2017 年 7 月 17 日，星期一

PyREBox - 可用 Python 编写脚本的逆向工程沙盒

作者：Xabier Ugarte Pedrero

在 Talos，我们不断努力提高我们自身的研究和威胁情报能力。因此，我们不仅利用标准工具进行分析，而且专注于创新，开发独有技术来应对新的挑战。此外，Talos 一直以来都支持开源项目，并开放了目前我们工作流程中使用的很多不同项目和工具的源代码（如 FIRST 和 BASS）。

在本文中，我们将介绍 PyREBox - 可用 Python 编写脚本的逆向工程沙盒。PyREBox 基于 QEMU，其目的是通过从不同的角度提供动态分析和调试功能，来帮助实施逆向工程。PyREBox 可以检查正在运行的 QEMU 虚拟机 (VM)，修改其内存或寄存器，并使用简单的 Python 脚本监测其执行情况。QEMU（当作为整个系统仿真程序时）可以模拟完整的系统（CPU、内存、设备…）。通过使用虚拟机内省 (VMI) 技术，它不需要对访客操作系统进行任何修改，因为它可以在运行时透明地从内存中检索信息。

有多个学术项目（如 DECAF、PANDA、S2E 或 AVATAR）之前就已将基于 QEMU 的监测用于逆向工程任务。这些项目允许使用 C/C++ 语言编写插件，并实施了多个高级功能，例如动态污点分析、符号执行，甚至包括记录和重放执行轨迹。借助 PyREBox，我们致力于通过此技术保持简单的设计，使系统满足威胁分析师的需求。

目标

- 提供一个完整的系统仿真平台，使其具备用于检查仿真访客系统的简单接口：对系统事件进行精细监测。
  - 基于 Volatility 的虚拟机内省 (VMI)。访客无需安装代理或驱动程序。
  - 一个基于 IPython 的外壳接口。
  - 一个基于 Python 的脚本引擎，允许利用基于此语言的任何安全工具（构成最大的生态系统之一）。

- 具有简洁的设计，实现了与 QEMU 的分离。许多基于 QEMU 构建的项目不会随 QEMU 升级而升级，因此缺少新功能和优化以及安全更新。为此，我们将 PyREBox 作为一个独立的模块加以实施，只需极少的修改即可将该模块与 QEMU 编译在一起。
- 支持不同的架构。目前，PyREBox 仅支持使用 x86 位和 x86-64 位架构的 Windows，但是其设计允许支持其他架构，例如 ARM、MIPS 或 PowerPC，以及其他操作系统。随着越来越多的设备（采用各种架构和操作系统）容易受到攻击，对这些系统的支持变得越来越重要。我们计划在未来支持其他架构和操作系统。

PyREBox 的工作原理是什么？

PyREBox 与 QEMU 构建在一起，通过极少的修改即可监视系统上的特定事件。QEMU 基于微型代码生成器 (TCG)，该引擎允许将不同架构的代码转换为通过虚拟 CPU 运行的中间语言，然后将此中间语言编译为运行 QEMU 的目标架构。PyREBox 允许监测这个转换的代码。因此，用户可以在运行时动态注册回调，而 PyREBox 会将所有必要的参数转换为可读的 python 对象。用户唯一需要知道的是，当系统中触发某个事件时，它将执行一个 python 函数。

此外，PyREBox 利用 Volatility 来执行虚拟机内省，这有助于弥合系统的物理视图（仿真机）和操作系统的逻辑视图（进程、模块、符号…）之间的语义差异。我们还使用 C/C ++ 实施了一些需要频繁调用的例程，以通过避免一直进入 Python 运行时环境来提高系统的效率。该方法允许检查正在运行的进程、它们加载的模块以及它们导出的符号，而不将任何代理或驱动程序插入到仿真系统中。

通过这种方法，用户可以在物理层面检查访客的状况，更重要的是，用户还可以了解每个时刻在运行哪个进程，将分析集中在一个或多个特定进程上，甚至为任何进程在任何层级（用户或核心空间）插入隐蔽断点。

PyREBox 能为我做什么？

PyREBox 为用户提供两个主要接口。一方面，在 QEMU 中运行访客系统时，用户可以启动外壳，并使用许多不同的命令来检查正在运行的虚拟机。此外壳基于 IPython，因此它允许在其 API 之上编写 python 代码片段，并使用 Python 表达式来表达 PyREBox 命令参数。另一方面，用户可以编写 python 脚本，为系统上的特定事件注册回调。

IPYTHON 外壳

要启动 PyREBox 外壳，只需在 QEMU 的显示器上键入 sh 命令。它将立即启动一个 IPython 外壳。此外壳记录命令历史以及定义的变量。例如，您可以保存一个值，并在之后再次启动外壳时，在不同的执行点重新获得该值。PyREBox 利用了 IPython 中的所有可用功能，例如自动完成、命令历史记录、多行编辑和自动命令帮助生成。

PyREBox 将允许您以相当隐蔽的方式调试系统（或特定进程）。不同于传统调试器会留在被调试的系统内（甚至修改被调试进程的内存以插入断点），PyREBox 完全保持位于受检测系统的外部，并且不需要向访客安装任何驱动器或组件。

PyREBox 提供了一整套用于检查和修改正在运行的虚拟机状态的命令。输入 *list_commands* 即可获得完整列表。

```
[3] pyrebox> list_commands

    MISCELLANEOUS COMMANDS
    ----------------------
    list_commands      - Print this list
    list_vol_commands - List volatility commands_
    vol                - Execute any volatility command. E.g.: vol pslist
    proc               - Select address space of process
    setcpu             - Select CPU to operate on
    mon                - Start monitoring process
    unmon              - Stop monitoring process
    savevm             - Save vm status
    loadvm             - Load vm status
    quit               - Exit this prompt
    q                  - Exit this prompt
    cont               - Exit this prompt
    c                  - Exit this prompt
    ctrl-d             - Exit this prompt

    ?                  - Use it to obtain help for a command. E.g.: ps?
    help(api)          - Get help for the pyrebox API you can import and use in the interactive shell
    help(r_cpu)    - Get help for a specific function of the API

    INSTROSPECTION COMMANDS
    -----------------------
    ps                 - List processes
    lm                 - List modules
    x                  - Show symbols matching pattern (module!function)
    ln                 - List nearest symbols to address

    CPU / MEMORY MANIPULATION
    -------------------------
    r                  - Write register
    db|dw|dd|dq        - Display memory byte, word, dword, qword
    eb|ew|ed|eq        - Edit memory byte, word, dword, qword
    write              - Write a buffer to memory
    dump               - Dump a buffer of memory into command line.
    print_cpu          - Show CPU status (registers)

    DISASSEMBLY
    -----------
    dis                - Disassemble N instructions starting from PC, on the context of the running process
    u                  - Disassemble N instructions starting from a given address, on the context of selected address space (proc)

    BREAKPOINTS
    -----------
    bp                 - Set execution breakpoint at address(es)
    bpw                - Set memory write breakpoint at address(es)
    bpr                - Set memory read breakpoint at address(es)
    bl                 - List breakpoints
    bd                 - Disable breakpoint
    be                 - Enable breakpoint


    SEARCH
    ------
    strings            - Show printable strings in a given memory area
    s                  - Search for string or byte parttern in a given memory area

    DINAMICALLY IMPORTED COMMANDS
    -----------------------------


    Use custom <command> <args..>

    set_target         - Set target process - Custom command
    custom_command_example        -  Example of custom command. This first line will be shown as command description when -list_commands is called.
```

在执行虚拟机期间，如果您键入 vol 和相应的 volatility 命令，就可以随时运行任何 volatility 插件。键入 *list_vol_commands*，即可获得可用 volatility 插件的完整列表。此列表自动生成，因此它还将显示您在 PyREBox 的 *volatiliy /* 路径上安装的任何 volatility 插件。

```
[4] pyrebox> %list_vol_commands

        Supported volatility commands:

                amcache         Print AmCache information
                apihooks        Detect API hooks in process and kernel memory
                atoms           Print session and window station atom tables
                atomscan        Pool scanner for atom tables
                auditpol        Prints out the Audit Policies from HKLM\SECURITY\Policy\PolAdtEv
                bigpools        Dump the big page pools using BigPagePoolScanner
                bioskbd         Reads the keyboard buffer from Real Mode memory
                cachedump       Dumps cached domain hashes from memory
                callbacks       Print system-wide notification routines
                clipboard       Extract the contents of the windows clipboard
                cmdline         Display process command-line arguments
                cmdscan         Extract command history by scanning for _COMMAND_HISTORY
                connections     Print list of open connections [Windows XP and 2003 Only]
                connscan        Pool scanner for tcp connections
                consoles        Extract command history by scanning for _CONSOLE_INFORMATION
                crashinfo       Dump crash-dump information
                deskscan        Poolscaner for tagDESKTOP (desktops)
                devicetree      Show device tree
                dlldump         Dump DLLs from a process address space
                dlllist         Print list of loaded dlls for each process
                driverirp       Driver IRP hook detection
                drivermodule    Associate driver objects to kernel modules
                driverscan      Pool scanner for driver objects
                dumpcerts       Dump RSA private and public SSL keys
                dumpfiles       Extract memory mapped and cached files
                dumpregistry    Dumps registry files out to disk
                editbox         Displays information about Edit controls. (Listbox experimental.)
                envars          Display process environment variables
```

```
[13] pyrebox(1f0)> vol pslist
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------
0x817caa00 System                    4      0     72      150 ------      0
0x817d9b28 smss.exe                312      4      3       19 ------      0 2017-06-02 20:31:44 UTC+0000
0x81621020 csrss.exe               408    312     11      305      0      0 2017-06-02 20:31:45 UTC+0000
0x81620ad0 winlogon.exe            432    312     21      503      0      0 2017-06-02 20:31:46 UTC+0000
0x81609a00 services.exe            484    432     16      246      0      0 2017-06-02 20:31:48 UTC+0000
0x81604da0 lsass.exe               496    432     23      325      0      0 2017-06-02 20:31:48 UTC+0000
0x8179f020 svchost.exe             652    484     19      193      0      0 2017-06-02 20:32:00 UTC+0000
0x815f0998 svchost.exe             760    484      9      208      0      0 2017-06-02 20:32:29 UTC+0000
0x815ee488 svchost.exe            1064    484     58     1028      0      0 2017-06-02 20:33:55 UTC+0000
0x81605a08 svchost.exe            1084    484      5       58      0      0 2017-06-02 20:33:57 UTC+0000
0x8160e838 svchost.exe            1148    484     13      178      0      0 2017-06-02 20:34:03 UTC+0000
0x817427a0 explorer.exe           1156   1120     11      305      0      0 2017-06-02 20:34:04 UTC+0000
0x8161c880 spoolsv.exe            1304    484     15      122      0      0 2017-06-02 20:34:11 UTC+0000
0x81594298 alg.exe                1768    484      7      100      0      0 2017-06-02 20:34:30 UTC+0000
0x8158f020 wscntfy.exe            1796   1064      1       26      0      0 2017-06-02 20:34:33 UTC+0000
0x815eb020 wuauclt.exe             112   1064      8      176      0      0 2017-06-02 20:35:04 UTC+0000
```

最后，您还可以在脚本中定义自己的命令！您只需要创建一个名称以 *"do_"* 开头的函数，PyREBox 将为您完成其余的操作。

如果您需要比命令更有表现力的东西，则可以使用 API 编写一个 Python 代码片段。

如需获取有关 API 的详细说明，您可以在外壳中键入 *help(api)*。

脚本化

PyREBox 允许动态加载可以注册回调函数的脚本。系统会在发生以下特定事件时调用这些函数：

- 指令和/或基本块执行
- 内存读取/写入
- 进程创建/终止
- 上下文切换
- TLB 未命中
- 网络接口和键盘活动

此框架受 DECAF 等项目的启发，因此，我们支持 DECAF 中支持的许多回调类型。

鉴于 PyREBox 集成了 Volatility，您可以利用所有 volatility 插件来在 Python 脚本中进行内存调查分析。许多最著名的逆向工程工具都是用 Python 实施的，或者至少有 Python 绑定。我们的方法允许将所有这些工具集成到任何脚本中。

脚本界面还允许定义自定义命令。脚本只需要声明特定原型后面的函数。这足以创建一个新命令，加载了脚本之后，该命令将在外壳中可用。此功能允许将任何 Python 工具不仅集成到脚本引擎中，还可以集成到 IPython 外壳上，只需用 Python 编写一个简单的包装器即可。

脚本还可以在发生特定事件或符合特定条件情况下，为您启动一个外壳。用户可以监控并记录事件，并且每当满足条件时，只需简单地调用 *start_shell()* 即可暂停虚拟机并在该特定点启动一个外壳。

以下代码片段代表一个简单的脚本，它能够在脚本加载到 PyREBox 时注册进程创建的回调。每次创建一个新进程时，系统都会启动一个 PyREBox 外壳。它还会实施一个叫做 *my_command* 的自定义命令，您可以通过键入 *custom my_command* 从 PyREBox 外壳调用该命令。

```
#!/usr/bin/python
import sys
import api
from ipython_shell import start_shell
from api import CallbackManager

#Callback manager
cm = None
#Printer
pyrebox_print = None

if __name__ == "__main__":
    #This message will be displayed when the script is loaded in memory
    print "[*] Loading python module %s" % (__file__)

def new_proc(pid,pgd,name):
    '''
    Process creation callback.Receives 3 parameters:
        :param pid: The pid of the process
        :type pid: int
        :param pgd: The PGD of the process
        :type pgd: int
        :param name: The name of the process
        :type name: str
    '''
    global pyrebox_print
    global cm

    #Print a message.
    pyrebox_print("New process created! pid: %x, pgd: %x, name: %s" % (pid,pgd,name))
    #Start a PyREBox shell exactly when a new process is created
    start_shell()


def initialize_callbacks(module_hdl,printer):
    '''
    Initilize callbacks for this module.
    '''
    global cm
```

```
    global pyrebox_print
    #Initialize printer function
    pyrebox_print = printer
    pyrebox_print("[*]    Initializing callbacks")
    #Initialize the callback manager
    cm = CallbackManager(module_hdl)

    #Register a process creation callback
    new_proc_cb = cm.add_callback(CallbackManager.CREATEPROC_CB,new_proc)

    pyrebox_print("[*]    Initialized callbacks")

def clean():
    '''
    Clean up everything.
    '''
    global cm
    print "[*]    Cleaning module"
    #This call will unregister all existing callbacks
    cm.clean()
    print "[*]    Cleaned module"

def do_my_command(line):
    ''' Short description of the custom command.

        Long description of the custom command
    '''
    global pyrebox_print
    global cm

    #Implementation of the command functionality
    pyrebox_print("This is a custom command")
```

最后，由于 python 回调可能会引起性能损失（特别是在发生诸如执行指令等频繁事件时），这可能也会创建触发器。触发器是使用 C/C++ 开发的本地代码插件，可在运行时，在 Python 调用执行前发生任何事件时动态插入。这样就可以限制触发 python 代码的事件数量，以及预先计算本地代码中的值。

有关可用功能的完整参考，可以参阅项目文档。

结论

我们相信 PyREBox 可以成为逆向工程的有用工具。其与 Python 和 Volatility 的集成支持无数应用途径，从恶意软件或漏洞攻击/漏洞分析到固件分析（将来，我们还计划支持其他架构和操作系统）。它可以轻松用 Python 实现的许多安全工具集成。利用此框架的设计，只需编写一个简单的包装器脚本即可轻松创建一组新的外壳命令，以与任何 python 库结合。

我们已开放这个内部开发的工具的源代码，因为我们相信它对整个社区是有价值的，并邀请研究人员贡献新的脚本，以发挥 PyREBox 的充分潜力。

发布者：WILLIAM LARGENT；发布时间：21:55
标签：PYTHON、逆向工程、逆向、沙盒、TALOS