CERT.PL
NASK

DRAKVUF Sandbox

Dynamic malware analysis from the **hypervisor** point of view

Paweł Srokosz

CyberChess 2023
#CyberShock session

cert.pl

Agenda

- ➔ 101: Automated malware unpacking 🎁
- ➔ Hypervisor-level sandboxing: is it making a difference?🤔
- ➔ DRAKVUF Sandbox: our contribution to the DRAKVUF project! 🚀

Introduction

to automated malware unpacking


PARACETAMOL-
GRINDEKS 500 mg
tabletes / tabletid
Paracetamolum
10
Grindex


Gelocatil
697021.8
Gelocatil
500mg granulado
paracetamol
Ferrer Internacional, S.A.
Gelocatil
500mg granulado
paracetamol
Vía oral
Alivio sintomático del dolor ocasional leve o moderado, como dolor de cabeza, dental, muscular (contracturas) o de espalda (lumbago) y la fiebre en adultos y adolescentes de más de 44 kg (a partir de 12 años).
Vía oral | 12 sobres
ferrer


661267.5
apiretal® 100 mg/ml
solución oral
paracetamol
90 ml

Paracetamol 500mg (just in different packaging)

Modern malware also have layers

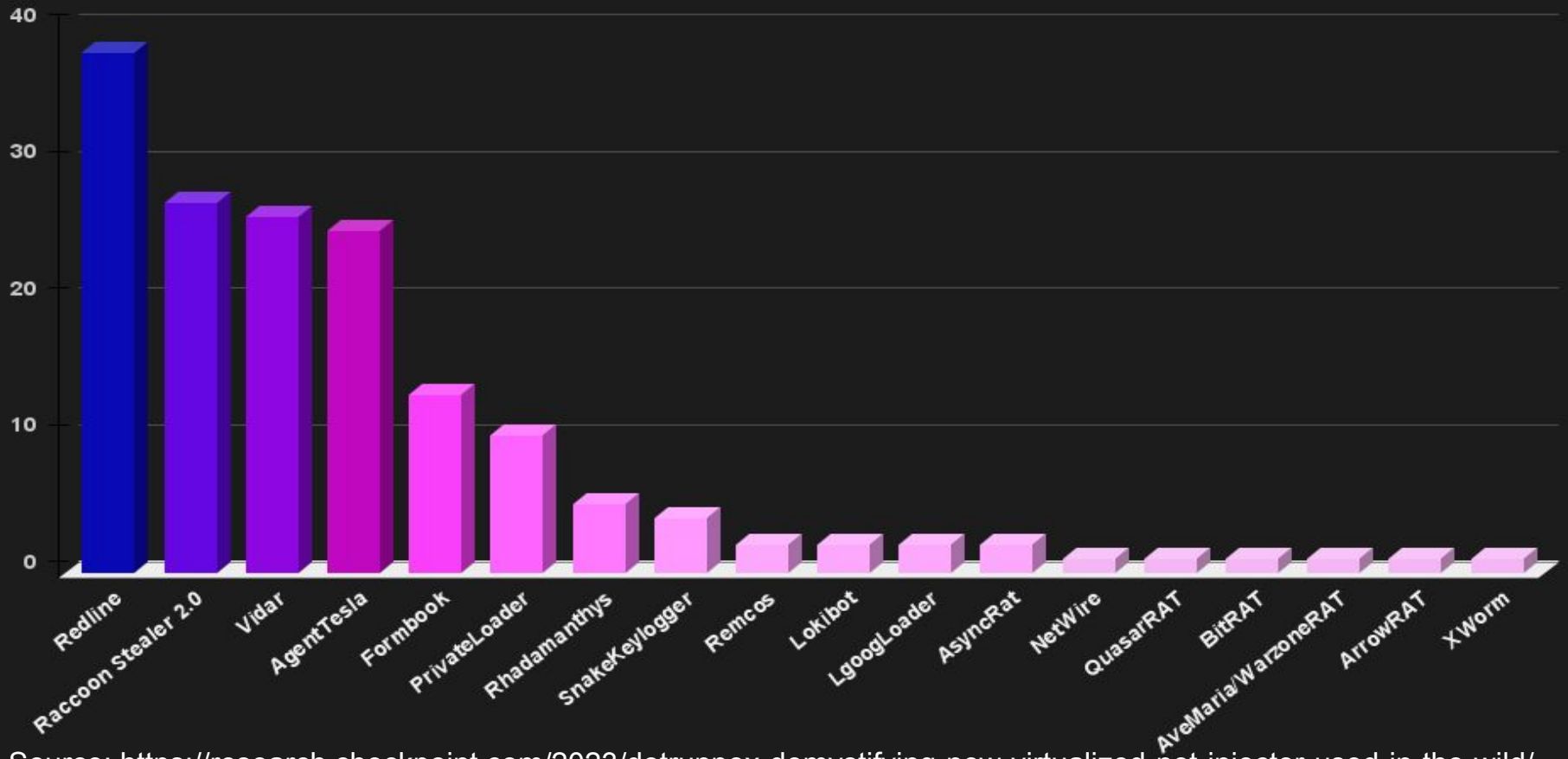
➔ Malware is usually wrapped with additional layers that also helps in “ingestion”:

- Usually highly obfuscated and difficult for static analysis;
- Bamboozling AV solutions, checking environment, providing additional randomness;
- Variety of stuff that is usually not that interesting, if we want to identify the actual, final payload;

Protector example: DotRunpeX

- ➔ .NET Protector
- ➔ Obfuscated by KoiVM virtualizer
- ➔ Uses LOLDrivers to bypass AV (procexp.sys, zemana.sys)
- ➔ Uses Process Hollowing for injection
- ➔ Older versions remapped ntdll.dll to bypass hooking

Malware Families Delivered by DotRunpeX



Source: https://research.checkpoint.com/2023/dotrunpex-demystifying-new-virtualized-net-injector-used-in-the-wild/

The idea of dynamic malware unpacking

➔ Let it execute and unpack itself

2.exe

The idea of dynamic malware unpacking

➔ Let it execute and unpack itself

DotRunpeX

The idea of dynamic malware unpacking

➔ Let it execute and unpack itself

➔ Trace the execution and make memory dumps of next layers until we get an unpacked sample

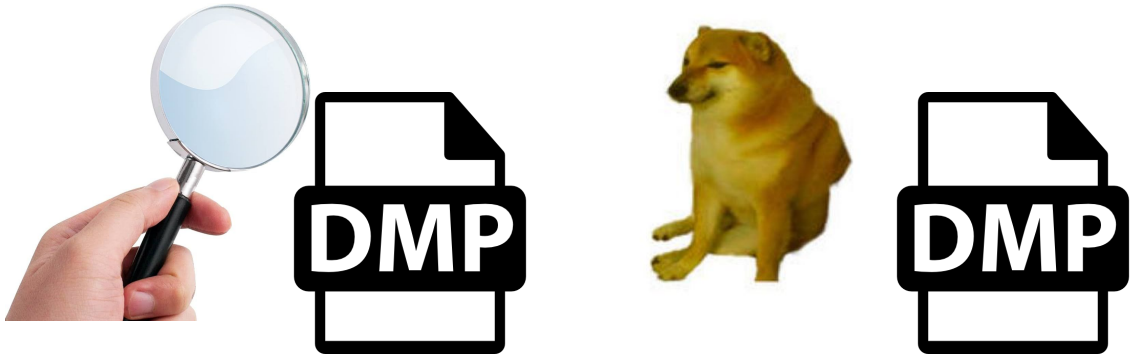
DotRunpeX

Agent Tesla

The idea of dynamic malware unpacking

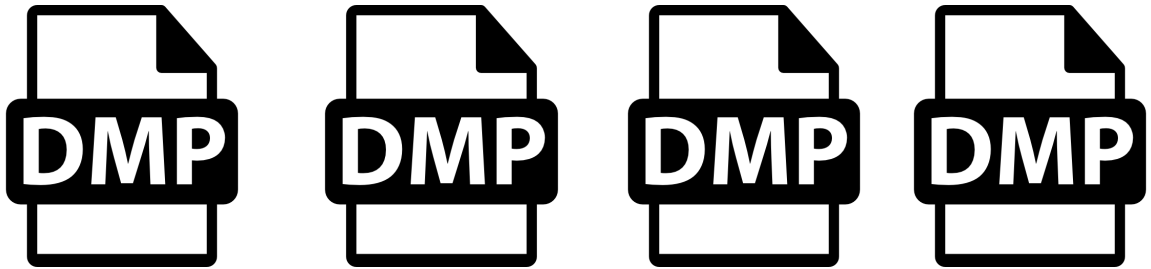
- ➔ Let it execute and unpack itself
- ➔ Trace the execution and make memory dumps of next layers until we get an unpacked sample
- ➔ Analyze dumps statically (using YARA rules, dedicated scripts etc.)

Agent Tesla



The idea of dynamic malware unpacking

But how to dump these stages?



The idea of dynamic malware unpacking

But how to dump these stages?

➔ **Randomly make a dump of whole process memory?**
Not very reliable and we get lots of useless data to process

The idea of dynamic malware unpacking

But how to dump these stages?

- ➔ **Randomly make a dump of whole process memory?**
  Not very reliable and we get lots of useless data to process
- ➔ **Scan memory with YARA rules and dump matching things?**
  Very often we need generic approach to dump the “unknown”

The idea of dynamic malware unpacking

But how to dump these stages?

- ➔ **Randomly make a dump of whole process memory?**
  Not very reliable and we get lots of useless data to process
- ➔ **Scan memory with YARA rules and dump matching things?**
  Very often we need generic approach to dump the "unknown"
- ➔ **Dump only "interesting" memory e.g. with RWX rights?**

The idea of dynamic malware unpacking

But how to dump these stages?

- ➔ **Randomly make a dump of whole process memory?**
  Not very reliable and we get lots of useless data to process
- ➔ **Scan memory with YARA rules and dump matching things?**
  Very often we need generic approach to dump the "unknown"
- ➔ **Dump only "interesting" memory e.g. with RWX rights?**
  Actually that works best!

The idea of dynamic malware unpacking

drakvuf / src / plugins / memdump / **memdump.cpp**

manorit2001 happy new year 2023 (#1582) ✓ 9 months ago

```
if (!c->memdump_disable_free_vm)
    if (!register_trap(nullptr, free_virtual_memory_hook_cb,    bp.for_syscall_name("NtFreeVirtualMemory")))
        throw -1;
if (!c->memdump_disable_protect_vm)
    if (!register_trap(nullptr, protect_virtual_memory_hook_cb, bp.for_syscall_name("NtProtectVirtualMemory")))
        throw -1;
if (!c->memdump_disable_terminate_proc)
    if (!register_trap(nullptr, terminate_process_hook_cb,      bp.for_syscall_name("NtTerminateProcess")))
        throw -1;
if (!c->memdump_disable_write_vm)
    if (!register_trap(nullptr, write_virtual_memory_hook_cb,   bp.for_syscall_name("NtWriteVirtualMemory")))
        throw -1;
if (!c->memdump_disable_create_thread)
    if (!register_trap(nullptr, create_remote_thread_hook_cb,   bp.for_syscall_name("NtCreateThreadEx")))
        throw -1;
if (!c->memdump_disable_set_thread && is64bit && json_wow)
    if (!register_trap(nullptr, set_information_thread_hook_cb, bp.for_syscall_name("NtSetInformationThread")))
```

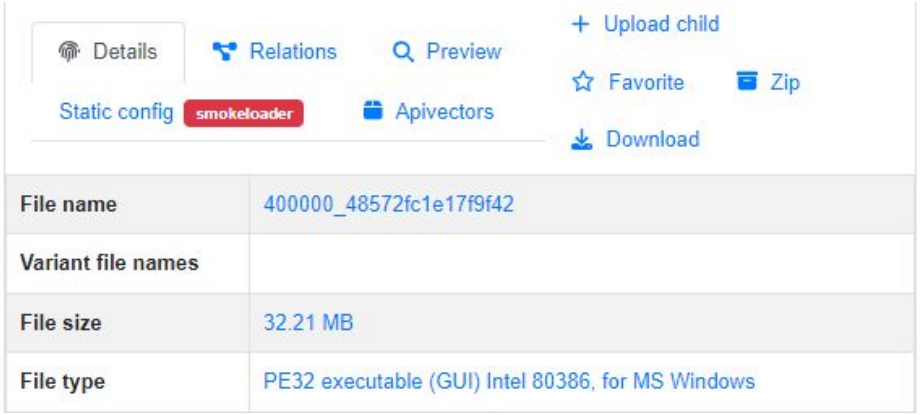
CERT.PL NASK

The idea of dynamic malware unpacking

What is “interesting” memory:

- ➔ Code injected into process memory
  NtWriteProcessMemory
- ➔ New binary mapped on process memory
  NtProtectVirtualMemory on memory with contents starting with “MZ”
- ➔ Injecting new threads or hijacking existing threads in another process
  NtCreateRemoteThreadEx, NtSetInformationThread

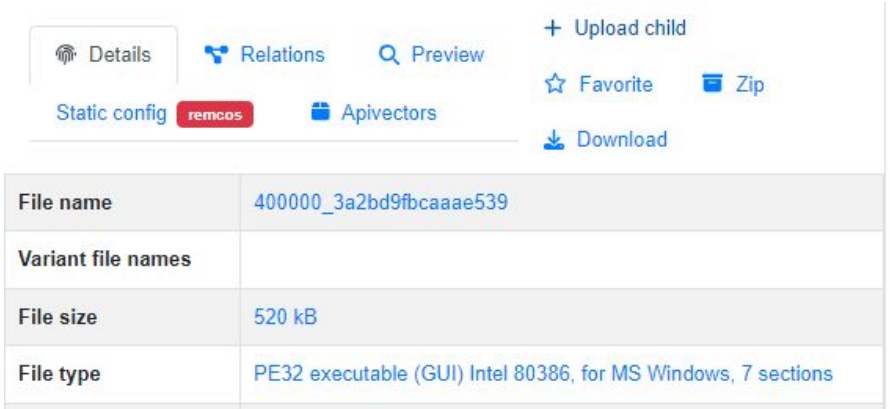
The idea of dynamic malware unpacking




```
{"Plugin": "memdump", "TimeStamp": "                    ", "PID": 1676, "PPID": 1896, "TID": 2232, "U
serName": "SessionID", "UserId": 1, "ProcessName": "                                                  {
                ", "Method": "NtTerminateProcess", "EventUID": "0x104ba8", "DumpReason": "Stack heu
ristic", "DumpPID": 1676, "DumpAddr": "0x400000", "DumpSize": "0x2037000", "DumpFilename": "400000
_48572fc1e17f9f42", "DumpsCount": 24}
```


CERT.PL NASK

The idea of dynamic malware unpacking



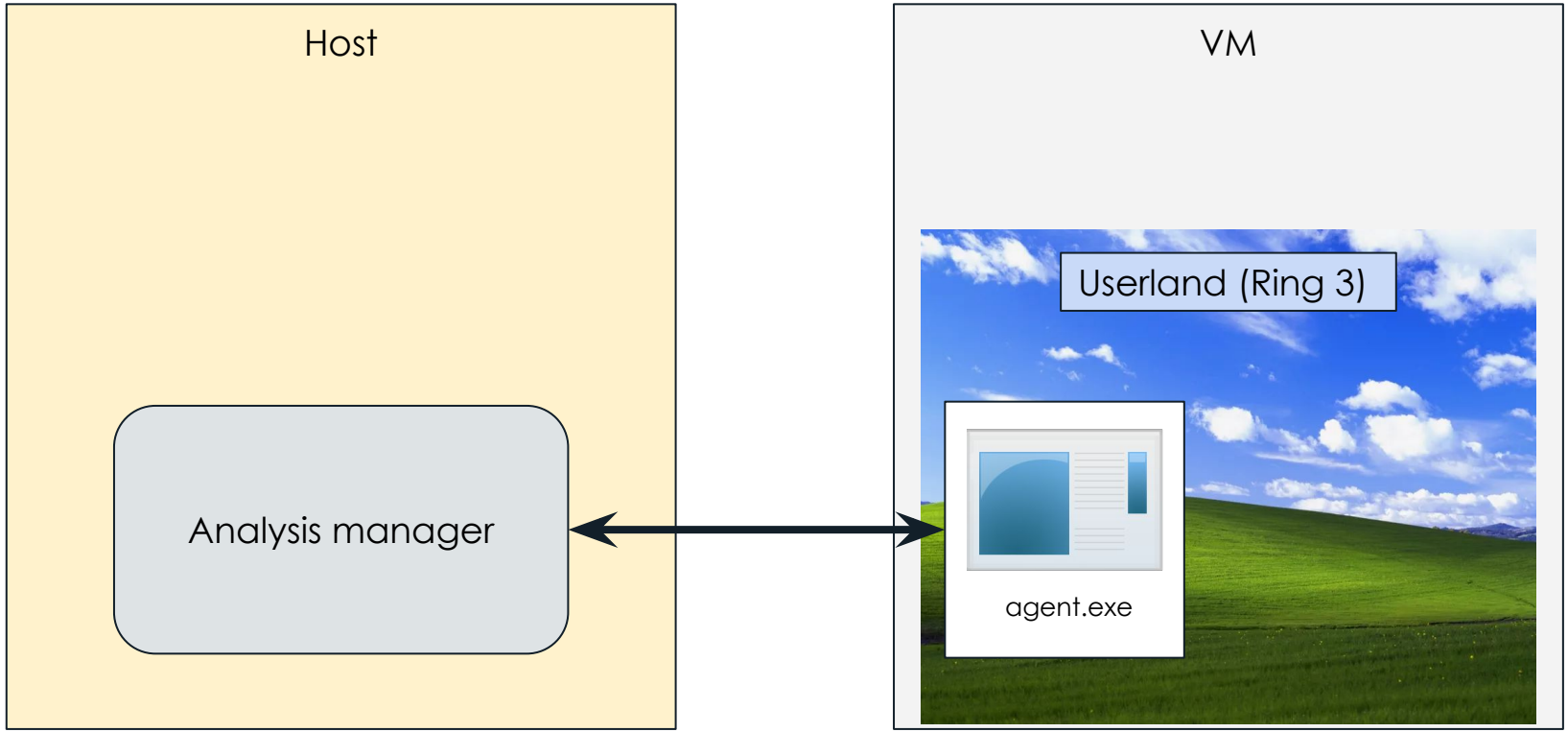

```
{"Plugin": "memdump", "TimeStamp": "[redacted]", "PID": 2196, "PPID": 1896, "TID": 2212, "UserName": "SessionID", "UserId": 1, "ProcessName": "[redacted]", "Method": "CryptAcquireContextA", "EventUID": "0x8eb", "DumpReason": "Stack heuristic", "DumpPID": 2196, "DumpAddr": "0x400000", "DumpSize": "0x82000", "DumpFilename": "400000_3a2bd9fbcaaae539", "DumpsCount": 4}
```

Hypervisor-level sandboxing

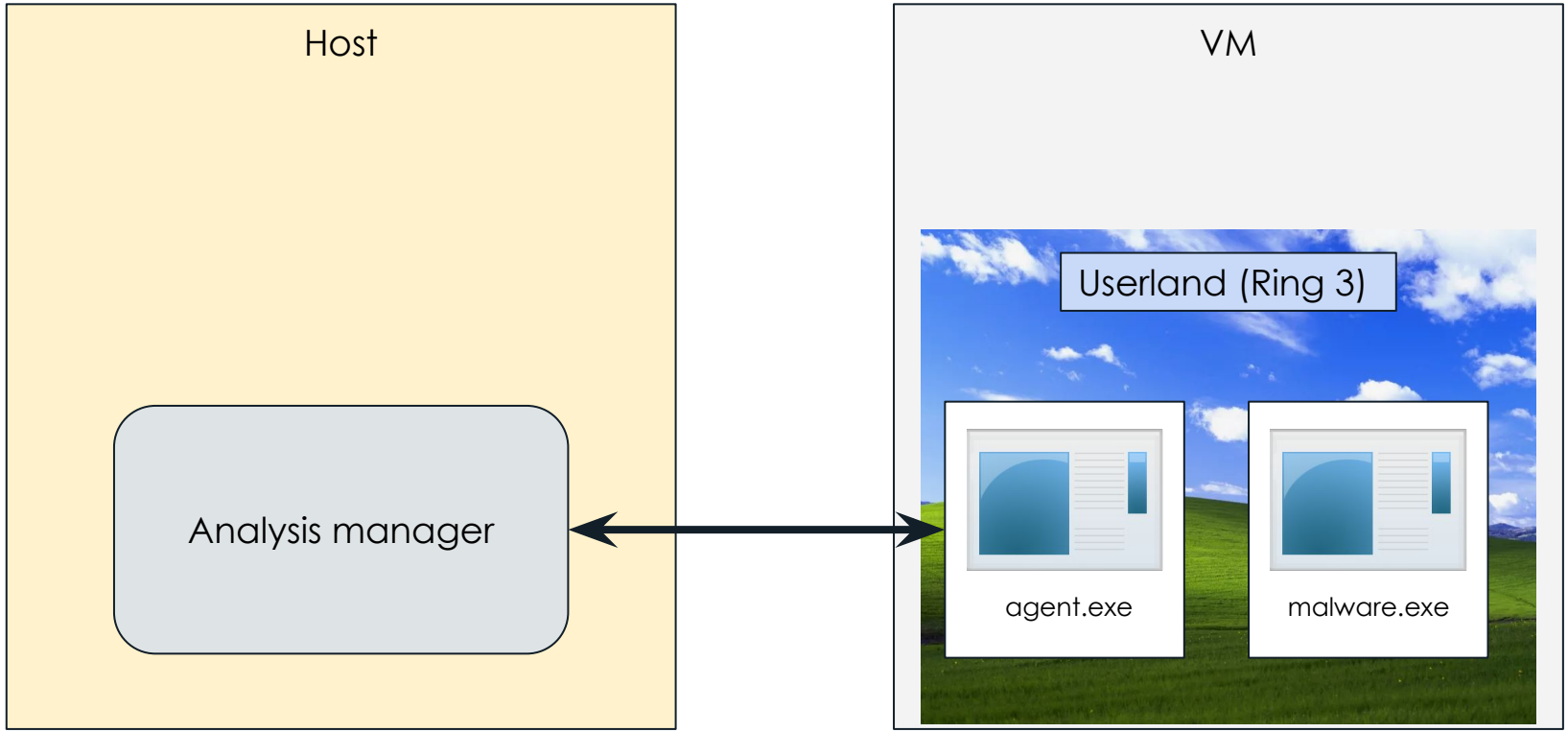
using DRAKVUF

CERT.PL NASK

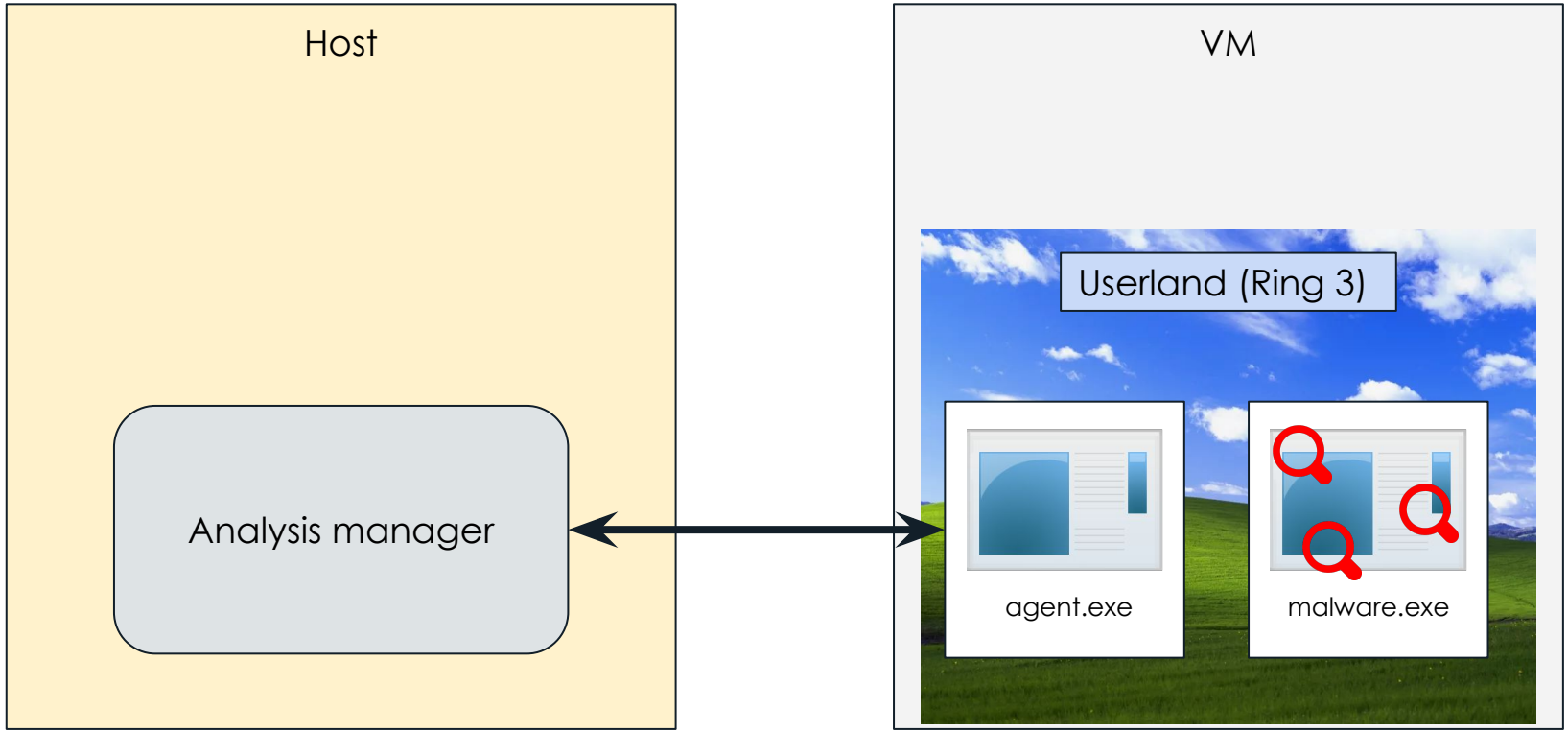
Agent-based sandbox



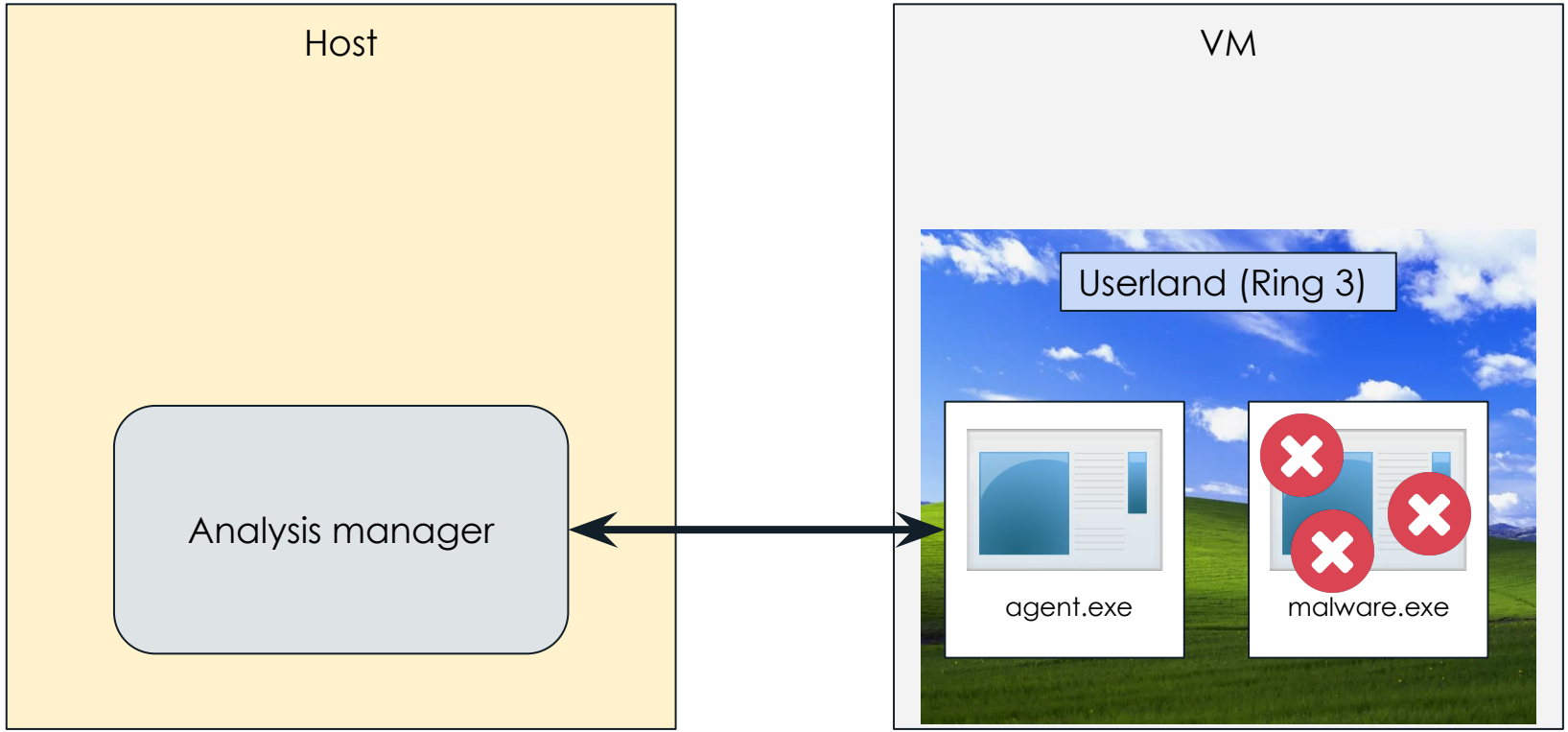
Agent-based sandbox



Agent-based sandbox



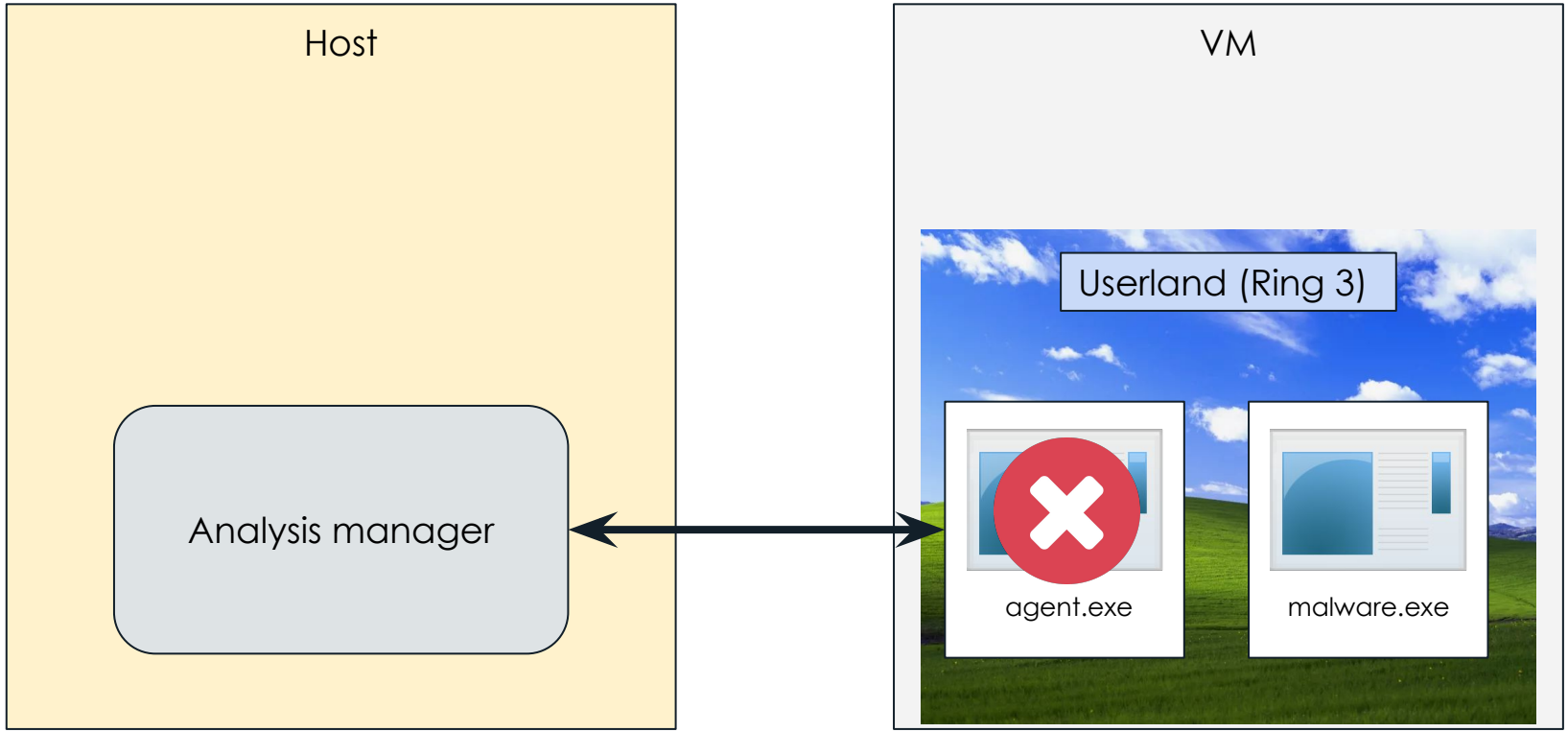
Agent-based sandbox



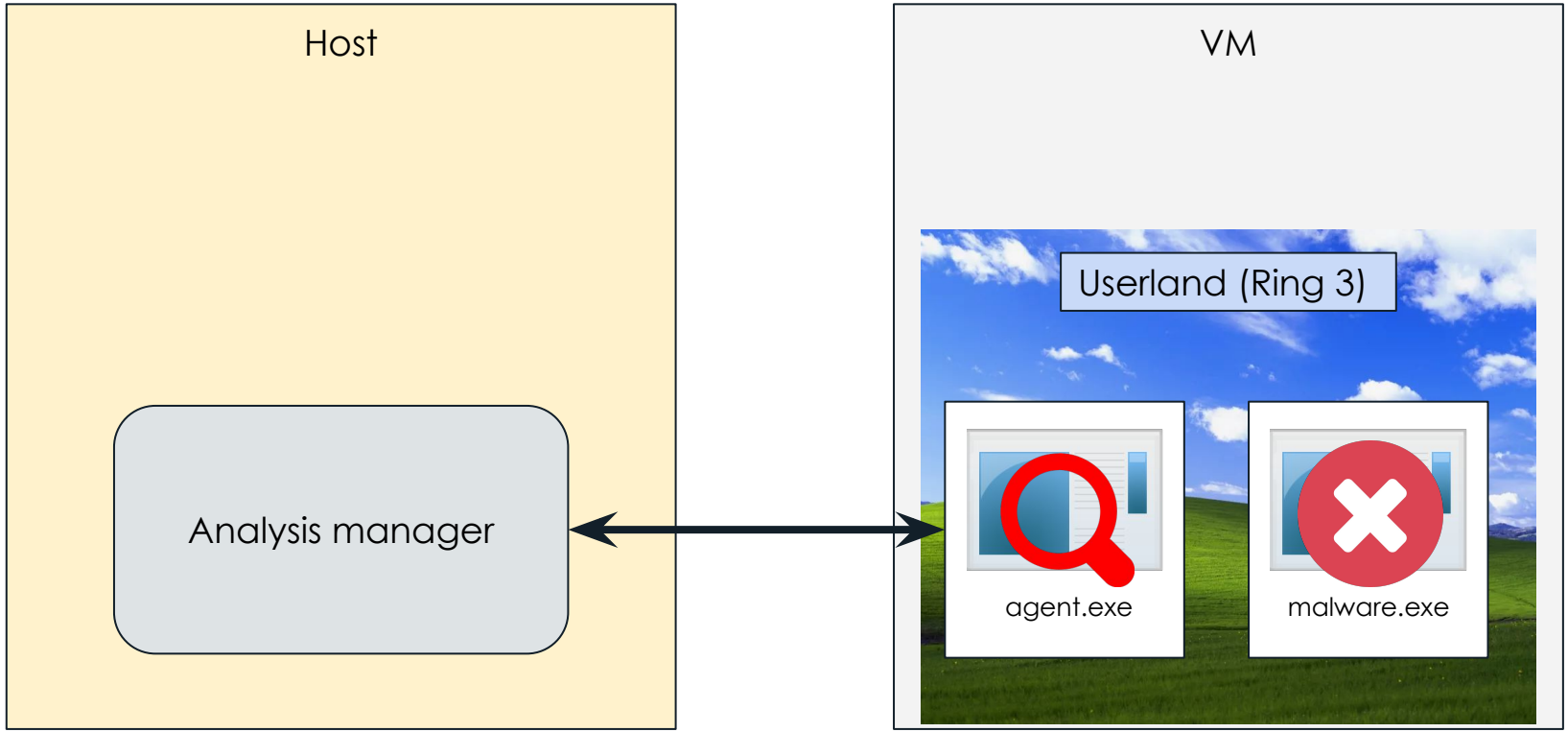
Protector example: DotRunpeX

- ➔ .NET Protector
- ➔ Obfuscated by KoiVM virtualizer
- ➔ Uses LOLDrivers to bypass AV (procexp.sys, zemana.sys)
- ➔ Uses Process Hollowing for injection
- ➔ Older versions remapped ntdll.dll to bypass hooking

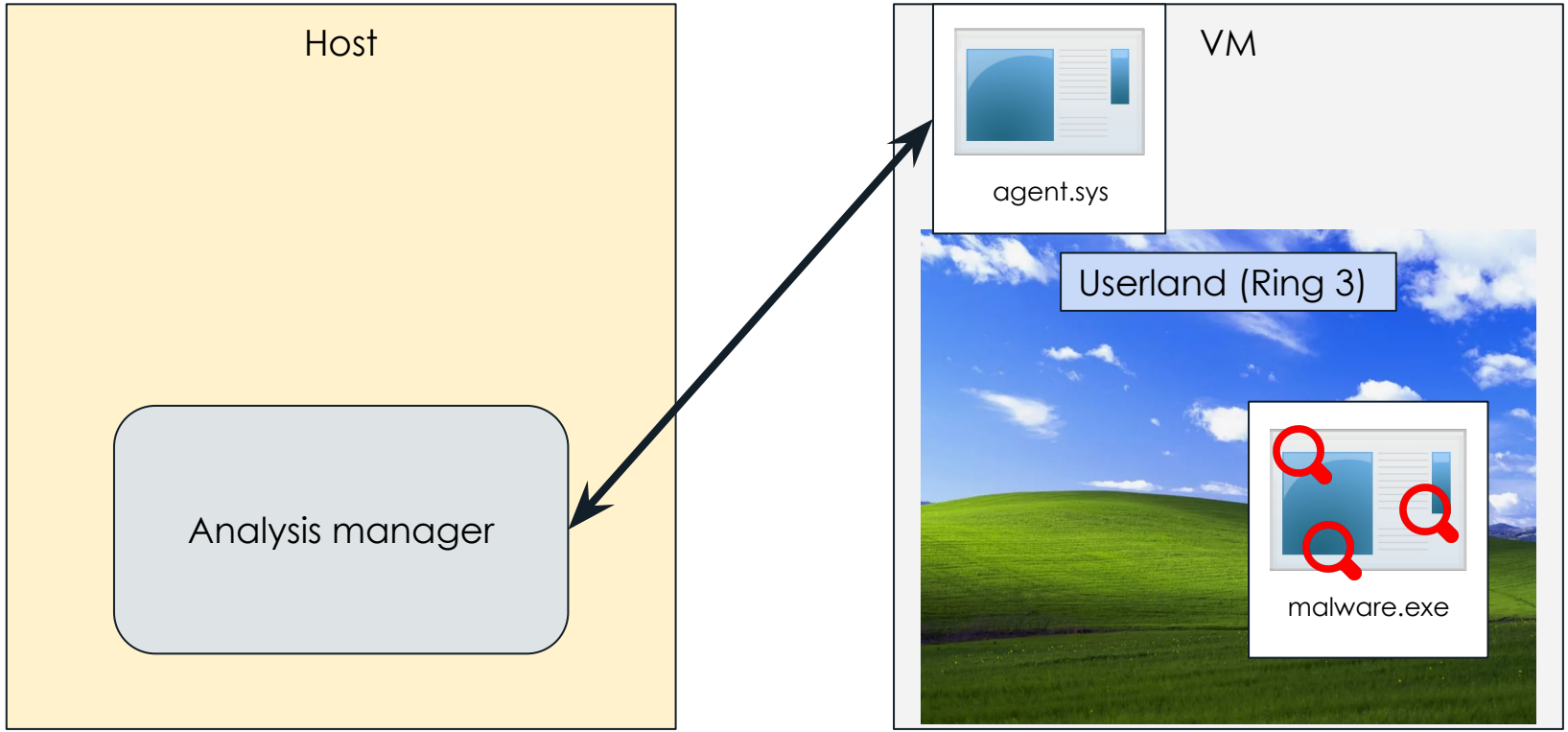
Agent-based sandbox



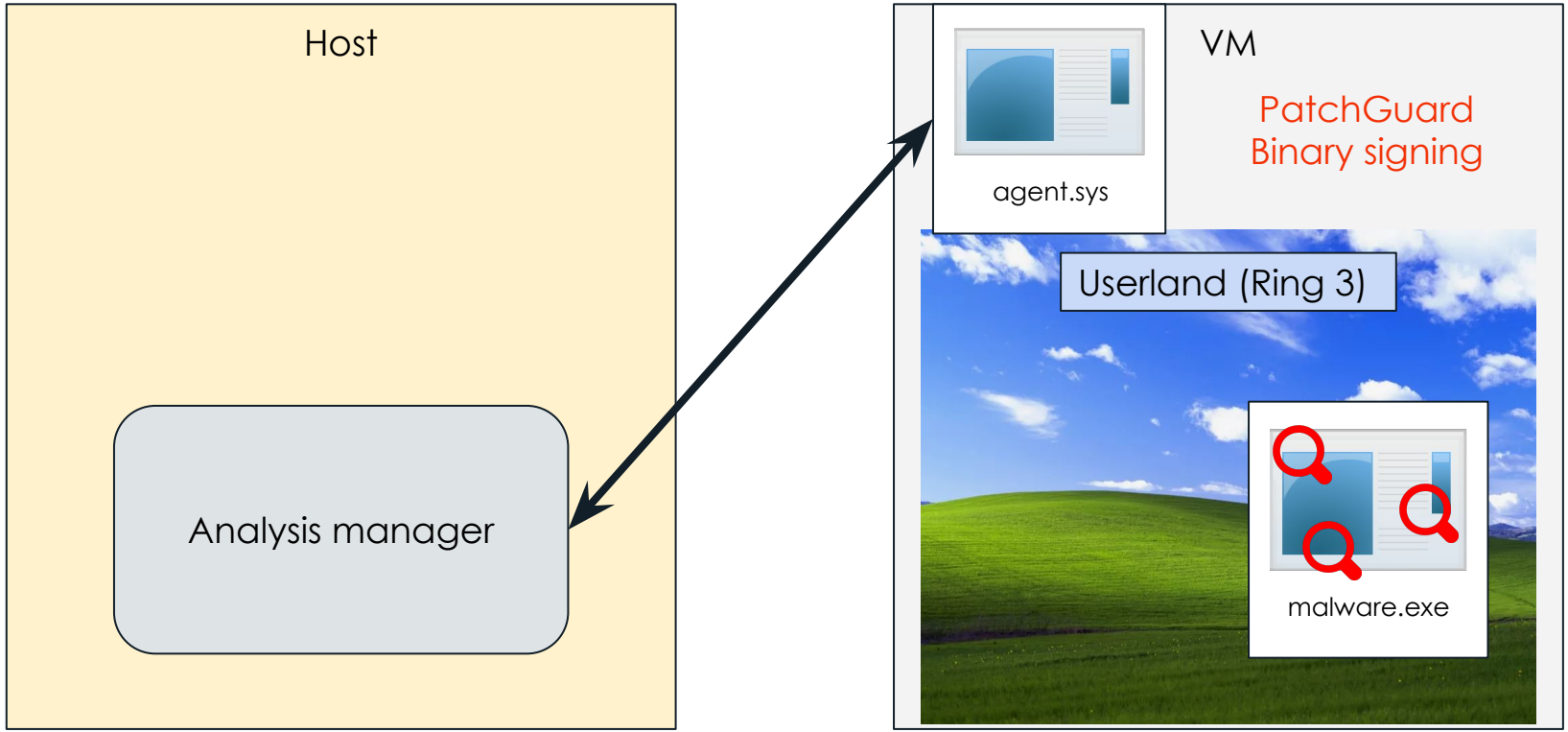
Agent-based sandbox



Agent-based sandbox



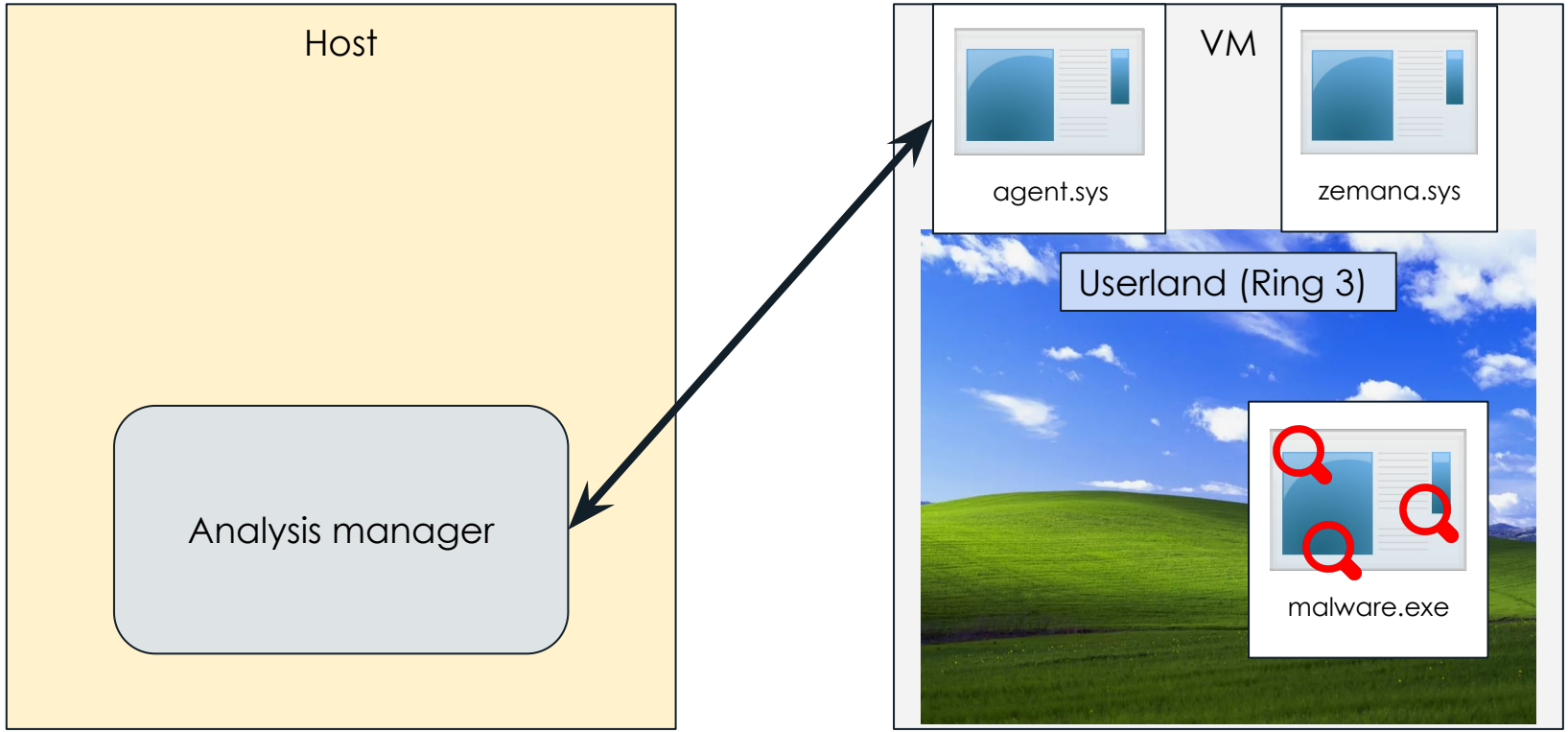
Agent-based sandbox



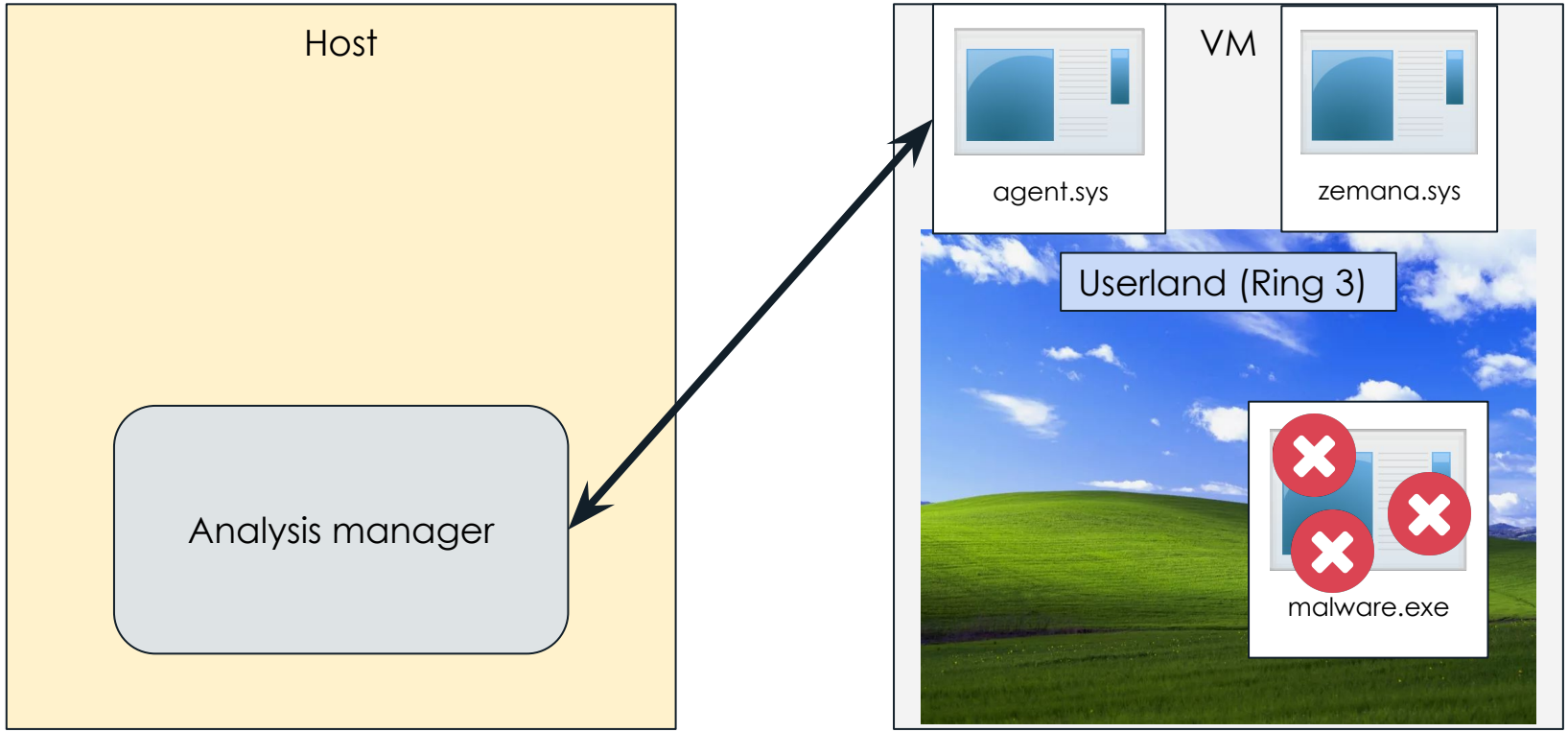
Protector example: DotRunpeX

- ➔ .NET Protector
- ➔ Obfuscated by KoiVM virtualizer
- ➔ Uses LOLDrivers to bypass AV (procexp.sys, zemana.sys)
- ➔ Uses Process Hollowing for injection
- ➔ Older versions remapped ntdll.dll to bypass hooking

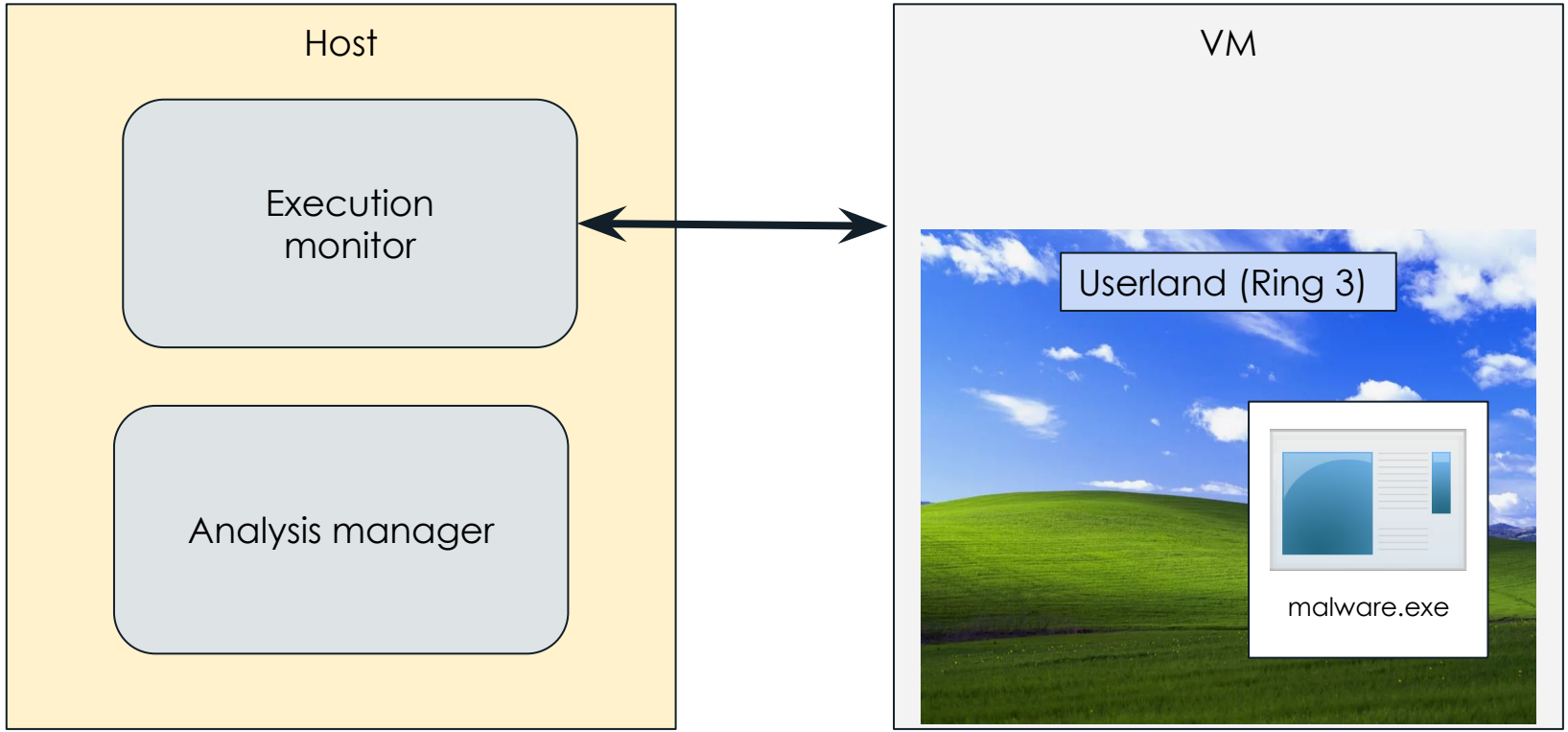
Agent-based sandbox



Agent-based sandbox



Agentless sandbox



Hypervisors are made for malware tracing!

- ➔ By design, they need to emulate/mock things to make virtualization transparent for the guest
- ➔ Hardware virtualization is a security boundary (very often described as **ring -1**)
- ➔ We can trace whole operating system without relying on operating system mechanisms

VT-x: VM exits

23.4 LIFE CYCLE OF VMM SOFTWARE

Figure 23-1 illustrates the life cycle of a VMM and its guest software as well as the interactions between them. The following items summarize that life cycle:

- Software enters VMX operation by executing a VMXON instruction.
- Using VM entries, a VMM can then enter guests into virtual machines (one at a time). The VMM effects a VM entry using instructions VMLAUNCH and VMRESUME; it regains control using VM exits.
- VM exits transfer control to an entry point specified by the VMM. The VMM can take action appropriate to the cause of the VM exit and can then return to the virtual machine using a VM entry.
- Eventually, the VMM may decide to shut itself down and leave VMX operation. It does so by executing the VMXOFF instruction.

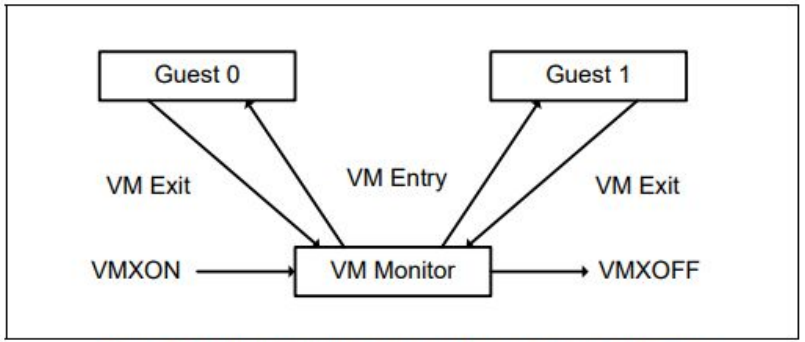


Figure 23-1. Interaction of a Virtual-Machine Monitor and Guests

VT-x: VM exits

Table 24-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 24.4.2).
3	Use TSC offsetting	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC offset field (see Section 24.6.5 and Section 25.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.

VT-x: VM exits

Table 24-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 24.4.2).
3	Use TSC offsetting	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC offset field (see Section 24.6.5 and Section 25.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.

VT-x: VM exits

Context switching

Single-stepping in VM

Table 24-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 24.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 25.1.3. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits.
21	Use TPR shadow	Setting this control to 1 enables TPR virtualization and other APIC-virtualization features. See Chapter 29.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 24.4.2).
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 24.6.4 and Section 25.1.3). For this control, "0" means "do not use I/O bitmaps" and "1" means "use I/O bitmaps." If the I/O bitmaps are used, the setting of the "unconditional I/O exiting" control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 25.5.2.

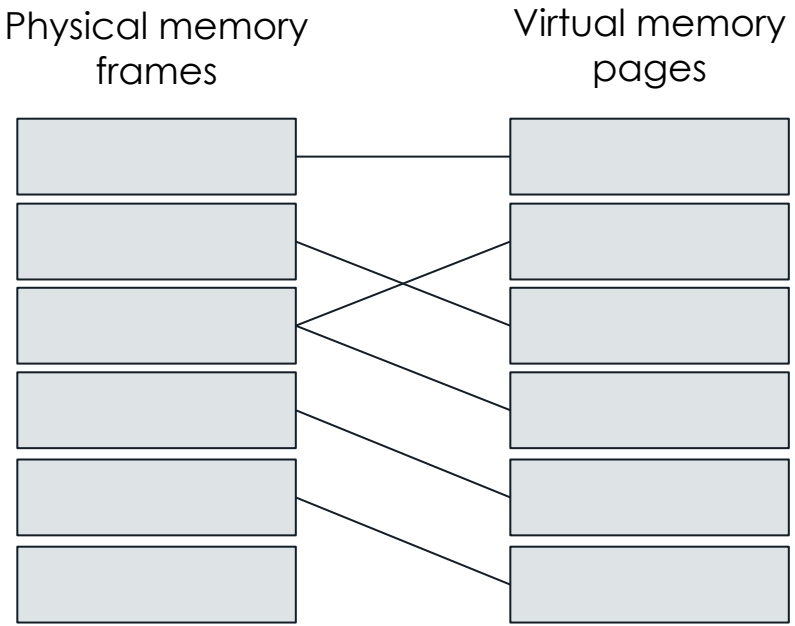
VT-x: VM exits and interrupts

VM exits can be also caused by:

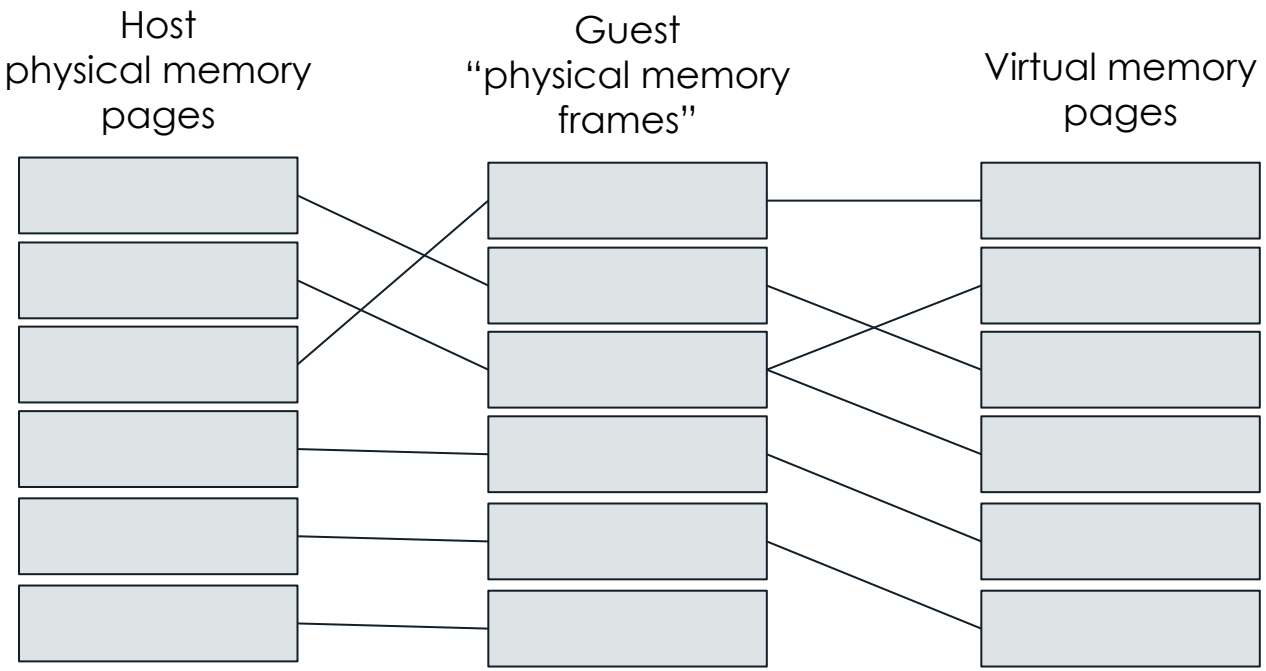
➔ Exception interrupts (including page faults, protection faults or classic **int3** software breakpoint)

➔ CPUID, RDRAND…

VT-x: Extended Page Tables



VT-x: Extended Page Tables

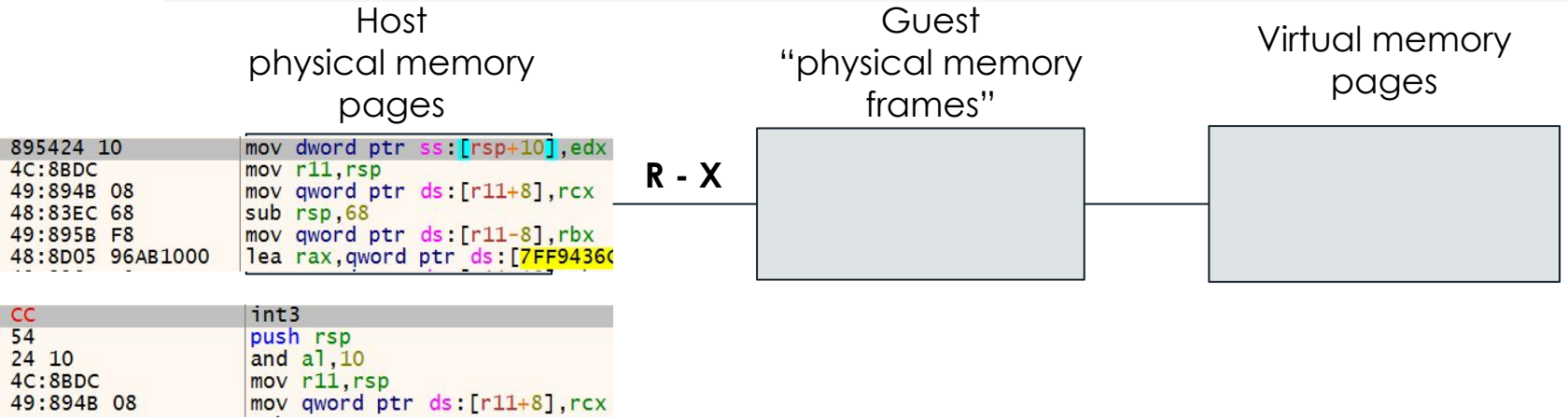


SLAT (Second-Level Address Translation)

VT-x: Extended Page Tables - altp2m

STEALTHY MONITORING WITH XEN ALTP2M

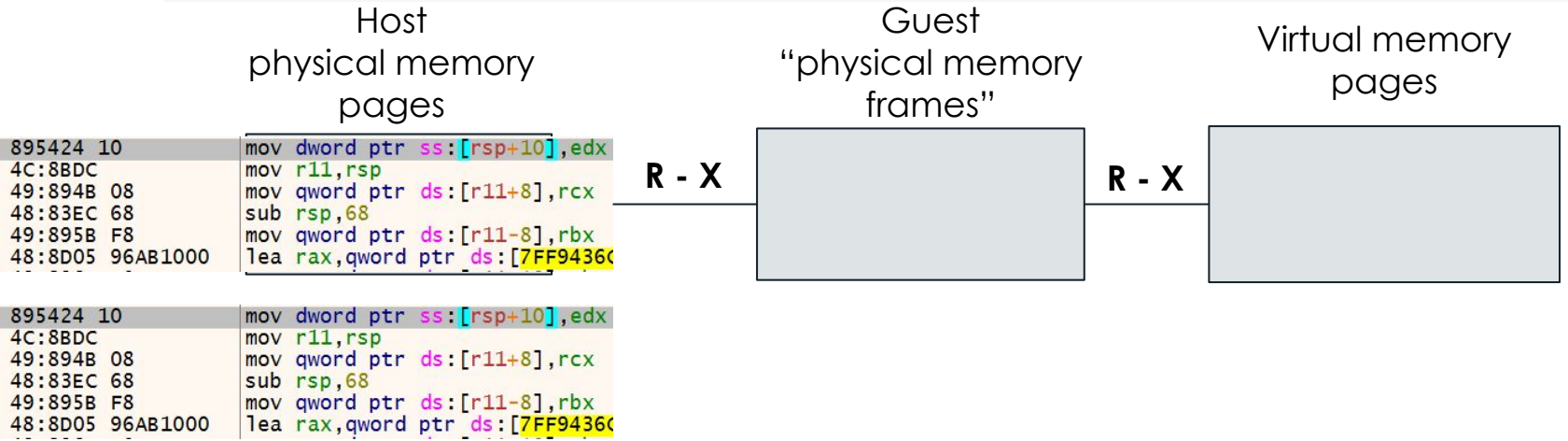
By Tamas K Lengyel | April 13, 2016 | Technical



VT-x: Extended Page Tables - altp2m

STEALTHY MONITORING WITH XEN ALTP2M

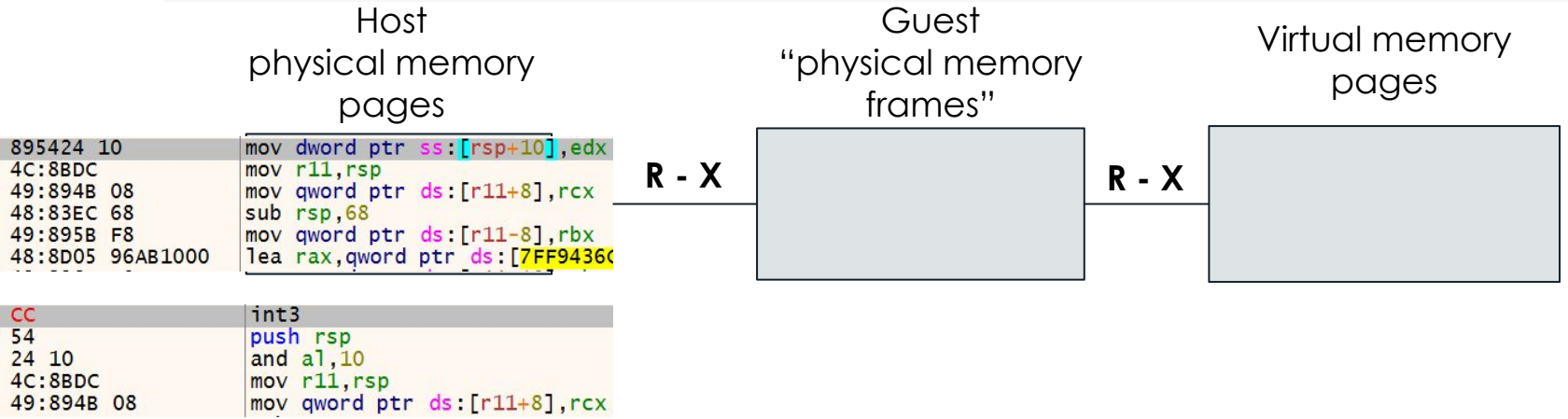
By Tamas K Lengyel | April 13, 2016 | Technical



VT-x: Extended Page Tables - altp2m

STEALTHY MONITORING WITH XEN ALTP2M

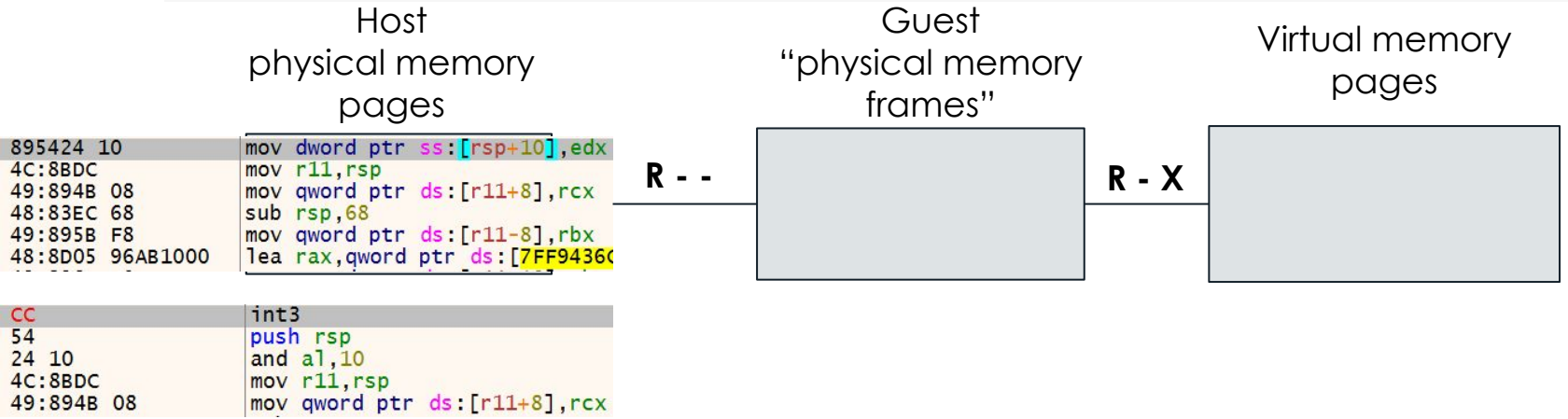
By Tamas K Lengyel | April 13, 2016 | Technical



VT-x: Extended Page Tables - altp2m

STEALTHY MONITORING WITH XEN ALTP2M

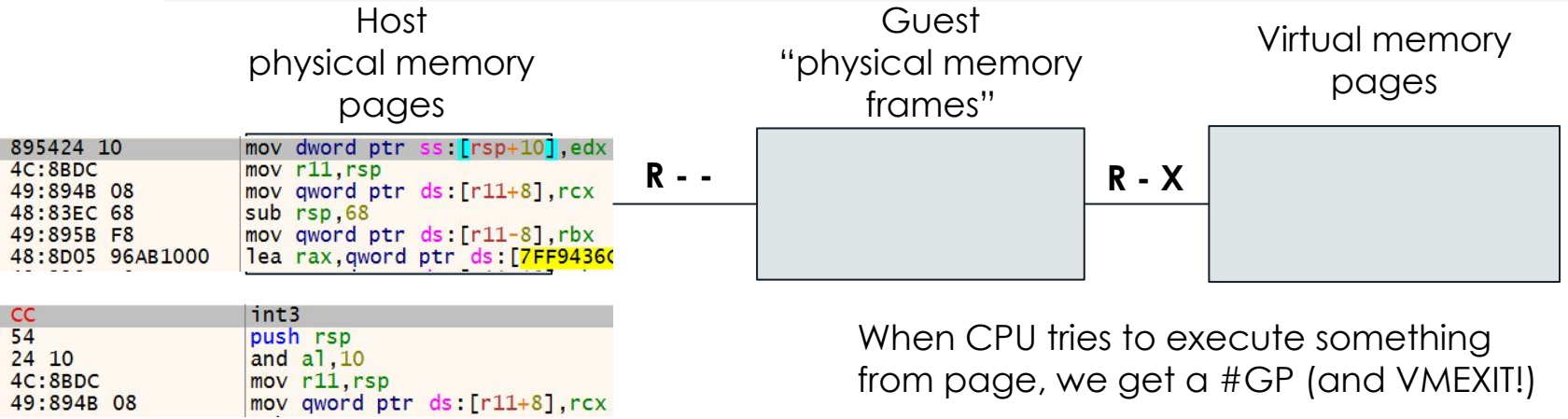
By Tamas K Lengyel | April 13, 2016 | Technical



VT-x: Extended Page Tables - altp2m

STEALTHY MONITORING WITH XEN ALTP2M

By Tamas K Lengyel | April 13, 2016 | Technical

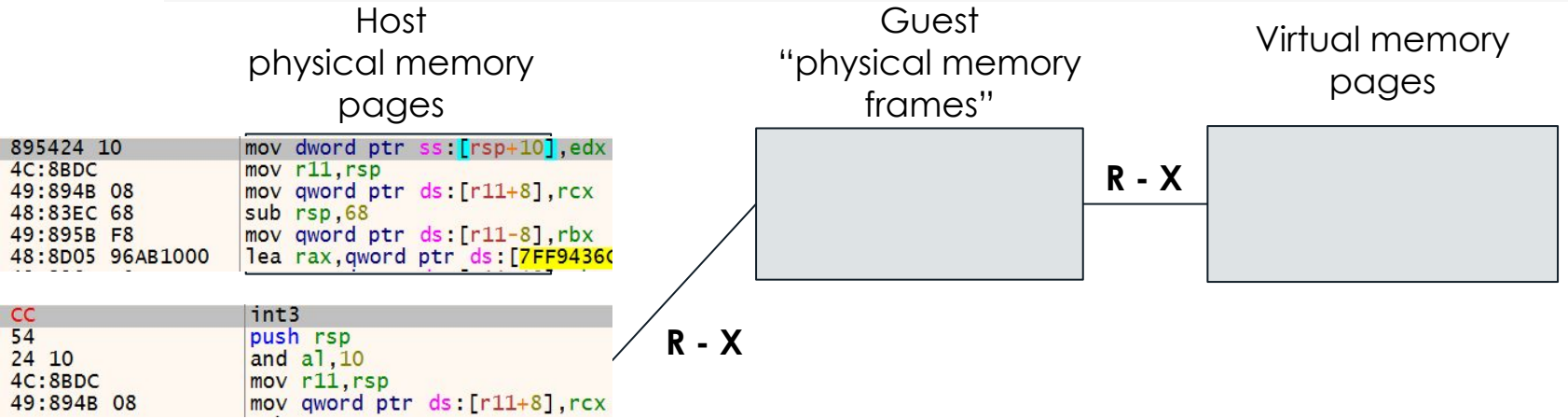


When CPU tries to execute something from page, we get a #GP (and VMEXIT!)

VT-x: Extended Page Tables - altp2m

STEALTHY MONITORING WITH XEN ALTP2M

By Tamas K Lengyel | April 13, 2016 | Technical



Semantic gap

- ➔ Where is kernel located in guest physical memory?
- ➔ Which page table belongs to which process?
- ➔ What is actually loaded into memory and what is paged out?

LibVMI (Virtual Machine Introspection)

LibVMI is a C library with Python bindings that makes it easy to **monitor the low-level details of a running virtual machine** by viewing its memory, trapping on hardware events, and accessing the vCPU registers. This is called **virtual machine introspection**.

Source: libvmi.com

LibVMI (Virtual Machine Introspection)

```
vmi_get_library_arch
vmi_translate_kv2p
vmi_translate_uv2p
vmi_translate_ksym2v
vmi_translate_sym2v
vmi_translate_v2sym
vmi_translate_v2ksym
vmi_pid_to_dtb
vmi_dtb_to_pid
vmi_pagetable_lookup
vmi_pagetable_lookup_extended
vmi_nested_pagetable_lookup
vmi_nested_pagetable_lookup_extended
vmi_read
vmi_read_8
vmi_read_16
vmi_read_32
vmi_read_64
vmi_read_addr
```

Test if LibVMI is working by running vmi-process-list:

```
sudo vmi-process-list windows7-sp1
```

Output should be something similar:

```
Process listing for VM windows7-sp1 (id=7)
[    4] System (struct addr:84aba980)
[  220] smss.exe (struct addr:85a44020)
[  300] csrss.exe (struct addr:85f67a68)
[  336] wininit.exe (struct addr:8601e030)
[  348] csrss.exe (struct addr:84ba4030)
[  384] winlogon.exe (struct addr:85966d40)
[  444] services.exe (struct addr:8614c030)
[  460] lsass.exe (struct addr:86171030)
[  468] lsm.exe (struct addr:8617b4f8)
[  564] svchost.exe (struct addr:861d9bc8)
[  628] svchost.exe (struct addr:863fb8a8)
[  816] sppsvc.exe (struct addr:86426838)
[  856] svchost.exe (struct addr:854abd40)
[  880] svchost.exe (struct addr:854c5030)
```

Source: drakvuf.com

Crossing the semantic gap

- ➔ Knowledge about specific offsets is based on PDB symbols from Microsoft Symbol Server
- ➔ They're parsed into digestible JSON form using popular forensic framework: volatility3

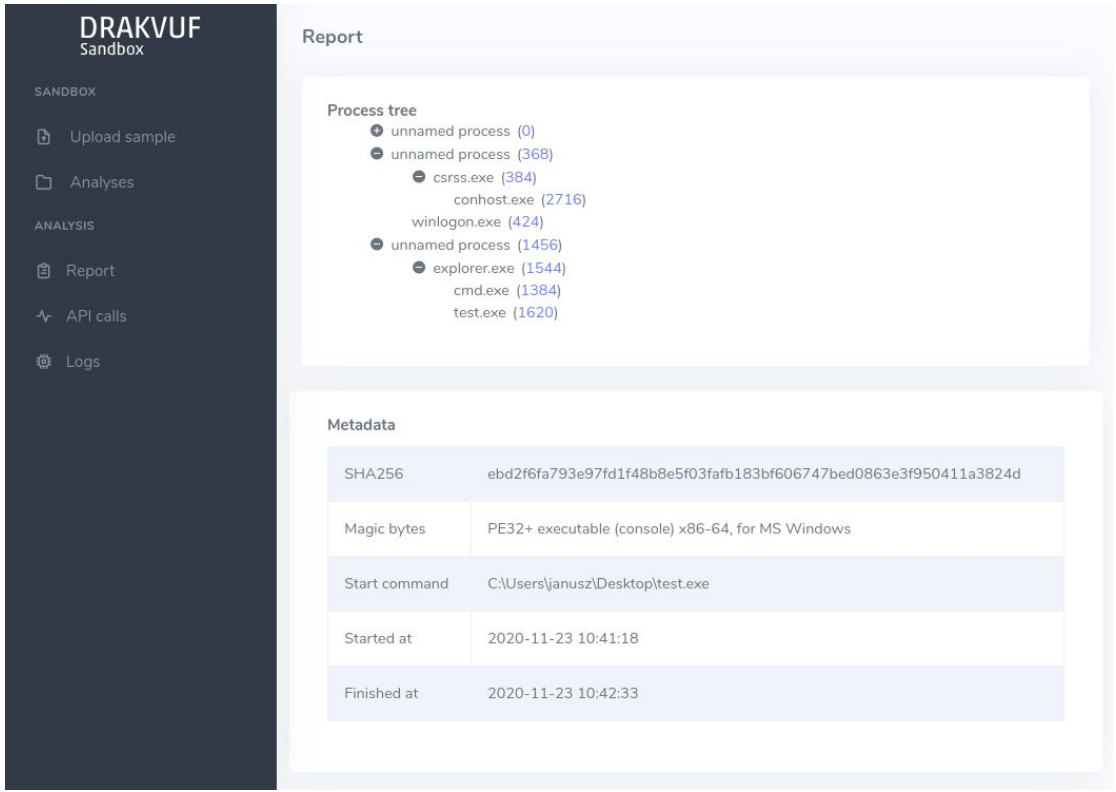
Volatility 3: The volatile memory extraction framework

Volatility is the world's most widely used framework for extracting digital artifacts from volatile memory (RAM) samples. The extraction techniques are performed completely independent of the system being investigated but offer visibility into the runtime state of the system. The framework is intended to introduce people to the techniques and complexities associated with extracting digital artifacts from volatile memory samples and provide a platform for further work into this exciting area of research.

How it actually works?

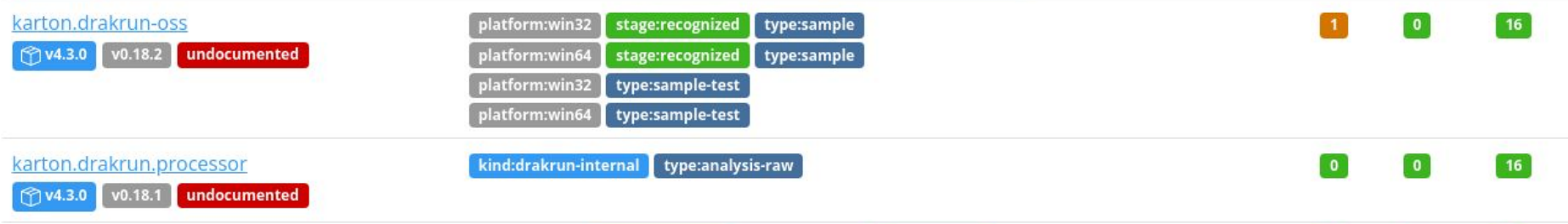
Let’s try with a demo🙏

DRAKVUF Sandbox - making an actual sandbox solution



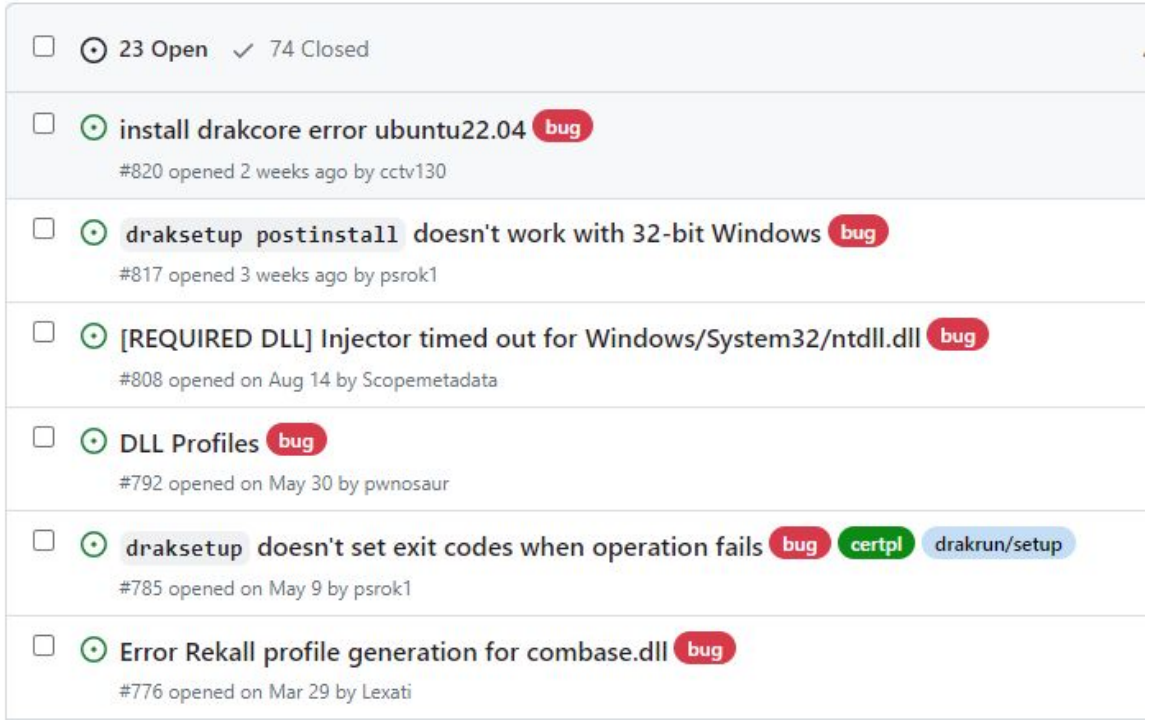
CERT.PL NASK

DRAKVUF Sandbox - making an actual sandbox solution



CERT.PL
NASK

DRAKVUF Sandbox - making an actual sandbox solution



DRAKVUF Sandbox - making an actual sandbox solution

Oct 27, 2021

github-actions

DRAKVUF Sandbox v0.18.0 Adiós Edition

rakovskij-stanislav on Aug 21, 2022

It's curious why this edition is called "Adiós". Does Drakvuf Sandbox development stop here?

Mark as answer ↑ 1

catsuryuu on Nov 17, 2022 Maintainer Author

Explain yourself, @chivay! 😊

Mark as answer

CERT.PL NASK

DRAKVUF Sandbox - making an actual sandbox solution

Commits on Sep 20, 2023

CI: Build debs for Buster and Bullseye (#807)
psrok1 committed 2 weeks ago ✓

Commits on Sep 14, 2023

Bump drakvuf to include fix/ignore-kernel-trap-frame-2 (#819)
psrok1 committed 3 weeks ago ✓

Commits on Sep 7, 2023

build(deps): Bump uwsgi from 2.0.20 to 2.0.22 in /drakcore (#810) ...
dependabot[bot] committed last month ✓

Run e2e tests on self-hosted runner (#816)
psrok1 committed last month ✓

CERT.PL NASK

Kudos!

- ➔ Tamas Lengyel: author of DRAKVUF engine
- ➔ CERT.pl DRAKVUF and DRAKVUF sandbox team for making enormous amount of work and research on both projects!
- ➔ GSoC 2021 students: Manorit Chawdrhy, Jan Gruber

https://www.youtube.com/watch?v=36SNbTX-RNE
https://icedev.pl/static/confidence2022.pdf

CERT.PL NASK

Questions?

cert.pl/en/contact
psrok1@cert.pl

@CERT_Polska_en
@_psrok1