All malwares are equals, but some malware are more equals than others

Joxean Koret

About this talk

- Introduction
- A very brief history of malware
- Working with bytes
- Working with graphs
- Other approaches
- Conclusions

Introduction

During my time researching malware I have used and written many tools.

Most of the tools I've written were for analyzing malware behaviour, as well as for clustering and indexing malware samples.

In this talk, I will discuss some of the tools I developed, some tools from other people as well as some theoretical or just non public approaches.

Let's start...

A very brief history of malware

Before 1970:

- The very first idea of a malware appeared first in 1949: “Theory of self-reproducing automata” by John Von Neumann.

1980’s:

- Professor Leonard M. Adleman employs in 1981 for the first time the term “malware”.
- Elk Cloner, a virus for Apple II, appears in 1982.
- The Morris Worm infected around 6,000 boxes (10% of internet) in 1988.

A very brief history of malware

1990’s:

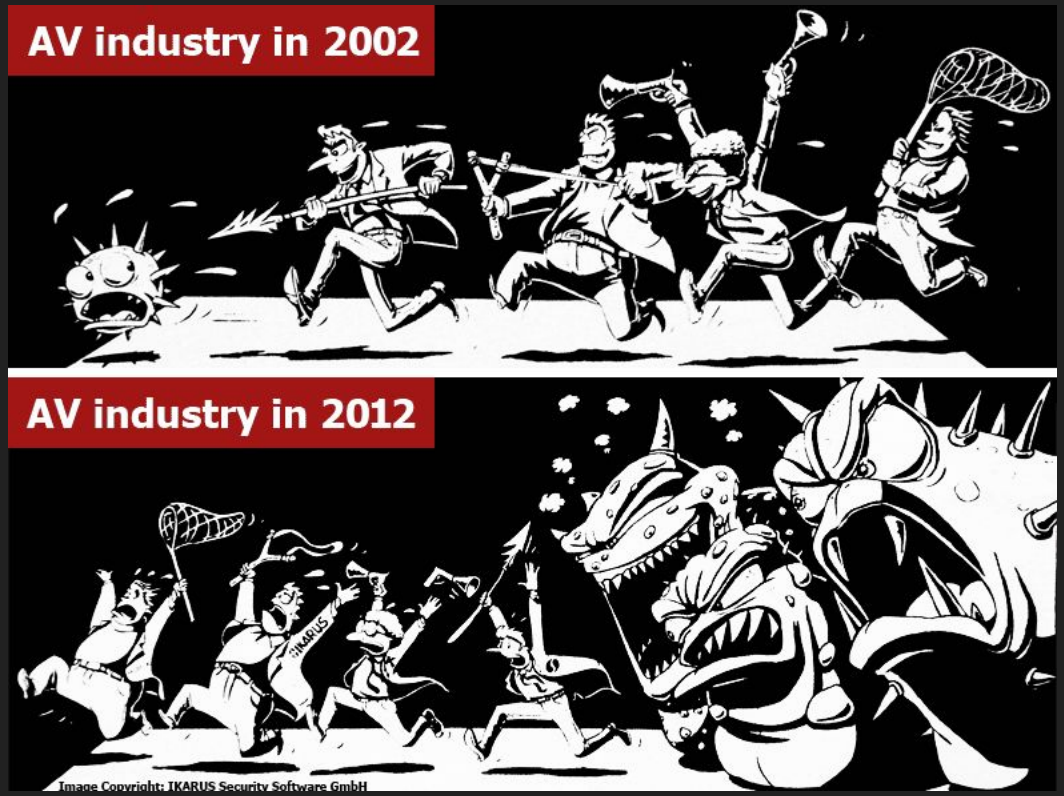
- Multiple high quality viruses for DOS appear.
- One of the most memorable ones was CIH. It’s believed to have damaged around 1 million machines.

Beginning of the 2000’s:

- These were the years of the worms: Love Letter, Sircam, Anna Kournikova, etc… were released just in 2001.
- AV companies used to work with 1 interesting malware a week: some 29A virus or the newest worm.

And then...

Awesome Picture by Ikarus GmbH



A very brief history of malware

During the first years of the 2000's in the AV industry, all the researchers worked on a specific piece of malware for 1 to 7 days, depending on its complexity.

- It wasn't the same fighting with Love Letter and against the new 29A creation.

The reason why malware was created during these days was often "ego":

- "Look what I can do!", "I guessed how to do that!", "Mine is bigger than yours!", etc…

Until it became a business. And later a tool for "cyber war".

A very brief history of malware

Today, most malware is created with the idea of getting an economic benefit (money) or for macro economic reasons (Stuxnet, the Saudi Aramco attacks, NotPetya, etc…).

Malware is a *good business*, as it can be used to:

- Steal bank data. Or crypto-wallets.
- Use compromised e-mail addresses to send spam or infected computers as zombies (botnets) or as a trampoline to attack the real target of a group of attackers.
- Or to “mine” crypto-coins, like BitCoin.
- Or to attack enemy countries governments and companies.

A very brief history of malware

Talking about the number of malware samples, we can say that “Yesterday” (2000’s) it was like one or a handful of malwares per week but today we have some new thousands per day:

- According to Panda Antivirus, in 2013 around 82,000 new malwares appeared daily.
  - They probably meant new files with different crypto-hashes, but anyway.
- According to Kaspersky, 315,000 are automatically detected and at least 125,000 new malwares appears. Daily.

Ignoring their propaganda, it's clear that the number of malwares created daily or monthly increased *significantly*.

A very brief history of malware

Due to the high volume of new malware samples to analyze, automation was the next step.

However, how can one work with big or huge malware datasets?

This is what this talk is all about...

Working with bytes

Working with bytes

One of the very first tools I used when I was working in some AV company was “ssdeep”.

- https://github.com/ssdeep-project/ssdeep

It’s a fuzzy hashing tool that, basically, takes as input files, regardless of their format, and output fuzzy signatures.

With “fuzzy”, what I mean is that the hash, in opposite to a cryptographic hash, is created with the aim of causing many collisions, thus, finding similar files.

SSDEEP

Let’s see a simple demo:

```
$ ssdeep -l *

ssdeep,1.1--blocksize:hash:hash,filename

768:hr+iJCLqH7WhlTVam6d4iUYr+iJCLqH7Whlz:9+i+qbmTAm8cA+i+qbmz,"7253d6e981a7fce89e9ae5c0bff04133a41bb0e0"

768:hr+iJCLqH7WhlTVam6d4FQYr+iJCLqH7Whlz:9+i+qbmTAm8DA+i+qbmz,"a6464cfd5a72724840ff7e3db31a5eaba7d68f3a"

768:hr+iJCLqH7WhlTVam6d40mYr+iJCLqH7Whlz:9+i+qbmTAm8aA+i+qbmz,"c5724188de1f167e23994dbf331397897418ba21"

768:hr+iJCLqH7WhlTVam6d44VYr+iJCLqH7Whlz:9+i+qbmTAm8BA+i+qbmz,"cb5254fa58190bbf04c169a666a2be995e61f6c1"

768:hr+iJCLqH7WhlTVam6d4gZYr+iJCLqH7Whlz:9+i+qbmTAm8ZA+i+qbmz,"dd0f0e8b8eb7adcf71d23f94937e01da4dc14899"
```

SSDEEP

Finding clusters:

```
$ ssdeep -l -gp *

** Cluster size 5

7253d6e981a7fce89e9ae5c0bff04133a41bb0e0

a6464cfd5a72724840ff7e3db31a5eaba7d68f3a

c5724188de1f167e23994dbf331397897418ba21

cb5254fa58190bbf04c169a666a2be995e61f6c1

dd0f0e8b8eb7adcf71d23f94937e01da4dc14899
```

SSDEEP

However, both the tool (ssdeep) and the techniques used by it (fuzzy hashing) has various problems:

- The block size in ssdeep is calculated automatically depending on the input's size, it cannot be changed and, naturally, 2 signatures with different block sizes cannot be matched.
- Fuzzy hashing techniques work with bytes and will only detect modifications at byte level.

Let's see the first mentioned problem easily...

Example: Problems with SSDEEP

```
$ cp /bin/ls .; cp /bin/cp . ; ssdeep -l *
ssdeep,1.1--blocksize:hash:hash,filename
3072:TL2A2F0sAu2UJGTKrMvsw0bsgXwmPBTZhDDdlLgOHV58FTOOig:f80j1TOMvjQ7yc,"cp"
3072:jj2ncgHbmsudiNaIZM/T7/5E0GkDKFYW:jjBKmVdHh/3/5ETeCY,"ls"
$ cat /bin/ls /bin/cp > new; ssdeep -l *
ssdeep,1.1--blocksize:hash:hash,filename
3072:TL2A2F0sAu2UJGTKrMvsw0bsgXwmPBTZhDDdlLgOHV58FTOOig:f80j1TOMvjQ7yc,"cp"
3072:jj2ncgHbmsudiNaIZM/T7/5E0GkDKFYW:jjBKmVdHh/3/5ETeCY,"ls"
6144:jjBKmVdHh/3/5ETeCYo80j1TOMvjQ7yc:jjBn4TeCt8KOMvU7z,"new"
```

SSDEEP

As we can see, because the block size changes according to the file size and it cannot be configured, neither “cp” nor “ls” are matched against the “new” binary we created.

While working with medium to big malware datasets I noticed it was a huge problem for me.

As so, I decided to do something to get rid of that problem and end up writing another fuzzy hashing tool: DeepToad.

DeepToad

Yet another Open Source fuzzy hashing tool and library:

- https://github.com/joxeankoret/deeptoad

It was created mainly to work with medium to big sized malware datasets and has various benefits over ssdeep:

- Block size, output signature size and “fuzziness” are configurable.
- It generates 3 different signatures.
- It can automatically cluster files in sub-directories.

Let’s see the previous example with ssdeep...

DeepToad block size

```
$ cp /bin/ls .; cp /bin/cp .; cat ls cp > new

$ sha1sum *

9a1d3884753a41f9ab7a87d309fed4bb4078b4e8  cp

b79f70b18538de0199e6829e06b547e079df8842  ls

c7c924be115c574fcf0903fa96a3acf6c21def76  new

$ deeptoad.py *

OzscHBwcY2NjY8nJyckcHBwcQEBAQMnJ;HBxjY8nJHBxAQMnJx8ecnH19CQn19UZ©ODjKykVFqKgkJD4+xsb9/a6uNTVeXs/P;cp

TEze3t7exsbGxpKSkpLh4eHhvr6+vqys;3t7GxpKS4eG+vqysxMR7ezk5xcUmJhcX;SUlgYFFRDw/AwAUFLCz5+eXlbm7a2jc3;ls

TEze3t7exsbGxpKSkpLh4eHhvr6+vqys;3t7GxpKS4eG+vqysxMR7ezk5xcUmJhcX;ODjKykVFqKgkJD4+xsb9/a6uNTVeXs/P;new
```

DeepToad

The block size in DeepToad is configurable and, also, it doesn’t change depending on the inputs size. As so, the previous example that was failing for us with ssdeep, is no more a problem.

Also, DeepToad generates 3 different signatures:

- The 1st column is the hash calculated from the start to the end.
- The 2nd column is the hash calculated from the end to the start.
- The 3rd column is calculated taking a block from the start, them another from the end, then 2nd from the start, then the 2nd from the end, and so on…

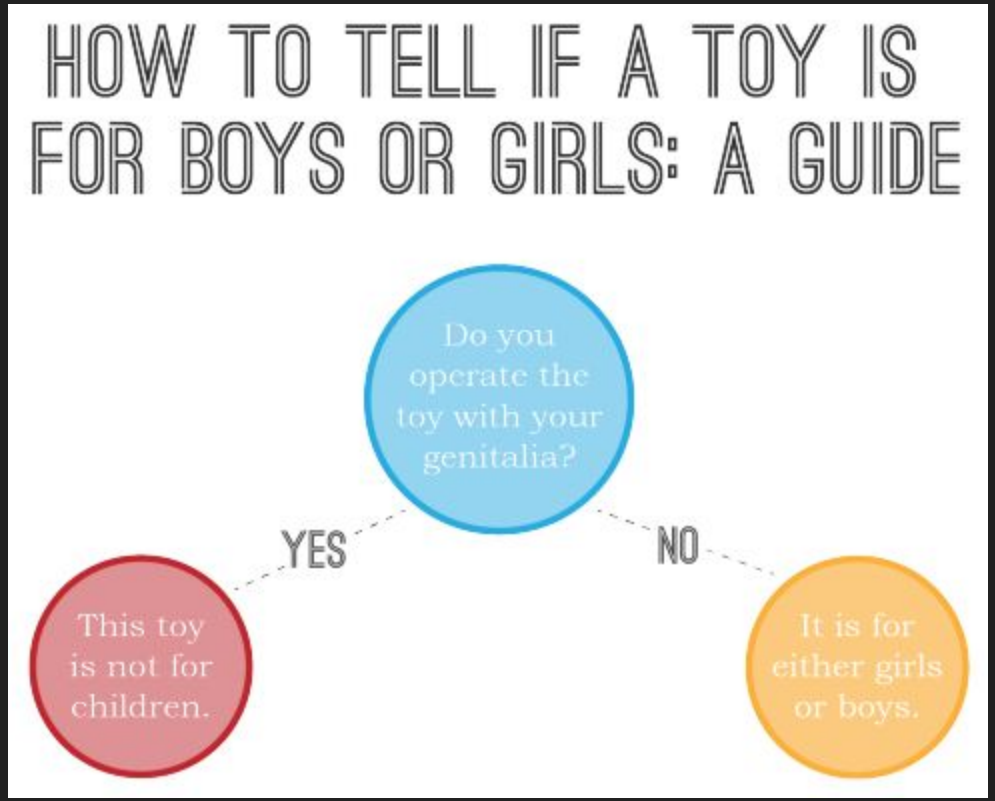
DeepToad

However, even when this tool works much better for me than ssdeep, I didn’t fix the problems inherent to fuzzy hashing:

- The same binary packed with 2 different packers, will not be matched.
- It will not match polymorphic malware.
- It will wrongly match file infectors (viruses) with the goodware it infected.

And this is how I decided to write yet another tool for “fixing” these problems...

Working with graphs



Working with Graphs

A program can be represented as a set of graphs:

- The relationships between all the functions in a binary can be represented as a Call Graph (CG).
- The functions and the relationships between their basic blocks can be represented as Control Flow Graphs (CFG).

Based on this basic concept, various tools and algorithms have been written. Some are mine and others are from other people.

Let’s see some of the tools and techniques...

Working with Control Flow Graphs

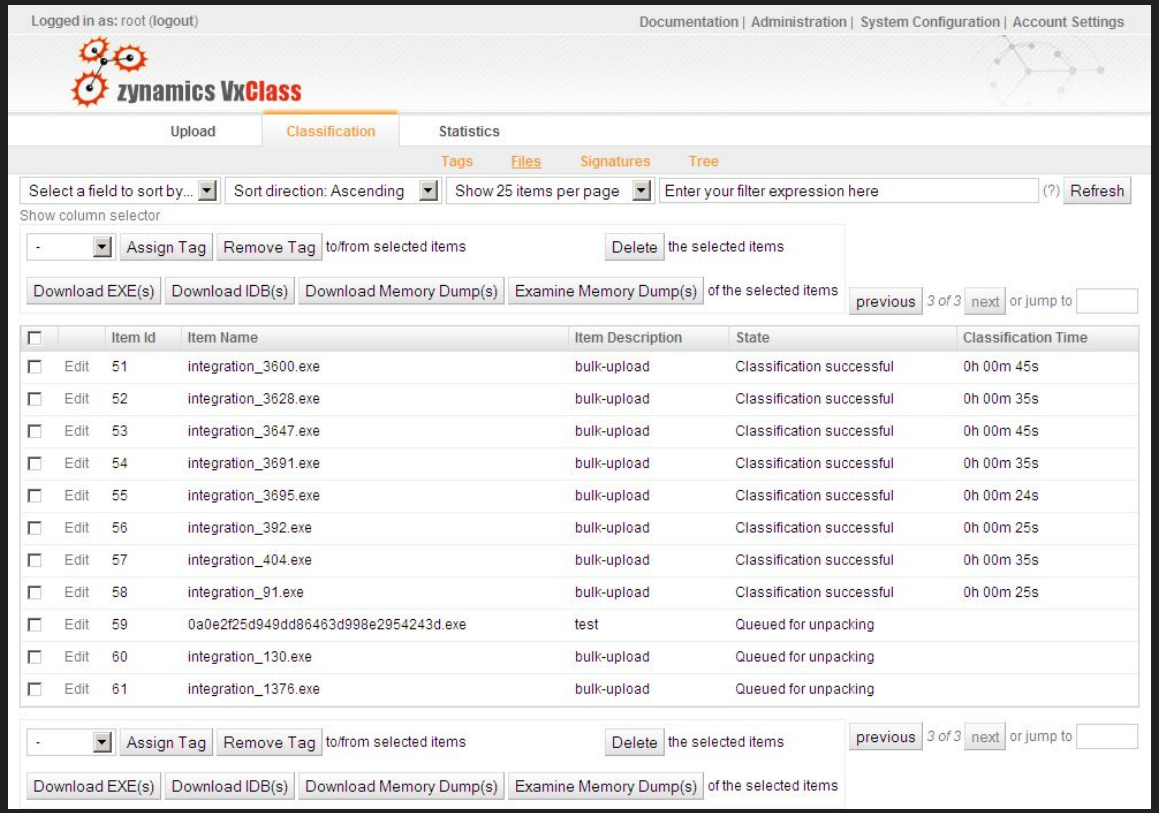
The very first tool I know that applied graph theory for malware research and built a commercial product was Zynamics and the tool was VxClass.

- https://www.zynamics.com/vxclass.html

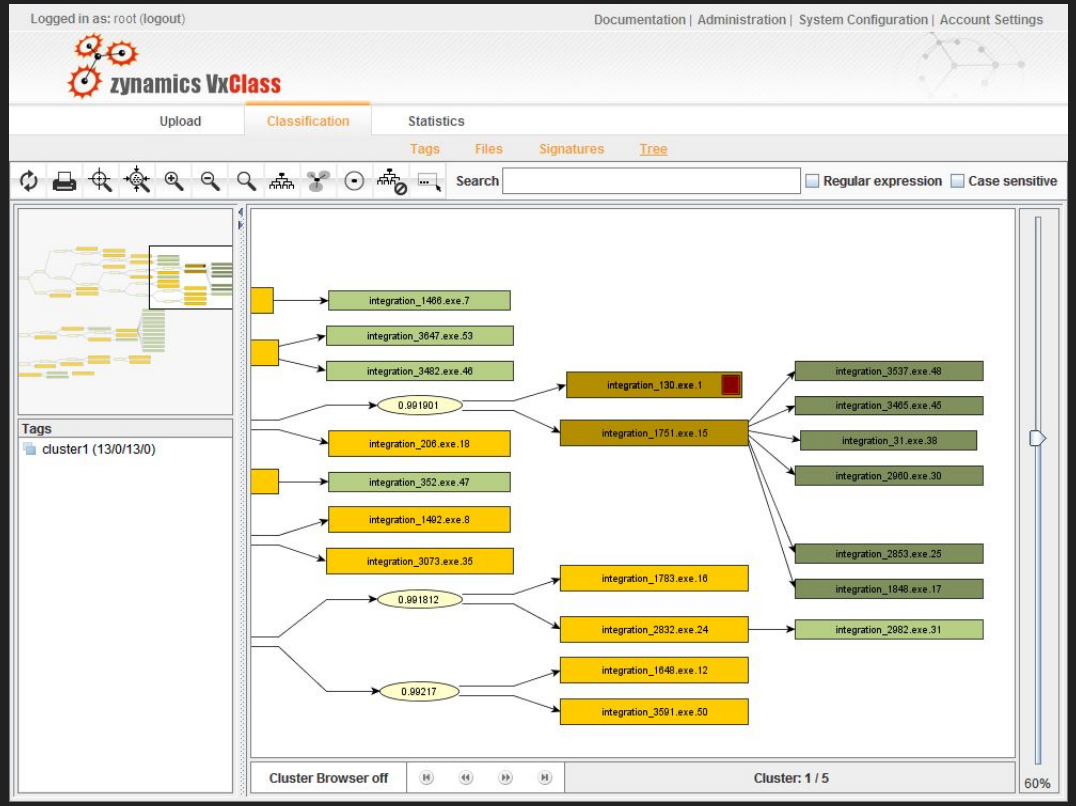
Zynamics was bought by Google and public development of VxClass was halt. No more versions of this commercial product was ever available.

Let’s see how this product used to work a bit...

VxClass


Logged in as: root (logout)
Documentation | Administration | System Configuration | Account Settings
zynamics VxClass
Upload
Classification
Statistics
Tags
Files
Signatures
Tree
Select a field to sort by...
Sort direction: Ascending
Show 25 items per page
Enter your filter expression here
(?)
Refresh
Show column selector
Assign Tag
Remove Tag
to/from selected items
Delete
the selected items
Download EXE(s)
Download IDB(s)
Download Memory Dump(s)
Examine Memory Dump(s)
of the selected items
previous
3 of 3
next
or jump to
Item Id
Item Name
Item Description
State
Classification Time
Edit 51 integration_3600.exe bulk-upload Classification successful 0h 00m 45s
Edit 52 integration_3628.exe bulk-upload Classification successful 0h 00m 35s
Edit 53 integration_3647.exe bulk-upload Classification successful 0h 00m 45s
Edit 54 integration_3691.exe bulk-upload Classification successful 0h 00m 35s
Edit 55 integration_3695.exe bulk-upload Classification successful 0h 00m 24s
Edit 56 integration_392.exe bulk-upload Classification successful 0h 00m 25s
Edit 57 integration_404.exe bulk-upload Classification successful 0h 00m 35s
Edit 58 integration_91.exe bulk-upload Classification successful 0h 00m 25s
Edit 59 0a0e2f25d949dd86463d998e2954243d.exe test Queued for unpacking
Edit 60 integration_130.exe bulk-upload Queued for unpacking
Edit 61 integration_1376.exe bulk-upload Queued for unpacking

VxClass


Logged in as: root (logout)
Documentation | Administration | System Configuration | Account Settings
zynamics VxClass
Upload
Classification
Statistics
Tags
Files
Signatures
Tree
Search
Regular expression
Case sensitive
Tags
cluster1 (13/0/13/0)
integration_1466.exe.7
integration_3647.exe.53
integration_3482.exe.46
0.991901
integration_206.exe.18
integration_352.exe.47
integration_1492.exe.8
integration_3073.exe.35
0.991812
0.99217
integration_130.exe.1
integration_1751.exe.15
integration_1783.exe.16
integration_2832.exe.24
integration_1648.exe.12
integration_3591.exe.50
integration_3537.exe.48
integration_3465.exe.45
integration_31.exe.38
integration_2980.exe.30
integration_2853.exe.25
integration_1848.exe.17
integration_2982.exe.31
Cluster Browser off
Cluster: 1 / 5
60%

VxClass

Well, it was a good (and not cheap) commercial tool and today is publicly dead.

In any case, I wanted to have such a tool, and there was only one way of getting such a tool: writing one.

And this is how I wrote Cosa Nostra.

Cosa Nostra

It is an Open Source graph based malware clusterization toolkit for program executables that work with any format supported by either Pyew, Radare2 or IDA Pro.

- https://github.com/joxeankoret/cosa-nostra

I aimed to present it first in SyScan 360 in 2017 but was finally first presented in Hack & Beers Bilbao 2017.

Cosa Nostra

How it works?

- Executable files are loaded, analysed with a RE tool and both CG and CFGs are extracted.
- Then, a graph based fuzzy signature is created.
- Then, samples with the same Call Graph Signature are considered “equal”.
  - Note: programs written in different languages with the same call graph will be too.
- Those that aren’t equal are compared at CFG level.
- And, finally, phylogenetic trees based on the comparison of CFGs between different binaries are generated, to display the clusters.

Let’s see it a bit more in detail...

Call Graph Signature

Phormula: hash = $\forall$ f $\in$ Functions , $\prod$ prime [CC_f]

A fuzzy call graph signature (CGS) is calculated the following way:

- The Cyclomatic Complexity of all functions found in a program are calculated.
- The N-th prime number corresponding to the Cyclomatic Complexity is assigned to that function.
  - For example: CC 3 → 3rd prime, so 5.
- Then, the small-primes-product of all functions is calculated.
- The final big number is the fuzzy call graph signature.

And that's it.

Cosa Nostra

If 2 or more binaries have the same CGS we can conclude that (they look like) they are structurally the same binary.

However, even for those files for which the CGS is not the same it’s still useful. How? Or better said, why?

- Because we have been using prime numbers and we can factor both CGS and determine how related the samples are.

Let’s talk a bit more in detail about that….

Cosa Nostra

Let’s say that we have 2 binaries:

- `A: with CGS: 2*2 * 5*5*5 * 3*3 * 7*7*7*7 * 13 = 140458500`
- `B: with CGS: 2*2 * 5*5   * 3*3 * 7*7*7   * 13 =   4013100`

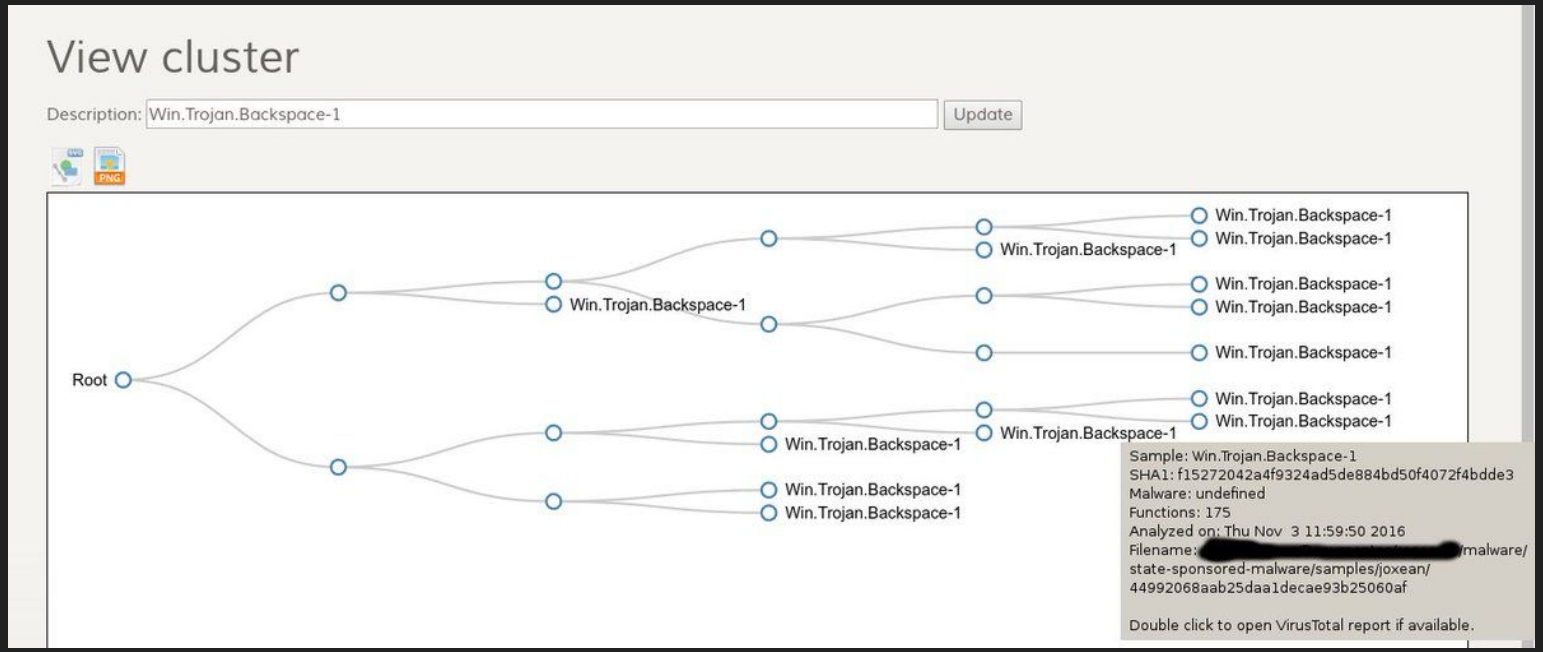
The CGSs are totally different. However, the prime numbers tell us that they aren’t in fact that-that different:

- Only 2 functions (with Cyclomatic Complexity 2 and 4) are different.

With Cosa Nostra, such 2 binaries would be put in the same cluster and they would be considered to be 83.33% similar.

Cosa Nostra

After comparing the samples, the next part is just running a boring Neighbor Joining algorithm + HTML + JS to output phylogenetic trees, that look like this:



Cosa Nostra

This approach works better than byte-level fuzzy hashing but has many problems:

- Cosa Nostra doesn’t have an unpacker, it will just cluster whatever is given: if packed files are given, it will cluster by packer.
- Even for unpacked samples, such a tool will put in the same cluster small and medium binaries with big runtimes (ie: GO executables).
- While very rare, 2 binaries that doesn’t have anything to do one with the other, might have the same call graph.
- A cartesian product is required to compare datasets, making it not suitable for big to huge datasets.
- ...

Cosa Nostra

And even more problems:

- 2 programs (2 DLLs, for example), with one or more big static libraries linked into it where only the code at main/WinMain/DllMain/… changes, will be put in the same cluster as most of the code is shared.
- There is an obvious but not viable at all option to remove such false positives:
  - Download most widely used Open Source libraries, build them with as most compilers as possible, and with all different compilation options (-O0, -O1, -OS etc…).
  - It doesn’t scale “well”.

And this is why I decided to write one more tool...

MAL tindex

MalTindex

Mal Tindex is an Open Source tool for indexing binaries and help attributing malware campaigns.

- https://github.com/joxeankoret/maltindex

In opposite to VxClass/Cosa Nostra, it doesn't use the CG for creating clusters and don't need to compare all files against all them, so no cartesian product here.

It's unaffected by how much code is shared between binaries, so no more affected by the problem of big runtimes and statically linked libraries, unless the functions are rare enough in the dataset.

But, how it works?

MalTindex

A malware indexing tool like MalTindex works the following way:

- Binaries are analyzed and for each CFG, one or more hashes are generated.
- Then, rare enough signatures/hashes are used as evidences.

For example:

- Let’s say that Group A uses some binaries and one of them has a rare function with signature XXYYZZ.
- Later on, we research some non attributed binaries and finds out that the rare function signature XXYYZZ is in the dataset, which might mean that some parts of the source are shared or even that they are the same group.

MalTindex

More technically speaking, MalTindex works this way:

- Binaries are analyzed with IDA and using a modified version of Diaphora, signatures are extracted for each CFG.
  - https://github.com/joxeankoret/diaphora
- Only 4 specific features extracted or calculated by Diaphora are used:
  - Bytes hash, Function hash, MD-Index and Pseudo-Code primes.
  - These signatures are the less false positive prone ones.
    - The KOKA hash (I will talk later on about it) could also be used, as it works even better than MD-Index often.
- Then, these signatures are inserted in a database and, finally, I just wrote a simple command line tool to find similarities.

MalTindex

Example usage:

```
MalIndex> match 9C7C7149387A1C79679A87DD1BA755BC AC21C8AD899727137C4B94458D7AA8D8

Searching for matches between 9C7C7149387A1C79679A87DD1BA755BC and AC21C8AD899727137C4B94458D7AA8D8...

Type | Name 1 | MD5 | Name 2 | Hash | Filename

M sub_402560 AC21C8AD899727137C4B94458D7AA8D8 sub_10004BA0 4.592895023151108863613470134
/malware/766d7d591b9ec1204518723a1e5940fd6ac777f606ed64e731fd91b0b4c3d9fc.idb

M sub_10004BA0 9C7C7149387A1C79679A87DD1BA755BC sub_402560 4.592895023151108863613470134
/malware/3e6de9e2baacf930949647c399818e7a2caea2626df6a468407854aaa515eed9.idb

Total of 2 row(s)
```

MalTindex

Example usage:

```
MalIndex> mdindex 4.592895023151108863613470134

md5 | name | file_name

AC21C8AD899727137C4B94458D7AA8D8 sub_10004BA0
/malware/766d7d591b9ec1204518723a1e5940fd6ac777f606ed64e731fd91b0b4c3d9fc.idb

9C7C7149387A1C79679A87DD1BA755BC sub_402560
/malware/3e6de9e2baacf930949647c399818e7a2caea2626df6a468407854aaa515eed9.idb
```

MalTindex

Do you happen to know the previous samples? No? Take a look to this tweet:

MalTindex

Pros:

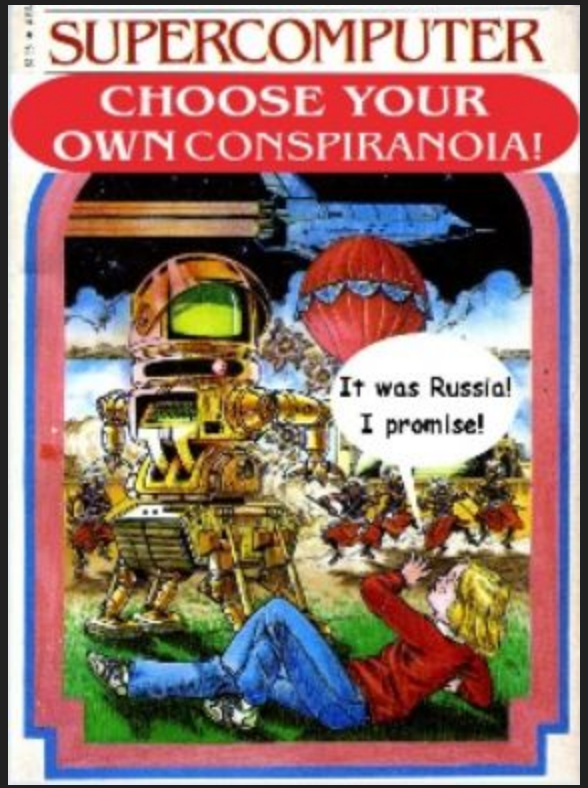
- Suitable for huge datasets.
- The more files are indexed, the better the indexes (the signatures) will be.
- Rare enough signatures in a big enough dataset are very good evidences.

Cons:

- It’s only suitable for huge datasets or a lot of false positives will happen.
- It’s perhaps not suitable for everyone and what is “big enough” is unclear.
    - Some people, like Halvar Flake, say that results can only be trusted after at least 1 million samples indexed.
- Also, is very sensitive to dataset bias (i.e., not feeding to it goodwares).

Problems...

The problems of not having a big enough dataset or a biased one (ie, only malware) and trusting the output of such a tool can be summarized in the picture at the right.



Other Algorithms & Open Source Tools for Indexing

KOKA hash: An algorithm suitable for large scale malware indexing.

- http://joxeankoret.com/blog/2018/11/04/new-cfg-based-heuristic-diaphora/

Functions SimSearch: A SimHash[1] tool used to find code similarities. Suitable for large scale malware indexing. By Thomas Dullien (Halvar Flake).

- https://github.com/googleprojectzero/functionsimsearch

[1] SimHash: technique for quickly estimating how similar two sets are.

Other Approaches

Other Approaches

The previously described tools, techniques and algorithms are in no way the only possible approaches to classify or cluster or distinguish malware families, actors, etc…

There are other algorithms, tools, approaches, commercial products, etc...

Let’s talk briefly about other possible options...

Behavioural Based Clusterization

I know no real public implementation, only academic papers (with no accompanying source or binaries, of course, as always happens with the academia):

- Malware Behaviour Clustering, by Engin Kirda.
  - http://sci-hub.se/https://doi.org/10.1007/978-1-4419-5906-5_848
- Analysis of Malware Behaviour: Using data Mining Clustering Techniques to Support Forensics Investigation, by Edem Inang et al.
  - http://sci-hub.se/10.1109/CTC.2014.10
- Scalable, Behavior-Based Malware Clustering, by Ulrich Bayer et al.
  - https://sites.cs.ucsb.edu/~chris/research/doc/ndss09_cluster.pdf
- ...

Behavioural Based Clusterization

Some more academic papers:

- Learning and Classification of Malware Behavior, by Konrad Rieck et al.
  - https://core.ac.uk/download/pdf/46908978.pdf
- Clustering analysis of malware behavior using Self Organizing Map, by Radu-Stefan Pirscoveanu et al.
  - http://sci-hub.se/https://ieeexplore.ieee.org/document/7503289

And many others. But they are just random ideas never implemented in most of the cases.

Behavioural Based Clusterization

Pros:

- Agnostic of the file format.
- Doesn’t care about packed or protected file formats.
- Any platform for which behaviour can be obtained can be clustered.

Cons:

- Extremely slow, average of 5 minutes required per sample.
- Easy to bypass: just wait for 1 hour before doing anything, for example.
- Easy to detect: VMs, hypervisors, monitors, debuggers, etc… are easy to detect.

Network Based Clusterization

Yet another approach to cluster malware. Instead of using its bytes, the program structure or the OS level program behaviour, the network packets, flow, C&Cs, etc… are used.

Again, I know no single public implementation of such a system, only academic papers with, as always, no accompanying binaries or source codes whatsoever because… academia, they only care about publishing papers, with only a few exceptions.

In any case, the following are the most interesting academic papers I have found based on this idea...

Network Based Clusterization

Academic papers:

- MalPaCA: Malware Packet Sequence Clustering and Analysis, by Azqa Nadeem et al.
  - https://arxiv.org/pdf/1904.01371.pdf
- FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors, by Zubair Rafique et al.
  - https://software.imdea.org/~juanca/papers/firma_raid13.pdf

Sadly, as I said, there is nothing beside a paper published. At least for now.

Network Behaviour Clusterization

Pros:

- As with OS level behavioural clusterization, agnostic of file format, packers and protectors.
- Good for clustering specific families with specific netflow patterns.
- Can be implemented by just recording PCAPs without requiring to install anything on the boxes running malware samples.

Cons:

- Can be even more slow than OS level behavioural clusterization.
- When used with VMs, hypervisors, etc… same drawbacks as with any kind of behavioural clusterization.

Conclusions

Conclusions

There are many options to clusterize and index malware samples and families, as well as to attribute malware samples, behaviour, network packets and netflows to specific actors, but there is no “best option” suitable for everyone.

- Some approaches aren’t suitable for large scale malware (Call Graph and Control Flow Graph clusterization).
- Some approaches aren’t suitable for anyone but huge companies (malware indexing).

Also, actors can target your specific tool, bypass it or fool it into believing that some malware was done by some other specific actor.

- Actually, I want to write such a tool since long ago. One day ;)

Conclusions

What method is suitable for you will largely depend on the following questions:

- How many malware samples do you need to process daily?
- What are you interested on?

Examples:

- Some approaches, like indexing and network based systems, are good for triggering alerts.
- CG and CFG clusterization systems are good for noise removal.

Also, with the only exception of AVs, I don’t think anyone really needs to employ all the methods I have discussed in this talk.

Eskerrik asko!

And, that’s it! Any questions?