

Python Debugging Outside the “Bochs”

HITBSecConf2008 - Dubai

Ero Carrera - ero.carrera@gmail.com

Reverse Engineer at zynamics GmbH

&

Independent Information Security Researcher

Talk outline

- ✦ Introduction
- ✦ The Tool. System architecture
- ✦ The Extensions
- ✦ Demos
- ✦ Obtaining the Tool

The Context: Anti-Virus Industry

- ✦ Increasing quantities of malware
- ✦ Need to automate analysis, specifically unpacking
- ✦ Packer specific unpacking algorithms are a lot of work, dedicated person/team
- ✦ Automation and generic unpacking can be tackled with sandboxes/emulation

Choosing and Approach

- ✦ Sandboxes are a lot of work
 - ✦ API emulation
 - ✦ CPU attributes and particularities
- ✦ Endless cat & mouse game
- ✦ APIs not emulated exactly are trivial to exploit

Choosing and Approach

- ✦ Why to attempt to re-implement all of Window's APIs?
- ✦ Lower level, full-system, emulation could allow to run a full Window O.S.
 - ✦ No need to worry about getting APIs right
 - ✦ Or VM detection*
 - ✦ Or anti-debug tricks

Choosing and Approach

- A CPU emulator or binary translation provides a scenario where we only need to focus on low level details
 - Advantages: full control
 - Drawbacks: speed

Available Options

- ✧ Binary translation
 - ✧ **Valgrind, QEmu**
- ✧ Partial Emulation, (Para)Virtualization
 - ✧ **VMWare, Virtual PC, VirtualBox, Parallels Workstation, Virtual Iron, Xen, Denali, etc**
- ✧ Full Emulation
 - ✧ **Simics, Bochs**

The Tool

- ✦ Chose Bochs [<http://bochs.sourceforge.net/>]
- ✦ It's a full emulator
- ✦ Allows for detailed and fine grained access and control of the emulated environment
- ✦ Has a powerful instrumentation interface
- ✦ Supports state save/restore
- ✦ Bochs allows to play in "*God mode*"

What could we do with it?

- ✦ Tracing
- ✦ Advanced heuristics
- ✦ Behavioral analysis
- ✦ See things that can't be seen with other tools/
environments
- ✦ All while not having to care too much about *Anti-Virtual Machine* and *Anti-Debugging* techniques

Potential Issues

- ✦ One sees a lot of (too much?) data, which is both good and bad
- ✦ Has to filter individual processes
 - ✦ Page directory base/CR3, tracking OS structures
- ✦ Anti-Bochs tricks? Possible, like everywhere, but we have a lot of control, unlike other environments
 - ✦ See Peter Ferrie's *Attacks on Virtual Machine Emulators* [<http://pferrie.tripod.com/papers/attacks.pdf>]

Shortcomings

- ✦ The debugger only implements limited features
- ✦ It's not scriptable
- ✦ Can't interact with other tools
- ✦ The instrumentation interface requires to recompile Bochs for any modifications

The Standard Debugger

- ✦ Option: *--enable-debugger --enable-disasm*
- ✦ Bochs will provide with an interactive debugger
- ✦ Commands:
 - ✦ **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**

Instrumentation

- ✦ Option: *--enable-instrumentation*
- ✦ Bochs will compile in a module of our choosing
- ✦ The module can implement functions that will be called on specific events
- ✦ some of those are... (all fully documented in *instrument/instrumentation.txt*)

Instrumentation 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**);

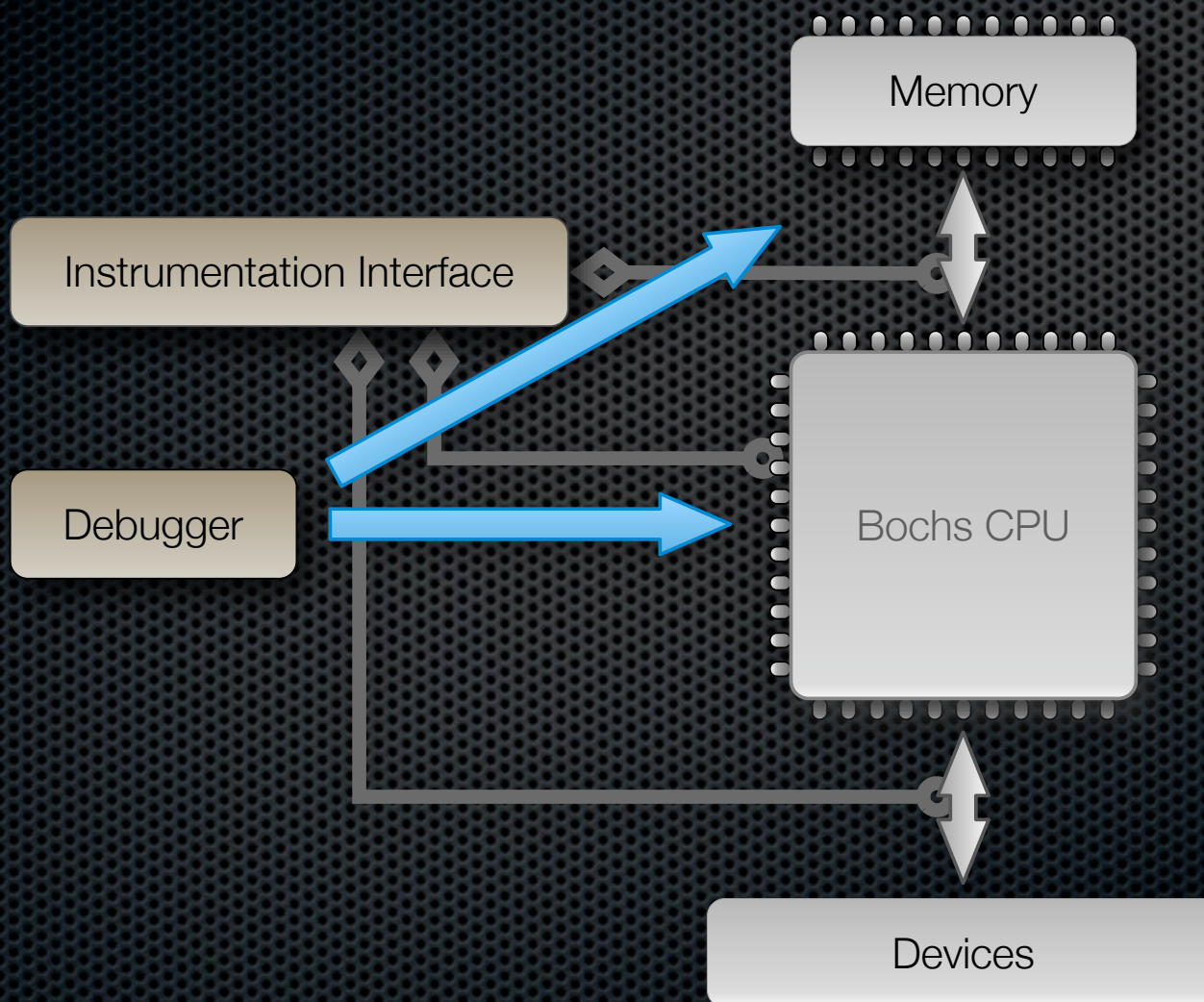
Memory Accesses

- void **bx_instr_lin_access**(
 unsigned **cpu**, bx_address **lin**,
 bx_address **phy**, unsigned **len**, unsigned **rw**);
- void **bx_instr_mem_data**(
 unsigned **cpu**, bx_address **linear**, 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**);

Flow related

- ✧ 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**);

Architecture Overview



Instrumentation Events

Linear memory accesses
Physical memory accesses

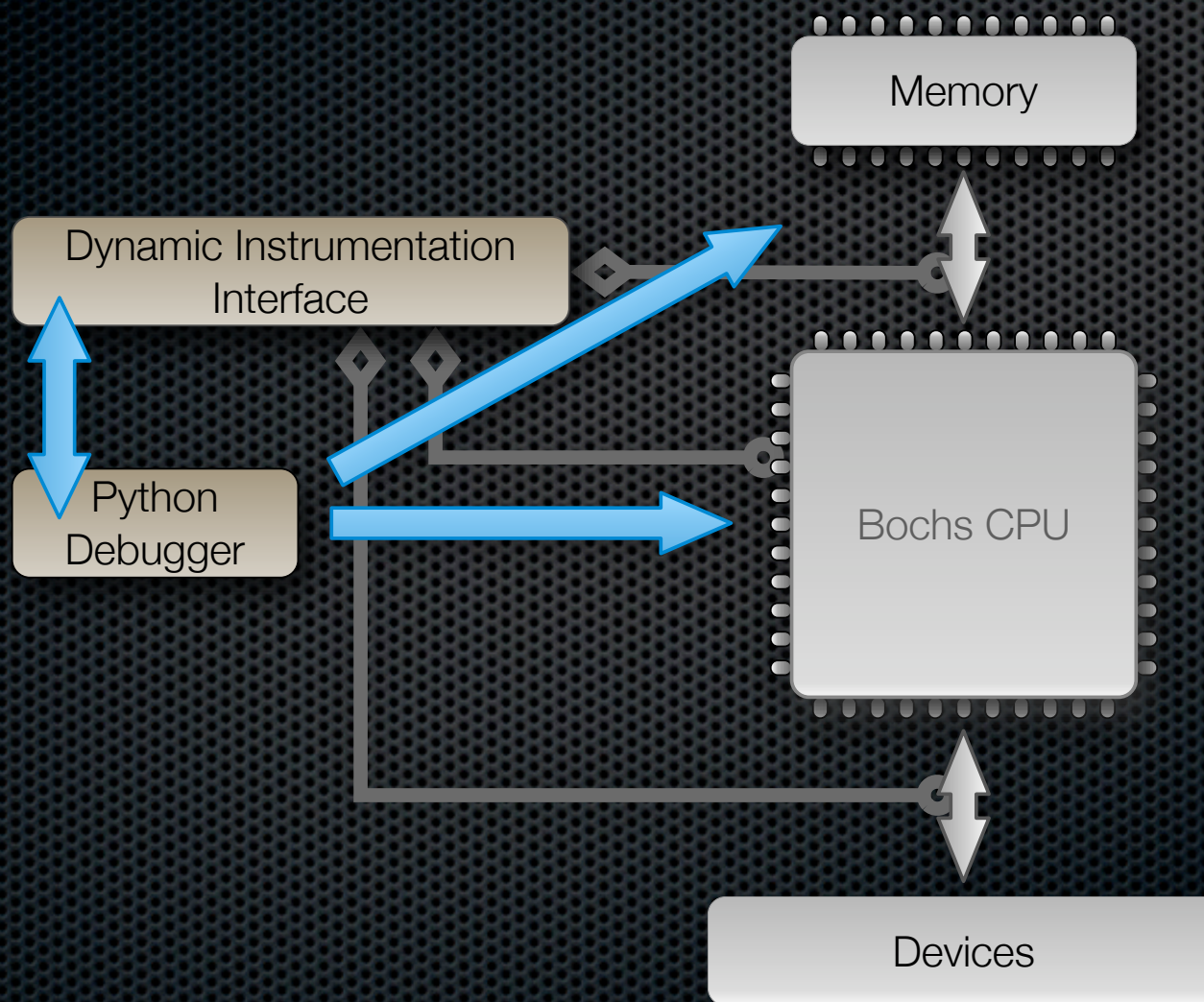
Instruction fetching
Instruction decoding
Instruction execution
Branching near/far
Prefix execution
Interrupts
Halt
Reset
...

IN/OUT
Hardware interrupts

Improving Bochs

- ✦ Shares the same components
- ✦ Added a Python interpreter command line instead of the debugger
- ✦ Exposed the instrumentation
- ✦ The instrumentation interface can now be controlled dynamically from the new Python debugger

Architecture Overview II



Instrumentation Events

Linear memory accesses
Physical memory accesses

Instruction fetching
Instruction decoding
Instruction execution
Branching near/far
Prefix execution
Interrupts
Halt
Reset
...

IN/OUT
Hardware interrupts

The Enhanced Debugger

- ✦ Bochs' debugger would be great if it supported scripting
- ✦ Having great scripting languages, there's no need to invent a new one
- ✦ *Why Python?*
 - ✦ It's easy to embed and extend applications with
 - ✦ Provides with an endless stream of tools and modules

The Enhanced Instrumentation Interface

- ✦ Once having a *Python* interpreter it only made sense to expose the instrumentation interface
- ✦ Dynamic callback allow for on-demand usage of the instrumentation interface
- ✦ Having full speed when needed...
- ✦ and full control when desired
- ✦ It exposes all the fine granularity provided by the standard instrumentation interface

Enabling it All

Once we:

- ✦ have the latest version of Bochs
- ✦ applied the corresponding patch
- ✦ configured Bochs with the option:
--enable-debugger --enable-instrumentation=python_hooks
- ✦ and compiled, Bochs should be ready

Inside the Python-enabled Debugger

- ✦ The module **bx** will provide with most of the original debugger's functionality
- ✦ The **cpu** module will provide with means of reading and writing to the CPU
- ✦ The **dbg** module encapsulates extra functionality allowing to set callbacks to the instrumentation interface

Callbacks

- Provided by the **dbg** module
- 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)
```


Setting Registers

```
import cpu

eax_value = cpu.get(cpu.EAX)

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

# Jumping over code. If the instruction to jump over is 3
bytes long

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


Tracing Instruction Execution

```
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()
```


Tracing Memory Accesses

```
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)

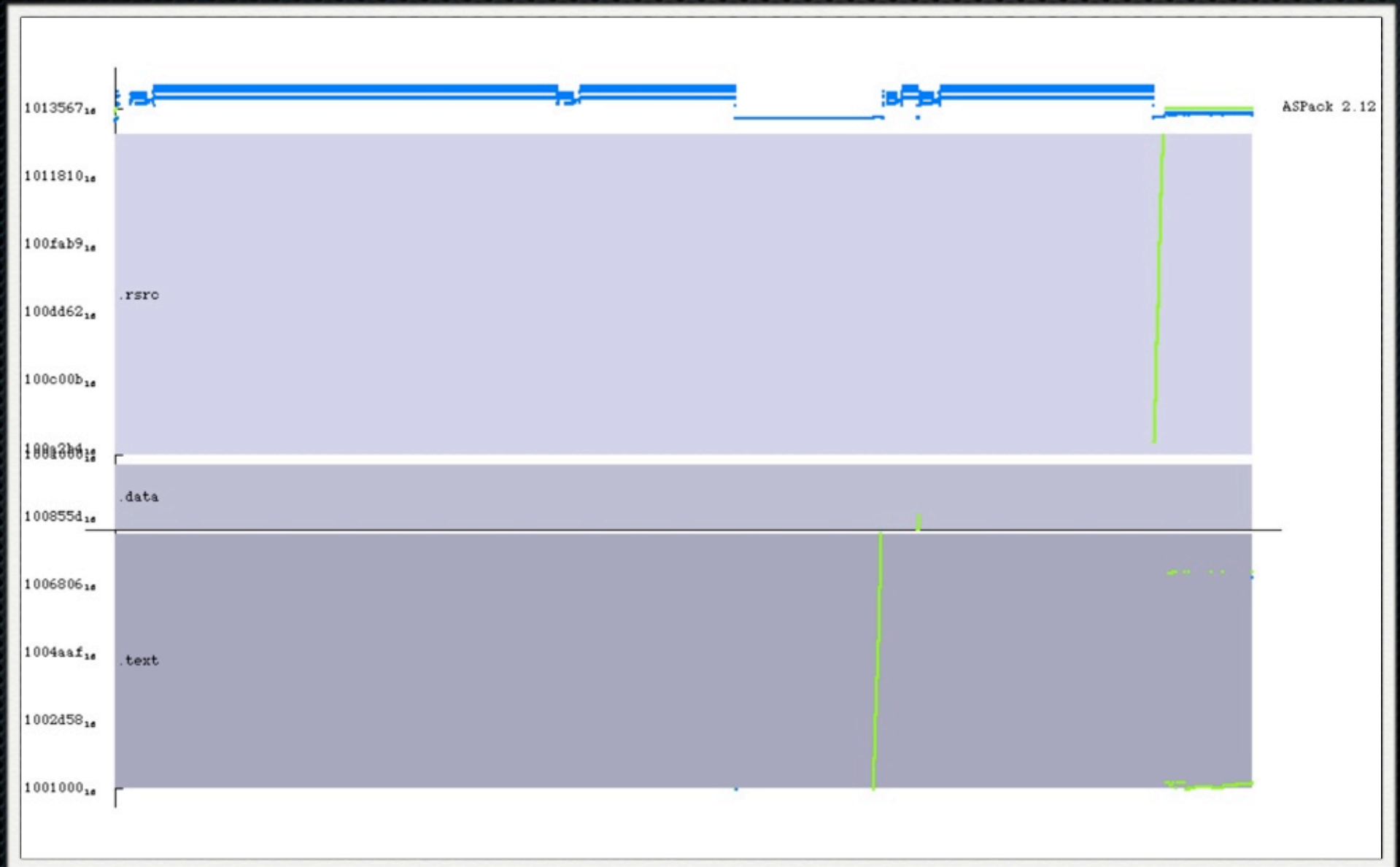
dbg.set_callback(dbg.INSTR_MEM_DATA, process_mem_access)

bx.cont()
```

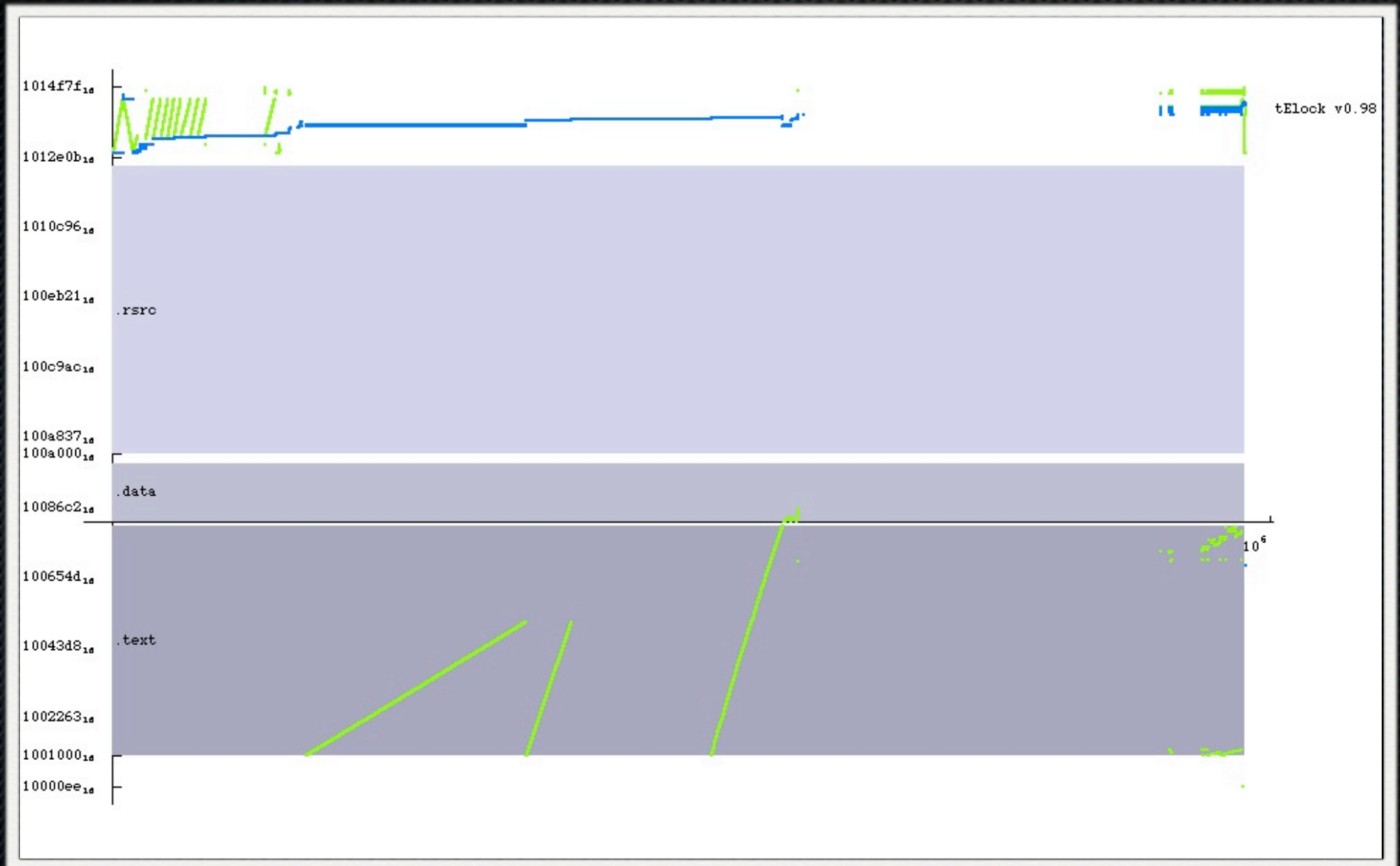

Detecting Suspicious Behavior

```
[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 ...
```

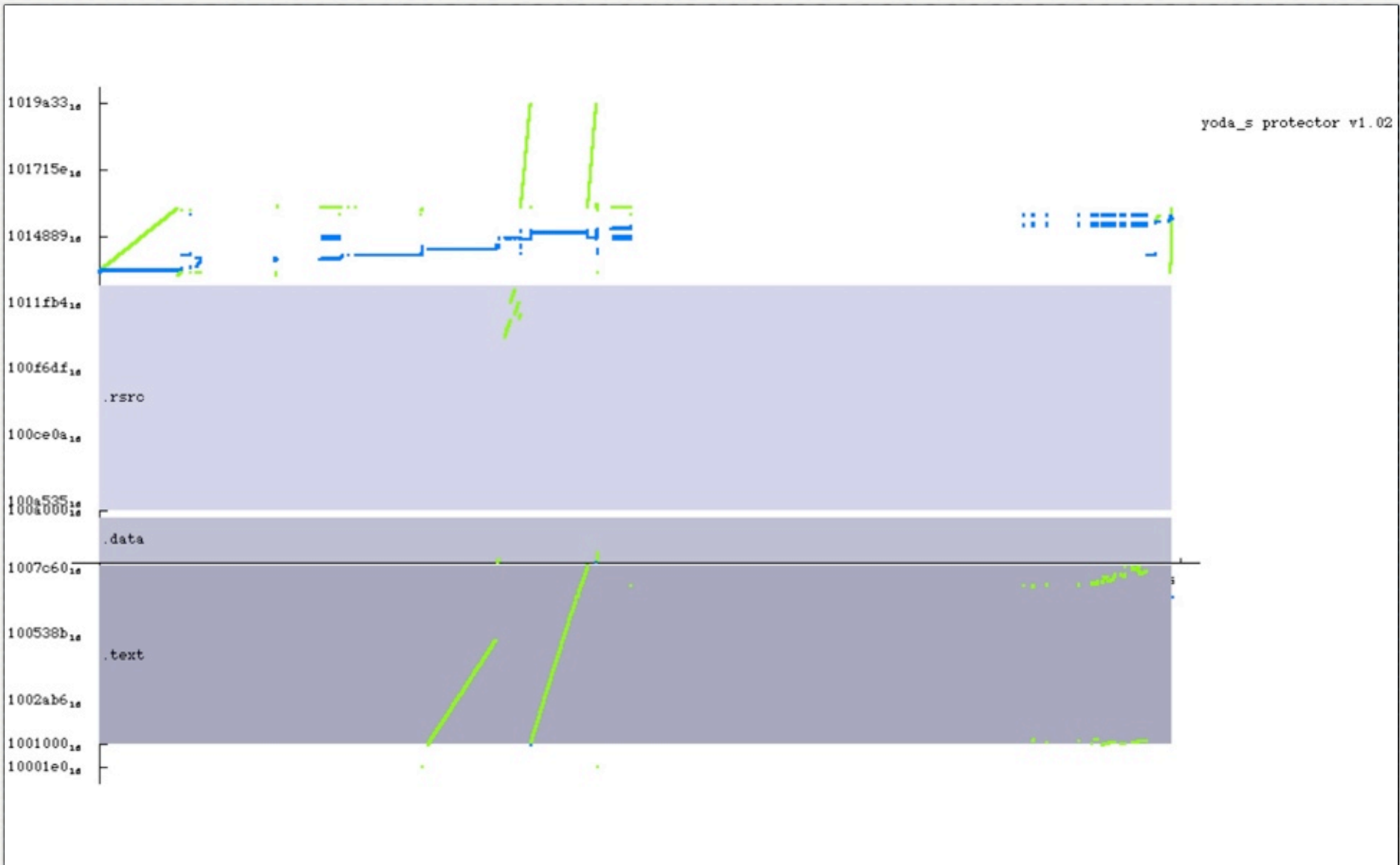

Packer Execution Traces (ASPack 2.12)



Packer Execution Traces (tElock)



Packer Execution Traces (Yoda's Protector v1.02)



Demos

- ✦ Scanning memory for strings
- ✦ Tracing API Calls
- ✦ Tracing Sys-Calls
- ✦ Getting the TIB and PEB

Getting your hands on it

- ✦ Like Bochs, It's free!
 - ✦ <http://bochs.sourceforge.net/>
- ✦ Has been contributed to the Bochs project
- ✦ So far it's available as a patch (*Python extensions for debugger and instrumentation interface*):
 - ✦ http://sourceforge.net/tracker/index.php?func=detail&aid=1937046&group_id=12580&atid=312580

Q&A