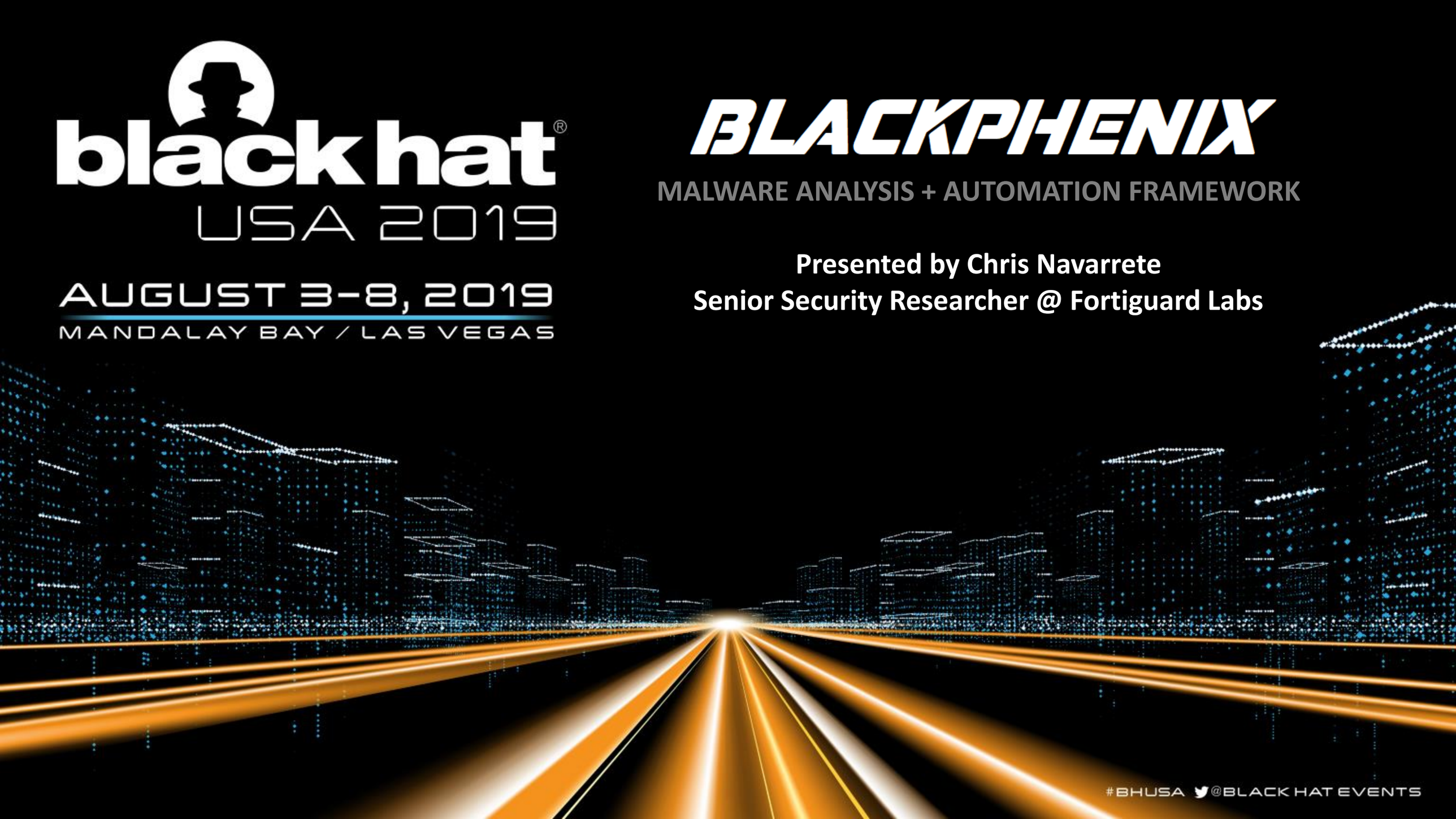

black hat
USA 2019
AUGUST 3–8, 2019
MANDALAY BAY / LAS VEGAS
BLACKPHENIX
MALWARE ANALYSIS + AUTOMATION FRAMEWORK
Presented by Chris Navarrete
Senior Security Researcher @ Fortiguard Labs
#BHUSA @BLACK HAT EVENTS

Disclaimer

- All content derives from my own opinions and does not represent my employer's views or opinions

BLACKPHENIX – Malware Analysis & Automation Framework

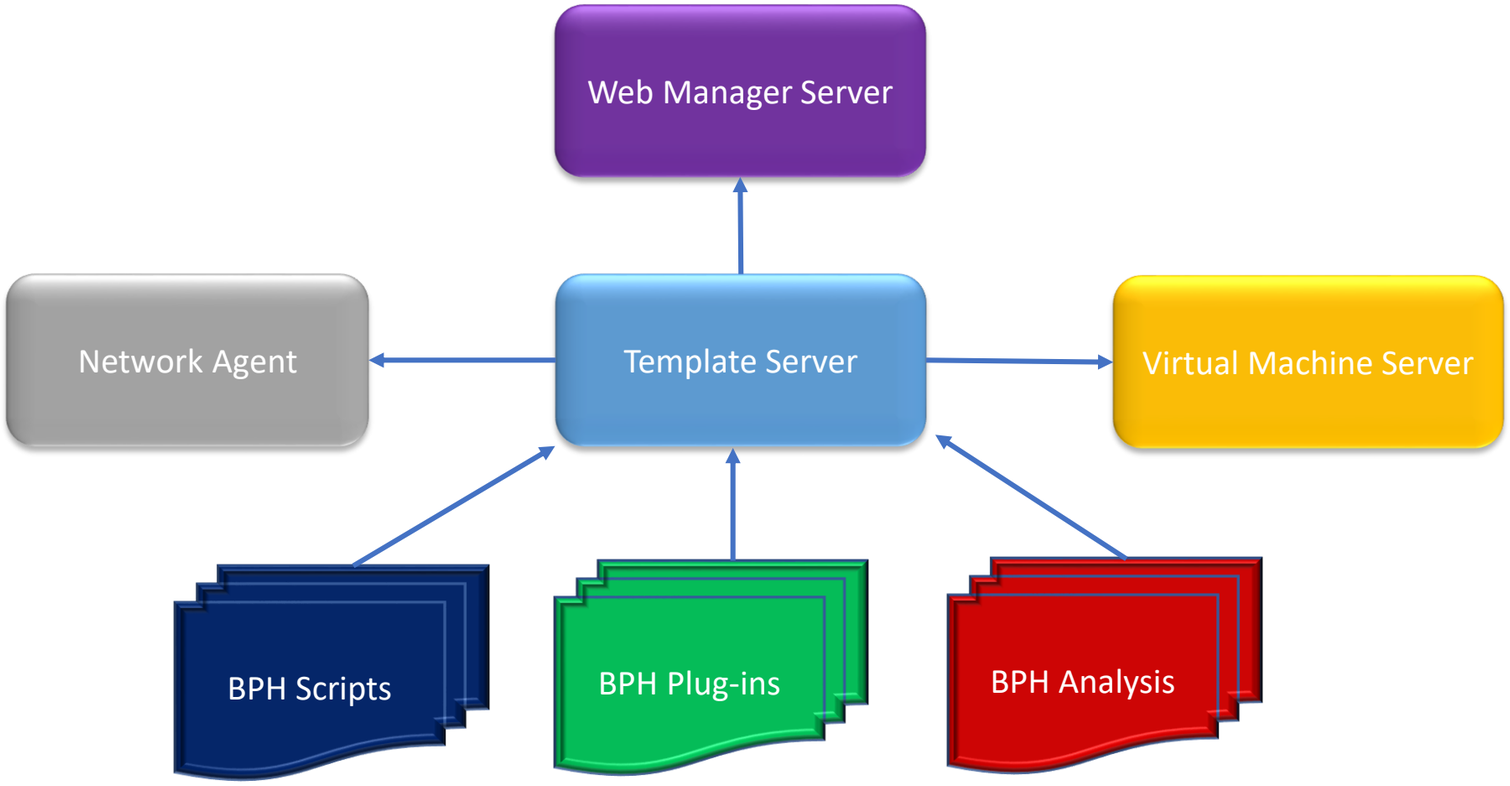
- **BLACKPHENIX** is an open source malware analysis automation framework composed of services, scripts, plug-ins, and tools and is based on a Command-and-Control (C&C) architecture
- Adapted many of the well-known malware analysis tools into Python modules
  - PEiD can be seen now as: ***PeId("MyMalwareSample.exe")***
- **What this framework is not?**
  - A Sandbox solution or replacement, but a complement – **Why a complement?**
- **What you can do with it?**
  - Automate any GUI or console tool and implement it easily into the framework
  - Develop BPH (Python) scripts to perform a single – or bundled execution of tools within the Virtual Machine, to later work on the generated data through BPH Analysis modules

BLACKPHENIX – CUSTOM ANALYSIS CONDITIONALS

If malware sample…

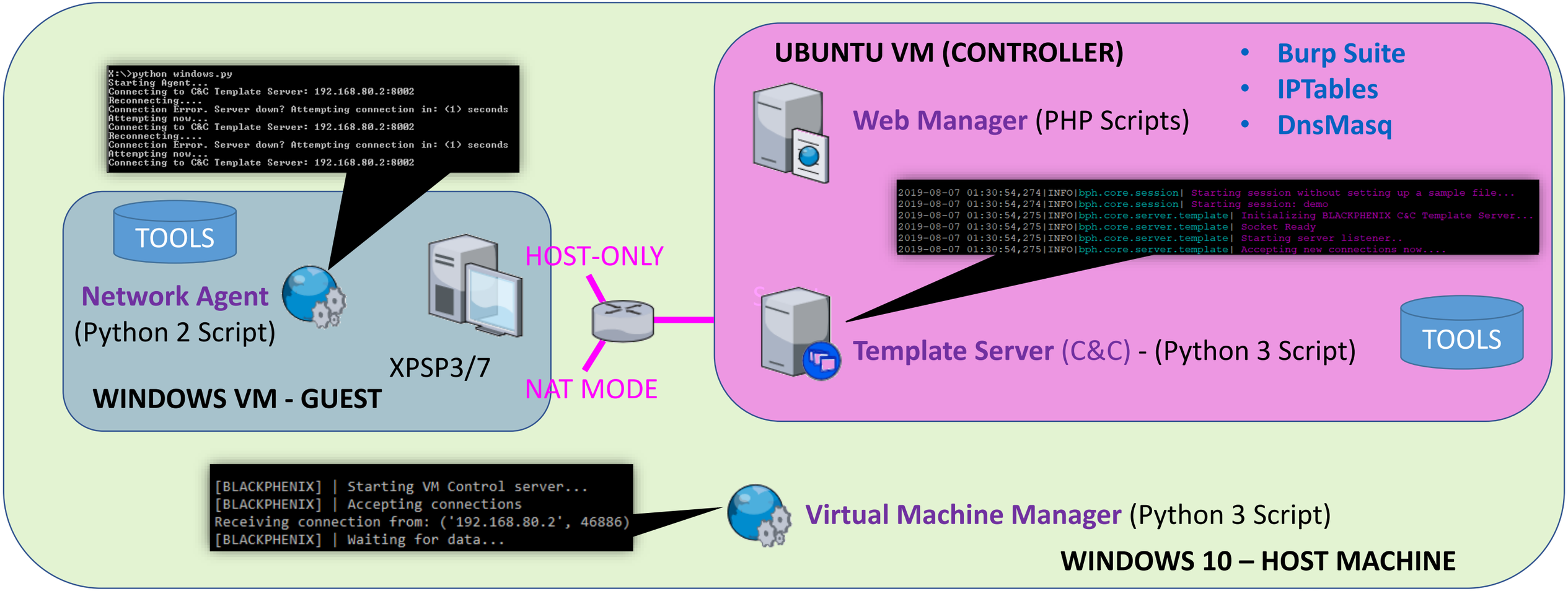
- **is Packed/Protected/Compressed with "EvilPacker_v1337"**, then run:
  A. Stand-Alone unpacker software (FUU-Faster-Universal-Unpacker, UPX, …)
  B. Stand-Alone Python unpacker Script
  C. Debugger/Disassembler Script (OllyDbg, IDA Python, WinDBG, …)
  D. App's Own Scripting Engine Script (010 Editor, CFF Explorer, CheatEngine, …)
- **Connects to remote services to fingerprint its external IP**, then restart and switch the guest VM to connect through a TOR (anonymous) communication channel
- **Downloads malicious components**, then pause its execution (before, during, or after) to generate a memory dump and perform further automated analysis

High-Level Overview



- **BPH Scripts** (Tool Commands)
- **BPH Plug-ins** (Tool Configuration)
- **BPH Analysis Modules** (Analysis Modules)

Main Architecture



Features

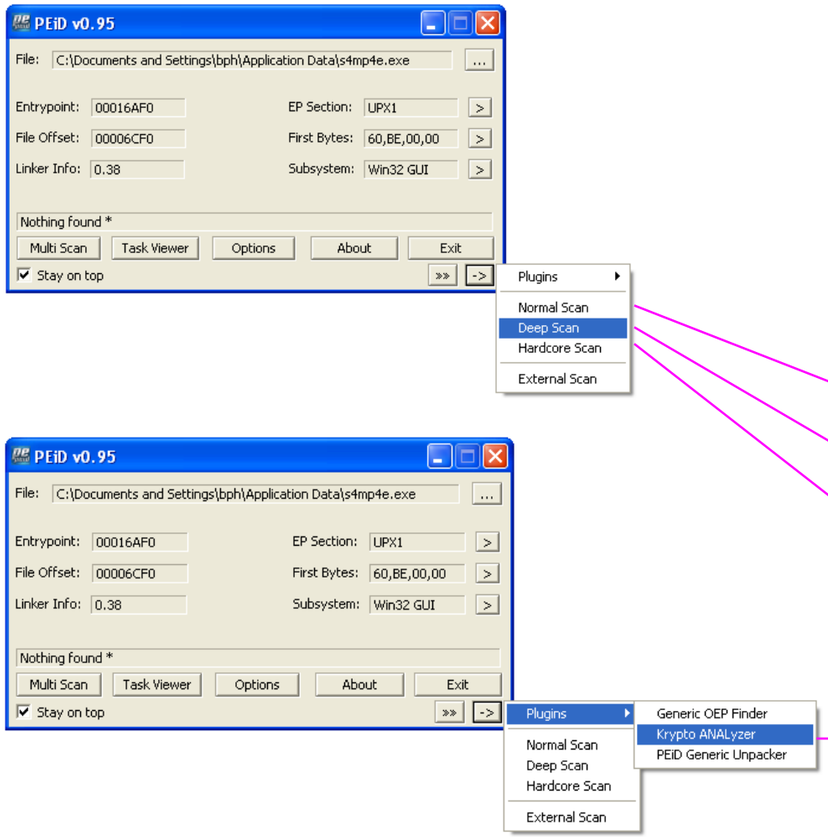
- Easy Installation & Deployment
- Python automation modules for malware analysis tools
- Virtual Machine Management
- Scripting support (BPH Scripts)
- Internet emulation
- Traffic redirection
- TOR support

```
# Tool Modules
from bph.tools.windows.peid import BphPeid as Peid
from bph.tools.windows.capturebat import BphCaptureBat as CaptureBat
from bph.tools.windows.regshot import BphRegShot as RegShot
from bph.tools.windows.procmon import BphProcMon as ProcMon
from bph.tools.windows.upx import BphUpx as Upx
from bph.tools.windows.pestudio import BphPeStudio as PeStudio
from bph.tools.windows.pd import BphPd as Pd
from bph.tools.windows.dumppe import BphDumpPE as DumpPE
from bph.tools.windows.depends import BphDepends as Depends
from bph.tools.windows.autoruns import BphAutoruns as Autoruns
from bph.tools.windows.exeinfope import BphExeInfoPe as ExeInfoPe
from bph.tools.windows.sysinspector import BphSysInspector as SysInspector
from bph.tools.windows.strings import BphStrings as Strings
from bph.tools.windows.floss import BphFloss as Floss
from bph.tools.windows.bintext import BphBinText as BinText
from bph.tools.windows.nircmd import BphNirCmd as NirCmd
from bph.tools.windows.ollydbg import BphOllyDbg as OllyDbg
from bph.tools.windows.resourcehacker import BphResourceHacker as ResourceHacker
from bph.tools.windows.xorsearch import BphXorSearch as XorSearch
from bph.tools.windows.xorstrings import BphXorStrings as XorStrings
from bph.tools.windows.networktrafficview import BphNetworkTrafficView as NetworkTrafficView

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.sample import BphSample as Sample
from bph.core.sample import BphLabFile as LabFile
from bph.core.session import BphSession as Session
from bph.core.vm import BphVmControl as VmControl
from bph.core.network.sniffer import BphSniffer as Sniffer

# Analysis Modules
from bph.analysis.basic.static import BphStringAnalysis as StringAnalysis
from bph.analysis.network import BphNetworkAnalysis as NetworkAnalysis
```

BPH Plug-In, Module & Script Structure Composition (PEiD GUI - Tool Automation)

BPH Plug-In, Module & Script Structure – PEiD GUI Elements

PEiD Tool



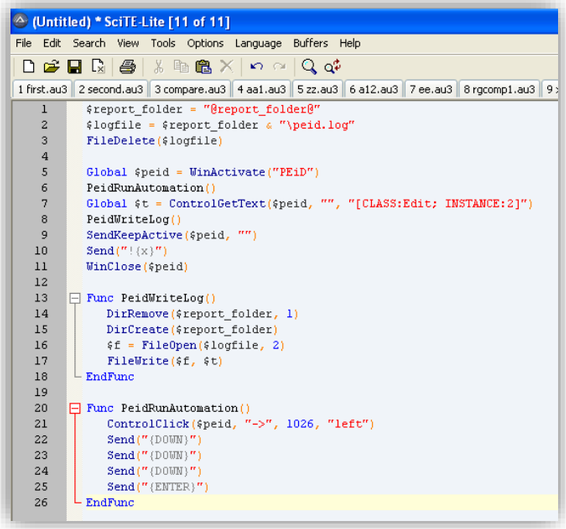
BPH Plug-in

peid.json

```
{
    "template": {
        "name": "PeID",
        "description": "PeiD plugin"
    },
    "tool": {
        "filename": "peid.exe",
        "version": "0.95",
        "type": "gui"
    },
    "configuration": {
        "execution": {
            "admin_required": false,
            "background_run": false
        },
        "reporting": {
            "report_files": true
        }
    },
    "actions": {
        "normal_scan": {
            "description": "Peid Normal Scan",
            "parameters": "",
            "automation": ""
        },
        "deep_scan": {
            "description": "Peid Deep Scan",
            "parameters": "@sample@",
            "automation": "JHJlcG9ydF9mb2xkZXIgPSAiQHJlcG
        },
        "hardcore_scan": {
            "description": "Peid Hardcore Scan",
            "parameters": "",
            "automation": ""
        },
        "kanal_scan": {
            "description": "Peid Kanal Crypto Scan",
            "parameters": "@sample@",
            "automation": "JHJlcG9ydF9mb2xkZXIgPSAiQHJlcG
        }
    }
}
```

BPH Plug-In

- **BPH Plug-In == Tool's JSON Configuration file**
  - Template & Tool – Meta-data information
  - Configuration – Execution & Reporting information
  - Actions – Tool's customized options
    - Description – Action's information
    - Parameters – Local or Remote variables (@sample@ = Sample's location inside of the VM)
    - Automation – Base64 Auto-It (automation) encoded data



BPH Module

peid.py

home ▸ bph ▸ blackphenix ▸ bph ▸ tools ▸ windows ▸ peid.py ▸ ...

```
1  from termcolor import colored
2
3  # Core imports
4  from bph.core.template import BphToolTemplateExecutor
5
6
7  class BphPeid(BphToolTemplateExecutor):
8      def __init__(self, target_file=None, tool_name='peid', arch='x86', version='0.95'):
9          super().__init__()
10         self.load_tool_config_file(tool_name, arch, version, target_file=target_file)
11
12     def normal_scan(self):
13         self.logger.log(colored('EXECUTING NORMAL SCAN', 'yellow'))
14         self.actions.action = "normal_scan"
15
16     def deep_scan(self):
17         self.logger.log(colored('EXECUTING DEEP SCAN', 'yellow'))
18         self.actions.action = "deep_scan"
19
20     def hardcore_scan(self):
21         self.logger.log(colored('EXECUTING HARDCORE SCAN', 'yellow'))
22         self.actions.action = "hardcore_scan"
23
24     def kanal_scan(self):
25         self.logger.log(colored('EXECUTING KANAL SCAN', 'yellow'))
26         self.actions.action = "kanal_scan"
27
```

- **BPH Module == Tool's Python Class**
  - A Class should be created: *BphMyTool()*
    - This class inherits from *BphToolTemplateExecutor* class, which is a class that acts as interface between BPH plug-ins, scripts a Virtual Machine Network Agent
  - Initialization function ("*__init__*" method) is in charge of:
    - Initializing the class with tool's name, architecture, and version number
    - Calling *super()* to inherit methods and attributes from its parent class
    - Loading BPH-Plug-In JSON configuration file by calling *self.load_tool_config_file()* method and tool's data as parameters
  - For each Tool's feature, a class method will be used
    - Each method must enable the action that will be called/executed
    - It modifies values inside the JSON (BPH Plug-In) configuration data
  - **In BLACKPHENIX, *all* tools are Python objects (Classes) with its correspondent attributes!**

BPH Script

peid.py ~/.../windows | peid.py ~/.../tools

```
1  # Tool imports
2  from bph.tools.windows.peid import BphPeid as Peid
3
4  # Core Imports
5  from bph.core.server.template import BphTemplateServer as TemplateServer
6  from bph.core.session import BphSession as Session
7  from bph.core.sample import BphLabFile as LabFile
8
9  session = Session(project_name='pony8')
10 session.start()
11
12 templateserver = TemplateServer()
13 templateserver.start()
14
15 sample_file = LabFile('/home/bph/blackphenix/session/pony8/launcher/s4mp4e.exe')
16
17 peid = Peid(sample_file)
18 peid.deep_scan()
19 peid.execute()
20 peid.output()
21
```

- **BPH Script == Execution Instructions**
  - The tool's Class should be imported
    - *From bph.tools.<os>.<tool_name> import BphMyTool as MyTool*
  - The tool's object can be created as a regular Python object
    - Some tools require initialization parameters, especially the ones that work with samples or files
    - Some other tools (such as Process Monitor or CaptureBAT are not tied to a sample, but collecting data from a virtualized sample execution
  - After object's creation, then its methods can be called
    - Tools that run inside the Virtual Machine requires a call to the execute() method
    - BPH uses two main methods: output() and files()
      - *Output()* returns the tool's output
      - *Files()* returns the folder location where the collected files resides on
  - On a single BPH Scripts a user can choose a single tool or a bundle to perform different actions and use the data generated by all of them!

BPH Plug-In, Module & Script Structure – All together

BPH Plug-in

peid.json

```
home ▸ bph ▸ blackphenix ▸ plugins ▸ basic ▸ static ▸ peid ▸ x86 ▸ 0.95 ▸ {} peid.json ▸
```

```
{
    "template": {
        "name": "PeID",
        "description": "PeiD plugin"
    },
    "tool": {
        "filename": "peid.exe",
        "version": "0.95",
        "type": "gui"
    },
    "configuration": {
        "execution": {
            "admin_required": false,
            "background_run": false
        },
        "reporting": {
            "report_files": true
        }
    },
    "actions": {
        "normal_scan": {
            "description": "Peid Normal Scan",
            "parameters": "",
            "automation": ""
        },
        "deep_scan": {
            "description": "Peid Deep Scan",
            "parameters": "@sample@",
            "automation": "JHJlcG9ydF9mb2xkZXIgPSAiQHJlcG
        },
        "hardcore_scan": {
            "description": "Peid Hardcore Scan",
            "parameters": "",
            "automation": ""
        },
        "kanal_scan": {
            "description": "Peid Kanal Crypto Scan",
            "parameters": "@sample@",
            "automation": "JHJlcG9ydF9mb2xkZXIgPSAiQHJlcG
        }
    }
}
```

BPH Module

peid.py

```
home ▸ bph ▸ blackphenix ▸ bph ▸ tools ▸ windows ▸ peid.py ▸ ...
```

```
from termcolor import colored

# Core imports
from bph.core.template import BphToolTemplateExecutor


class BphPeid(BphToolTemplateExecutor):
    def __init__(self, target_file=None, tool_name='peid', arch='x86', version='0.95'):
        super().__init__()
        self.load_tool_config_file(tool_name, arch, version, target_file=target_file)

    def normal_scan(self):
        self.logger.log(colored('EXECUTING NORMAL SCAN', 'yellow'))
        self.actions.action = "normal_scan"

    def deep_scan(self):
        self.logger.log(colored('EXECUTING DEEP SCAN', 'yellow'))
        self.actions.action = "deep_scan"

    def hardcore_scan(self):
        self.logger.log(colored('EXECUTING HARDCORE SCAN', 'yellow'))
        self.actions.action = "hardcore_scan"

    def kanal_scan(self):
        self.logger.log(colored('EXECUTING KANAL SCAN', 'yellow'))
        self.actions.action = "kanal_scan"

```

BPH Script

peid.py ~/.../windows | peid.py ~/.../tools

```
home ▸ bph ▸ blackphenix ▸ scripts ▸ custom ▸ tools ▸ peid.py ▸ ...
```

```
# Tool imports
from bph.tools.windows.peid import BphPeid as Peid

# Core Imports
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()

templateserver = TemplateServer()
templateserver.start()

sample_file = LabFile('/home/bph/blackphenix/session/pony8/launcher/s4mp4e.exe')

peid = Peid(sample_file)
peid.deep_scan()
peid.execute()
peid.output()

```

Only <u>three files</u> (one JSON configuration file and two Python scripts) are required to have a fully functional and integrated tool

PEID DEMO

BPH Analysis Modules – NetworkTrafficView Tool

- BPH Analysis scripts are used to consume the data collected by the tools, or in other words, the results given by the BPH scripts

1. Import network analysis – ***NetworkAnalysisCsvReader*** class

```
# Analysis Imports
from bph.analysis.network import BphNetworkAnalysisCsvReader as NetworkAnalysisCsvReader
```

2. Extract (***ntv.fetch()***) all domains from the collected CSV file

```
ntv = NetworkTrafficView()
ntv.start()
ntv.execute()

sample_exec = NirCmd(LabFile(session.launcher_abs_path))
sample_exec.configuration.execution.background_run = True
sample_exec.start_process(program='@sample@')
sample_exec.execute(delay=15)

ntv.stop()
ntv.execute()

for csv_file in ntv.files():
    ntv = NetworkAnalysisCsvReader(tool_name='networktrafficview', csv_file=csv_file)
    ntv.fetch(data_type='domains')
```

```
class BphNetworkAnalysisCsvReader(BphNetworkAnalysis):
    def __init__(self, tool_name=None, csv_file=None):
        super().__init__()
        self.tool_name = tool_name
        self.csv_file = csv_file

    def fetch(self, data_type=None):
        with open(self.csv_file) as csv_file_h:
            csv_reader = csv.reader(csv_file_h, delimiter=',')

            domains = []

            for row in csv_reader:

                if self.tool_name == "networktrafficview":
                    if "Destination" not in str(row):
                        if "[" in str(row[3]):
                            domain = row[3].split(' ')[0]
                            if domain not in domains:
                                domains.append(domain)


            print(domains)
            return domains
```

NetworkTrafficView + Analysis Module + Python One-liner

```python
# Tool imports
from bph.tools.windows.networktrafficview import BphNetworkTrafficView as NetworkTrafficView
from bph.tools.windows.nircmd import BphNirCmd as NirCmd

# Core Imports
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

# Analysis Imports
from bph.analysis.network import BphNetworkAnalysisCsvReader as NetworkAnalysisCsvReader

import time

session = Session(project_name='demo')
session.start()

templateserver = TemplateServer()
templateserver.start()

ntv = NetworkTrafficView()
ntv.start()
ntv.execute()

nircmd = NirCmd()
nircmd.configuration.reporting.report_files = True
nircmd.start_process(program=r'python -c "import urllib2 ; print(urllib2.urlopen(\"http://icanhazip.com\").read().strip())" > @report_folder@\\nircmd.log')
nircmd.execute(delay=5)
nircmd.output()

ntv.stop()
ntv.execute()

for csv_file in ntv.files():
    ntv = NetworkAnalysisCsvReader(tool_name='networktrafficview', csv_file=csv_file)
    ntv.fetch(data_type='domains')
```

UPX Decompression

home ▸ bph ▸ blackphenix ▸ scripts ▸ custom ▸ tools ▸ upx.py ▸ ...

```
1   from bph.tools.windows.upx import BphUpx as Upx
2   from bph.core.server.template import BphTemplateServer as TemplateServer
3   from bph.core.session import BphSession as Session
4   from bph.core.sample import BphLabFile as LabFile
5
6   session = Session(project_name='pony8')
7   session.start()
8   session.set_launcher(move_sample=False)
9
10  templateserver = TemplateServer()
11  templateserver.start()
12
13  upx = Upx(LabFile(session.launcher_abs_path))
14  upx.decompress()
15  upx.execute()
16  upx.output()
17  upx.files()
```

TOR + Python Support

```
from bph.core.vm import BphVmControl as VmControl
from bph.core.network.sniffer import BphSniffer as Sniffer

import time

session = Session(project_name='pony8')
session.start()

vm = VmControl()
vm.set_vm('basic_static', network_id=1)
vm.start()

templateserver = TemplateServer()
templateserver.start()

# Some additional seconds to allow the TOR channel be established.
time.sleep(10)
nircmd = NirCmd()
nircmd.configuration.reporting.report_files = True
nircmd.start_process(program=r'python -c "import urllib2 ; print(urllib2.urlopen(\"http://icanhazip.com\").read().strip())" > @report_folder@\\nircmd.log')
nircmd.execute(delay=3)
nircmd.output()

templateserver.stop()
vm.stop()

vm = VmControl()
vm.set_vm('basic_static', network_id=2)
vm.start()

templateserver = TemplateServer()
templateserver.start()

time.sleep(20)

nircmd = NirCmd()
nircmd.configuration.reporting.report_files = True
nircmd.start_process(program=r'python -c "import urllib2 ; print(urllib2.urlopen(\"http://icanhazip.com\").read().strip())" > @report_folder@\\nircmd.log')
nircmd.execute(delay=3)
nircmd.output()

templateserver.stop()
vm.stop()

print("Done")
```

ExeInfoPe
+
UPX Decompression
+
Imports Data

```python
# Tool imports
from bph.tools.windows.exeinfope import BphExeInfoPe as ExeInfoPe
from bph.tools.windows.upx import BphUpx as Upx

# Core Imports
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

import re

session = Session(project_name='pony8')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

sample_file = LabFile(session.launcher_abs_path)

exeinfope = ExeInfoPe(sample_file)
exeinfope.default()
exeinfope.execute()

sig_match = re.search(r'(upx.*3.91.*)', "".join(exeinfope.output()), re.I)

if sig_match:
    print("SAMPLE IS UPX PACKED => VERSION 3.91. DECOMPRESSING...")

    for symbol, function_data in sample_file.symbols(type='imports').items():
        print(symbol)

        for data in function_data:
            print(data)

    upx = Upx(sample_file)
    upx.decompress()
    upx.execute()

    unpack_result = upx.output()

    if "Unpacked 1 file." in unpack_result:
        print("SAMPLE WAS UNPACKED SUCCESSFULLY...!!!11")

        files_found = upx.files()

        for file_found in files_found:
            if "unpacked.exe" in file_found:

                unpacked_file = LabFile(file_found)
                exeinfope = ExeInfoPe(unpacked_file)
                exeinfope.default()
                exeinfope.execute()
                exeinfope.output()

                for symbol, function_data in unpacked_file.symbols(type='imports').items():
                    print(symbol)

                    for data in function_data:
                        print(data)
```

DUMPPE + WINAPI FUNCTION PARSING

```python
# Tool Modules
from bph.tools.windows.dumppe import BphDumpPE as DumpPE

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

dumppe = DumpPE(LabFile(session.launcher_abs_path))
dumppe.default()
dumppe.execute()
dumppe.output()

dumppe.win_apis()
```

```python
import re

# Custom imports
from bph.core.constants import *
from bph.core.template import BphToolTemplateExecutor
from bph.analysis.basic.static import BphWinApiStrings as WinApiStringAnalysis

class BphDumpPE(BphToolTemplateExecutor):

    def __init__(self, target_file=None, tool_name='dumppe', arch='x86', version='2.32'):
        super().__init__()
        self.load_tool_config_file(tool_name, arch, version, target_file=target_file)

    def default(self):
        print("Default")
        self.configuration.reporting.report_files = True
        self.actions.action = "default"

    def win_apis(self):
        #self._search(search_type='domains')
        for file in self.files():
            if "dumppe.log" in file and self.rid in file:

                a = WinApiStringAnalysis()
                a.set_log(file)
                a.search()
```

```python
class BphWinApiStrings(BphStringAnalysis):

    def search(self):
        self.logger.log("Searching for API function names...")

        apis = ['LoadLibrary', 'GetProcAddress']

        for d_string in BphStringAnalysis.strings_file.readlines():
            d_string = d_string.strip()

            if len(d_string) > 5:
                search = re.search(r'([A-Z]([A-Z]*[a-z][a-z]*[A-Z]|[a-z]*[A-Z][A-Z]*[a-z])[A-Za-z]*)', d_string)

                if search and len(search.group()) > 5:
                    string_found = search.group()

                    if string_found in apis:
                        self.logger.log(colored("Found: {}".format(string_found), 'green'))
```

BinText

```python
# Tool Imports
from bph.tools.windows.bintext import BphBinText as BinText

# Core Imports
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

bintext = BinText(LabFile(session.launcher_abs_path))
bintext.default()
bintext.execute(delay=3)
bintext.output()
bintext.files()
```

XOR Search
+
XOR Strings
+
PEiD KANAL Scan
+
FLOSS

```python
# Tool Modules
from bph.tools.windows.peid import BphPeid as Peid
from bph.tools.windows.floss import BphFloss as Floss
from bph.tools.windows.xorsearch import BphXorSearch as XorSearch
from bph.tools.windows.xorstrings import BphXorStrings as XorStrings

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

xorsearch = XorSearch(LabFile(session.launcher_abs_path))
xorsearch.search_www()
xorsearch.execute()
xorsearch.output()

xorstrings = XorStrings(LabFile(session.launcher_abs_path))
xorstrings.search_xored()
xorstrings.execute()
xorstrings.output()

#"""
peid = Peid(LabFile(session.launcher_abs_path))
peid.kanal_scan()
peid.execute()
peid.output()

floss = Floss(LabFile(session.launcher_abs_path))
floss.search()
floss.execute()
floss.output()
```

CaptureBAT

```
# Tool Modules
from bph.tools.windows.capturebat import BphCaptureBat as CaptureBat

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.sample import BphSample as Sample
from bph.core.sample import BphLabFile as LabFile
from bph.core.session import BphSession as Session
import time

session = Session(project_name='pony8')
session.start()

templateserver = TemplateServer()
templateserver.start()

capturebat = CaptureBat()
capturebat.cleanup()
capturebat.execute()

capturebat.start()
capturebat.execute(delay=15)


capturebat.stop()
capturebat.execute()

capturebat.collect()
capturebat.execute()
capturebat.files()
```

RegShot

```python
# Tool Modules
from bph.tools.windows.regshot import BphRegShot as RegShot

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()

templateserver = TemplateServer()
templateserver.start()

regshot = RegShot()
regshot.first_shot()
regshot.execute()

regshot.second_shot()
regshot.execute()

regshot.compare()
regshot.execute()
regshot.files()
```

Process Monitor

```
# Tool Modules
from bph.tools.windows.procmon import BphProcMon as ProcMon

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()

templateserver = TemplateServer()
templateserver.start()

procmon = ProcMon()
procmon.capture_xp()
procmon.execute(delay=10)

procmon.terminate()
procmon.execute(delay=15)

procmon.export()
procmon.execute(delay=10)

procmon.files()
```

NirCmd – OS Command (PING)

```
# Tool Modules
from bph.tools.windows.nircmd import BphNirCmd as NirCmd

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session

session = Session(project_name='pony8')
session.start()

templateserver = TemplateServer()
templateserver.start()

nircmd = NirCmd()
nircmd.configuration.reporting.report_files = True
nircmd.start_process(program='ping 4.2.2.2 > @report_folder@\\nircmd.log')
nircmd.execute(delay=5)
nircmd.output()
nircmd.files()
```

OllyDbg – OllyScript

```
# Tool Modules
from bph.tools.windows.ollydbg import BphOllyDbg as OllyDbg

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

ollydbg = OllyDbg(LabFile(session.launcher_abs_path))
ollydbg.configuration.execution.background_run = True
ollydbg.ollyscript(ollyscript_file_name='logapicall.osc')
ollydbg.execute()
```

Resource Hacker

```
# Tool Modules
from bph.tools.windows.resourcehacker import BphResourceHacker as ResourceHacker

# Core Modules
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile

session = Session(project_name='pony8')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

reshacker = ResourceHacker(LabFile(session.launcher_abs_path))
reshacker.extract_resources()
reshacker.execute()
reshacker.files()
```

Autoruns

```
1   # Tool Modules
2   from bph.tools.windows.autoruns import BphAutoruns as Autoruns
3
4   # Core Modules
5   from bph.core.server.template import BphTemplateServer as TemplateServer
6   from bph.core.session import BphSession as Session
7   from bph.core.sample import BphLabFile as LabFile
8
9   session = Session(project_name='pony8')
10  session.start()
11
12  templateserver = TemplateServer()
13  templateserver.start()
14
15  autoruns = Autoruns()
16  autoruns.analysis_basic()
17  autoruns.execute(delay=5)
18  autoruns.files()
```

Malware Execution + CaptureBAT

```python
# Tool Imports
from bph.tools.windows.nircmd import BphNirCmd as NirCmd
from bph.tools.windows.pd import BphPd as Pd
from bph.tools.windows.capturebat import BphCaptureBat as CaptureBat

# Core Imports
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile


session = Session(project_name='aop')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

capturebat = CaptureBat()
capturebat.cleanup()
capturebat.execute()

capturebat.start()
capturebat.execute()

sample_exec = NirCmd(LabFile(session.launcher_abs_path))
sample_exec.configuration.execution.background_run = False
sample_exec.start_process(program='@sample@')
sample_exec.execute()

capturebat.stop()
capturebat.execute()

capturebat.collect()
capturebat.execute()
capturebat.files()
```

Malware Execution + pd32

```python
# Tool Imports
from bph.tools.windows.nircmd import BphNirCmd as NirCmd
from bph.tools.windows.pd import BphPd as Pd

# Core Imports
from bph.core.server.template import BphTemplateServer as TemplateServer
from bph.core.session import BphSession as Session
from bph.core.sample import BphLabFile as LabFile


session = Session(project_name='aop')
session.start()
session.set_launcher(move_sample=False)

templateserver = TemplateServer()
templateserver.start()

sample_exec = NirCmd(LabFile(session.launcher_abs_path))
sample_exec.configuration.execution.background_run = True
sample_exec.start_process(program='@sample@')
sample_exec.execute()

pd = Pd()
pd.dump_process(process_name='@sample_filename@')
pd.execute(delay=5)

files_found = pd.files()

for file_found in files_found:
    if file_found.endswith('.exe'):

        dumped_file = LabFile(file_found)
        dumped_file_imports = dumped_file.symbols(type='imports')

        if dumped_file_imports is not False:
            for symbol, function_data in dumped_file_imports.items():
                print(symbol)

                for data in function_data:
                    print(data)

        print(dumped_file.abs_path)

kill_process = NirCmd()
kill_process.kill_process(program='@sample_filename@')
kill_process.execute()
```

Supported Tool Automation

BASIC STATIC ANALYSIS	BASIC DYNAMIC ANALYSIS (BEHAVIORAL ANALYSIS)
• PeID (+KANAL)	• RegShot
• ExeInfoPe	• Process Monitor
• BinText	• Autoruns
• Floss	• NirCmd
• Dependency Walker	• CaptureBAT
• DumpPE	
• UPX	
• XOR Search/Strings	
• Resource Hacker	

ADVANCED DYNAMIC	ADVANCED STATIC
• OllyDbg (OllyScript)	• *IDA Pro (IDA Python)
• *Cheat Engine (LUA)	

MEMORY ANALYSIS	NETWORK ANALYSIS
• Pd32/64	• NetworkTrafficView
	• Tcpdump

How to Contribute

- Developing BPH Scripts or Analysis Modules and push them into the official GitHub repository
- Introducing new tools to the Framework – Check the tool's licensing terms before starting!
- Fixing bugs, and/or adding new features
- If you have doubts, don't hesitate to ask! (See contact details)
- Have a cool tool in mind? I will guide you to integrate it!

Contact

Name: Chris Navarrete

Personal Email: acid.ctrl.ak@gmail.com

BPH Email: bph_framework@fortinet.com

Twitter: @ctrl_ak

IRC@FREENODE: #bph-framework