
cuckoo

INTRODUCTION

HERE

- **Claudio** "nex" **Guarnieri** @botherder
  - Security Researcher at **Rapid7**
  - Core member of **The Shadowserver Foundation**
  - Core member of **The Honeynet Project**
  - Dictator of **Cuckoo Sandbox**

Not Here

- **Mark** “rep” **Schloesser** @repmovsb
  - Here?
  - German coding machine
- **Jurriaan** “skier” **Bremer** @skier_t
  - Our Dutch Windows wizard
- **Alessandro** “jekil” **Tanasi** @jekil
  - Italian Ferrari

SANDBOXING

PROBLEMS

- Process high volumes?
- Automate specific tasks?
- Integrate with internal security?
- Support your tier-1 analysts?

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

PREPARATION

- Define **requirements** and **expectations**
  - **Goal**
  - Throughput
- Design the analysis environment
- Design proper integration
  - **Make sense of the data!**

CUCKOO SANDBOX

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

Free as in Freedom

Powered by **RAPID7**

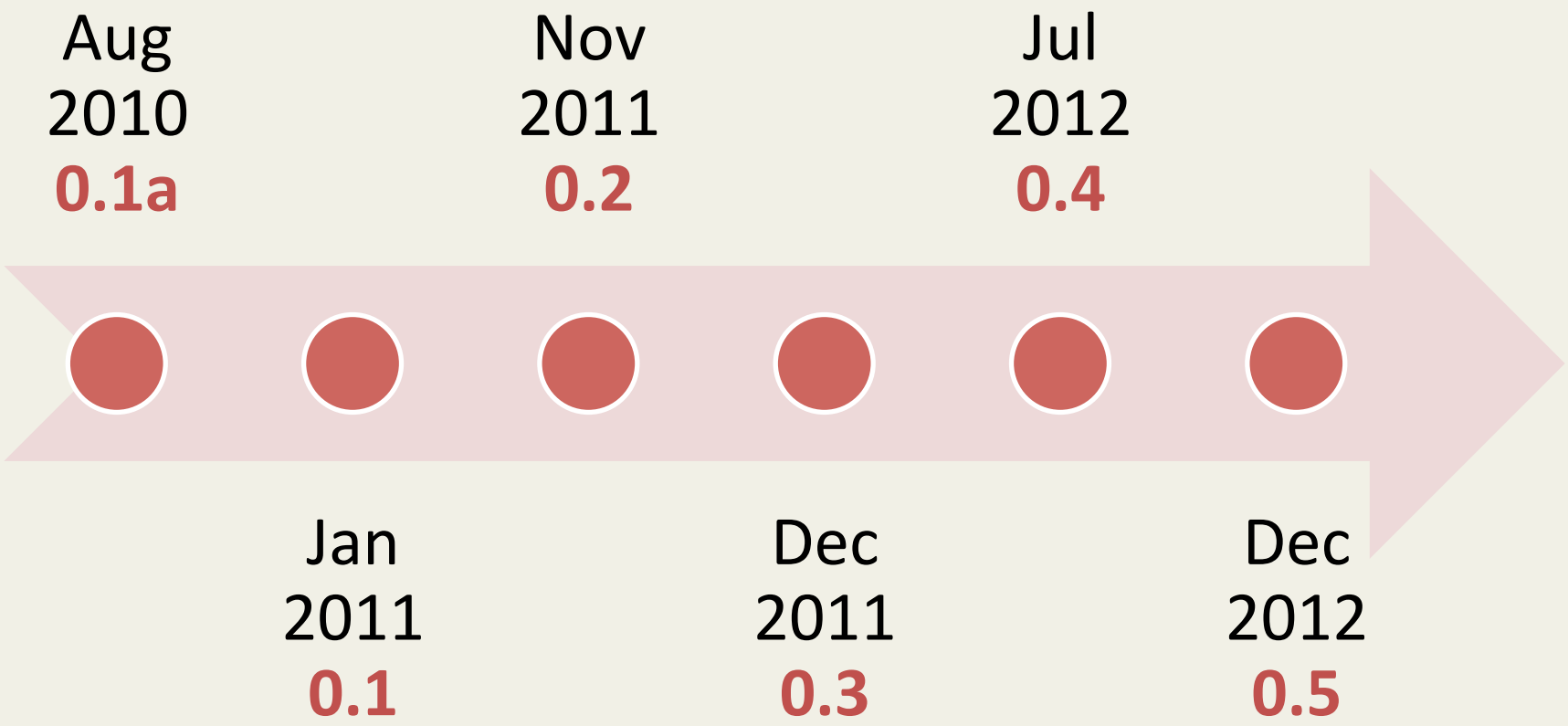
WHY?

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

Some numbers

- Almost **50000** lines of code, **Python** and **C**
- **4** core developers
- ~**25** contributors over time
- ~**8000** downloads of the last version

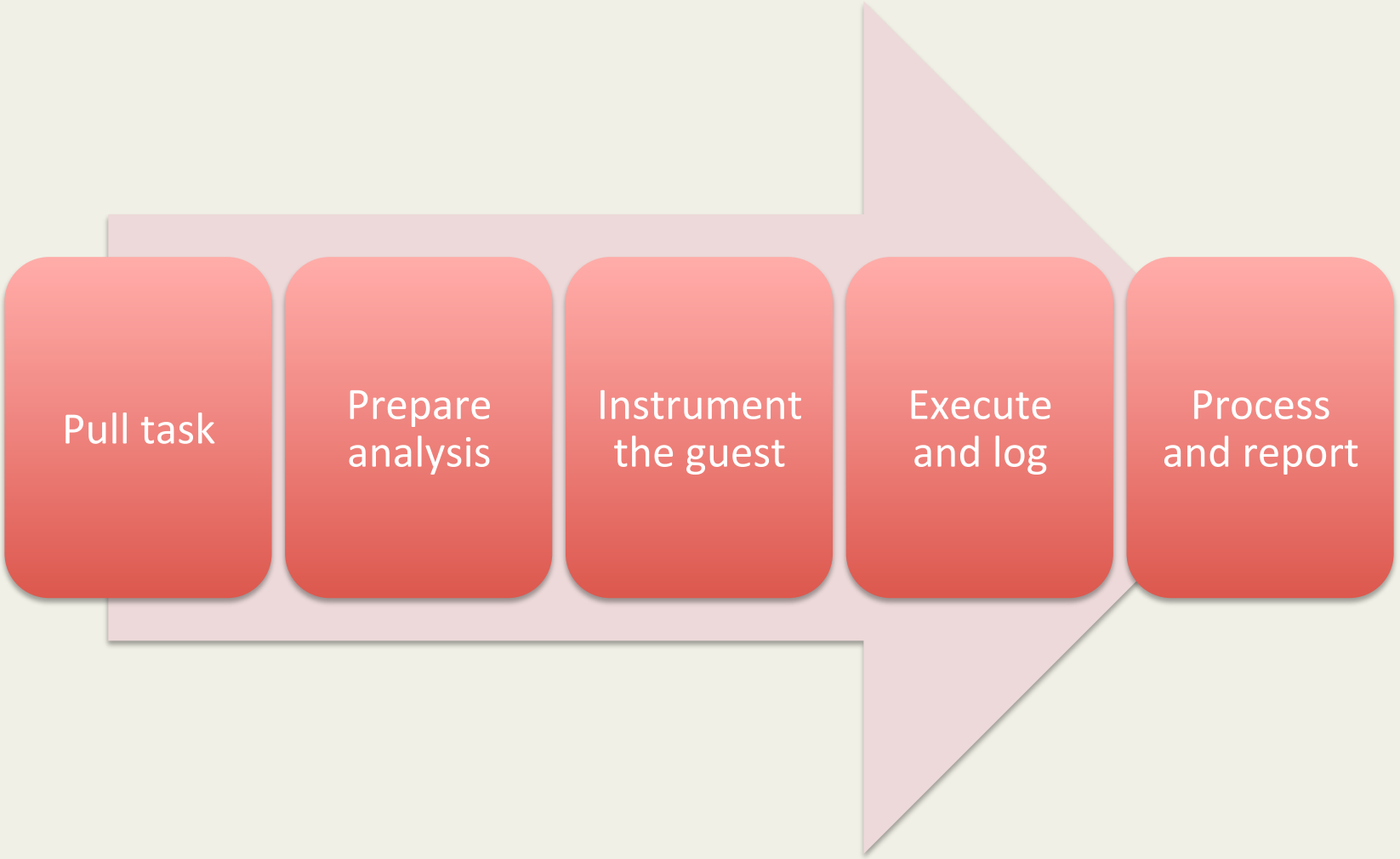
Bits of History



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!

How it works


Pull task
Prepare analysis
Instrument the guest
Execute and log
Process and report

KEY FEATURES

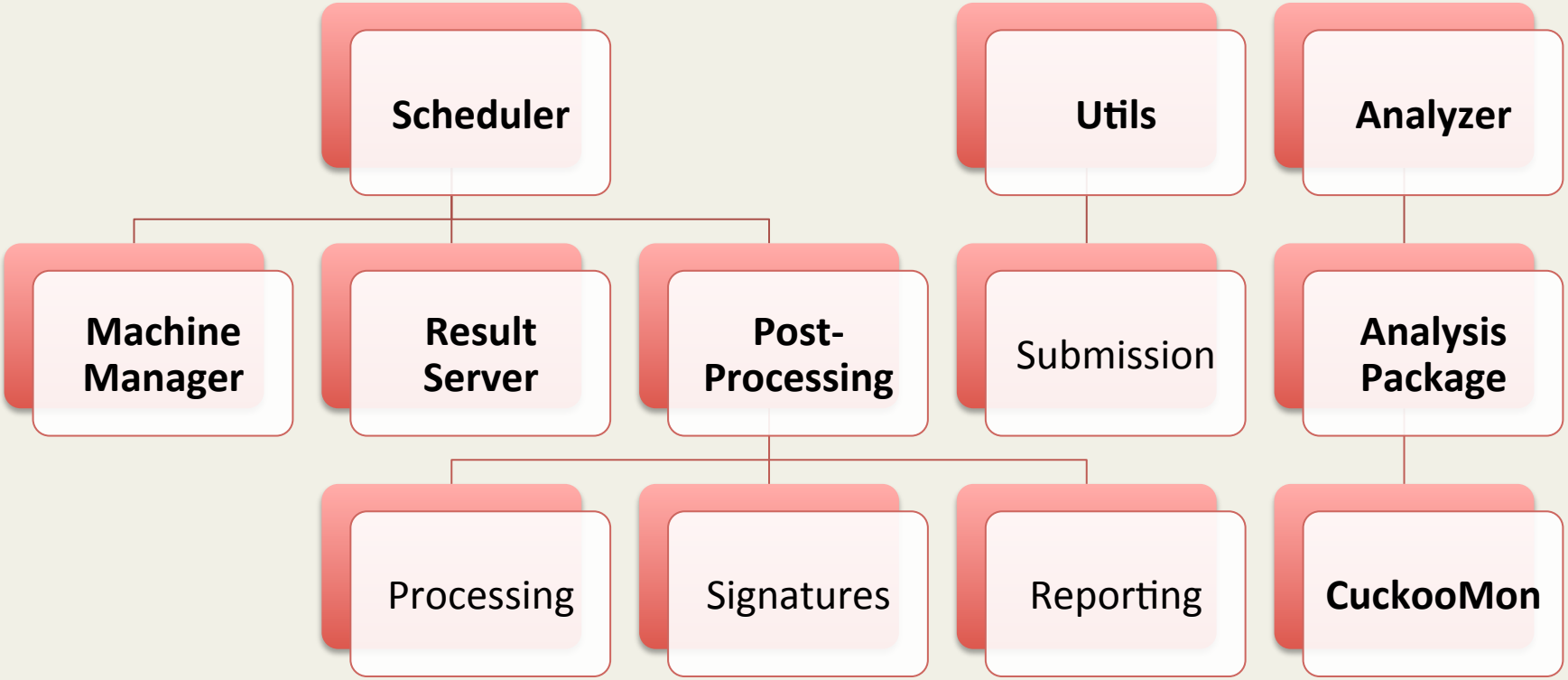
- Completely automated
- Run **concurrent** analysis
- Able to **trace** processes recursively
- **Customize analysis** process
- Create behavioral **signatures**
- Customize processing and reporting

RESULTS

- Behavioral Logs
- File dumps
- Screenshots
- Network traffic
- Memory dumps

DEMO

COMPONENTS


Scheduler
Utils
Analyzer
Machine Manager
Result Server
Post-Processing
Submission
Analysis Package
Processing
Signatures
Reporting
CuckooMon

SUBMISSION

- Python API
- Command-line utility
- Web utility
- REST API
- Options:
  - Priority
  - Timeout
  - Machine
  - Package
  - Arguments
  - Memory dump

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

Helper Functions

- Create process
- Monitor process status
- Inject DLL
- Take process memory dump

```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()
            return p.pid
        else:
            return None

    def check(self):
        return True

    def finish(self):
        return True
```

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)
```

Processing Modules

- In **Core** (under *modules/processing/*)
- Python modules
- Process raw results
- Populate collection of abstracted 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 modules
- 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
```

DEMO

Reporting Modules

- In **Core** (under *analyzer/windows/modules/reporting/*)
- Python modules
- 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)

```

COMMUNITY

- **Community Repository**
  - https://github.com/cuckoobox/community
- ***utils/community.py***

Use Case

- **APT! APT!** APT!
- Automatically collect and analyze PoisonIvy
- Extract configurations
- Report PoisonIvy C&C to a backend

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

class ProcDump(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()
            return p.pid
        else:
            return None

    def check(self):
        return True

    def finish(self):
        for pid in self.pids:
            p = Process(pid=pid)
            p.dump_memory()

        return True
```

```
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"])
```

DEMO

CONCLUSIONS

JUICY IDEAS

- Automate extraction of **bankers** configs
- Automate extraction of **RAT** configs ✔
- Automate **process memory** forensic
- Automate **unpacking**
- **Any others?**

Summing Up

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

Future

- **0.6 to be released soon!**

  then

- Simplify the analysis results
- Add a proper web interface
- Improve performances
- Bare-metal support (almost done)
- Add Mac OS X support
- **Feedback?**

Other Stuff

- **Malwr**
  - https://malwr.com
- **VxCage**
  - https://github.com/cuckoobox/vxcage

www.cuckoosandbox.org
@cuckoosandbox

nex@cuckoosandbox.org
@botherder