cuckoo

black hat USA 2013

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.

GPL Free Software

Free as in Freedom

Powered by RAPID7

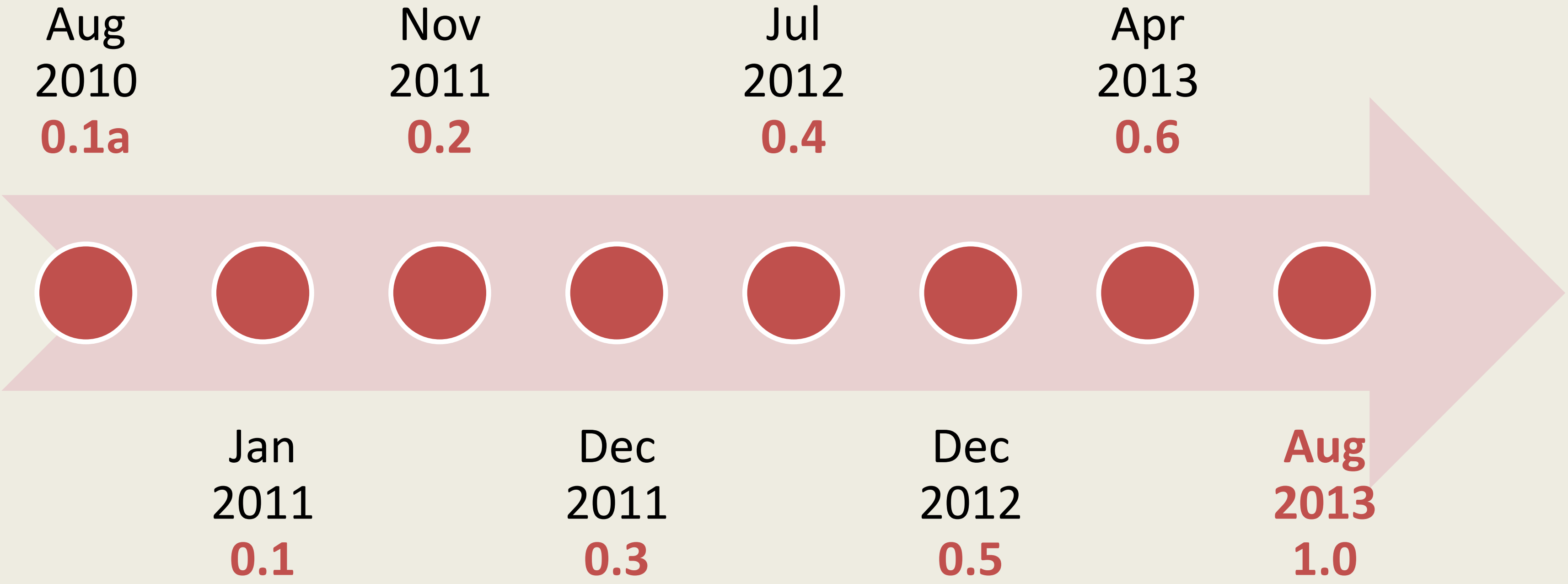
Why?

- 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



Aug 2010 **0.1a**

Jan 2011 **0.1**

Nov 2011 **0.2**

Dec 2011 **0.3**

Jul 2012 **0.4**

Dec 2012 **0.5**

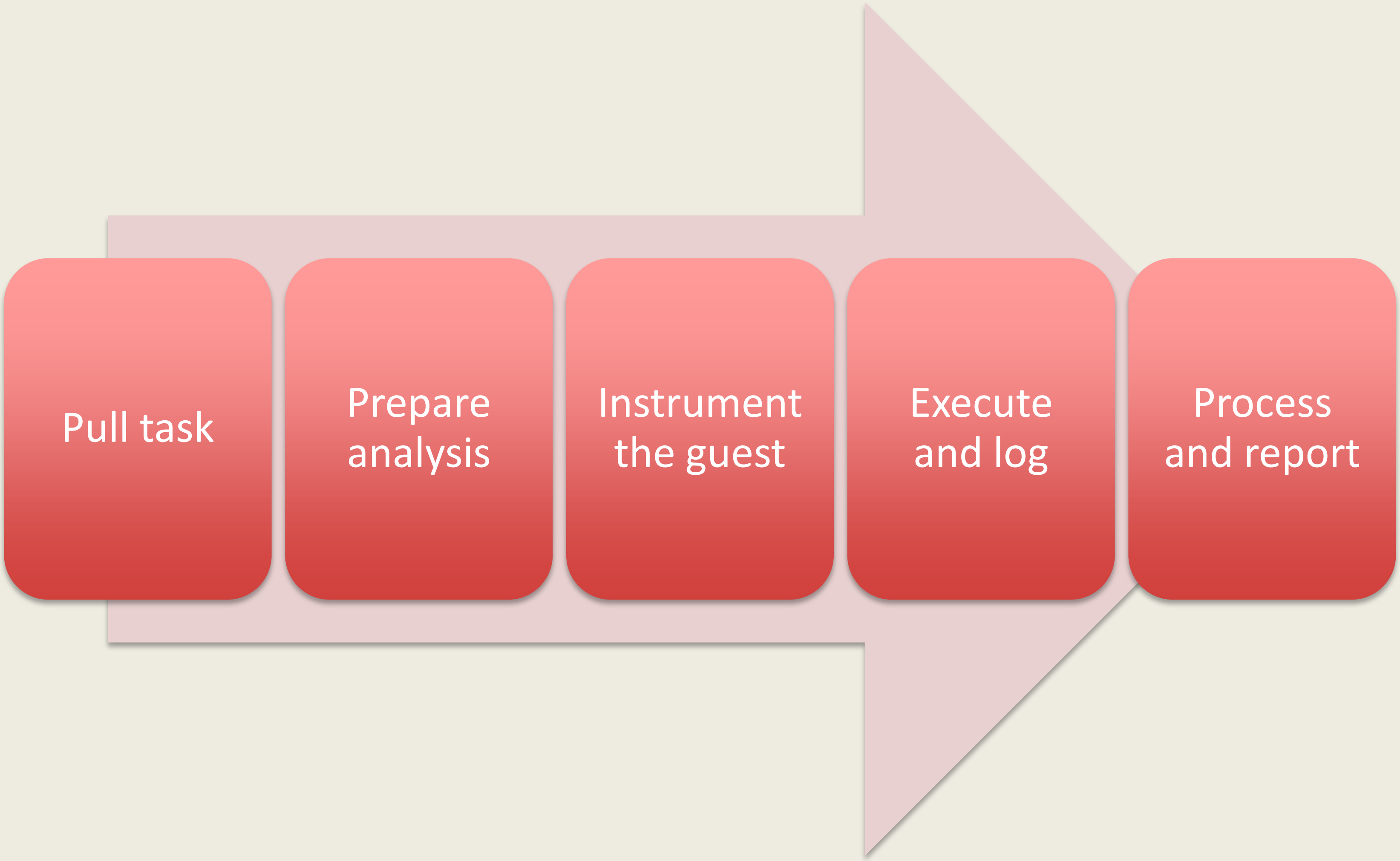
Apr 2013 **0.6**

Aug 2013 1.0

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



Pull task

Prepare analysis

Instrument the guest

Execute and log

Process and report

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**

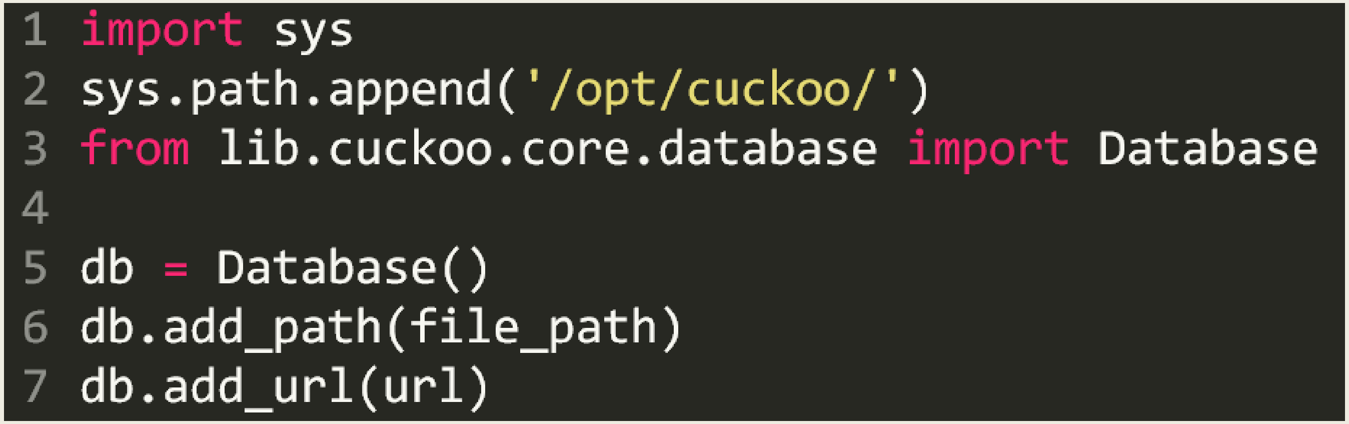
Setup Disclaimers

- 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



```python
import sys
sys.path.append('/opt/cuckoo/')
from lib.cuckoo.core.database import Database

db = Database()
db.add_path(file_path)
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*

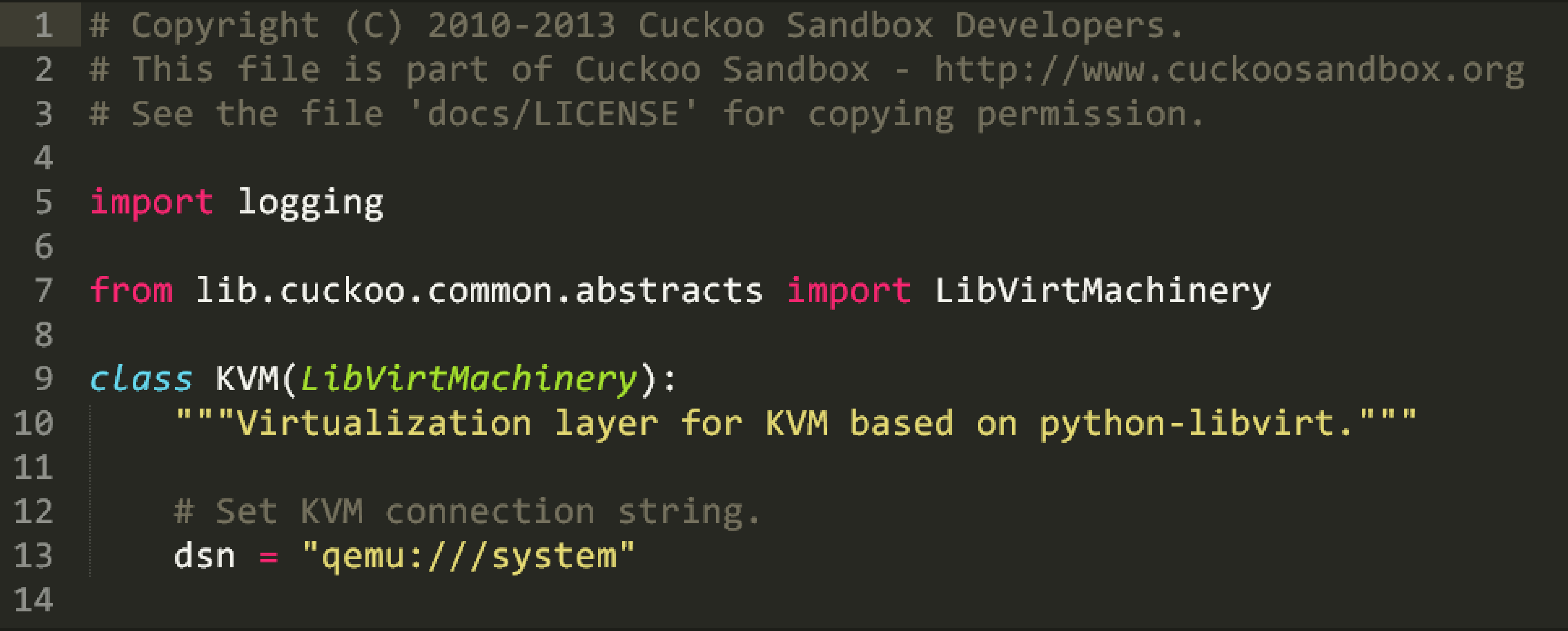
Results

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

CORE MODULES

Machinery Modules

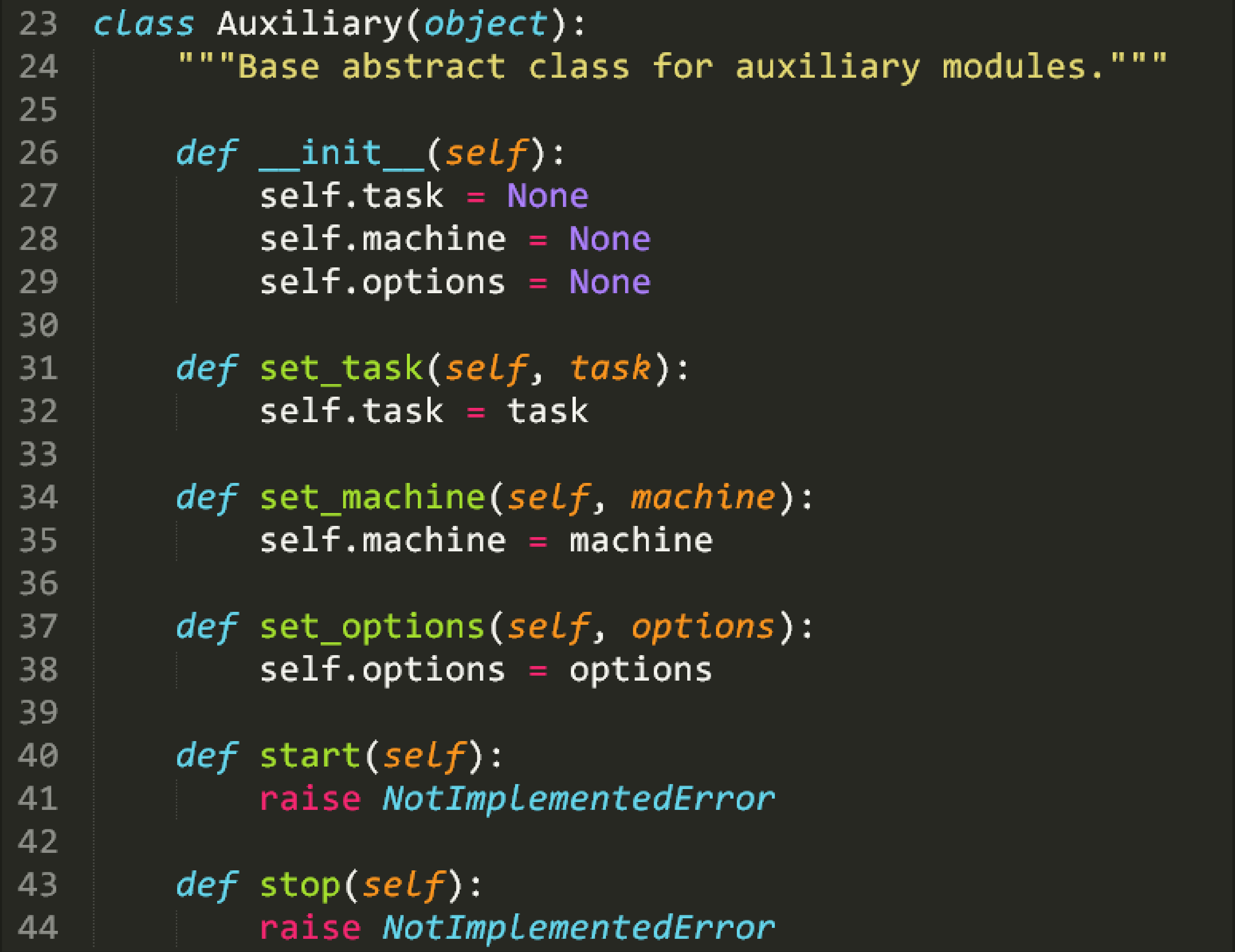
- In **Core** (under *modules/machinery/*)
- Python class
- **Define interaction** with the virtualization software
- Default:
  - VirtualBox
  - VMWare
  - QEMU/KVM
  - Generic LibVirt



```python
# Copyright (C) 2010-2013 Cuckoo Sandbox Developers.
# This file is part of Cuckoo Sandbox - http://www.cuckoosandbox.org
# See the file 'docs/LICENSE' for copying permission.

import logging

from lib.cuckoo.common.abstracts import LibVirtMachinery

class KVM(LibVirtMachinery):
    """Virtualization layer for KVM based on python-libvirt."""

    # Set KVM connection string.
    dsn = "qemu:///system"

```

AUXILIARY MODULES

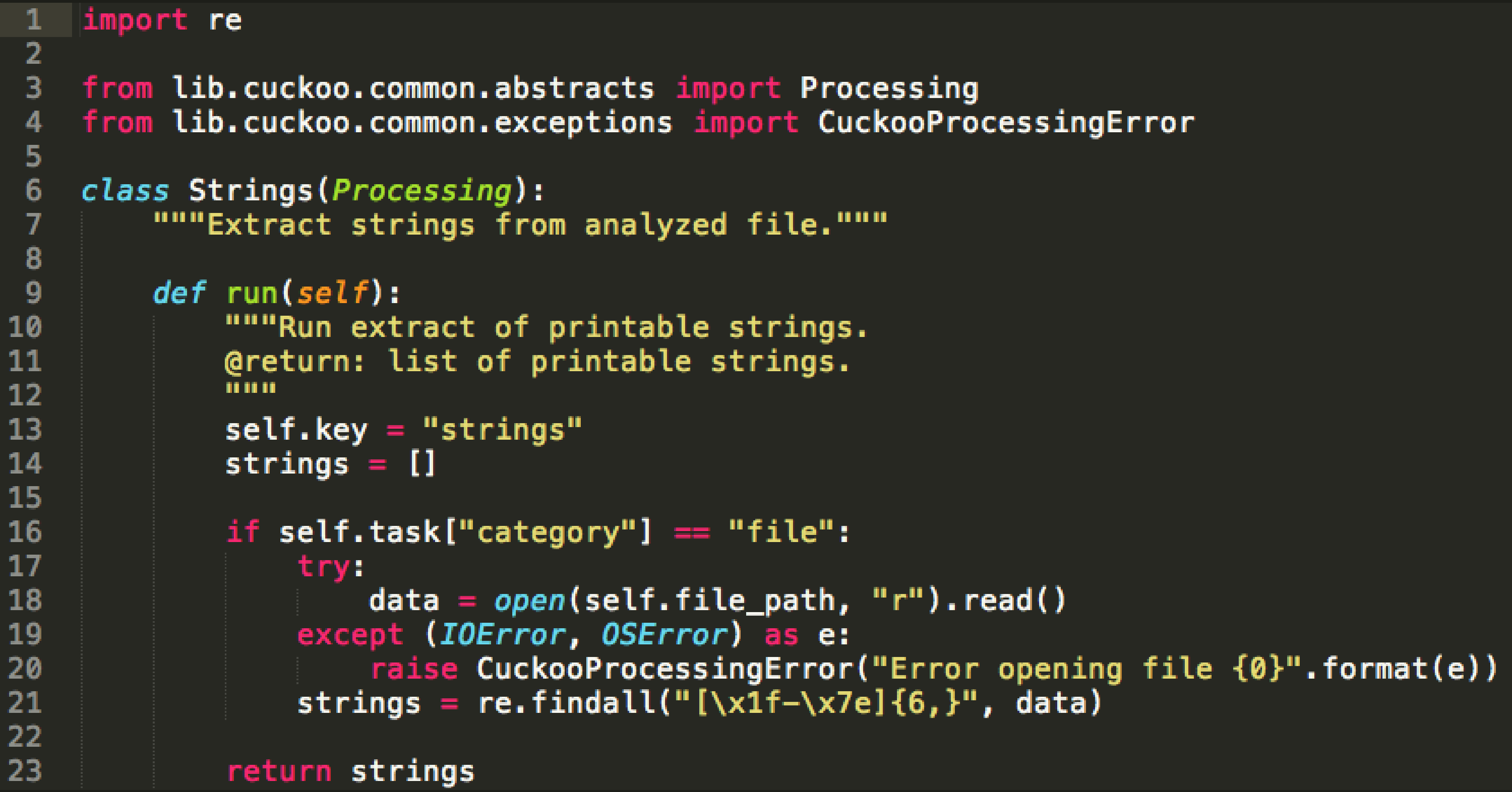
- In **Core** (under *modules/auxiliary/*)
- Python class
- No specific use, just **run concurrently** to each analysis.
- Default:
  - Network traffic capture



```python
class Auxiliary(object):
    """Base abstract class for auxiliary modules."""

    def __init__(self):
        self.task = None
        self.machine = None
        self.options = None

    def set_task(self, task):
        self.task = task

    def set_machine(self, machine):
        self.machine = machine

    def set_options(self, options):
        self.options = options

    def start(self):
        raise NotImplementedError

    def stop(self):
        raise NotImplementedError
```

Processing Modules

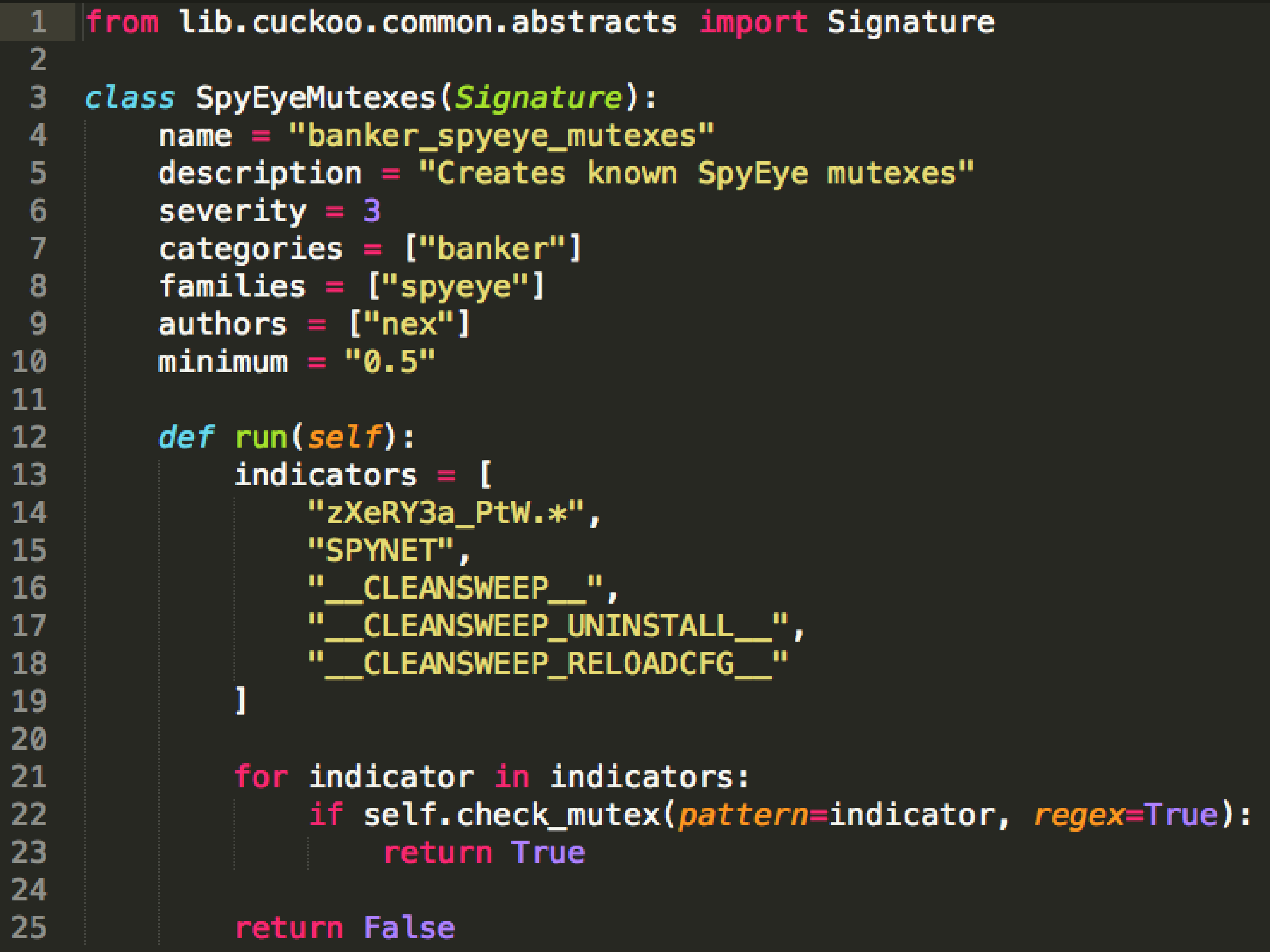
- In **Core** (under *modules/processing/*)
- Python class
- Process **raw results** (sample, API logs, files, memory)
- Populate **collection** of results



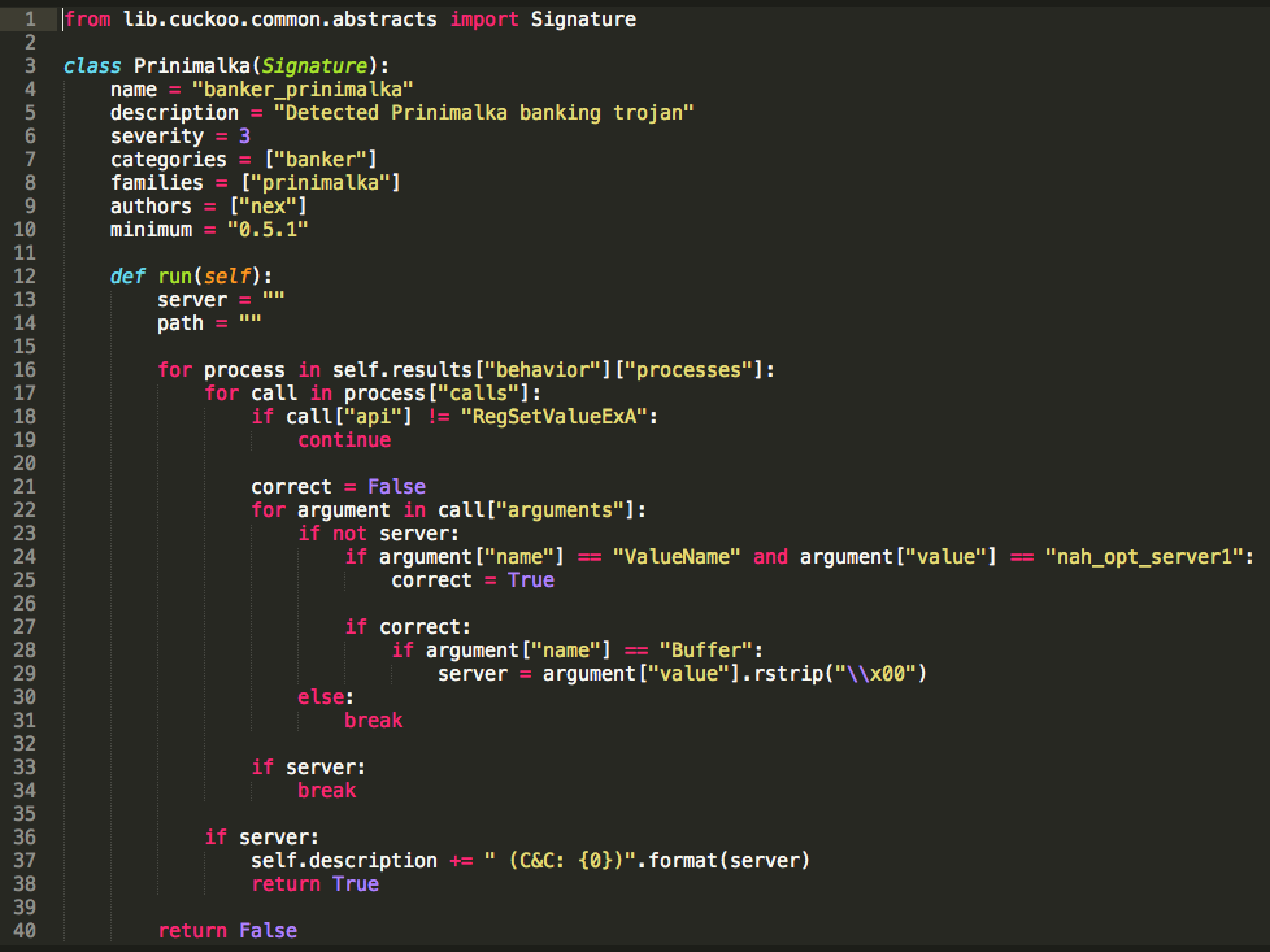
```python
import re

from lib.cuckoo.common.abstracts import Processing
from lib.cuckoo.common.exceptions import CuckooProcessingError

class Strings(Processing):
    """Extract strings from analyzed file."""

    def run(self):
        """Run extract of printable strings.
        @return: list of printable strings.
        """
        self.key = "strings"
        strings = []

        if self.task["category"] == "file":
            try:
                data = open(self.file_path, "r").read()
            except (IOError, OSError) as e:
                raise CuckooProcessingError("Error opening file {0}".format(e))
            strings = re.findall("[\x1f-\x7e]{6,}", data)

        return strings
```

SIGNATURES

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



```python
from lib.cuckoo.common.abstracts import Signature

class SpyEyeMutexes(Signature):
    name = "banker_spyeye_mutexes"
    description = "Creates known SpyEye mutexes"
    severity = 3
    categories = ["banker"]
    families = ["spyeye"]
    authors = ["nex"]
    minimum = "0.5"

    def run(self):
        indicators = [
            "zXeRY3a_PtW.*",
            "SPYNET",
            "__CLEANSWEEP__",
            "__CLEANSWEEP_UNINSTALL__",
            "__CLEANSWEEP_RELOADCFG__"
        ]

        for indicator in indicators:
            if self.check_mutex(pattern=indicator, regex=True):
                return True

        return False
```



```python
from lib.cuckoo.common.abstracts import Signature

class Prinimalka(Signature):
    name = "banker_prinimalka"
    description = "Detected Prinimalka banking trojan"
    severity = 3
    categories = ["banker"]
    families = ["prinimalka"]
    authors = ["nex"]
    minimum = "0.5.1"

    def run(self):
        server = ""
        path = ""

        for process in self.results["behavior"]["processes"]:
            for call in process["calls"]:
                if call["api"] != "RegSetValueExA":
                    continue

                correct = False
                for argument in call["arguments"]:
                    if not server:
                        if argument["name"] == "ValueName" and argument["value"] == "nah_opt_server1":
                            correct = True

                        if correct:
                            if argument["name"] == "Buffer":
                                server = argument["value"].rstrip("\\x00")
                    else:
                        break

                if server:
                    break

            if server:
                self.description += " (C&C: {0})".format(server)
                return True

        return False
```

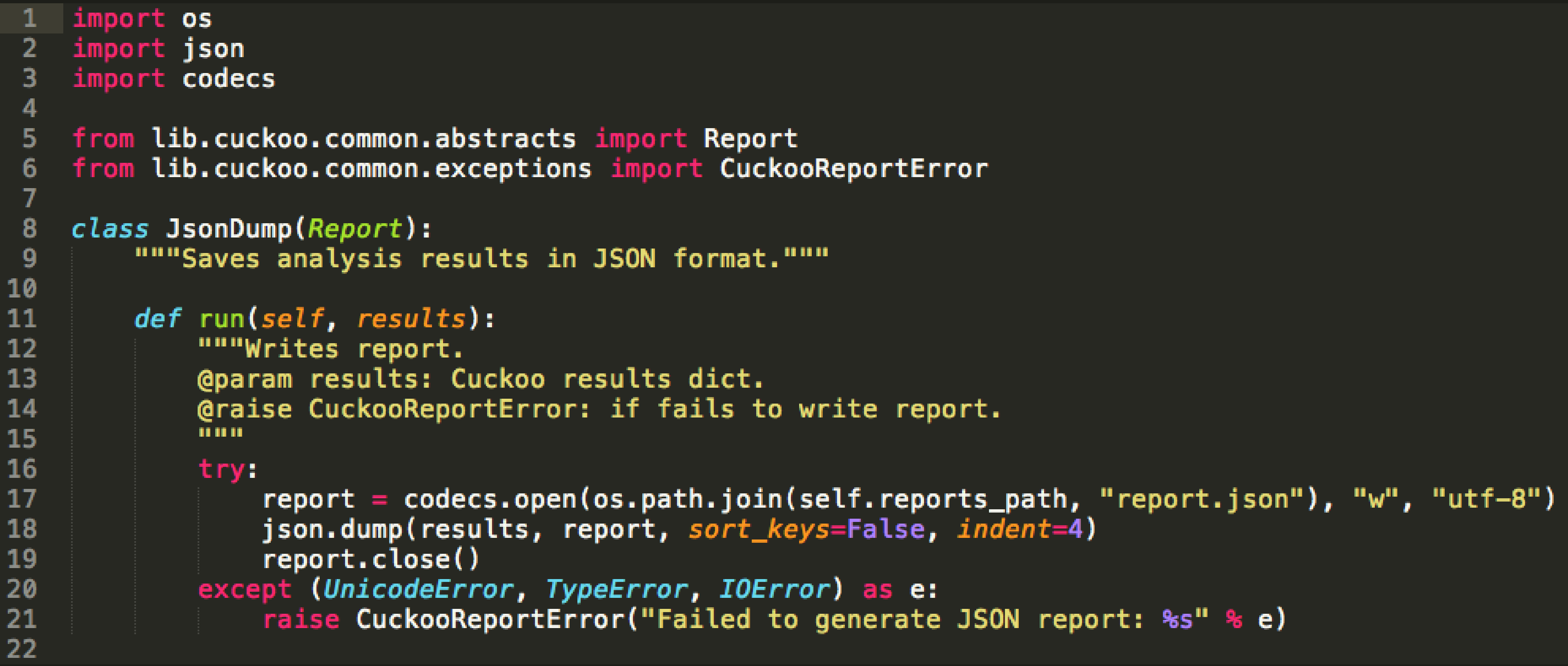
Community Signatures

- **Community Repository**
  - https://github.com/cuckoobox/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

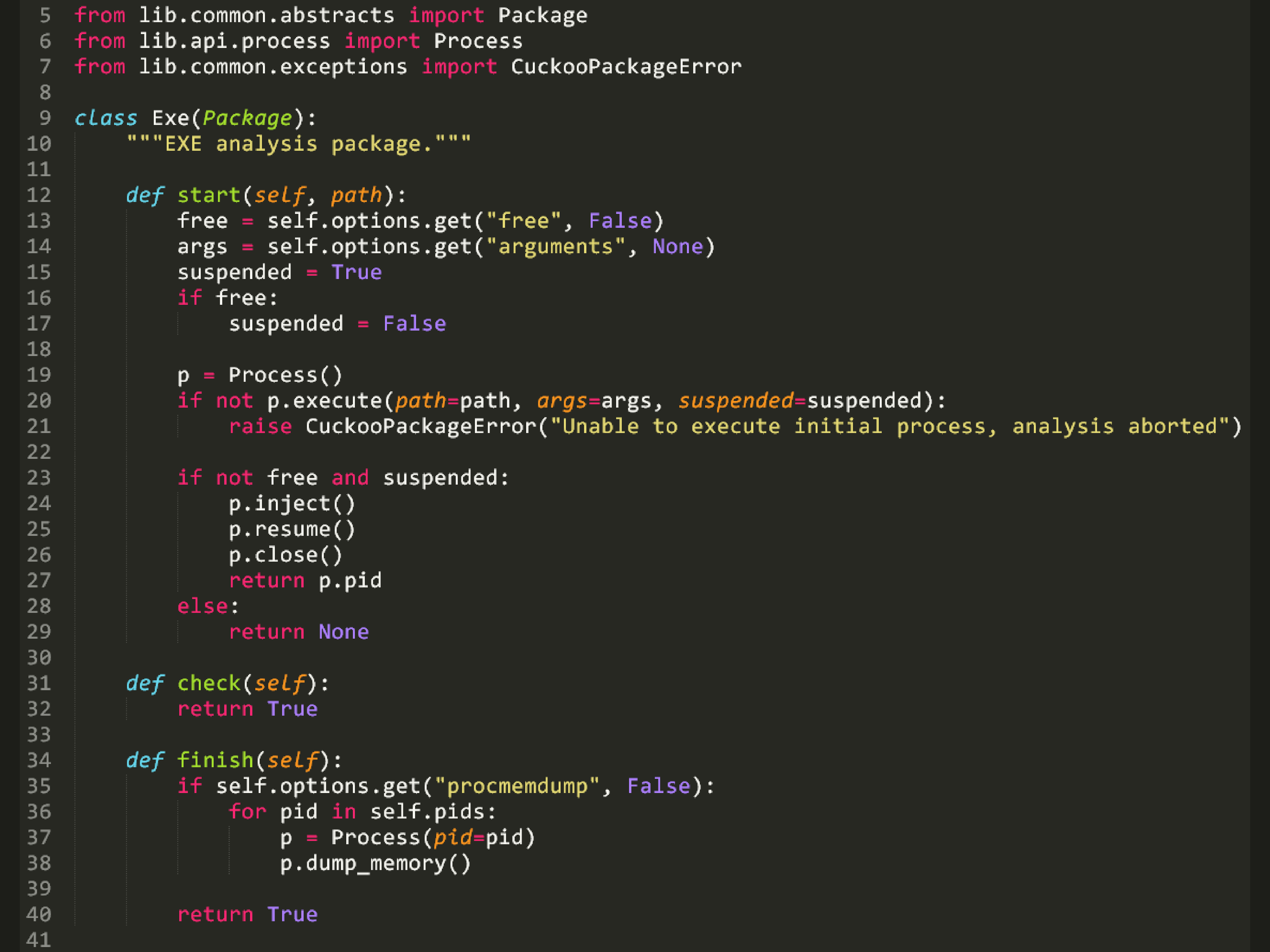


```python
import os
import json
import codecs

from lib.cuckoo.common.abstracts import Report
from lib.cuckoo.common.exceptions import CuckooReportError

class JsonDump(Report):
    """Saves analysis results in JSON format."""

    def run(self, results):
        """Writes report.
        @param results: Cuckoo results dict.
        @raise CuckooReportError: if fails to write report.
        """
        try:
            report = codecs.open(os.path.join(self.reports_path, "report.json"), "w", "utf-8")
            json.dump(results, report, sort_keys=False, indent=4)
            report.close()
        except (UnicodeError, TypeError, IOError) as e:
            raise CuckooReportError("Failed to generate JSON report: %s" % e)

```

ANALYZER MODULES

Analysis Packages

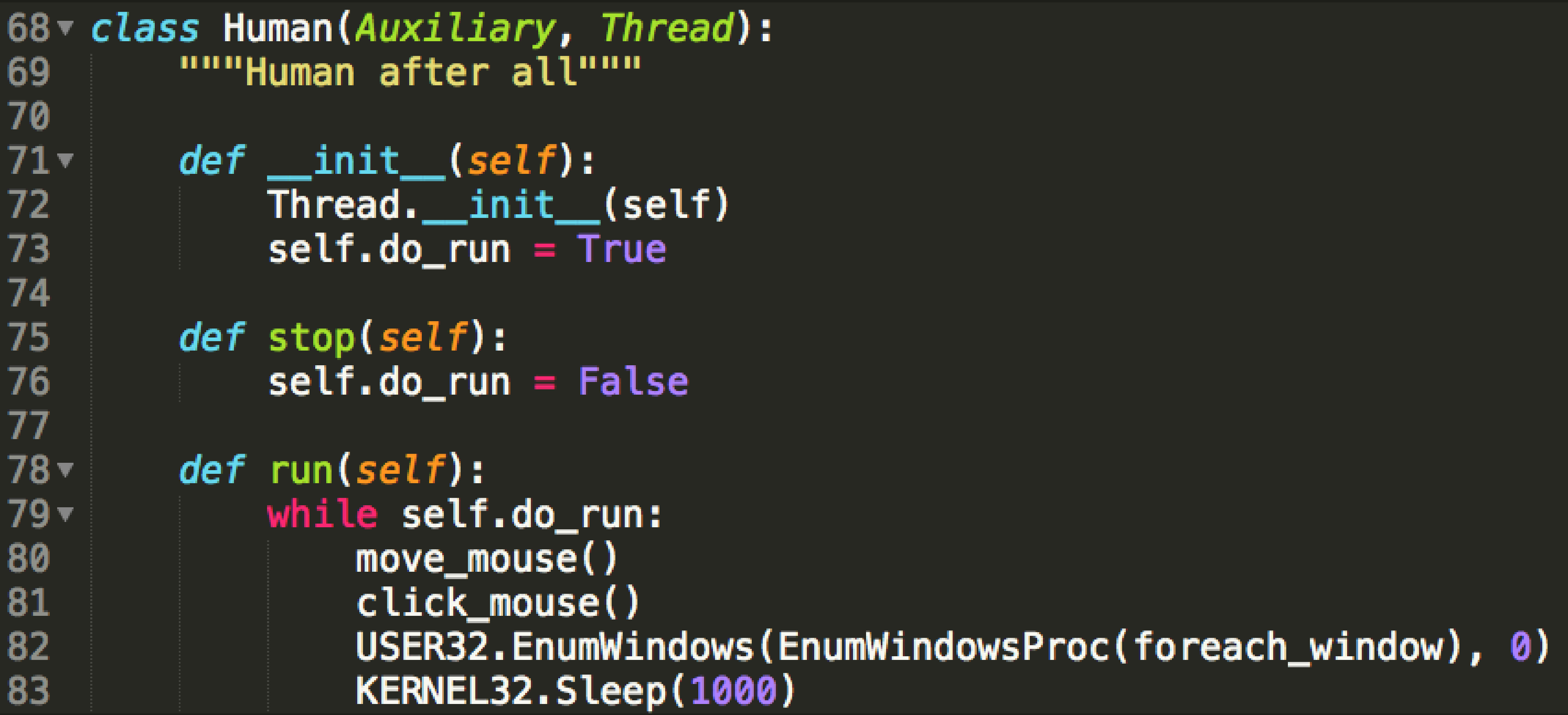
- In **Analyzer** (under *analyzer/windows/modules/packages/*)
- Python modules
- Define how to interact with the malware and the system
- Can be used for scripting tasks



```python
from lib.common.abstracts import Package
from lib.api.process import Process
from lib.common.exceptions import CuckooPackageError

class Exe(Package):
    """EXE analysis package."""

    def start(self, path):
        free = self.options.get("free", False)
        args = self.options.get("arguments", None)
        suspended = True
        if free:
            suspended = False

        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
        else:
            return None

    def check(self):
        return True

    def finish(self):
        if self.options.get("procmemdump", False):
            for pid in self.pids:
                p = Process(pid=pid)
                p.dump_memory()

        return True

```

AUXILIARY MODULES

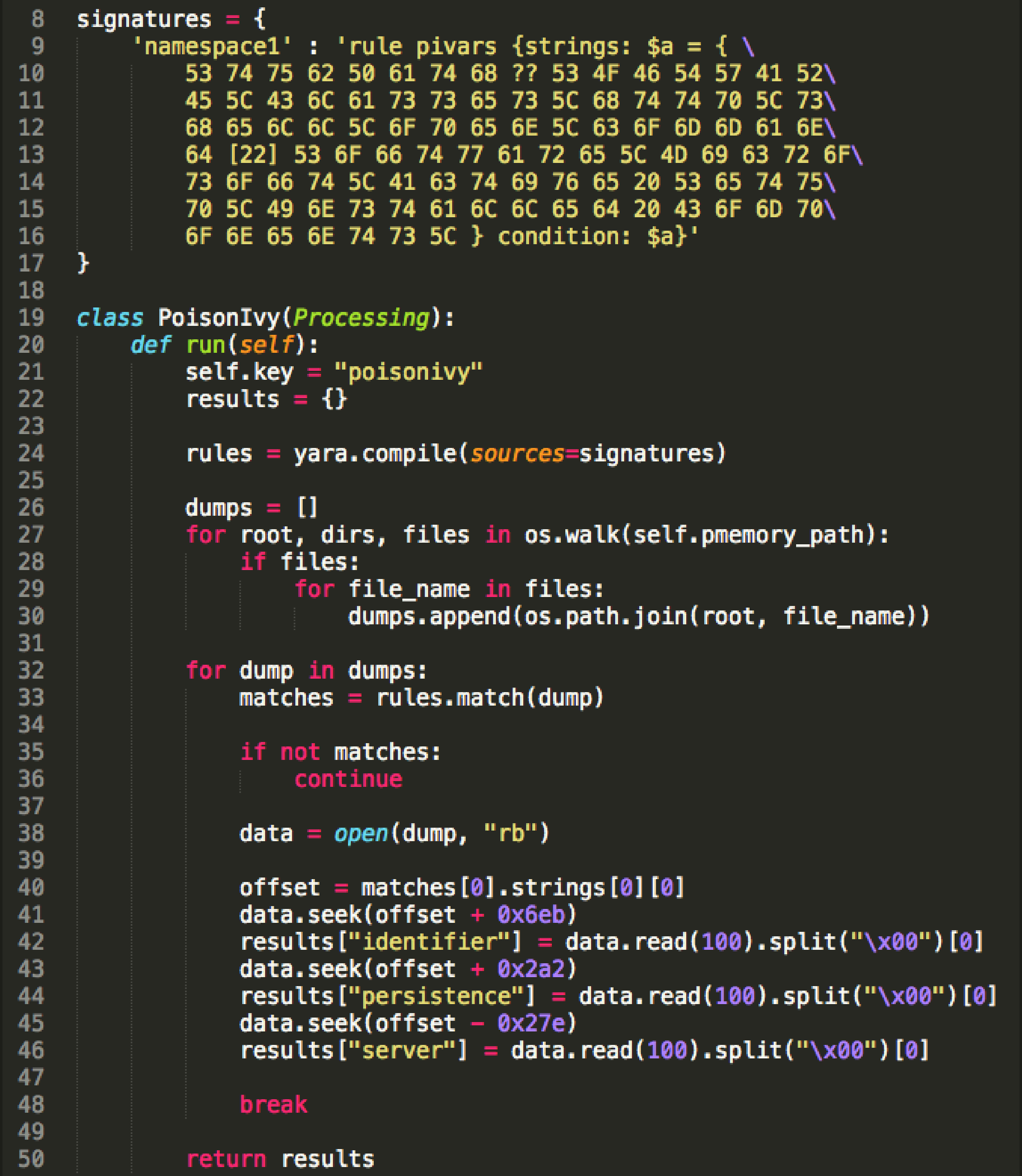
- In **Analyzer** (under *analyzer/windows/modules/auxiliaries/*)
- Python modules
- Run concurrently to the analysis
- Default:
  - Screenshots
  - Emulation of human interaction



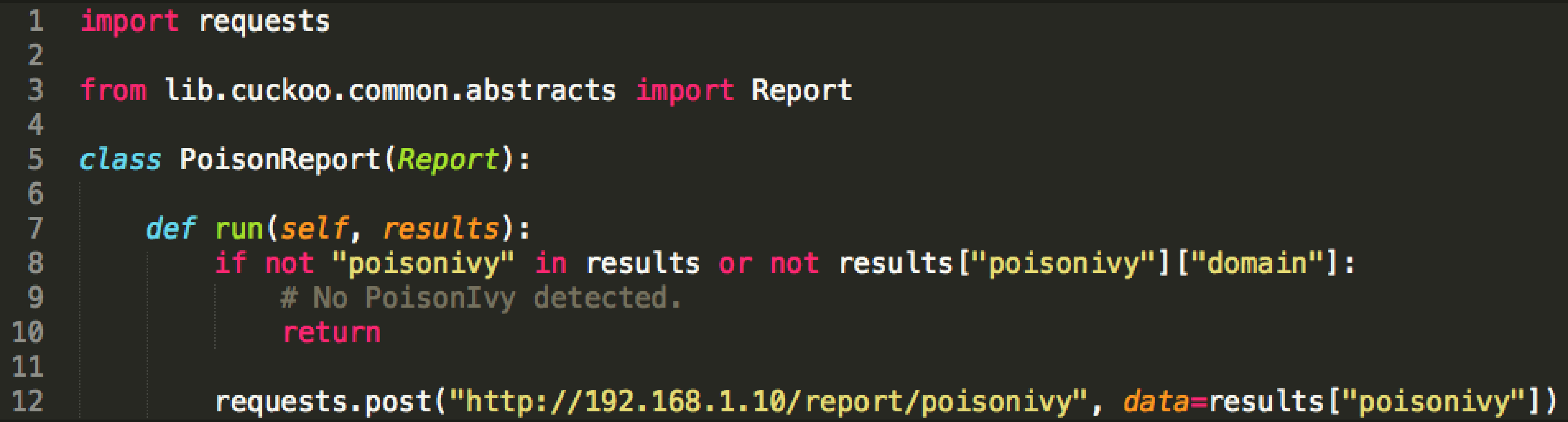
```python
class Human(Auxiliary, Thread):
    """Human after all"""

    def __init__(self):
        Thread.__init__(self)
        self.do_run = True

    def stop(self):
        self.do_run = False

    def run(self):
        while self.do_run:
            move_mouse()
            click_mouse()
            USER32.EnumWindows(EnumWindowsProc(foreach_window), 0)
            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



```python
signatures = {
    'namespace1' : 'rule pivars {strings: $a = { \
        53 74 75 62 50 61 74 68 ?? 53 4F 46 54 57 41 52\
        45 5C 43 6C 61 73 73 65 73 5C 68 74 74 70 5C 73\
        68 65 6C 6C 5C 6F 70 65 6E 5C 63 6F 6D 6D 61 6E\
        64 [22] 53 6F 66 74 77 61 72 65 5C 4D 69 63 72 6F\
        73 6F 66 74 5C 41 63 74 69 76 65 20 53 65 74 75\
        70 5C 49 6E 73 74 61 6C 6C 65 64 20 43 6F 6D 70\
        6F 6E 65 6E 74 73 5C } condition: $a}'
}

class PoisonIvy(Processing):
    def run(self):
        self.key = "poisonivy"
        results = {}

        rules = yara.compile(sources=signatures)

        dumps = []
        for root, dirs, files in os.walk(self.pmemory_path):
            if files:
                for file_name in files:
                    dumps.append(os.path.join(root, file_name))

        for dump in dumps:
            matches = rules.match(dump)

            if not matches:
                continue

            data = open(dump, "rb")

            offset = matches[0].strings[0][0]
            data.seek(offset + 0x6eb)
            results["identifier"] = data.read(100).split("\x00")[0]
            data.seek(offset + 0x2a2)
            results["persistence"] = data.read(100).split("\x00")[0]
            data.seek(offset - 0x27e)
            results["server"] = data.read(100).split("\x00")[0]

            break

        return results
```



```python
import requests

from lib.cuckoo.common.abstracts import Report

class PoisonReport(Report):

    def run(self, results):
        if not "poisonivy" in results or not results["poisonivy"]["domain"]:
            # No PoisonIvy detected.
            return

        requests.post("http://192.168.1.10/report/poisonivy", data=results["poisonivy"])
```

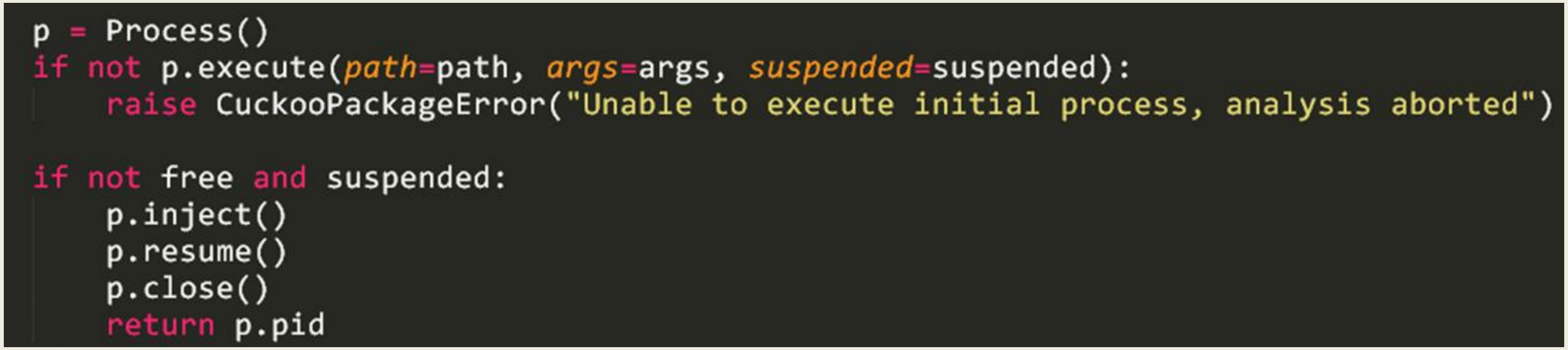
CUCKOOMON

CuckooMon

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

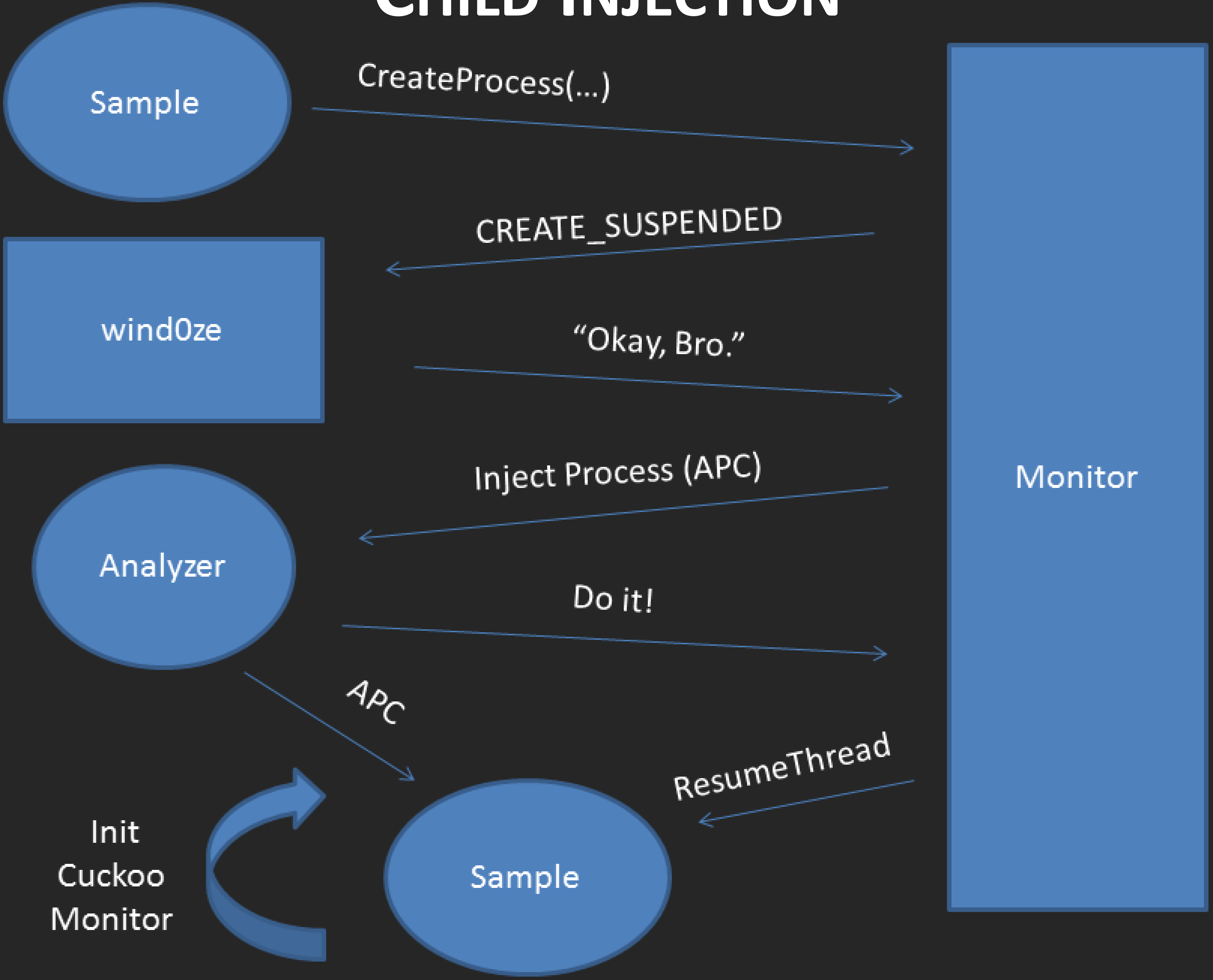
ANALYZER PACKAGE

- 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



Sample

CreateProcess(...)

wind0ze

CREATE_SUSPENDED

"Okay, Bro."

Analyzer

Inject Process (APC)

Do it!

APC

Monitor

ResumeThread

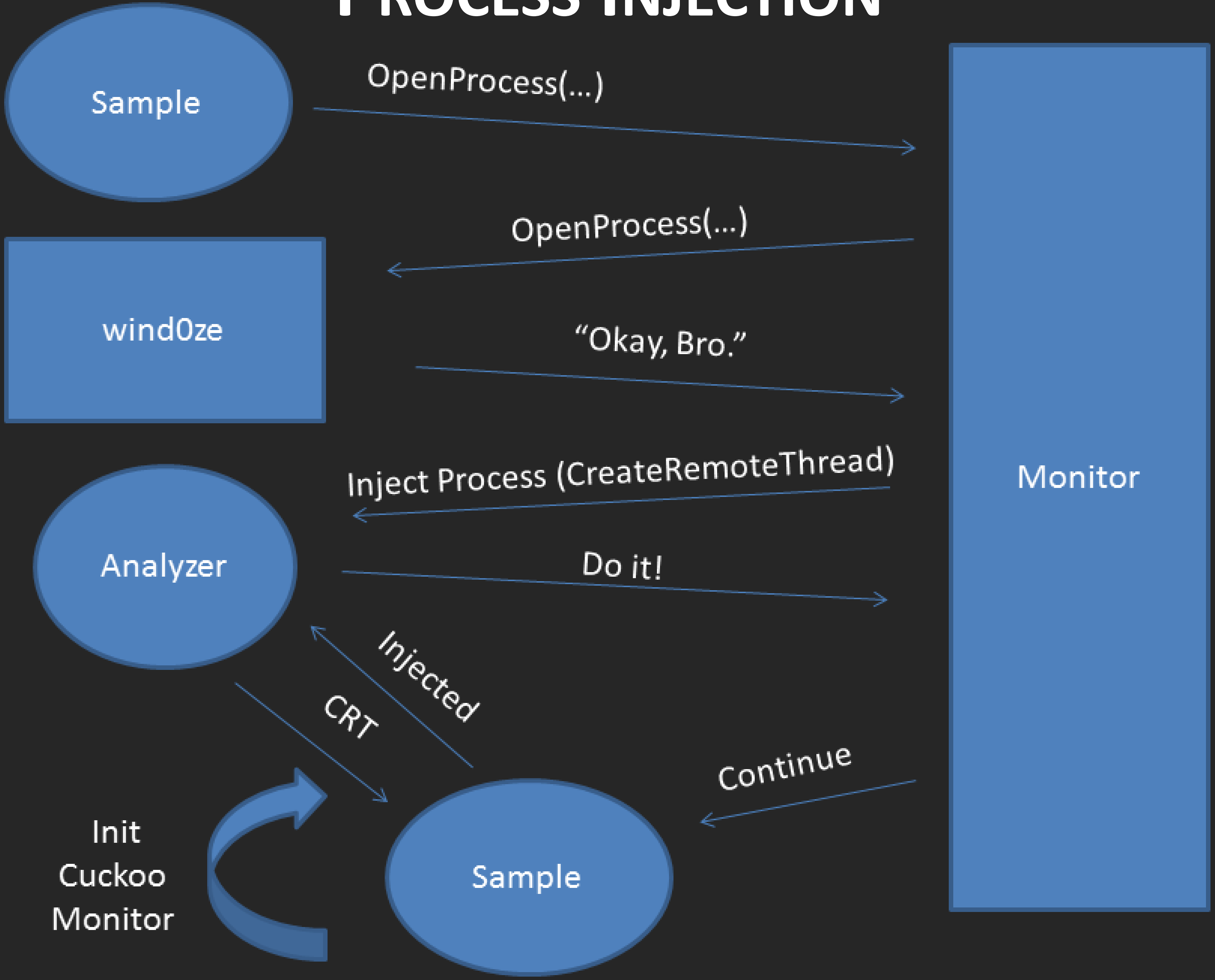
Init
Cuckoo
Monitor

Sample

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

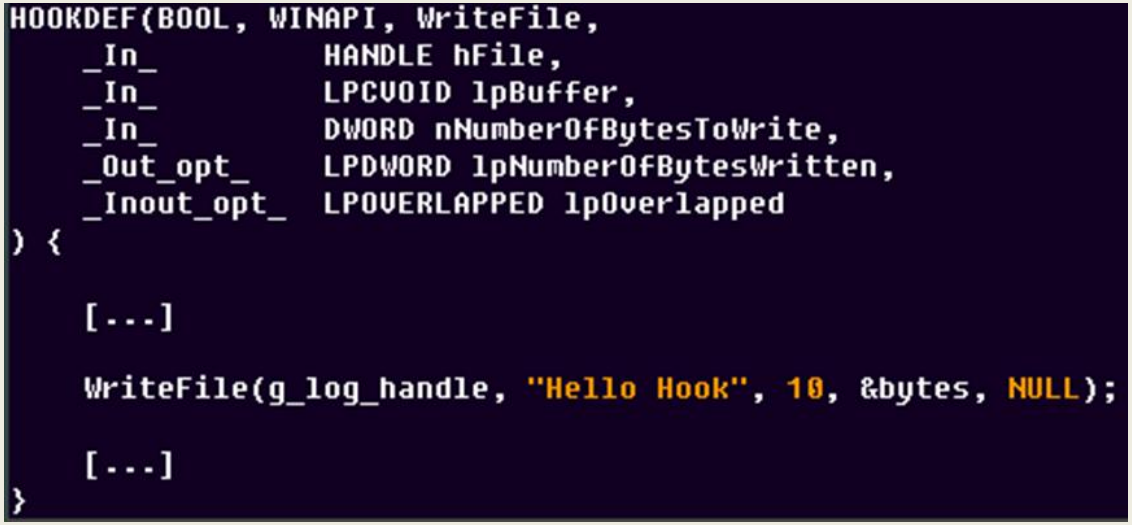


API Hooking Overview

- 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

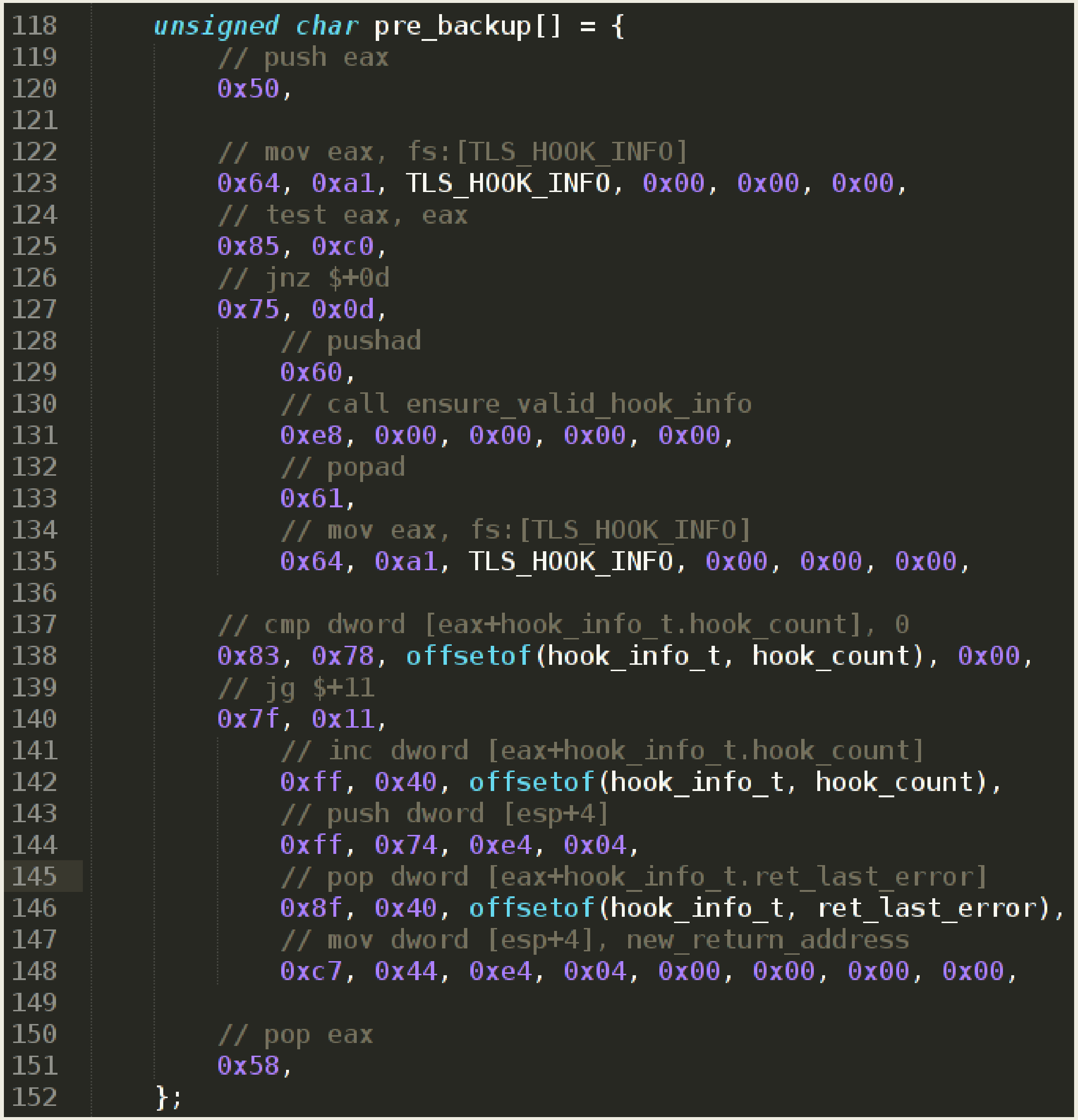
- Use standard **inline hooking** with a few twists
  - Support for **random preambles** (jmp/push+ret/etc)



```
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);

    [...]
}
```

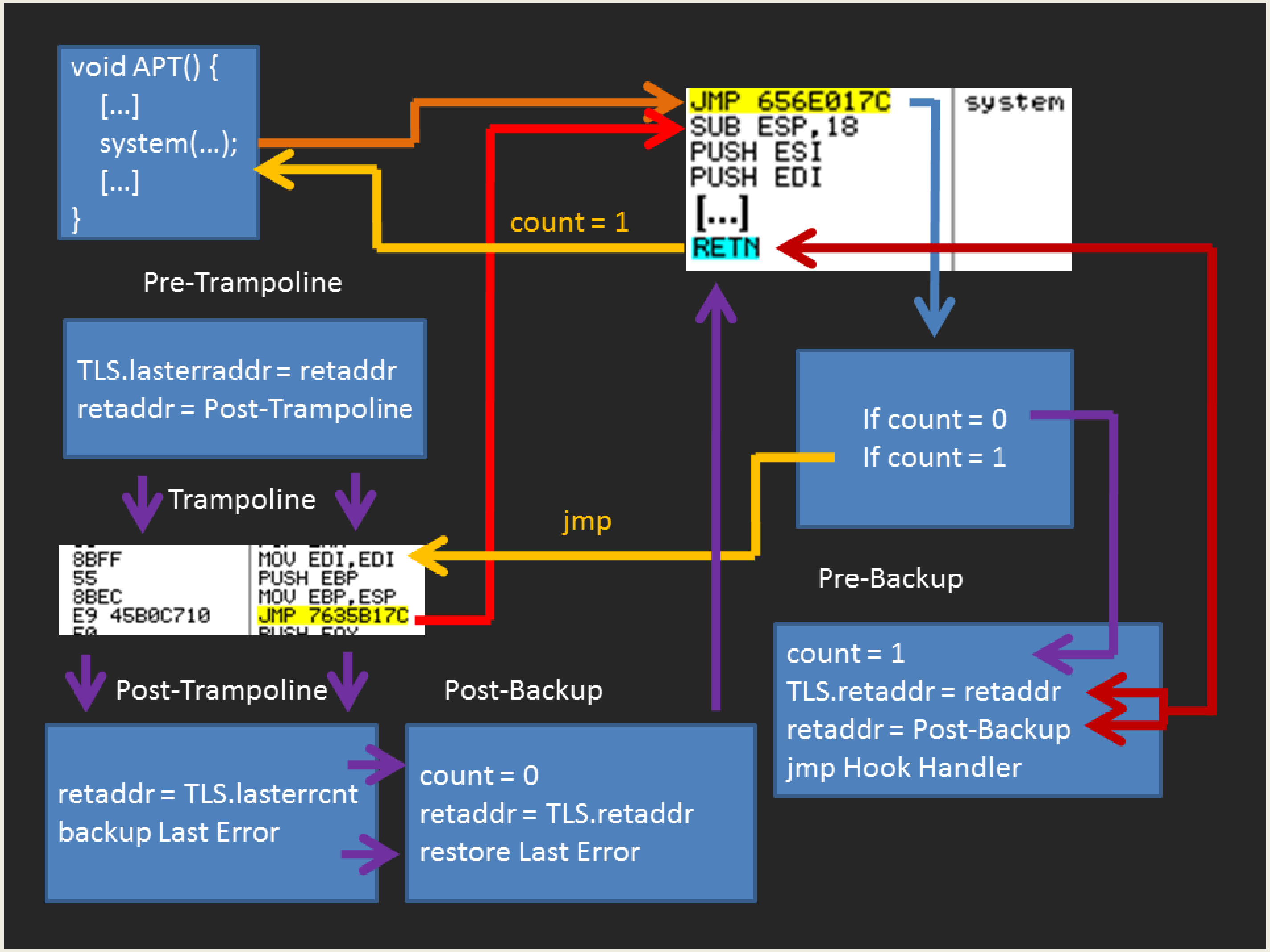
- First hook run is interesting, **ignore recursive ones** down on the callstack
- Transparently manage these situations in hooking mechanism

Assembly Trampolines



```
    unsigned char pre_backup[] = {
        // push eax
        0x50,

        // mov eax, fs:[TLS_HOOK_INFO]
        0x64, 0xa1, TLS_HOOK_INFO, 0x00, 0x00, 0x00,
        // test eax, eax
        0x85, 0xc0,
        // jnz $+0d
        0x75, 0x0d,
            // pushad
            0x60,
            // call ensure_valid_hook_info
            0xe8, 0x00, 0x00, 0x00, 0x00,
            // popad
            0x61,
            // mov eax, fs:[TLS_HOOK_INFO]
            0x64, 0xa1, TLS_HOOK_INFO, 0x00, 0x00, 0x00,

        // cmp dword [eax+hook_info_t.hook_count], 0
        0x83, 0x78, offsetof(hook_info_t, hook_count), 0x00,
        // jg $+11
        0x7f, 0x11,
            // inc dword [eax+hook_info_t.hook_count]
            0xff, 0x40, offsetof(hook_info_t, hook_count),
            // push dword [esp+4]
            0xff, 0x74, 0xe4, 0x04,
            // pop dword [eax+hook_info_t.ret_last_error]
            0x8f, 0x40, offsetof(hook_info_t, ret_last_error),
            // mov dword [esp+4], new_return_address
            0xc7, 0x44, 0xe4, 0x04, 0x00, 0x00, 0x00, 0x00,

        // pop eax
        0x58,
    };
```

Resulting Hooks



```
void APT() {
    [...]
    system(...);
    [...]
}
```

```
JMP 656E017C
SUB ESP,18
PUSH ESI
PUSH EDI
[...]
RETN
```

count = 1

Pre-Trampoline

TLS.lasterraddr = retaddr
retaddr = Post-Trampoline

Trampoline

```
8BFF          MOV EDI,EDI
55            PUSH EBP
8BEC          MOV EBP,ESP
E9 45B0C710   JMP 7635B17C
```

jmp

If count = 0
If count = 1

Pre-Backup

count = 1
TLS.retaddr = retaddr
retaddr = Post-Backup
jmp Hook Handler

Post-Trampoline

retaddr = TLS.lasterrcnt
backup Last Error

Post-Backup

count = 0
retaddr = TLS.retaddr
restore Last Error

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)

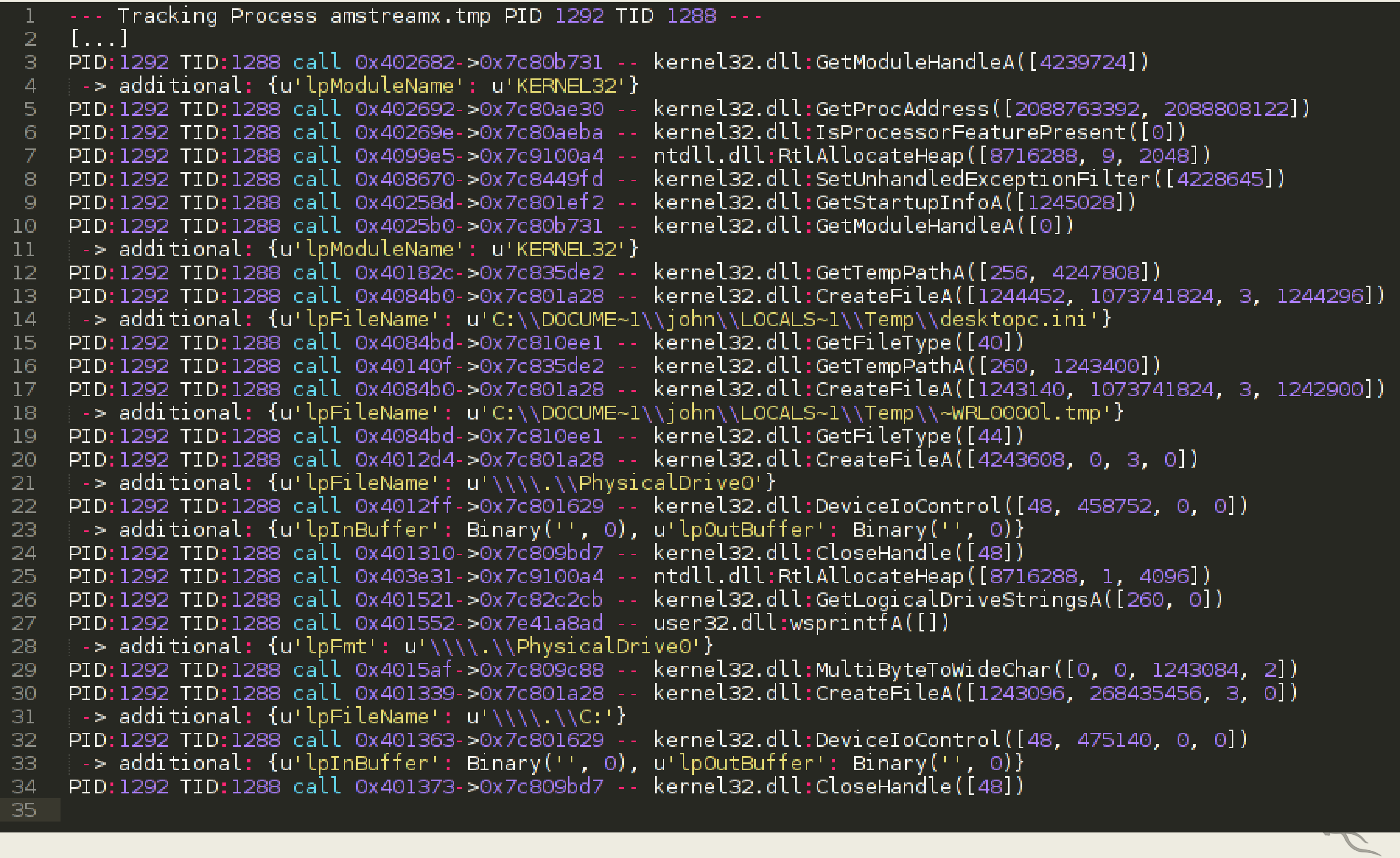
Automated Logging cont.

- 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



```
--- Tracking Process amstreamx.tmp PID 1292 TID 1288 ---
[...]
PID:1292 TID:1288 call 0x402682->0x7c80b731 -- kernel32.dll:GetModuleHandleA([4239724])
 -> additional: {u'lpModuleName': u'KERNEL32'}
PID:1292 TID:1288 call 0x402692->0x7c80ae30 -- kernel32.dll:GetProcAddress([2088763392, 2088808122])
PID:1292 TID:1288 call 0x40269e->0x7c80aeba -- kernel32.dll:IsProcessorFeaturePresent([0])
PID:1292 TID:1288 call 0x4099e5->0x7c9100a4 -- ntdll.dll:RtlAllocateHeap([8716288, 9, 2048])
PID:1292 TID:1288 call 0x408670->0x7c8449fd -- kernel32.dll:SetUnhandledExceptionFilter([4228645])
PID:1292 TID:1288 call 0x40258d->0x7c801ef2 -- kernel32.dll:GetStartupInfoA([1245028])
PID:1292 TID:1288 call 0x4025b0->0x7c80b731 -- kernel32.dll:GetModuleHandleA([0])
 -> additional: {u'lpModuleName': u'KERNEL32'}
PID:1292 TID:1288 call 0x40182c->0x7c835de2 -- kernel32.dll:GetTempPathA([256, 4247808])
PID:1292 TID:1288 call 0x4084b0->0x7c801a28 -- kernel32.dll:CreateFileA([1244452, 1073741824, 3, 1244296])
 -> additional: {u'lpFileName': u'C:\\DOCUME~1\\john\\LOCALS~1\\Temp\\desktopc.ini'}
PID:1292 TID:1288 call 0x4084bd->0x7c810ee1 -- kernel32.dll:GetFileType([40])
PID:1292 TID:1288 call 0x40140f->0x7c835de2 -- kernel32.dll:GetTempPathA([260, 1243400])
PID:1292 TID:1288 call 0x4084b0->0x7c801a28 -- kernel32.dll:CreateFileA([1243140, 1073741824, 3, 1242900])
 -> additional: {u'lpFileName': u'C:\\DOCUME~1\\john\\LOCALS~1\\Temp\\~WRL0000l.tmp'}
PID:1292 TID:1288 call 0x4084bd->0x7c810ee1 -- kernel32.dll:GetFileType([44])
PID:1292 TID:1288 call 0x4012d4->0x7c801a28 -- kernel32.dll:CreateFileA([4243608, 0, 3, 0])
 -> additional: {u'lpFileName': u'\\\\.\\PhysicalDrive0'}
PID:1292 TID:1288 call 0x4012ff->0x7c801629 -- kernel32.dll:DeviceIoControl([48, 458752, 0, 0])
 -> additional: {u'lpInBuffer': Binary('', 0), u'lpOutBuffer': Binary('', 0)}
PID:1292 TID:1288 call 0x401310->0x7c809bd7 -- kernel32.dll:CloseHandle([48])
PID:1292 TID:1288 call 0x403e31->0x7c9100a4 -- ntdll.dll:RtlAllocateHeap([8716288, 1, 4096])
PID:1292 TID:1288 call 0x401521->0x7c82c2cb -- kernel32.dll:GetLogicalDriveStringsA([260, 0])
PID:1292 TID:1288 call 0x401552->0x7e41a8ad -- user32.dll:wsprintfA([])
 -> additional: {u'lpFmt': u'\\\\.\\PhysicalDrive0'}
PID:1292 TID:1288 call 0x4015af->0x7c809c88 -- kernel32.dll:MultiByteToWideChar([0, 0, 1243084, 2])
PID:1292 TID:1288 call 0x401339->0x7c801a28 -- kernel32.dll:CreateFileA([1243096, 268435456, 3, 0])
 -> additional: {u'lpFileName': u'\\\\.\\C:'}
PID:1292 TID:1288 call 0x401363->0x7c801629 -- kernel32.dll:DeviceIoControl([48, 475140, 0, 0])
 -> additional: {u'lpInBuffer': Binary('', 0), u'lpOutBuffer': Binary('', 0)}
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/cuckoobox/vxcage

?

www.cuckoosandbox.org

@cuckoosandbox