

PyREBox




...reloaded



TALOSTM



 @xabiugarte

Malware Research Team
@



TALOS

PyREBox

- Motivation
- Design principles / architecture
- Features
- Use cases
- Future work

Dynamic Binary Instrumentation

- ▶ Techniques to **“trace”** the execution of a binary (or system)
- ▶ **Monitor** different events
 - ▶ E.g.: An instruction is executed, a memory address is written...
- ▶ Allow to **write our own instrumentation** code

Many instrumentation frameworks...

PIN
PyKD
DynamoRIO
Unicorn
TEMU/DECAF
PyDbg
S2E
PANDA
Frida
WINAPPCDBG
Avatar
DynInst

Technical aspects

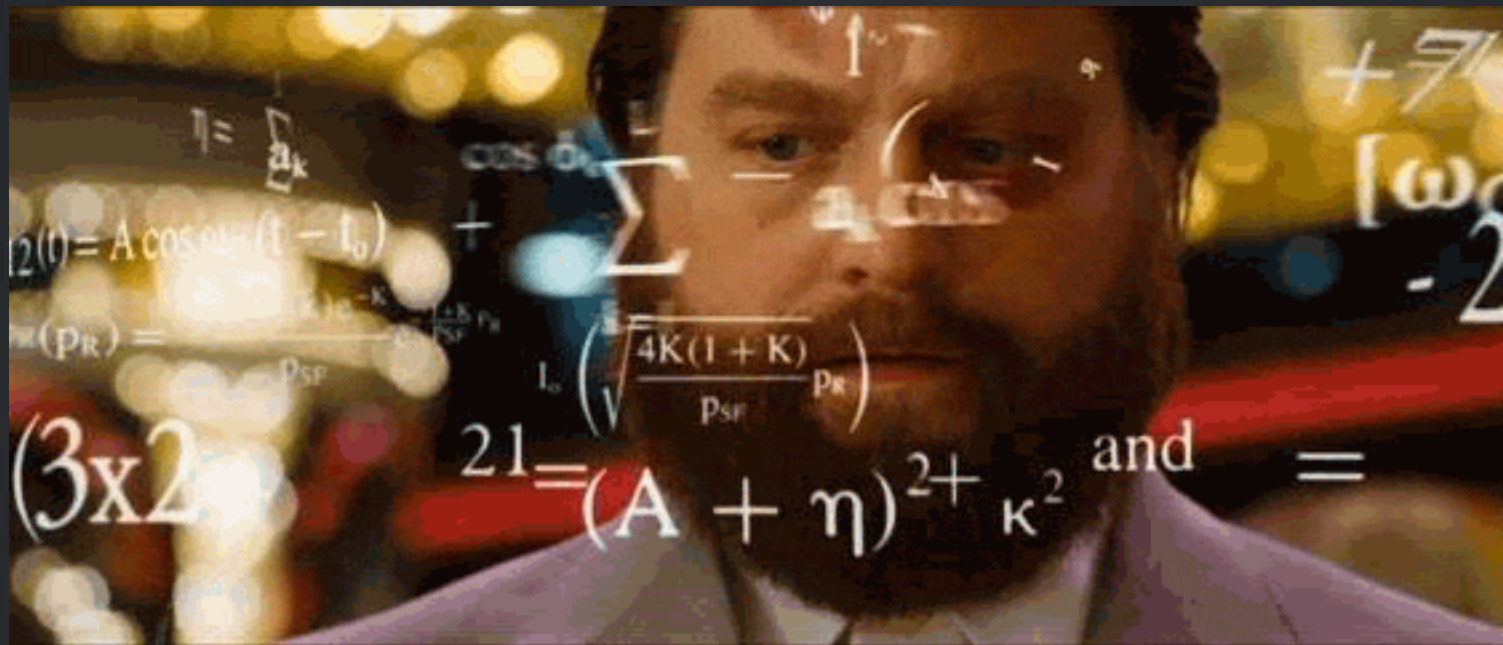
- ▶ Single process/binary, or whole system?
- ▶ What events does it hook / instrument?
- ▶ Transparency?

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 instrumentation**

▶ Full system == ...

- Monitors all the operating system
- Allows to instrument / inspect kernel
- Allows to monitor inter-process interaction

QEMU

- ▶ Started using TEMU and Decaf
- ▶ Based on QEMU
 - ▶ User-mode emulation
 - ▶ Hypervisor (KVM)
 - ▶ **Full system emulation**
- ▶ Emulate CPU, BIOS, memory, devices
 - Boot and fully emulate unmodified O.S.
- ▶ Tiny Code Generator (TCG)

TCG

TCG code

Guest machine code

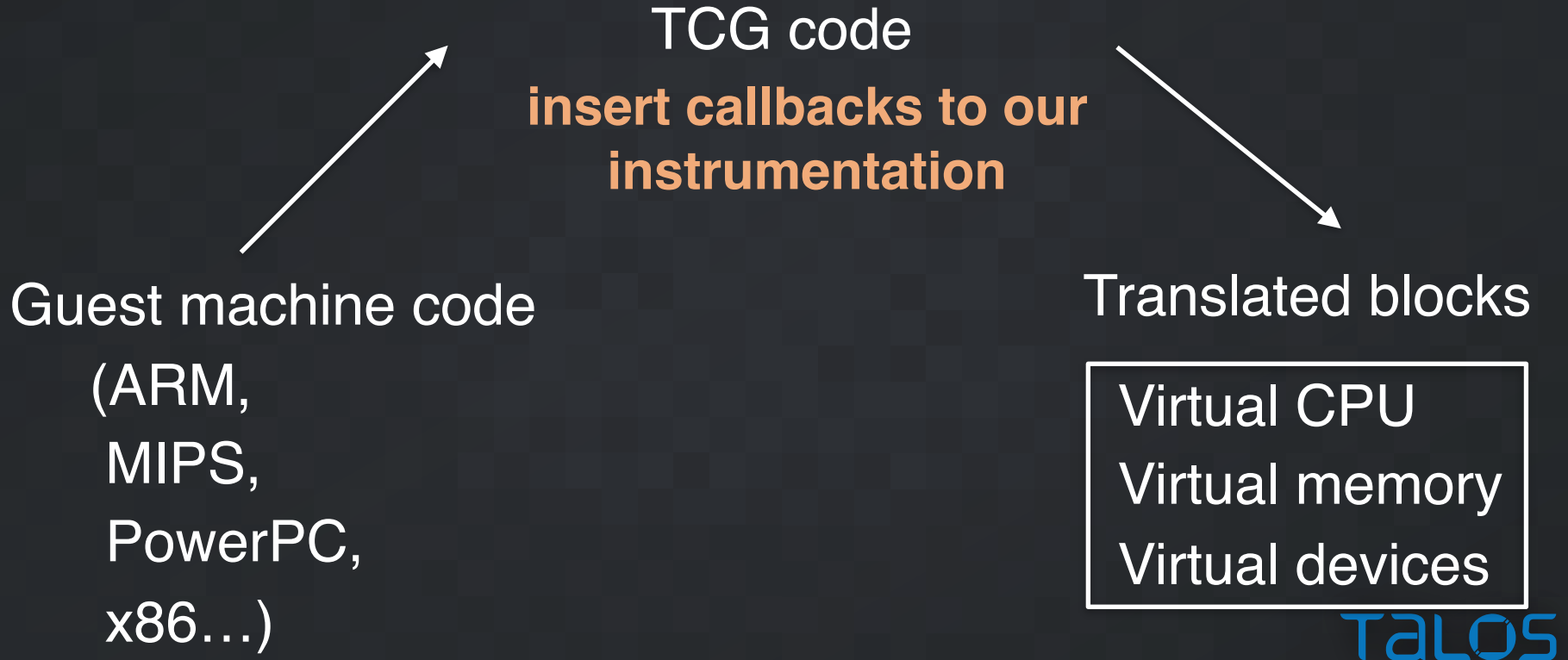
(ARM,
MIPS,
PowerPC,
x86...)

Translated blocks

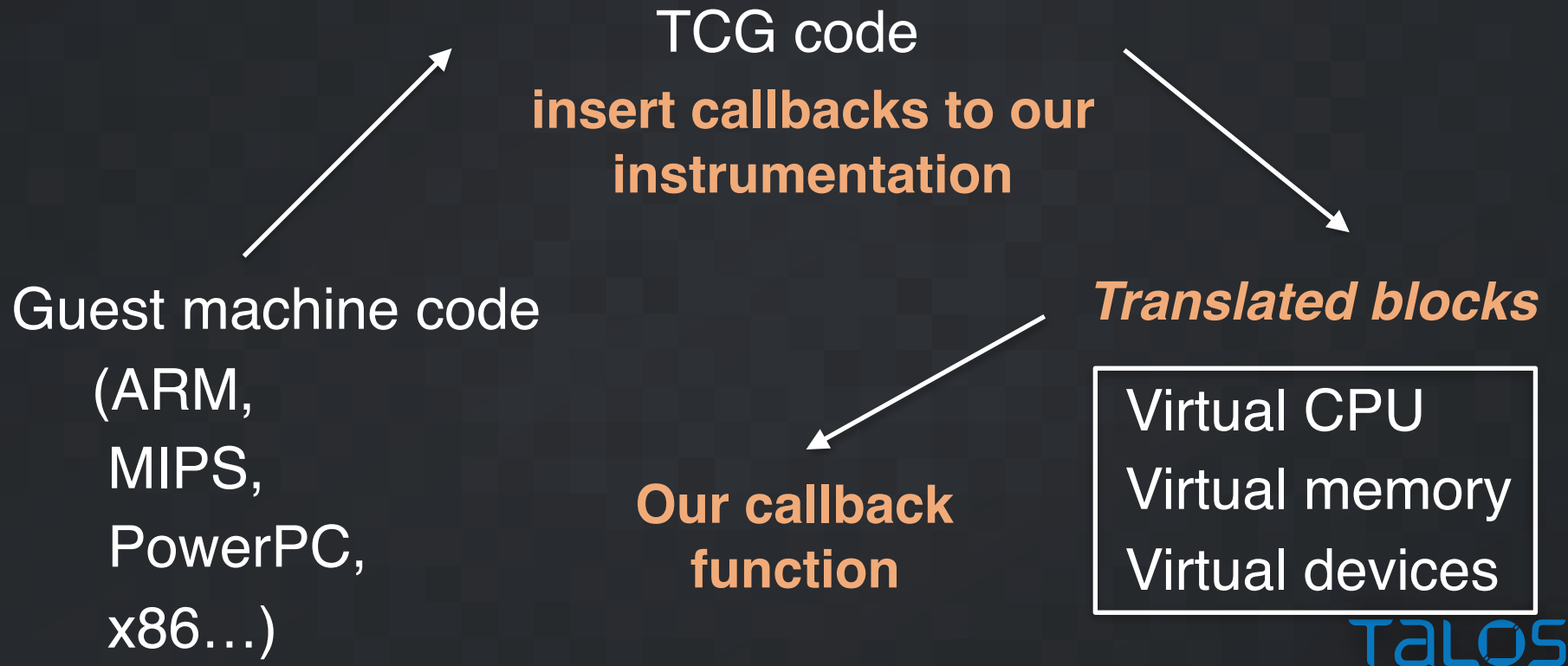
Virtual CPU
Virtual memory
Virtual devices

TALOS

TCG



TCG



QEMU

- ▶ “Transparent” instrumentation
 - Emulated **memory** is **not modified**
 - No agent needed

Some shortcomings...

- ▶ PANDA, DECAF... Why reinvent the wheel?
 - Plugins are coded in C/C++
 - I prefer **python**!
 - Faster development
 - Great libraries
- ▶ Complex QEMU modifications
 - Risk of not updating frequently
 - QEMU evolves, vulnerabilities get fixed...

Intel VT

What about **hardware assisted virtualization**?

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

So, what does PyREBox offer?

▶ **IPython shell**

- ▶ Inspect the system (memory/registers)
- ▶ Set breakpoints...
- ▶ In a nutshell: interactive analysis

▶ **Scripting** (python)

- ▶ Callbacks on events (execution, memory, o.s. events...)
- ▶ Define new commands

Scripting

- ▶ Loaded or unloaded at any moment
- ▶ Callbacks (on demand, dynamically)
 - ▶ Instruction/block begin/end
 - ▶ Memory read/write
 - ▶ Specific opcode execution
 - ▶ Process create/remove
 - ▶ Module load/unload
 - ▶ TLB flush / context change

Scripting

- ▶ Can start a shell at any time

```
>start_shell()
```

- ▶ Can read/write registers, memory
- ▶ Can set breakpoints
- ▶ Use any Python library!

Agent, for automation

- ▶ **File transfer** and **execution**
- ▶ Communication 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)`

Compatibility, documentation...

- ▶ Compiles and runs (tested):
 - ▶ Linux
 - ▶ Windows (thanks to linux subsystem)
 - ▶ Docker is supported
- ▶ Supports Windows and Linux guests
 - ▶ 32 and 64 bit (intel)
- ▶ Example scripts provided
- ▶ Complete PyREBox documentation

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


Updated

- ▶ Updated regularly
 - ▶ Currently, latest stable QEMU version
- ▶ It is free!! (as in freedom)
- ▶ <https://github.com/Cisco-Talos/pyrebox>


General Public License

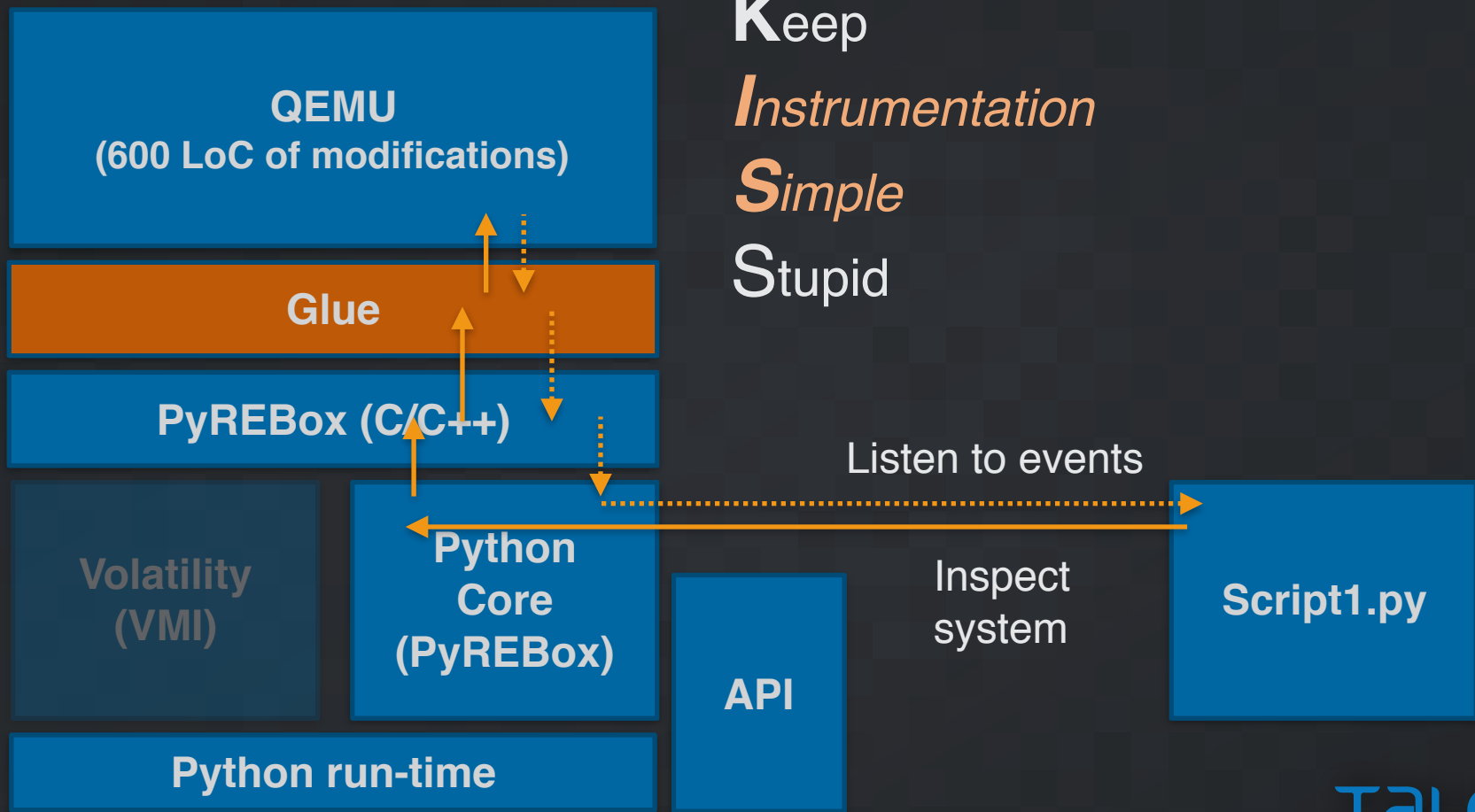
:- (

- ▶ No support for...
 - ▶ Taint analysis (PANDA, DECAF)
 - ▶ Record & replay (PANDA)
 - ▶ Other architectures (ARM, MIPS...)
- ▶ But it will, hopefully, in the future



Design

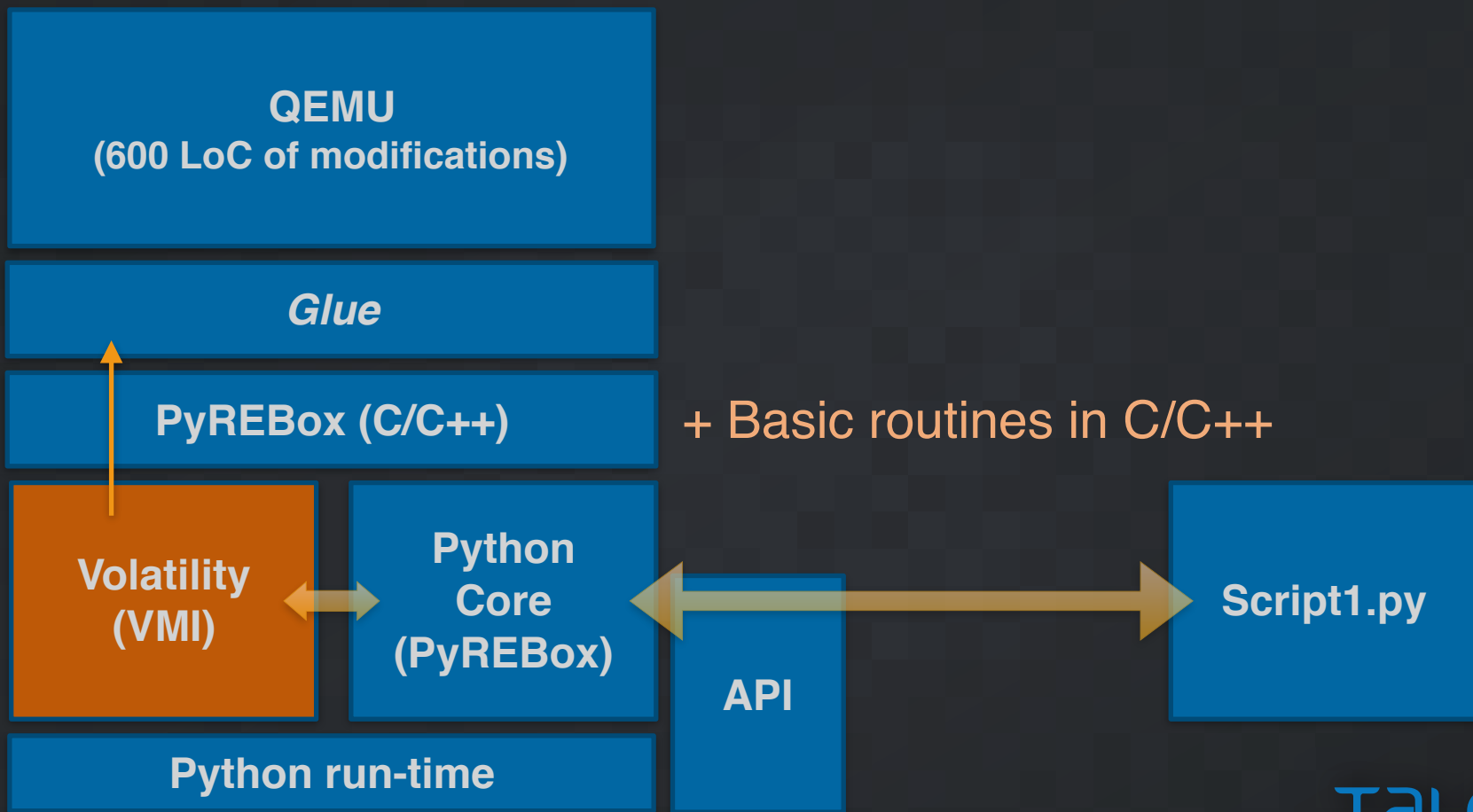




Keep
Instrumentation
Simple
Stupid

VMI

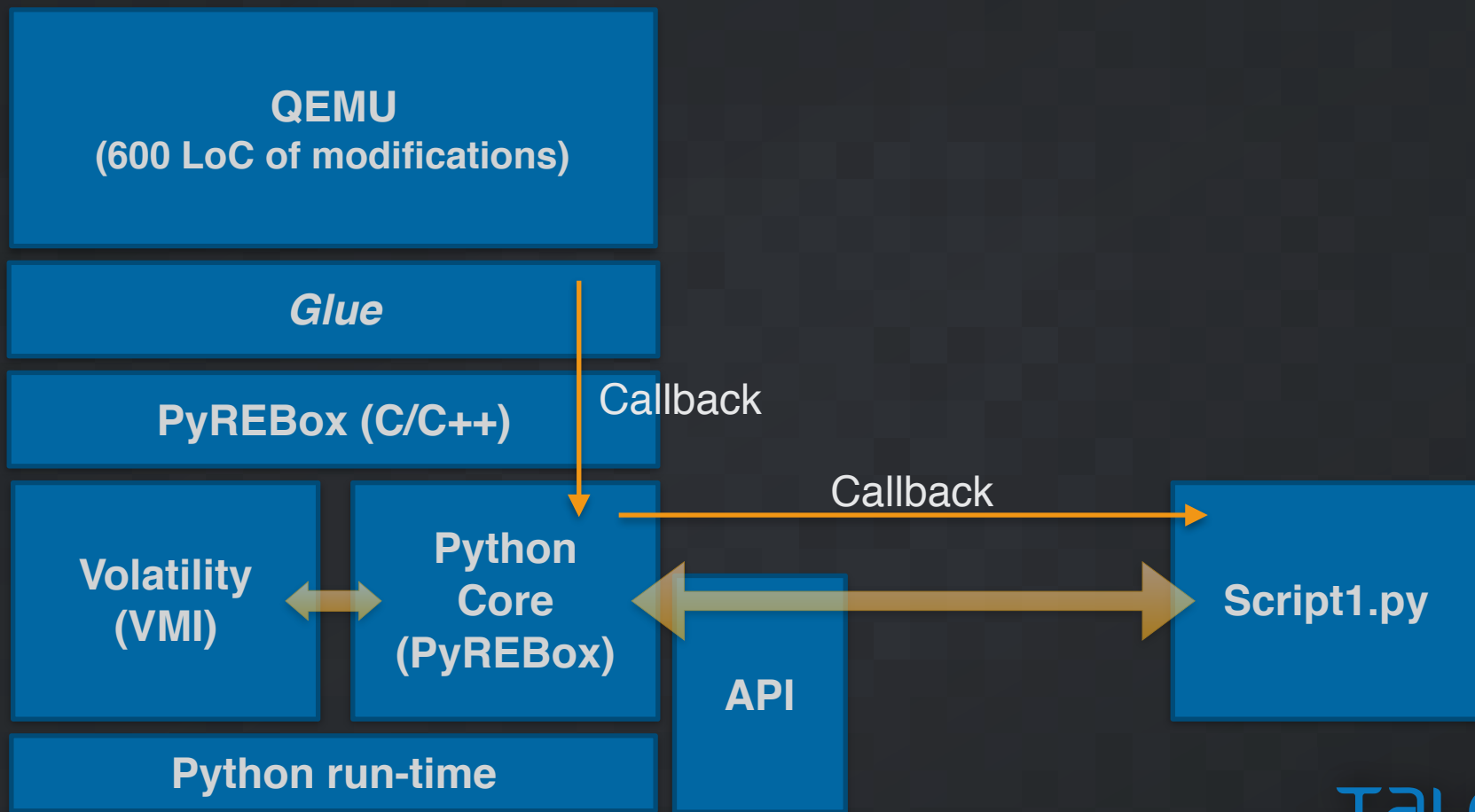
- ▶ We see the system as a *raw* CPU!!
- ▶ Only memory, registers, devices
- ▶ Sequence of instructions
- ▶ Processes, threads, handles, libraries...
 - **Abstractions** of the **O.S.**
- ▶ **Virtual Machine Introspection**
 - Understand these abstractions

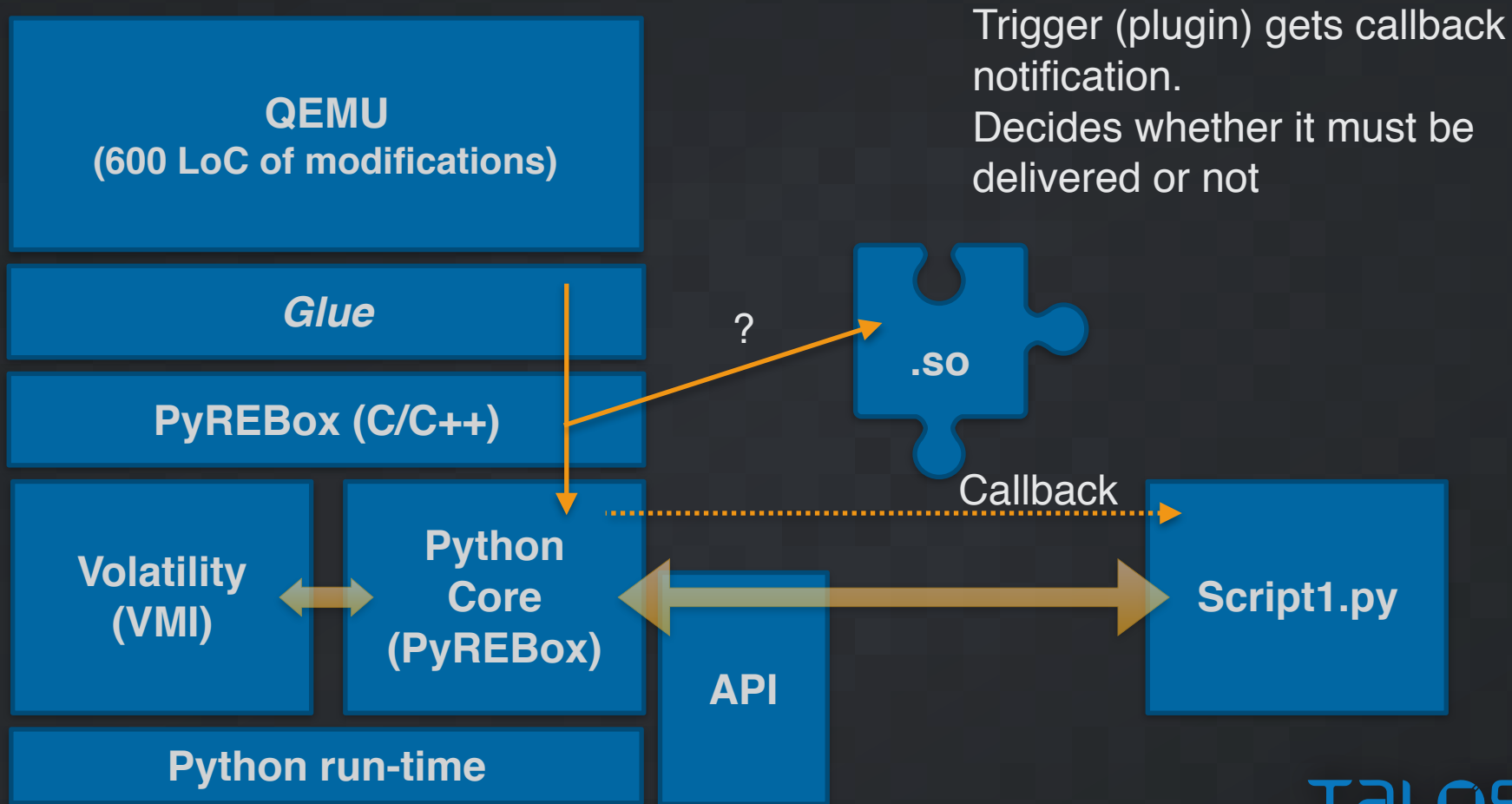



Triggers

- ▶ 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;  
}
```







Demo time!



PyREBox shell


QEMU monitor

- Regular QEMU commands
 - E.g. Attach a USB




PyREBox shell

- **Pauses the guest**
- Inspect / modify
 - Built-in commands
- Run **volatility** commands
- Run **custom** commands
- Run python code (**ipython**)
- Autocompletion, syntax



Use cases



Malware Monitor

- ▶ Set of PyREBox scripts
- ▶ Presented at HITB Amsterdam
- ▶ Sample execution automation, + analysis
 - ▶ **API tracer**
 - Can extract parameters
 - ▶ **Memory dumper**
 - ▶ **Code coverage**
 - ▶ **Memory monitor**
 - ▶ Track injections, droppers, unpacked shellcodes...

Generic Unpacker

- ▶ **Extremely simple generic unpacker**
 - ▶ ~250 LoC script
 - ▶ Heuristics to track W+X at page level
 - ▶ Leverages triggers to reduce overhead
 - ▶ Leverages volatility for memory dump / memory info
 - ▶ Fully automates sample execution

- ▶ Releasing the code today!

Generic Unpacker

- ▶ Simple model
 - ▶ Monitor memory writes and memory execution
 - ▶ Page level

Current layer: 0



Generic Unpacker

- ▶ Simple model
 - ▶ Monitor memory writes and memory execution
 - ▶ Page level

Current layer: 0

W														
X	0	0	0											

Generic Unpacker

- ▶ Simple model
 - ▶ Monitor memory writes and memory execution
 - ▶ Page level

Current layer: 0



Generic Unpacker

- ▶ Simple model
 - ▶ Monitor memory writes and memory execution
 - ▶ Page level

Current layer: 0



Generic Unpacker

- ▶ Simple model
 - ▶ Monitor memory writes and memory execution
 - ▶ Page level

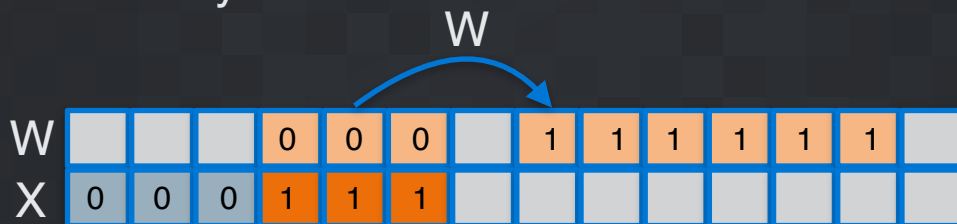
Current layer: 0

W				0	0	0									
X	0	0	0	1	1	1									

Generic Unpacker

- ▶ Simple model
 - ▶ Monitor memory writes and memory execution
 - ▶ Page level

Current layer: 0



Generic Unpacker

- ▶ Simple model
 - ▶ Monitor memory writes and memory execution
 - ▶ Page level

Current layer: 0

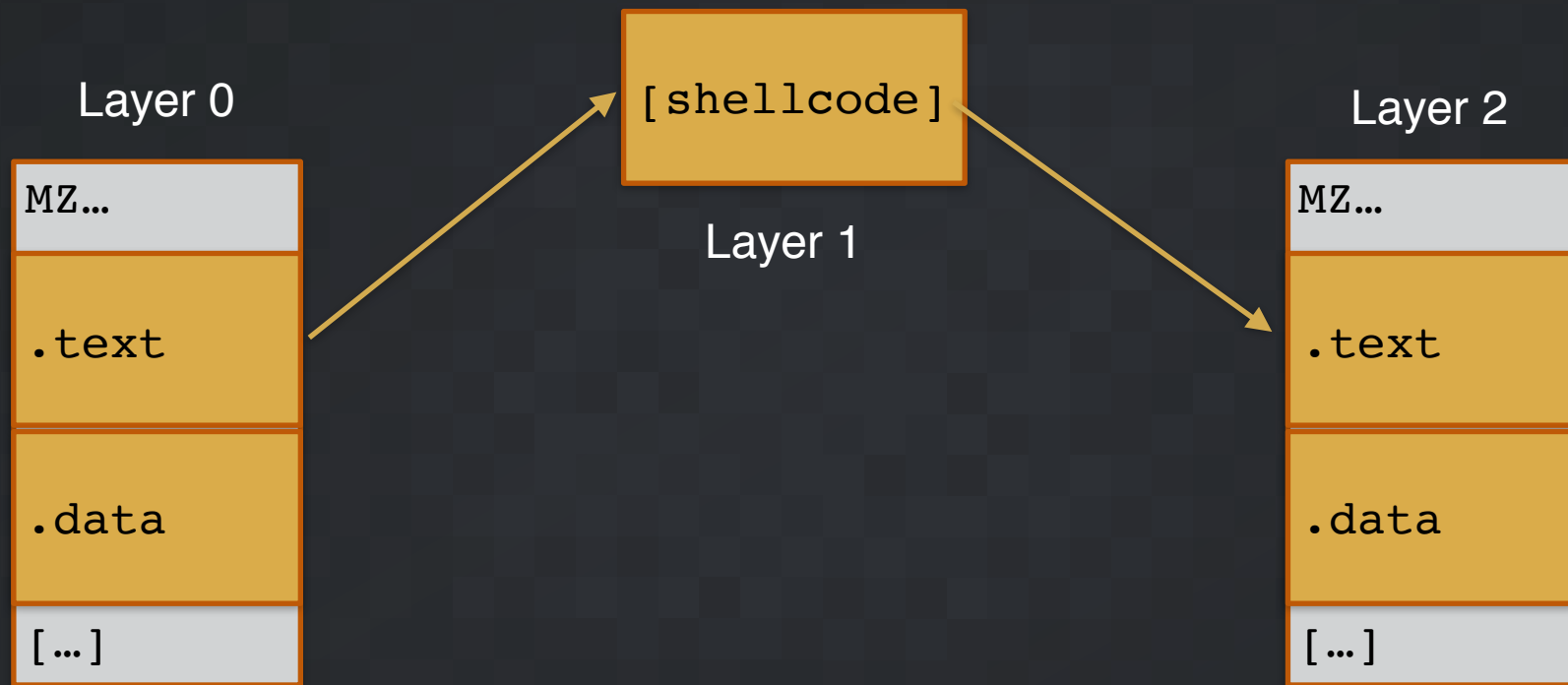
W				0	0	0		1	1	1	1	1	1	
X	0	0	0	1	1	1		2	2	2	2	2	2	



X

DUMP HERE!

Demo: Dridex



Exploit analysis helpers

- ▶ (Demo 2)
- ▶ **Shadow stack**
 - ▶ Detection of stack overflows
- ▶ (Demo 3)
- ▶ **Stack pivoting detector**
 - ▶ For instance: ROP chain on Heap
- ▶ **Shellcode detector**
 - ▶ Code being executed outside module address space
 - ▶ Heap, Stack...

Shadow stack

- ▶ **Monitor** all **CALL** instructions
 - ▶ Keep track of return addresses (push to shadow stack)
- ▶ **Monitor** all **RET** instructions
 - ▶ Check if return address is in the shadow stack
- ▶ If a return address is not a return point:
 - ▶ **Stop execution**, start a shell:
 - ▶ Shellcode?
 - ▶ ROP chain / return to libc?

Exploit
MS WORD
LIKE IN THE 90S

Demo: MS Word 2016

- ▶ Microsoft Word (Equation Editor) **CVE-2017-11882**
 - ▶ Stack based buffer overflow
 - ▶ 32 bit process, no ASLR, no stack protection!
 - ▶ Trivial to exploit

Stack pivoting detector

- ▶ Monitor **modifications** to **ESP/RSP**
 - ▶ If ESP/RSP shifted $> X$ bytes
 - ▶ Check if ESP/RSP points outside stack
 - ▶ ROP chain should be there
 - ▶ Need to consider:
 - ▶ Each thread has a stack
 - ▶ User mode \leftrightarrow Kernel mode

Shellcode detector


- ▶ Monitor **modifications** to **EIP**
 - ▶ If EIP/RIP shifted $> X$ bytes
 - ▶ Check if EIP points outside of a module
 - ▶ Shellcode *may* be there
 - ▶ FP prone
 - ▶ Build a whitelist per application?

Demo - Foxit Reader


- ▶ Foxit Reader 7.1.5 (No CVE?)
 - ▶ Reported by Sascha Schirra in 2015
 - ▶ PoC on exploit-db
- ▶ PNG parsing vulnerability
 - ▶ PNG to PDF conversion
- ▶ Heap buffer overflow
- ▶ Partial overwrite of pointer to object

Demo - Foxit Reader

- ▶ Foxit Reader 7.1.5
 - ▶ (1) Overwrite 2 bytes on object pointer
 - ▶ (2) Object is dereferenced, vtable is dereferenced, function is called, we have control!
 - ▶ (3) JOP gadget to do stack pivot to HEAP
 - ▶ (4) ROP chain on HEAP (controlled buffer)
 - ▶ Disables DEP
 - ▶ Jumps into shellcode
 - ▶ (5) Shellcode



Whats next?



What's next?

- ▶ Support for additional architectures (ARM / MIPS)
- ▶ Debugging backend for r2 / IDA
- ▶ R2 as a disassembler inside PyREBox
- ▶ Support for other backends (PANDA?)



Questions?



TALOS™

talosintelligence.com

blog.talosintel.com

@talossecurity

