Pycodin: Instrumentando código sin dolor

Adrián Manrique (adrian@coresecurity.com)

Andrés López Luksenberg
(aluksenberg@coresecurity.com)

Agenda

- Introducción
- Pycodin
- Traceando un shellcode
- Conclusiones

Qué significa instrumentar código?

- **Según Wikipedia : es la habilidad de monitorear o medir el nivel de performance de un producto (software), para diagnosticar errores y escribir información de traceo.**

Por qué instrumentar?

- Debugging
- Unit Testing
- Reverse Engineering
- Fuzzing

Qué es Pycodin?

Es una **librería** implementada en **python y C** para **instrumentación de código** máquina. Soporta la ejecución de código para varias plataformas.

Qué es un shellcode?

Es código máquina y por lo general se lo utiliza para explotar vulnerabilidades en un proceso

```
#---------------------------------------------------- exception handler
# handler:
code += "\x55"                                  # push ebp
code += "\x8B\xEC"                              # mov ebp, esp
code += "\x53"                                  # push ebx
code += "\x56"                                  # push esi
code += "\xBE\x00\x00\x00\x70"                  # mov esi, 0x70000000
code += "\xBB\x00\x00\x01\x00"                  # mov ebx, 0x10000
code += "\x33\xD2"                              # xor edx, edx
code += "\x64\x8B\x12"                          # mov edx, fs:[edx]
code += "\x8B\x12"                              # mov edx, [edx]
code += "\x83\xEA\x04"                          # sub edx, 4
code += "\x8B\x52\x0C"                          # mov edx, [edx+0xc]
code += "\x83\xC2\x04"                          # add edx, 4
code += "\x8B\x02"                              # mov eax, [edx]            ; get counter
code += "\xFF\x02"                              # inc dword ptr [edx]
code += "\x0F\xAF\xC3"                          # imul eax, ebx
code += "\x03\xF0"                              # add esi, eax
code += "\x33\xD2"                              # xor edx,edx
code += "\x64\x8B\x12"                          # mov edx, fs:[edx]
code += "\x8B\x12"                              # mov edx, [edx]
code += "\x83\xEA\x04"                          # sub edx, 4
code += "\x8B\x52\x0C"                          # mov edx, [edx+0xc]
code += "\x89\x32"                              # mov [edx], esi            ; store k32addr
code += "\x8B\xC6"                              # mov eax, esi
code += "\x8B\x55\x08"                          # mov edx, [ebp+8]
code += "\x83\xEA\x04"                          # sub edx, 4
code += "\x8B\x12"                              # mov edx, [edx]
code += "\x81\xC2\xA0\x00\x00\x00"              # add edx, 0xa0
code += "\x89\x02"                              # mov [edx], eax            ; set esi in exception context
code += "\x5E"                                  # pop esi
code += "\x33\xC0"                              # xor eax, eax
code += "\x5B"                                  # pop ebx
code += "\x5D"                                  # pop ebp
code += "\xC3"                                  # ret
```

Virtualización

- **Es la abstracción de los recursos de una computadora (memoria , cpu, devices, etc).**

- **Es un software que corre sobre una computadora (*host*) que permite simular computadoras de diferentes arquitecturas (*guests*).**

QEMU

SWSOFT PARALLELS®

vmware®

- Emulador de procesadores

- Basado en la traducción dinámica de binarios.

- Puede ejecutarse en cualquier tipo de Microprocesador o arquitectura (x86, x86-64,PowerPC, MIPS, SPARC, etc.).

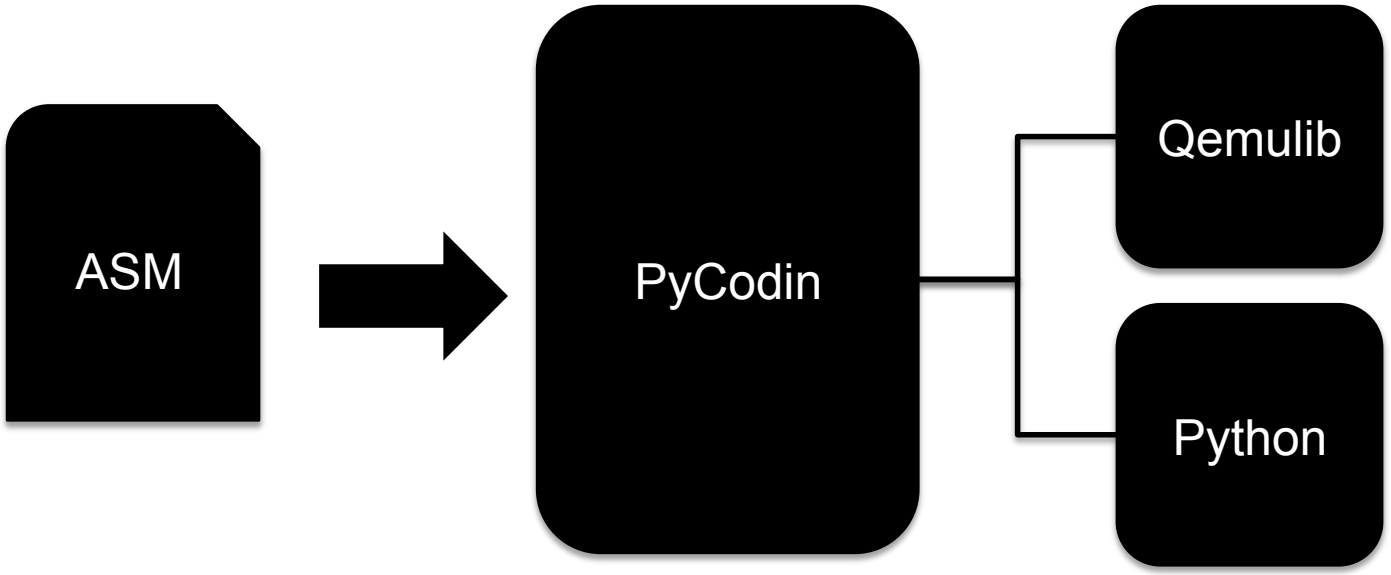
Pycodin – Funcionalidad

- **Crear** un ambiente de ejecución (memoria , registros de cpu)
- **Controlar** la ejecución de código máquina
- **Inspeccionar** el estado de ejecución y modificarlo.
- **Hookear** llamados a funciones y handlearlos desde python.

Pycodin - Diseño

- **Exporta una interfaz con primitivas para ejecución de código.**

- **Utiliza como backend el código de qemu modificado para funcionar como librería.**



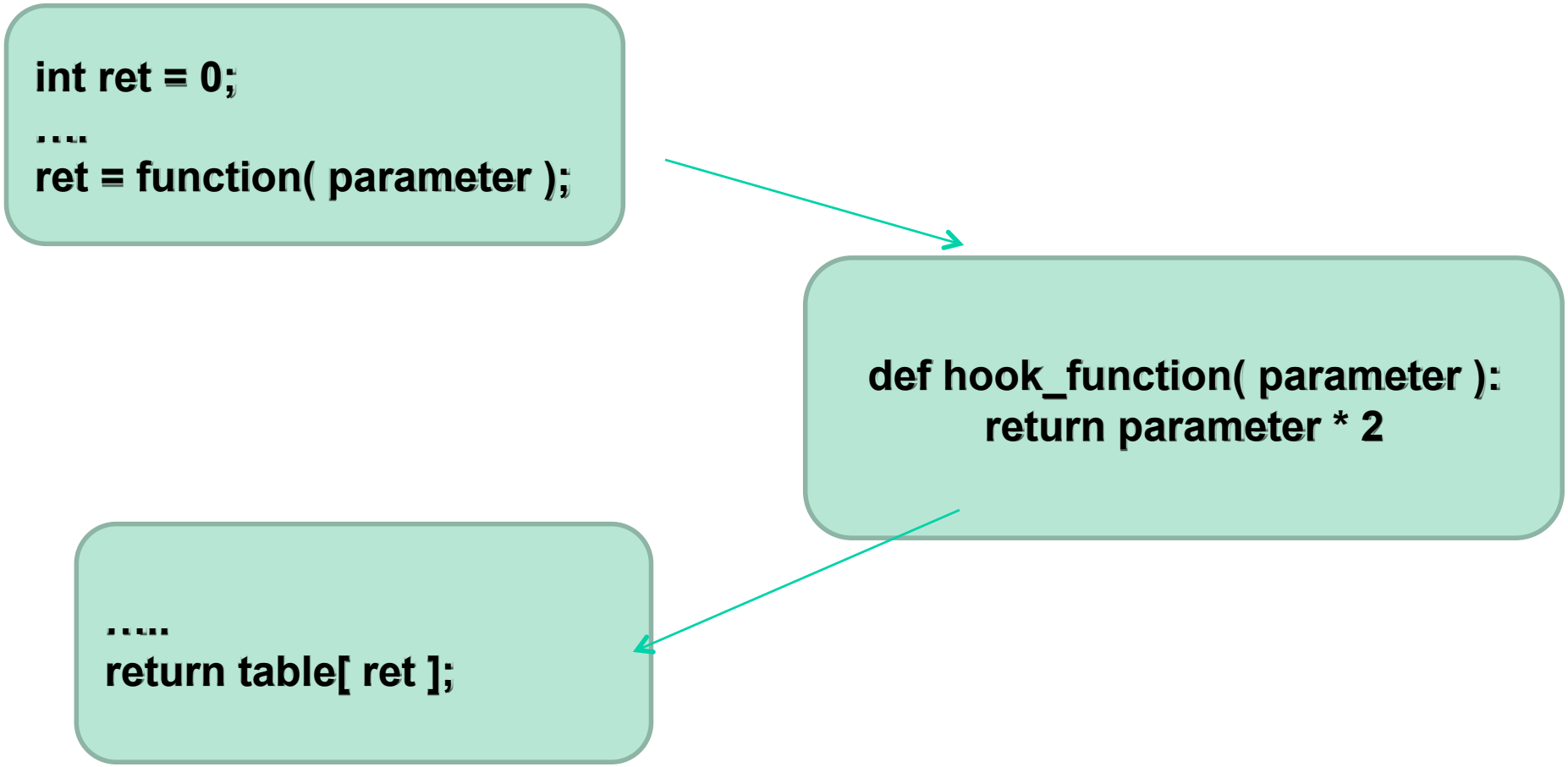
Pycodin - Interacción con qemu

- **Utilizamos ctypes para realizar llamadas al código en C.**

- **Utilizamos qemu**
  - Sin optimizar (no se cachea el código traducido ni se genera de forma optimizada)
  - En single step.

Pycodin – hook de funciones

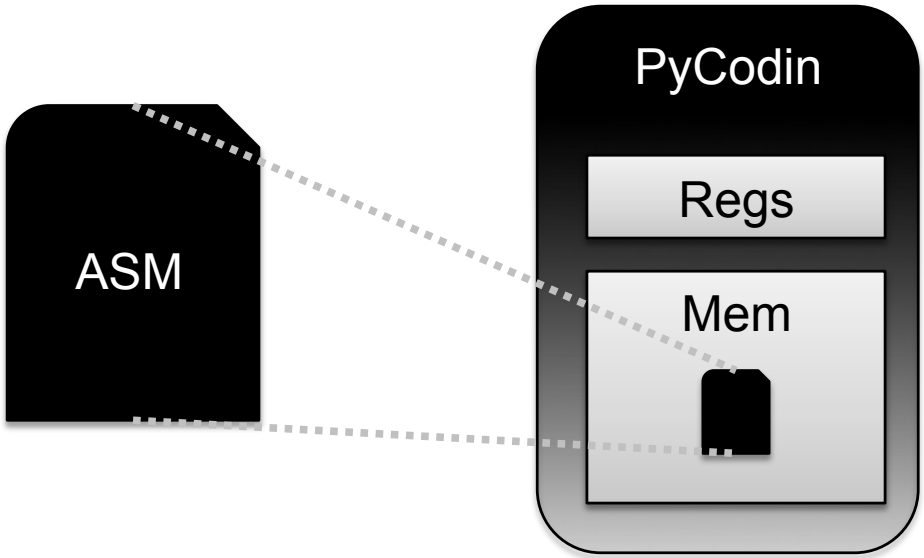
- **Los llamados a funciones del codigo a instrumentar, pueden ser hookeados y handleados por funciones en python.**



Pycodin – Memoria virtual

- **Mapeo en un espacio de "Memoria Virtual"**
  - El codigo a instrumentar
  - Las secciones y direcciones a las cual quiera acceder.

- **Qemu implementa un MMU por software que permite traducir direcciones virtuales en direcciones del host.**



Pycodin - Instrumentación de un codigo simple


```
vm = I386VM()

vm.set_memory_map ( {0x0044777F:0x1024 } )

vm.init_vm( code_begin_addr )

vm.exec_code(
```

Pycodin - Hooks


```
vm.set_memory_map ( {0x0044777F:0x1024,
0x40404040:FUNCTION_HOOK_LEN, 0x40404041: 0x1,
0x00120000:0x200 , 0x40404041:50} )

vm.hook_function( 0x40404040, test_function_hook, "stdcall", IARG_INT,
*( IARG_ULONG, IARG_ULONG, IARG_PTR  ) )
```

URLs utiles

- **Qemu :** http://wiki.qemu.org/Main_Page
- **Pycodin :** http://oss.coresecurity.com
- **Pin :** http://www.pintool.org

Trabajo futuro

- Publicar el código bajo alguna licencia copada
- Hacer el código de la libreria platform independent (que corra en windows y linux al menos)
- Implementar otras arquitecturas
- Implementar inyección del interprete de python para tracear procesos starteados previamete

Preguntas