



HERE

- **Claudio “nex” Guarnieri @botherder**
 - Security Researcher at **Rapid7** Labs
 - Core member of **The Shadowserver Foundation**
 - Core member of **The HoneyNet Project**
 - Creator of **Cuckoo Sandbox**
 - Founder of **Malwr.com**



HERE

- **Mark “rep” Schloesser** [@repmovsb](#)
 - Security Researcher at **Rapid7** Labs
 - Core Member of **The HoneyNet Project**
 - Core developer of **Cuckoo Sandbox**
 - Developed other tools such as **Dionaea**

HERE

- **Jurriaan “skier” Bremer @skier_t**
 - Freelance Security Researcher
 - Core developer of **Cuckoo Sandbox**



NOT HERE

- **Alessandro “jekil” Tanasi @jekil**
 - Core developer of **Cuckoo Sandbox**
 - Co-founder of **Malwr.com**
 - Creator of **Hostmap**
 - Creator of **ImageForensics.org**



AGENDA

- Introduction to Sandboxing
- Introduction to Cuckoo
- Components of Cuckoo
- Anti-Anti-Virtualization
- Virtual Machine Introspection

SANDBOXING

How does a sandbox look like?
Software or hardware **appliances**
that receive suspicious files and
returns an **overview of their**
functionality.

PROBLEMS

- Process **high volumes**?
- **Automate** specific tasks?
- **Integrate** with defenses?
- Support your T1 **analysts**?
- **Digital forensics/incident response**?

PROS

- **Automate** the whole analysis process
- Process **high volumes** of malware
- Usable by virtually **anyone**
- Get the actual **executed code**
- Can be very effective if used smartly

CONS

- Can be **expensive** :-(
• Some portions of the **code might not be triggered**
• Environment **could be detected**
• Can be a complete waste

CUCKOO SANDBOX

Automated **malware analysis** system, easy to use and customize.



Powered by **RAPID7**



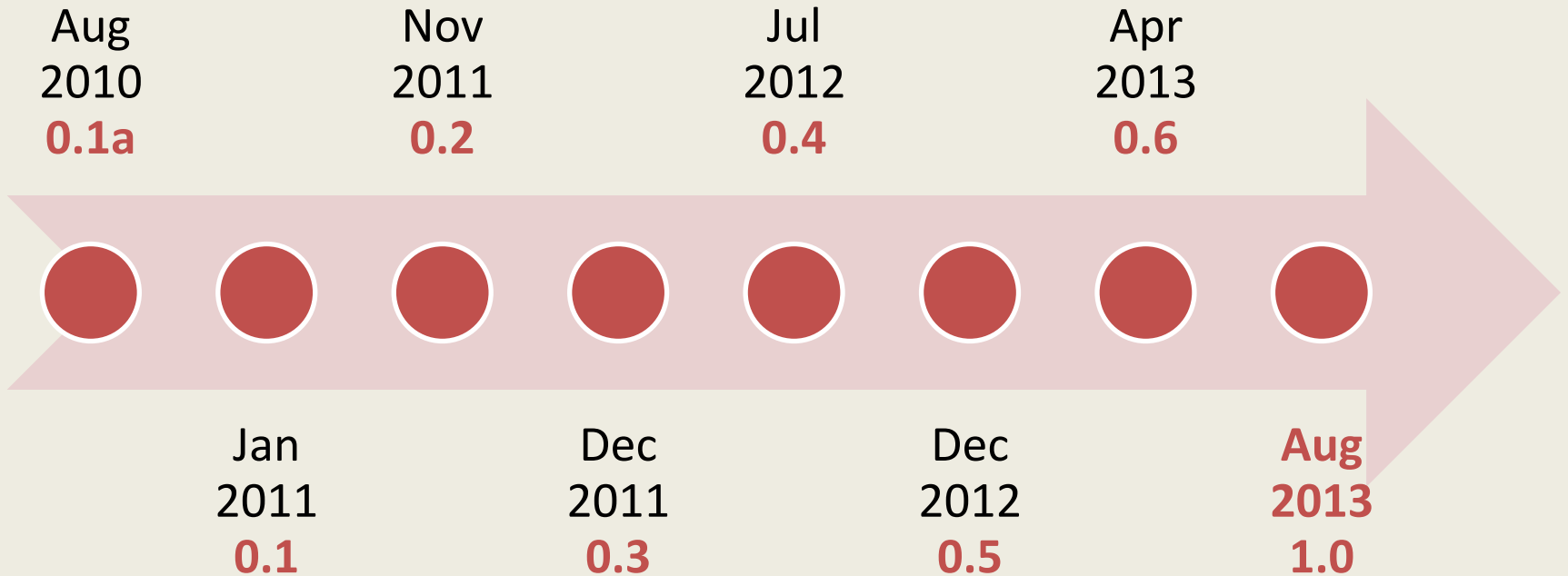
WHY?

- We **believe** in open source
- Empower **students** and researchers
- Open architecture for more **flexibility** and **creativity**

SOME NUMBERS

- Around **50000** lines of code, **Python** and **C**
- More than **2000** commits
- **4** core developers
- **~25** contributors over time
- **~15000** downloads in the last 6 months

BITS OF HISTORY

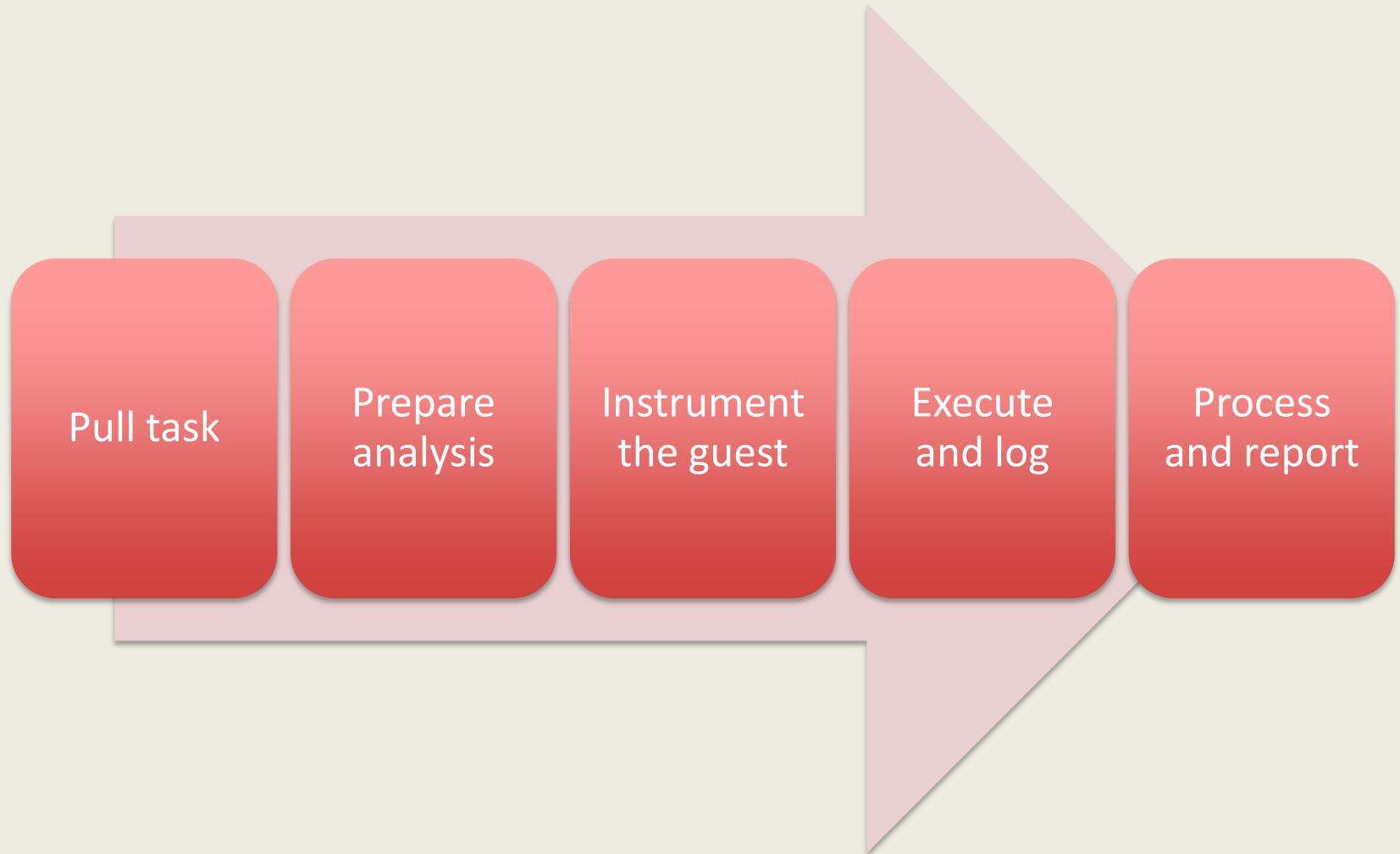


WHAT YOU NEED TO KNOW

- Basic usage of **Linux**
- Basic usage of **virtual machines**
- Knowledge to leverage the results
 - Windows APIs
 - Malicious behaviors
- **With Python you can get awesome!**
 - Customization
 - Modules



HOW IT WORKS



KEY FEATURES

- Almost **everything is a module**
- Completely automated
- Run **concurrent** analysis
- Able to **trace** processes recursively
- **Customize analysis** process
- Create behavioral **signatures**
- Customize processing and reporting

GETTING STARTED

REQUIREMENTS AND EXPECTATIONS

- **What is your goal?**
- **Who** is going to use the sandbox?
- How are they going to **consume the data**?
- **How many samples** do you expect?
- What kind of results are **mostly relevant**?
- Do you need **all features** to meet your goal?

DESIGN YOUR ENVIRONMENT

- Do you want to run **Office** exploits?
- Do you want to run **PDF** exploits?
- Do you want to run **64 bit** malware?
- Do you want to run **URLs**?
- Do you need **script interpreters**?

IDEAS

- Look for the **most exploitable** version of applications (*metasploit, exploitdb, etc.*)
- Create **multiple VMs** with multiple versions of applications
- Leave some **fake credentials** and tokens around
- **Disguise the VM** as much as possible

INSTALLATION IN A NUTSHELL

- Install **VirtualBox**, **VMWare** or **QEMU/KVM**
- Download & extract **Cuckoo**
- Install **dependencies**
- Create a virtual machine, copy over and run ***agent.py*** and take a snapshot (**need to be able to communicate with the host**).
- Configure the files in ***conf/***
- **\$ *python cuckoo.py***



SETUP DISCLAIMERS

- It's not point-and-click, you need to work a bit
- Virtualization software **are not intended** for massive and continuous restore
- There are some **key steps** to do, if one is skipped nothing works
- There's an **extensive documentation**, mailing list and Q&A platform: check them out.

USAGE

SUBMISSION

- *utils/submit.py*
- *utils/api.py*
- Django Web Interface
- Python API

```
1 import sys
2 sys.path.append('/opt/cuckoo/')
3 from lib.cuckoo.core.database import Database
4
5 db = Database()
6 db.add_path(file_path)
7 db.add_url(url)
```

OPTIONS

- **Analysis Package** + Options
- Timeout
- Priority
- Machine
- Platform
- Memory Dump
- Enforce Timeout
- Clock

RESULTS

- Raw results stored in *storage/analysis/<id>/*
- Reports stored in *storage/analysis/<id>/reports/*
 - Depends on what was enabled in *conf/reporting.conf*

RESULTS

- Trace of **API calls**
- File dumps
- Screenshots
- **Network traffic**
- Process memory dump
- System memory dump

CORE MODULES

MACHINERY MODULES

- In **Core** (under *modules/machinery/*)
- Python class
- **Define interaction** with the virtualization software
- Default:
 - VirtualBox
 - VMWare
 - QEMU/KVM
 - Generic LibVirt


```
1 # Copyright (C) 2010-2013 Cuckoo Sandbox Developers.
2 # This file is part of Cuckoo Sandbox - http://www.cuckoosandbox.org
3 # See the file 'docs/LICENSE' for copying permission.
4
5 import logging
6
7 from lib.cuckoo.common.abstracts import LibVirtMachinery
8
9 class KVM(LibVirtMachinery):
10     """Virtualization layer for KVM based on python-libvirt."""
11
12     # Set KVM connection string.
13     dsn = "qemu:///system"
14
```

AUXILIARY MODULES

- In **Core** (under *modules/auxiliary/*)
- Python class
- No specific use, just **run concurrently** to each analysis.
- Default:
 - Network traffic capture

```
23 class Auxiliary(object):
24     """Base abstract class for auxiliary modules."""
25
26     def __init__(self):
27         self.task = None
28         self.machine = None
29         self.options = None
30
31     def set_task(self, task):
32         self.task = task
33
34     def set_machine(self, machine):
35         self.machine = machine
36
37     def set_options(self, options):
38         self.options = options
39
40     def start(self):
41         raise NotImplementedError
42
43     def stop(self):
44         raise NotImplementedError
```

PROCESSING MODULES

- In **Core** (under *modules/processing/*)
- Python class
- Process **raw results** (sample, API logs, files, memory)
- Populate **collection of results**

```
1 import re
2
3 from lib.cuckoo.common.abstracts import Processing
4 from lib.cuckoo.common.exceptions import CuckooProcessingError
5
6 class Strings(Processing):
7     """Extract strings from analyzed file."""
8
9     def run(self):
10         """Run extract of printable strings.
11         @return: list of printable strings.
12         """
13         self.key = "strings"
14         strings = []
15
16         if self.task["category"] == "file":
17             try:
18                 data = open(self.file_path, "r").read()
19             except (IOError, OSError) as e:
20                 raise CuckooProcessingError("Error opening file {0}".format(e))
21             strings = re.findall("[\x1f-\x7e]{6,}", data)
22
23         return strings
```

SIGNATURES

- In **Core** (under *analyzer/windows/modules/signatures/*)
- Python class
- Isolate specific events
 - Identify malware family
 - Identify malicious behavior
 - Extract configuration
 - ...

```
1 from lib.cuckoo.common.abstracts import Signature
2
3 class SpyEyeMutexes(Signature):
4     name = "banker_spyeye_mutexes"
5     description = "Creates known SpyEye mutexes"
6     severity = 3
7     categories = ["banker"]
8     families = ["spyeye"]
9     authors = ["nex"]
10    minimum = "0.5"
11
12    def run(self):
13        indicators = [
14            "zXeRY3a_PtW.*",
15            "SPYNET",
16            "__CLEANSWEEP__",
17            "__CLEANSWEEP_UNINSTALL__",
18            "__CLEANSWEEP_RELOADCFG__"
19        ]
20
21        for indicator in indicators:
22            if self.check_mutex(pattern=indicator, regex=True):
23                return True
24
25        return False
```

```
1 from lib.cuckoo.common.abstracts import Signature
2
3 class Prinimalka(Signature):
4     name = "banker_prinimalka"
5     description = "Detected Prinimalka banking trojan"
6     severity = 3
7     categories = ["banker"]
8     families = ["prinimalka"]
9     authors = ["nex"]
10    minimum = "0.5.1"
11
12    def run(self):
13        server = ""
14        path = ""
15
16        for process in self.results["behavior"]["processes"]:
17            for call in process["calls"]:
18                if call["api"] != "RegSetValueExA":
19                    continue
20
21                correct = False
22                for argument in call["arguments"]:
23                    if not server:
24                        if argument["name"] == "ValueName" and argument["value"] == "nah_opt_server1":
25                            correct = True
26
27                    if correct:
28                        if argument["name"] == "Buffer":
29                            server = argument["value"].rstrip("\\x00")
30
31                    else:
32                        break
33
34                if server:
35                    break
36
37            if server:
38                self.description += " (C&C: {0})".format(server)
39                return True
40
41        return False
```


COMMUNITY SIGNATURES

- **Community Repository**
 - <https://github.com/cuckooobox/community>
- ***utils/community.py --signatures (--force)***



SHARING IS CARING!

REPORTING MODULES

- In **Core** (under *analyzer/windows/modules/reporting/*)
- Python class
- Make use of abstracted results
- Default:
 - JSON
 - HTML
 - MAEC
 - MongoDB

```
1 import os
2 import json
3 import codecs
4
5 from lib.cuckoo.common.abstracts import Report
6 from lib.cuckoo.common.exceptions import CuckooReportError
7
8 class JsonDump(Report):
9     """Saves analysis results in JSON format."""
10
11     def run(self, results):
12         """Writes report.
13         @param results: Cuckoo results dict.
14         @raise CuckooReportError: if fails to write report.
15         """
16         try:
17             report = codecs.open(os.path.join(self.reports_path, "report.json"), "w", "utf-8")
18             json.dump(results, report, sort_keys=False, indent=4)
19             report.close()
20         except (UnicodeError, TypeError, IOError) as e:
21             raise CuckooReportError("Failed to generate JSON report: %s" % e)
22
```

ANALYZER MODULES

ANALYSIS PACKAGES

- In **Analyzer** (under *analyzer/windows/modules/packages/*)
- Python modules
- Define how to interact with the malware and the system
- Can be used for scripting tasks

```
5 from lib.common.abstracts import Package
6 from lib.api.process import Process
7 from lib.common.exceptions import CuckooPackageError
8
9 class Exe(Package):
10     """EXE analysis package."""
11
12     def start(self, path):
13         free = self.options.get("free", False)
14         args = self.options.get("arguments", None)
15         suspended = True
16         if free:
17             suspended = False
18
19         p = Process()
20         if not p.execute(path=path, args=args, suspended=suspended):
21             raise CuckooPackageError("Unable to execute initial process, analysis aborted")
22
23         if not free and suspended:
24             p.inject()
25             p.resume()
26             p.close()
27             return p.pid
28         else:
29             return None
30
31     def check(self):
32         return True
33
34     def finish(self):
35         if self.options.get("procmemdump", False):
36             for pid in self.pids:
37                 p = Process(pid=pid)
38                 p.dump_memory()
39
40         return True
41
```

AUXILIARY MODULES

- In **Analyzer** (under *analyzer/windows/modules/auxiliaries/*)
- Python modules
- Run concurrently to the analysis
- Default:
 - Screenshots
 - Emulation of human interaction


```
68 ▾ class Human(Auxiliary, Thread):
69     """Human after all"""
70
71 ▾     def __init__(self):
72         Thread.__init__(self)
73         self.do_run = True
74
75     def stop(self):
76         self.do_run = False
77
78 ▾     def run(self):
79 ▾         while self.do_run:
80             move_mouse()
81             click_mouse()
82             USER32.EnumWindows(EnumWindowsProc(foreach_window), 0)
83             KERNEL32.Sleep(1000)
```

CUSTOMIZATION: POISONIVY

- Leverage Cuckoo process dumping to automatically extract PoisonIvy configuration
- Custom Processing Module to match patterns in the dumps
- In case of successful extraction, upload to special server for further monitoring

```

8 signatures = {
9     'namespace1' : 'rule pivars {strings: $a = { \
10         53 74 75 62 50 61 74 68 ?? 53 4F 46 54 57 41 52\
11         45 5C 43 6C 61 73 73 65 73 5C 68 74 74 70 5C 73\
12         68 65 6C 6C 5C 6F 70 65 6E 5C 63 6F 6D 6D 61 6E\
13         64 [22] 53 6F 66 74 77 61 72 65 5C 4D 69 63 72 6F\
14         73 6F 66 74 5C 41 63 74 69 76 65 20 53 65 74 75\
15         70 5C 49 6E 73 74 61 6C 6C 65 64 20 43 6F 6D 70\
16         6F 6E 65 6E 74 73 5C } condition: $a}'
17 }
18
19 class PoisonIvy(Processing):
20     def run(self):
21         self.key = "poisonivy"
22         results = {}
23
24         rules = yara.compile(sources=signatures)
25
26         dumps = []
27         for root, dirs, files in os.walk(self.pmemory_path):
28             if files:
29                 for file_name in files:
30                     dumps.append(os.path.join(root, file_name))
31
32         for dump in dumps:
33             matches = rules.match(dump)
34
35             if not matches:
36                 continue
37
38             data = open(dump, "rb")
39
40             offset = matches[0].strings[0][0]
41             data.seek(offset + 0x6eb)
42             results["identifier"] = data.read(100).split("\x00")[0]
43             data.seek(offset + 0x2a2)
44             results["persistence"] = data.read(100).split("\x00")[0]
45             data.seek(offset - 0x27e)
46             results["server"] = data.read(100).split("\x00")[0]
47
48             break
49
50         return results

```

```
1 import requests
2
3 from lib.cuckoo.common.abstracts import Report
4
5 class PoisonReport(Report):
6
7     def run(self, results):
8         if not "poisonivy" in results or not results["poisonivy"]["domain"]:
9             # No PoisonIvy detected.
10            return
11
12        requests.post("http://192.168.1.10/report/poisonivy", data=results["poisonivy"])
```

CUCKOOMON

CUCKOO MON

- **DLL Injection**
- **Inline Hooking**
- Logging to the host over **TCP connection**
- **Follow execution** of child processes or injection of target processes

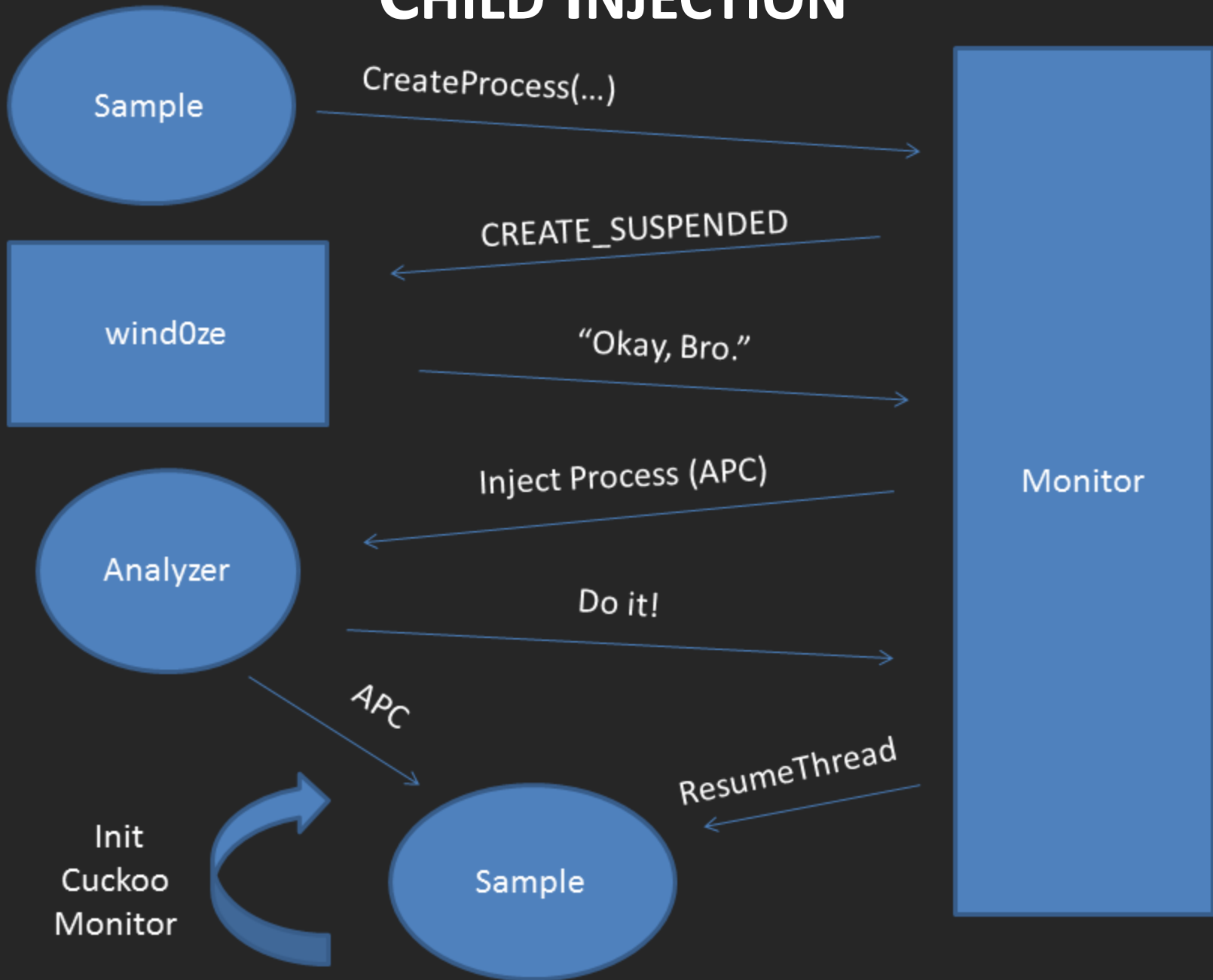
ANALYZER PACKAGE

- Analyzer is uploaded to the VM through the Agent
- By default the **analysis package** will:
 - Start suspended process
 - Inject CuckooMon
 - Resume process

```
p = Process()
if not p.execute(path=path, args=args, suspended=suspended):
    raise CuckooPackageError("Unable to execute initial process, analysis aborted")

if not free and suspended:
    p.inject()
    p.resume()
    p.close()
    return p.pid
```

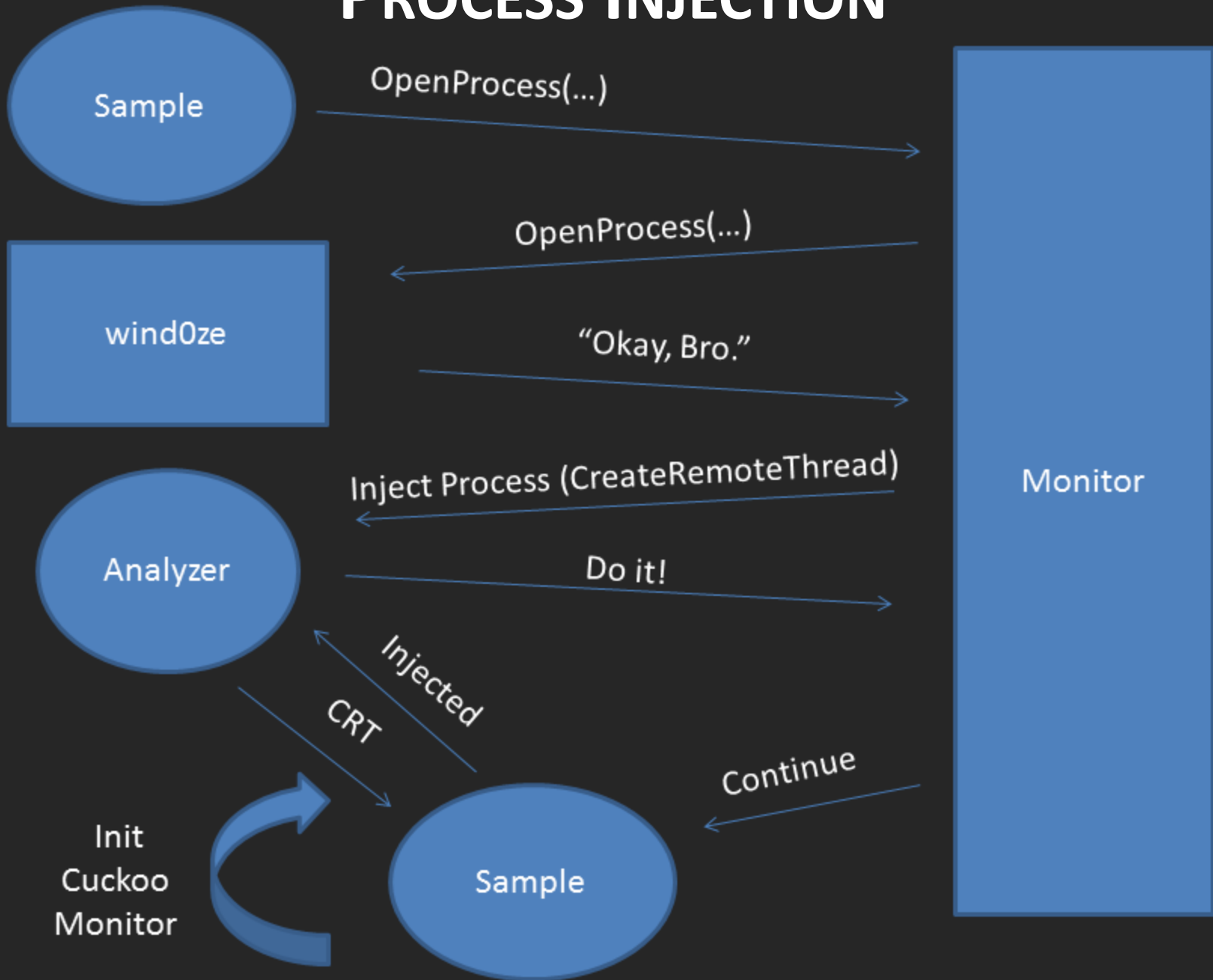
CHILD INJECTION



EVASION ARMS RACE

- Malware often injects into other processes to **avoid detection** (e.g. *iexplore.exe*)
- Also creates **child processes** for other purposes
- To track this, we **monitor for such events** and **inject CuckooMon** in 3rd processes too.

PROCESS INJECTION



API HOOKING OVERVIEW

- Cuckoo logs **about 170 APIs**
- Hook lowest APIs **without loosing context**
 - Not CreateProcessA
 - Not CreateProcessW
 - Not CreateProcessInternalA
 - But CreateProcessInternalW
- However also higher level APIs
 - ShellExecute (protocol handlers, URLs)
 - system (pipe multiple processes)

HOOKING + MAGIC = PROFIT

- Use standard **inline hooking** with a few twists
 - Support for **random preambles** (jmp/push+ret/etc)

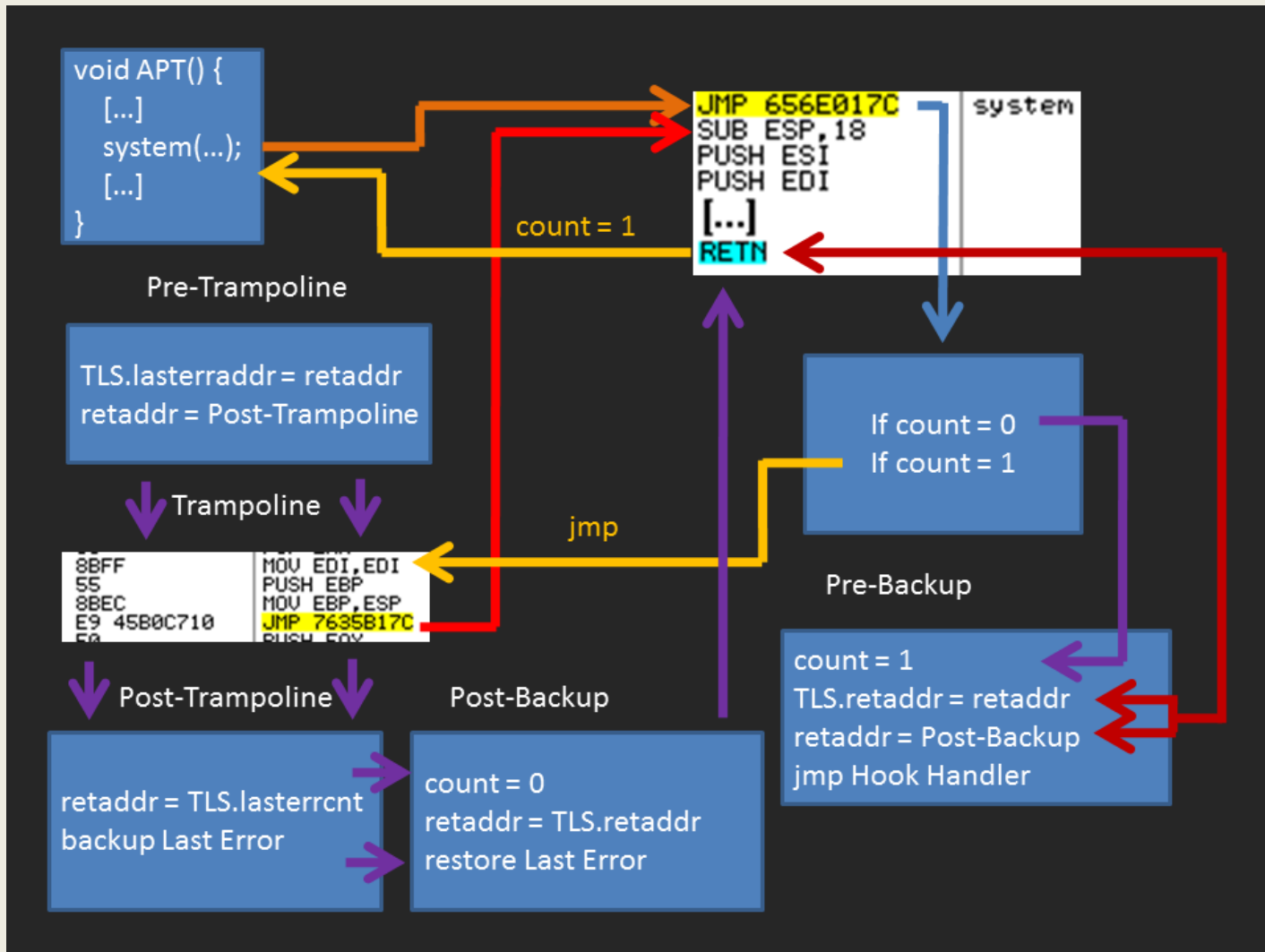
```
HOOKDEF(BOOL, WINAPI, WriteFile,  
_In_ HANDLE hFile,  
_In_ LPCVOID lpBuffer,  
_In_ DWORD nNumberOfBytesToWrite,  
_Out_opt_ LPDWORD lpNumberOfBytesWritten,  
_Inout_opt_ LPOVERLAPPED lpOverlapped  
) {  
  
    [...]  
  
    WriteFile(g_log_handle, "Hello Hook", 10, &bytes, NULL);  
  
    [...]  
}
```

- First hook run is interesting, **ignore recursive ones** down on the callstack
- Transparently manage these situations in hooking mechanism

ASSEMBLY TRAMPOLINES

```
118     unsigned char pre_backup[] = {
119         // push eax
120         0x50,
121
122         // mov eax, fs:[TLS_HOOK_INFO]
123         0x64, 0xa1, TLS_HOOK_INFO, 0x00, 0x00, 0x00,
124         // test eax, eax
125         0x85, 0xc0,
126         // jnz $+0d
127         0x75, 0x0d,
128         // pushad
129         0x60,
130         // call ensure_valid_hook_info
131         0xe8, 0x00, 0x00, 0x00, 0x00,
132         // popad
133         0x61,
134         // mov eax, fs:[TLS_HOOK_INFO]
135         0x64, 0xa1, TLS_HOOK_INFO, 0x00, 0x00, 0x00,
136
137         // cmp dword [eax+hook_info_t.hook_count], 0
138         0x83, 0x78, offsetof(hook_info_t, hook_count), 0x00,
139         // jg $+11
140         0x7f, 0x11,
141         // inc dword [eax+hook_info_t.hook_count]
142         0xff, 0x40, offsetof(hook_info_t, hook_count),
143         // push dword [esp+4]
144         0xff, 0x74, 0xe4, 0x04,
145         // pop dword [eax+hook_info_t.ret_last_error]
146         0x8f, 0x40, offsetof(hook_info_t, ret_last_error),
147         // mov dword [esp+4], new_return_address
148         0xc7, 0x44, 0xe4, 0x04, 0x00, 0x00, 0x00, 0x00,
149
150         // pop eax
151         0x58,
152     };
```

RESULTING HOOKS



WORK IN PROGRESS

- **Return address + module tracking**
 - Only log when coming from interesting sources
(reduce noise when malware injects into other processes)
- **StubDLL**
 - Don't hook, shadow DLL that "overloads"
functions
(avoid inline hooking countermeasures / detection)

ANTI-ANTI-SANDBOX

With sandboxes getting popular, malware writers are increasingly trying to bypass them.

COMMON TRICKS

- **Sleep** before main execution
- Monitor **mouse events** (SetWindowsHookEx 0x07, 0x0E)
- Check for **virtualization software**:
 - Files
 - Processes
 - Devices (CD-ROM, HDD)
 - Registry keys

ANTI-SLEEP

- Cuckoo Sandbox **skips sleeps** that are launched **within the first seconds** of a process execution.

ANTI-MOUSE-MONITOR

- Cuckoo Sandbox **emulates human interaction**
 - Move the mouse cursor
 - Click on mouse buttons
 - Click on dialogs

ANTI-VIRTUALIZATION

- It's **painful**
- **Depends** on the virtualization software of your choice
- You can **do something** about it
- However you **won't be able to kill all indicators**

VIRTUALBOX EXTRA DATA

\$ VBoxManage setextradata <label> VBoxInternal/Devices/ +

- pcbios/0/Config/DmiBIOSFirmwareMajor
- pcbios/0/Config/DmiBIOSFirmwareMinor
- pcbios/0/Config/DmiBIOSReleaseDate
- pcbios/0/Config/DmiBIOSReleaseMajor
- pcbios/0/Config/DmiBIOSReleaseMinor
- pcbios/0/Config/**DmiBIOSVendor**
- pcbios/0/Config/DmiBIOSVersion
- pcbios/0/Config/DmiChassisAssetTag
- pcbios/0/Config/DmiChassisSerial
- pcbios/0/Config/**DmiChassisVendor**
- pcbios/0/Config/DmiChassisVersion
- pcbios/0/Config/**DmiSystemFamily**
- pcbios/0/Config/DmiSystemProduct
- pcbios/0/Config/DmiSystemSKU
- pcbios/0/Config/DmiSystemSerial
- pcbios/0/Config/DmiSystemUuid
- pcbios/0/Config/**DmiSystemVendor**
- pcbios/0/Config/DmiSystemVersion
- piix3ide/0/Config/Port0/ATAPIProductId
- piix3ide/0/Config/Port0/ATAPIRevision
- piix3ide/0/Config/Port0/**ATAPIVendorId**
- piix3ide/0/Config/PrimaryMaster/FirmwareRevision
- piix3ide/0/Config/PrimaryMaster/**ModelNumber**
- piix3ide/0/Config/PrimaryMaster/SerialNumber

DO **NOT** INSTALL
THE **GUEST ADDITIONS.**



WINDOWS REGISTRY

- HKLM\HARDWARE\Description\System\SystemBiosVersion
- HKLM\HARDWARE\Description\System\VideoBiosVersion
- HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0
- HKLM\SYSTEM\CurrentControlSet\Enum\IDE\

CUCKOOVMI

ALTERNATIVE ANALYSIS TECHNIQUES

- CuckooMon: userland DLL injection
 - comfortable, simple, still effective
 - sadly **easy to detect/circumvent**
- Commercial sandboxes often kernel based tracing, sometimes combined with userland components
- Even harder to detect: **introspection from outside the OS**

Cuckoo VMI?



GENERALIZING CUCKOO LOG DATA

- Necessary changes to Cuckoo
 - Generalizing behavior semantics for Mac/Linux platforms anyway
- More visibility / possibilities with VMI
 - Might need more flexible configuration of the analyzer engine

VIRTUAL MACHINE INTROSPECTION

- Observe the memory and **execution flow from the outside**
- Look at kernel structures to differentiate between processes / libraries
- Depending on virtualization technique use its features to pause VM execution **and extract function arguments / memory contents**

WINDOWS KERNEL DETAILS

- What do we need for inspecting Windows from the outside?
 - Processes (track cr3)
 - Libraries / Modules
- Kernel structures:
 - EPROCESS (ActiveProcessHead list)
 - Process Object Tables (HANDLE_TABLE)
 - Virtual Address Descriptor tree (VAD tree)

WIP: CUCKOOVMI BASED ON QEMU

- QEMU: binary translation engine: TCG (Tiny Code Generator)
- Great base for both coarse- and fine-grained tracing of the guest and its processes
- Focus on Windows XP/7 – find kernel process structs and track their executable memory
- Full tracing or specific locations
- Never miss executed code

AUTOMATED FUNCTIONCALL LOGGING

- Windows APIs mostly use stdcall calling convention
 - Callee cleans up the stack, EAX = returnvalue
- This allows for generic parameter logging
 - Note stack pointer when entering function
 - Note stack pointer when returning
 - Everything in between was a parameter
- Still needs knowledge of types for special logging (Strings, structs, etc)

AUTOMATED LOGGING CONT.

- Type information can be automatically extracted from development headers

```
NTSTATUS NtCreateFile(HANDLE* FileHandle, FILE_ACCESS_MASK DesiredAccess,
OBJECT_ATTRIBUTES* ObjectAttributes, IO_STATUS_BLOCK* IoStatusBlock,
LARGE_INTEGER* AllocationSize, FILE_ATTRIBUTES_ULONG FileAttributes, FileShareMode
ShareAccess, NtCreateDisposition CreateDisposition, NtCreateOptions CreateOptions,
VOID* EaBuffer, ULONG EaLength)
```

- Specify list of interesting variables in all those structs, generate dereference/offset code automatically
- Comes down to only implementing specific code for elementary types (char *, wchar_t *, UNICODE_STRING)

CUCKOOVMI EXAMPLE

```
1 --- Tracking Process amstreamx.tmp PID 1292 TID 1288 ---
2 [...]
3 PID:1292 TID:1288 call 0x402682->0x7c80b731 -- kernel32.dll:GetModuleHandleA([4239724])
4 -> additional: {u'lpModuleName': u'KERNEL32'}
5 PID:1292 TID:1288 call 0x402692->0x7c80ae30 -- kernel32.dll:GetProcAddress([2088763392, 2088808122])
6 PID:1292 TID:1288 call 0x40269e->0x7c80aeba -- kernel32.dll:IsProcessorFeaturePresent([0])
7 PID:1292 TID:1288 call 0x4099e5->0x7c9100a4 -- ntdll.dll:RtlAllocateHeap([8716288, 9, 2048])
8 PID:1292 TID:1288 call 0x408670->0x7c8449fd -- kernel32.dll:SetUnhandledExceptionFilter([4228645])
9 PID:1292 TID:1288 call 0x40258d->0x7c801ef2 -- kernel32.dll:GetStartupInfoA([1245028])
10 PID:1292 TID:1288 call 0x4025b0->0x7c80b731 -- kernel32.dll:GetModuleHandleA([0])
11 -> additional: {u'lpModuleName': u'KERNEL32'}
12 PID:1292 TID:1288 call 0x40182c->0x7c835de2 -- kernel32.dll:GetTempPathA([256, 4247808])
13 PID:1292 TID:1288 call 0x4084b0->0x7c801a28 -- kernel32.dll:CreateFileA([1244452, 1073741824, 3, 1244296])
14 -> additional: {u'lpFileName': u'C:\\\\DOCUME~1\\john\\LOCALS~1\\Temp\\desktopc.ini'}
15 PID:1292 TID:1288 call 0x4084bd->0x7c810ee1 -- kernel32.dll:GetFileType([40])
16 PID:1292 TID:1288 call 0x40140f->0x7c835de2 -- kernel32.dll:GetTempPathA([260, 1243400])
17 PID:1292 TID:1288 call 0x4084b0->0x7c801a28 -- kernel32.dll:CreateFileA([1243140, 1073741824, 3, 1242900])
18 -> additional: {u'lpFileName': u'C:\\\\DOCUME~1\\john\\LOCALS~1\\Temp\\~WRL0000l.tmp'}
19 PID:1292 TID:1288 call 0x4084bd->0x7c810ee1 -- kernel32.dll:GetFileType([44])
20 PID:1292 TID:1288 call 0x4012d4->0x7c801a28 -- kernel32.dll:CreateFileA([4243608, 0, 3, 0])
21 -> additional: {u'lpFileName': u'\\\\.\\PhysicalDrive0'}
22 PID:1292 TID:1288 call 0x4012ff->0x7c801629 -- kernel32.dll:DeviceIoControl([48, 458752, 0, 0])
23 -> additional: {u'lpInBuffer': Binary('', 0), u'lpOutBuffer': Binary('', 0)}
24 PID:1292 TID:1288 call 0x401310->0x7c809bd7 -- kernel32.dll:CloseHandle([48])
25 PID:1292 TID:1288 call 0x403e31->0x7c9100a4 -- ntdll.dll:RtlAllocateHeap([8716288, 1, 4096])
26 PID:1292 TID:1288 call 0x401521->0x7c82c2cb -- kernel32.dll:GetLogicalDriveStringsA([260, 0])
27 PID:1292 TID:1288 call 0x401552->0x7e41a8ad -- user32.dll:wsprintfA([])
28 -> additional: {u'lpFmt': u'\\\\.\\PhysicalDrive0'}
29 PID:1292 TID:1288 call 0x4015af->0x7c809c88 -- kernel32.dll:MultiByteToWideChar([0, 0, 1243084, 2])
30 PID:1292 TID:1288 call 0x401339->0x7c801a28 -- kernel32.dll:CreateFileA([1243096, 268435456, 3, 0])
31 -> additional: {u'lpFileName': u'\\\\.\\C:'}
32 PID:1292 TID:1288 call 0x401363->0x7c801629 -- kernel32.dll:DeviceIoControl([48, 475140, 0, 0])
33 -> additional: {u'lpInBuffer': Binary('', 0), u'lpOutBuffer': Binary('', 0)}
34 PID:1292 TID:1288 call 0x401373->0x7c809bd7 -- kernel32.dll:CloseHandle([48])
```

DEMO



RELATED WORK: DECAF PLATFORM

- Qemu based analysis framework out of Berkeley
- Base of Android analysis project “**DroidScope**”
- Also supports tracing / analysing x86 Windows guests
- Parts from closed **TEMU** and other related projects
- Rich hooking API
 - Specific addresses, all basic blocks, memory write, etc
- Experimental taint tracking features
- **Too many features and too invasive** (outdated QEMU, etc) **for our purpose**

ALTERNATIVE VMI SOLUTIONS

- Thin hypervisor for VM performance
 - Use page protection faults to trap to the hypervisor at interesting locations
- Other rootkit techniques? UEFI drivers?
- Cuckoo hopefully grows to other platforms and several analyzer techniques to choose from
 - Brings even more customization / flexibility

CONCLUSIONS

SUMMING UP

- Open source solution (and will remain so)
- Flexible and customizable
- Easy to integrate
- Very actively developed

FUTURE

- Improve **performances**
- Continue work on **VMI** techniques
- **Bare-metal** support (almost done)
- Add **Linux** support
- Add **Mac OS X** support
- **Feedback?**

OTHER STUFF

- **Malwr**
 - <https://malwr.com>
- **VxCage**
 - <https://github.com/cuckooobox/vxcage>



www.cuckoosandbox.org

@cuckoosandbox