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.
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

- Aug 2010 0.1a
- Nov 2011 0.2
- Jul 2012 0.4
- Jan 2011 0.1
- Dec 2011 0.3
- Dec 2012 0.5
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
  - Machine Manager
  - Result Server
  - Post-Processing
    - Processing
    - Signatures
- Utils
- Analyzer
  - Submission
  - Analysis Package
    - 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
from lib.common.abstacts 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
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)
PROCESSING MODULES

- In Core (under modules/processing/)
- Python modules
- Process raw results
- Populate collection of abstracted results
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
  - ...
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
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
                        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/cuckoobbox/community

• `utils/community.py`
USE CASE

• APT! APT! APT!
• Automatically collect and analyze PoisonIvy
• Extract configurations
• Report PoisonIvy C&C to a backend
from lib.common.abstacts 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 = {
    'namespace': '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 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 6F 66 74 75
        70 5C 49 6E 73 74 61 6C 6C 6F 6D 64 20 43 6F 6D 70
        6F 6E 65 6E 74 73 75 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
```
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!

• 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