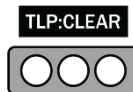
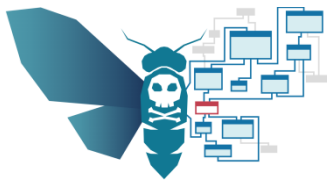


10 Years of Large-Scale Malware Comparison: Going Deeper With Machoke



Stefan Le Berre / Tristan Pourcelot





ExaTrack

Who are we?

ExaTrack

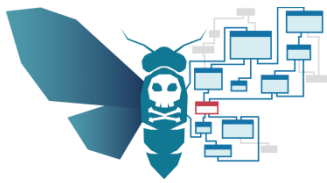
Small company focused on finding unknown attackers

- Incident Response
- Threat Hunting
- Threat Intelligence

Stefan Le Berre: CEO/Cofounder

Tristan Pourcelot: Senior Threat Hunter

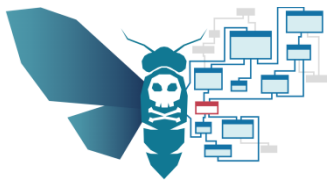




Overview

1. What is Machoc?
2. Evolving into Machoke
3. A wild Zubat appears
4. Conclusion



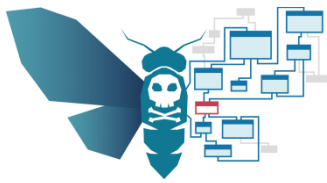


Malware challenges

Some challenges of “day to day” malware analysis:

- “Here is a USB drive with 100 samples, reverse them for yesterday”
- Do I already know this sample?
- Does it belong to a known family?
- What’s new in this sample vs an older one?
- Does this sample share functions with another one?





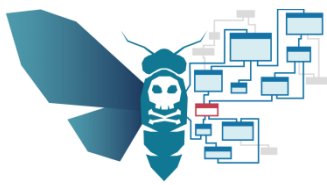
“Karate Chop”

We developed the [Machoc](#) hash at ANSSI/CERT-FR

We [published](#) it 10 years ago at SSTIC

- Able to compare hundreds of malwares in a few seconds.
- But we want to scale to a larger dataset (million of samples)
- And we wanted to go deeper





Machoc

Machoc in a couple of seconds:

The objectives were the following:

- Simple algorithm
- Fast to calculate
- Resistant to small changes (such as a malware C2 update)

For each function:

- Extract the summary of a function:
 - Control Flow Graph (CFG)
 - Inner Calls
- Make a textual representation of these information
- Hash it



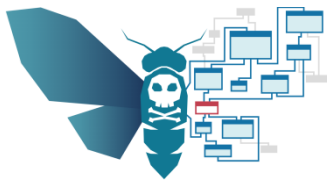


1:2,3;
2::
3:4,10;
4:6;
5:6;
6:c,7;
7:c,8
8:5,9;
9:10;
10::

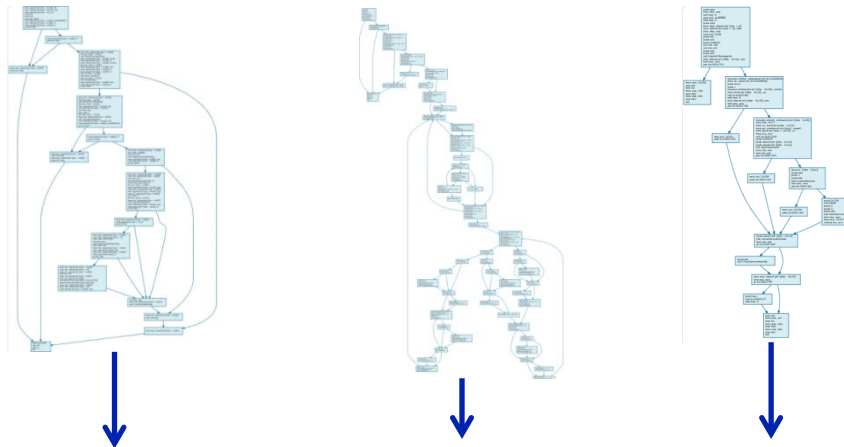


Address of the
function





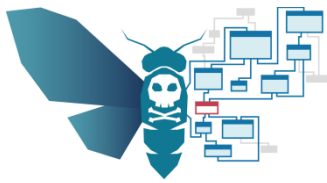
Repeat this operation for each function and concatenate all results to a big string.



40168:c814dbb3;402c0:94b076dc;405d0:0fec5a12;...

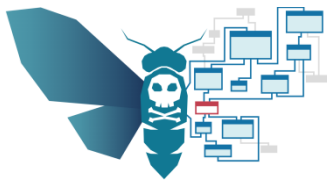
Matching is done by calculating a Jaccard distance between two sets of hashes





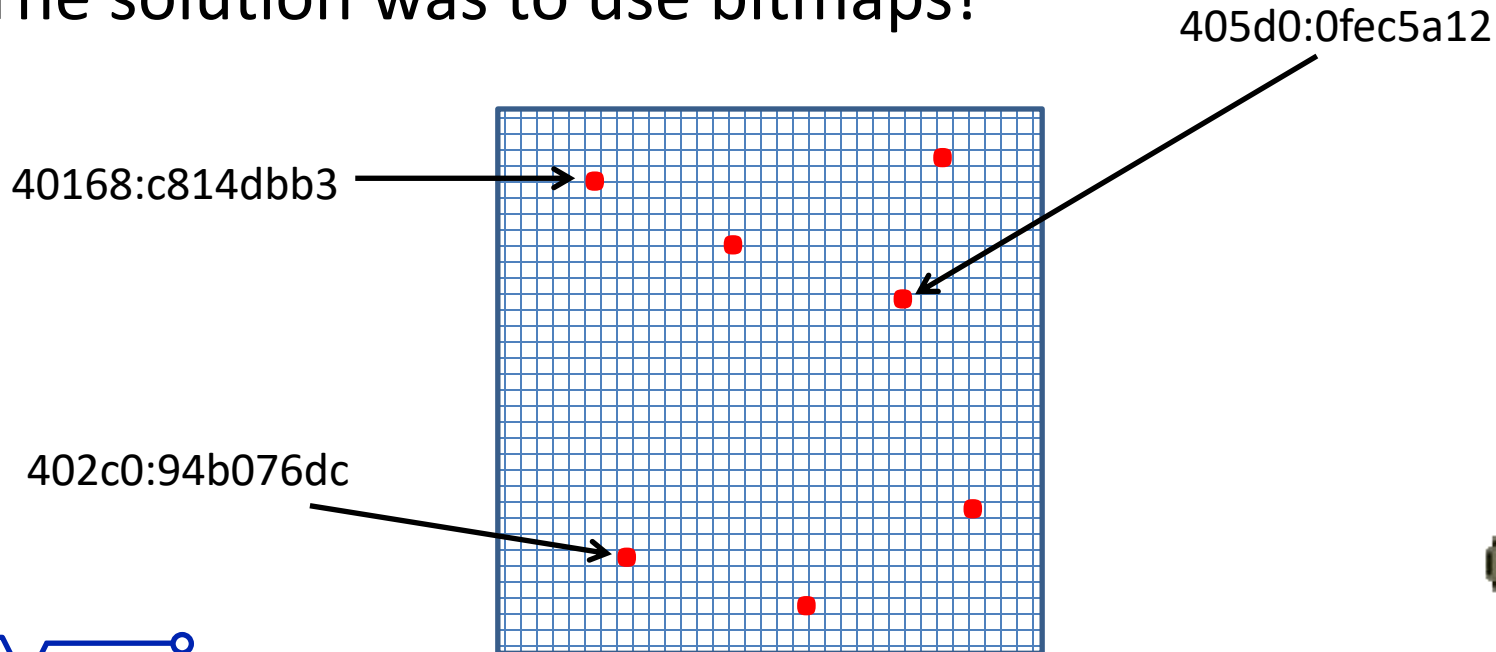
- Successes:
 - Malware clustering
 - Identification and attribution of unknown malware samples
 - Label propagation for common functions between samples
 - Identification of shared libs between firmware versions
 - Identification of changed functions between releases of firmware
 - We were surprised by the numbers of people using it
- Limits:
 - Runtime similarity (Hi OpenSSL!)
 - “Small” functions
 - Exponential complexity of diffing, doesn’t scale to 100k samples

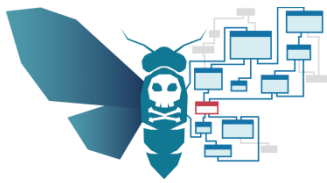




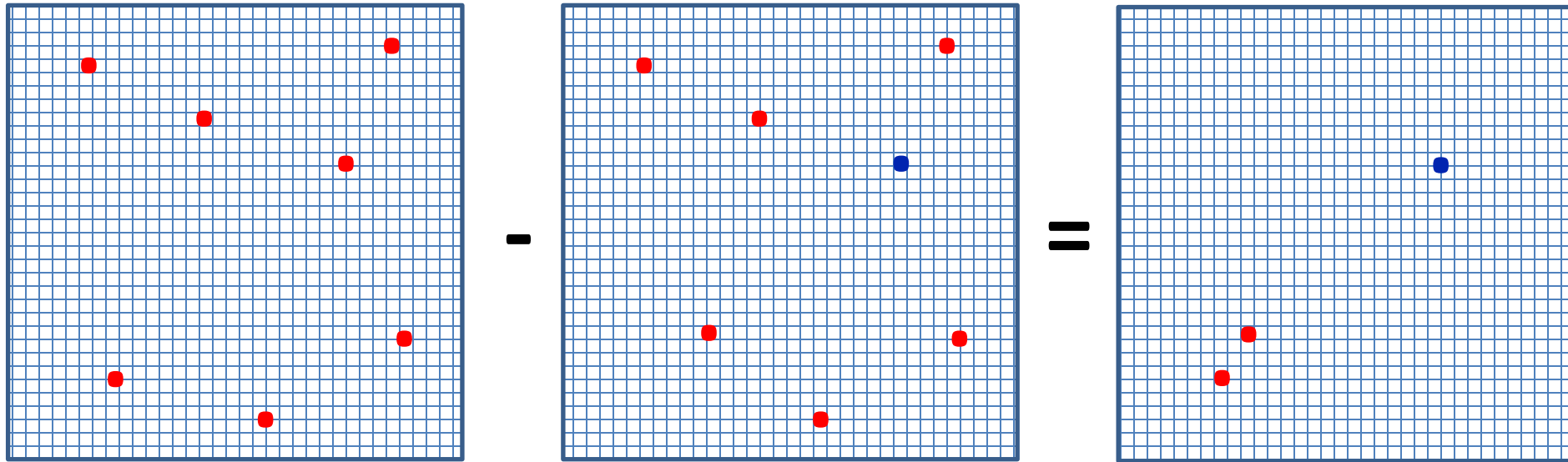
We must change our approach to scale on a large dataset, and transform this exponential calculation to linear.

The solution was to use bitmaps!



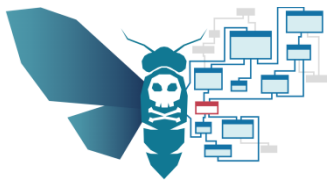


Comparing bitmaps is done by subtraction:



Blue point is a difference on density in this point





Machoke

Cool! We now have the capacity to compare thousands of bitmaps in a linear time :)

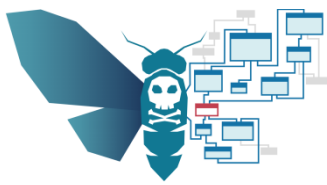
But!

It's too slow again :'(

Because we have to compare all bitmaps each time.

So we added two levels of clustering





Machoke – Func Counting

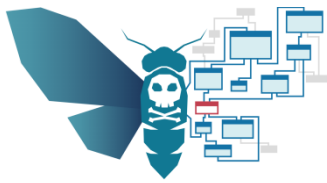
An executable with 100 functions will never have a 80% match with a 50 functions binary.

- Add a filter layer with a count of disassembled functions.

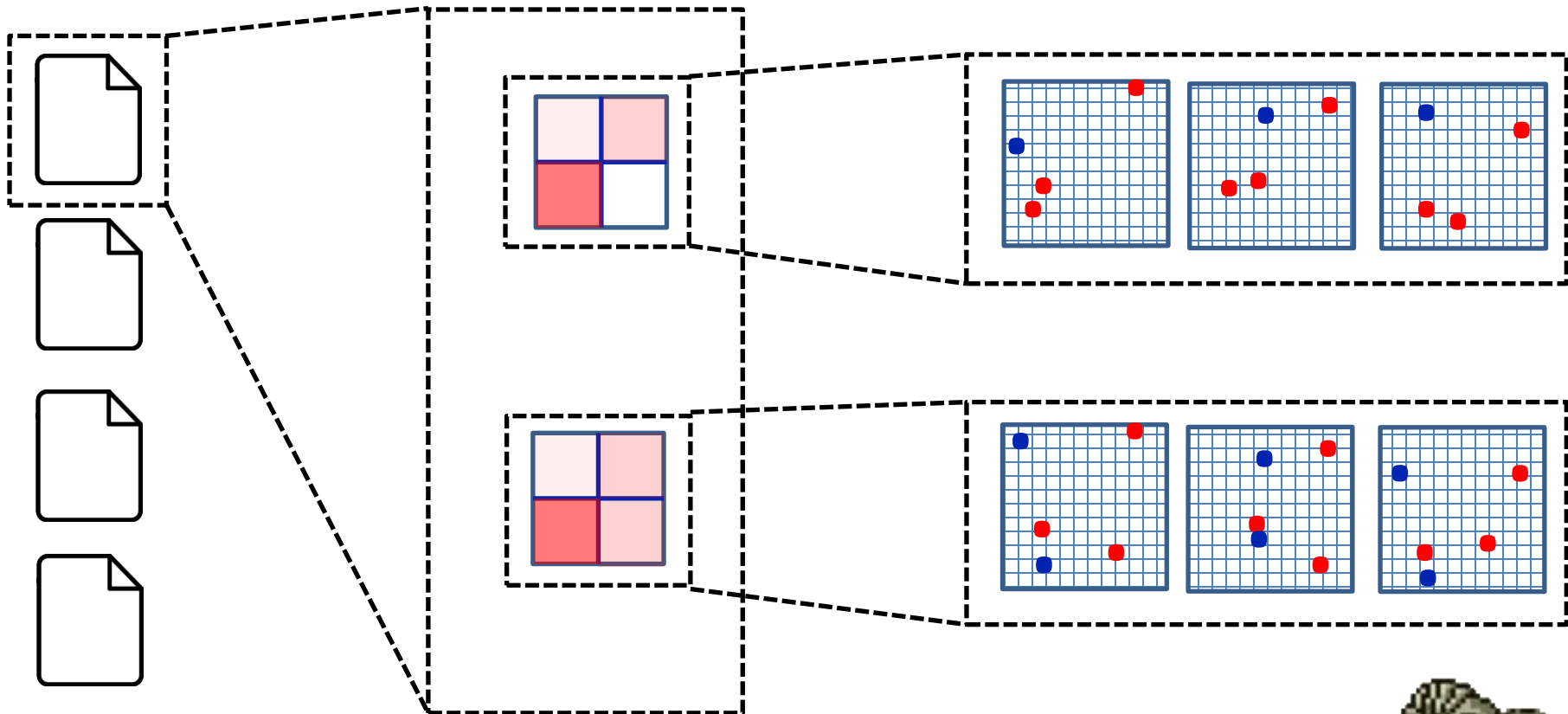
We added a second clusterization level using a reduction of bitmaps density

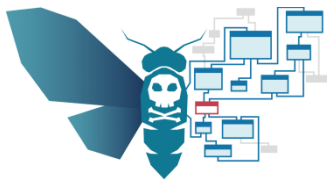
- We had to generate small heatmaps representing the global density of the sample's bitmaps.



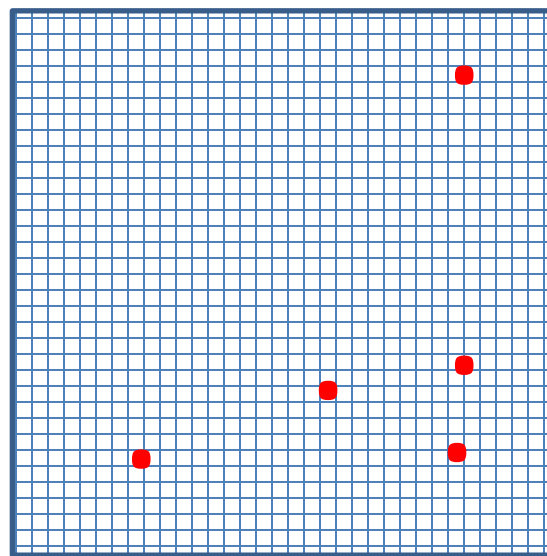
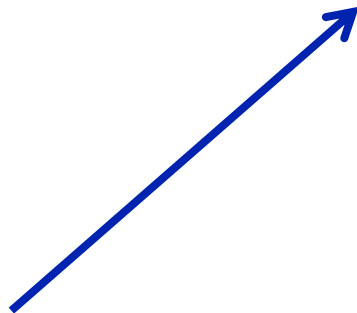
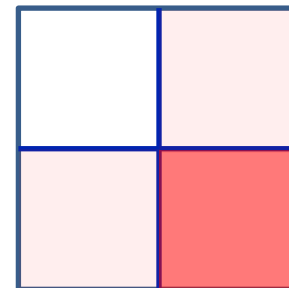
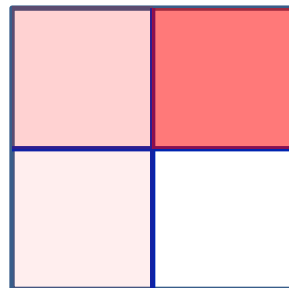
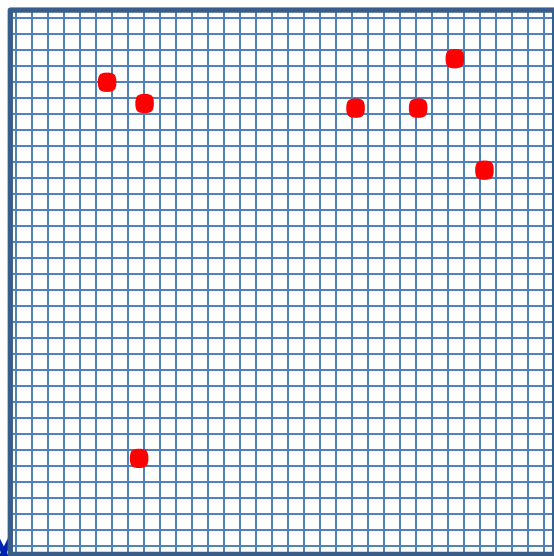
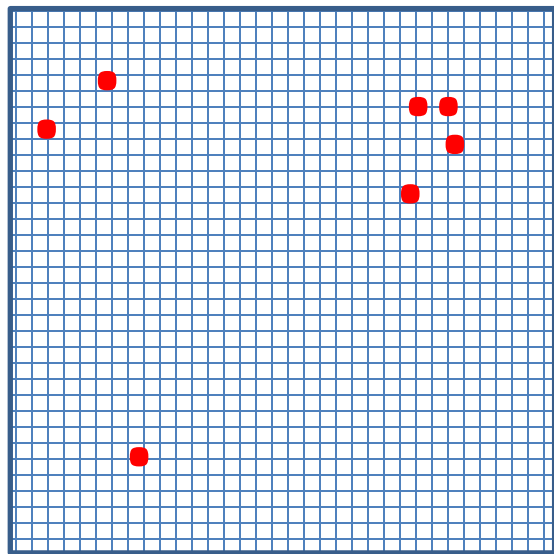


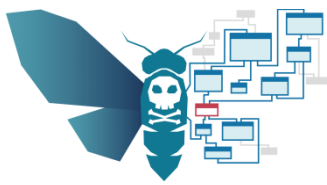
Machoke – 3 steps





Machoke - HeatMap



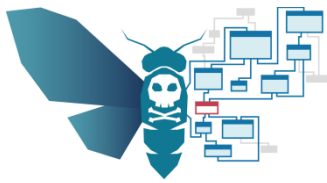


Machoke

The three steps of Machoke filters:

- We start by listing binaries with a similar functions count.
- We compare the heatmaps to quickly filter possible matches.
- Then we compare the full bitmaps of each sample and find exactly which ones are matching.

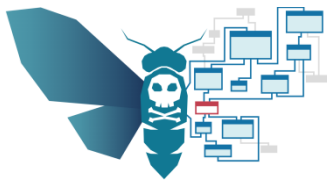




We still got some problems:

- Samples share static libraries (Hi OpenSSL again!)
- Compilers include runtimes
- More and more “all-in-one” binaries (Hi Go!)

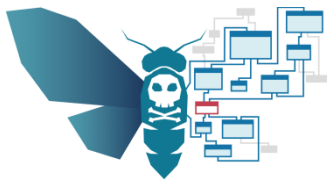




How to solve the runtime problems?

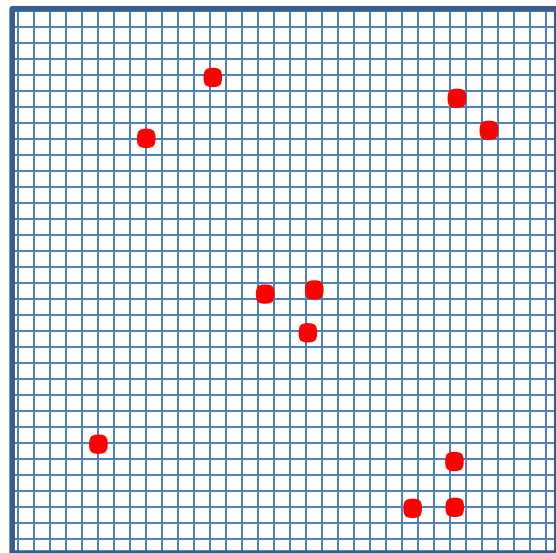
- We compute the bitmaps of benign samples, libraries, etc.
- Each malware is compared with them and matching points are extracted to a new “runtime_bitmap”.
- For each of these bitmaps we subtract the nearest “runtime_bitmap” to every sample in the list
- We now have only the “malware” code to match



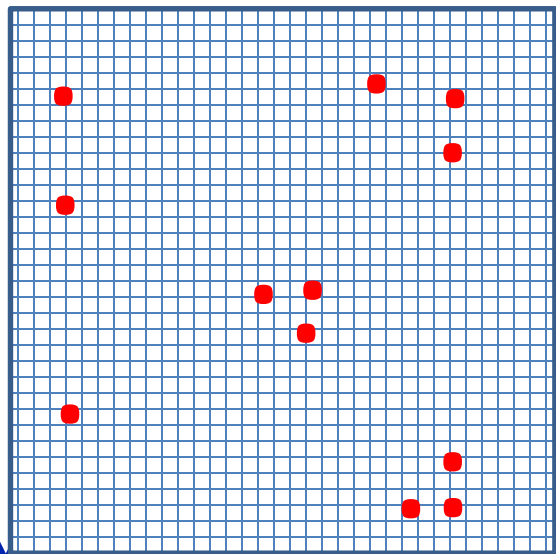


ExaTrack

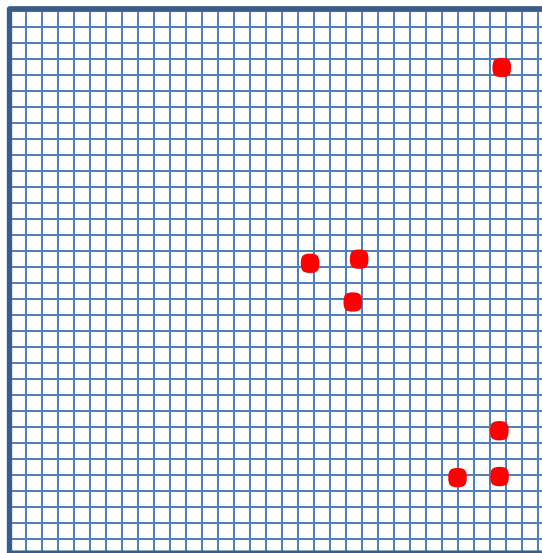
Machoke



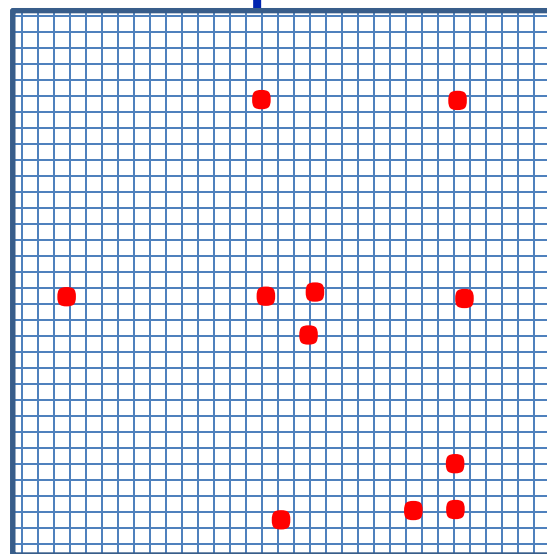
Bin1.go



Bin2.go

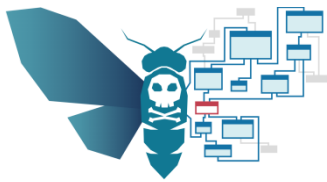


Probable Go
runtime_bitmap



Bin{N}.go






Machoke - Demo



We used these techniques in the [Exalyze.io](https://exalyze.io) platform for comparing samples.

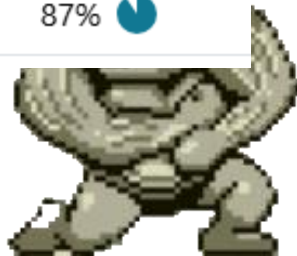
For exemple, here is the match of two SysJoker samples (8 months of dev separate them)

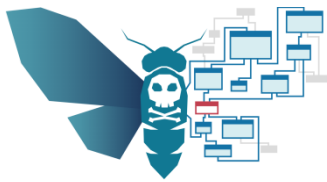


Sample
SHA256: 1
[Reanalyze](#)

Samples matching 1ffd6559d21470c40dcf9236da51e5823d7ad58c93502279871c3fe7718c901c

Sample	Function similiarity
6c8471e8c37e0a3d608184147f89d81d62f9442541a04d15d9ead0b3e0862d95	87% 
e076e9893adb0c6d0c70cd7019a266d5fd02b429c01cfe51329b2318e9239836	87% 





Machoke - Demo

Demo: Finding Mélofée variants

Mélofée (TripleZero) is a malware family used by CN actors

<https://blog.exatrack.com/melofee/> :

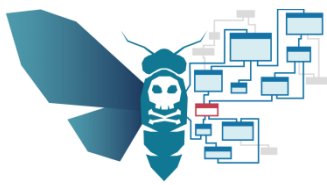
36161b6afed04c084470a703b3d8f50d

https://blog.xlab.qianxin.com/analysis_of_new_melofee_variant_en/ : 603e38a59efcf6790f2b4593edb9faf5

Undetected variant:

22c49ea17617361b5323922c2252d42e



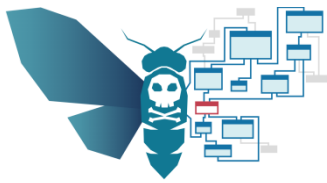


Cool! We can now compare tons of samples!

But we would like to compare each function with all other functions stored inside a dataset, not just the “sum” of functions for each binary.

This lead to the development of Zubat, an algorithm we created to go deeper in malware tracking.



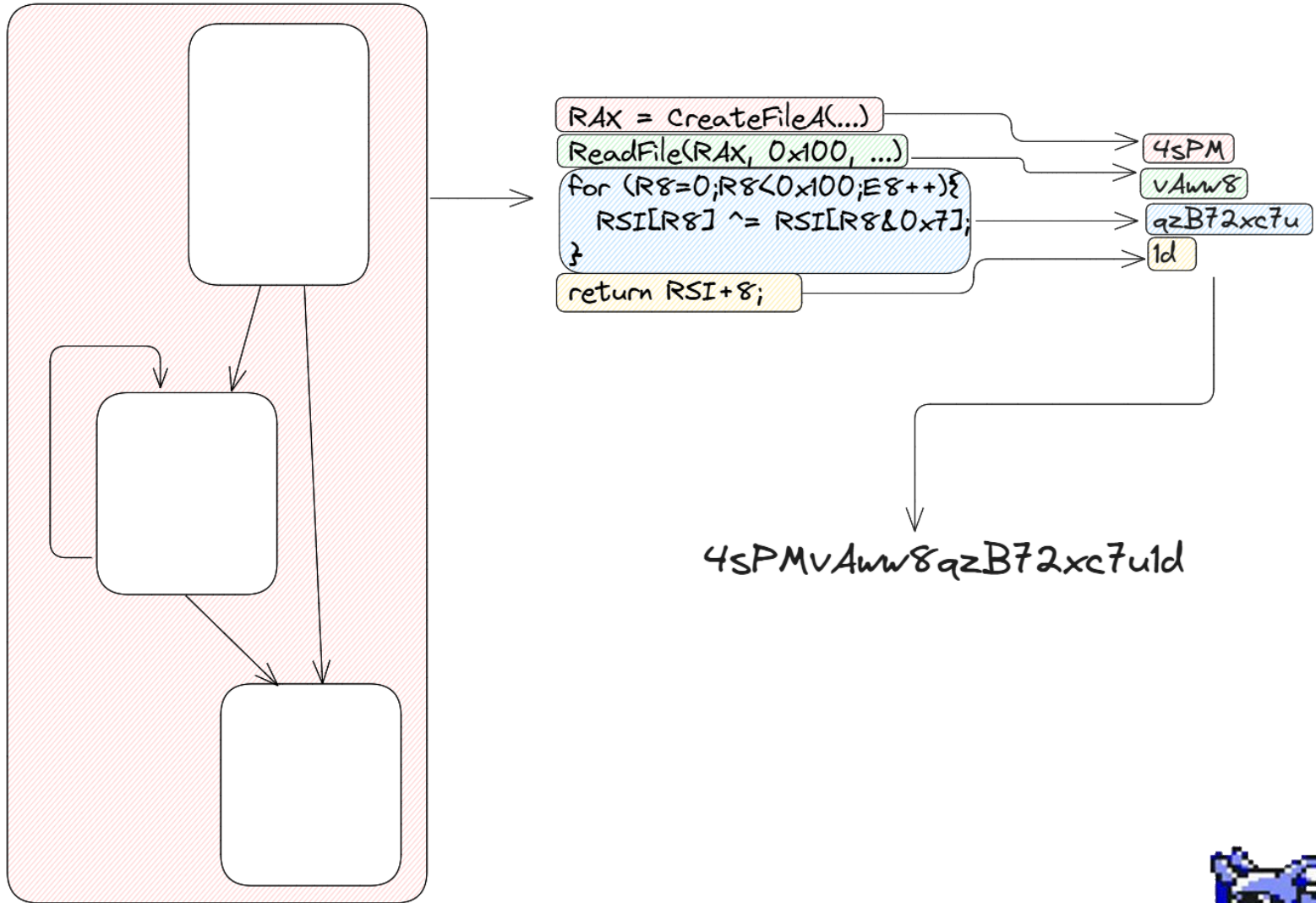
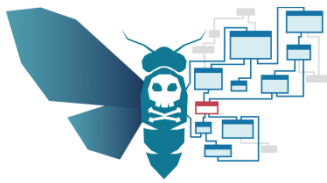


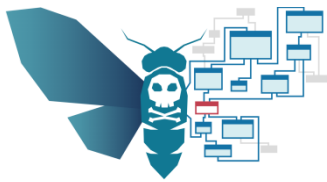
After a function is disassembled and its CFG rebuilt, we extract the main logic of it and built an intermediate representation of this logic.

Representation must isolate “features” independent of the global function logic.

Then we transform this profile into a fuzzy-hash string.

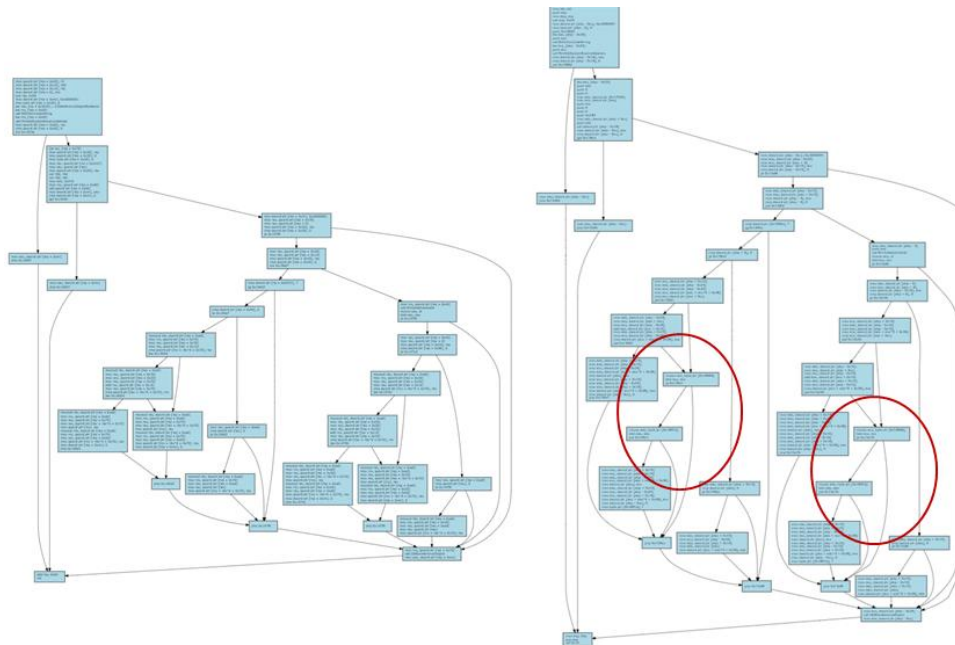


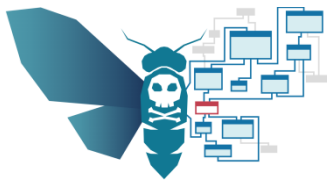




During our testing (on millions of functions) we could even get matches between modified functions.

Here is a 74% match between 2 functions of 2 ZxShell rootkits, the first is x86 and the second one is x64 ;)

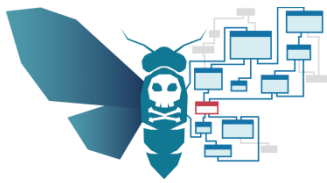




Conclusion

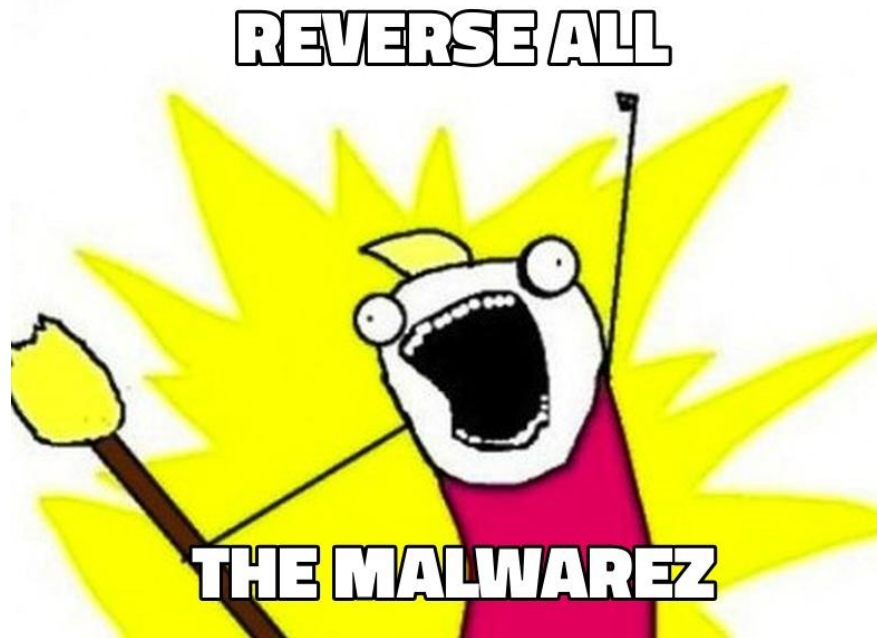
- Code Similarity Analysis is complex but useful
- Using Machoke, we found some relations between attacker's codes and variants
- Code matching help a lot during reverse engineering
- To diff large datasets of malwares you also need to integrate a lot of benign codes





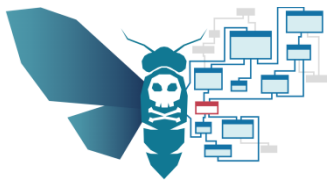
Thank you

Thank you for your attention, any questions ?



Machoke similarity analysis (and more) is available on
<https://exalyze.io>

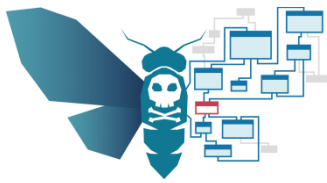




Similar algorithms

- MCRIT (<https://github.com/danielplohmann/mcrit>)
- Ghidra Bsim
(<https://github.com/NationalSecurityAgency/ghidra/blob/master/GhidraDocs/GhidraClass/BSim/README.md>)
- <https://github.com/googleprojectzero/functionsimsearch>
- TLSH (<https://documents.trendmicro.com/assets/wp/wp-locality-sensitive-hash.pdf>)
- Kesakode (<https://doc.malcat.fr/analysis/kesakode.html>)





Who used Machoc?

- [GData](#)
- [CarbonBlack](#)
- [Rapid7](#)
- ANSSI
- Exatrack ;-)
- Probably more!

