challenge* (que se puede encontrar acá) lo resolví de la siguiente manera:

```
try {
    l = true;
    if (h) {throw;} else {skip;}
    l = false;
} catch {
    skip;
}
```

Se logra realizar un leak de información de `h` a `l`. Esto es porque si bien se categoriza el `throw` como H por estar dentro de un `if` con guarda H, y entonces no se podría asignarle valor a una variable L en el `catch`, basta con asignarle el valor a `l` fuera del `if` y luego dependiendo del valor de `h`, tirar la excepción o no. El `throw` entonces funciona como un GOTO, permitiendo saltear cierta parte del código, cambiar de contexto. De esta manera, se mantiene el valor previamente asignado a `l` o se modifica, de acuerdo al valor de `h`.

Un cambio que se le podría hacer a las reglas de tipado para evitar un leak de información como el antes descripto, es no permitir escritura en variables L dentro del `try` luego de una sentencia H. De esa manera se aseguraría que el `throw` no se utilice para cambiar de contexto dependiendo de una variable H, ya que dejaría de haber dos contextos L luego del `throw`.