EVERY STEP YOU TAKE

Profiling the System

Jamie Levy (gleeda)

Purpose

- DFIR investigations spanning multiple machines
- Provides a mechanism for cutting up the data into smaller digestible chunks

- Make use of mechanisms from the disk forensics realm:
  - Baselining/Whitelisting/Blacklisting
  - Indicators of Compromise (IOCs)
    - CybOX
  - Profiling

Baselines

- In order to find changes or irregularities you first need to know “what’s normal”
- Create a baseline from a clean production system
  - Software installed
    - Versions
  - Files
    - Executables
    - DLLs
    - Modules
  - Registry keys
  - Services

Baselines (continued)

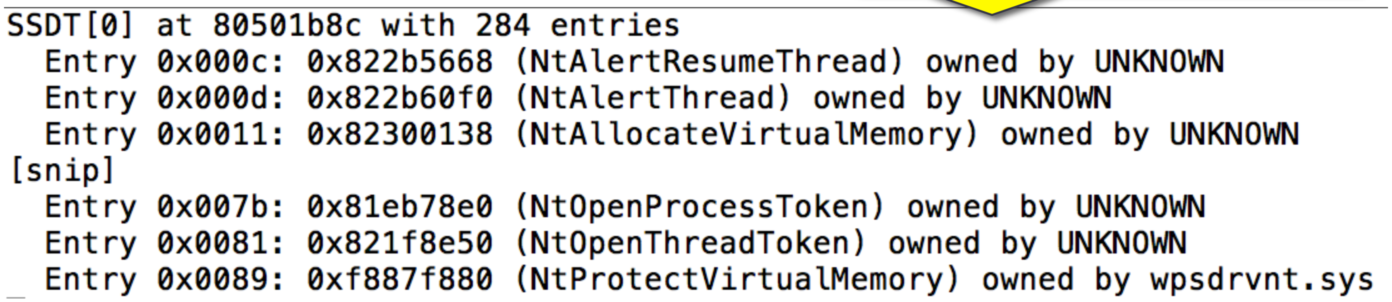
- Memory only
  - Able to see what's normal during a running state
    - Processes and heritage, services, loaded DLLs, modules etc
  - Able to capture "normal" hooks (AV SSDT hooks)
  - Caveat: Not all software is running, there may be different files in use at different states of running software
    - Volatility plugin: profiler
- Disk
  - Able to fill-in the gaps for some missing information during runtime (DLLs, exes etc)
    - Baseliner EnScript
  - Able to "diff" registry keys from disk
    - Quicker
    - No swapped keys
      - regdiff.py

Caveat: Hook comparisons

- Some of the items that we want to examine include hooked code (SSDT hooks, ApiHooks).
- The assembly will include memory addresses, so we need regex comparisons (Yara)
- Painful to do by hand if there are many hooks/jumps (though not impossible)
  - So we'll automate this as well…

These are actually Symantec Hooks

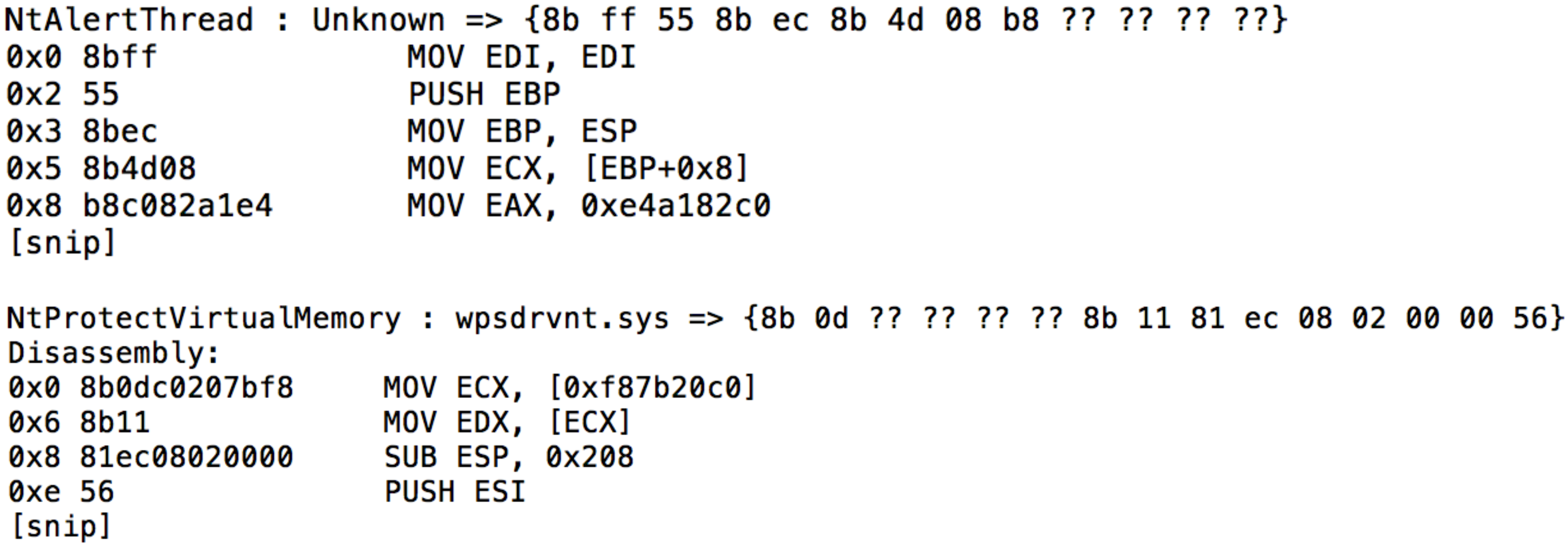
- Evil?



```
SSDT[0] at 80501b8c with 284 entries
  Entry 0x000c: 0x822b5668 (NtAlertResumeThread) owned by UNKNOWN
  Entry 0x000d: 0x822b60f0 (NtAlertThread) owned by UNKNOWN
  Entry 0x0011: 0x82300138 (NtAllocateVirtualMemory) owned by UNKNOWN
[snip]
  Entry 0x007b: 0x81eb78e0 (NtOpenProcessToken) owned by UNKNOWN
  Entry 0x0081: 0x821f8e50 (NtOpenThreadToken) owned by UNKNOWN
  Entry 0x0089: 0xf887f880 (NtProtectVirtualMemory) owned by wpsdrvnt.sys
```

Hook comparisons (continued)

- If you have legitimate software (like AV) that makes hooks, the hooks are often the same (for the same OS)and therefore can be whitelisted



```
NtAlertThread : Unknown => {8b ff 55 8b ec 8b 4d 08 b8 ?? ?? ?? ??}
0x0 8bff                 MOV EDI, EDI
0x2 55                   PUSH EBP
0x3 8bec                 MOV EBP, ESP
0x5 8b4d08               MOV ECX, [EBP+0x8]
0x8 b8c082a1e4           MOV EAX, 0xe4a182c0
[snip]

NtProtectVirtualMemory : wpsdrvnt.sys => {8b 0d ?? ?? ?? ?? 8b 11 81 ec 08 02 00 00 56}
Disassembly:
0x0 8b0dc0207bf8       MOV ECX, [0xf87b20c0]
0x6 8b11               MOV EDX, [ECX]
0x8 81ec08020000       SUB ESP, 0x208
0xe 56                 PUSH ESI
[snip]
```

Whitelisting/Blacklisting

- Once we have our baseline it’s easy to see if a machine has items running that are not included in it
    - Could be maintained as a list and output items not in the list

- We can also use “known bad” items as a blacklist and see if these items are found on the machine
    - Could be maintained as a list and items found from this list are output

(We can probably do better though…)

Indicators of Compromise (IOCs)

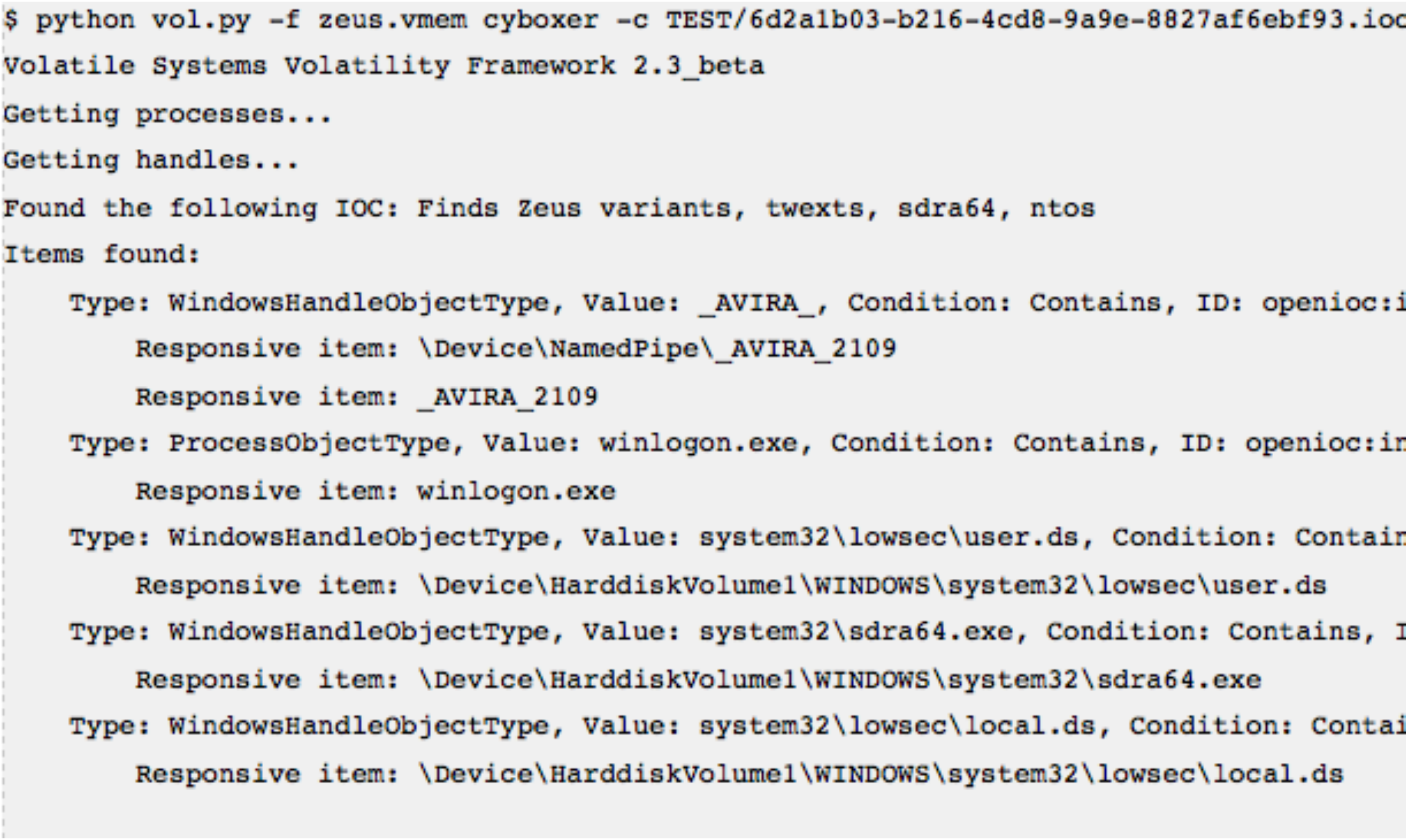
- Artifacts of interest to indicate malware is present on a machine
- Contains logic
  - Useful for combining several artifacts
  - Adds flexibility and helps remove false positives
- Able to share indicator packages

- CybOX:
  - https://github.com/CybOXProject/python-cybox
  - Python bindings
  - Able to convert OpenIOCs to CybOX format
  - Easier to control programmatically

Cyboxer Plugin

- Uses the python-cybox library
- Needs a single CybOX xml file or a directory of xml files
- If supported memory objects are found (including their appropriate logic), the title from each xml file and items found are printed
- Supported items:
  - Process names
  - IP addresses/domains
  - Mutexes
  - Open(ed) files
  - Services
  - Registry Keys/Values

Cyboxer Plugin Example



```
$ python vol.py -f zeus.vmem cyboxer -c TEST/6d2a1b03-b216-4cd8-9a9e-8827af6ebf93.ioc
Volatile Systems Volatility Framework 2.3_beta
Getting processes...
Getting handles...
Found the following IOC: Finds Zeus variants, twexts, sdra64, ntos
Items found:
    Type: WindowsHandleObjectType, Value: _AVIRA_, Condition: Contains, ID: openioc:i
        Responsive item: \Device\NamedPipe\_AVIRA_2109
        Responsive item: _AVIRA_2109
    Type: ProcessObjectType, Value: winlogon.exe, Condition: Contains, ID: openioc:in
        Responsive item: winlogon.exe
    Type: WindowsHandleObjectType, Value: system32\lowsec\user.ds, Condition: Contain
        Responsive item: \Device\HarddiskVolume1\WINDOWS\system32\lowsec\user.ds
    Type: WindowsHandleObjectType, Value: system32\sdra64.exe, Condition: Contains, I
        Responsive item: \Device\HarddiskVolume1\WINDOWS\system32\sdra64.exe
    Type: WindowsHandleObjectType, Value: system32\lowsec\local.ds, Condition: Contai
        Responsive item: \Device\HarddiskVolume1\WINDOWS\system32\lowsec\local.ds
```

Profiling

- Allows us to answer various things about the system.
- Can follow a system over time to see if things changed
- Can answer specific questions about the system:
  - A list of items that are “normal”
  - A list of items that are “abnormal”
  - Is software X installed?
    - What version?
  - What artifacts are left after installing software X?
  - What artifacts are left after running malware X?
    - Create a CybOX package for sharing IOCs

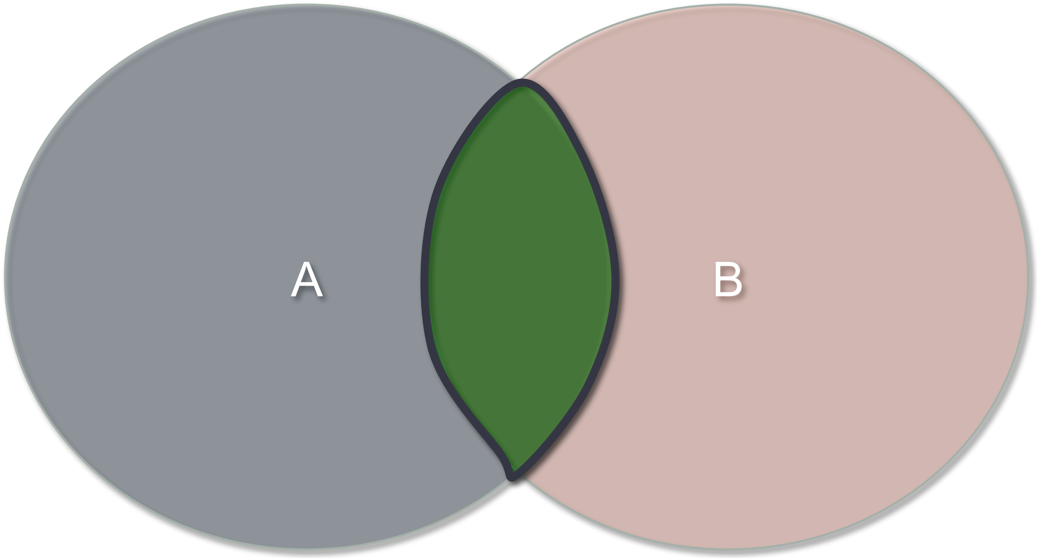
Profiling (continued)

- So it’s easy to answer some of these questions for one machine at a time, but suppose you want to examine several states of the same machine over time or several machines at once.
- What is needed is a way to categorize each machine into a separate collection of interesting items

- Each machine will have a “profile”
- The profile code can be used offline or imported for use with Volatility

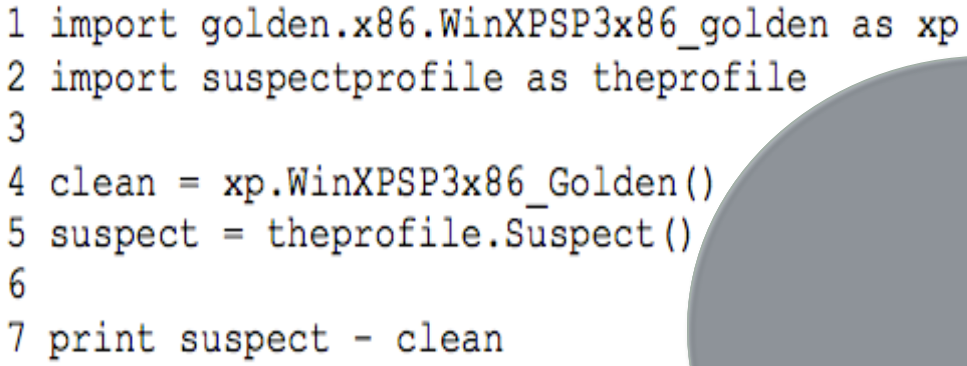
Profiles

- A and B represent two different machines (can be more)
- Each machine contains its own baseline of items (profiles)
  - Processes (and heritage)
  - Services
  - SSDTs (yara sigs)
  - Connections
  - ApiHooks (yara sigs)
  - DLLs
  - EXEs
  - Mutexes
  - Modules (modules)
  - Drivers (driver objects)
  - Registry keys
  - …

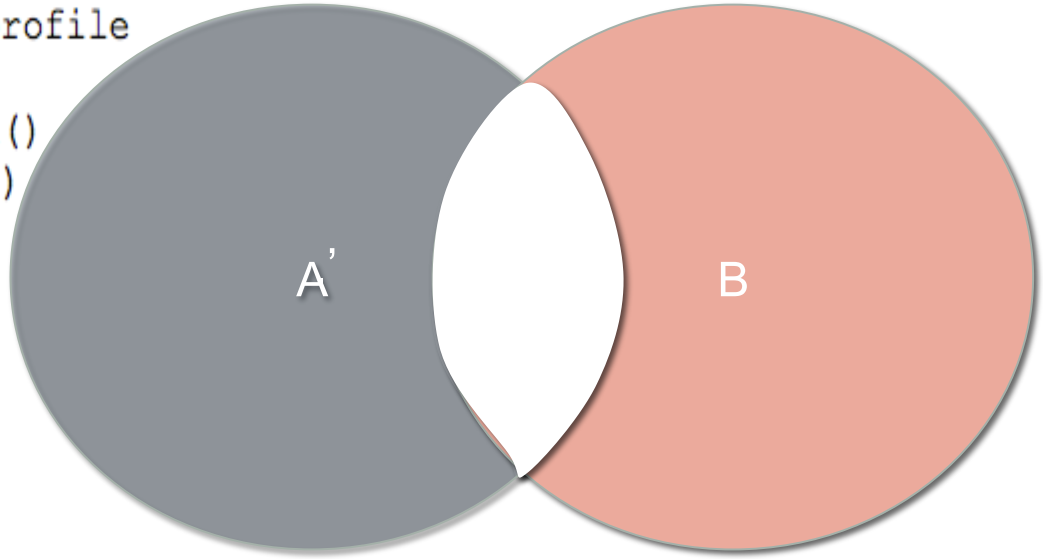


Set Difference

- New set with elements in A but not B
- Useful for removing items from the baseline (whitelist)
- A – B = A′

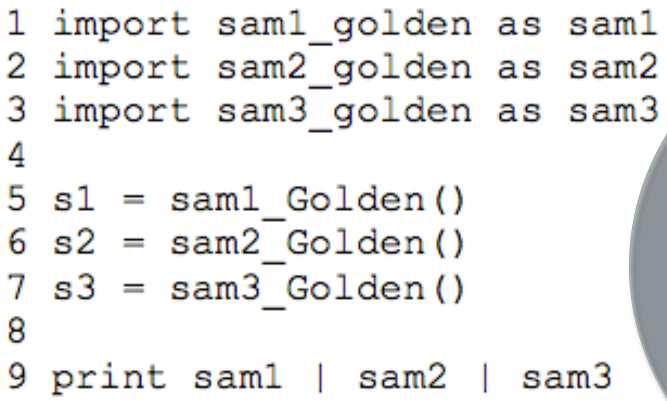


```
import golden.x86.WinXPSP3x86_golden as xp
import suspectprofile as theprofile

clean = xp.WinXPSP3x86_Golden()
suspect = theprofile.Suspect()

print suspect - clean
```

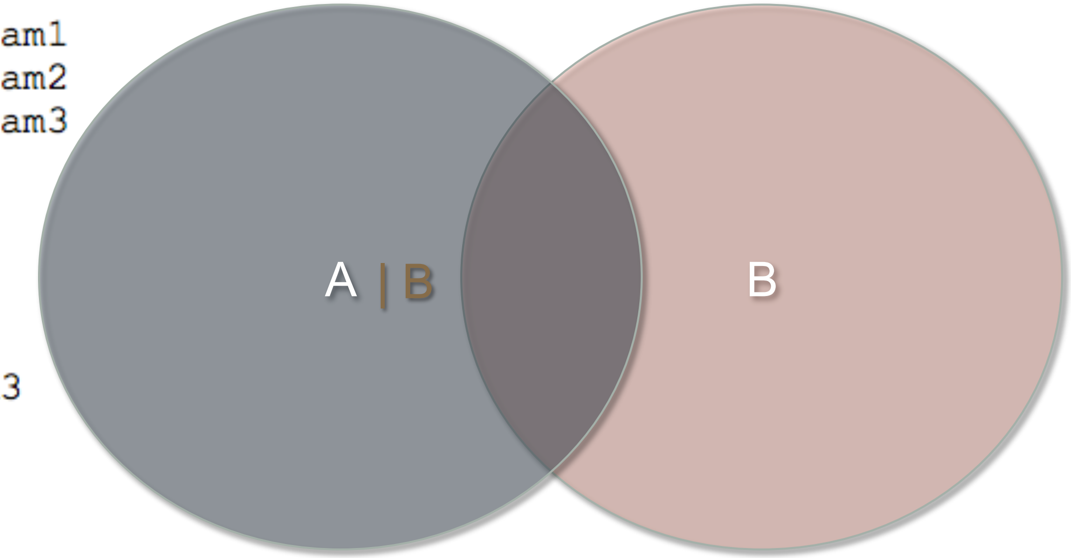


Union

- New set with elements from both A and B
- Good for combining baselines into one
- A | B

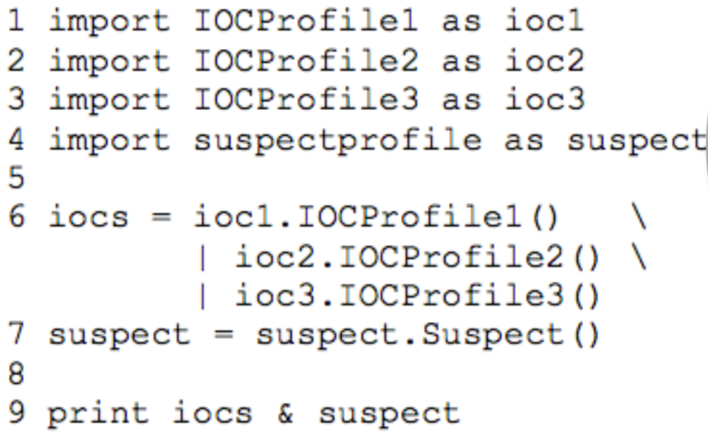


```
import sam1_golden as sam1
import sam2_golden as sam2
import sam3_golden as sam3

s1 = sam1_Golden()
s2 = sam2_Golden()
s3 = sam3_Golden()

print sam1 | sam2 | sam3
```

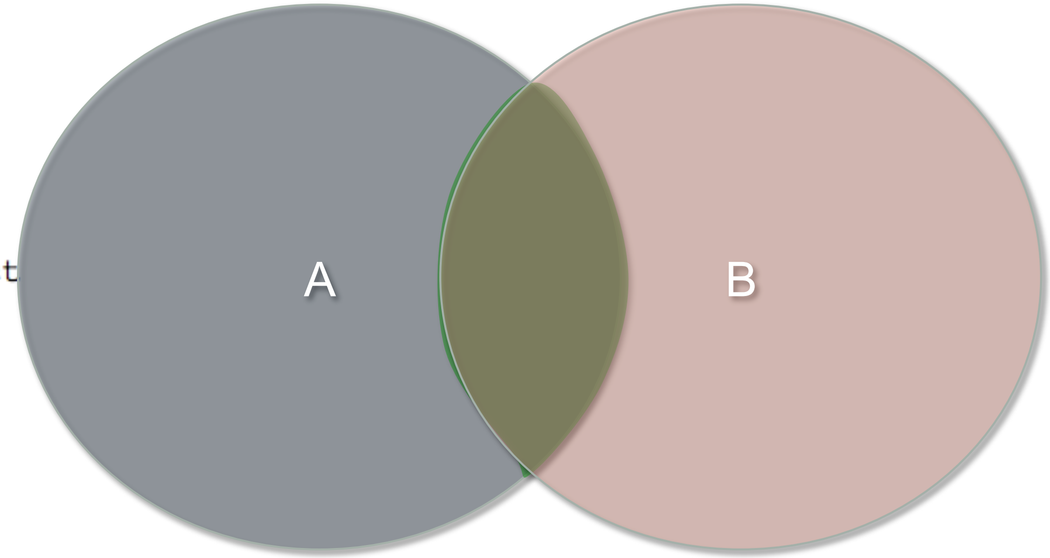


Intersection

- New set with elements common to both A and B
- Useful for finding items that are common between multiple machines
  - Example: IOCs
- A & B



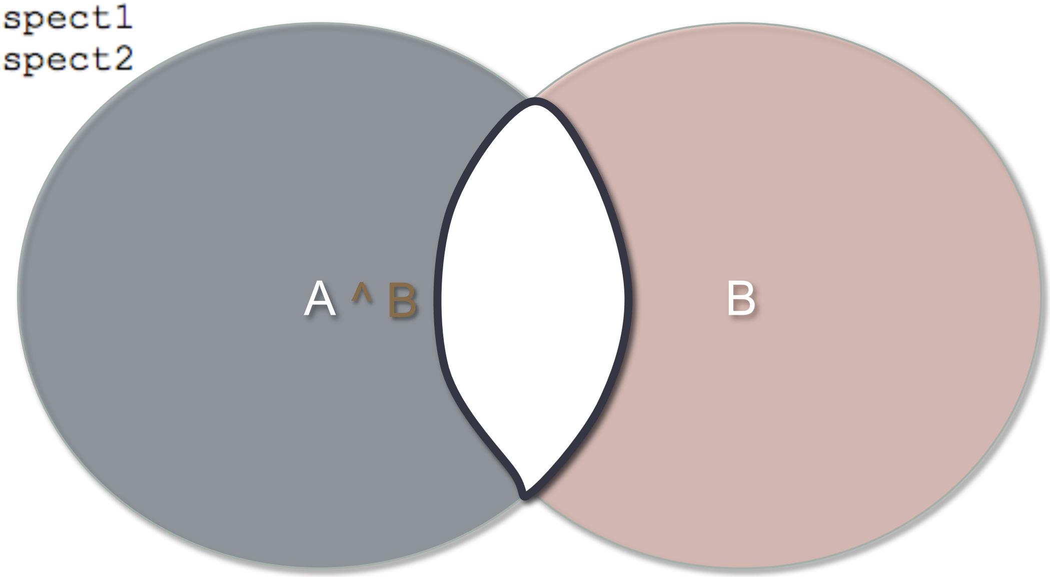
```
import IOCProfile1 as ioc1
import IOCProfile2 as ioc2
import IOCProfile3 as ioc3
import suspectprofile as suspect

iocs = ioc1.IOCProfile1()    \
       | ioc2.IOCProfile2() \
       | ioc3.IOCProfile3()
suspect = suspect.Suspect()

print iocs & suspect
```



Symmetric Difference

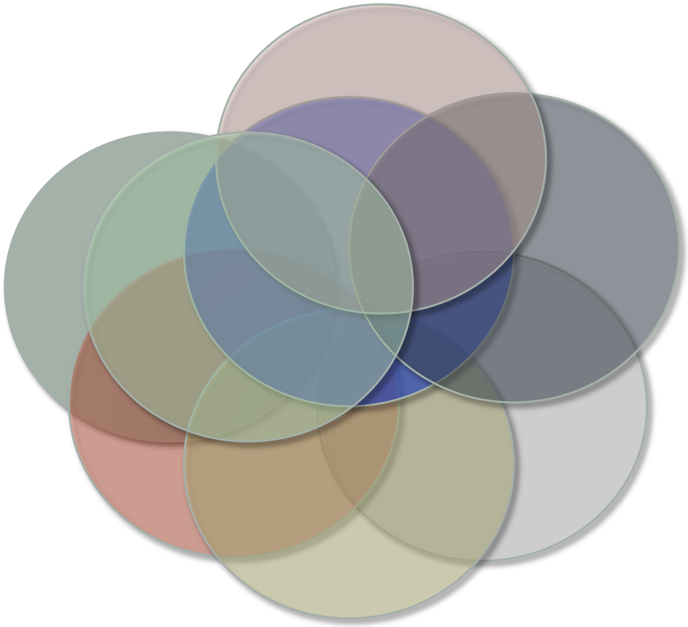
- New set with elements from either A or B, but not both
- Useful for finding items that are unique to each machine
- A ^ B

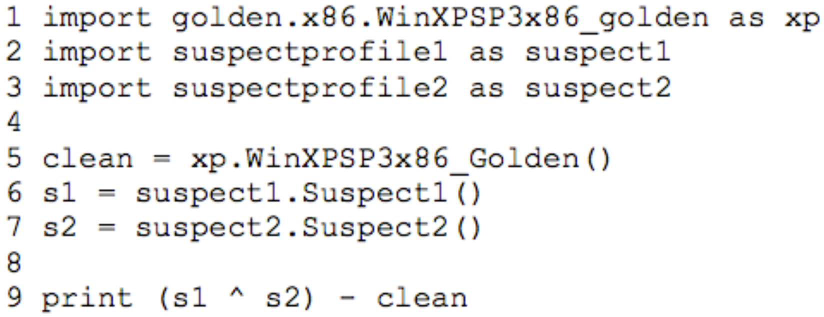
```
import suspectprofile1 as suspect1
import suspectprofile2 as suspect2

s1 = suspect1.Suspect1()
s2 = suspect2.Suspect2()

print s1 ^ s2
```



Multiple Profiles

- We can use this logic against several machines at once
- Each machine (or each software/malware sample) has its own profile
- We can combine them or use differences/ intersections to see their relationships
- Profiles have different output options:
  - Text, JSON, CybOX and Profile (Python code)





```
import golden.x86.WinXPSP3x86_golden as xp
import suspectprofile1 as suspect1
import suspectprofile2 as suspect2

clean = xp.WinXPSP3x86_Golden()
s1 = suspect1.Suspect1()
s2 = suspect2.Suspect2()

print (s1 ^ s2) - clean
```

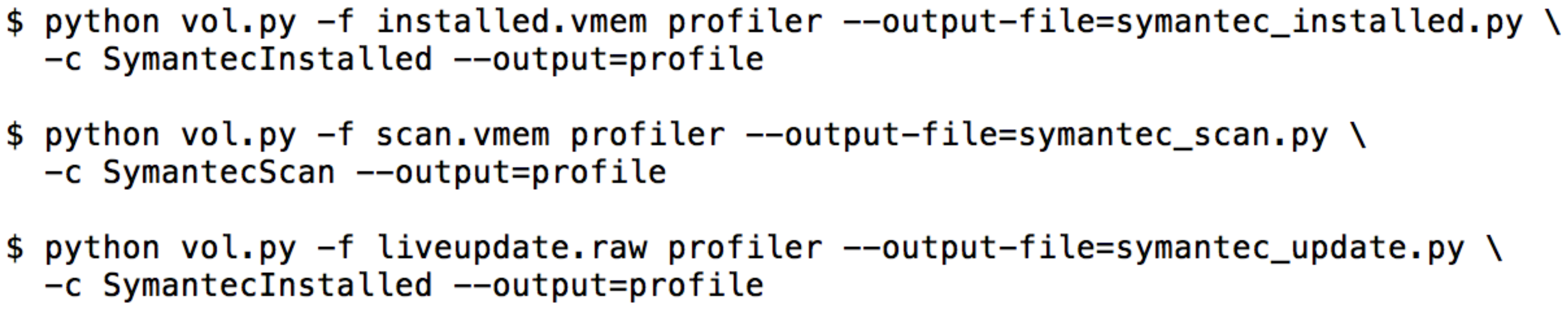
Profiler Plugin

- The profiler plugin basically collects all of these things about the machine and outputs them in one of the following outputs:
  - Text
  - JSON
  - CybOX
  - Profile
- This plugin can be inherited and extended to use other profiles using any of the previously mentioned logic

Profiler Plugin (continued)

- Observing various states of Symantec AV
- Create the profiles:

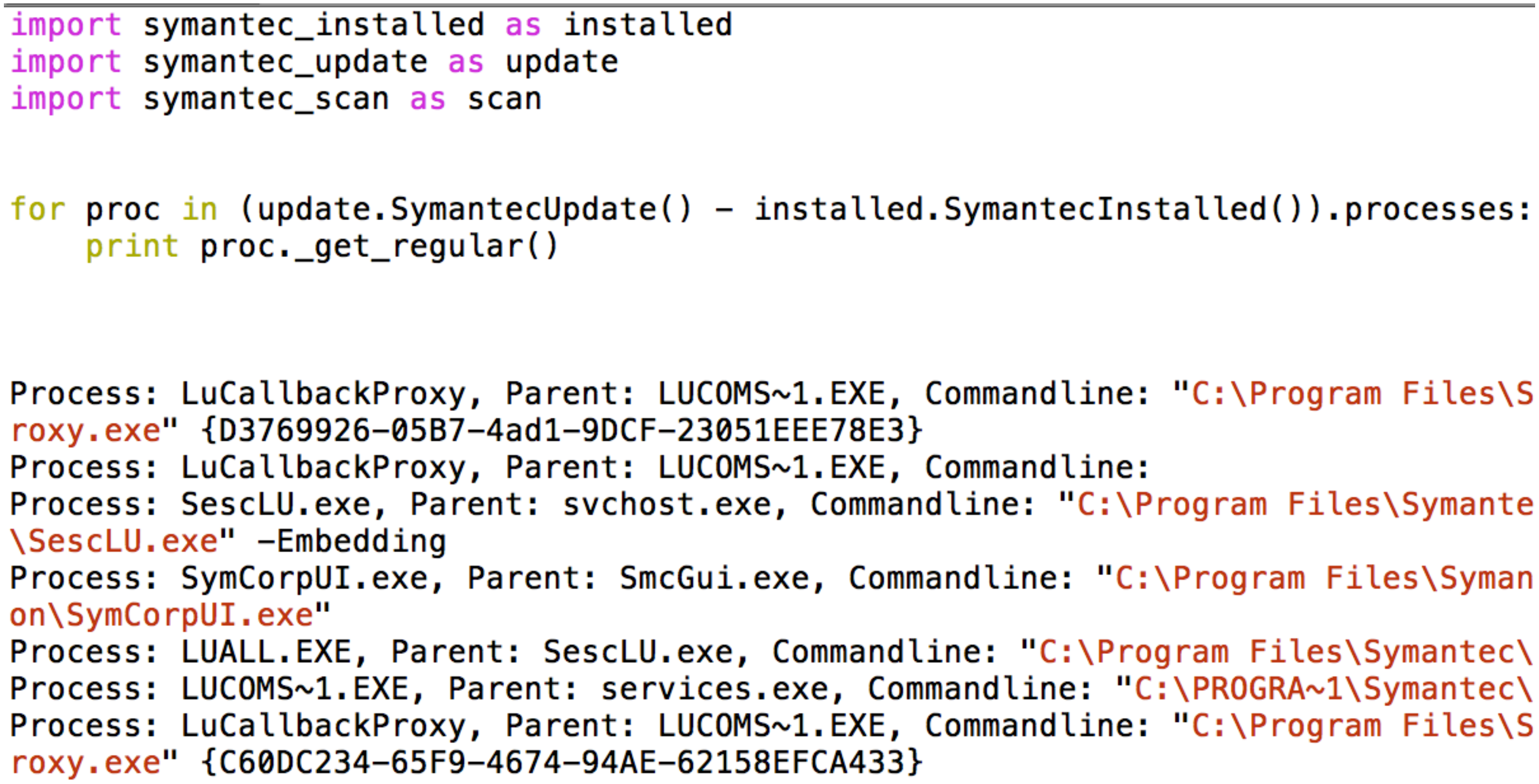
-c ClassName for profile
--output=profile makes sure to save the output as a profile



```
$ python vol.py -f installed.vmem profiler --output-file=symantec_installed.py \
  -c SymantecInstalled --output=profile

$ python vol.py -f scan.vmem profiler --output-file=symantec_scan.py \
  -c SymantecScan --output=profile

$ python vol.py -f liveupdate.raw profiler --output-file=symantec_update.py \
  -c SymantecInstalled --output=profile
```

Profiler Plugin (continued)

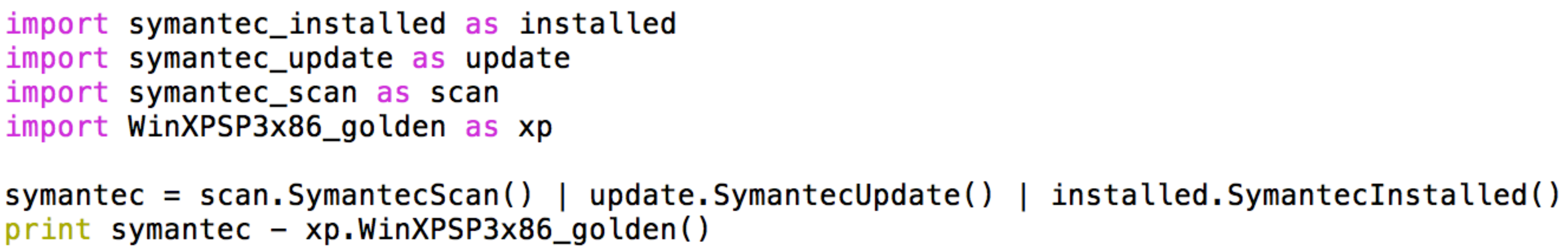
- Let’s see the difference between a fresh install and an update:



```
import symantec_installed as installed
import symantec_update as update
import symantec_scan as scan


for proc in (update.SymantecUpdate() - installed.SymantecInstalled()).processes:
    print proc._get_regular()


Process: LuCallbackProxy, Parent: LUCOMS~1.EXE, Commandline: "C:\Program Files\S
roxy.exe" {D3769926-05B7-4ad1-9DCF-23051EEE78E3}
Process: LuCallbackProxy, Parent: LUCOMS~1.EXE, Commandline:
Process: SescLU.exe, Parent: svchost.exe, Commandline: "C:\Program Files\Symante
\SescLU.exe" -Embedding
Process: SymCorpUI.exe, Parent: SmcGui.exe, Commandline: "C:\Program Files\Syman
on\SymCorpUI.exe"
Process: LUALL.EXE, Parent: SescLU.exe, Commandline: "C:\Program Files\Symantec\
Process: LUCOMS~1.EXE, Parent: services.exe, Commandline: "C:\PROGRA~1\Symantec\
Process: LuCallbackProxy, Parent: LUCOMS~1.EXE, Commandline: "C:\Program Files\S
roxy.exe" {C60DC234-65F9-4674-94AE-62158EFCA433}
```

Profiler Plugin (continued)

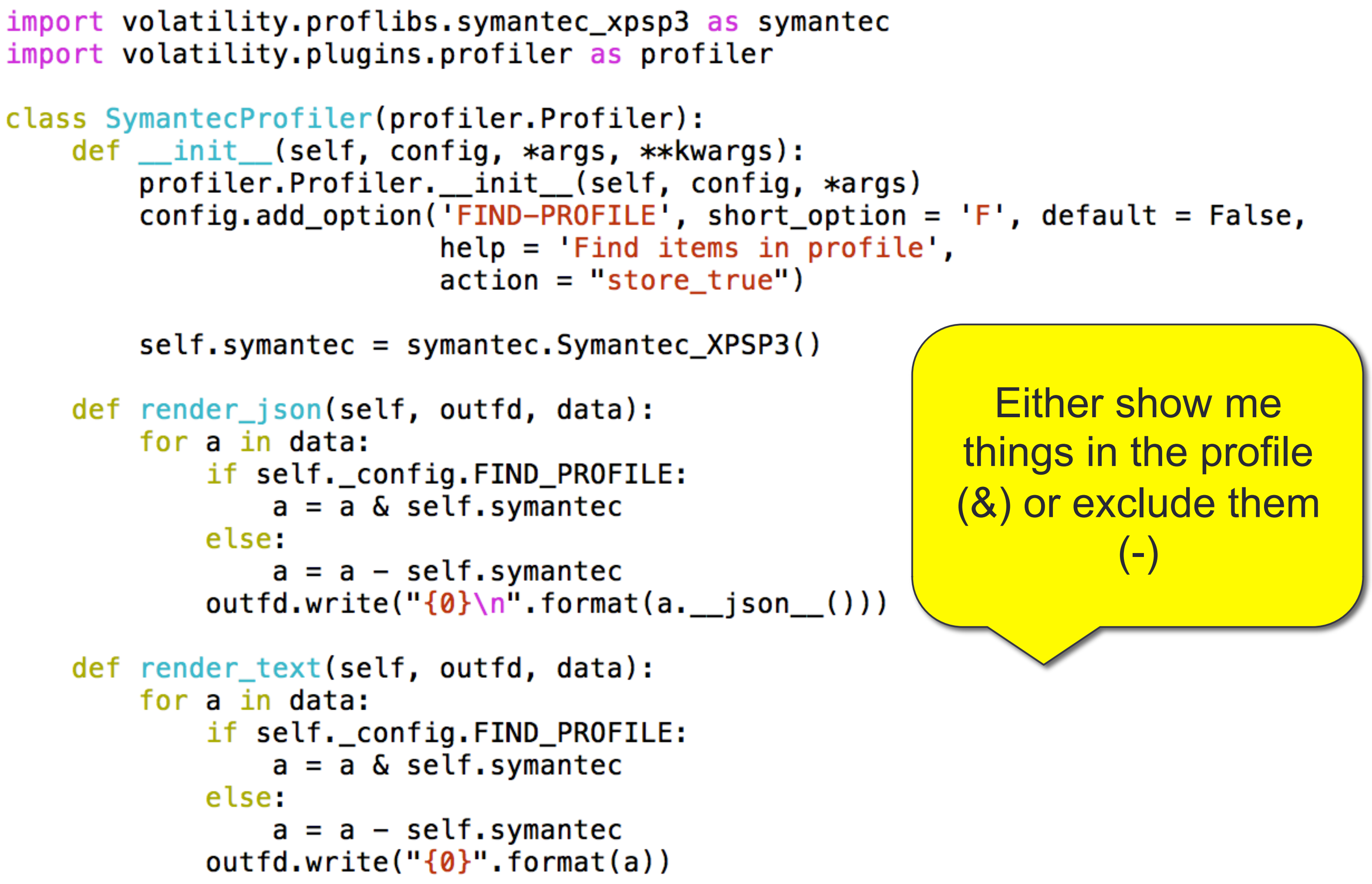
- We can combine all these profiles into one large Symantec profile:



```
import symantec_installed as installed
import symantec_update as update
import symantec_scan as scan
import WinXPSP3x86_golden as xp

symantec = scan.SymantecScan() | update.SymantecUpdate() | installed.SymantecInstalled()
print symantec - xp.WinXPSP3x86_golden()
```

- We can then use this profile in a more specific “profiler” plugin

Symantecprofiler Plugin



```
import volatility.proflibs.symantec_xpsp3 as symantec
import volatility.plugins.profiler as profiler

class SymantecProfiler(profiler.Profiler):
    def __init__(self, config, *args, **kwargs):
        profiler.Profiler.__init__(self, config, *args)
        config.add_option('FIND-PROFILE', short_option = 'F', default = False,
                          help = 'Find items in profile',
                          action = "store_true")

        self.symantec = symantec.Symantec_XPSP3()

    def render_json(self, outfd, data):
        for a in data:
            if self._config.FIND_PROFILE:
                a = a & self.symantec
            else:
                a = a - self.symantec
            outfd.write("{0}\n".format(a.__json__()))

    def render_text(self, outfd, data):
        for a in data:
            if self._config.FIND_PROFILE:
                a = a & self.symantec
            else:
                a = a - self.symantec
            outfd.write("{0}".format(a))
```

Either show me things in the profile (&) or exclude them (-)

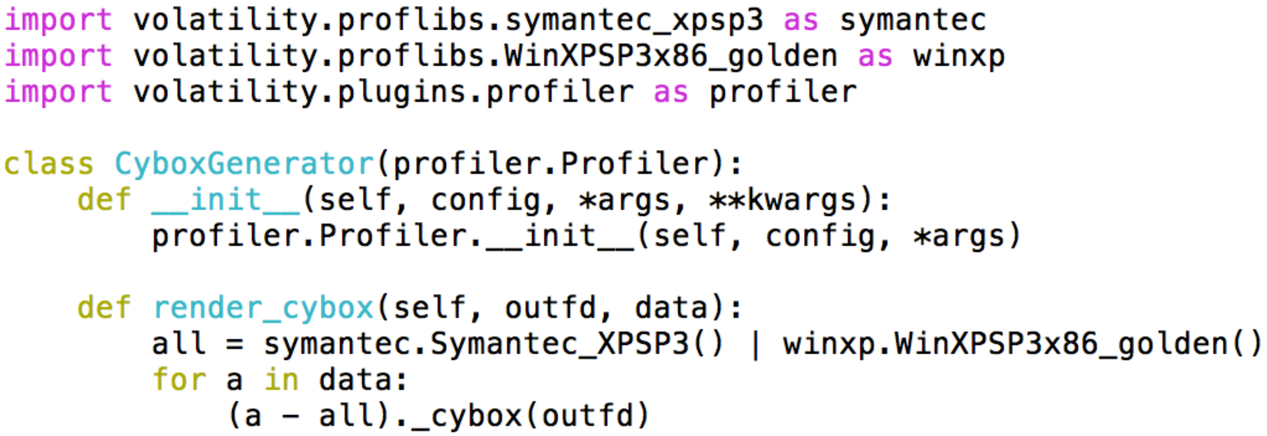
Profiler Plugin Discussion

- We can use the same idea to profile and monitor live machines
- Imagine you have a baseline of machines in the enterprise
- You can use F-Response or EnCase to sample a machine live and subtract the baseline to see if anything stands out
- You can then take that output and feed the legit items back into your baseline or create a new “blacklist” profile or CybOX package

CybOX (IOC) generation

- You can easily generate CybOX observables with the profile code
- Methodology:
  - Create a baseline for your machine for before the malware is run
  - Then print the CybOX output for the difference of the new profile and the baseline profile

You can also combine multiple profiles for the baseline



```
import volatility.proflibs.symantec_xpsp3 as symantec
import volatility.proflibs.WinXPSP3x86_golden as winxp
import volatility.plugins.profiler as profiler

class CyboxGenerator(profiler.Profiler):
    def __init__(self, config, *args, **kwargs):
        profiler.Profiler.__init__(self, config, *args)

    def render_cybox(self, outfd, data):
        all = symantec.Symantec_XPSP3() | winxp.WinXPSP3x86_golden()
        for a in data:
            (a - all)._cybox(outfd)
```

DEMO

Jack Crook DFIR Challenge

- http://blog.handlerdiaries.com/?p=14
- There are 4 machines in this challenge (WinXPSP3x86 and Win2003SP0x86)
- For just a quick overview, we can take each of these machines, generate profiles and then subtract out baseline profiles for each of these operating systems

➢ It is important to note that if a baseline profile is created from similar machines beforehand, there more whitelisted items will be removed

Processes

We only have 23 processes to go through instead of 44

```
Process: ctfmon.exe, Parent: explorer.exe, Commandline: "C:\WINDOWS\system32\c
Process: msimn.exe, Parent: explorer.exe, Commandline: "C:\Program Files\Outlook E
Process: explorer.exe, Parent: mdd.exe, Commandline: C:\WINDOWS\Explorer.EXE
Process: ismserv.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\ismserv.exe
Process: ps.exe, Parent: cmd.exe, Commandline: ps \\172.16.223.47 -accepteula -c c:\windows\webui\system5.bat
Process: wc.exe, Parent: svchost.exe, Commandline: wc.exe -e -o h.out
Process: srvcsurg.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\serverappliance\srvcsurg.exe
Process: svchost.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\svchost.exe -k iissvcs
Process: cmd.exe, Parent: explorer.exe, Commandline: "C:\WINDOWS\system32\cmd.exe"
Process: msmsgs.exe, Parent: explorer.exe, Commandline: "C:\Program Files\Messenger\msmsgs.exe" /background
Process: ps.exe, Parent: cmd.exe, Commandline: ps \\172.16.223.47 -accepteula -c c:\windows\webui\system4.bat
Process: ntfrs.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\ntfrs.exe
Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o iis-memdump.bin
Process: ps.exe, Parent: cmd.exe, Commandline: ps \\172.16.223.47 -accepteule cmd /c ipconfig
Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o callb-memdump.bin
Process: PSEXESVC.EXE, Parent: services.exe, Commandline: C:\WINDOWS\PSEXESVC.EXE
Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o dc-memdump.bin
Process: inetinfo.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\inetsrv\inetinfo.exe
Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o c:\memdump-amirs.bin
Process: wins.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\wins.exe
Process: dns.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\dns.exe
Process: POP3Svc.exe, Parent: services.exe, Commandline: c:\windows\system32\pop3server\pop3svc.exe
Process: appmgr.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\serverappliance\appmgr.exe
```

Processes

This would be even more obvious if we had more true baselines

~~Process: ctfmon.exe, Parent: explorer.exe, Commandline: "C:\WINDOWS\system32\c~~
Process: msimn.exe, Parent: explorer.exe, Commandline: "C:\Program Files\Outlook E
~~Process: explorer.exe, Parent: mdd.exe, Commandline: C:\WINDOWS\Explorer.EXE~~
~~Process: ismserv.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\ismserv.exe~~
Process: ps.exe, Parent: cmd.exe, Commandline: ps \\172.16.223.47 -accepteula -c c:\windows\webui\system5.bat
Process: wc.exe, Parent: svchost.exe, Commandline: wc.exe -e -o h.out
~~Process: srvcsurg.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\serverappliance\srvcsurg.exe~~
~~Process: svchost.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\svchost.exe -k iissvcs~~
~~Process: cmd.exe, Parent: explorer.exe, Commandline: "C:\WINDOWS\system32\cmd.exe"~~
~~Process: msmsgs.exe, Parent: explorer.exe, Commandline: "C:\Program Files\Messenger\msmsgs.exe" /background~~
Process: ps.exe, Parent: cmd.exe, Commandline: ps \\172.16.223.47 -accepteula -c c:\windows\webui\system4.bat
~~Process: ntfrs.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\ntfrs.exe~~
~~Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o iis-memdump.bin~~
Process: ps.exe, Parent: cmd.exe, Commandline: ps \\172.16.223.47 -accepteule cmd /c ipconfig
~~Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o callb-memdump.bin~~
Process: PSEXESVC.EXE, Parent: services.exe, Commandline: C:\WINDOWS\PSEXESVC.EXE
~~Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o dc-memdump.bin~~
Process: inetinfo.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\inetsrv\inetinfo.exe
~~Process: mdd.exe, Parent: cmd.exe, Commandline: mdd.exe -o c:\memdump-amirs.bin~~
Process: wins.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\wins.exe
~~Process: dns.exe, Parent: services.exe, Commandline: C:\WINDOWS\System32\dns.exe~~
Process: POP3Svc.exe, Parent: services.exe, Commandline: c:\windows\system32\pop3server\pop3svc.exe
Process: appmgr.exe, Parent: services.exe, Commandline: C:\WINDOWS\system32\serverappliance\appmgr.exe

Executables

c:\windows\psexesvc.exe
c:\windows\system32\cmd.exe
c:\mdd.exe
c:\windows\system32\pop3server\pop3svc.exe
c:\windows\system32\wc.exe
c:\windows\system32\inetsrv\inetinfo.exe
c:\windows\system32\ctfmon.exe
c:\windows\system32\ntfrs.exe
c:\windows\system32\serverappliance\srvcsurg.exe
c:\windows\webui\ps.exe
c:\program files\messenger\msmsgs.exe
c:\windows\system32\wins.exe
c:\windows\system32\dns.exe
c:\windows\system32\serverappliance\appmgr.exe
c:\itshare\mdd.exe
c:\program files\outlook express\msimn.exe
c:\windows\system32\ismserv.exe

We only have 17 exes to go through instead of 31

Again, this would be even more obvious if we had more true baselines

DLLs

c:\windows\system32\imm32.dll

c:\windows\system32\ismsmtp.dll

c:\windows\system32\msctf.dll

c:\windows\system32\6to4ex.dll

c:\windows\system32\ntdsbsrv.dll

c:\windows\system32\iismap.dll

c:\windows\system32\ddraw.dll

[snip]

Backdoor DLL stands out in only a handful of DLLs

Jack Crook DFIR Challenge

- Now we have a starting point in our investigation
- We have:
  - ➢Several processes of interest
  - ➢Several files of interest
  - ➢A DLL of interest
  - ➢Plus several API Hooks that we can also investigate (not shown)
- We can easily see which machines have these items of interest and investigate them more thoroughly as needed
- Also, one of the machines was not compromised, so we shouldn’t find these items of interest in its profile

Conclusion

We can see how we can use profiling in order to:

- Figure out artifacts from installed software/malware
- Create CybOX IOC packages from VMs/sandboxes
- Easily use profiles to find relationships between machine or software/malware artifacts
- Quickly cut through lots of data to find outliers

Questions?

Email: jamie.levy@gmail.com
Twitter: @gleeda

Upcoming trainings:

- November 11th-15th 2013: Reston, VA
- January 20th-24th 2014: San Diego, CA
- June 9th-13th 2014: London, UK

References

- CybOX http://cybox.mitre.org/
- Leveraging CybOX with Volatility http://volatility-labs.blogspot.com/2013/09/leveraging-cybox-with-volatility.html
- Python sets http://docs.python.org/2/library/sets.html
- Baseliner EnScript https://github.com/gleeda/misc-scripts/blob/master/EnScripts/Baseliner.EnScript