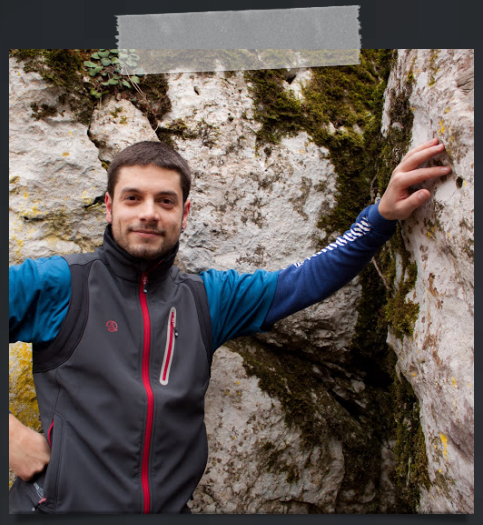
PyREBox

Making Dynamic Instrumentation Great Again

TALOS™



@xabiugarte

Malware Research Team

@

TALOS

[*advertising space…*]

Deep Packer Inspector

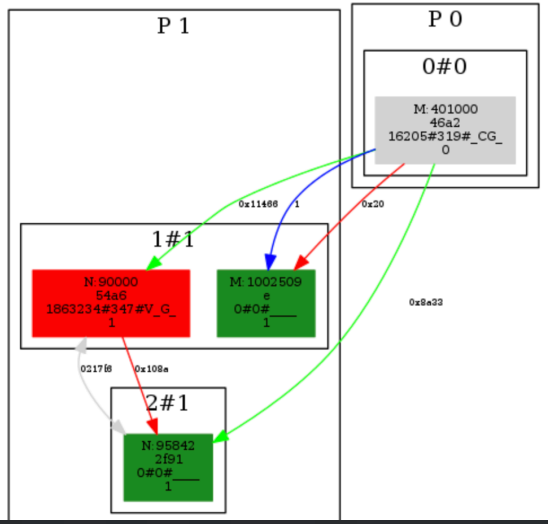
https://packerinspector.github.io

https://packerinspector.com

TALOS

Summary

Visibility	Public
Main file's SHA256	592ce33c826a4a6cfef63781fd6f48eaf5ab8db7ba15e89ab6f68a21539a56e2
Complexity	Type IV
Packer identification (signature based)	-
File entropy	7.04759
Number of processes	2
Number of layers	3




WHY IS THIS GUY TALKING ABOUT
ABOUT PACKER INSPECTOR??
imgflip.com

TALOS

Many instrumentation frameworks…

PyKD

PIN

Unicorn

TEMU/DECAF

DynamoRIO

PyDbg

S2E

PANDA

Frida

WinAppDbg

Avatar

DynInst

TALOS

PyREBox

- ☐ Motivation
- ☐ Principles
- ☐ Design / architecture
- ☐ Features
- ☐ Malware monitor
- ☐ Future work

TALOS

Technical aspects

- Single process/binary, or whole system?
- What events does it hook / instrument?
- Transparency?

Practical aspects

- How ‘easy’ is it to use?
- Programming languages?

Other aspects

- How often is it ‘updated’?
- Community?
- Is the project even alive?

TALOS

Frameworks based on emulation

- **Full system** emulator (vs. user-mode)
- QEMU!
- Emulate CPU, BIOS, memory, devices
  - Boot and fully emulate unmodified O.S.
  - (Linux, Solaris, Windows, DOS, BSD…)
- Different guest architectures on different host architectures (TCG)

TALOS

QEMU

- “Transparent” instrumentation
  - Emulated **memory** is **not modified**
- No agent needed
- Full system == …
  - Allows to monitor inter-process interaction
  - Allows to instrument / inspect kernel

TALOS

Some shortcomings...

- PANDA, DECAF, etc…
  - Plugins are coded in C/C++
  - I prefer **python**!
    - Faster development
    - Great libraries
- Complex QEMU modifications
  - Risk of not updating frequently as QEMU evolves

TALOS

So, what is PyREBox?

Yet another dynamic instrumentation engine

- **Interactive** analysis
  - Allows inspecting memory/registers
  - Useful built-in commands
  - IPython

TALOS

So, what is PyREBox?

- **Scripting** (python)
- Callback types…
  - Instruction/block begin/end
  - Memory read/write
  - Specific opcode execution
  - Process create/remove
  - Module load/unload
  - TLB flush / context change
- Extend shell with **new commands**

TALOS

QEMU

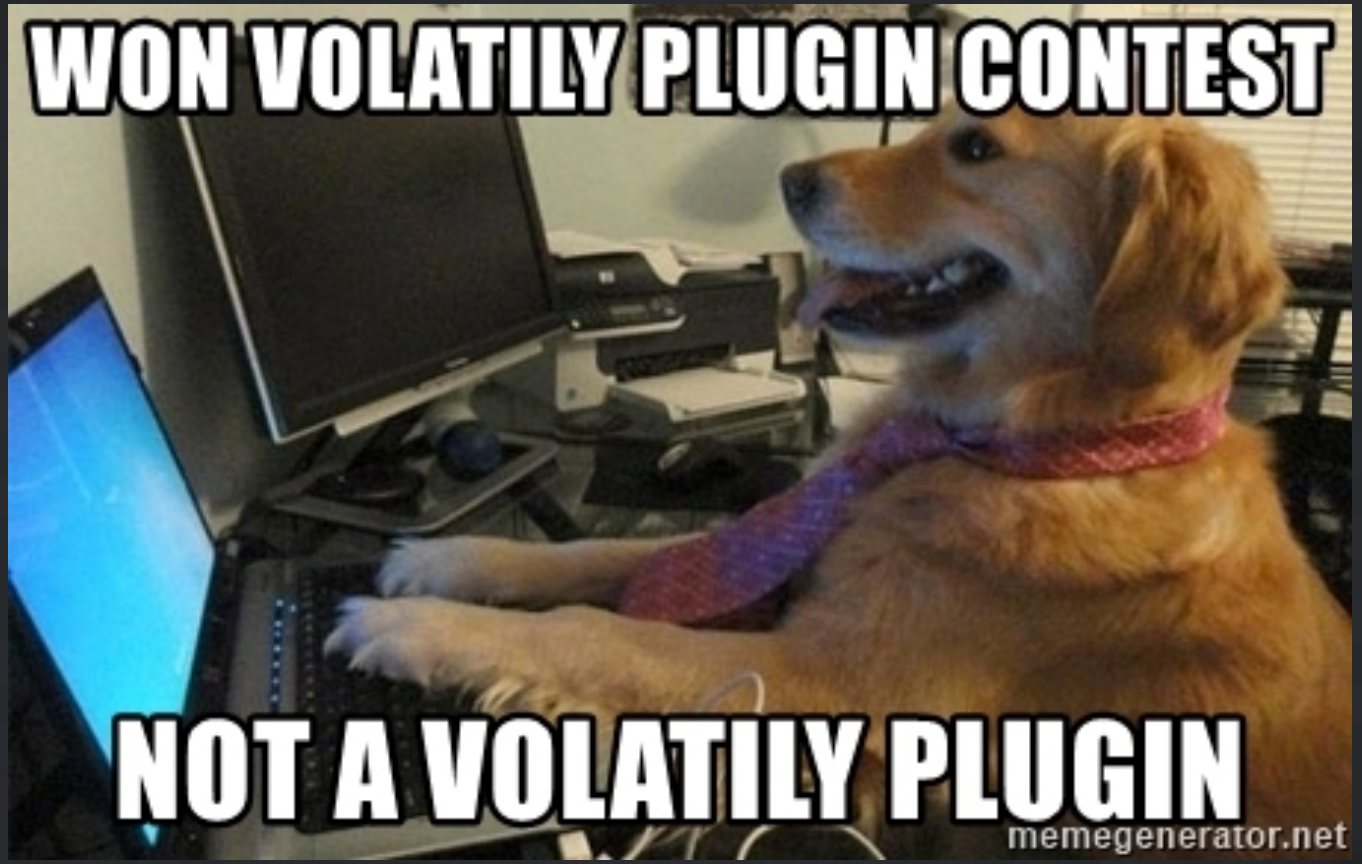
- Full system emulator (QEMU)

  What about **hardware assisted virtualization**?

- E.g.: KVM

- Target & host arch. must be the same
- Host O.S. dependent
  - (e.g.: KVM won't run on Windows)

TALOS


WON VOLATILY PLUGIN CONTEST
NOT A VOLATILY PLUGIN
memegenerator.net

TaLOS

So, what is PyREBox?

- **Leverages Volatility** for memory introspection
- It is free!! (as in freedom)

General Public License

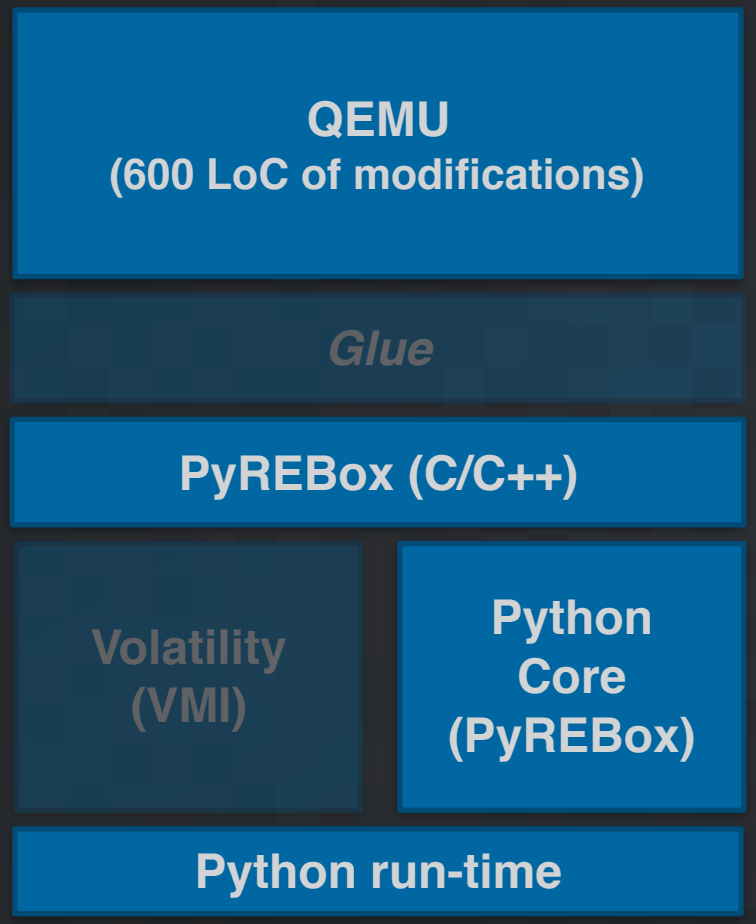
TALOS

Design

Some principles...

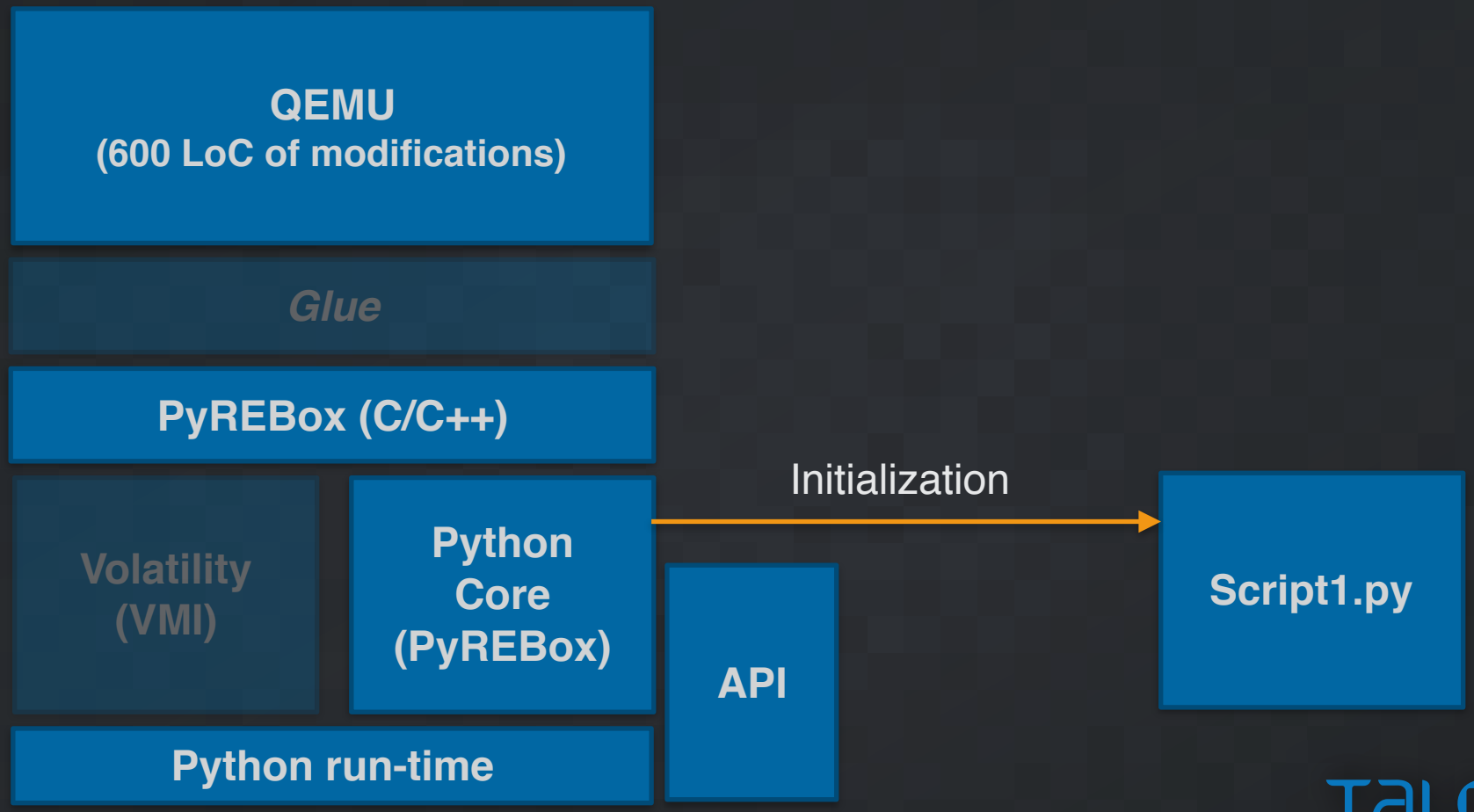
- Interaction and scripts based in **python**
  - Tradeoff: high overhead
- KISS: Keep ***Instrumentation Simple*** Stupid
  - **Minimal modifications** to QEMU
  - Core of the framework **de-coupled** from QEMU
  - **Easier to upgrade** to new QEMU versions
  - Tradeoff: advanced features
    - Taint analysis, record replay…

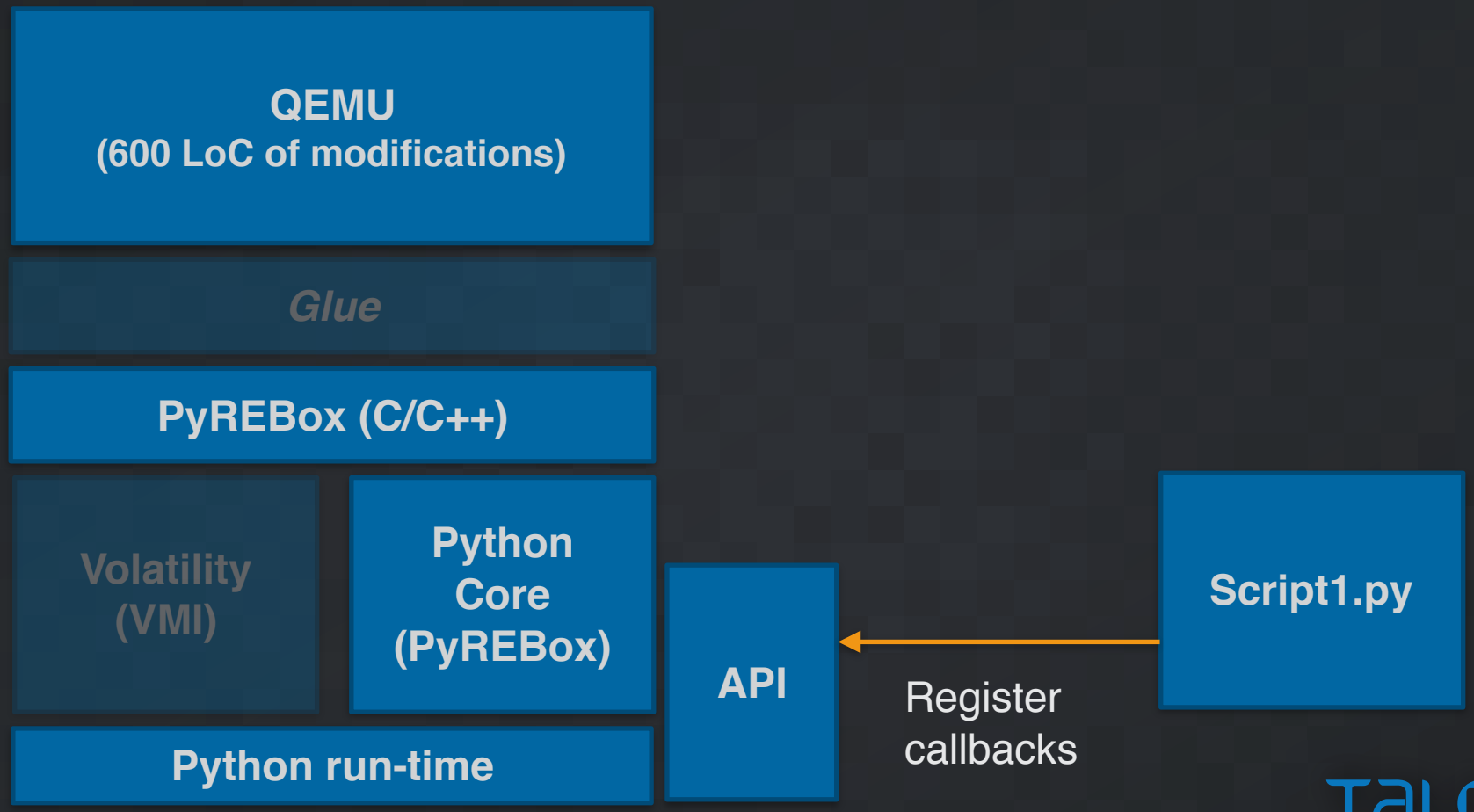
TALOS

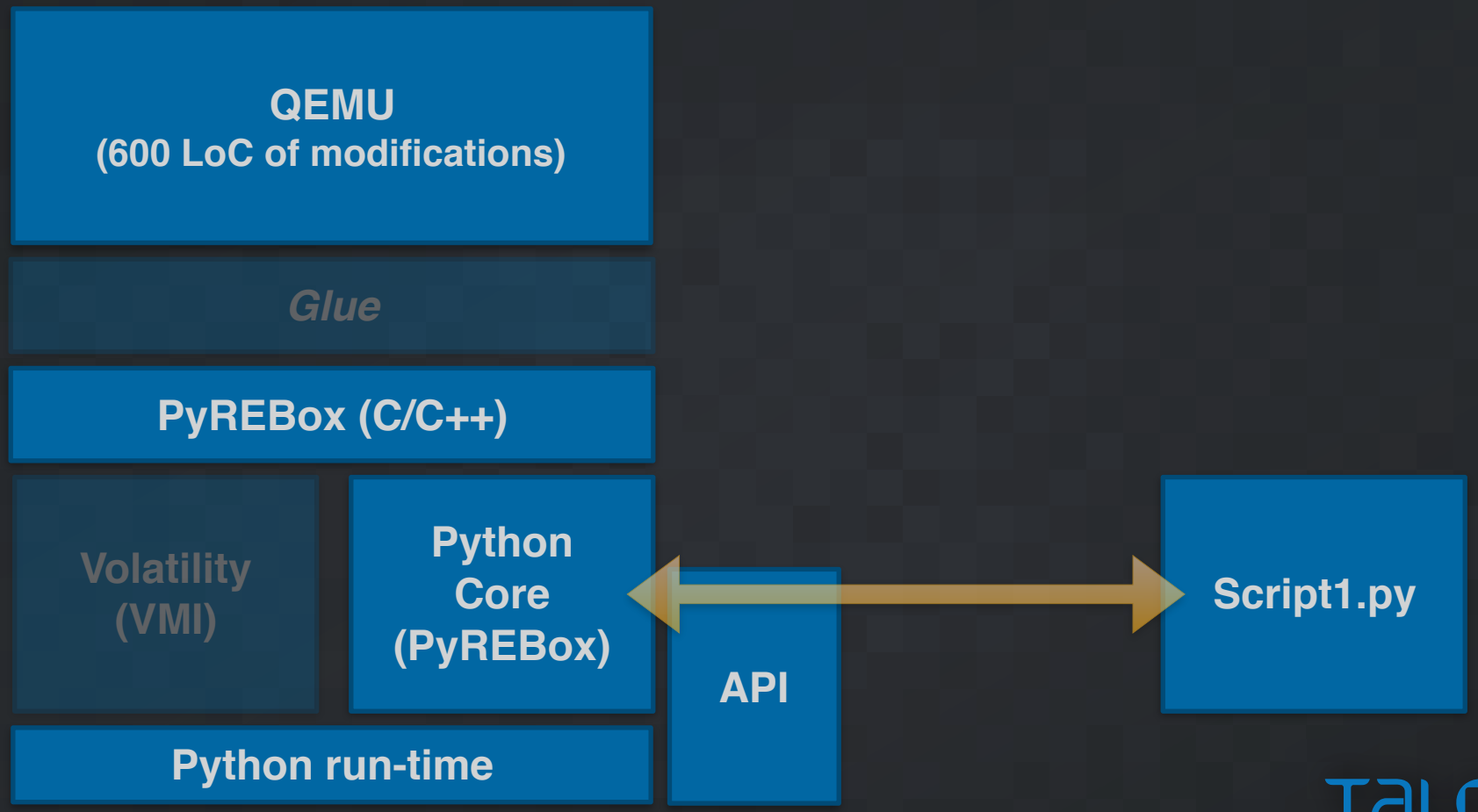


Interactive shell
Python-based API

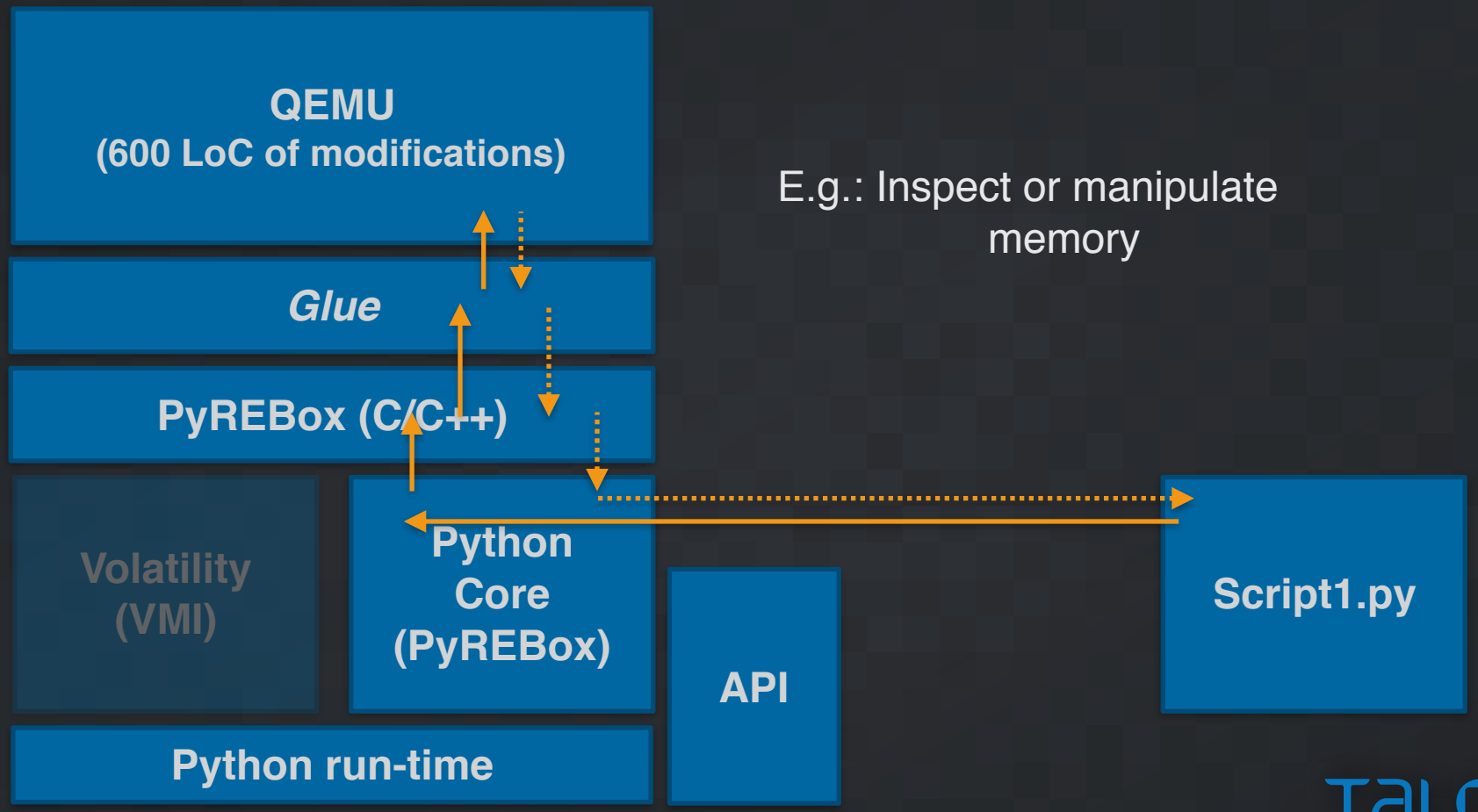
TALOS


QEMU
(600 LoC of modifications)
Glue
PyREBox (C/C++)
Volatility
(VMI)
Python
Core
(PyREBox)
Python run-time
API
Initialization
Script1.py


QEMU
(600 LoC of modifications)
Glue
PyREBox (C/C++)
Volatility
(VMI)
Python
Core
(PyREBox)
Python run-time
API
Register
callbacks
Script1.py


QEMU
(600 LoC of modifications)
Glue
PyREBox (C/C++)
Volatility
(VMI)
Python
Core
(PyREBox)
API
Script1.py
Python run-time

TALOS



E.g.: Inspect or manipulate memory

TALOS

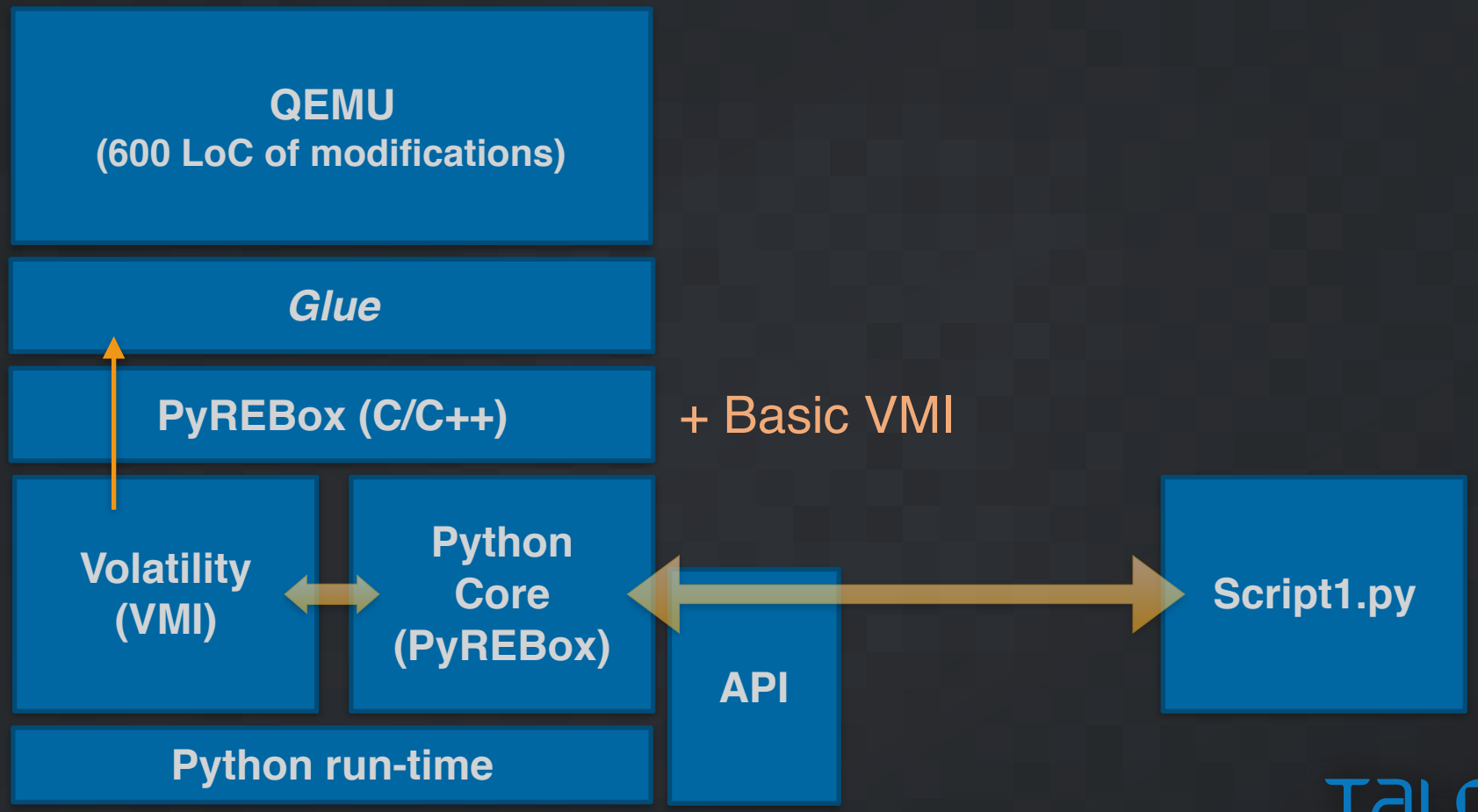
VMI

- We see the system as a *raw* CPU!!

- Only memory, registers, devices
- Sequence of instructions
- Processes, threads, handles, libraries…
  - **Abstractions** of the **O.S.**

- **Virtual Machine Introspection**
  - Understand these abstractions

TALOS


QEMU
(600 LoC of modifications)
Glue
PyREBox (C/C++)
+ Basic VMI
Volatility
(VMI)
Python
Core
(PyREBox)
API
Script1.py
Python run-time

TALOS

VMI

- Support for **Windows and Linux, 32 and 64 bit**
  - Process enumeration
  - Module (DLL / shared library) enumeration
  - Symbol resolution (exported symbols)
- Deliver certain callbacks

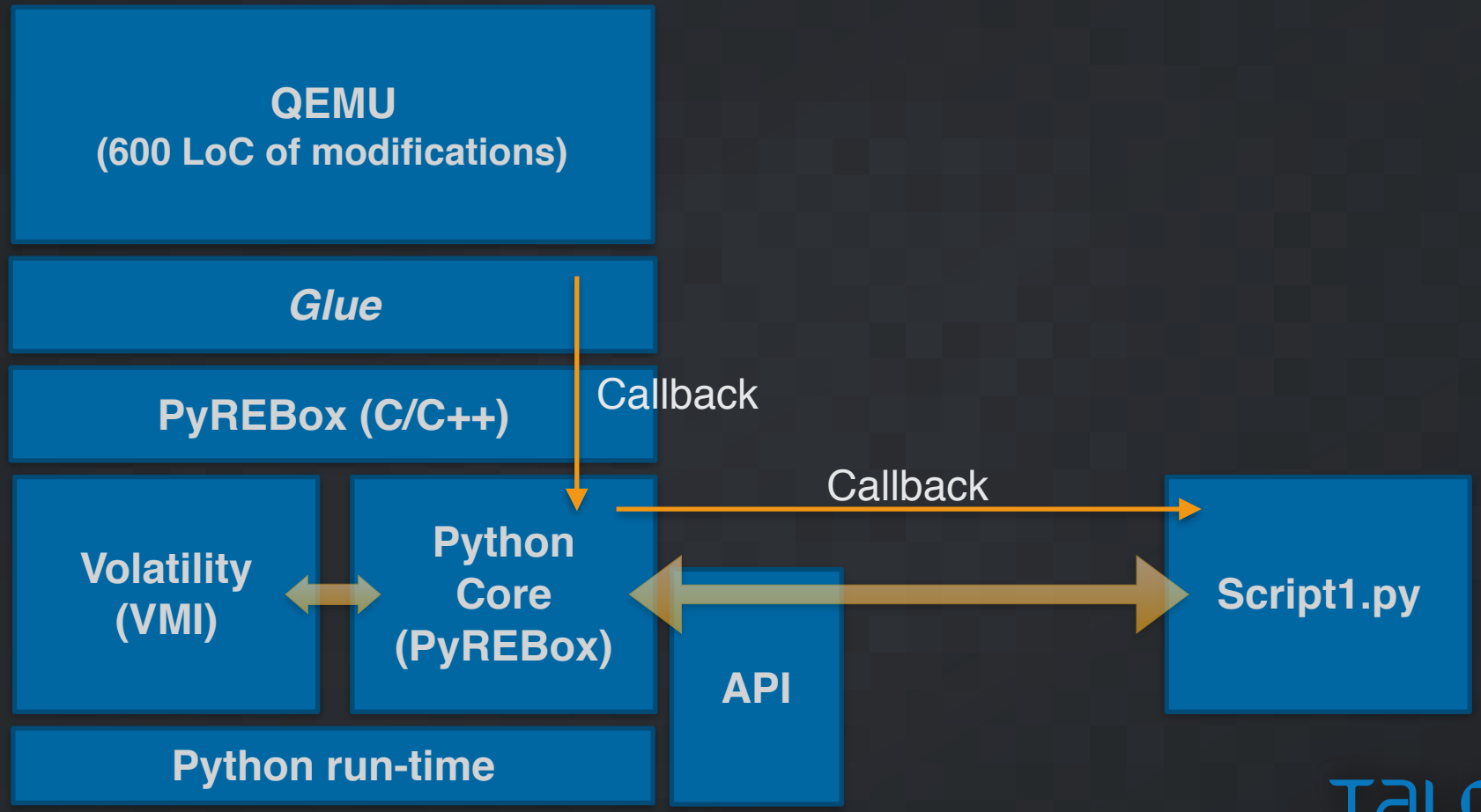
TALOS

Triggers

- Python can be **prohibitively expensive**
  - Instruction begin, memory read…
- Triggers
  - C/C++ snippets
  - Compiled as shared libraries (.so)
  - Loaded at runtime
  - Returns 0 if callback should not be delivered, 1 otherwise.

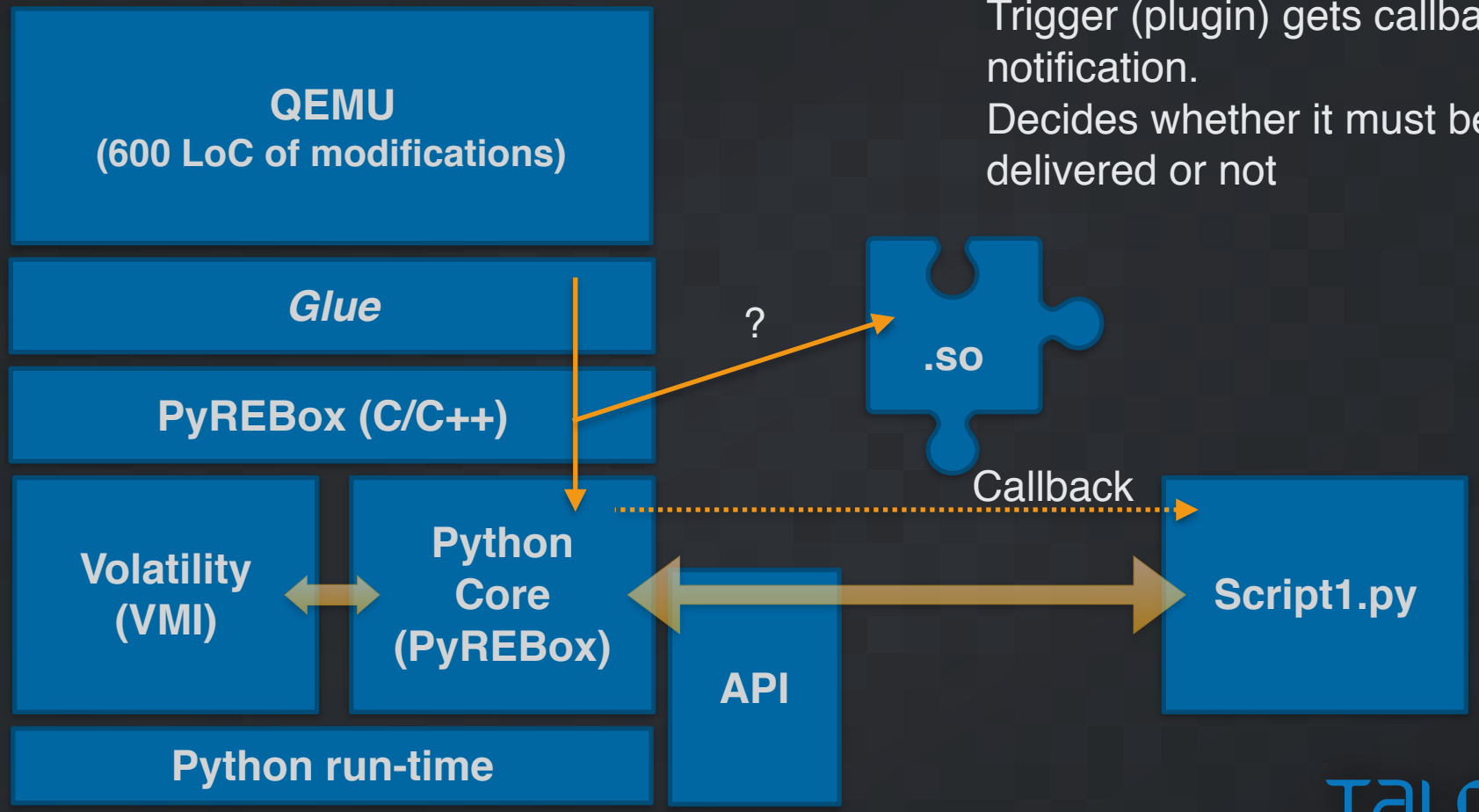
```
int trigger(callback_handle_t handle, callback_params_t params){
return should_deliver;
}
```

TALOS


QEMU
(600 LoC of modifications)
Glue
PyREBox (C/C++)
Callback
Callback
Volatility
(VMI)
Python
Core
(PyREBox)
API
Script1.py
Python run-time

TALOS

Trigger (plugin) gets callback notification.
Decides whether it must be delivered or not



TALOS

PyREBox usage

Easy to compile, install

- Compiles and runs (tested):
  - Linux
  - Windows (thanks to linux subsystem)
  - Docker is supported

TALOS

Easy to compile, install

- Starting PyREBox is like starting any QEMU session.
- QEMU options via command line arguments (Check QEMU docs)
- Example scripts provided
- PyREBox configuration file
- Complete PyREBox documentation

https://pyrebox.readthedocs.io/en/latest/

TALOS

PyREBox shell

PyREBox shell

QEMU monitor

- Regular QEMU commands
- Snapshot management
- PyREBox script management

PyREBox shell

- **Pauses the guest**
- Inspect regs/mem
- Modify regs/mem
- Run built-in commands
- Run volatility commands
- Run custom commands
- Run python code (ipython)
- Autocompletion, syntax

TALOS

Scripting

Scripting

- Loaded/unloaded/reloaded
  - Startup script
  - QEMU command
- Can start a shell at any time

```
>start_shell()
```

- Can import and use any python library

TALOS

```python
53 def initialize_callbacks(module_hdl, printer):
54     '''
55     Initilize callbacks for this module. This function
56     will be triggered whenever import_module command
57     is triggered.
58     '''
59     global cm
60     global pyrebox_print
61     from api import CallbackManager
62     # Initialize printer
63     pyrebox_print = printer
64     pyrebox_print("[*]    Initializing callbacks")
65     cm = CallbackManager(module_hdl)
66     cm.add_callback(CallbackManager.CREATEPROC_CB, new_proc, name="vmi_new_proc")
67     cm.add_callback(CallbackManager.REMOVEPROC_CB, remove_proc, name="vmi_remove_proc")
68     pyrebox_print("[*]    Initialized callbacks")
```

TALOS

```python
def clean():
    '''
    Clean up everything. At least you need to place this
    clean() call to the callback manager, that will
    unregister all the registered callbacks.
    '''
    global cm
    pyrebox_print("[*]    Cleaning module")
    cm.clean()
    pyrebox_print("[*]    Cleaned module")
```

TALOS

Script life-cycle

- Script requirements:

```
> requirements = ["plugins.guest_agent"]
```

- Once it is initialized, it will be executed when:
  - An installed callback is triggered
  - A defined command is executed

```
> def do_command(line):
```

TALOS

```
31 def new_proc(pid, pgd, name):
32     global cm
33     pyrebox_print("Process %s: PID:%x PGD:%x" % (name, pid, pgd))
34
35
36 def remove_proc(pid, pgd, name):
37     global cm
38     pyrebox_print("Removed process %s: PID:%x CR3:%x" % (name, pid, pgd))
```

TALOS

Scripting

- Key concepts
  - Processes are identified by their address space (PGD / CR3)
  - Callbacks have different behavior
    - `Check docs!`
  - Monitored process
    - Certain callbacks are only triggered for monitored processes
    - From shell: `mon/unmon`
    - From script: `api.start_monitoring_process`

TALOS

Scripting

- Several scripts provided as **examples**
  - Automatically running a binary and starting a shell on entry point
  - Monitoring memory write + memory execution (unpacked code detection)
  - Tests for every callback type
  - Usage of triggers
- Complete API documentation provided

TALOS

Agent

Agent

- **File transfer** and **execution**
- Process running on the guest that communicates with host via **invalid opcodes**
- Windows and Linux guests supported, 32 & 64 bits
- From shell or scripts:

```
> agent.copy_file(src_path, dest_path)
> agent.execute_file(path, args=[], env={},
  exit_afterwards=False)
```

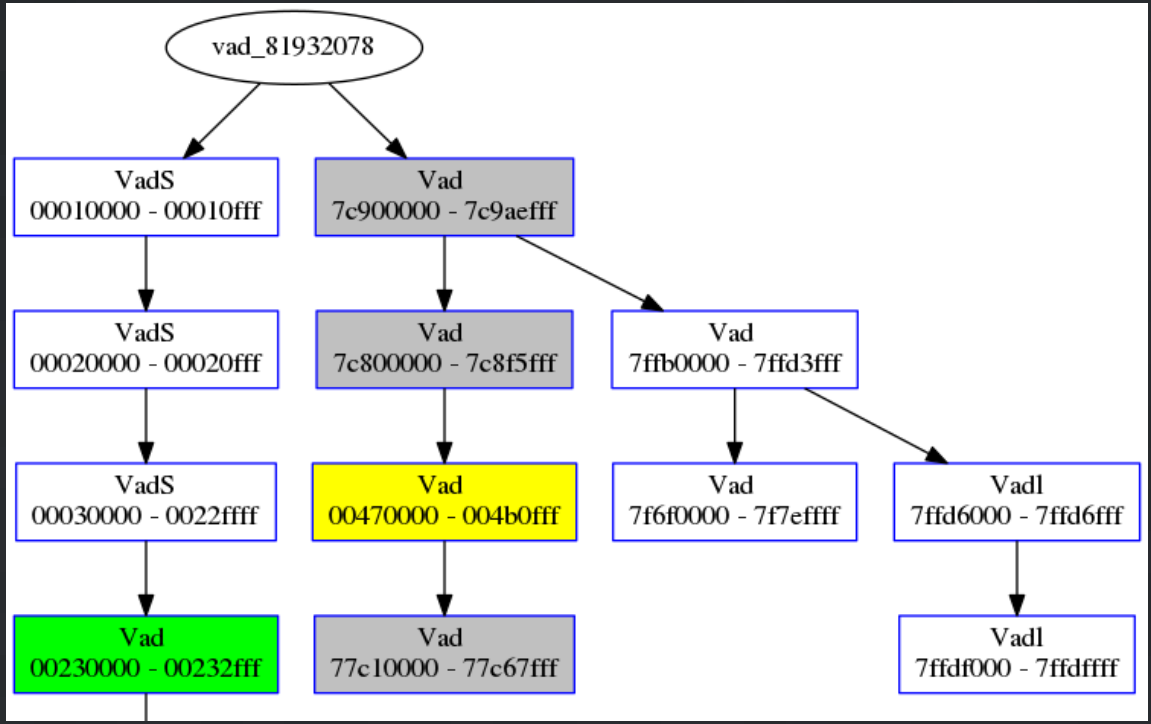
TALOS

Malware Monitor

Malware Monitor

- 4 different modules, configurable (json)
- **API tracer**
  - Text log
  - Binary log (import in IDA)
  - Optionally, **can extract parameters**
- **Memory dumper**
  - Automatically dump under certain conditions
- **Code coverage**
  - Binary log (colorize B.B.s in IDA)
  - Text log (identify jumps between VAD regions)

TALOS


vad_81932078
VadS
00010000 - 00010fff
Vad
7c900000 - 7c9aefff
VadS
00020000 - 00020fff
Vad
7c800000 - 7c8f5fff
Vad
7ffb0000 - 7ffd3fff
VadS
00030000 - 0022ffff
Vad
00470000 - 004b0fff
Vad
7f6f0000 - 7f7effff
Vadl
7ffd6000 - 7ffd6fff
Vad
00230000 - 00232fff
Vad
77c10000 - 77c67fff
Vadl
7ffdf000 - 7ffdffff

TALOS

Malware Monitor

- **Memory event logger** (interproc)
  - Events monitored:
    - Memory allocation / deallocation
    - Process creation, process handle opening
    - Remote memory writes / memory sharing
    - File reading/writing. File mapping
    - Memory permission changes
  - Useful to **track injections, droppers, downloaders**
  - Outputs a condensed text-based report
    - + A log of events

TALOS

Future work

What's next?

- Support for additional architectures (ARM / MIPS)
- Support for other Operating Systems

- Debugging backend for IDA or r2
- Integration into PyREBox of other tools

- Support for other backends (PANDA?)

TALOS

Questions?

talosintelligence.com
blog.talosintel.com
@talossecurity