Code Cartographer’s Diary

@push_pnx

2018-12-05 | Botconf, Toulouse

Daniel Plohmann
daniel.plohmann@fkie.fraunhofer.de

Steffen Enders
steffen.enders@tu-dortmund.de

Paul Hordiienko
pavlo.hordiienko@fkie.fraunhofer.de

Elmar Padilla
elmar.padilla@fkie.fraunhofer.de

Fraunhofer FKIE

The

Agenda

Agenda

- Malpedia
  - Project Overview
  - Progress
- Windows API Usage Recovery & Analysis for Malware Characterization
  - Tools: ApiScout / ApiVectors
  - Evaluation Results
- Code-based Similarity Analysis
  - Tools: SMDA & MCRIT
  - Current State / Results
- Summary

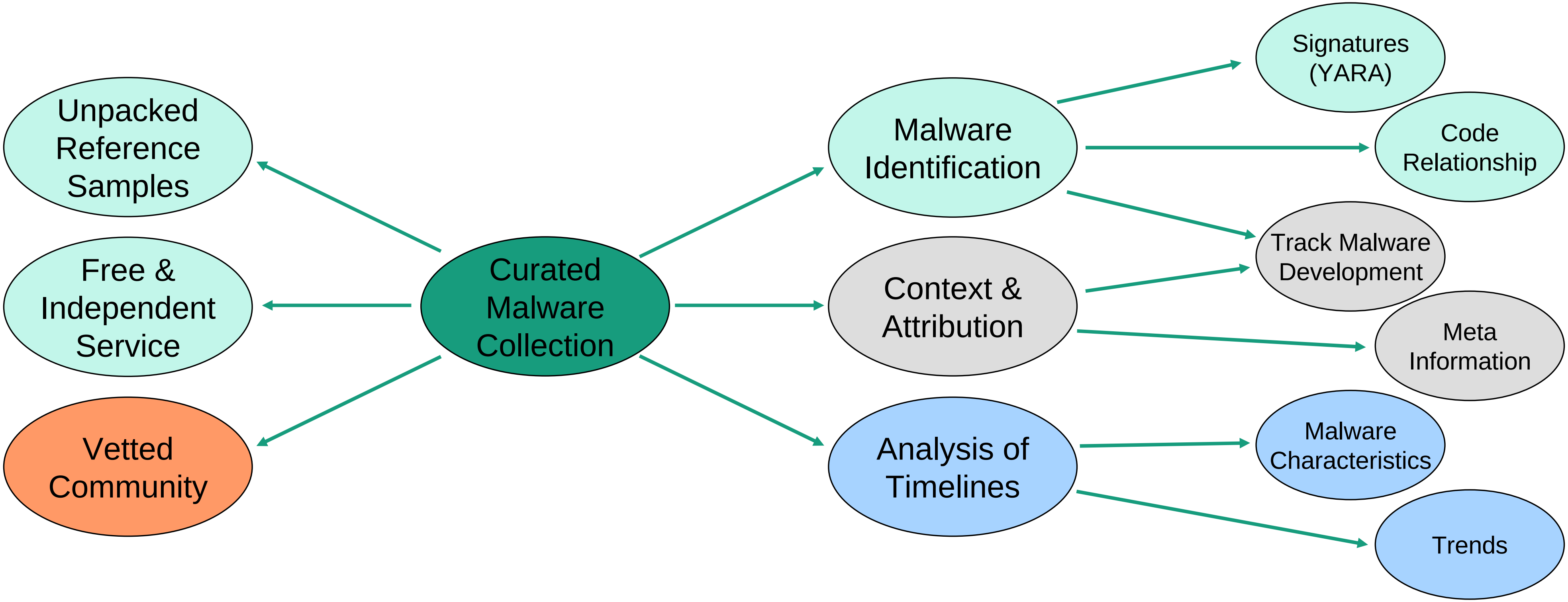
Fraunhofer
FKIE

malpedia

Fraunhofer
FKIE

Overview

Motivation



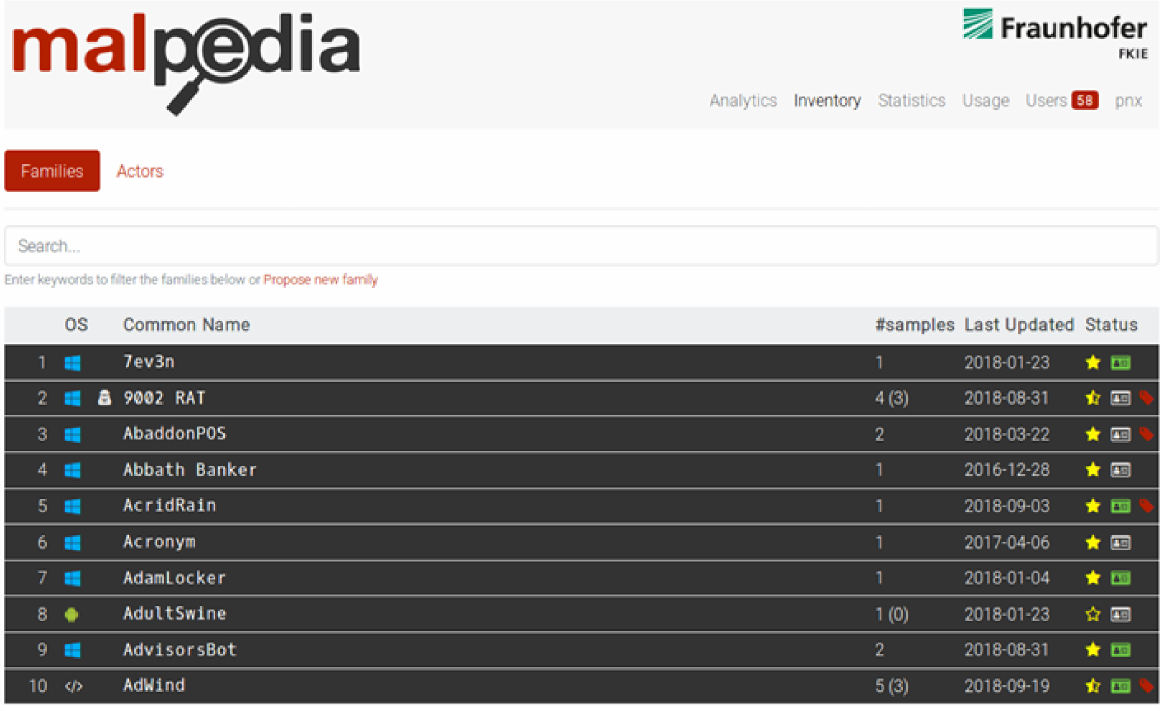
[1] https://malpedia.caad.fkie.fraunhofer.de
[2] https://malpedia.io

Fraunhofer FKIE

Overview
Context

- Launched @ Botconf 12/2017 [3]

- Full paper outlines project goals:



+ REST API & git repo

[1] https://malpedia.caad.fkie.fraunhofer.de
[2] https://malpedia.io
[3] https://journal.cecyf.fr/ojs/index.php/cybin/article/view/17

Fraunhofer FKIE

Overview

Progress

	31 October 2017	26 November 2018
Users	~120	~850
Contributions	~300	2908
Malware Families	614	1126
Malware Samples	1630	2989
References	906	2379
YARA Rules	113 / 116 / 20	775 / 209 / 54

A HUGE ***THANK YOU*** *TO ALL CONTRIBUTORS!*

Want an account? Ping me!

[1] https://malpedia.caad.fkie.fraunhofer.de
[2] https://malpedia.io

Overview
Operationalizing Malpedia

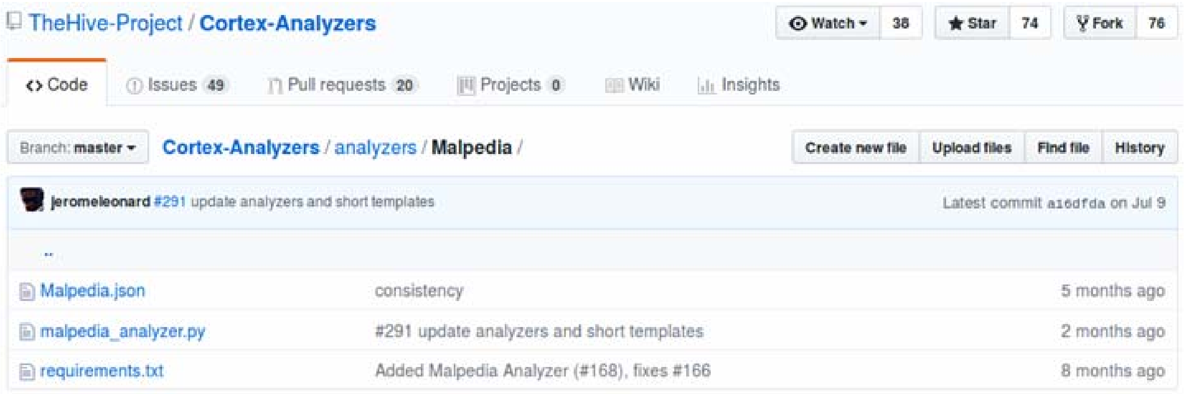
- Identification
  - YARA
  - Search / Comparison
  - Label Provider (Clustering)
- Contextualization
  - Publication references for families, actors, …
- QA / Regression Testing
  - Tools, Config extractors, etc

[1] https://github.com/TheHive-Project/Cortex-Analyzers/tree/master/analyzers/Malpedia

Fraunhofer FKIE

Overview
Operationalizing Malpedia

- Identification
  - YARA
  - Search / Comparison
  - Label Provider (Clustering)
- Contextualization
  - Publication references for families, actors, …
- QA / Regression Testing
  - Tools, Config extractors, etc



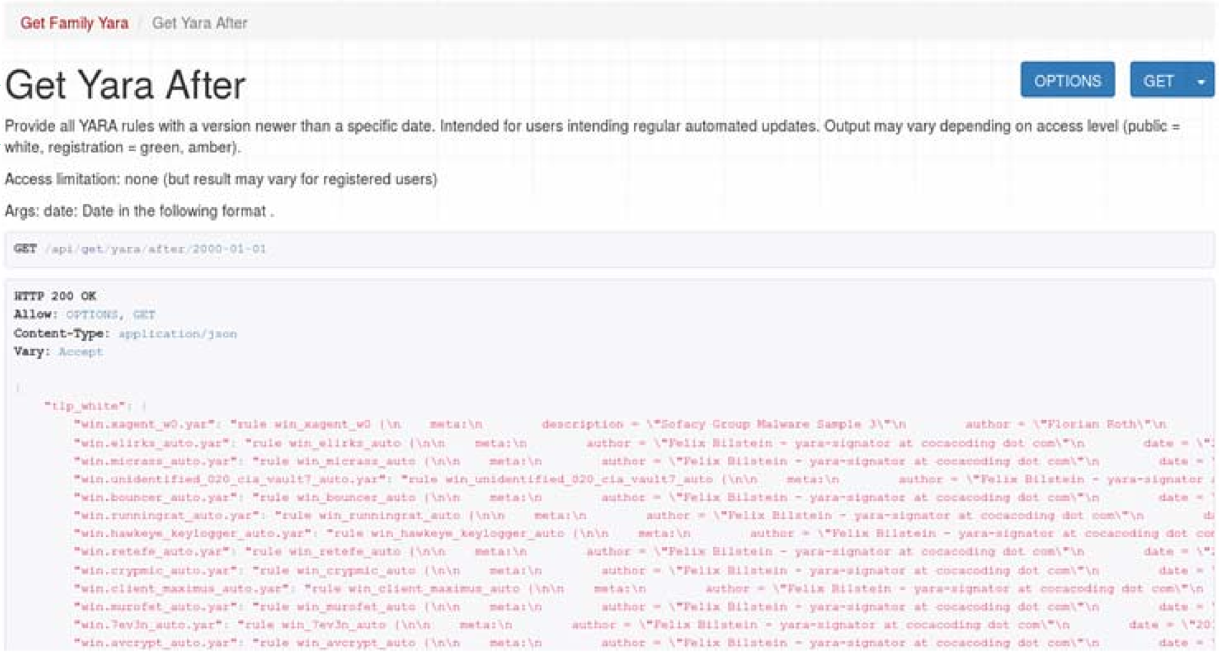
[1] https://github.com/TheHive-Project/Cortex-Analyzers/tree/master/analyzers/Malpedia

Overview

Operationalizing Malpedia

- Identification
  - YARA
  - Search / Comparison
  - Label Provider (Clustering)
- Contextualization
  - Publication references for families, actors, …
- QA / Regression Testing
  - Tools, Config extractors, etc

Malpedia REST API



[1] https://malpedia.caad.fkie.fraunhofer.de/api/get/yara/after/2000-01-01

Fraunhofer FKIE

Overview

Operationalizing Malpedia

- Identification
  - YARA
  - Search / Comparison →
  - Label Provider (Clustering)
- Contextualization →
  - Publication references for families, actors, …
- QA / Regression Testing
  - Tools, Config extractors, etc

MISP Threat Sharing

[1] https://github.com/MISP/misp-galaxy/blob/master/clusters/malpedia.json

Fraunhofer FKIE

Malware Code Cartography - Part I

Windows API Usage Recovery & Analysis for Malware Characterization

joint work with Steffen Enders, Elmar Padilla

Fraunhofer FKIE

Windows API Usage Recovery
Motivation

„(Windows) API interactions are an essential cornerstone for effective reverse engineering“

Fraunhofer FKIE

Windows API Usage Recovery

Overview

- Tool: ApiScout [1]
  - *Originally introduced at Botconf, December 2017*
  - Library for painless (Windows) API reconstruction in known environments
  - Idea: API function offset bruteforcing based on databases

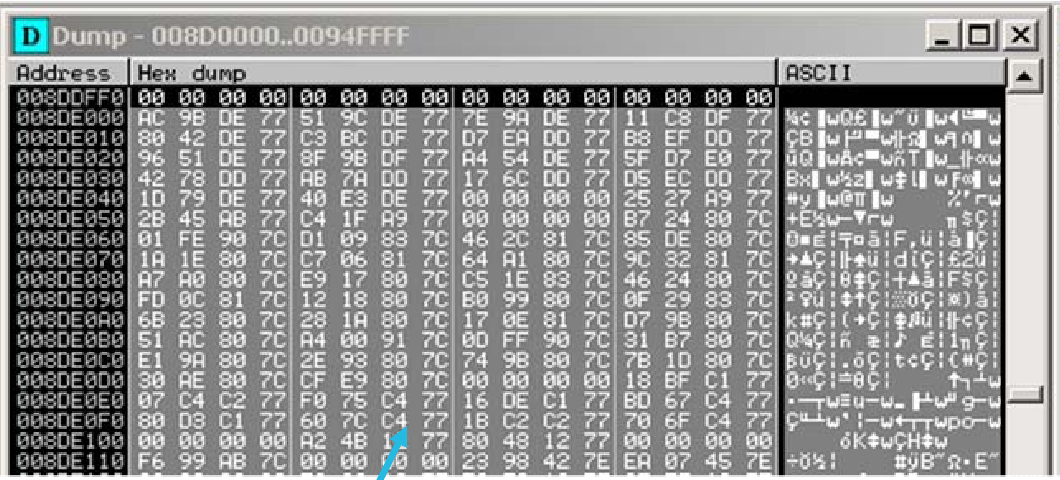
- Extension: ApiVectors
  - Compact representation (bit vector) indicating the presence of relevant WinAPI functions
  - Enables fast assessment of malware's potential capabilities
  - Allows similarity analysis based on WinAPI usage characteristics

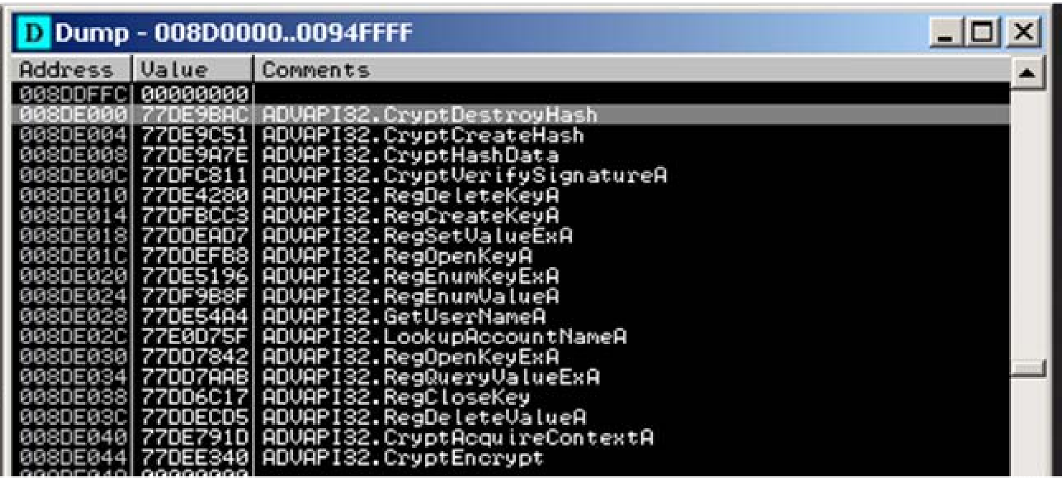
[1] https://github.com/danielplohmann/apiscout

Fraunhofer FKIE

Windows API Usage Recovery

ApiScout: Approach





These are
pretty static offsets…
-> Build a database!

Fraunhofer FKIE

Windows API Usage Recovery

ApiScout: WinAPI Measurements

		All		Unique		
Name	Version/Build	APIs	DLLs	APIs	DLLs	Address Collisions
Win XP	NT5.1/2600	128,408	1,597	101,701	1,584	1
Win 7	NT6.1/7601	251,186	3,828	168,176	2,215	178
Win 8.1	NT6.3/9600	282,802	5,154	183,424	3,024	**55,181**
Win 10	NT10.0/17134	338,456	5,971	234,528	3,751	**115,022**
Unique				**323,851**	**5,686**	

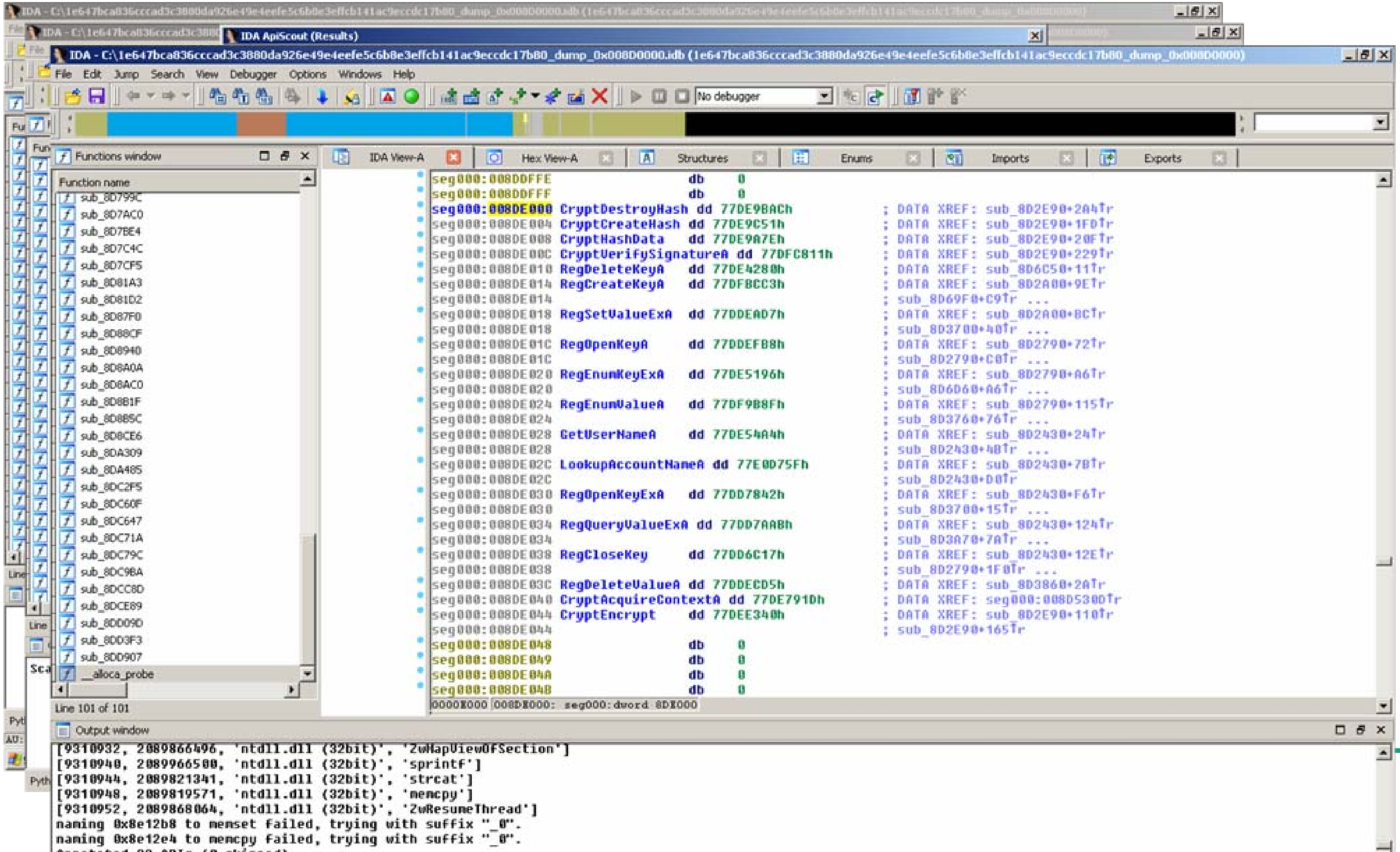
Only 4,664 APIs from 64 DLLs observed being used across 702 malware families.

Win8+: Forced ASLR!
0x10000000 / 0x180000000
Database only valid for running state :(

Fraunhofer FKIE

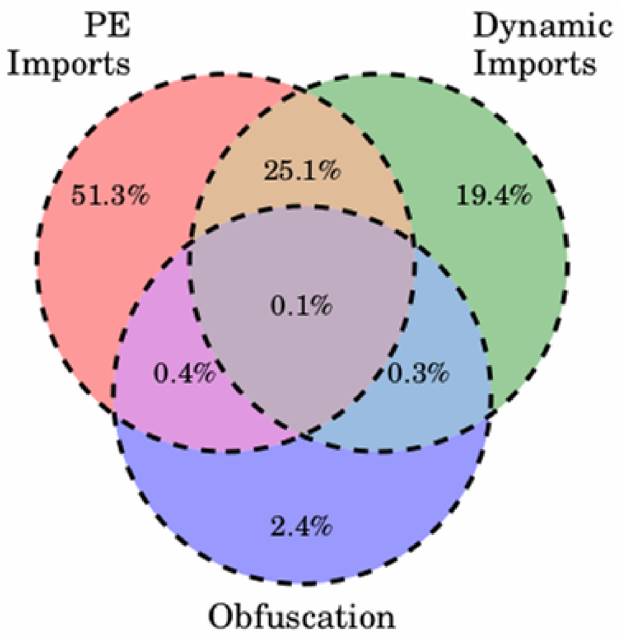
Windows API Usage Recovery

ApiScout Methodology



Windows API Usage Recovery

WinAPI Availability for Static Analysis / Methods of API Usage



- Across 702 families *(90 ignored -> .net)*
- PE Imports:
  - From PE Header Import Table only
- Dynamic + Cached:
  - LoadLibrary / GetProcAddress
    ApiHashing -> Custom IAT

Covered by ApiScout [1]

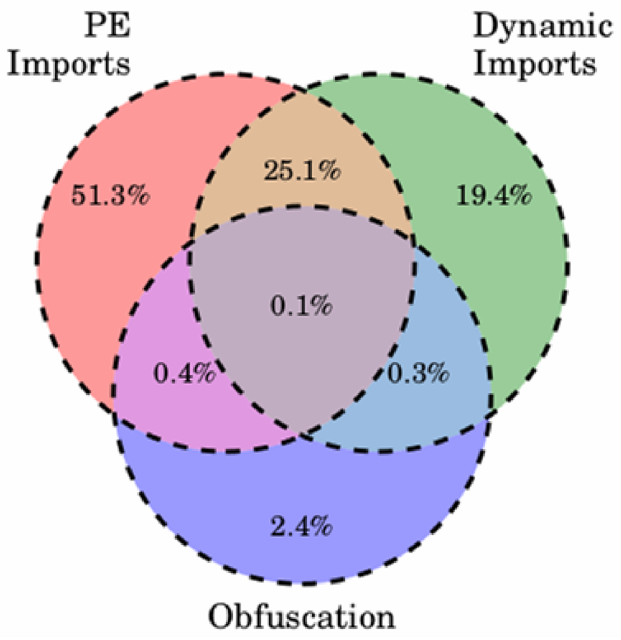
- Obfuscation:
  - Custom Jump Table *(Andromeda)*
    Offset-based Hook Avoidance *(Chthonic)*
    On-Demand Table *(Dridex)*
    Dynamic Resolving *(Shifu)*
    Imports on Stack / Heap *(PIVY, Cryptowall)*
    XORed Imports *(Qadars)*
    … more

[1] https://github.com/danielplohmann/apiscout

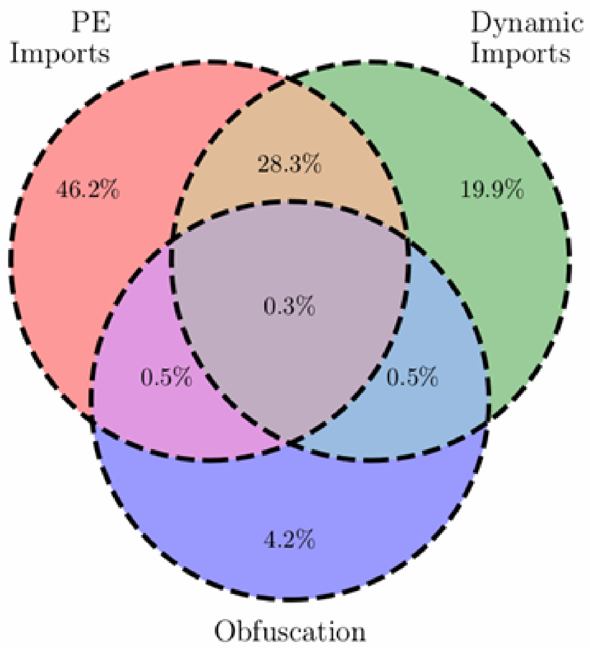
Fraunhofer FKIE

Windows API Usage Recovery

WinAPI Availability for Static Analysis / Methods of API Usage



- 2018, 702 families



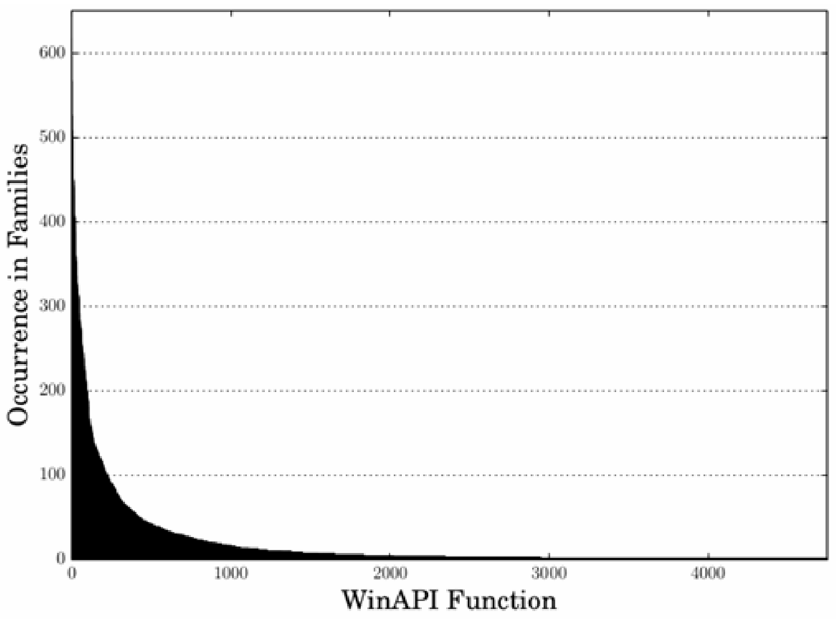
- 2017, 382 families

[1] https://github.com/danielplohmann/apiscout

Windows API Usage Recovery

Occurrence Frequency of Individual WinAPI Functions

- Occurrence frequency per Windows API function



- There are only very few „omnipresent“ APIs
  - Only 48 API functions in > 50% families
  - 4,392 (92.52%) of API functions <= 10% families
- API compositions are highly specific per family
  - Indeen good for (identification) tools like
    - ImpHash [1]
    - ImpFuzzy [2]
    - ApiVectors!

[1] https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html
[2] http://blog.jpcert.or.jp/2017/03/malware-clustering-using-impfuzzy-and-network-analysis---impfuzzy-for-neo4j-.html

Windows API Usage Recovery

Semantic Context for Windows API Functions

Kudos to Quoscient.io
for their contributions!
(Patrick Ventuzelo, Lukas Bernhard)

- Define: API Context Groups
  - Manually labelled ~4.500 APIs, primary (12) and secondary class (115)

GUI	1392
System	636
Execution	590
String	458
Network	387
FileSystem	352
Device	170
Crypto	131
Other	127
Memory	118
Registry	80
Time	44

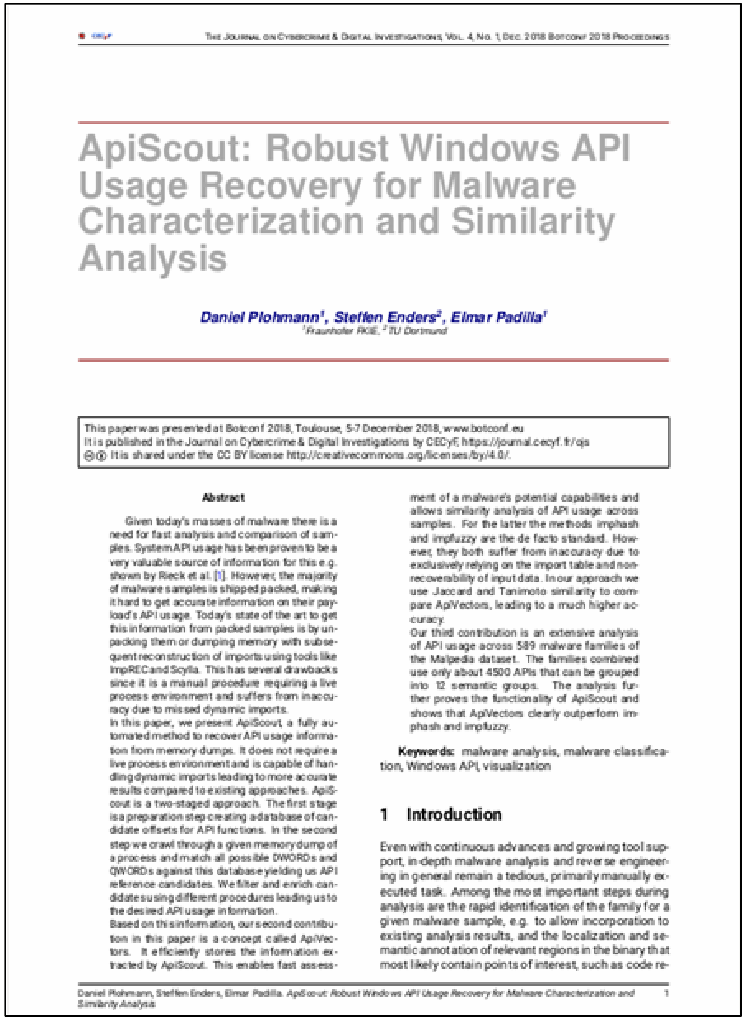
Windows API Usage Recovery

Semantic Context for Windows API Functions

- Goal: Find an (optimal?) vector composition based on this!

GUI	1392
System	636
Execution	590
String	458
Network	387
FileSystem	352
Device	170
Crypto	131
Other	127
Memory	118
Registry	80
Time	44

- We wrote a paper on this.
  - Extensive description & evaluation



THE JOURNAL ON CYBERCRIME & DIGITAL INVESTIGATIONS, VOL. 4, NO. 1, DEC. 2018 BOTCONF 2018 PROCEEDINGS

ApiScout: Robust Windows API Usage Recovery for Malware Characterization and Similarity Analysis

Daniel Plohmann[1], Steffen Enders[2], Elmar Padilla[1]
[1]Fraunhofer FKIE, [2]TU Dortmund

Abstract

Given today's masses of malware there is a need for fast analysis and comparison of samples. System API usage has been proven to be a very valuable source of information for this e.g. shown by Rieck et al. [1]. However, the majority of malware samples is shipped packed, making it hard to get accurate information on their payload's API usage. Today's state of the art to get this information from packed samples is by unpacking them or dumping memory with subsequent reconstruction of imports using tools like ImpREC and Scylla. This has several drawbacks since it is a manual procedure requiring a live process environment and suffers from inaccuracy due to missed dynamic imports.

In this paper, we present *ApiScout*, a fully automated method to recover API usage information from memory dumps. It does not require a live process environment and is capable of handling dynamic imports leading to more accurate results compared to existing approaches. ApiScout is a two-staged approach. The first stage is a preparation step creating a database of candidate offsets for API functions. In the second step we crawl through a given memory dump of a process and match all possible DWORDs and QWORDs against this database yielding us API reference candidates. We filter and enrich candidates using different procedures leading us to the desired API usage information.

Based on this information, our second contribution in this paper is a concept called ApiVectors. It efficiently stores the information extracted by ApiScout. This enables fast assessment of a malware's potential capabilities and allows similarity analysis of API usage across samples. For the latter the methods imphash and impfuzzy are the de facto standard. However, they both suffer from inaccuracy due to exclusively relying on the import table and non-recoverability of input data. In our approach we use Jaccard and Tanimoto similarity to compare ApiVectors, leading to a much higher accuracy.

Our third contribution is an extensive analysis of API usage across 589 malware families of the Malpedia dataset. The families combined use only about 4500 APIs that can be grouped into 12 semantic groups. The analysis further proves the functionality of ApiScout and shows that ApiVectors clearly outperform imphash and impfuzzy.

Keywords: malware analysis, malware classification, Windows API, visualization

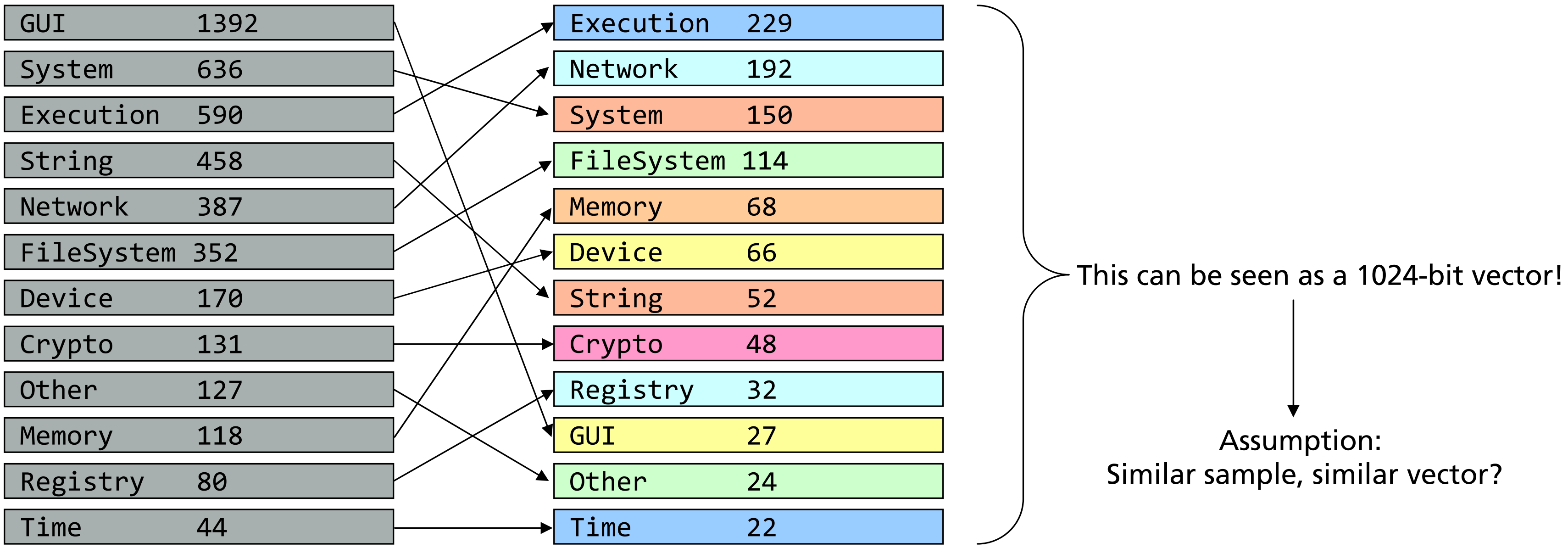
1 Introduction

Even with continuous advances and growing tool support, in-depth malware analysis and reverse engineering in general remain a tedious, primarily manually executed task. Among the most important steps during analysis are the rapid identification of the family for a given malware sample, e.g. to allow incorporation to existing analysis results, and the localization and semantic annotation of relevant regions in the binary that most likely contain points of interest, such as code re-

Windows API Usage Recovery

WinAPI Reference Vector

- Define: API Context Groups
  - Reduce this set to 1024 WinAPIs (~80% hierarchy, ~20% based on domain-knowledge)
  - Vector yields 90% coverage (mean) for APIs found by ApiScout for ~600 malware families

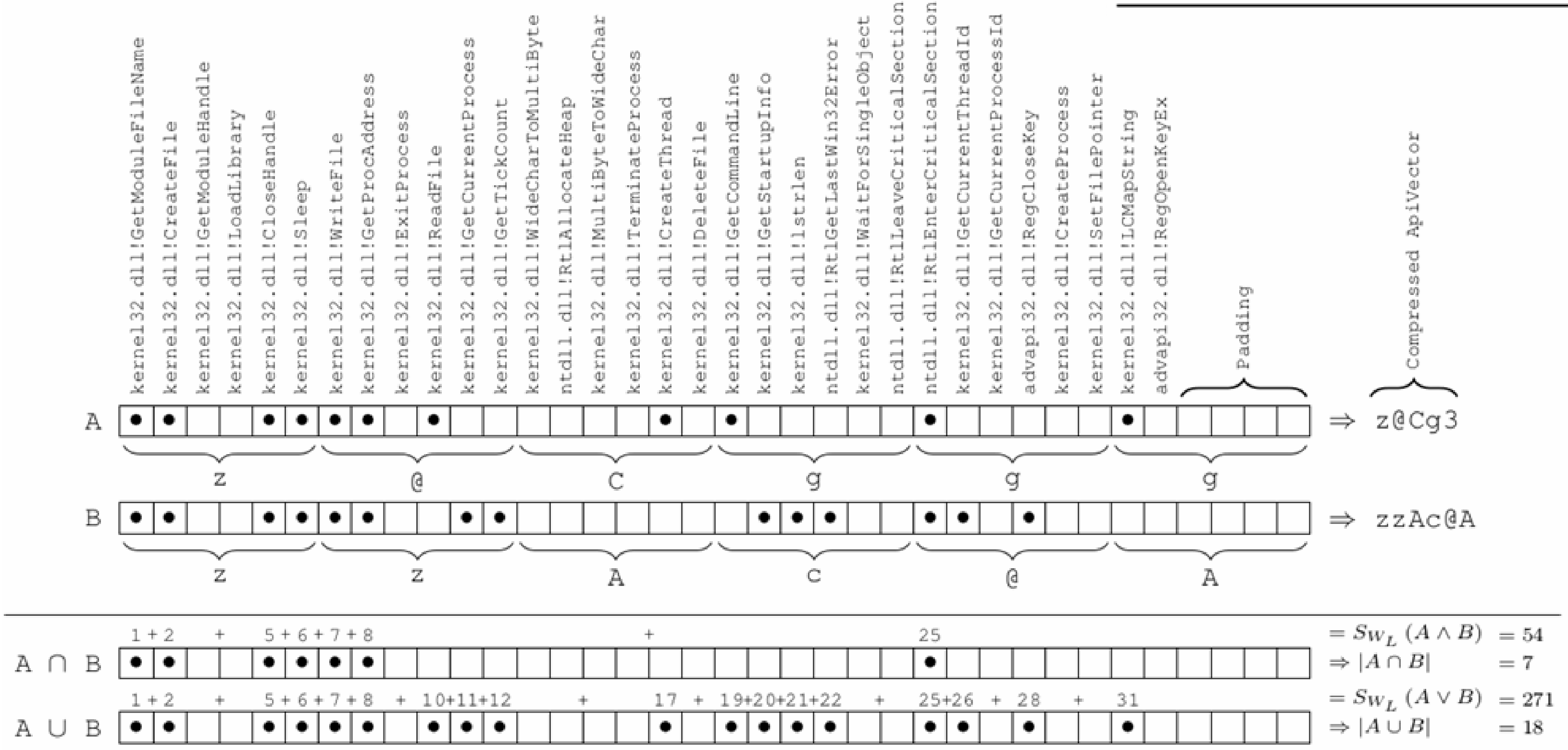


Fraunhofer FKIE

Windows API Usage Analysis

Vector Construction

000000	A	011010	a	110100	@	111010	*
000001	B	011011	b	110101	}	111011	/
000010	C	011100	c	110110	]	111100	?
...	...	...	...	110111	^	111101	,
011000	Y	110010	y	111000	+	111110	.
011001	Z	110011	z	111001	-	111111	_



$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{7}{18} \approx 0.39$$

$$J'_{W_L}(A,B) = \frac{Sw_L(A \wedge B)}{Sw_L(A \vee B)} = \frac{54}{271} \approx 0.20$$

Fraunhofer FKIE

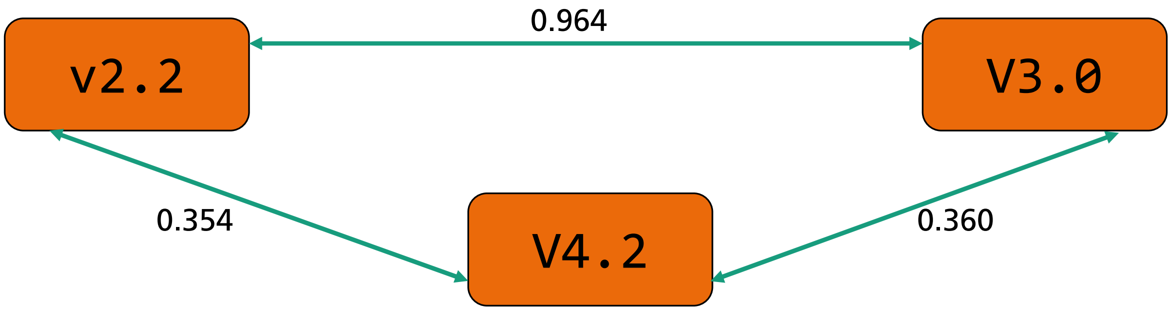
Windows API Usage Analysis

Comparison of ApiVectors

- Example Vectors
  - Base64-like encoding (Run-Length compressed) - 4-172 bytes long


```
A42gA28KA13   CAAMA16BABAAJAECAxMAACkAAQUA7CJBCgAgUBA3 kQCBAHJSRjU^q-*}_pb__N,__^?
A42gA28KA13   CAAMA16BABAAJAEAAxMAACkAAQUA7CJBCgAAUBA3 kQCBAHJSRjU^q-*}_pL__N,._^?
A41BA29CA4IA9gCA9gA8Q  BAAJAEAABMA3 gAAQA8 QJRCgAgUBAAHkQARCDIADDBGAqQAgCcGOIOp,f?
```

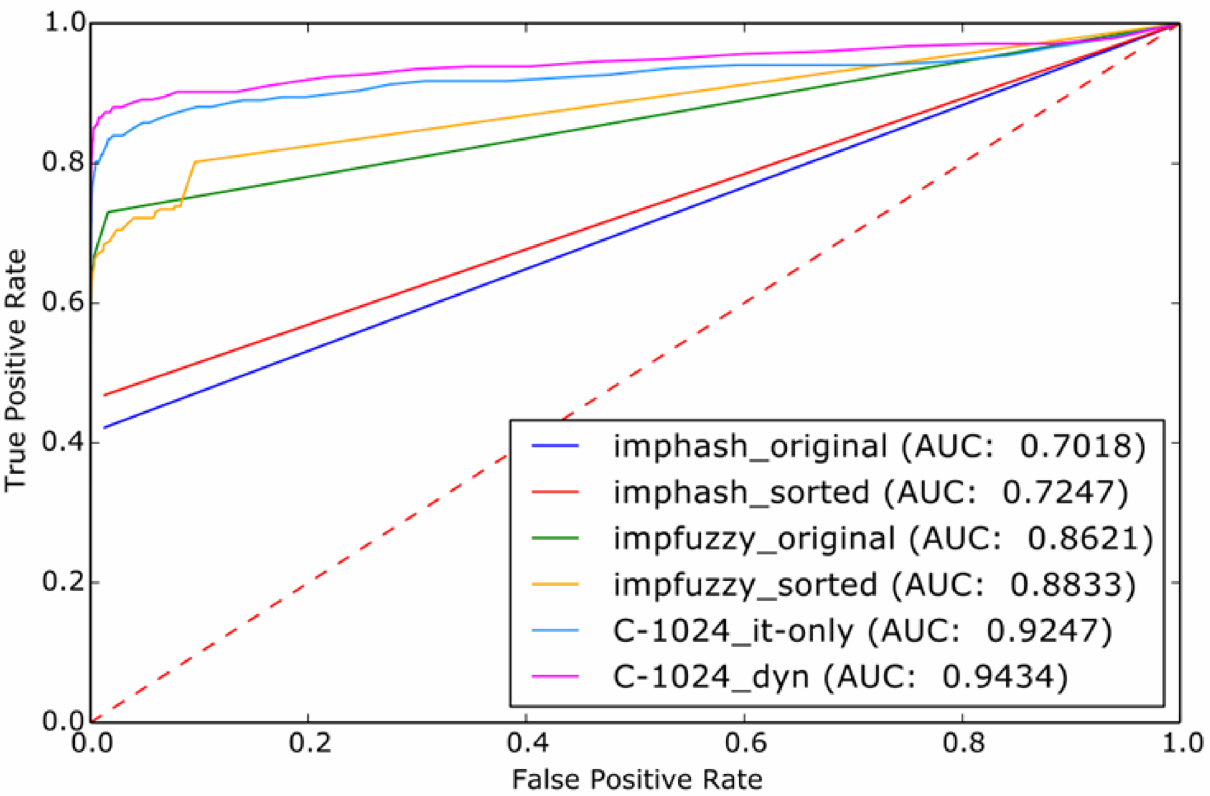

TeslaCrypt 2.2, 3.0, 4.2



Fraunhofer FKIE

Windows API Usage Analysis

Evaluation of Matching Performance

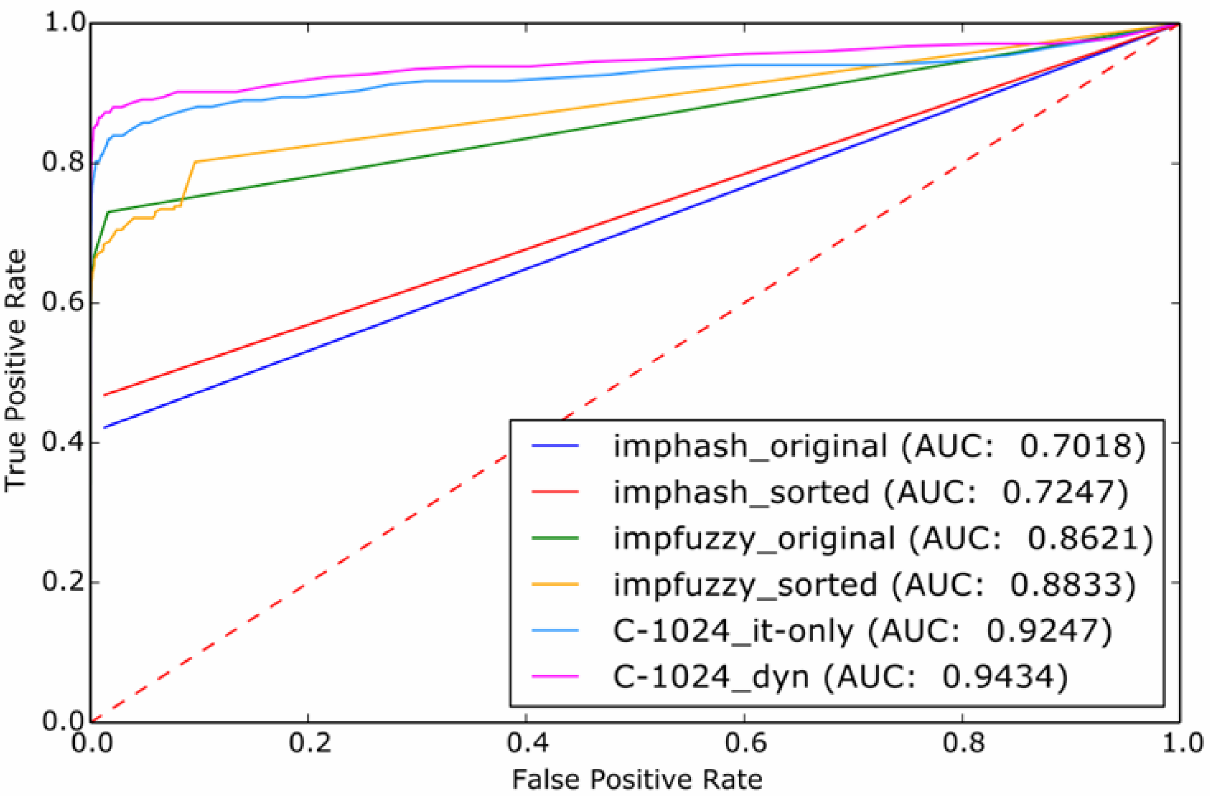


- Data set: Malpedia (2018-05-17)
  - 673 families, 1854 samples
- Comparison with ImpHash, ImpFuzzy
  - Mean Fingerprint sizes:
    - ImpHash: 32 bytes
    - ImpFuzzy: 54.4 bytes
    - ApiVector: 74.3 bytes
  - ApiVector: recoverable info
- Performance @ Thresholds
  - T: 0.18 – 90.18% TPR, 9.45% FPR
  - T: 0.22 – 89.10% TPR, 4.74% FPR
  - T: 0.32 – 86.55% TPR, 0.99% FPR
  - T: 0.55 – 80.72% TPR, 0.09% FPR

Fraunhofer FKIE

Windows API Usage Analysis

Evaluation of Matching Performance



- General Challenges to API-based similarity analysis
  - Packers
  - .NET / scripts
  - Statically linked code (MSVCRT, Delphi, Go, …)

Windows API Usage Recovery & Analysis

How to operationalize this?

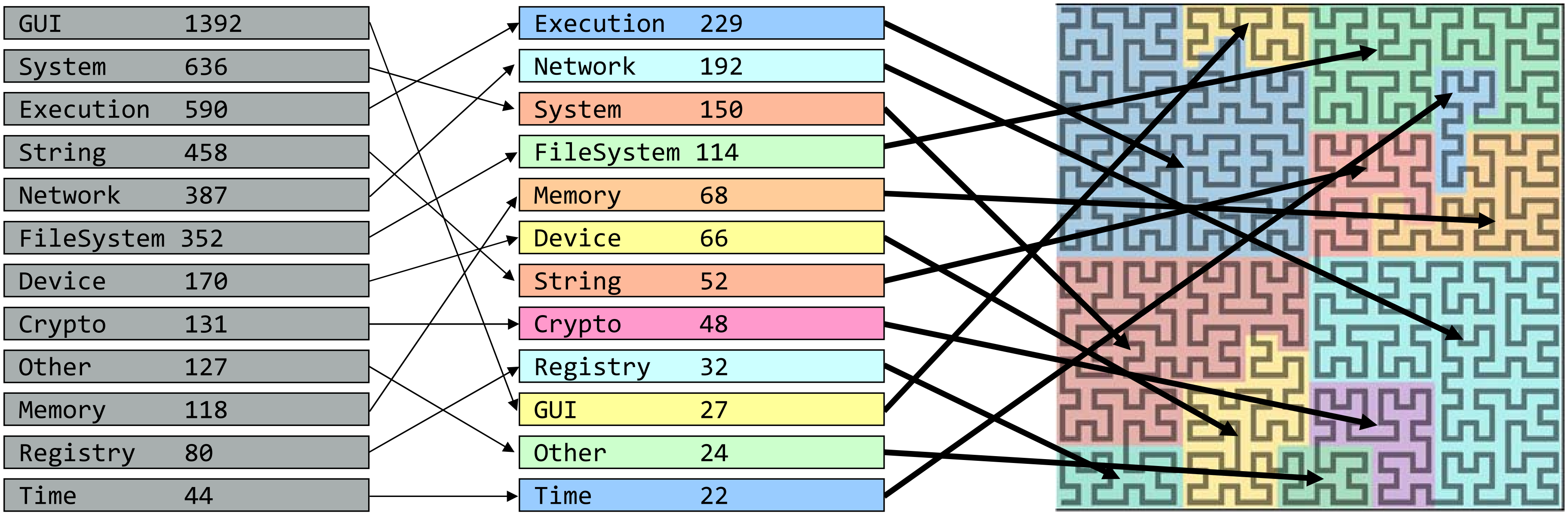
- ApiScout available on GitHub [1]
- Projects using ApiScout:
  - Angad [2] by Ankur Tyagi, presented @ BsidesZurich [3]
  - Master of Clusters by Andrea Garavaglia presented @ MISP Summit / hack.lu [4]
  - AssemblyLine
- Malpedia!

[1] https://github.com/danielplohmann/apiscout
[2] https://github.com/7h3rAm/angad
[3] https://bsideszh.ch/agenda/abstracts/
[4] https://2018.hack.lu/misp-summit/

Windows API Usage Recovery & Analysis

Vector Visualization

- Visualize Vectors:
  - Hilbert Curve to ensure neighboring of contexts

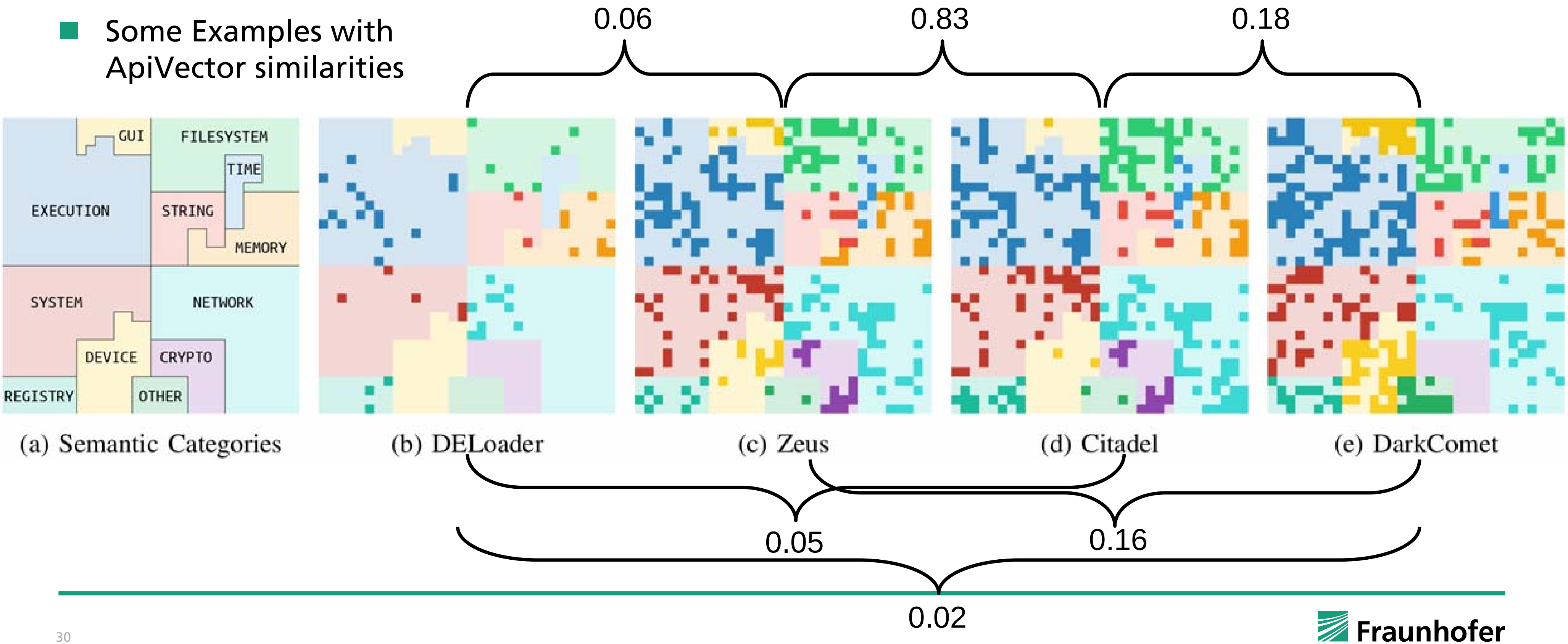


Fraunhofer FKIE

Windows API Usage Recovery & Analysis

Vector Visualization - ApiQR

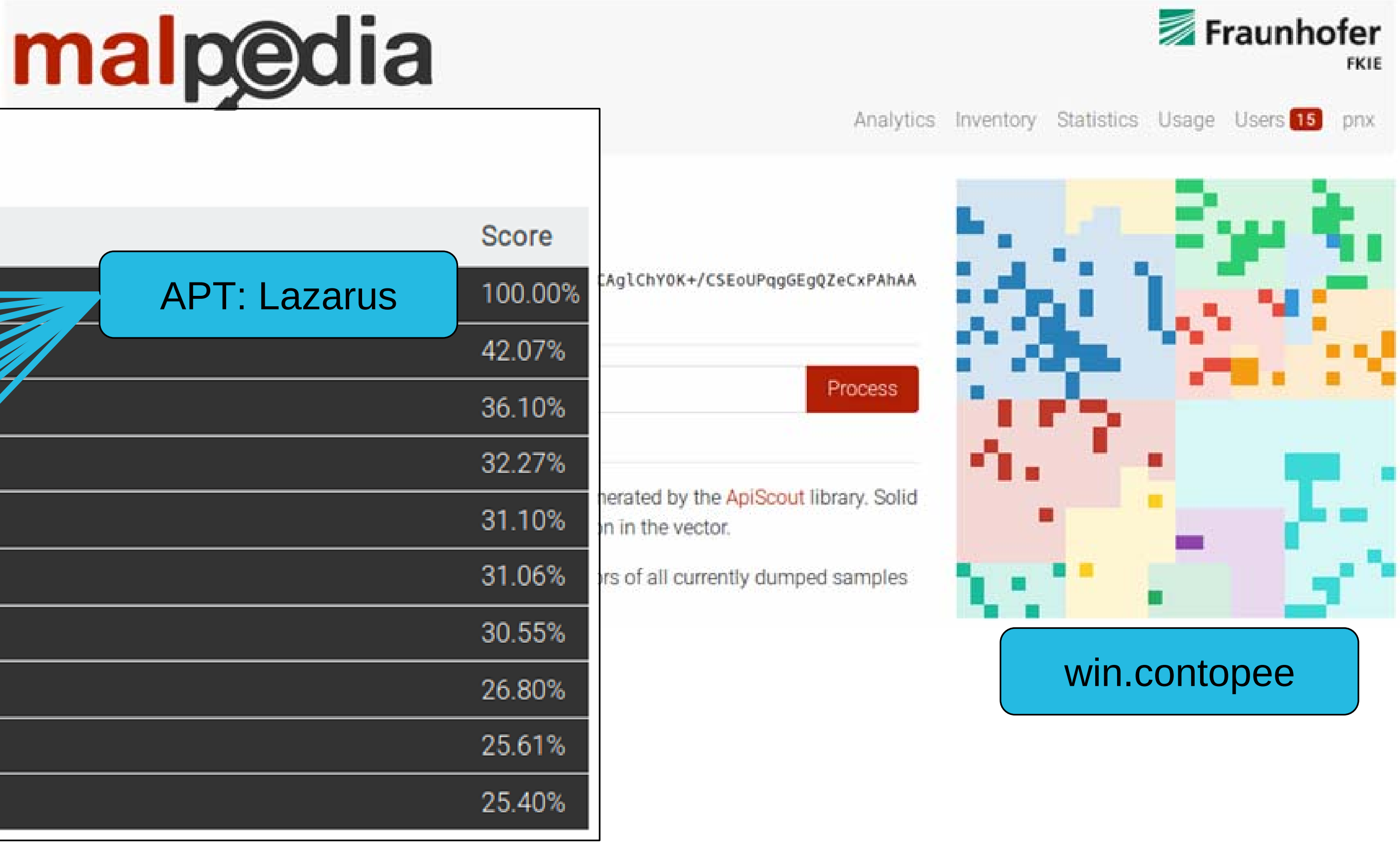
- Some Examples with ApiVector similarities



Fraunhofer FKIE

Windows API Usage Recovery & Analysis

ApiVectors Similarity Analysis



Match Results:

Top 10 Family Matches:

Family	Score
win.contopee	100.00%
win.alreay	42.07%
win.volgmer	36.10%
win.unidentified_042	32.27%
win.unidentified_032	31.10%
win.romeos	31.06%
win.sierras	30.55%
win.joanap	26.80%
win.reaver	25.61%
win.duuzer	25.40%

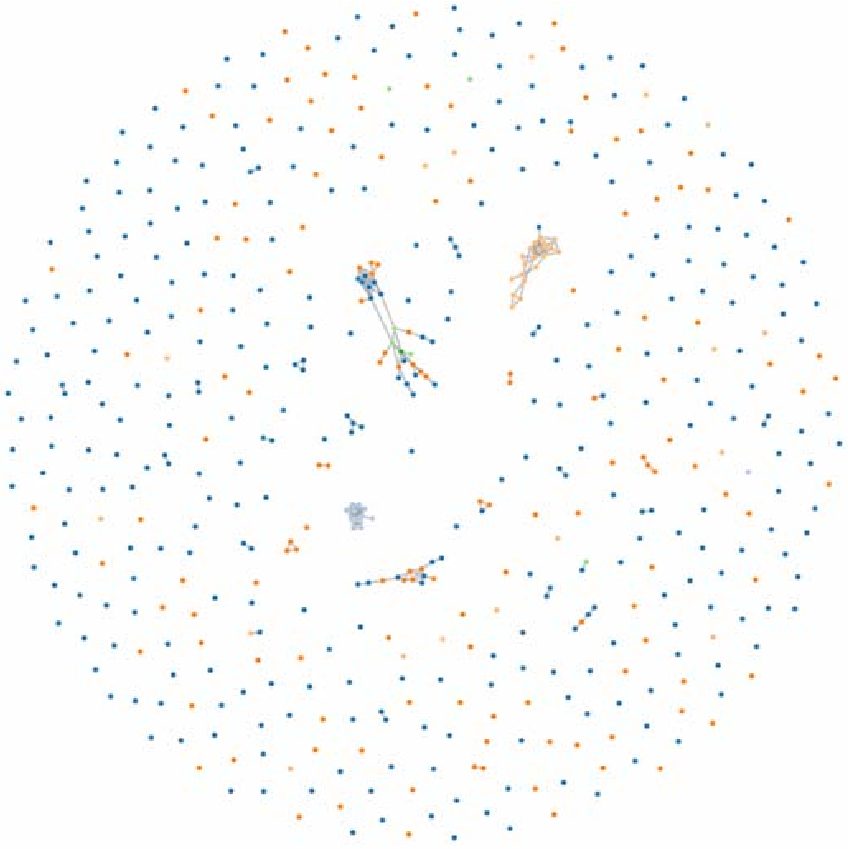
[1] https://malpedia.caad.fkie.fraunhofer.de/apiqr/

Fraunhofer FKIE

Windows API Usage Recovery & Analysis

Clusters

- Lazarus is an extreme case (also known for some degree of code-reuse across families)!
- However, there are definitely other interesting clusters to explore.
- Hypothesis: WinAPI usage patterns seem to be correlating with code-similarity?



cross-family matches, threshold > 0.5

[1] https://malpedia.caad.fkie.fraunhofer.de/apiqr/

Malwar Code Cartography: Part II

Code-based Similarity Analysis

joint work with Paul Hordiienko, Steffen Enders, Elmar Padilla

(Work in Progress)

Fraunhofer FKIE

Code-based Similarity Analysis

Motivation

- Code Similarity Analysis
  - Identify (3rd party) shared library code: automated annotation / exclusion from analysis scope
  - Isolate code that is immanent to a given code base / author
- Related Work:
  - Kam1n0 [1] by Stephen Ding et al.
  - FunctionSimSearch [2] by Thomas Dullien et al.
  - CosaNostra / MalTindex [3] by Joxean Koret
  - More…

[1] https://github.com/McGill-DMaS/Kam1n0-Community
[2] https://github.com/googleprojectzero/functionsimsearch
[3] https://github.com/joxeankoret/

Code-based Similarity Analysis

Overview

- Tool: SMDA [2]
  - „SMDA is a minimalist recursive disassembler library that is optimized for accurate Control Flow Graph (CFG) recovery from memory dumps."
  - Work in progress – built on top of Capstone [1], *already silently released on GitHub [2]*
  - ~95% accuracy on an internal test data set (50 manually labeled memory dumps of malware families)
  - Formal evaluation underway

- Tool: MCRIT
  - „MinHash-based Code Relationship Identification Toolkit"
  - Work in progress, to be released

[1] https://github.com/aquynh/capstone
[2] https://github.com/danielplohmann/smda

Code-based Similarity Analysis

MinHash 101

- MinHashing
  - „Min-wise independent permutations“ - Locality Sensitive Hashing (LSH) scheme [1]
  - Fast estimation of set similarity (approximation of Jaccard similarity coefficient)

- Use cases:
  - text documents / websites (duplicates, plagiarism)
  - genome sequencing
  - code similarity! [2]

[1] “Min-wise independent permutations”. Broder et al., In: Proceedings of the 30th ACM Symposium on Theory of Computing (STOC '98), New York, NY, USA.
[2] “Binary Function Clustering using Semantic Hashes”. Jin et al., Carnegie Mellon University, 2012.

Code-based Similarity Analysis

MinHash 101

Kudos to Lukas Bernhard
for the fruitful discussions!

- MinHash procedure:
  - Extract a range of descriptive features („shingles“) for each object
  - Hash them n times with different hash functions (e.g. different seeds)
  - Select the minimum hash value for each of the n groups
  - The resulting sequence of n values is considered as the object‘s fingerprint

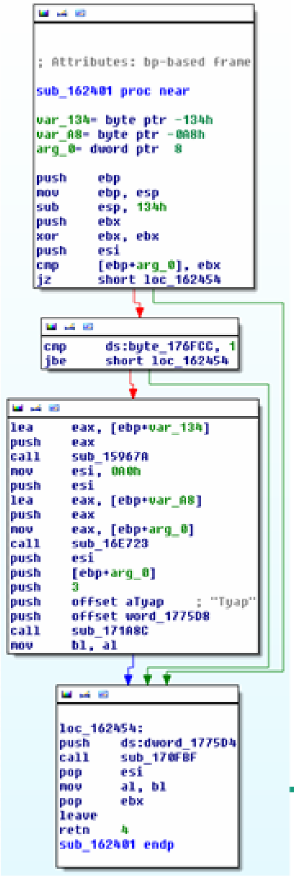
- Matching fingerprints:
  - Given two fingerprints, count the number of equal fields at same positions

- Various optimizations:
  - Single-hash XORing, Banding or n-key sorting, b-bit representation, …

Code-based Similarity Analysis

MCRIT

- Simplified example with a hash function that maps to a single output byte (0-255)



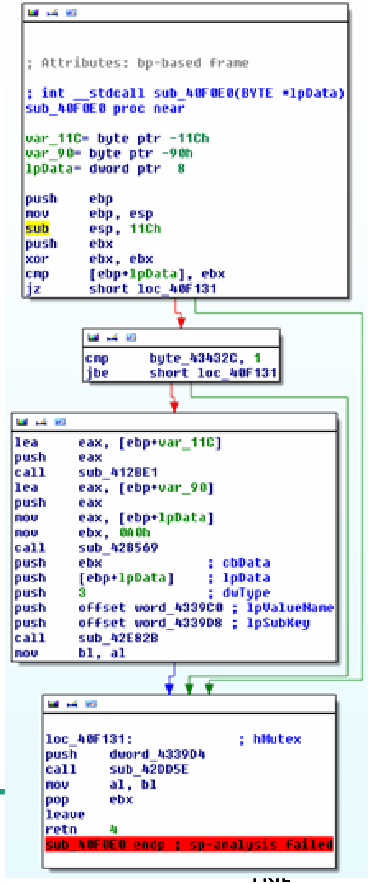
Derive shingles such as → ← Same shingles for all functions

Fingerprint A		Fingerprint B
Statistics:		Statistics:
„num_ins:33“		„num_ins:30“
„num_blocks:4“		„num_blocks:4“
„num_calls:4“		„num_calls:4“
…		…
Mnemonic N-grams:		Mnemonic N-grams:
„push-mov-sub-push“		„push-mov-sub-push“
„mov-sub-push-xor“,		„mov-sub-push-xor“,
„sub-push-xor-push“,		„sub-push-xor-cmp“
…		…

Fingerprint A: [77, 67, 82, 73, 84, 32, 121, 97, 121, 33]

Fingerprint B : [77, 99, 82, 73, 84, 45, 121, 97, 112, 33]

Graphlets, …
Strings, …
Fuzzy Abstractions, …
Fuzzy Windowing, …



Overlap: 7 / 10, Score: 0.7

Code-based Similarity Analysis

MCRIT

- Small test data set (in-memory):
  - 50 samples, 40 families
  - 26,097 functions with 20,611 indexable (greater or equal to 10 instructions or 3 basic blocks)

- Application of MCRIT
  - All function pairs: 20,611 * 20,610 / 2 = 212,396,355
  - Filter candidates down to 35,651 pairs (using „banding“)
  - This results in 19,732 matches above threshold (0.7)
  - Indexing + Matching in-memory takes ~2min on this laptop (i5, 8GB RAM).

- Formal validation pending
  - Win/Linux goodware binaries with symbols

BinDiff Threshold	0.90	0.99
BinDiff Matches	12,035	8,263
MCRIT Threshold	0.70	0.85
MCRIT Matches	19,732	11,648
MCRIT TPs	9,350	7,968
MCRIT TPR	0.7769	0.9643
MCRIT FPs (?)	3,515	766

Preliminary Results!

Code-based Similarity Analysis

MCRIT

- Malpedia data set (mongodb):
  - 2,403 samples, 773 families
  - 1,927,361 functions with 1,233,321 indexable (greater or equal to 10 instructions or 3 basic blocks)

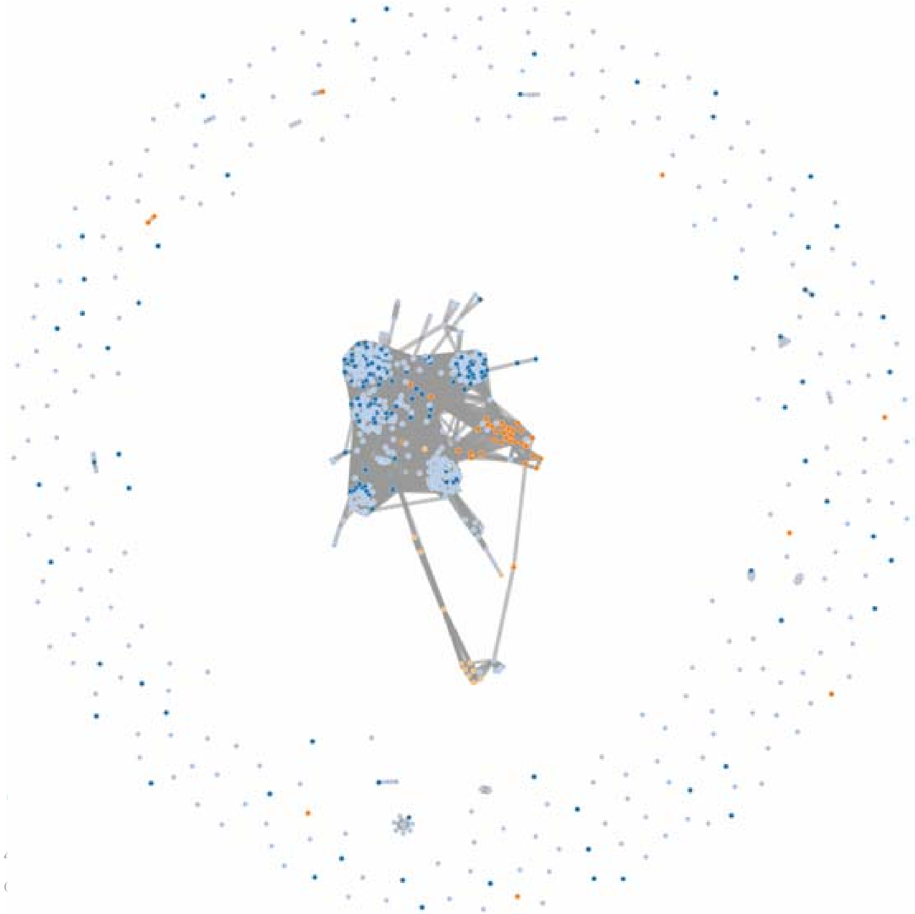
- Application of MCRIT
  - All function pairs: 1,233,321 * 1,233,320 / 2 = 760,539,727,860
  - Filter candidates down to 63,694,525 pairs
  - This results in 27,901,621 matches above threshold (0.7)
    -> 998,707 / 1,233,321 functions have a match.

- Runtime
  - Indexing: 13,902 sec (03:51:42h) – 138,64 FNs/sec
  - Candidate Identification: 6,380 sec (01:46:20h)
  - Matching: 31,840 sec (08:50:40h) – 1666,52 Pairs/sec
  - Total: 18h from disassembly to full matching results

Fraunhofer FKIE

Code-based Similarity Analysis
MCRIT Results

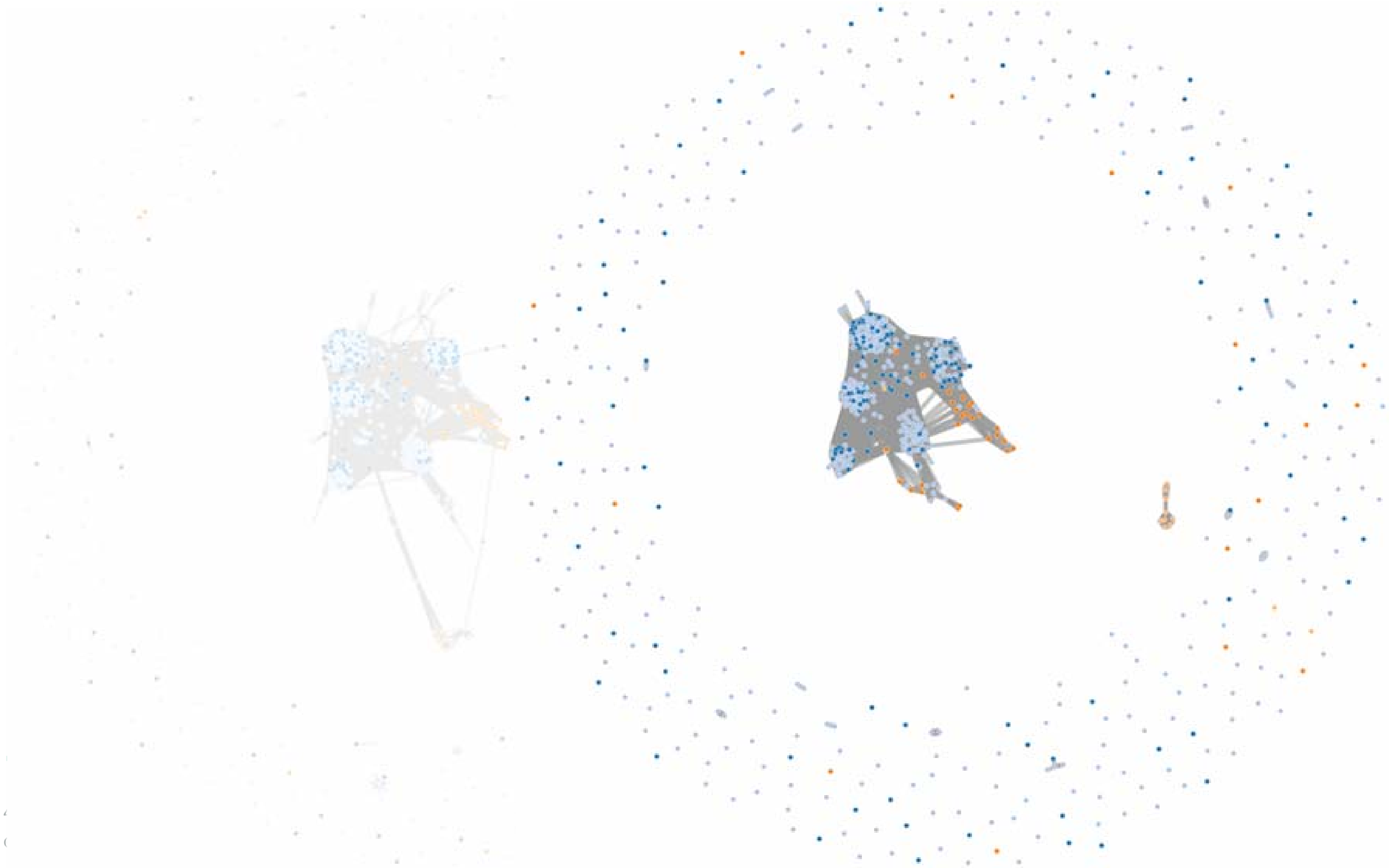
- Let‘s look at similarity between families! Let‘s try a threshold of… 0.2!



Code-based Similarity Analysis

MCRIT Results

- Let's look at similarity between families! Let's try a threshold of… ~~0.2~~! 0.3!



Code-based Similarity Analysis

MCRIT Results

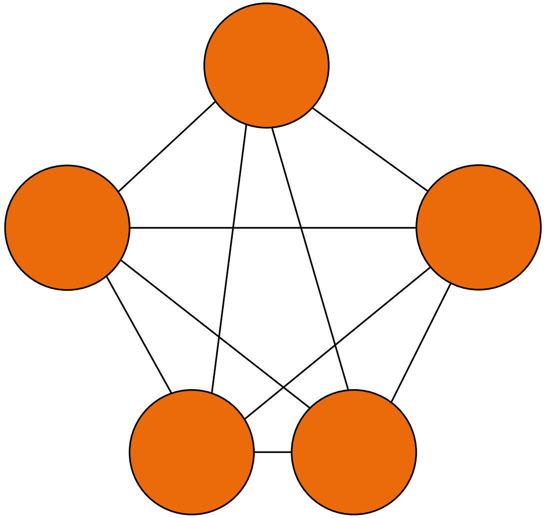
- Let's look at similarity between families! Let's try a threshold of… ~~0.2~~! ~~0.3~~! 0.5!

Code-based Similarity Analysis

MCRIT Results: Function Match Clusters

- A significant part of these matches is potentially the result of common 3rd party code

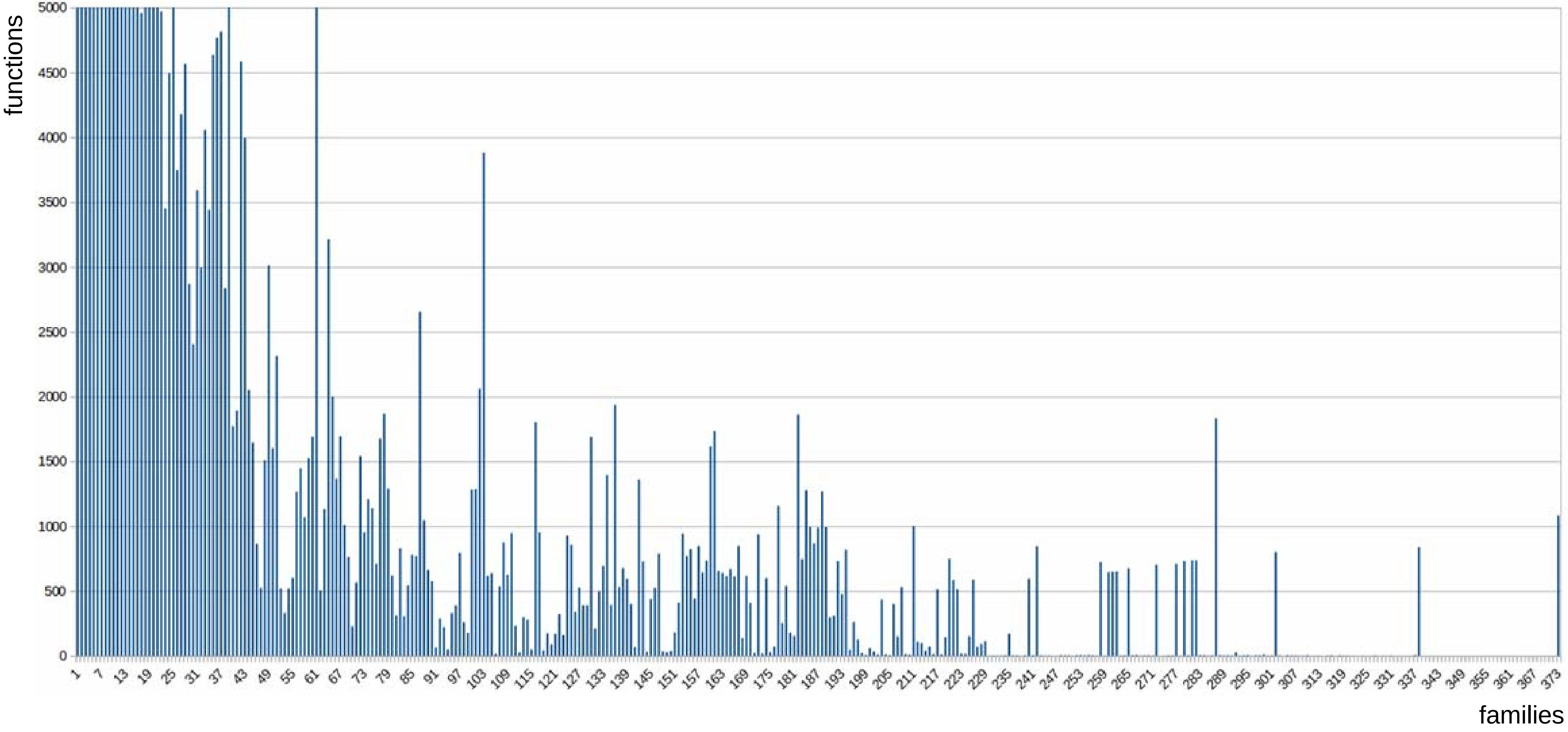
- How to identify them?
- Function Match Clusters:
  - A group of samples/families, where one of their function matches into all the others
  - Also known as: Strongly Connected Component (SCC) :)



Fraunhofer FKIE

Code-based Similarity Analysis

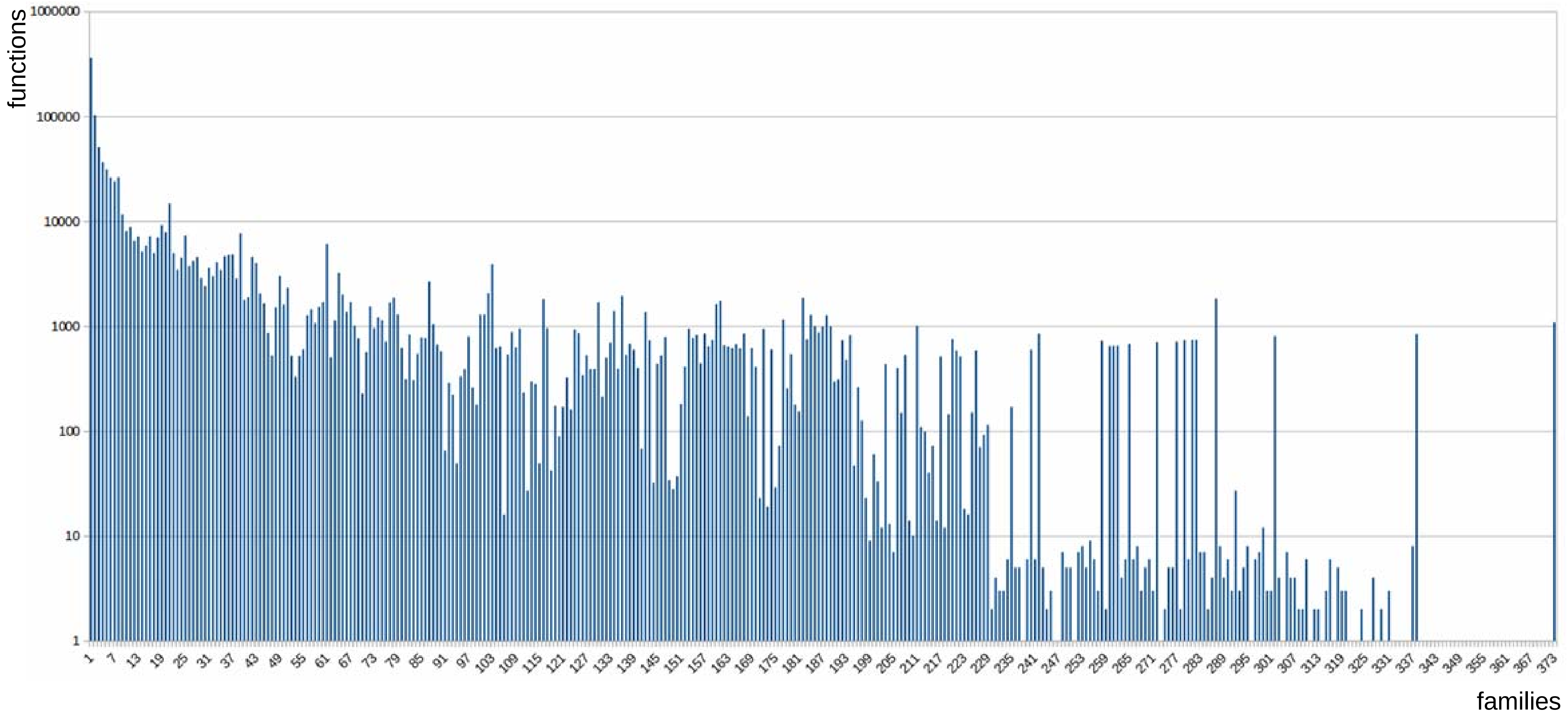
MCRIT Results: Function Match Clusters



Fraunhofer FKIE

Code-based Similarity Analysis

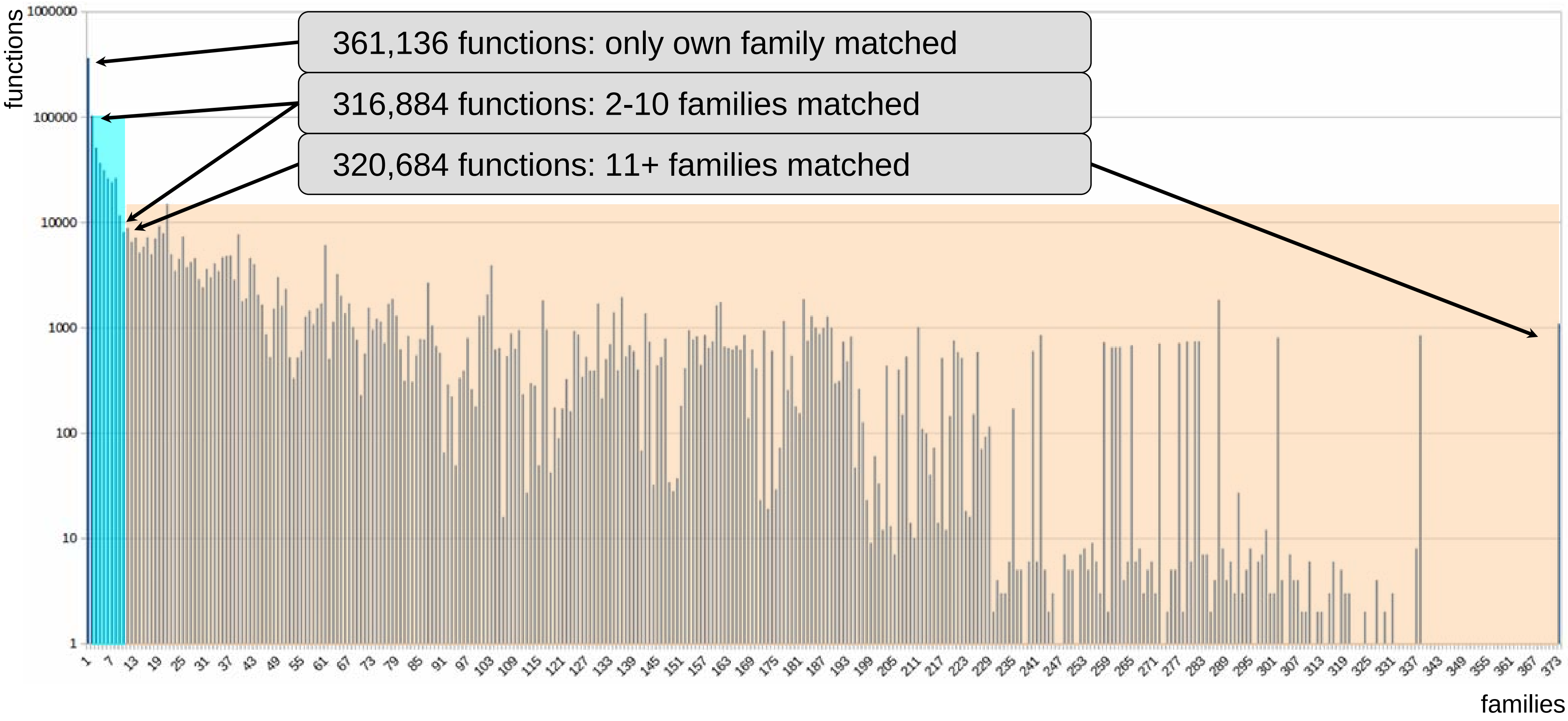
MCRIT Results: Function Match Clusters (logscale)



Fraunhofer FKIE

Code-based Similarity Analysis

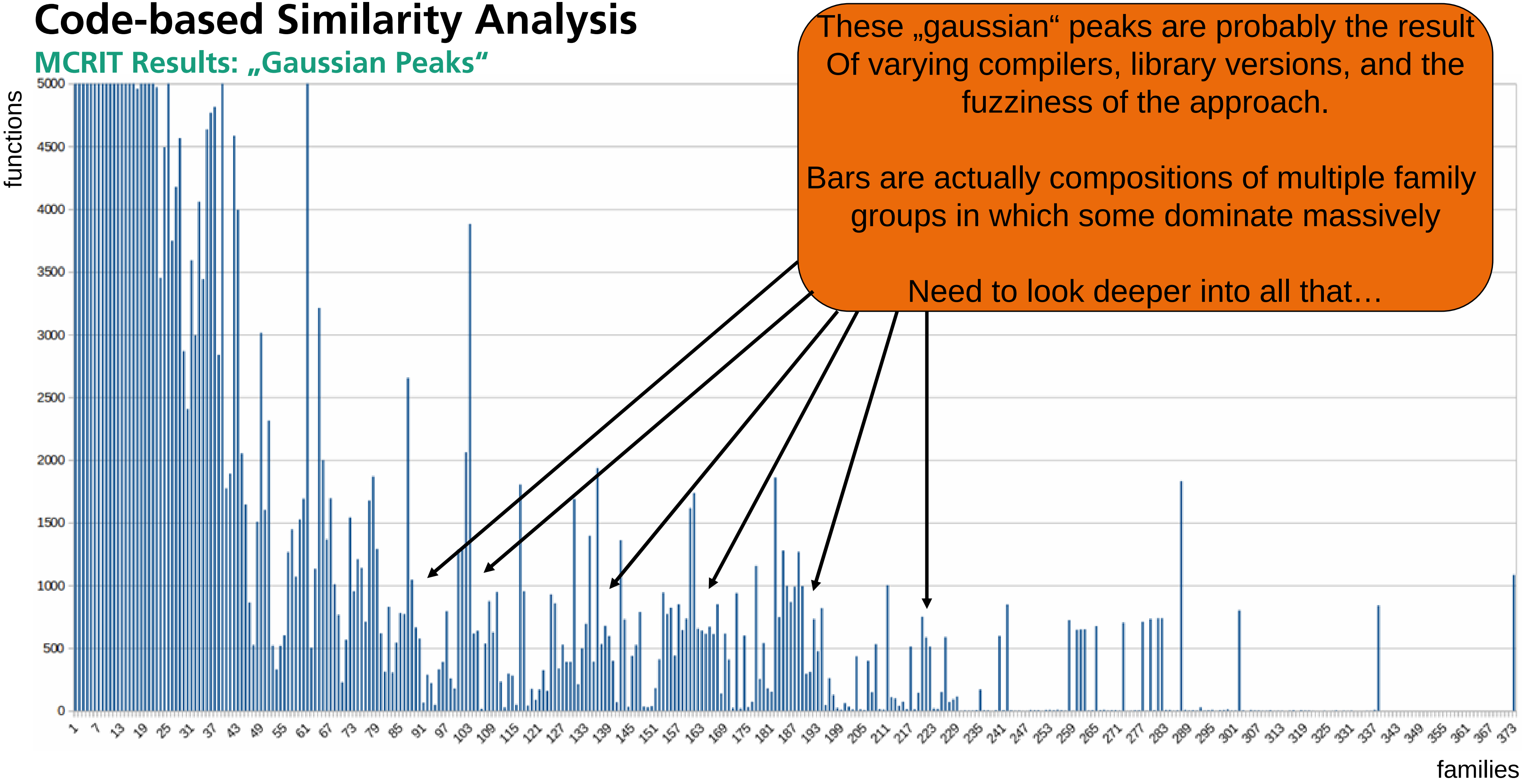
MCRIT Results („Approximation“ of shared code clusters)



Fraunhofer FKIE

Code-based Similarity Analysis

MCRIT Results: „Gaussian Peaks“

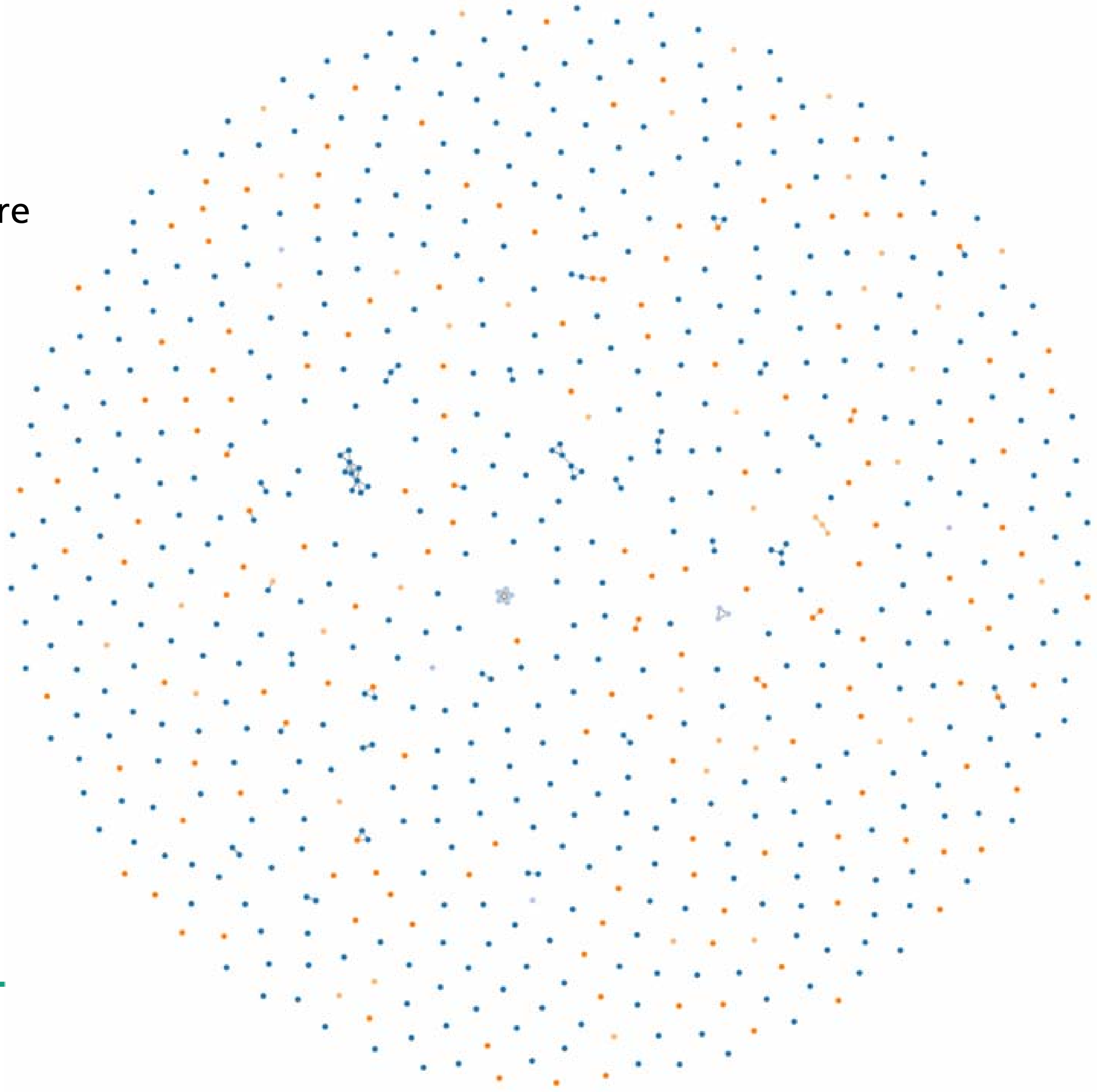


Fraunhofer
FKIE

Code-based Similarity Analysis

MCRIT Results: Filtered Results

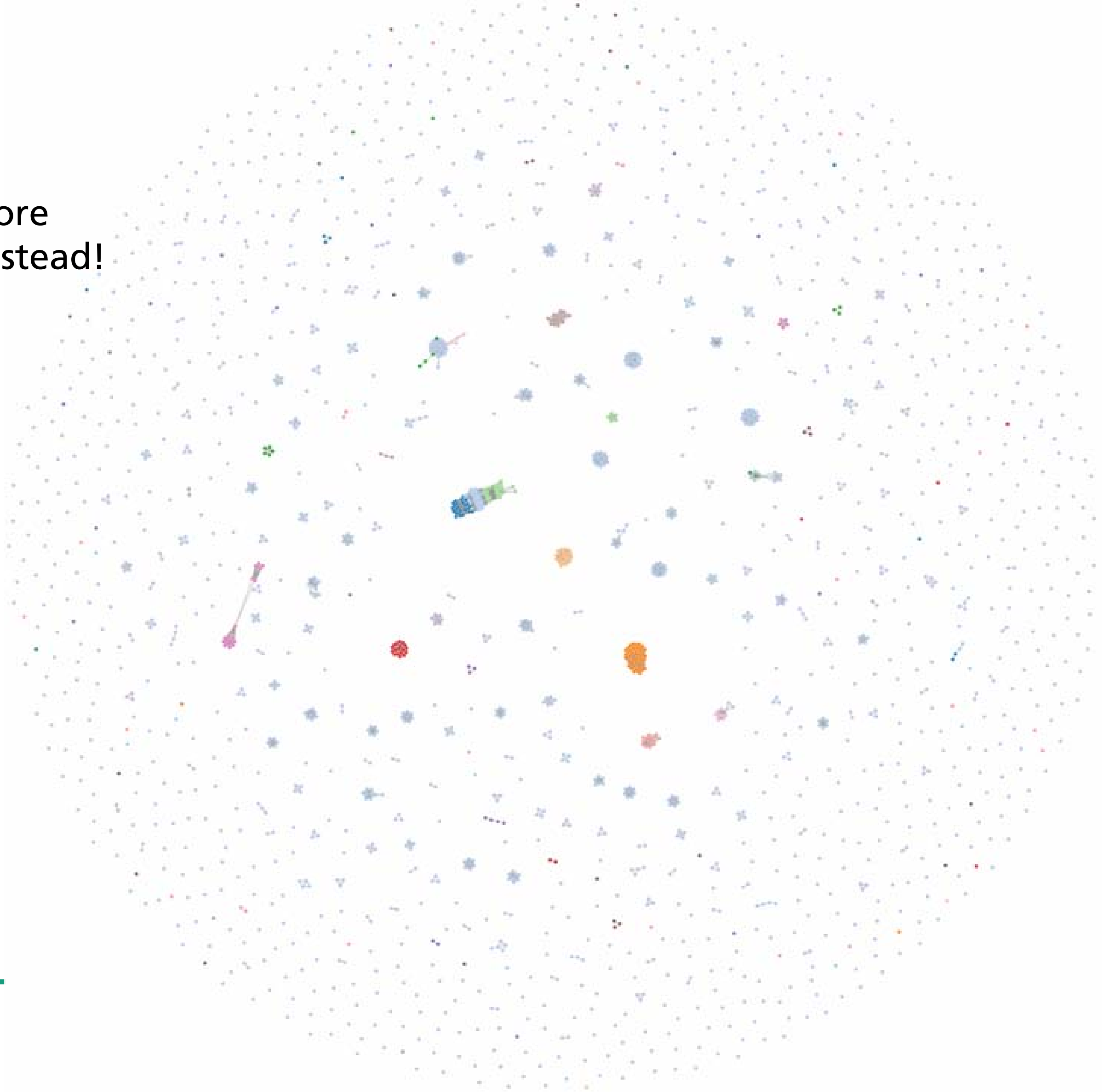
- Let's filter out all match clusters with more than 10 families!!
  And let's try a threshold of… 0.2!



Code-based Similarity Analysis

MCRIT Results: Filtered Results / Samples

- Let‘s filter out all match clusters with more than 10 families but now use samples instead!
- „Most“ samples already cluster nicely into their families



Code-based Similarity Analysis
MCRIT

- Next steps
  - Improve matching quality
    - Turns out, this is actually not easy. :D
    - Tweak / verify against multiple ground truth data sets
    - Recognize and filter out known goodware/libraries
  - Make it usable
    - Deployable framework with some kind of (REST) API
    - Integrations with other analysis tools?
  - Extensive evaluation on Malpedia data set

- Hosted service along Malpedia?

Summary

Fraunhofer
FKIE

Summary
Code Cartographer‘s Diary

- **The Malpedia Vision: A curated, free, high-quality malware corpus for research**
- Want Access?
  - Talk to me (Know Met Trust (KMT) -> ensures K&M already)
  - Get an invite by another existing member that can vouch for you
  - Procedure can be potentially accelerated based on your background (GOV/LEA, …)

- Windows API Usage Recovery & Analysis
  - ApiScout: Convenient & reliable WinAPI usage recovery from memory dumps
  - ApiVectors: Compact representation, decent matching performance

- Code-based Similarity Analysis
  - SMDA: Recursive disassembler (FOSS) optimized for memory dumps
  - MCRIT: Scalable code-based similarity analysis has huge potential

Thank You for Your Attention!

Daniel Plohmann
daniel.plohmann@fkie.fraunhofer.de

@push_pnx
@malpedia

Fraunhofer FKIE