EKO-PARTY 2008

Inyección de codigo en máquinas virtuales

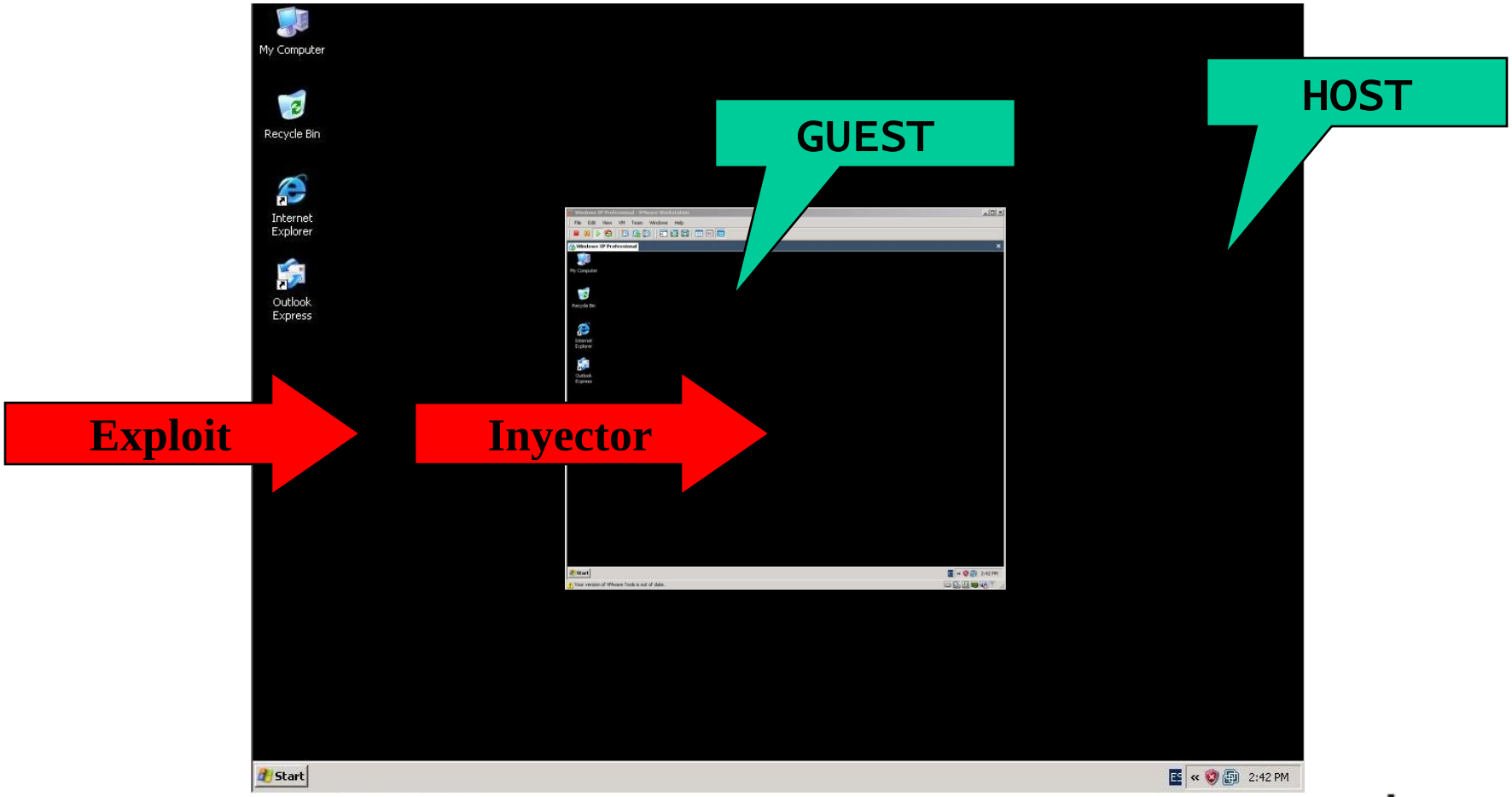
Por Nicolás A. Economou

Algunos terminos

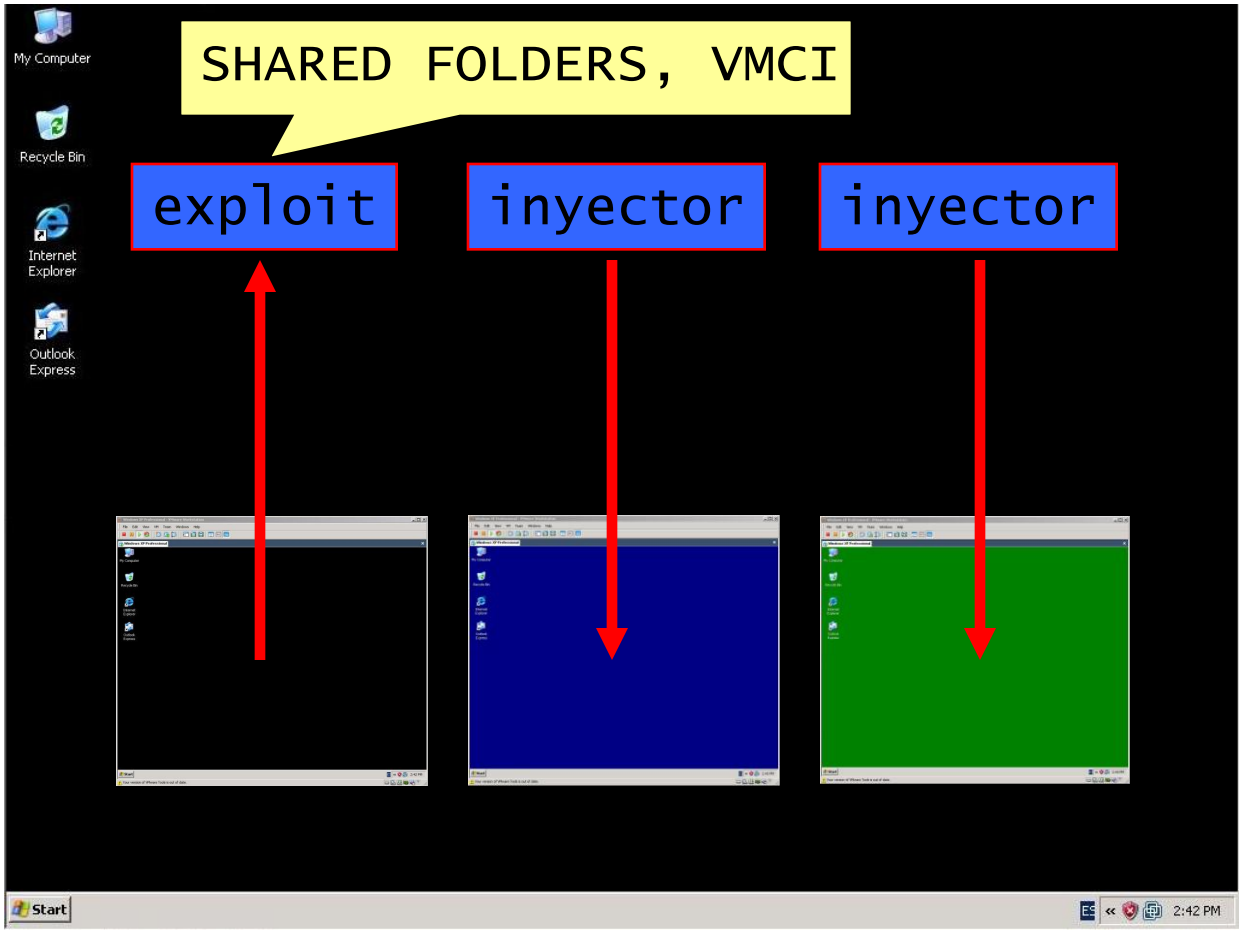
- **Host:** Maquina fisica que ejecuta a la maquina virtual.
- **Guest:** La maquina virtual.
- **vmware-vmx.exe:** Proceso en Windows que utiliza Vmware donde contiene a la maquina virtual.
- Este proceso lo usan Vmware Player, Workstation, Server, etc…

  Es un proceso como cualquier otro y corre con los permisos del usuario que lo haya ejecutado … ??? .

Posible Escenario



Posible Escenario sobre VMWare



Leyendo memoria

- **Prototipo en Windows:**

- The ReadProcessMemory function reads memory in a specified process. The entire area to be read must be accessible, or the operation fails.

- BOOL ReadProcessMemory(
- HANDLE *hProcess*, // handle to the process whose memory is read
- LPCVOID *lpBaseAddress*, // address to start reading
- LPVOID *lpBuffer*, // address of buffer to place read data
- DWORD *nSize*, // number of bytes to read
- LPDWORD *lpNumberOfBytesRead* // address of number of bytes read
-);

Leyendo memoria

- **Prototipo en Linux:**

  long ptrace (
  - **enum __ptrace_request** request,
  - **pid_t** *pid*,
  - **void** ***addr**,
  - **void** ***data**);

  Ej: value = ptrace (PTRACE_PEEKDATA , 1999 , 0x30fd0000 , NULL);

- **Prototipo en Mac OSX:**

  kern_return_t vm_read_overwrite **(**
  - **vm_task_t** *target_task*,
  - **vm_address_t** *address*,
  - **vm_size_t** *size*,
  - **pointer_t** *data_in*,
  - **target_task** *data_count***);**

  Ej: vm_read_overwrite (3456 , 0x30fd0000 , 4 , buffer , &bytes_leidos);

Idea

- Buscar un patrón de datos dentro del sistema operativo **GUEST** del proceso vmware.
- En este caso, “vmware-vmx.exe”, usando la función **ReadProcessMemory** reiteradamente.
- **TOOLS:** “ppattern.exe” y “pread.exe”.

Escribiendo memoria

- **Prototipo en Windows:**

- The WriteProcessMemory function writes memory in a specified process. The entire area to be written to must be accessible, or the operation fails.

- BOOL WriteProcessMemory(
  HANDLE *hProcess*, // handle to process whose memory is written to
  LPVOID *lpBaseAddress*, // address to start writing to
  LPVOID *lpBuffer*, // pointer to buffer to write data to
  DWORD *nSize*, // number of bytes to write
  LPDWORD *lpNumberOfBytesWritten* // actual number of bytes written
  );

Escribiendo memoria

- **Prototipo en Linux:**

  long ptrace (
  - **enum __ptrace_request** request,
  - **pid_t** *pid*,
  - **void ***addr*,
  - **void ***data*);

  Ej: ptrace (PTRACE_POKEDATA , 1999 , 0x30fd0000 , 0x12345678);

- **Prototipo en Mac OSX:**

  kern_return_t vm_write **(**
  - **vm_task_t** *target_task*,
  - **vm_address_t** *address*,
  - **pointer_t** *data*,
  - **mach_msg_type_number_t** *data_count***);**

  Ej: vm_read_overwrite (3456 , 0x30fd0000 , buffer , 4);

Idea

- Analogamente a la función para leer memoria, podría escribir cualquier zona del proceso vmware-vmx.exe usando la función **WriteProcessMemory**.

- **TOOLS:** “pread.exe” y “pwrite.exe”.

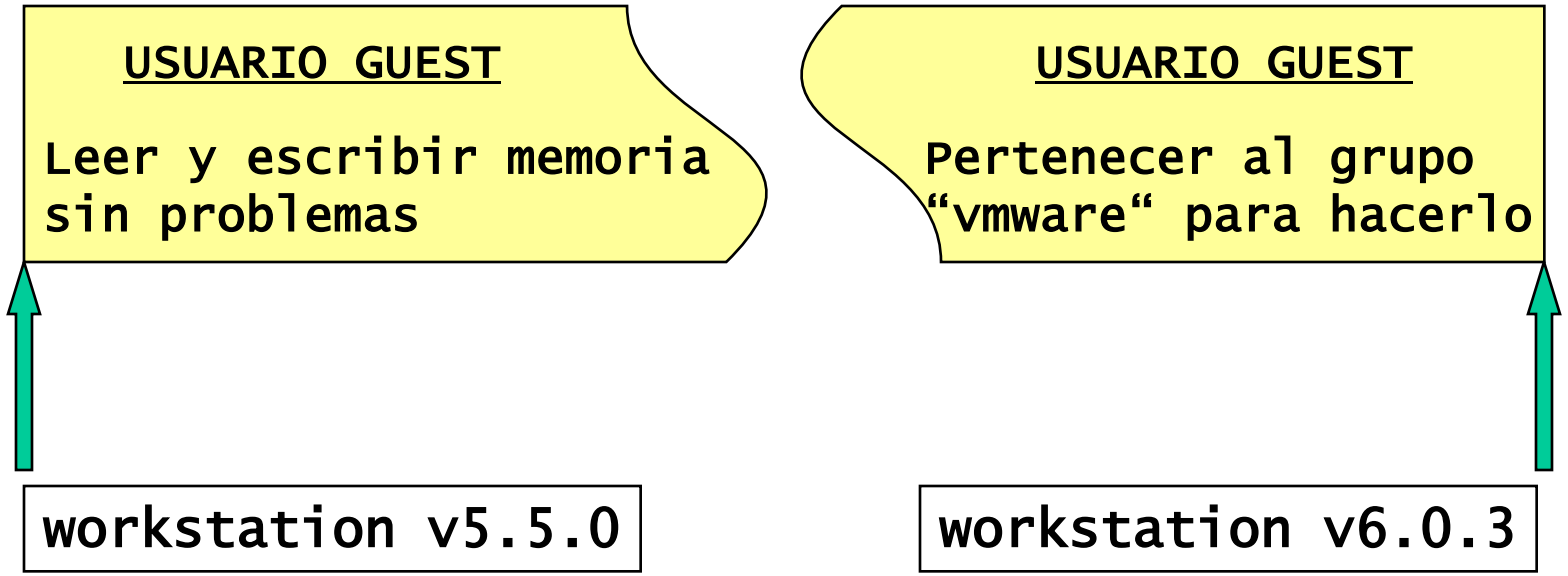
Detección de codigo

- ¿ Como puedo detectar un proceso que está corriendo dentro del operativo guest ?

- **Posible solución**: busco un patrón de instrucciones de una función del programa que quiero ubicar dentro de esta.
- Sí encuentro alguna dirección que contiene el patrón, **puede ser** que haya ubicado la parte del programa que estoy buscando.

- Ej: Buscar el pedazo de codigo que ejecuta “calc.exe” cuando apretamos una tecla usando la tool “**ppattern.exe**”.

Privilegios de vmware-vmx.exe



Ejecutar codigo con que permisos ?

- Usuario sin privilegios ?
- Usuario system ?
- Kernel ?

Ejecutar codigo con que permisos ?

- **Usuario sin privilegios:** Podría ser el proceso “explorer.exe”
- **Usuario system:** Podrían ser los servicios: “services.exe”, “lsass.exe”, “svchost.exe”, etc.
- **Kernel:** Algun driver del sistema operativo o que el VMware proveé al operativo guest: por ej: “hgfs.sys”, “vmci.sys”, etc.

Opción viable

- Ejecutar codigo a nivel de usuario “system”
- **Windows:** el proceso “services.exe”
- **Mac OSX**: el proceso “mDNSResponder”
- **Linux:** el kernel, el proceso “sshd”, etc

Problemas

- **Tener en cuenta para inyectar codigo:**
  - Hacer un buscador de patrones que corra en el host;
  - Paginación interna del proceso vmware;
  - Particionar el shellcode;
  - Concurrencia;
  - Restaurar la memoria del proceso;

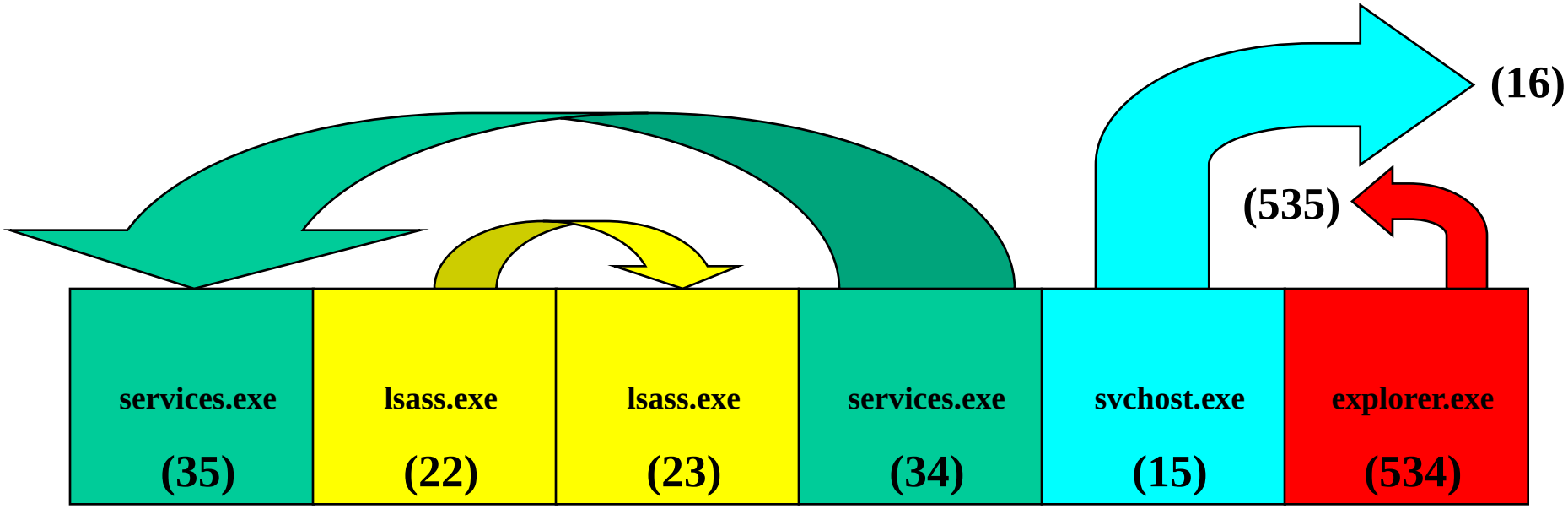
Buscador de Patrones

- Correr en el host.
- Codigo hecho en cualquier lenguaje.
- Utiliza las funciones descriptas anteriormente.
- Cuanto más inteligente sea, mejor.
  - Por ejemplo sí hace un analisis de la data que está leyendo, es más facil detectar zonas de codigo levemente modificadas entre versiones del mismo programa.
  - Escenario comun en Linux donde los programas se recompilan seguido.

Paginación Interna

- Memoria de los procesos no continua.
- Divididos en páginas de 4 kb = 0x1000 bytes.
- Puedo confíar en los 12 bits menos significativos de las direcciones.
- Por ej: “printf” está en 0x00401017 de “program.exe”, leyendo desde afuera podría llegar a encontrarla en la dirección 0x12345017

Paginación Interna



0x03210000	0x03211000	0x03212000	0x03213000	0x03214000	0x03215000
Mapa de Memoria de vmware-vmx.exe					

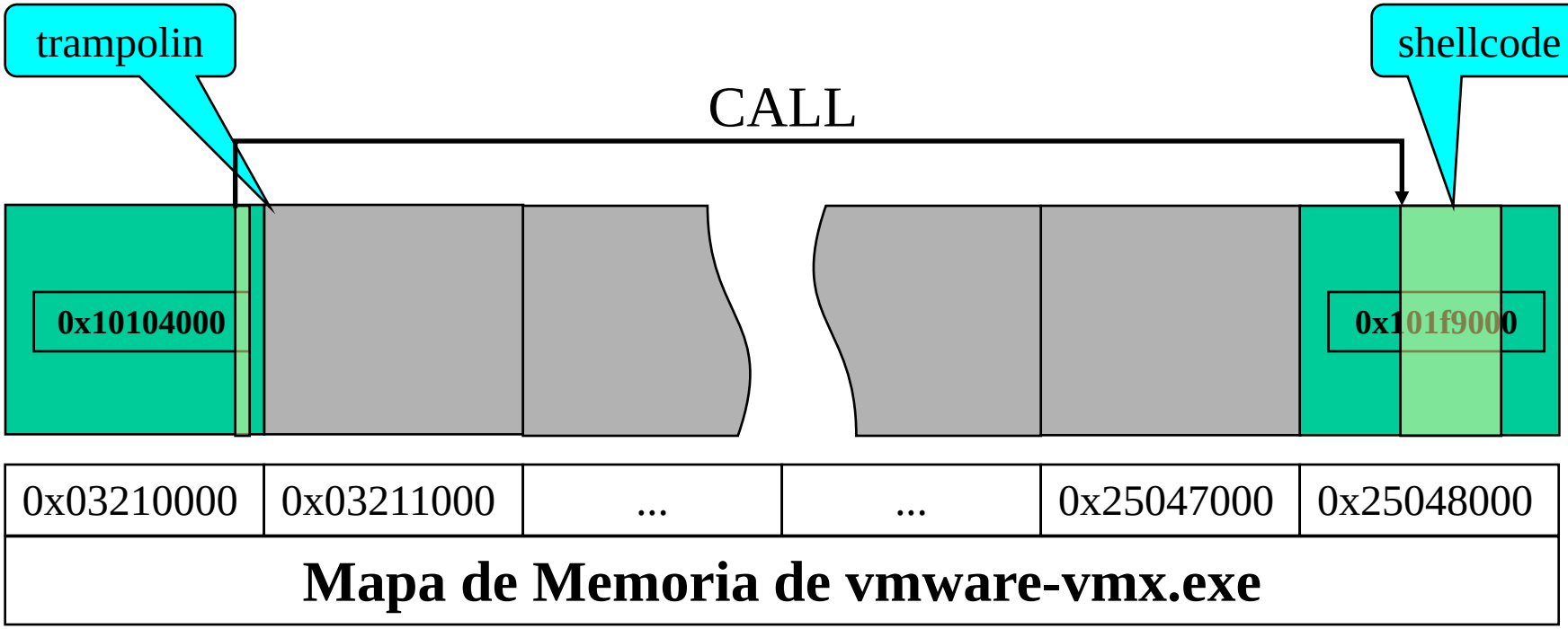
Particionar el shellcode

- Conociendo la existencia de no continuidad.

- Separo el shellcode en 2 partes (Hago 2 busquedas):
  - 1ra. parte va a una función TRAMPOLIN.
  - 2da. parte va casi al final del proceso a inyectar, a la zona de nombres de funciones exportadas, por ejemplo.

  - 1ra. parte escribo la instrucción "**call**" hacia la segunda parte.
  - 2da. parte escribo el **shellcode**.

Particionar el shellcode



Concurrencia

- **La idea**: Poner una barrera que evite la ejecución del codigo inyectado más de una vez, tal vez, por distintos threads.

- **Solución**: Usar una zona de memoria del proceso inyectado como semaforo.

Escribir y Restaurar Memoria

- **Escribo el proceso:**
  - 1ro. Reemplazo una zona de memoria no usada por el proceso con mi **shellcode**.
  - 2do. Reemplazo las primeras instrucciones una función usada como trampolin para que salte al **shellcode**.

- **Restauro el proceso**:
  - 1ro. Restauro la función trampolin para evitar que otro thread ejecute el salto a mi **shellcode**.
  - 2do. Restauro la zona de memoria no usada.

Donde funciona esto ?

- **Vmware:** Player, Workstation, Server, GSX, ESX, etc.
- **Virtual PC:** Creando un thread que inyecte el codigo del agente desde adentro.
- **Virtual Box:** Lo mismo que para Virtual PC.
- **Bosch:** No debería tener ningun tipo de protección.

Haciendo una Demo ...

- Inyectar un agente de Impact desde el host al guest.
- Operativo Host: Windows XP SP2.
- Operativo Guest: Windows 2000 SP4.
- Proceso a ser inyectado: “services.exe”
- Función trampolin: “ROpenSCManagerA”

PREGUNTAS