Practica 1: Introducción a la seguridad informática y la confidencialidad

Entrega de ejercicios seleccionados de la práctica

De *Joaquín Caporalini*
para el curso *Seguiridad informática*
Dictado por *Cristiá, Maximiliano* y
Hernández, Alejandro

1 de octubre de 2024

2. Uso de e-mails corporativos

Enunciado

Las empresas usualmente restringen a sus empleados el uso de e-mails para fines laborales, pero permiten un mínimo uso para razones personales.

1. Piense cómo podría una empresa detectar un uso excesivo de e-mails personales, sin tener que leerlos.
2. Intuitivamente, parece razonable evitar TODO uso personal de e-mail. Explicar porque la mayoría de las empresas no hacen esto.

Propuesta

Es posible implementar tácticas para intentar mitigar el uso de e-mails corporativos para uso personal sin leer el contenido del mail enviado. Los correros van acompañados de metadata que es utilizable para diseñar filtros. Algunas de las técnicas a implementar:

- Comparar el proveedor de mail y el destinatario contra una base de datos de clientes o correos bloqueados, un ejemplo de correo bloqueado podría ser el de la competencia.
- Filtrado por hora de envío y la ip que hizo la petición de enviar el e-mail al servidor de correo.

- Medir la cantidad de interacciones entre el usuario y los correos de la empresa.
- Utilizar sistemas de predicción estadísticos como los utilizados para mitigar los fraudes con tarjetas o el correo SPAM.
- Entrenar modelos de IA para clasificar paquetes con toda la metadata disponible para los mails de la empresa.
-

Las desventajas de implementar sistemas de este tipo son visibles en algunos de los campos que se nombraron (y por eso no se usan cada vez menos con tarjetas y en el marcado de mails como SPAM). Marcar e-mails como *personales* y evitar su envío o recepción puede conllevar en la **perdidas** de nuevas ventas o la perdida de información importante y urgente. Además, la generación de falsos positivos suele llevar a una baja de la **usabilidad** por lo que los empleados terminarían usando otros medios para las comunicaciones y no las direcciones de la empresa. La implementación de estos filtros tiene **costos** aparejados que también deben ser tenidos en cuenta.

7. Compromiso de confidencialidad e integridad

Enunciado

Dé un ejemplo de una situación en la cual un compromiso de confidencialidad conduce a un compromiso de integridad.

Propuesta

Supongamos el caso hipotético en el cual un gerente deja sus credenciales de acceso expuestas físicamente o digitalmente, puesto que son *muy complicadas de recordad* se pegó una nota con el **usuario** y la **contraseña**(que se asume secreta) en su escritorio. Dentro de las capacidades que tiene el gerente es la de modificar las ordenes de trabajo y el manejo de caja, además de poder ver los perfiles de los clientes y modificarlos.

Si dicho usuario y contraseña son comprometidos, perdiendo su estado de **confidenciales**, se podría modificar la información sensible al funcionamiento de la organización, dando un fallo de **integridad**.

15. Programar en Haskell

Enunciado

a) Implementar en Haskell una función:

```
grant :: User -> File -> Access -> Boole
grnat user file bole = undefine
```

Que, dado un usuario (sujeto), un archivo (objeto) y un tipo de acceso (lectura, escritura), devuelva un booleano indicando si se puede llevar adelante el acceso, siguiendo la política del modelo Bell-LaPadula.

OBS: *puede agregar parámetro(s) adicional(es) con la información necesaria para poder devolver lo que corresponda.*

b) Si no lo hizo en el inciso anterior, modifique la función de manera que mantenga el estado de archivos accedidos (abiertos), para tener en cuenta también la propiedad *****.

Propuesta

Para resolver ambos enunciados se propone la siguiente implementación copiando la especificación formal hecha en **Z** en la teoría.

Para `leer` nos debemos de fijar:

1. Obtenemos los archivos abiertos para el `user`.
2. Buscamos los permisos para `user`.
3. Buscamos los permisos para `file`.
4. Hacemos las verificaciones de la especificación:
   - El `user` es del sistema.
   - El `file` es del sistema.
   - no está el El `file` abierto ya para el `user` en ese modo.
   - Se poseen los permisos suficientes
   - Se verifica la condición *****.

```
grant :: SecState -> User -> File -> Access -> Bool
grant (S sc oobj) user file RM
  = let oobju = findWithDefault [] user oobj
        scu = sc ! user
        scf = sc ! file
    in user `member` sc && file `member` sc &&
    notElem file [rf | (rf, acc) <- oobju, acc == RM] &&
    scu `dominates` scf &&
    and [(sc ! wf) `dominates` scf | (wf, acc) <- oobju, acc == WM]
```

Para `Escribir` nos debemos de fijar:

1. Obtenemos los archivos abiertos para el `user`.
2. Buscamos los permisos para `user`.
3. Buscamos los permisos para `file`.
4. Hacemos las verificaciones de la especificación:
   - El `user` es del sistema.
   - El `file` es del sistema.
   - no está el El `file` abierto ya para el `user` en ese modo.
   - Se poseen los permisos suficientes
   - Se verifica la condición *.

Para el caso de las escrituras

```
grant (S sc oobj) user file WM
  = let oobju = findWithDefault [] user oobj
        scu = sc ! user
        scf = sc ! file
    in user `member` sc && file `member` sc &&
    notElem file [wf | (wf, acc) <- oobju, acc == WM] &&
    scu == scf &&
    and [scf `dominates` (sc ! rf) | (rf, acc) <- oobju, acc == RM]
```

Sindo así que los casos son muy similares.

21.3. Flujo indebido

Enunciado

Determinar si hay flujo indebido de información en cada uno de estos programas.

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

Propuesta

En este ejemplo **hay** flujo indebido de información. El contenido de la variable `x` en si mismo no es *filtrado*, pero si su uso, rompiendo la idea de que los usuarios de *bajo* nivel no deben de ser interferidos por las acciones de nivel superior.

22.3.

Enunciado

Para los programas del ejercicio anterior, explique cómo funcionaría la ejecución bajo el sistema de seguridad por multi-ejecución.

Propuesta

Tomemos la traza del **programa 3** (el utilizado en el ejercicio anterior)

1. El programa seria iniciado con clasificación **L**.
2. Se ejecutan sentencias de ese nivel hasta llegar a la primera linea mostrada.
3. Al intentar ejecutar

   ```
   x = readH();
   ```

el programa realiza un `fork()`. En proceso padre continua con permisos **L** y el nuevo sigue su ejecución con permisos **H**.

4. Según el nivel pararan distintos comportamientos para las sentencias

```
writeL("Loading...");
while (x-- > 100)
    writeL(".");
```

   - **L**: El programa imprimirá los mensajes.
   - **H**: El programa no podrá imprimir nada porque lo haría en un nivel inferior.

5. Llega el momento de ejecutar la ultima sentencia

```
writeH(x);
```

   - **L**: este programa no vera ninguna modificación.
   - **H**: El programa escribe el valor de la variable puesto que estamos en el entorno **H** y se hace en un lugar clasificado como **H**.

23.4.

Enunciado

Resuelva los desafíos propuestos en el siguiente link: http://ifc-challenge.appspot.com/ Para los que pueda resolver, explique las conclusiones a las que arribe, por ejemplo:

- Qué tipo de ataque logró llevar adelante?
- Qué limitación tiene el sistema de reglas?
- Qué cambio habría que hacerle para impedir el ataque?

Propuesta

El ejercicio que elegí fue el 6 donde el ataque venia de la mano de `try <stm>catch <stm>` y `throw;`.

Pude reproducir el ataque de la siguiente forma:

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

En este ataque a la confidencialidad una variable de nivel de seguridades superior `h` es vulnerada puesto qeu su valor es copiado en una variable con nivel de seguridad inferior `l`. De esta forma podríamos filtrar secretos para desclasificarlos. Una forma de evitar este ataque es que si se accede a la lectura de variables seguras no se permita escribir en esos entornos otras variables con niveles inferiores, elevando así a *high* los permisos del **try** y **catch**

A. Apendice de código

A continuación el código completo para el ejercio 15.

```
import Data.Map

type Cat = String
type Obj = Int
type SL = Int
type SC = (SL, [Cat])

type User = Obj
type File = Obj

dominates :: SC -> SC -> Bool
dominates (sl1, scat1) (sl2, scat2) = sl2 <= sl1 && scat2 `subset` scat1
  where
    subset list1 list2 = all (`elem` list2) list1

data Access = RM | WM deriving (Show, Eq)

data SecState = S {
  sc :: Map Obj SC,
  oobj :: Map User [(File, Access)]
}
```

```
grant :: SecState -> User -> File -> Access -> Bool

grant (S sc oobj) user file RM
  = let oobju = findWithDefault [] user oobj in
        scu = sc ! user
        scf = sc ! file
    in user `member` sc && file `member` sc &&
    notElem file [rf | (rf, acc) <- oobju, acc == RM] &&
    scu `dominates` scf &&
    and [(sc ! wf) `dominates` scf | (wf, acc) <- oobju, acc == WM]

grant (S sc oobj) user file WM
  = let oobju = findWithDefault [] user oobj in
        scu = sc ! user
        scf = sc ! file
    in user `member` sc && file `member` sc &&
    notElem file [wf | (wf, acc) <- oobju, acc == WM] &&
    scu == scf &&
    and [scf `dominates` (sc ! rf) | (rf, acc) <- oobju, acc == RM]
```