Bughunting

el arte de buscar vulnerabilidades en software

Carlos Sarraute

José Orlicki

Pedro Varangot

Core Security Technologies

ECI 2007 – DC - UBA

Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct
13"ᵤc (032) MP - MC ~~1.9821647000~~ ~~2.13047764915 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.13047764915
correct 2.13067764915
Relays 6-2 in 033 failed special speed test
tm relay " 10.00 test.
Relays changed
Relay 2145
Relay 3370
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545  Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
~~1700~~ 1630 Arctangent started.
1700 closed down.

CORE LABS RESEARCH & DEVELOPMENT

Quienes somos?

- Trabajamos en CoreLabs, el laboratorio de investigación de Core Security Technologies.

- La empresa hace:
- Core Impact, herramienta para penetration testing.
  - 500 clientes en 40 países
- Consultoría
  - pen tests
  - auditoría de código
- Investigación
  - en CoreLabs
  - en otras áreas le dedican parte de su tiempo.

Agenda

- Introducción.
- Vulnerabilidades en binarios:
  - como explotar un buffer overflow
- Como se descubren vulnerabilidades?
- Proceso de reporte
  - ciclo de vida de una vulnerabilidad

Introducción

Preguntas motivadoras

- ¿Que es la seguridad en software?
  - Garantía de que el software no hace *más* ni *menos* de lo deseado.
  - Es una escala, casi continua, de riesgos, y se incorpora al proceso de desarrollo de software.
  - La seguridad es un arte de lo particular, empieza en los limites de los modelos generales.
- ¿Porque buscamos fallas de seguridad?
  - Los sistemas informáticos son cada vez más complejos (*n+1* capas entre usuario y hardware) y no hay garantía por construcción, solo por *resistencia a ataques*.
  - Es divertido, es un desafío intelectual.
  - Es (in)moral, de acuerdo al destino de los hallazgos.

Preguntas motivadoras

- Que son las vulnerabilidades?
  - Cuestiones que se obviaron en diseño/implementación del software.
  - Permiten ampliar o reducir la funcionalidad del sistema, muchas veces sin las credenciales apropiadas.
- Como se descubren?
  - Azar, auditoría de diseño/código o inspiración.
- Que pasa con las fallas encontradas?
  - Se reportan al responsable para ser resueltas, o no.
  - Se convierten en *exploits*, que muestran el alcance y consecuencias de la falla.
  - Hasta que no haya un patch publico, son *exploits zero-day*, es decir amenazas inesperadas a la seguridad.

¿Qué es una vulnerabilidad?

- Es la posibilidad de desviarse del comportamiento esperado de un sistema, las fallas son evidencia de su existencia.
- En la mente del investigador:
  - No es un error, sino un posible “feature” escondido.
- Exploit: método o programa para sacar provecho de uno de estos “features”, hacerlos efectivos.
- El buffer overflow es el ejemplo paradigmático de vulnerabilidad en binarios ejecutables.
  - veamos algún caso donde explotamos la vulnerabilidad.

Vulnerabilidades en binarios

Ejemplo de buffer overflow

- Vamos a ver como explotar un buffer overflow **básico**
- Vamos a explotar un binario para windows de coretex03.c, el ejemplo está tomado de la competencia Coretex 3 que se realizó en Noviembre del 2006
- Coretex 3 fue la primer competencia que realizó CoreLabs dedicada 100% a la explotación de código vulnerable. Más información en http://coretex.coresecurity.com

Ejemplo de buffer overflow

- La consigna de la competencia era tomar el control de programas vulnerables para que estos impriman “explotado!”

- El tercer programa a explotar era el siguiente:

Código de coretex03.c

```
- int main(int argc, char *argv[])
- {
-  int x,y;
-  char buf[50];
-
-  scanf("%d", &y);
-  x = y;
-  gets(buf);
-  x -= y;

-  if (x == 5) printf("explotado!\n");
- }
```

Conceptos basicos

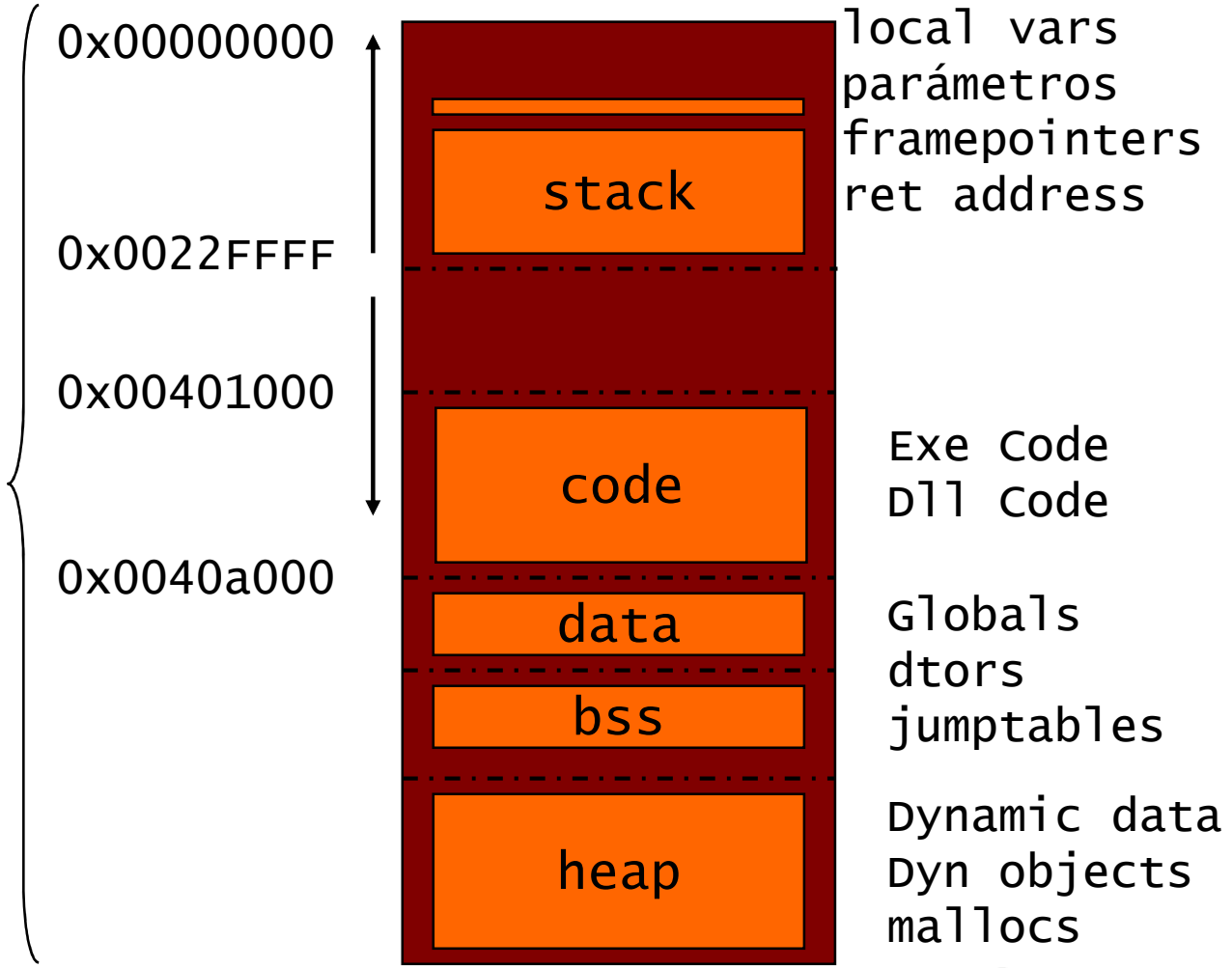
- Para entender como funciona un buffer overflow es necesario entender varios conceptos de “bajo nivel”, inherentes al funcionamiento del SO, el compilador, y a la arquitectura sobre la que estos corren.
- No vamos a explicar todo esto ahora, ya que vamos a ver un ejemplo corto de solo unos minutos.
- Igual repasemos algunos de los conceptos básicos…

Registros y memoria

Registros:

- eip — Instruction pointer
- esp — stack pointer
- ebp — frame pointer
- General Purpose
- flags

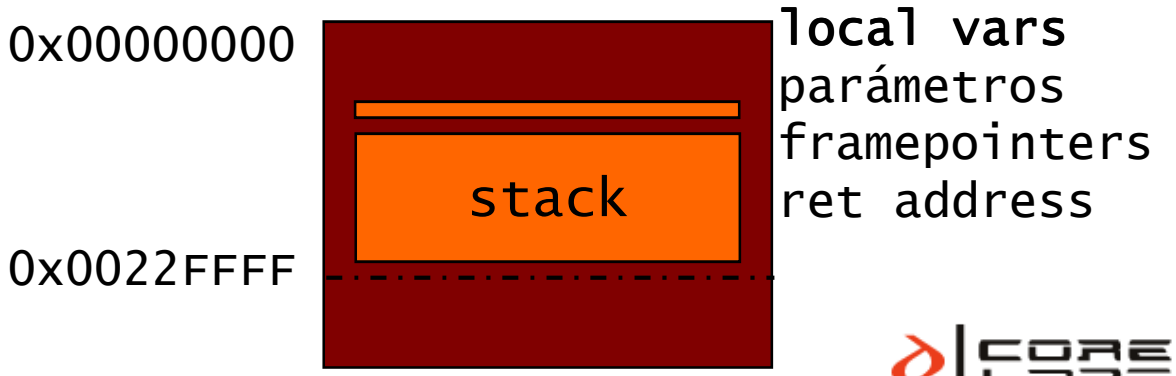
Memoria:



Variables locales

- Las variables estáticas o automáticas de nuestro programa van a estar en el stack

```
int main(int argc, char
*argv[])
{
   int x,y;
   char buf[50];
```



Variables locales

- Al compilar un programa el código generado reserva espacio en el stack para todas las variables locales.
- Por ejemplo nuestro programa reservaría 50 bytes para el arreglo, y 64 bits para los dos enteros.
- El stack siempre crece **de abajo hacia arriba.** O sea, lo primero que se guarda, es lo que mas abajo va a quedar, y lo último que va a salir.

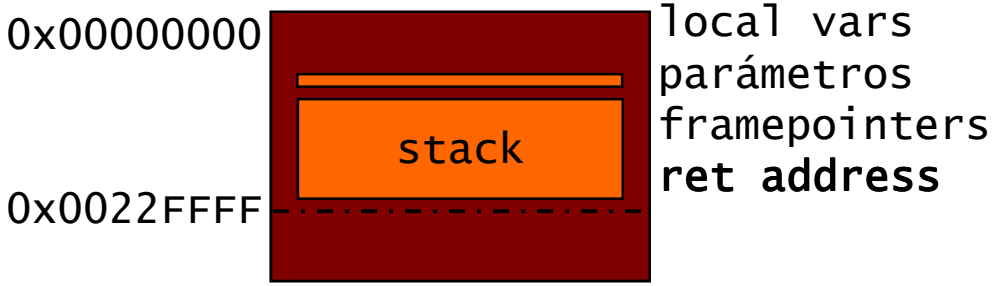
Flujo de control

- Pero el stack también se utiliza para recordar adonde debe volver el flujo de control luego de una llamada a función

Antes de llamar a la función Main el SO guarda el eip en el stack…

```
int main(int argc, char
*argv[])
{
    if (x == 5)
        printf("explotado!\n");
}
```

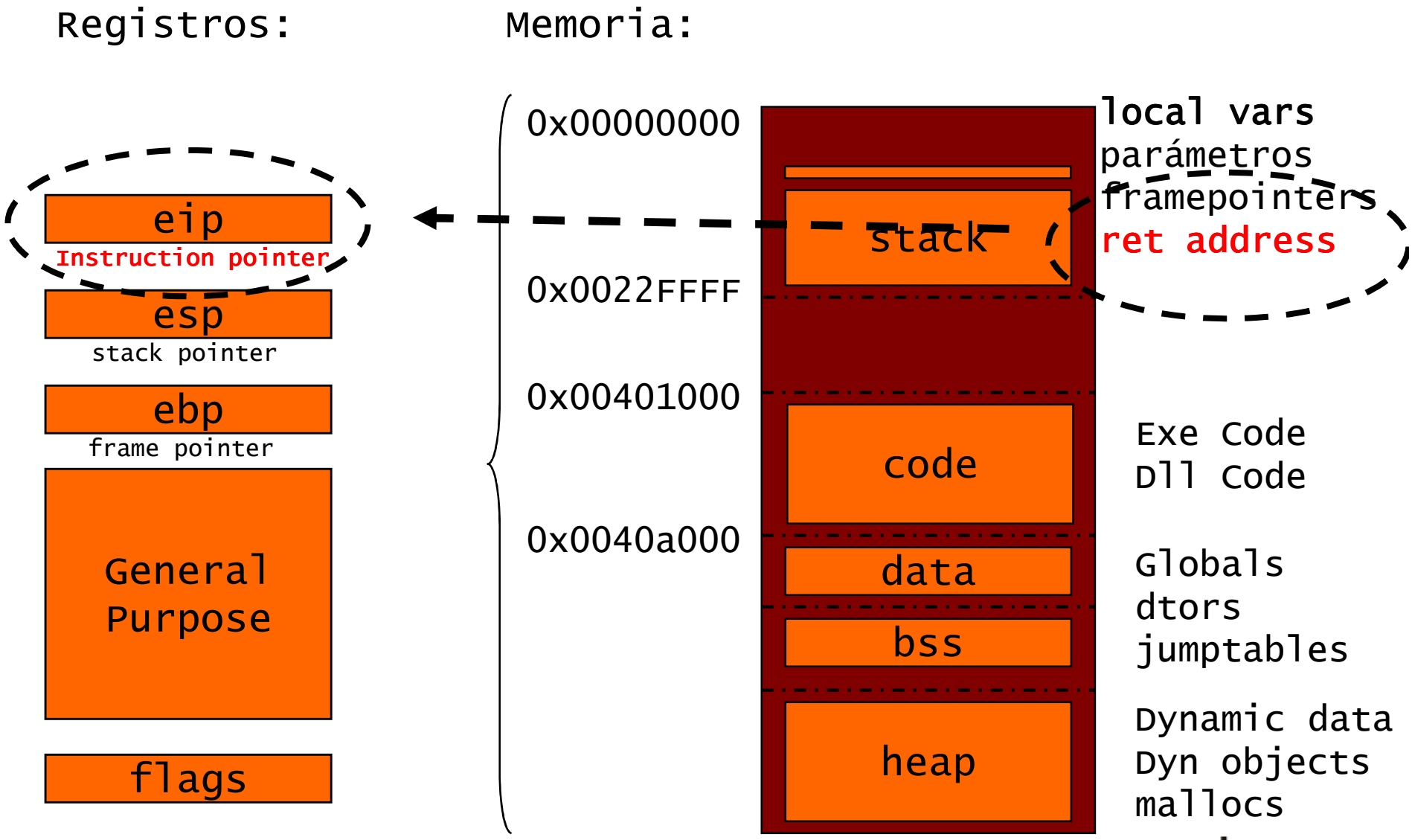
para luego de ejecutarla resumir el flujo normal de ejecución.



Flujo de control

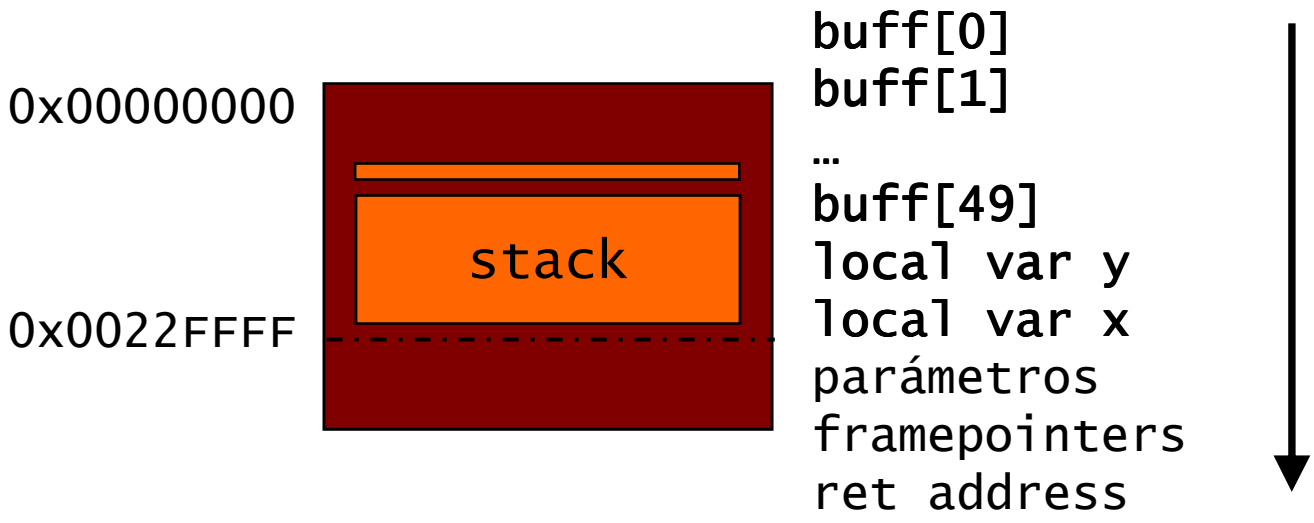
Significado	Pseudo-código	Assembler
cambia el eip y transfiere el control del programa al nuevo eip.	eip ← 00401020	jmp 401020
transfiere el control del programa a una función	push eip eip ← 00401020	call 401020
termina una función y devuelve el control del programa	pop eip	ret

Flujo de la información



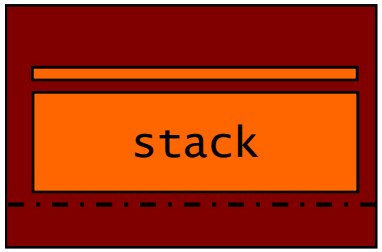
Explotando un buffer overflow

- Como, por ejemplo, un arreglo se escribe **de arriba hacia abajo** en la memoria**…**



Explotando un buffer overflow

- Si un programa escribe en una variable local algo de longitud arbitraria sobre lo que tenemos control, podemos sobreescribir el eip antiguo, y cuando termine la función **¡saltar el flujo de ejecución adonde querramos!**



local vars
parámetros
framepointers
ret address

```
int main(int argc, char *argv[])
{
  int x,y;
  char buf[50];

  scanf("%d", &y);
  x = y;
  gets(buf);
  x -= y;

  if (x == 5) printf("explotado!\n");
}
```

Ejercicios en vivo!

- Ejemplo explotado!

Ejercicios en vivo!

- Si podemos ejecutar en el stack, podemos también ejecutar código propio vía otro programa vulnerable.
- Como escribimos en el stack, podemos escribir ahí código ejecutable. Después pisamos el eip, y ponemos ahí la dirección del mismo lugar del stack donde acabamos de escribir.
- Ejemplo notepad.exe

Como se descubren vulnerabilidades?

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

Auditoría de código

- Con acceso al código fuente
  - RTFS = read the ¿fine? source
  - buscar patrones de vulns conocidas
  - hacer una búsqueda exhaustiva del código por patrones sospechosos
  - requiere mucho tiempo y esfuerzo

Reverse Engineering

- Sin acceso al código fuente
  - ingeniería inversa para entender como funciona (si esta disponible el binario)

- Herramientas
  - Desensambladores como el IDA (interactive dis-assembler)
    - » requiere mucho tiempo y esfuerzo
  - Debuggers
    - » seguir la ejecución paso a paso

Fuzzear

- Fuerza bruta
- Dar input al azar ó con forma sospechosa
- Un fuzzer es una herramienta diseñada especialmente para eso
- Web server
  - mandarle requests malformados
  - basandose en bugs de seguridad conocidos
- Appliance de red

El proceso de reporte

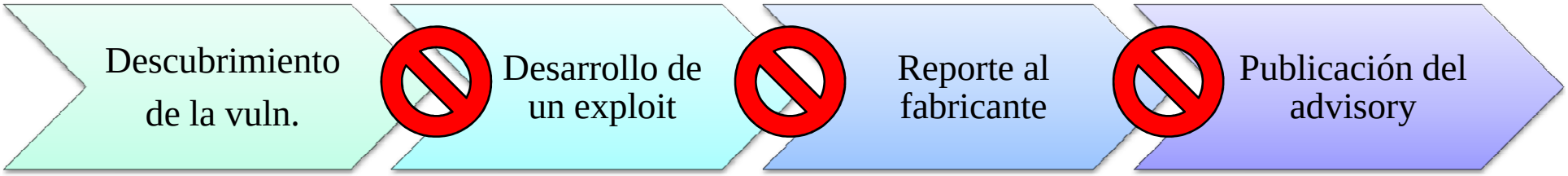
Proceso de reporte de vulnerabilidades

- Actores del proceso
  - Investigador que descubre la vulnerabilidad
  - Fabricante del software vulnerable
  - Usuarios
  - Centro de coordinación como el CERT

- Core ha sido un participante activo desde 1996

Del lado del investigador

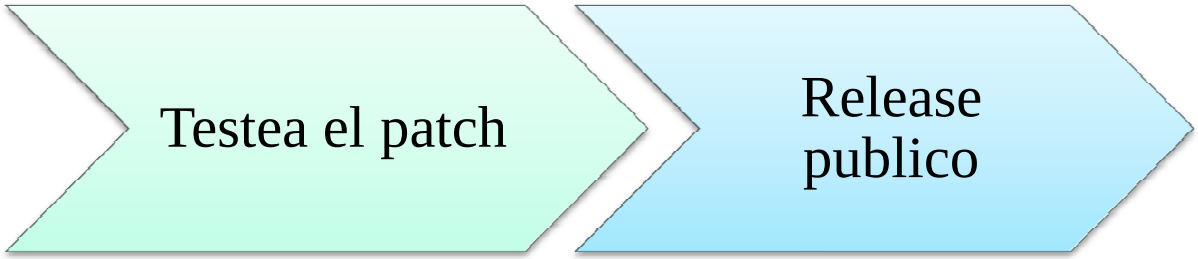
- El investigador después de descubrir una vulnerabilidad:
- puede desarrollar un proof of concept y/o un exploit
- puede reportar al fabricante
- puede hacer publico un advisory describiendo la vulnerabilidad



ZERO DAY

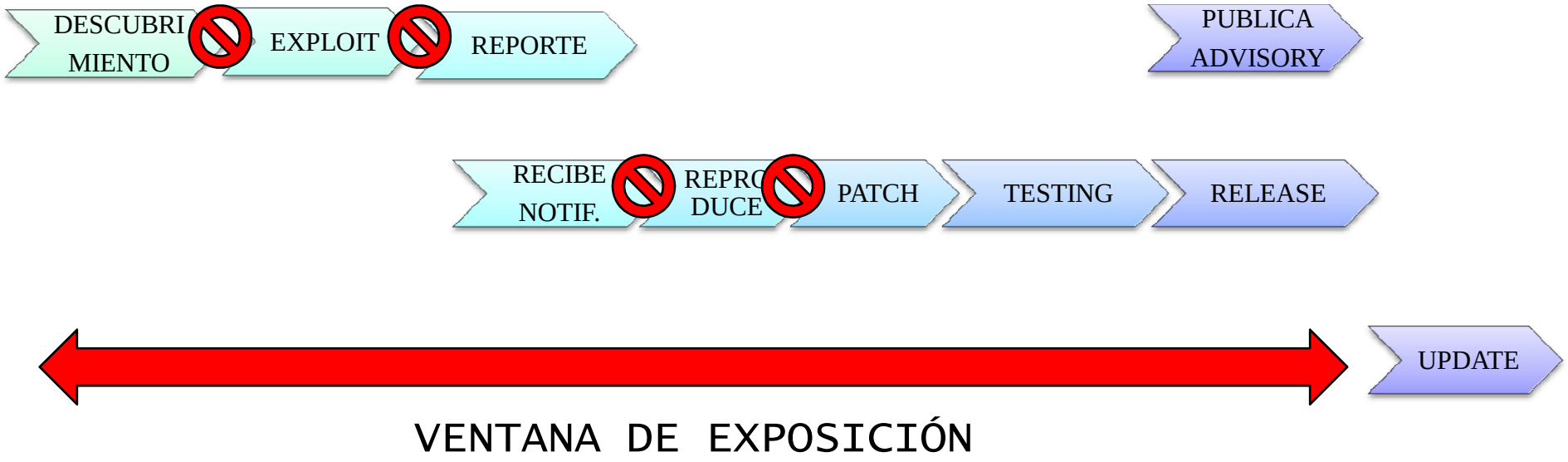
Del lado del fabricante y los usuarios

- El fabricante del software:



- Los usuarios:

Ventana de exposición



Nuevos actores desde 2004

- WabiSabiLabi – desde 2007
  - http://www.wslabi.com/
  - mercado de vulnerabilidades tipo eBay
- Zero Day Initiative (ZDI) – desde 2005
  - programa de TippingPoint, una división de 3Com
  - http://dvlabs.tippingpoint.com/

- iDefense VCP (Vuln Contributor Program) – desde 2004
  - http://labs.idefense.com/vcp/

Política de disclosure de Core

- No hay Zero-days en Impact.

- “Ethical disclosure” de las vulnerabilidades:
  - reportar la vulnerabilidad al fabricante con el maximo de información para que pueda reproducir el bug.
  - colaborar en la elaboración y testeo del patch (caso opensource).
  - publicar el advisory en forma coordinada con el release del patch.

- Parte de las dificultades a sortear:
  - coordinar con los fabricantes
  - presión de los clientes para tener info antes de que sea publica
  - presión de los fabricantes para que cierta info no se haga publica

Política de disclosure de Core

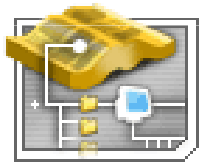
- Creemos que es la forma correcta de proteger los usuarios del software vulnerable.
- Dar en el advisory toda la información que un usuario necesita para:
  - entender el impacto de una vulnerabilidad
  - determinar si es vulnerable
  - patchear sus sistemas / workaround para minimizar el impacto

- Seguir la filosofía del “full disclosure” en forma responsable.
  - analisis tecnico detallado de la vulnerabilidad y forma de explotarla
  - forma de compartir conocimientos y contribuir a la comunidad

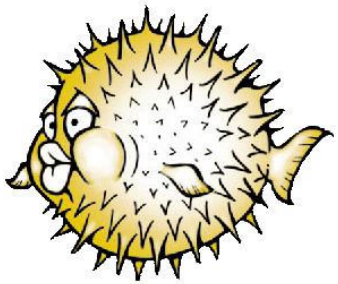
Dos experiencias

- En nuestro laboratorio venimos haciendo investigación en seguridad desde 1996
- 55+ advisories publicados por Core

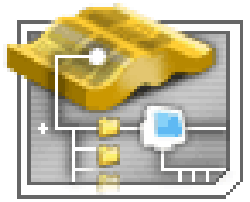
- Dos casos de vulnerabilidades que descubrimos:



- Ejemplo 1: Active Directory
- Ejemplo 2: OpenBSD



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 por Eduardo Arias y su equipo de Bugweek:
  - 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

- 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:

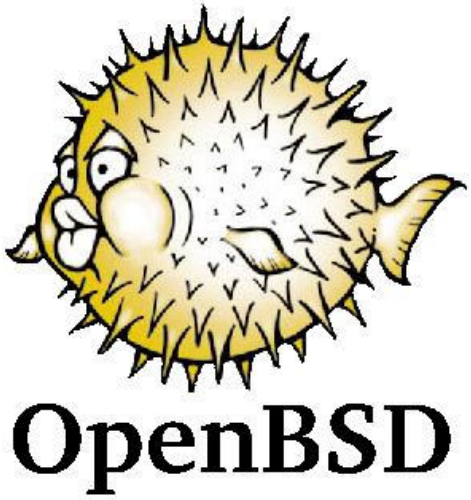
  AND (AND (AND (cond1) (cond2)) (cond3)) (cond4) ...

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 OpenBSD



- OpenBSD es un sistema operativo tipo UNIX
- desarrollado con el foco puesto en la seguridad
  - "Only one remote hole in the default install, in more than 10 years!"

- Alfredo Ortega encontró una vulnerabilidad que resulta en corrupción de memoria en el kernel de OpenBSD
  - en el codigo que maneja paquetes IPv6
  - mandando paquetes ICMPv6 fragmentados, un atacante puede provocar un overflow de estructuras mbuf (en memoria de kernel) que resultan en ejecución remota de codigo arbitrario.

OpenBSD: cronograma

- 2007-02-20: notificación enviada por Core.
- 2007-02-21: Core manda un borrador de advisory y un PoC que demuestra remote kernel panic.
- 2007-02-26: el equipo de OpenBSD desarrolla un fix y lo comitea en el source tree (sin avisar)
  - detección muy veloz del codigo vulnerable
  - lo considera “reliability fix”

- 2007-03-05: Core desarrolla un codigo PoC que demuestra ejecución remota de código en el kernel, explotando el overflow de mbuf.

OpenBSD: Vulnerabilidad remota en kernel

- 2007-03-13: Core publica el advisory.

- El equipo de OpenBSD cambia la calificación a “security fix” y de hecho cambia la home page:

Only **two remote holes** in the default install, in more than 10 years!

OpenBSD: Conclusión

- Incluso un sistema operativo con foco en seguridad como OpenBSD tiene vulnerabilidades
- Protocolo IPv6 es menos usado y también menos testeado
  - Lugar interesante para buscar vulnerabilidades
- Técnica usada: ganar control del flujo de ejecución sobreescribiendo un puntero a una función.
  - Exploit interesante.
  - Presentación de Alfredo Ortega en Black Hat.

- La importancia de desarrollar un exploit
  - demuestra la existencia y el impacto de la vulnerabilidad.
  - argumento irrefutable!

www.coresecurity.com/corelabs

Jose Orlicki - termo@coresecurity.com
Carlos Sarraute - carlos@coresecurity.com
Pedro Varangot - peter@coresecurity.com