Bochs & Python

LaCon

Ero Carrera - ero.carrera@gmail.com

Opciones

- Sandboxes conllevan mucho trabajo
  - Emulación de APIs
  - Peculiaridades de la CPU
- Demasiado cat & mouse
- APIs no emuladas precisamente son triviales de explotar

Opciones

- Una emulación completa a bajo nivel permitiría correr una versión de Windows completa
  - No nos tenemos que preocupar de emular APIs
  - O detección de Vms
  - O trucos anti-debugging

Opciones

- Binary translation
  - **Valgrind**, **QEmu**
- Emulación parcial, (Para)Virtualization
  - **VMWare**, **Virtual PC**, **VirtualBox**, **Parallels Workstation**, **Virtual Iron**, **Xen**, **Denali**, etc
- Emulación completa
  - **Simics**, **Bochs**

Bochs

- Bochs [http://bochs.sourceforge.net/]
- Emulación completa
- Permite un control y acceso detallado del entorno emulado
- Tiene un buen interfaz de instrumentación
- Permite guardar y cargar el estado de la máquina

Que podémos hacer con el?

- Trazar
- Heurísticas
- Analisis del comportamiento de aplicaciones
- Y todo sin tener que preocuparnos demasiado de técnicas anti-VM o anti-debugging

Problemas potenciales

- Se ven demasiados datos
- Hay que filtrar para seguir a un sólo proceso
  - Page directory base/CR3, estructuras del OS
- Trucos Anti-Bochs ? Posibles pero es un entorno donde es posible hacer algo al respecto con facilidad
  - Peter Ferrie *Attacks on Virtual Machine Emulators* [http://pferrie.tripod.com/papers/attacks.pdf]

Limitaciones

- Su debugger es limitado
- No es escriptable
- No puede interactuar con otras herramientas
- Cambios en el interfaz de instrumentación requieren recompilar Bochs

El Debugger estandard

- Opción: *--enable-debugger --enable-disasm*
- Comandos:
  - **continue, step, quit, vbreak, lbreak, pbreak, info break, bpe, bpd, delete, x, xp, info {cpu, registers, sse, mmx, fpu, etc}, disassemble, trace {on,off}, instrument {start,stop,reset,print}, etc**

Instrumentación

- Opción: *--enable-instrumentation*
- Bochs compilará la extensión que elijamos
- El módulo puede implementar funciones que serán llamadas para eventos específicos
- Están todos definidos en (*`instrument/instrumentation.txt`*)

Callbacks

- void **bx_instr_init**(unsigned **cpu**);
- void **bx_instr_shutdown**(unsigned **cpu**);
- void **bx_instr_fetch_decode_completed**(
  unsigned **cpu**, bxInstruction_c ***i**);
- void **bx_instr_prefix**(unsigned **cpu**, Bit8u **prefix**);
- void **bx_instr_inp**(Bit16u **addr**, unsigned **len**);
- void **bx_instr_outp**(Bit16u **addr**, unsigned **len**);
- void **bx_instr_inp2**(Bit16u **addr**, unsigned **len**, unsigned **val**);
- void **bx_instr_outp2**(Bit16u **addr**, unsigned **len**, unsigned **val**);

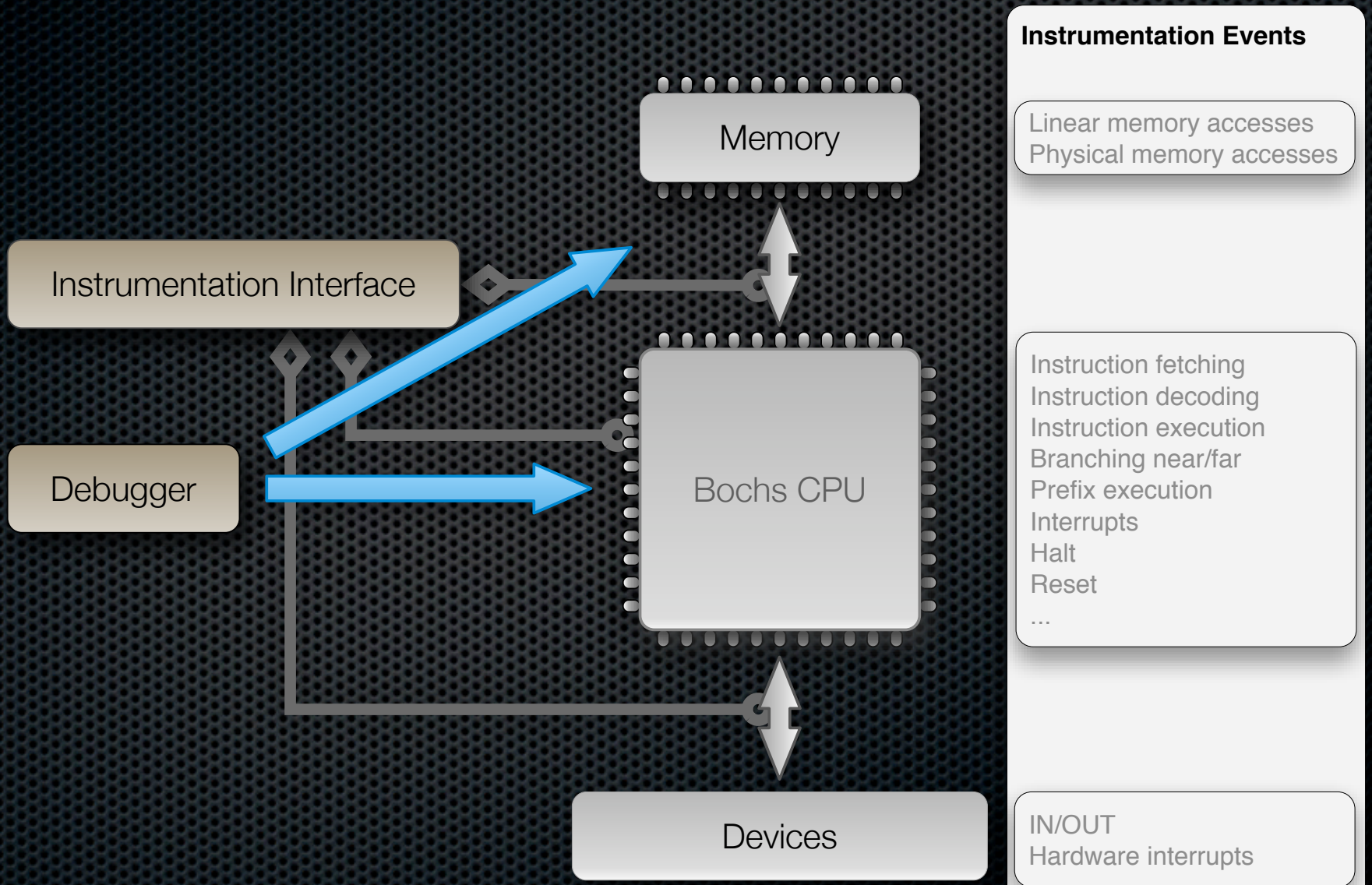
Accesos de Memoria

- void **bx_instr_lin_access**(
  unsigned **cpu**, bx_address **lin**,
  bx_address **phy**, unsigned **len**, unsigned **rw**);
- void **bx_instr_mem_data_access**(
  unsigned **cpu**, unsigned **seg**, bx_address **offset**, unsigned **len**, unsigned **rw**);
- void **bx_instr_phy_read**(
  unsigned **cpu**, bx_address **addr**, unsigned **len**);
- void **bx_instr_phy_write**(
  unsigned **cpu**, bx_address **addr**, unsigned **len**);

Flujo de ejecución

- void **bx_instr_new_instruction**(unsigned **cpu**);
- void **bx_instr_cnear_branch_taken**(
  unsigned **cpu**, bx_address **new_eip**);
- void **bx_instr_cnear_branch_not_taken**(unsigned **cpu**);
- void **bx_instr_ucnear_branch**(
  unsigned **cpu**, unsigned **what**, bx_address **new_eip**);
- void **bx_instr_far_branch**(
  unsigned **cpu**, unsigned **what**,
  Bit16u **new_cs**, bx_address **new_eip**);

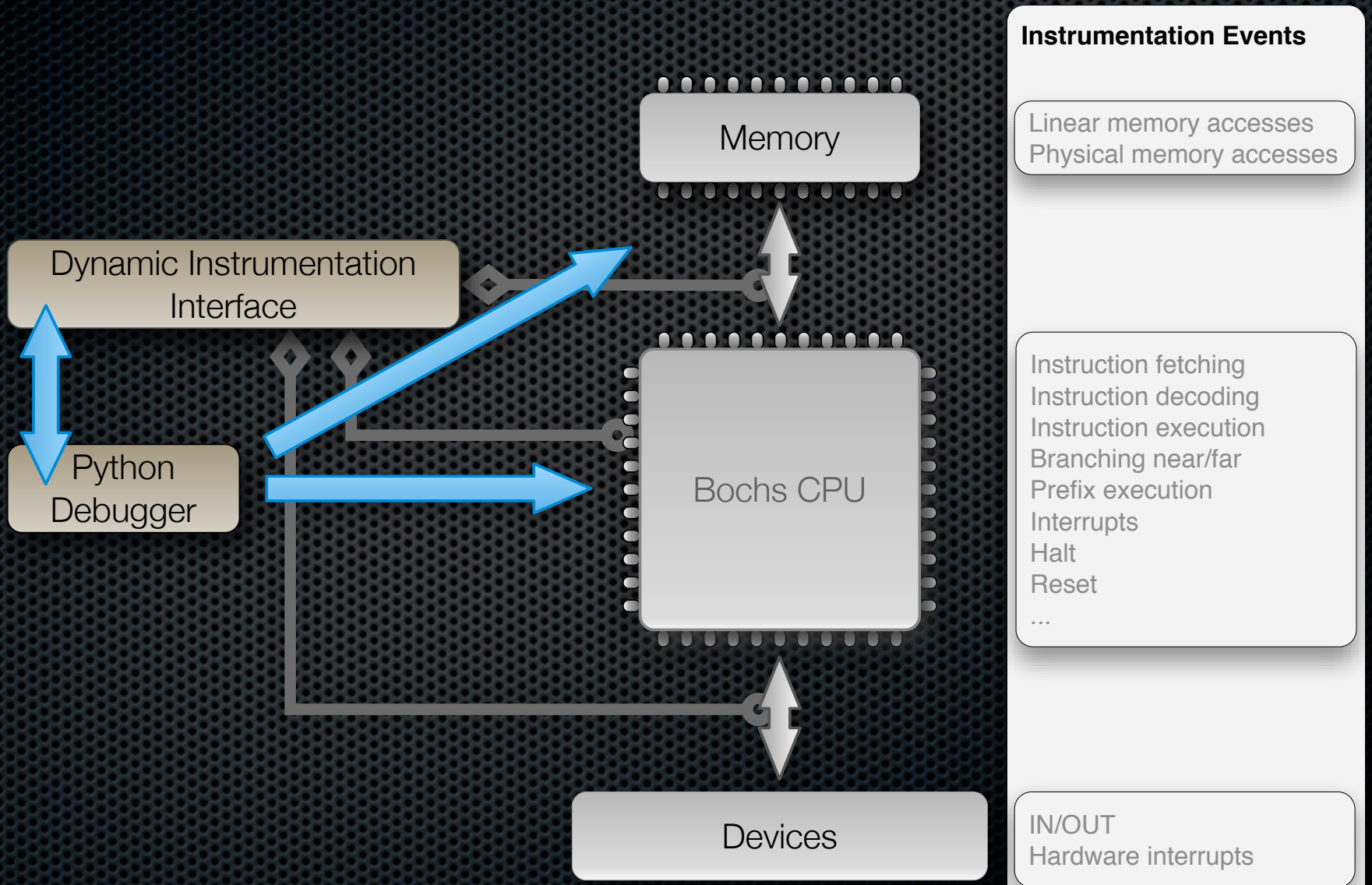
Arquitectura



Mejorando Bochs

- Comparte los mismo componentes
- Añade un intérprete de Python en vez del debugger
- Expone también el interfaz de instrumentación
- El interfaz de instrumentación puede ser controlado dinamicamente desde el intérprete Python

Arquitectura II



El debugger Python

- El módulo **bx** proveerá con el interfaz original
- El módulo **cpu** proveerá el acceso de lectura escritura de la CPU
- El módulo **dbg** encapsula la funcionalidad extra que permitirá configurar callbacks interfaz de instrumentación

Callbacks

- En el módulo **dbg**

- INSTR_INIT, INSTR_SHUTDOWN, INSTR_RESET, INSTR_HLT, INSTR_NEW_INSTRUCTION, INSTR_CNEAR_BRANCH_TAKEN, INSTR_CNEAR_BRANCH_NOT_TAKEN, INSTR_UCNEAR_BRANCH, INSTR_FAR_BRANCH, INSTR_INTERRUPT, INSTR_EXCEPTION, INSTR_HWINTERRUPT, INSTR_MEM_CODE, INSTR_MEM_DATA, INSTR_LIN_ACCESS, INSTR_PHY_READ, INSTR_PHY_WRITE, INSTR_WRMSR, INSTR_INP, INSTR_OUTP, INSTR_OPCODE, INSTR_FETCH_DECODE_COMPLETED, INSTR_PREFIX, INSTR_BEFORE_EXECUTION, INSTR_AFTER_EXECUTION, INSTR_REPEAT_ITERATION, INSTR_CACHE_CNTRL, INSTR_TLB_CNTRL, INSTR_PREFETCH_HINT

```
import dbg
dbg.set_callback(dbg.INSTR_INTERRUPT, process_int)
dbg.read_memory_block_linear(cpu.get(cpu.EIP), 1024)
```

Cambiando registros

```
import cpu

eax_value = cpu.get(cpu.EAX)

cpu.set(cpu.EAX, eax_value + 10)


cpu.set( cpu.get(cpu.EIP) + 0x3 )
```

Trazando ejecución

```
import bx
import dbg
import cpu
import time

def process_instruction(cpu_num):
   eip = cpu.get(cpu.EIP)
   print 'Executing at %08x' % eip
   time.sleep(.5)

dbg.set_callback(
   dbg.INSTR_NEW_INSTRUCTION, process_instruction)

bx.cont()
```

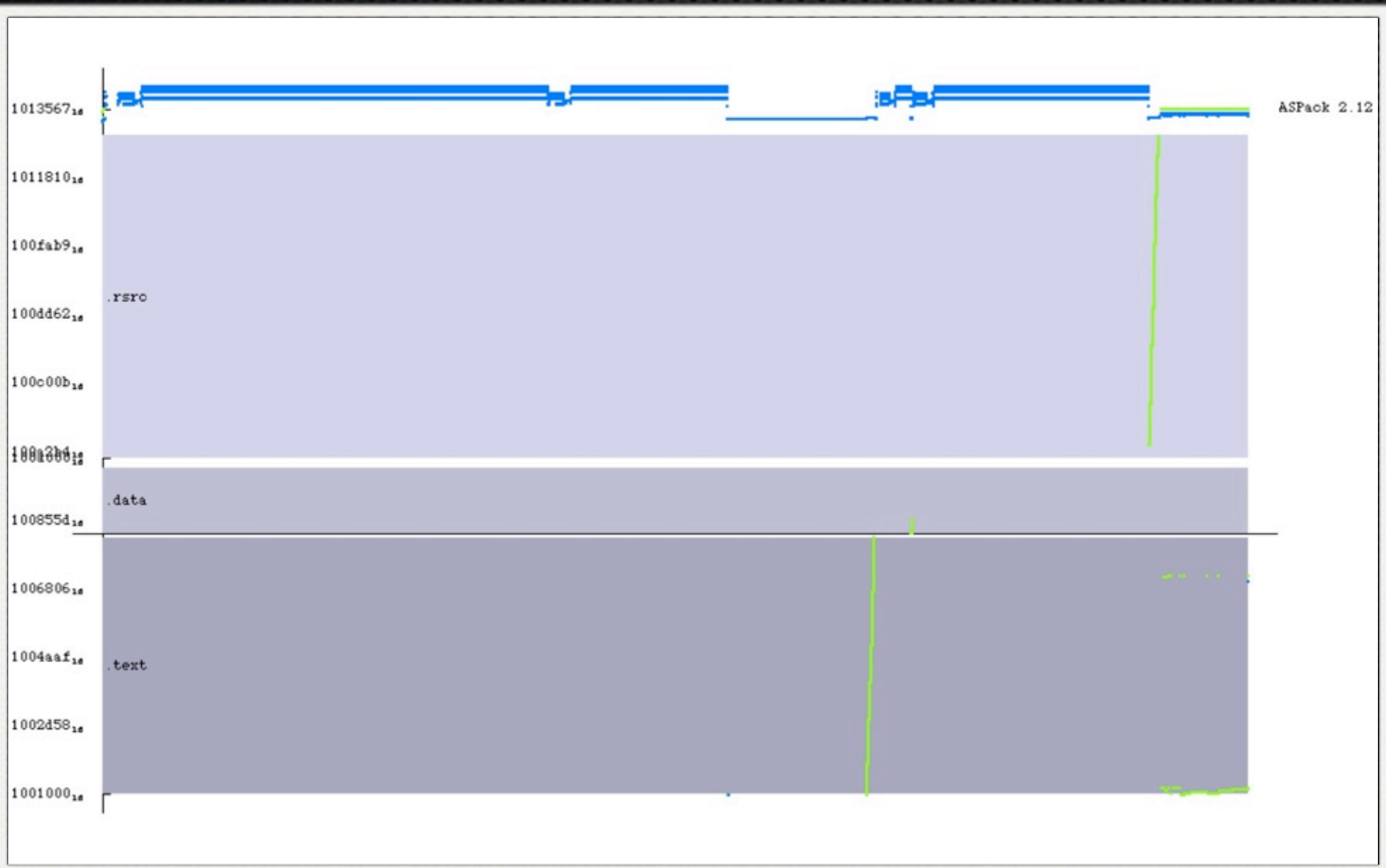
Accesos de memoria

```
import bx
import dbg
import cpu
import time

BX_READ, BX_WRITE, BX_RW = 0, 1, 2
access_type = dict( (
   (0, 'BX_READ'),
   (1, 'BX_WRITE'),
   (2, 'BX_RW') ) )

def process_mem_access(cpu_num, linear_addr, length, rw):
   eip = cpu.get(cpu.EIP)
   print 'Accessing[%s] address %08x at %08x' % (
      access_type[rw], linear_addr, eip)
   time.sleep(.5)

dbg.set_callback(dbg.INSTR_MEM_DATA, process_mem_access)

bx.cont()
```

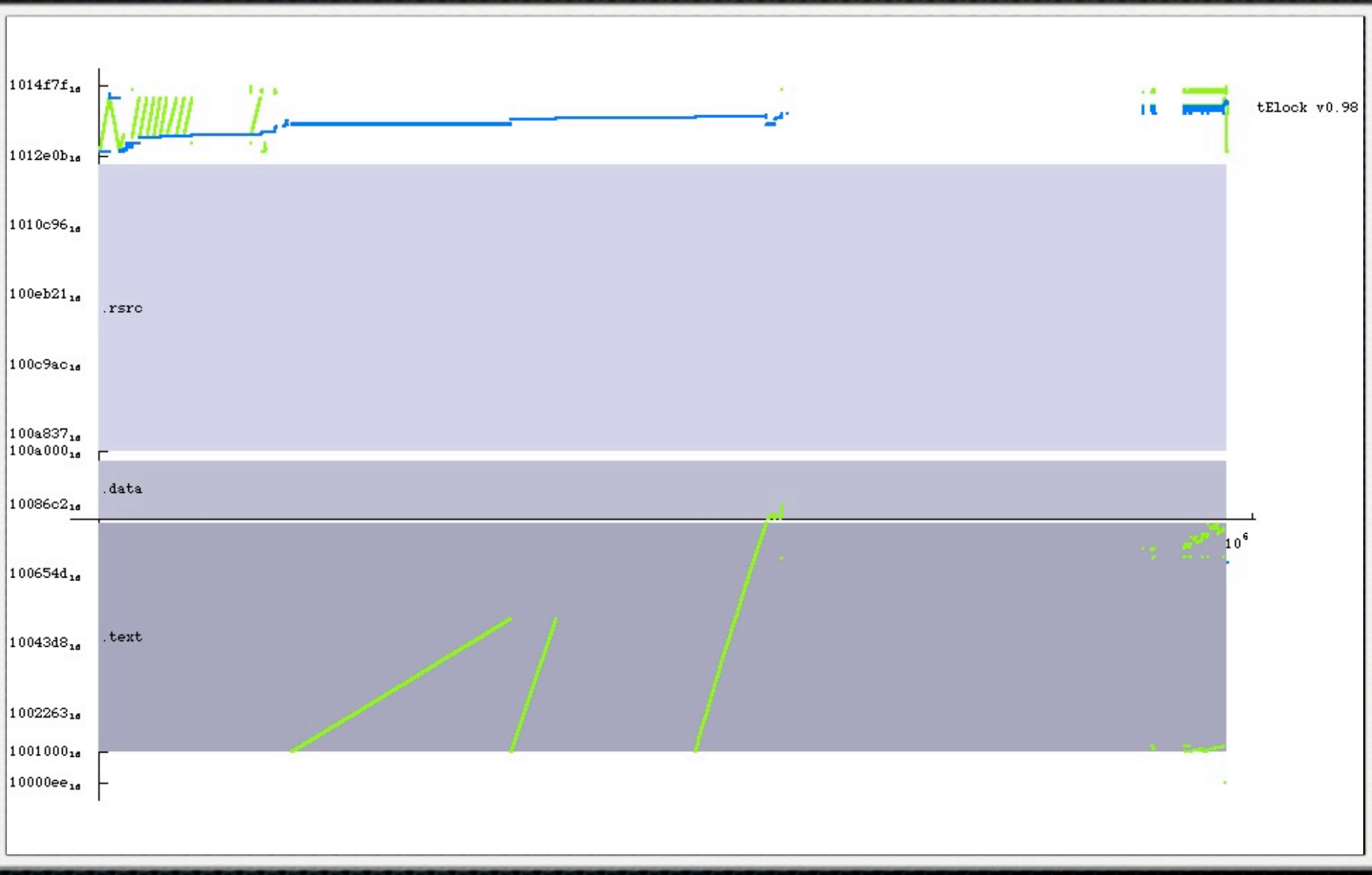
Comportamientos sospechosos

```
[EIP:0120d667] Detected: LOCK Instruction (suspicious): F0 86 18 ...
[EIP:0114bce0] Detected: RDTSC Instruction (suspicious)
[EIP:0121753a] Detected: TIB/TIB.ExceptionList access [7ffdf000]
[EIP:012175c4] Detected: VirtualPC test A: 0F 3F 07 0B 64 8F 05
[EIP:01217760] Detected: VMWare communications channel test
[EIP:0121777a] Detected: TIB/TIB.ExceptionList access [7ffdf000]
[EIP:009086cd] Detected: TIB.Self access [7ffdf018]
[EIP:0082b4c9] Detected: PEB.GlobalFlag access [7ffdc068]
[EIP:012443f7] Detected: LOCK Instruction (suspicious): F0 86 18 0A ...
[EIP:00912210] Detected: PEB.BeingDebugged access [7ffdc002]
[EIP:00411bee] Detected: SLDT: 0F 00 45 F8 0F B6 45 F9 50 0F ...
```

ASPack 2.12


1013567₁₆
1011810₁₆
100fab9₁₆
100dd62₁₆
100c00b₁₆
100855d₁₆
1006806₁₆
1004aaf₁₆
1002d58₁₆
1001000₁₆
ASPack 2.12
.rsrc
.data
.text

tElock


1014f7f16
1012e0b16
1010c9616
100eb2116
100c9ac16
100a83716
100a00016
10086c216
100654d16
10043d816
1002263 16
1001000 16
10000ee16
.rsrc
.data
.text
tElock v0.98
10^6

Yoda's Protector v1.02