A Graph Based Malware Clustering Toolkit

Joxean Koret

SyScan 2k16, Shanghai

Introduction

Introduction

How it works?

Tool

Examples

Known Problems

Future

Questions

Introduction

Cosa Nostra is a free and open source graph based malware clustering engine/toolkit that:

- Clusters executable files of any supported type.
- Generates phylogenetic trees based on the graphs.

The name “Cosa Nostra” is kind of a joke.

- I was finding a name for “a thing that makes families” or anything “families related.”
- As my main dataset was state sponsored malware, the word “mafia” come to mind.
- The name “Cosa Nostra” seemed appropriate.
- I hope I don't have problems with the copyright owners...

Introduction

- Cosa Nostra uses a 3rd party code analyser in order to extract information from executable programs.
- Currently, it supports Pyew, Radare2 and IDA.
  - The final user can choose whatever analysis engine he/she prefers.
  - Support for Hopper or BinaryNinja is not even planned as I don't have a license for such products.

How it works?

How it works?

First, a 3rd party code analyser (Pyew, Radare2 or IDA) is used to:

- Load executable files.
- Analyse executable files.
- Extract call graph and flow graphs data.
- Generate a fuzzy call graph signature.
- Optionally, get the ClamAV malware name.

The call graph signature will be used later on to detect structurally equal programs as well as structurally similar looking programs.

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 is 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.
- That's it. That easy. Basically.

Call graph signatures

If 2 executable programs share the exact same CGS, we can conclude they are structurally equal.

If the signatures are different, we still can quickly determine how similar they are by factoring prime numbers and determining the different ones for each sample.

- The generated number is not that-that big after all.

It's a fast and easy fuzzy graph matching algorithm for clustering samples that is resilient to functions re-ordering.

- In any case, it doesn't happen too often.

Clustering

All right, we have a set of fuzzy signatures for a set of program executables.

How can we use that information to create phylogenetic trees in order to discover malware families and their relationships?

Assumption: new variants, usually, will contain more code or more complex functions than previous ones.

As so, if 2 programs share a very similar CGS but one of them has more functions or more complex ones, that is a later generation.

Clustering

Based on this assumption and using the previously generated signatures, phylogenetic trees are generated using a “Neighbor Joining” algorithm.

It creates a tree that, perhaps, can be useful in order to determine malware samples relationships, which version is based on a specific branch, if 2 or more malware samples descend from the same code base, etc...

The Tool

The Tool

The current version of the clustering tool is divided in 3 parts:

- A batch tool to analyse and extract signatures.
- A clustering daemon that looks for new samples every some time and cluster new samples.
- A web based GUI.

The Batch Tool

The batch tool is used, as previously explained, to extract the call graph signature from executable files.

The batch tool uses Pyew, Radare2 or IDA for analysing PE, ELF, Boots, BIOS, ... files and extract the CGS.

The batch tool, also, uses pyclamd (if available) to give a descriptive name to the malware samples being analysed.

- Not available in the IDA's batch tool.

The Clustering Daemon

This is the most complex part of the toolkit.

This is the process responsible of actually creating the phylogenetic trees using a Neighbor Joining algorithm.

The core of that engine was written at some point during 2013 and only optimizations have been made to the code in order to make it faster.

- And also not to eat the whole machine's resources.

In the future, I would love to make that daemon parallel. But is not trivial.

The GUI

The web based GUI is based on:

- web.py: All the logic, templates, etc...
- D3.js: Used for displaying the cute graphs we all love to see.

So far, it's a single-user tool.

In the not so far future, I plan to make it multi-user and, also, put an API on top.

- So instead of running batch tools on the same machine where the database is, you can upload samples via the Web API.

Examples

(Demo)

Known Problems

Known Problems

The known problems of Cosa Nostra, so far, are the following:

- Viewing/Manipulating large clusters is not easy.
- Packed samples will be clustered into a single cluster for each packer.
- Statically compiled binaries can be wrongly assigned to a cluster because the runtime is not discarded.
  - This can be less problematic with IDA due to FLIRT but still large libraries statically compiled inside a program are problematic.
- Clustering samples takes a lot of time.

Performance

The whole clustering process takes very long even for just 1K samples:

- The analysis time (Pyew/R2/IDA) takes most of the time.
- The other costly part of the project is finding the neighbours of each sample.
  - Basically, a Cartesian product of the total number of samples.
  - Go try with, say, 10K samples.
    - It finishes, but not “soon.”

Future

The not so far future

In the next weeks or months I plan to add the following features:

- Web API on top to upload and analyse samples, find samples in clusters, find similar clusters, etc...
- A more usable graph displaying engine.

The really far away future

For the next versions, if this Open Source project has any luck, I plan to add the following features:

- An optional “generic” unpacker based on Intel PIN.
- Automatic signatures generation:
  - Yara and/or ClamAV.
- A new C/C++ code analyser for x86/x86_64 that could be used in a desktop.
  - For matching call graph signatures.
  - Probably, Radare2 embedded using ZeroVM, just in case.

And that's all!

Questions?