Seguridad en Software Libre

Carlos Sarraute – Ariel Futoransky

7 junio 2005

USUARIA 2005

Motivación

- Hay alguna diferencia estratégica perceptible entre la seguridad de soluciones de código abierto vs. código cerrado?

Plan de la charla

- Que son las vulnerabilidades?
- Como se descubren?
- El proceso de reporte
- Tres ejemplos de la vida real
- Análisis

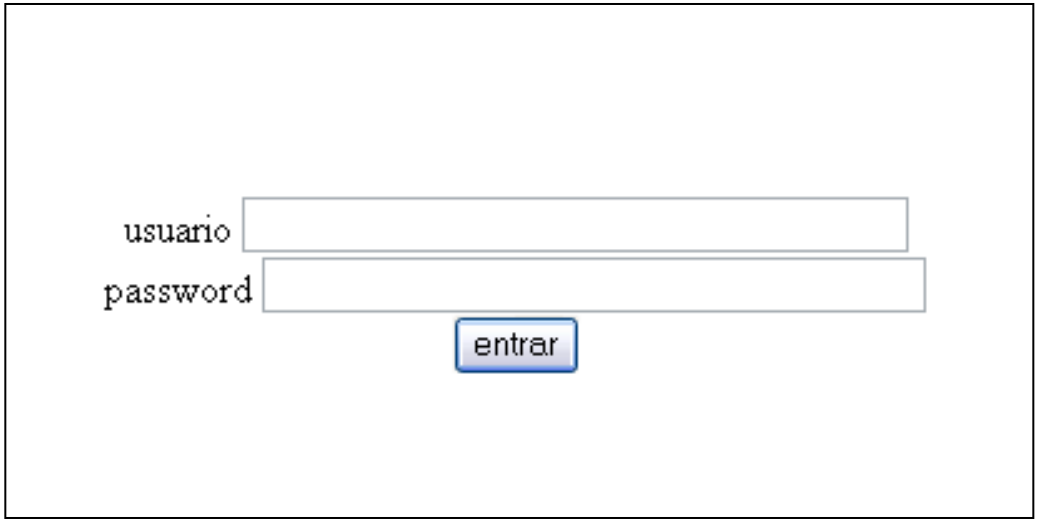
Vulnerabilidades y seguridad

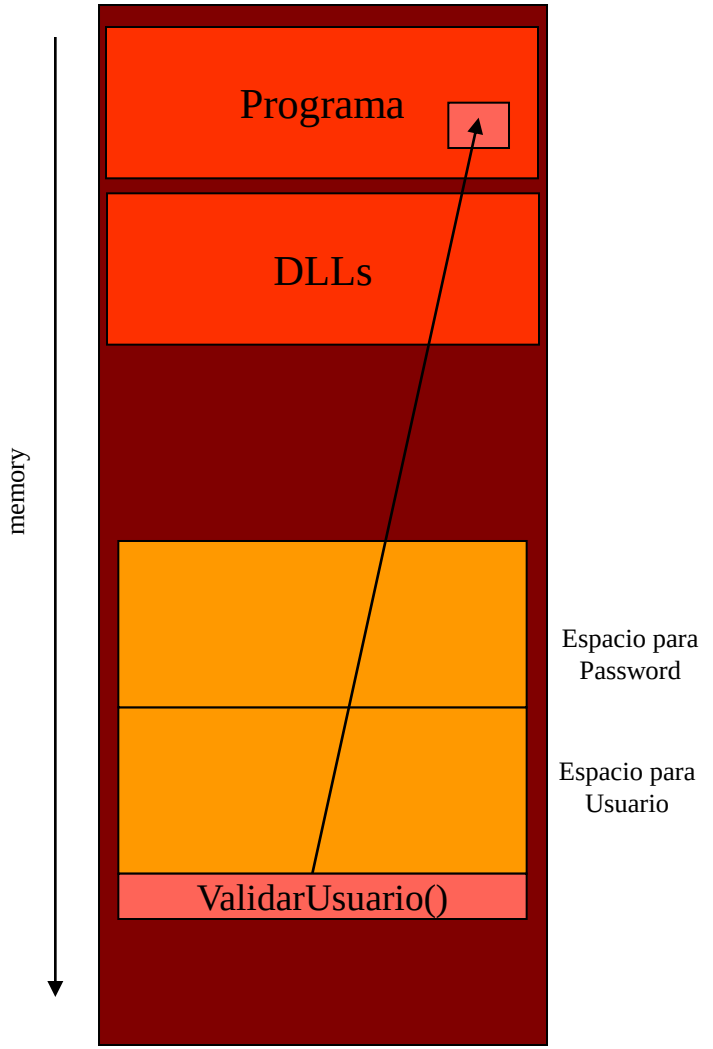
- Aspectos del software con influencia en seguridad:
  - Herramientas de seguridad disponibles
  - Vulnerabilidades & exploits

Qué es una vulnerabilidad?

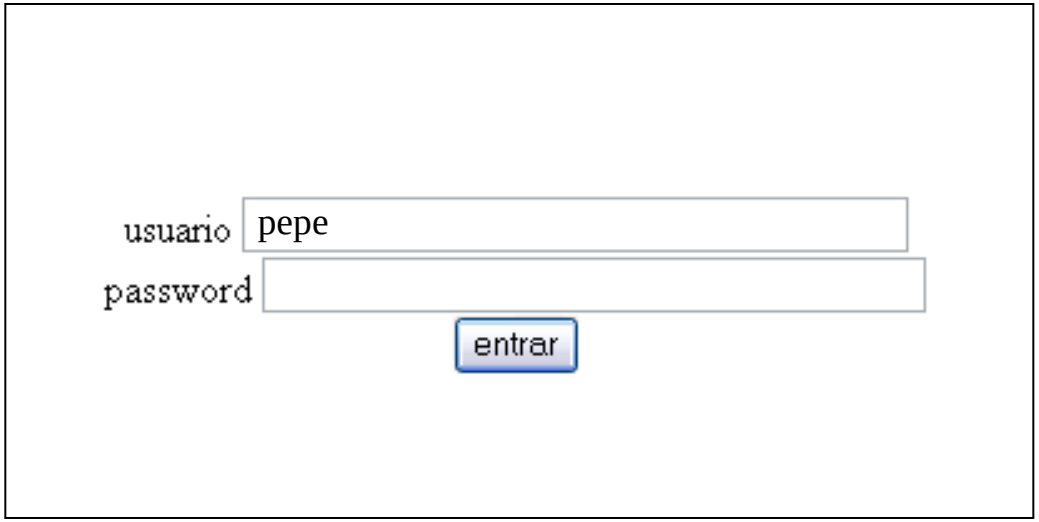
- Informalmente:
  - No es un error, es un “feature” escondido
  - opción de menú escondida
- Exploits: método o programa para sacar provecho de uno de estos “features”
- Ejemplo paradigmático: el buffer overflow

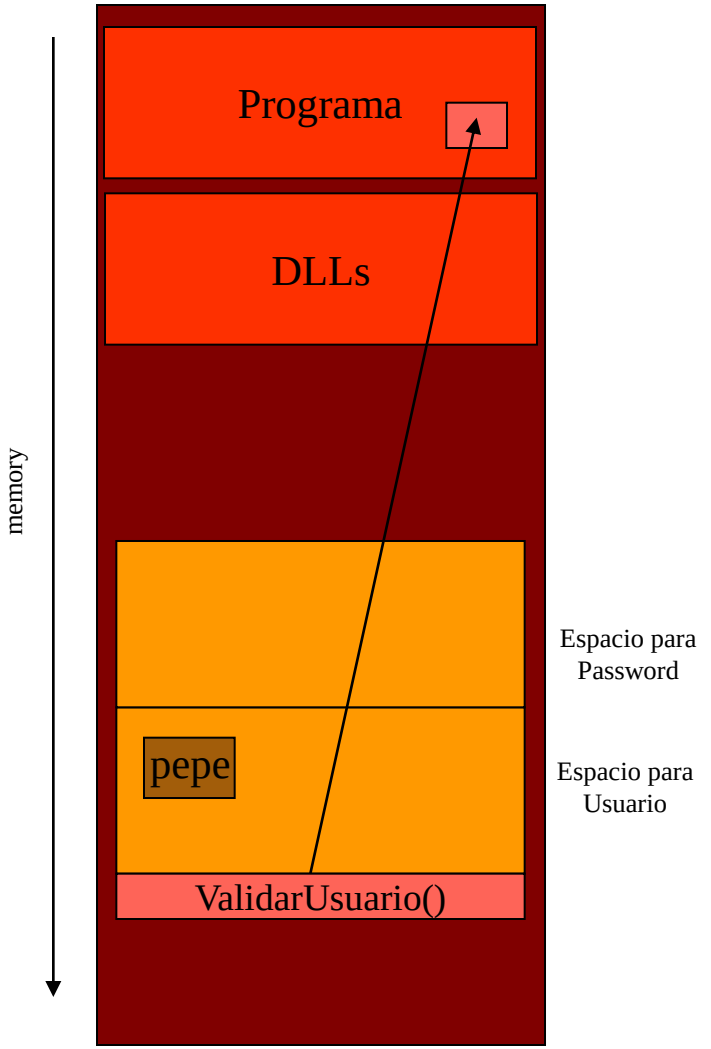
Buffer overflow en el stack (la pila)



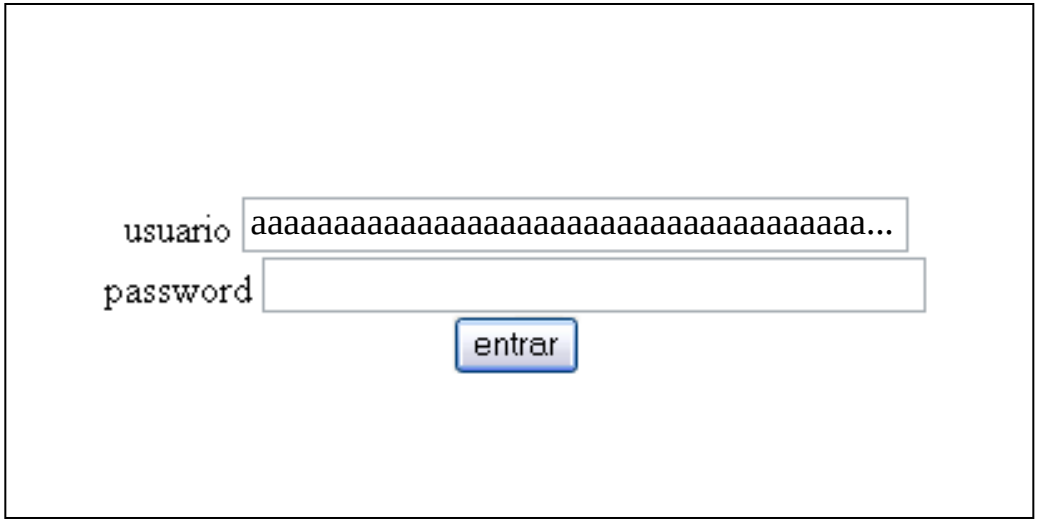


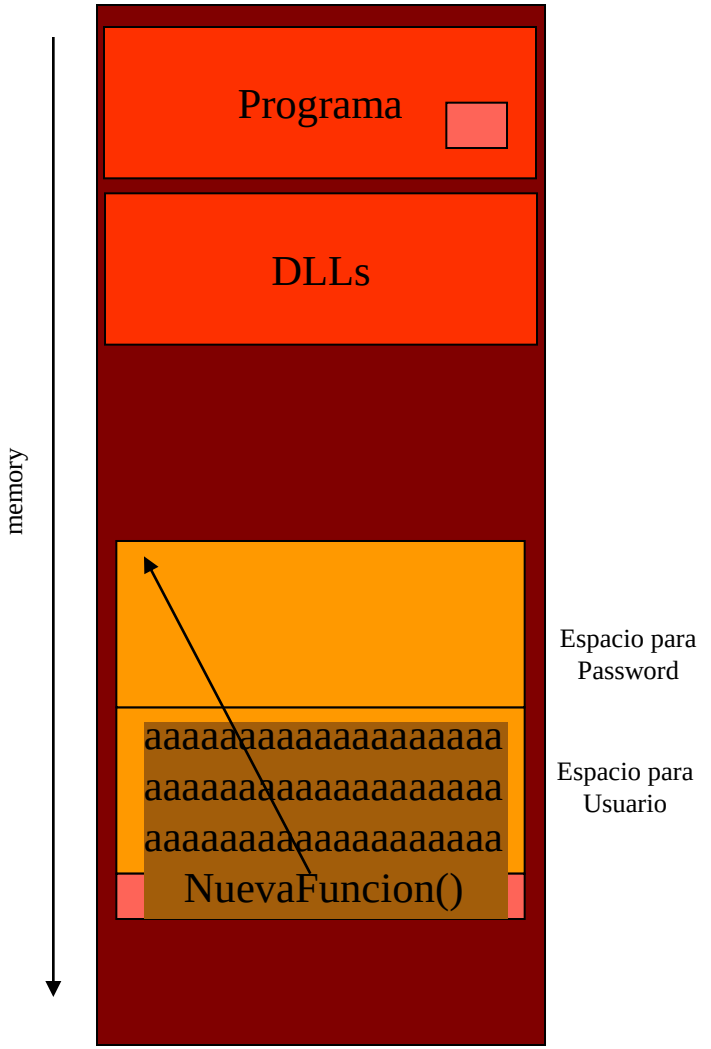
Buffer overflow en el stack (la pila)





Buffer overflow en el stack (la pila)





Como se descubren vulnerabilidades?

- El atacante / investigador / consultor descubre vulnerabilidades:
  - Buscando patrones comunes vulnerables
  - por azar: un blue screen es un accidente feliz...
  - Transpolando una nueva vulnerabilidad desde otra aplicación
  - Nuevos técnicas, nuevas ideas (inspiración)
  - Manipulando las posibles entradas del programa examinado

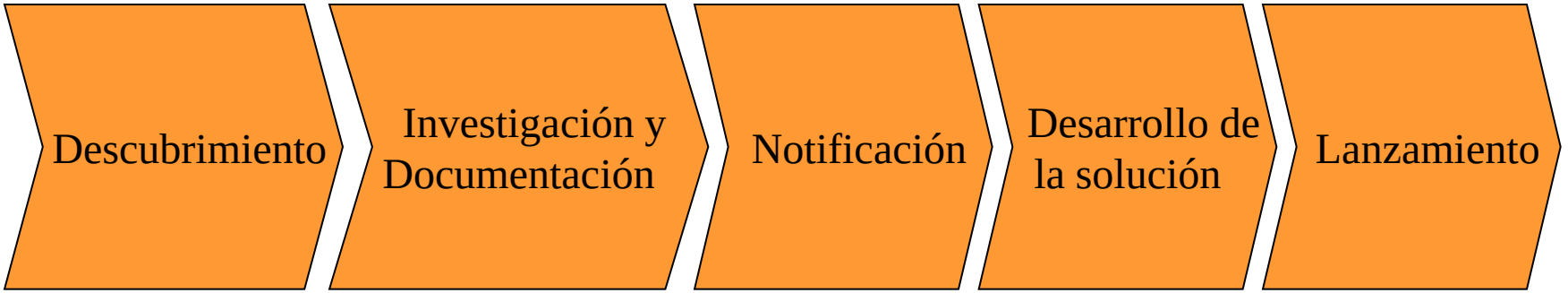
Diferentes técnicas para buscar vulnerabilidades

- Con acceso al código fuente
  - auditoria de código
  - buscar patrones de vulns conocidas
  - hacer una búsqueda exhaustiva del código por patrones sospechosos

- Sin acceso al código fuente
  - dar input al azar ó con forma sospechosa
  - usar un fuzzer
  - ingeniería inversa para entender como funciona (si esta disponible el binario)
    - » herramientas: desensambladores, debuggers, …

Proceso de reporte de vulnerabilidades

- actores del proceso
  - Investigadores que descubren la vulnerabilidad
  - Los autores del software vulnerable
  - Centro de coordinación como el CERT
  - Usuarios



Características del proceso

- Respuesta del fabricante
  - antes era más lenta y aleatoria
  - ahora los vendedores importantes tienen un departamento que se ocupa de responder a los reportes de vulns
  - fabrica un patch (remedio a la vulnerabilidad) independiente ó integrado a una nueva versión
- Respuesta del usuario
  - Antes no había procesos automáticos de actualización
  - El impacto y alcance del problema influye en la urgencia definida entre el fabricante y el investigador, y asumida por los usuarios

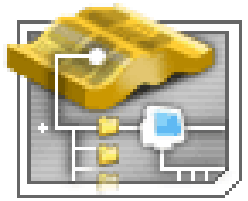
Ejemplos de la vida real

- En nuestro laboratorio venimos haciendo investigacion en seguridad desde 1996
- 45+ advisories publicados por Core

- Tres casos de vulnerabilidades que descubrimos:

- ejemplo 1: Active Directory
- ejemplo 2: Snort
- ejemplo 3: MSN Messenger

Ejemplo 1: vulnerabilidad en Active Directory



- Active Directory es un componente esencial de Windows 2000 Server y siguientes
- permite a las organizaciones administrar y compartir recursos en una red, actuando como autoridad central de la seguridad de la red
- El problema descubierto:
  - pedido con mas de 1000 operaciones “AND”
  - causa un Stack Overflow en el servicio
  - Y provoca un crash del servidor (denial of service)

Active Directory: cronograma de comunicaciones

- 2003-05-16 CORE notifica a Microsoft
- 2003-05-19 Microsoft notifica CORE de que el problema está resuelto en SP4
- 2003-06-26 lanzamiento de Windows 2000 Service Pack 4
- 2003-07-02 se publica nuestro advisory

- probamos el exploit desarrollado en nuestro laboratorio (para win2000 sp3) con el service pack 4
- resultado: es vulnerable!!

Active Directory: cronograma 2

- 2003-08-11 CORE notifica Microsoft de que la vulnerabilidad no está arreglada
- 2004-04-13 Microsoft publica Microsoft Security Bulletin MS04-011

- La vulnerabilidad arreglada en SP4 era para pedidos de tipo:

  AND (cond1) (cond2) (cond3) ...

- La vulnerabilidad que habíamos encontrado era para pedidos del tipo:

  AND (AND (AND (cond1) (cond2)) (cond3)) (cond4) ...

Active Directory: conclusión

- al no tener acceso al código fuente, los investigadores no pueden determinar exactamente donde está el problema
- descubren síntomas, no necesariamente la causa
- los ingenieros del fabricante arreglan los síntomas que les reportaron
- Los investigadores no tuvieron acceso a la solución antes de la publicación

- El valor de un exploit: nos dimos cuenta del problema porque teníamos forma de explotar realmente la vulnerabilidad

Ejemplo 2: vulnerabilidad en Snort

- Snort es un IDS (intrusion detection system)

  muy popular, open source
- detecta cientos de ataques analizando los paquetes que recibe y aplicando un conjunto de reglas de pattern matching

- encontramos un heap overflow
- mandando trafico “malicioso” en una red donde está Snort corriendo
- se puede explotar y ejecutar comandos arbitrarios en la máquina que corre Snort

Snort: la vulnerabilidad

- if((spd->seq_num + spd->payload_size) <= s->last_ack)

  offset = spd->seq_num - s->base_seq (offset = 0xffdc)

  memcpy(buf + offset, spd->payload, spd->payload_size)

- detección precisa de donde ocurre la vulnerabilidad y bajo que condiciones

- 2003-03-21 Core notifica a los autores de Snort
- 2003-04-15 Publicación del patch y del advisory

Snort: conclusión

- Validamos el patch previo a su publicación (código fuente)
- las nuevas vulnerabilidades en aplicaciones open source son cada vez más dificiles de encontrar y explotar
- posteriormente, los autores de Snort nos pidieron una auditoría del código fuente de Snort

Ejemplo 3: vulnerabilidad en MSN Messenger

- popular programa de mensajería instantánea
- usado por 130 millones de personas

- vulnerabilidad en el parseo de imágenes PNG
- permite a un ataque ejecutar código arbitrario en la máquina donde corre el Messenger

Messenger: cronograma

- 2004-08-04: Publicación del advisory y del patch para la librería open source libPNG
- 2004 agosto: los distribuidores SuSE, Mandrake, Gentoo, RedHat, Conectiva sacan patches para esta vulnerabilidad

- Experimentamos con un patron de ataque similar en Messenger y resulto vulnerable
- la vulnerabilidad se debe a un problema en la libPNG (que es open source!)

- 2004-08-23: Notificación de Core al vendedor
- 2005-02-08: Publicación de los patches y advisories

Messenger: meses de testing...

- Varios programas vulnerable, que necesitaron rondas de testing en múltiples plataformas y configuraciones:
  - MSN Messenger 6.1
  - MSN Messenger 6.2
  - Windows Messenger 4.7.2009
  - Windows Messenger 4.7.3000
  - Windows Messenger 5.0
  - Windows Media Player 9 series (CVE CAN-2004-1244)

- Vulnerabilidad muy crítica, que requirió hacer obligatorio la actualización a la nueva versión de Messenger

Messenger: el worm potencial

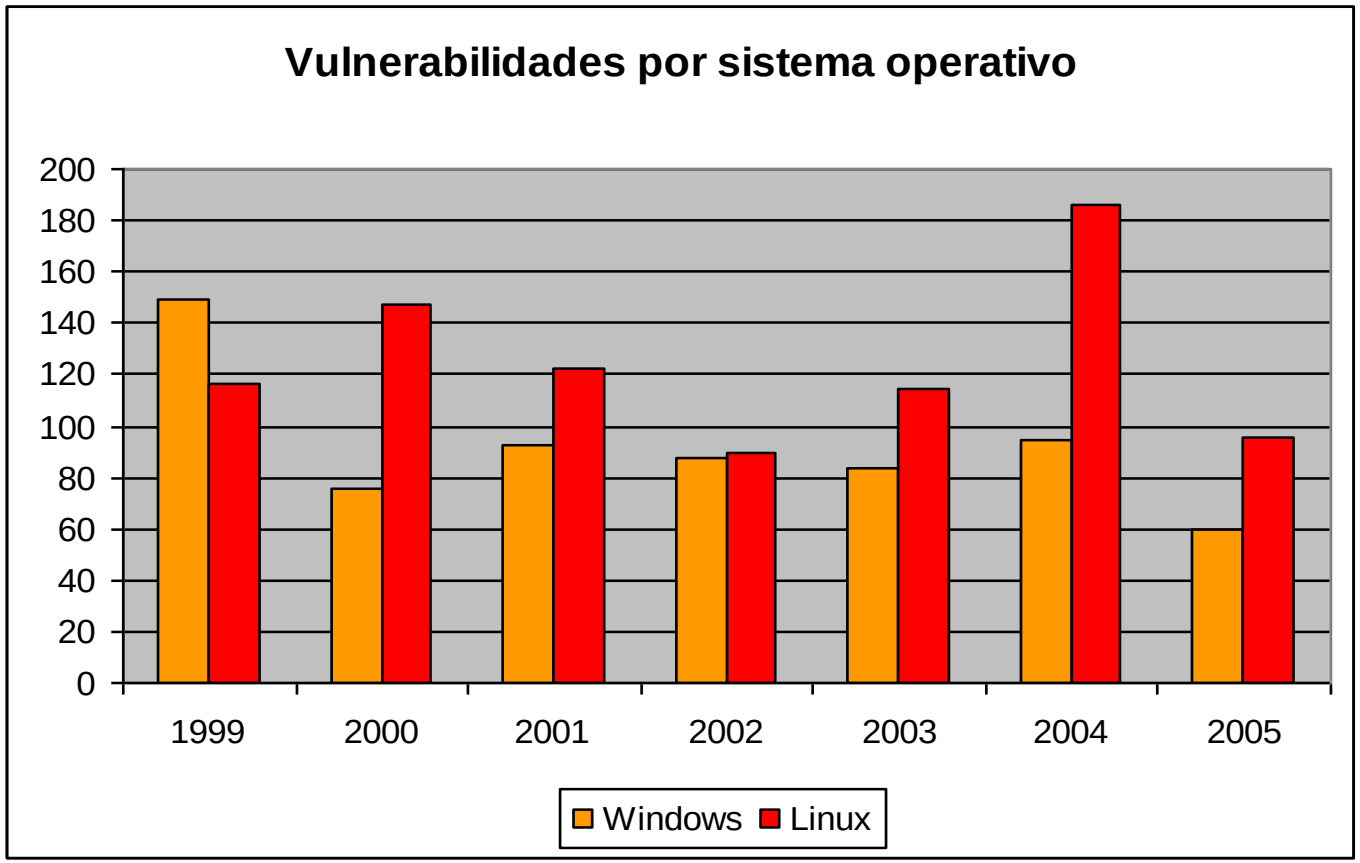
- Messenger usa imágenes con formato PNG para mostrar:
  - la imagen del usuario
  - íconos que muestra en el texto (caritas)
  - thumbnail de las imágenes que se transmiten

- Con un PNG malicioso, un atacante puede explotar un buffer overflow y ejecutar código arbitrario en la máquina de la víctima

- No genera actividad ni tráfico sospechoso
- Worm se podría diseminar sin ser detectado por la red de usuarios de Messenger

Análisis

Estadísticas de vulnerabilidades reportadas (fuente CVE)



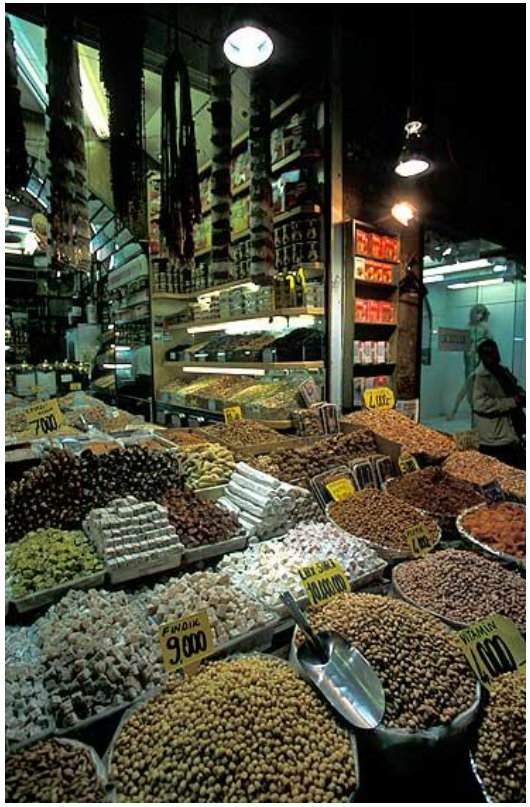
De la vulnerabilidad al ataque

- Severidad de la vulnerabilidad
- Complejidad de construcción del exploit
  - en Linux, los exploits son cada vez mas complejos
- Precisión lograda
- Disponibilidad del ataque
  - Acceso a los exploits
  - Acceso a maquinas vulnerables
- Ventana de exposición

Ventajas de tener acceso al código fuente

- calidad del reporte de alguien que tiene acceso al código

- "La catedral y el bazar"
  - ensayo de Eric Raymond
  - la catedral es el software de código cerrado, elaborado en una empresa con estructura jerárquica
  - el bazar es el software open source, desarrollado por múltitud de hackers y programadores



De la dificultad de reportar y corregir bugs

- un mismo error puede tener múltiples síntomas, dependiendo de los detalles del entorno y forma de operar del usuario
- es difícil para el programador reproducir las sutilezas del entorno en el cual el tester encontró el bug
- en el caso open source, el tester y el desarrollador pueden compartir su modelo mental del programa
- el tester encuentra la fuente del bug en el código, se puede eliminar de un golpe decenas de síntomas relacionados

Conclusiones

- Código fuente disponible implica:
  - El investigador puede proponer una solución
  - Puede validar el patch antes de su publicación
  - Resultado: se reducen los tiempos y mejora la calidad
- La diferencia entre las distribuciones de vulnerabilidades descubiertas es significativa
- Las herramientas de ingeniería inversa están reduciendo la brecha entre buscar vulnerabilidades sin código y con código
- Aunque cabe esperar innovaciones tecnológicas importantes para auditoria de código fuente

Unos enlaces…

- www.coresecurity.com/corelabs
- www.coresecurity.com/corelabs/advisories
  - Active Directory Stack Overflow
  - Snort TCP Stream Reassembly Integer Overflow Vulnerability
  - MSN Messenger PNG Image Parsing Vulnerability
- www.securityfocus.com
  - Bugtraq

CORE SECURITY TECHNOLOGIES

GRACIAS!

Carlos Sarraute
carlos@coresecurity.com
Ariel Futoransky
ariel.futoransky@coresecurity.com