

Inside commercial malware sandboxes

 web.archive.org/web/20210613070852/https://albocoder.github.io/malware/2021/06/01/SandboxStudy.html


Erin Avllazagaj


June 1, 2021


Jun 1, 2021

 Koro Sensei sandbox

Introduction

In this blog post I will explore the commercial malware sandboxes. It appears malware is **allowed** to access the internet in many sandboxes . So with that access I decided to collect all the environmental features I could think of and send them to my discord channel. Since Python is the language with the largest library support ever (maybe?) I wrote a python script to take many environment features through `psutil`, `platform`, `cpuinfo` etc. The whole *python code* is here.

Then we use `pyinstaller` to create a single executable (~10 MB ) using a command like this: `pyinstaller --onefile --noup sandbox-env-stealer.py`. Then for the sake of completeness we compressed the executable in the latest upx and uploaded to VirusTotal. After a few crashes we ended with 4 samples:

- [47ed17bdea1dab10fdee...](#)
- [07a783fc3ae6a065dc0b...](#)
- [e472c0493a9a35b7975c...](#)
- [88d38301327da310c5c0...](#) (disregard the file name )

Of course, you can download the data I collected and play for yourself.

Experiment setup

I submitted one of the first samples on *17/03/2020* (because corona lockdown was getting boring). Then life happened. After deciding to revisit this project I submitted the other 4 samples around March 2021. I collected the following features (some omitted for brevity):

- Windows version + arch
- CPU name + core count
- local and internet IP address
- CPU counters
- disk partitions and counters

What did learn?

OS version

First, I noticed that **78%** of all the sandboxes run Windows 7 build 7601 (the most famous pirated version 🐸). That accounts for 40 out of the 51 executions. The table below illustrates it all. As it seems, the only versions are Windows 7 (build 7600 and 7601) and some flavor of Windows 10. A malware sample will have a high chance of being run on a real machine if the detected OS is anything but the following.

OS version	# execs
Win 7 build 7601	40
Win 10.0.18362	3
Win 7 build 7600	3
Win 10.0.14393	3
Win 10.0.17134	2

Timeline analysis

In this section I will analyze some runtime features of the sandboxes. Here there are some interesting features malware authors can use to quickly identify sandboxes. There are also some lessons I learned from looking at the executions.

First off, I wanted to see how many executions we were getting per malware and how often. This is particularly important since in [our paper](#) we showed that a malware need to run in 3 random environments at least every 3 weeks.

It appears, for the oldest malware sample **10** executions appear on the same day as the submission, **1** execution 1 day later, **3** executions 33 days later and **2** executions 62 days later. This means that the sample was deemed interesting 2 months from its first “appearance”. However, we can’t conclude that this is what happens for all the samples, with different number of VT engine detections or with more interest from individual AV vendors. For the record, I did get an execution report back from the first sample just yesterday, after clicking the “*reanalyze*” on VT 🐷. Either way, the malware was executed about 2 to 3 times every month, which is close enough to 3 weeks (that we recommend in our paper), but we demonstrated in the paper that on average 1 week stale of data decreases the detection rate.

Environment analysis

This sections will show some environment features that the malware can read.

The running processes

A common routine seen on many malware and benign samples is that of iterating the running processes. I (as in python libraries) use a similar routine to retrieve the running processes. The table below shows some of the running processes and the number of machines they were seen to run on. Something interesting we can see here are the “special” programs. In some machines we see the appearance of `bitcoin-qt.exe` , `infinium.exe` etc, while in some others we see `steam.exe` , `SteamService.exe` etc, in some others `filezilla.exe` or `centralcreditcard.exe` . This is usually done to see if the malware is a crypto miner, a game hack, a file infector or a point-of-sale malware respectively.

Processes	Number of machines
...	...
conhost.exe	47
lsass.exe	46
spoolsv.exe	46
wininit.exe	46
smss.exe	46
System Idle Process	46
explorer.exe	46
winlogon.exe	46
System	46
services.exe	46
opera.exe	43
firefox.exe	43
dwm.exe	41
lsm.exe	40
Skype.exe	29
OSPPSVCSVC.EXE	27
taskeng.exe	24
...	...
bitcoin-qt.exe	14

infium.exe	14
qip.exe	14
communicator.exe	14
bitcoind.exe	14
steam.exe	14
sppsvc.exe	12
vslvqrlijtvi.exe	12
splwow64.exe	12
artifact.exe	10
fontdrvhost.exe	10
SteamService.exe	9
SearchProtocolHost.exe	9
SearchFilterHost.exe	9
GoogleUpdate.exe	8
dllhost.exe	8
ioynossujx.exe	8
wqwupyjrsx.exe	8
notepad.exe	8
taskmgr.exe	7
ONENOTEM.EXE	7
sihost.exe	6
vmtoolsd.exe	6
SearchUI.exe	6
TrustedInstaller.exe	6
1a7446534577bab0984f5eb275bdf1f43ed92dfc.exe	6
ivpvkimw.exe	6
utg2.exe	6

Helios12.exe	5
OfficeClickToRun.exe	5
OmniPOS.exe	5
ifs.exe	5
EdcSvr.exe	5
Registry	5
OUTLOOK.EXE	5
wmpnetwk.exe	5
CentralCreditCard.exe	5
8lfuaq3.exe	4
SophosFileScanner.exe	4
e5d46536.exe	4
SgrmBroker.exe	4
nvtray.exe	4
hmpalert.exe	4
350befaf.exe	4
1a34b48b.exe	4
avp.exe	3
SEDSservice.exe	3
Tcpview.exe	3
mp3tray.exe	3
InstallRite.exe	3
SavService.exe	3
scap.exe	3
mscorsvw.exe	3
SAVAdminService.exe	3
sdrsservice.exe	3


StartMenuExperienceHost.exe	3
Procmon.exe	3
backgroundTaskHost.exe	3
HttpLog.exe	3
ShellExperienceHost.exe	3
taskhostw.exe	3
popwack.exe	3
msdtc.exe	3
sedsvc.exe	3
avpui.exe	3
procexp64.exe	3
Procmon64.exe	3
SophosCleanM64.exe	2
module-cargo.exe	2
WindowsInternal.ComposableShell.Experiences.TextInput.InputApp.exe	2
MemCompression	2
DS5FEMT81XbOM0LW.exe	2
SecurityHealthService.exe	2
88d38301327da310c5c00a0b3ae8209730e033f3c57575a447096d94a647e816.exe	2
swc_service.exe	2
KMSAuto Net.exe	2
3sO7_zsS.exe	2
SophosFS.exe	2
TiWorker.exe	2
s35zi2y.exe	2
ld8itap.exe	2
SophosNtpService.exe	2

GoogleUpdateSetup.exe	2
SSPService.exe	2
swi_service.exe	2
pythonw.exe	2
WbjiETqs.exe	2
ctfmon.exe	2
pw.exe	2
swi_filter.exe	2
hltpwzd.exe	2
Sophos.Encryption.BitLockerService.exe	2
uniform-98682.exe	2
8eu3umxnf.exe	2
SophosIPS.exe	2
union_rechnung_install_39213.exe	2
userinit.exe	2
gzqhbp.exe	2
05a62b54.exe	2
SophosSafestore64.exe	2
sdcservice.exe	2
swi_fc.exe	2
3myJvMOn.exe	2
WmiApSrv.exe	2
cuckoo-47ed17bdea1dab10fdee7f61dff8b8f33ad5d092b3e1e5f0f5a3522a27798183.exe	2
mtwebooS.exe	2
05a62b54e6e32c406f33d22634b03fe8.exe	2
SnrUNWUv.exe	2
msiexec.exe	2

follow-sneaky-on-twitch.exe	2
SophosHealth.exe	2
...	...
absolutetelnet.exe	1
gmmeby.exe	1
outlook.exe	1
isspos.exe	1
qgksae.exe	1
totalcmd.exe	1
ncftp.exe	1
whatsapp.exe	1
igfxCUIService.exe	1
winscp.exe	1
coreftp.exe	1
barca.exe	1
socbristol.exe	1
rundll32.exe	1
accupos.exe	1
bedrooms-story-avoid.exe	1
active-charge.exe	1
fling.exe	1
vrmafl.exe	1
gmailnotifierpro.exe	1
pidgin.exe	1
diaryrecent.exe	1
creditservice.exe	1
operamail.exe	1

centralcreditcard.exe	1
unsecapp.exe	1
AutoKMS.exe	1
medical reservoir.exe	1
alftp.exe	1
netsh.exe	1
wspsvc.exe	1
scriptftp.exe	1
spgagentservice.exe	1
slwvdq.exe	1
edcsvr.exe	1
Sysmon.exe	1
american.exe	1
wmi64.exe	1
flashfxp.exe	1
axdnik.exe	1
webpagepioneer.exe	1
skype.exe	1
Memory Compression	1
fpos.exe	1
ApplicationFrameHost.exe	1
filezilla.exe	1
spcwin.exe	1

Unfortunately, we can also see things like `cuckoo-47ed17bdea1dab10fdee7f61dff8b8f33ad5d092b3e1e5f0f5a3522a27798183.exe` or `1a7446534577bab0984f5eb275bdf1f43ed92dfc.exe` which is simply the checksum hash of

the sample. An attacker can simply compute the popular checksums (the ones on the [details](#) section in VT) and see if its name is any of those and terminate .

Machine names

One thing I wanted to see is the username the malware in the sandbox will run on. For the most part I was underwhelmed. It appears the machine name remains the same across executions, which may be easy for an attacker to just submit bogus “malware” just to harvest all the machine names. In terms of the context I did see that sometimes the sample was executed as [Administrator](#) meaning that some sandboxes give the “malware” admin privileges. This is done to make sure that malware can execute.

Username	Number of executions
art-PC	7
PC-4a095e27cb	5
w7sb64-01	3
w7x64	3
z97OtiH0P4v-PC	2
AMAZING-AVOCADO	2
mgvwazbfy	1
WIN-IMCGBF4ZV49	1
HAPUBWS-PC	1
DESKTOP-VXO5LFI	1
QGI87k-PC	1
IOFXBF742797820	1
WIN-QVM8C8V0B1E	1
XDuwTfOno	1
dillon	1
WIN-BROIECEJLD2	1
CZAC38122213349	1
Xj8Uz1ljKXdt-PC	1
DESKTOP-ILTLN65	1

OngJeNyHDSzmoUHw	1
Anna-PC	1
DcXhINjDfk-PC	1
WIN-FYI1QSCHQHU	1
WIN-LQOUJKDIROR	1
XGUW12547433669	1
AL MUKALLA	1
WIN-KRAZH63AMC2	1
PC	1
GCSPJUXFT667743	1
DESKTOP-D019GDM	1
WIN-UJ21PNWQMR2	1
Lisa-PC	1
vX3juZIWR5Wy-PC	1
WIN-TGCR76AWNUB	1
WIN-U1DY5TBDUI7	1


Hardware analysis

CPU and memory analysis

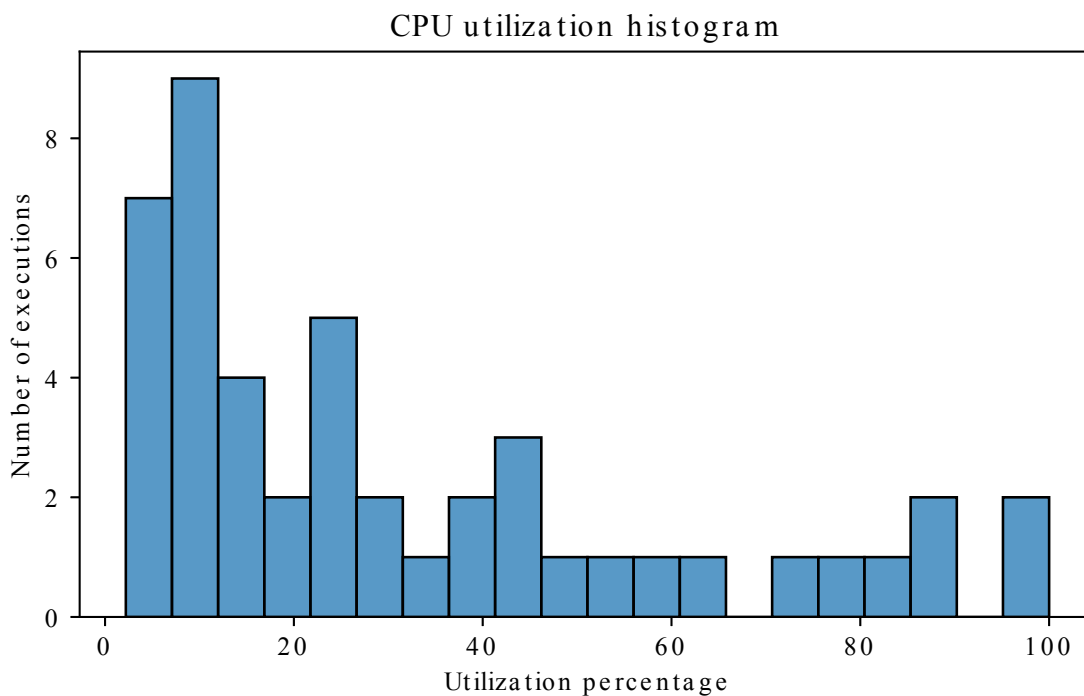
Next we are interested to know what CPU are the sandboxes running on? I realized the `platform.processor()` was not returning the actual CPU name, but it was too late when I realized so I used the available information from this command and scrapped some tables online to get the processor name ([here](#) is the *pkI* file that aggregates all data). In the latest version I added this awesome library called `cpuinfo`, so in case you need to collect your own data the `cpuinfo` will do the work.

number of cores	potential cores
1	{ 8 , 4 }
1	{ 8 , 4 }

number of cores	potential cores
1	{ 8 , 4 }
4	{ 16 }
2	{ 8 , 4 }
1	{ 8 , 4 }
1	{ 8 , 4 }
1	{ 8 , 4 }
1	{ 8 , 4 }

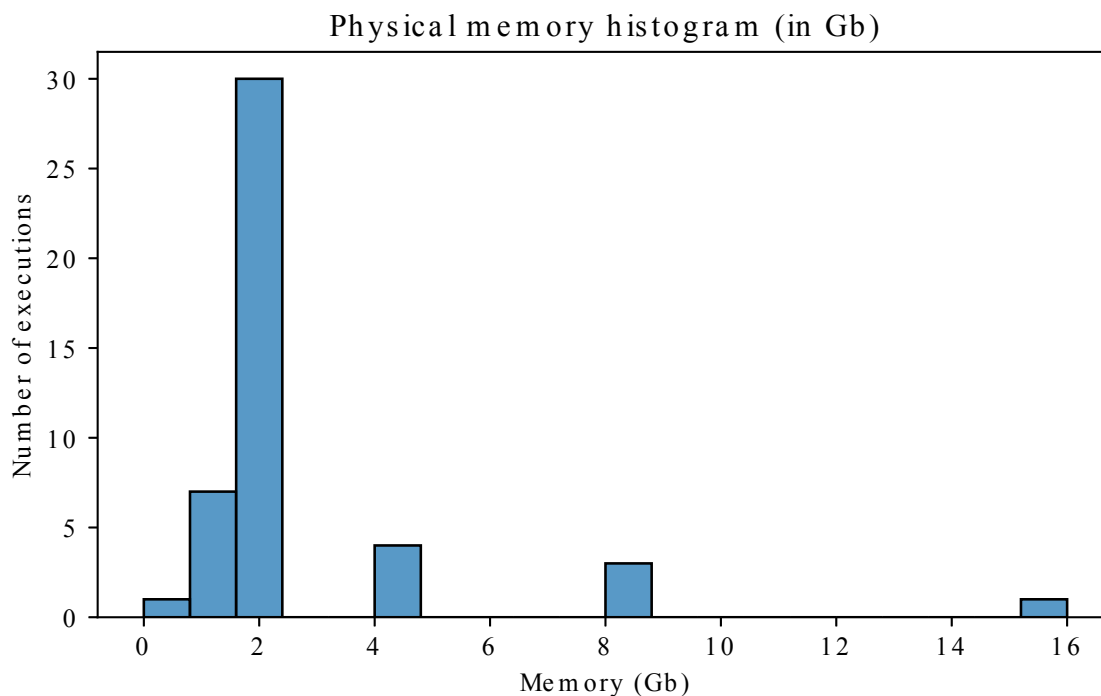
As hypothesized, the machine has the wrong number of cores for the CPU name (VICTORY ). In the 9 machines, I found that the most used CPUs are Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz , Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz and Intel(R) Xeon(R) W-2140B CPU @ 3.20GHz . The machines seem to have 1 core for the most times but the name is that of a cpu with 4 or 8 cores.

I also checked the CPU utilization rates. Hypothetically, the sandboxes would have low CPU utilization, since they are only meant to run the malware.



While most of the sandboxes have a utilization less than 20% there are still cases where sandboxes have 80% or even 100%. I believe the CPU utilization cannot be a feature to distinguish between sandboxes and real machines.

Lastly for this section we look into memory. How much memory can commercial sandboxes spare?

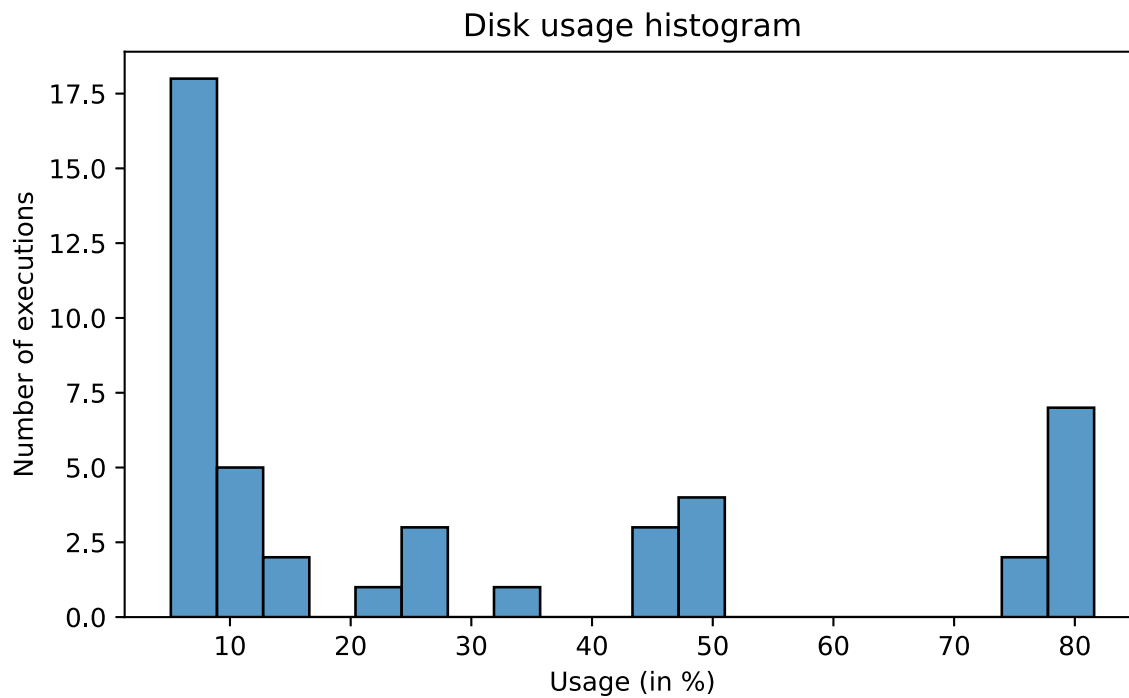
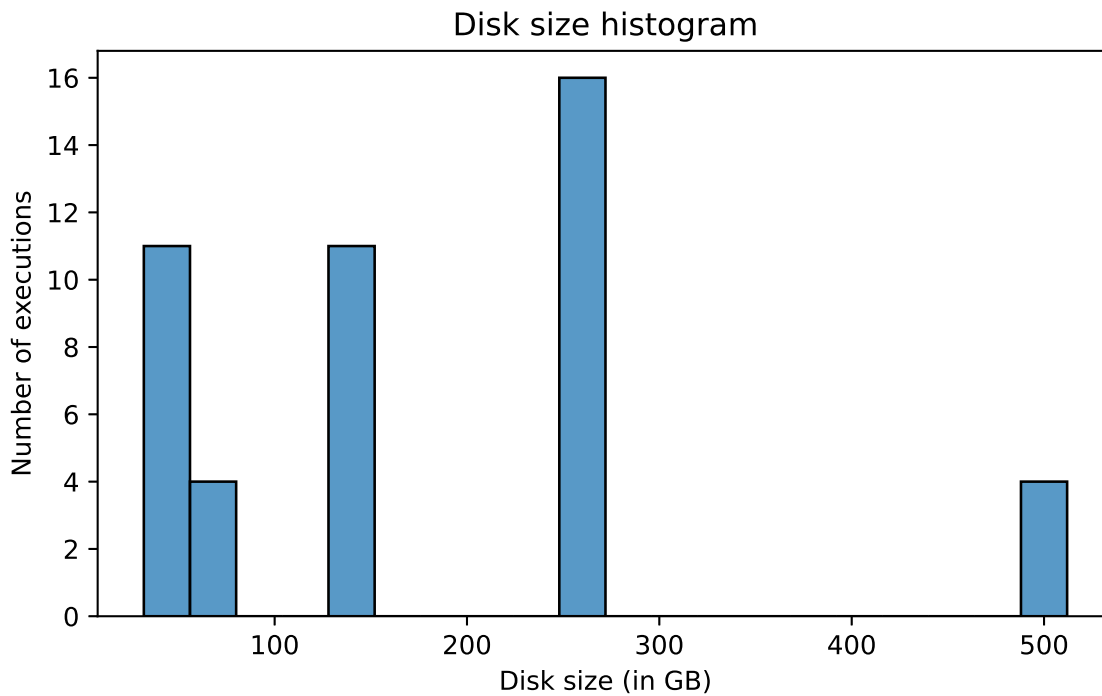


It appears most of the machines have 1 to 2 GB of RAM to spare. In 2 cases I found machines with 512MB of memory like its 2010 🤔. Props to the AV vendor(s) that allocated 4, 8 and 16GB for a sandbox to run a malware sample.

Disk Partitions

An interesting result here is that in ALL sandboxes there seems to be 1 disk partition of type `cdrom` 🐸. Hypothetically this may happen because during the VM creation the Windows installation disk was left inserted (more work needs to be done to verify this).

For the main disk partition (where Windows is installed) we notice quite a spectrum of sizes, but the usage in general is quite low (except for the `32GB` disks with high usage because Windows is installed there). In 2021 I wouldn't expect there to be many machines left with 32GB in the main drive, so this may raise some suspicions from the attackers perspective.



Network Interfaces

In general 2 to 3 interfaces and **5** network interfaces in 1 sandbox. The most prevalent are of course `Loopback Pseudo-Interface 1` and `Local Area Connection`. Then I noticed some pattern of `isatap.{<random GUID here>}` and `Teredo Tunneling Pseudo-Interface`. Upon further analysis I found that these interface exist to enable IPv6 communication, so there is nothing special that an attacker can use here 🤪.

Battery

Not a single sandbox had a battery. Attackers right now pulling a high IQ strat 🤖.

Time analysis


One thing its hard to keep up to date while restoring the snapshot (at least in Windows) is local time. Its a bit more difficult when you consider the geolocation the machine is supposed to be at. For this I collected the local time, the global UTC time and the external IP address of the sandbox and metadata for that IP. Thanks to [ipfy](#), [just-the-time](#), and [ipinfo](#) for the awesome service. The process I followed is pretty simple. I use the data from [ipinfo](#) to get the geolocation and convert the UTC time to the local time for that geolocation, then I calculated the time skew between the time of the geolocation and the sandboxes local time.

time difference(hours)	number of machines
-19.0	1
-9.0	2
-8.0	1
-3.0	1
-2.0	18
-1.0	4
0.0	19
3.0	1
5.0	1
9.0	2

As it appears, around **65%** of the sandboxes have some sort of a time skew (and I believe the other 35% are simply lucky to VPN on the same timezone or not even VPN at all, will get to this later 🌍).

This is a very low effort feature a real malware can use to check if its inside a sandbox given that such info can be checked for free. Even through a CnC server the attacker can measure whether the malware has made it into the sandboxes. Throughout our [paper](#) we also noticed that there are some real users' machines with an outdated clock (sometimes out of date by about 50 years 🤖) however this made up for *less than 0.001%* of the machines in the real world not 65%, so a malware author may simply choose to forgive these outdated machines just to be safe from analysis. And from a defender's prespective, PLEASE UPDATE THE CLOCK BEFORE ROUTING THE NET TRAFFIC TO NARNIA.

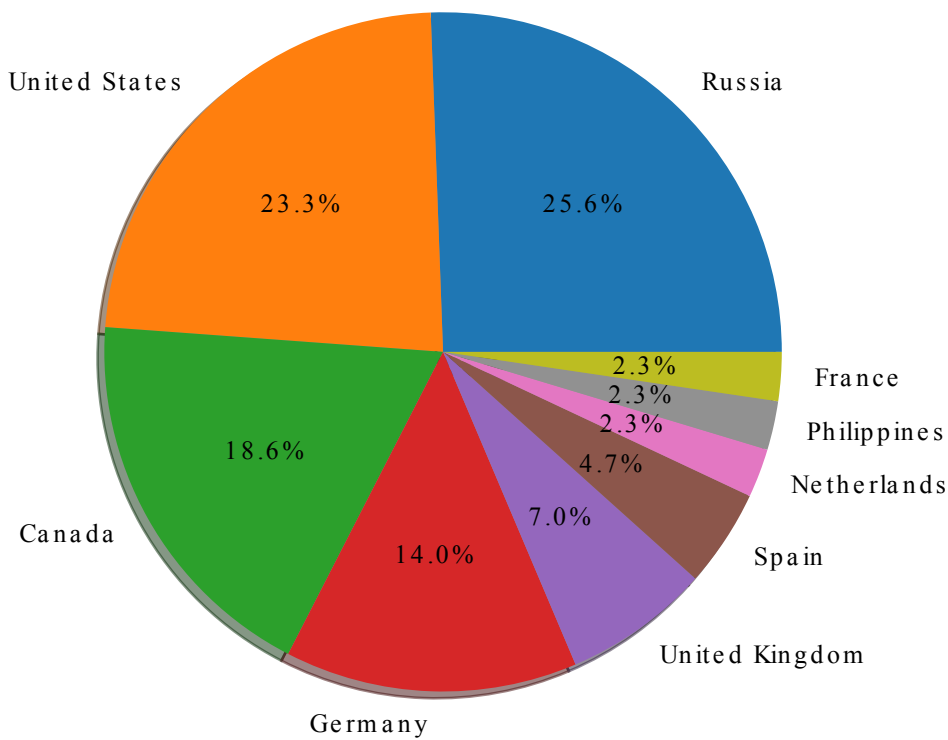
IP analysis

This is the meat of the blog post, in my opinion. Thanks again to [ipinfo](#) for the 7 day free trial. As the most interesting to me, I first looked at “Who owns these IP addresses? Who are the companies/ISPs?”. As shown in the table below the network traffic in all the sandboxes is routed through VPNs that they get from dedicated servers. It appears CrowdStrike has their own IP range that they use to route traffic (registered under their official name , hide the pain Harold).

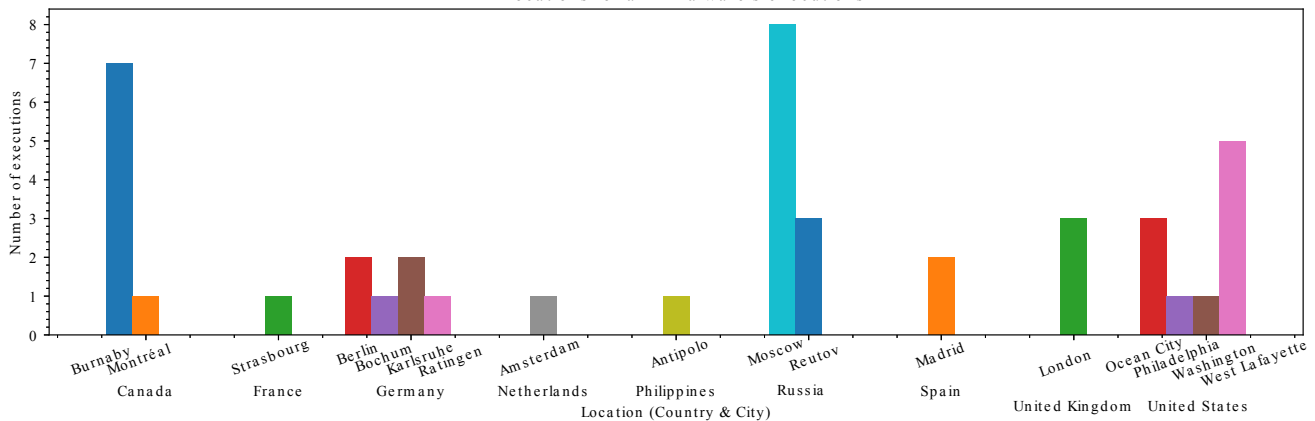
IP owner company	number of IPs in the dataset
TELUS Communications Inc.	5
Wintek Corporation	5
PJSC Vimpelcom	5
Verizon Business Special Project	3
LLC Digital Network	3
Cox Communications Inc.	3
Zwiebelfreunde e.V.	2
Bell Canada	2
111250 Russia Moscow SOVINTEL/EDN	2
1337 Services LLC	1
Vodafone D2 GmbH	1
Telecom Colocation, LLC	1
IPG	1
111250 Russia MOscow EDN/Sovintel	1
Bungee Servers SP	1
CrowdStrike Services	1
Dedicated Servers	1
LeaseWeb Netherlands B.V.	1
ARCOR AG	1
Deutsche Telekom AG	1
Datacamp Limited	1

Now we want to know where exactly are these IPs from. With this much data it's hard to say which IPs are owned by the AV vendors, since they are also just buying VPN access, but I wanted to see where they are buying from. It appears Russia, US, Canada and Germany are the most prominent countries, and in Russia, Moscow seems to be the city with the highest number of IPs.

Location of the sandbox IPs



IP locations for all 4 malware's executions



Conclusion

There is no silver bullet to detect sandboxes, but there are some features and bugs 🐞 the attacker can use to detect them. On the other hand the sandboxes can also cover these weaknesses. It's all about that arms race 🌍.