

Gray Hat Python

Python Programming for Hackers and Reverse Engineers

by Justin Seitz

Errata updated to print 14

| Page | Error | Correction | Print corrected |
|------|--|---|-----------------|
| 31 | <pre>def open_process(self,pid): h_process = kernel32.OpenProcess(PROCESS_ALL_ACCESS,pid,False) return h_process def attach(self,pid): self.h_process = self.open_process(pid) # We attempt to attach to the process # if this fails we exit the call if kernel32.DebugActiveProcess(pid): self.debugger_active = True self.pid = int(pid) self.run() else: print "[*] Unable to attach to the process."</pre> | <pre>def open_process(self,pid): h_process = kernel32.OpenProcess(PROCESS_ALL_ACCESS,False,pid) return h_process def attach(self,pid): self.h_process = self.open_process(pid) # We attempt to attach to the process # if this fails we exit the call if kernel32.DebugActiveProcess(pid): self.debugger_active = True self.pid = int(pid) else: print "[*] Unable to attach to the process."</pre> | Print 8 |

| Page | Error | Correction | Print corrected |
|------|---|---|-----------------|
| 37 | <pre>def enumerate_threads(self): --snip-- if snapshot is not None: --snip-- while success: if thread_entry.th32OwnerProcessID == self.pid: thread_list.append(thread_entry.th32ThreadID) success = kernel32.Thread32Next(snapshot, byref(thread_entry)) kernel32.CloseHandle(snapshot) return thread_list else: return False def get_thread_context (self, thread_id): context = CONTEXT() context.ContextFlags = CONTEXT_FULL CONTEXT_DEBUG_REGISTERS</pre> | <pre>def enumerate_threads(self): --snip-- if snapshot is not None: --snip-- while success: if thread_entry.th32OwnerProcessID == self.pid: thread_list.append(thread_entry.th32ThreadID) success = kernel32.Thread32Next(snapshot, byref(thread_entry)) kernel32.CloseHandle(snapshot) return thread_list else: return False def get_thread_context (self, thread_id=None,h_thread=None): context = CONTEXT() context.ContextFlags = CONTEXT_FULL CONTEXT_DEBUG_REGISTERS if not h_thread: self.open_thread(thread_id)</pre> | Print 8 |

| Page | Error | Correction | Print corrected |
|------|---|--|-----------------|
| 42 | <pre> def get_debug_event(self): debug_event = DEBUG_EVENT() continue_status= DBG_CONTINUE if kernel32.WaitForDebugEvent(byref(debug_event),INFINITE): # Let's obtain the thread and context information self.h_thread = self.open_thread(debug_event.dwThreadId) self.context = self.get_thread_context(self.h_thread) print "Event Code: %d Thread ID: %d" % (debug_event.dwDebugEventCode, debug_event.dwThreadId) # If the event code is an exception, we want to # examine it further. if debug_event.dwDebugEventCode == EXCEPTION_DEBUG_EVENT: # Obtain the exception code exception = debug_event.u.Exception.ExceptionRecord.ExceptionCode self.exception_address = debug_event.u.Exception.ExceptionRecord .ExceptionAddress if exception == EXCEPTION_ACCESS_VIOLATION: print "Access Violation Detected." # If a breakpoint is detected, we call an internal # handler. elif exception == EXCEPTION_BREAKPOINT: continue_status = self.exception_handler_breakpoint() elif ec == EXCEPTION_GUARD_PAGE: print "Guard Page Access Detected." elif ec == EXCEPTION_SINGLE_STEP: print "Single Stepping." kernel32.ContinueDebugEvent(debug_event.dwProcessId, debug_event.dwThreadId, continue_status) ... def exception_handler_breakpoint(): </pre> | <pre> def get_debug_event(self): debug_event = DEBUG_EVENT() continue_status= DBG_CONTINUE if kernel32.WaitForDebugEvent(byref(debug_event),INFINITE): # Let's obtain the thread and context information self.h_thread = self.open_thread(debug_event.dwThreadId) self.context = self.get_thread_context(h_thread=self.h_thread) print "Event Code: %d Thread ID: %d" % (debug_event.dwDebugEventCode, debug_event.dwThreadId) # If the event code is an exception, we want to # examine it further. if debug_event.dwDebugEventCode == EXCEPTION_DEBUG_EVENT: # Obtain the exception code exception = debug_event.u.Exception.ExceptionRecord.ExceptionCode self.exception_address = debug_event.u.Exception.ExceptionRecord .ExceptionAddress if exception == EXCEPTION_ACCESS_VIOLATION: print "Access Violation Detected." # If a breakpoint is detected, we call an internal # handler. elif exception == EXCEPTION_BREAKPOINT: continue_status = self.exception_handler_breakpoint() elif exception == EXCEPTION_GUARD_PAGE: print "Guard Page Access Detected." elif exception == EXCEPTION_SINGLE_STEP: print "Single Stepping." kernel32.ContinueDebugEvent(debug_event.dwProcessId, debug_event.dwThreadId, continue_status) ... def exception_handler_breakpoint(self): </pre> | Print 8 |

| Page | Error | Correction | Print corrected |
|------|--|--|-----------------|
| 44 | <pre>self.breakpoints[address] = address,original_byte)</pre> | <pre>self.breakpoints[address] = original_byte)</pre> | Print 8 |
| 48 | <pre>def bp_set_hw(self, address, length, condition): --snip-- # We want to set the debug register in every thread for thread_id in self.enumerate_threads(): context = self.get_thread_context(thread_id=thread_id) # Enable the appropriate flag in the DR7 # register to set the breakpoint context.Dr7 = 1 << (available * 2) # Save the address of the breakpoint in the # free register that we found if available == 0: context.Dr0 = address elif available == 1: context.Dr1 = address elif available == 2: context.Dr2 = address elif available == 3: context.Dr3 = address # Set the breakpoint condition context.Dr7 = condition << ((available * 4) + 16) # Set the length context.Dr7 = length << ((available * 4) + 18) # Set thread context with the break set h_thread = self.open_thread(thread_id) kernel32.SetThreadContext(h_thread,byref(context))</pre> | <pre>def bp_set_hw(self, address, length, condition): --snip-- # We want to set the debug register in every thread for thread_id in self.enumerate_threads(): context = self.get_thread_context(thread_id=thread_id) # Enable the appropriate flag in the DR7 # register to set the breakpoint context.Dr7 = 1 << (available * 2) # Save the address of the breakpoint in the # free register that we found if available == 0: context.Dr0 = address elif available == 1: context.Dr1 = address elif available == 2: context.Dr2 = address elif available == 3: context.Dr3 = address # Set the breakpoint condition context.Dr7 = condition << ((available * 4) + 16) # Set the length context.Dr7 = length << ((available * 4) + 18) # Set thread context with the break set h_thread = self.open_thread(thread_id) kernel32.SetThreadContext(h_thread,byref(context))</pre> | Print 8 |
| 49 | <pre>elif self.context.Dr6 & 0x8 and self.hardware_breakpoints.has_key(3): slot = 3 else:</pre> | <pre>elif self.context.Dr6 & 0x8 and self.hardware_breakpoints.has_key(3): slot = 3 else:</pre> | Print 14 |