Code Cartographer's Diary

malpedia

@push_pnx

2018-10-02 | One Conference, Den Haag

Daniel Plohmann
daniel.plohmann@fkie.fraunhofer.de

Steffen Enders
steffen.enders@tu-dortmund.de

Martin Clauß
martin.clauss@fkie.fraunhofer.de

Paul Hordiienko
pavlo.hordiienko@fkie.fraunhofer.de

Elmar Padilla
elmar.padilla@fkie.fraunhofer.de

Fraunhofer FKIE

The

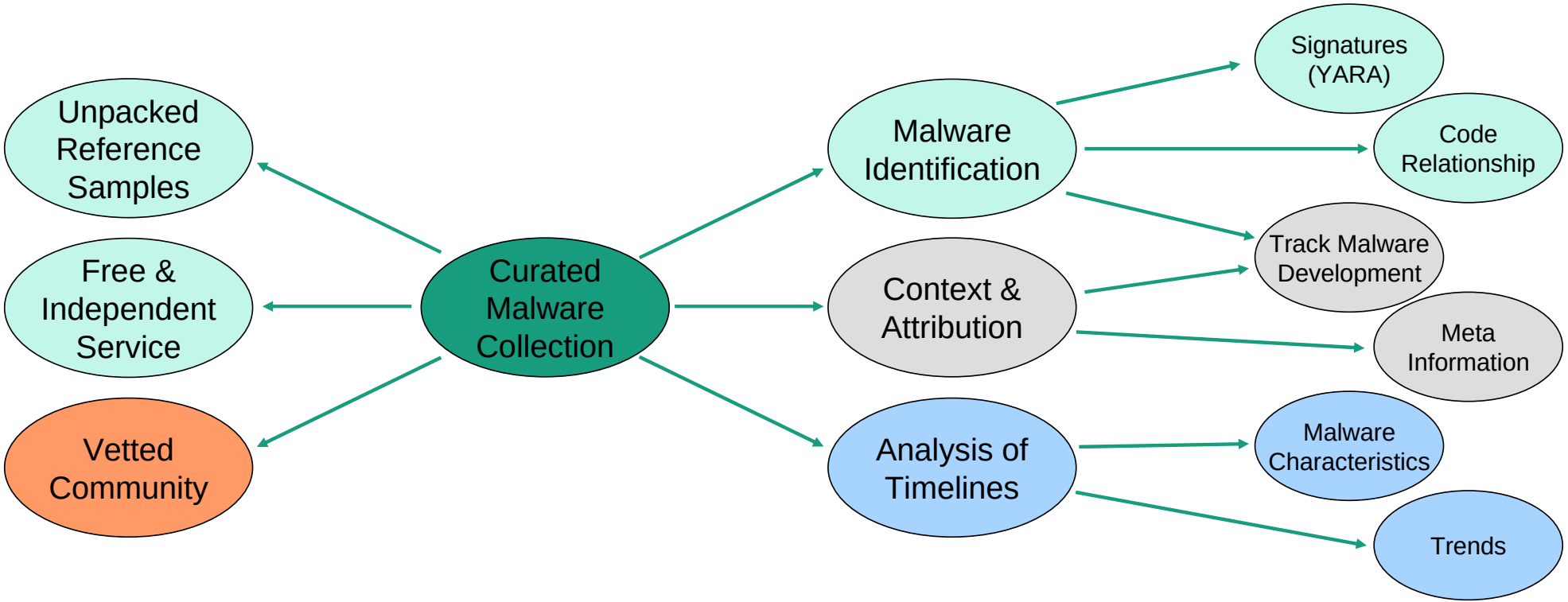
Agenda

Agenda

- Overview: Malpedia
  - Motivation & Overview
  - Operationalizing Malpedia
- Windows API Usage Recovery & Analysis for Malware Characterization
  - Tools: ApiScout / ApiVectors
  - Evaluation Results
- Code-based Similarity Analysis
  - Tools: SMDA & MCRIT
  - Evaluation Outlook
- Summary

Overview

What is Malpedia?



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

Fraunhofer FKIE

Overview
Statistics

- Launched @ Botconf 12/2017 [3]
  - ~800 users, 150 monthly active
- 2000+ user contributions since then
  - THX to „you know who you are“!! :)

[1] https://malpedia.caad.fkie.fraunhofer.de
[2] https://malpedia.io
[3] https://www.botconf.eu/wp-content/uploads/2017/12/2017-DanielPlohmann-Malpedia.pdf

Overview

What does it look like?

Alternatively:

Access through **REST API**
or
git clone malpedia
to retrieve all data for offline use at once

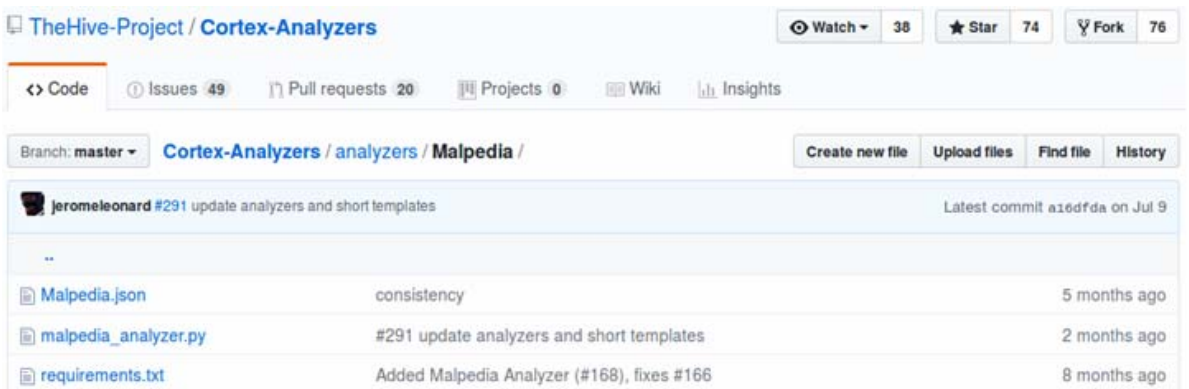
[1] https://malpedia.caad.fkie.fraunhofer.de
[2] https://malpedia.io

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

Fraunhofer FKIE

Malpedia Usage Example:

Windows API Usage Recovery & Analysis for Malware Characterization

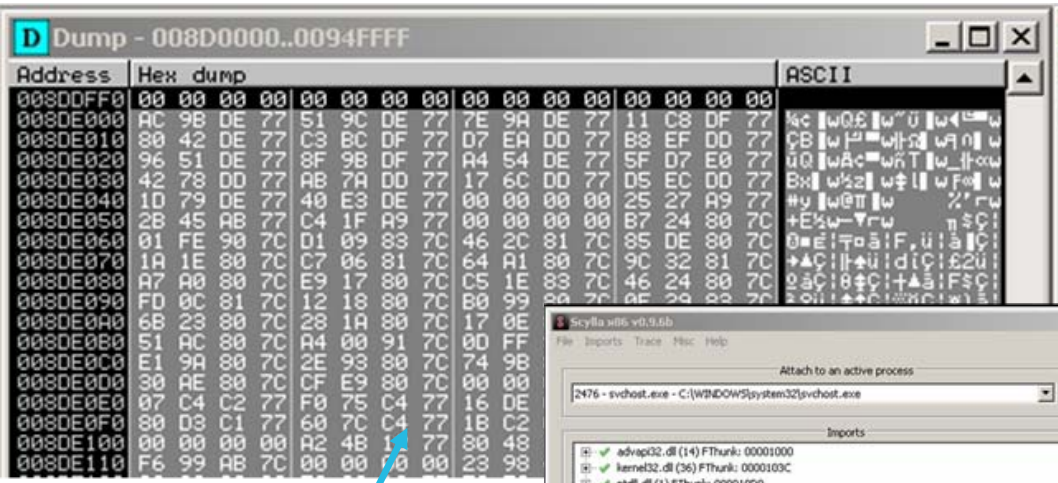
Windows API Usage Recovery

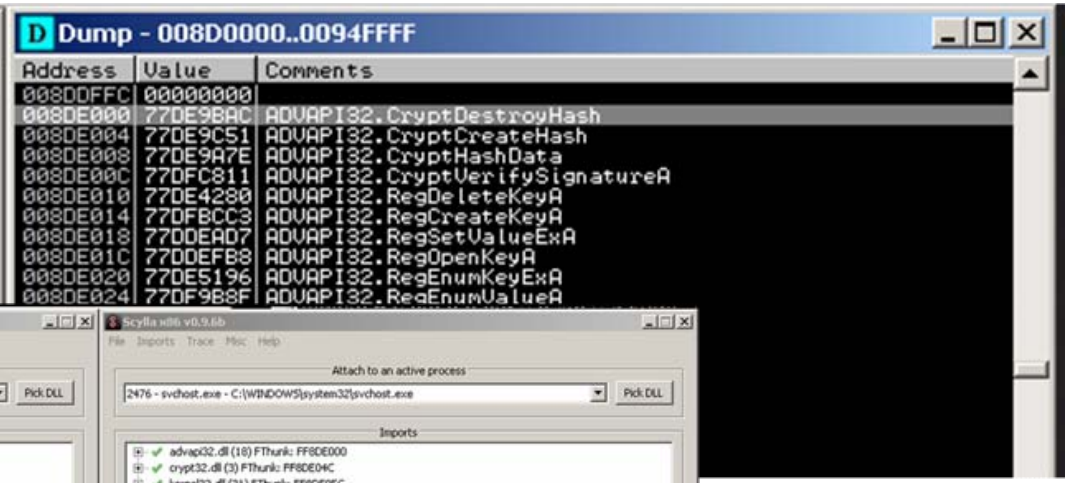
Motivation

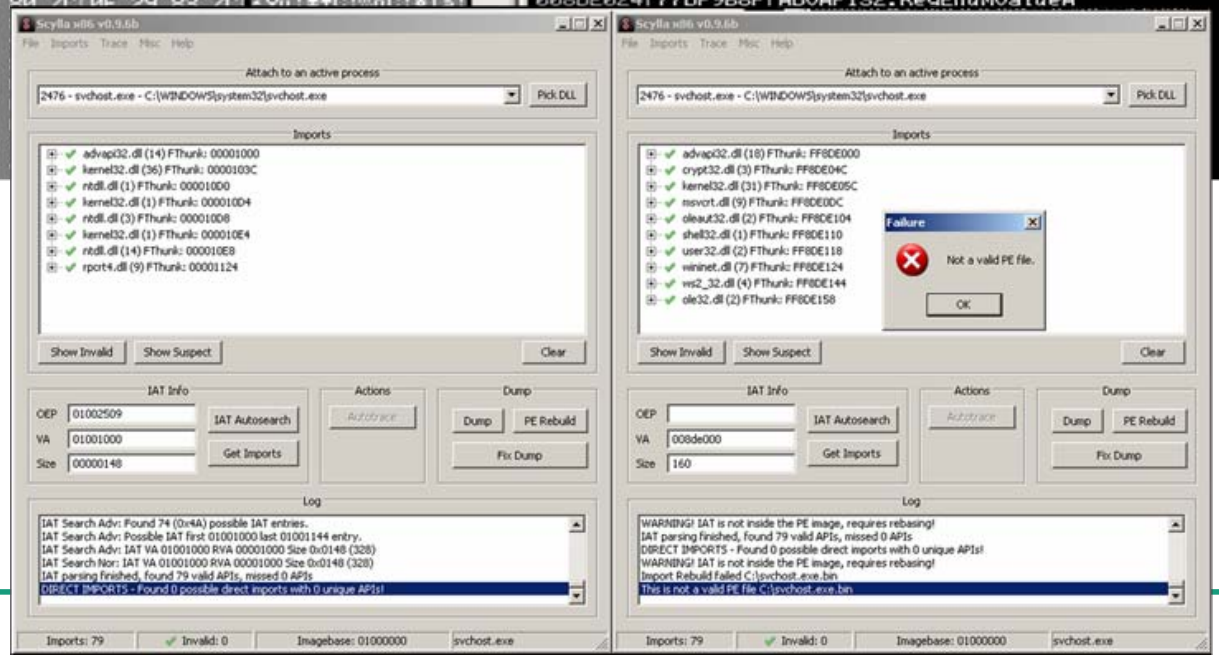
„(Windows) API interactions are an essential cornerstone for effective reverse engineering“

Windows API Usage Recovery

Current „State of the Art“







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

Fraunhofer FKIE

Windows API Usage Recovery

Overview

- Tool: ApiScout [1]
  - *Originally presented 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

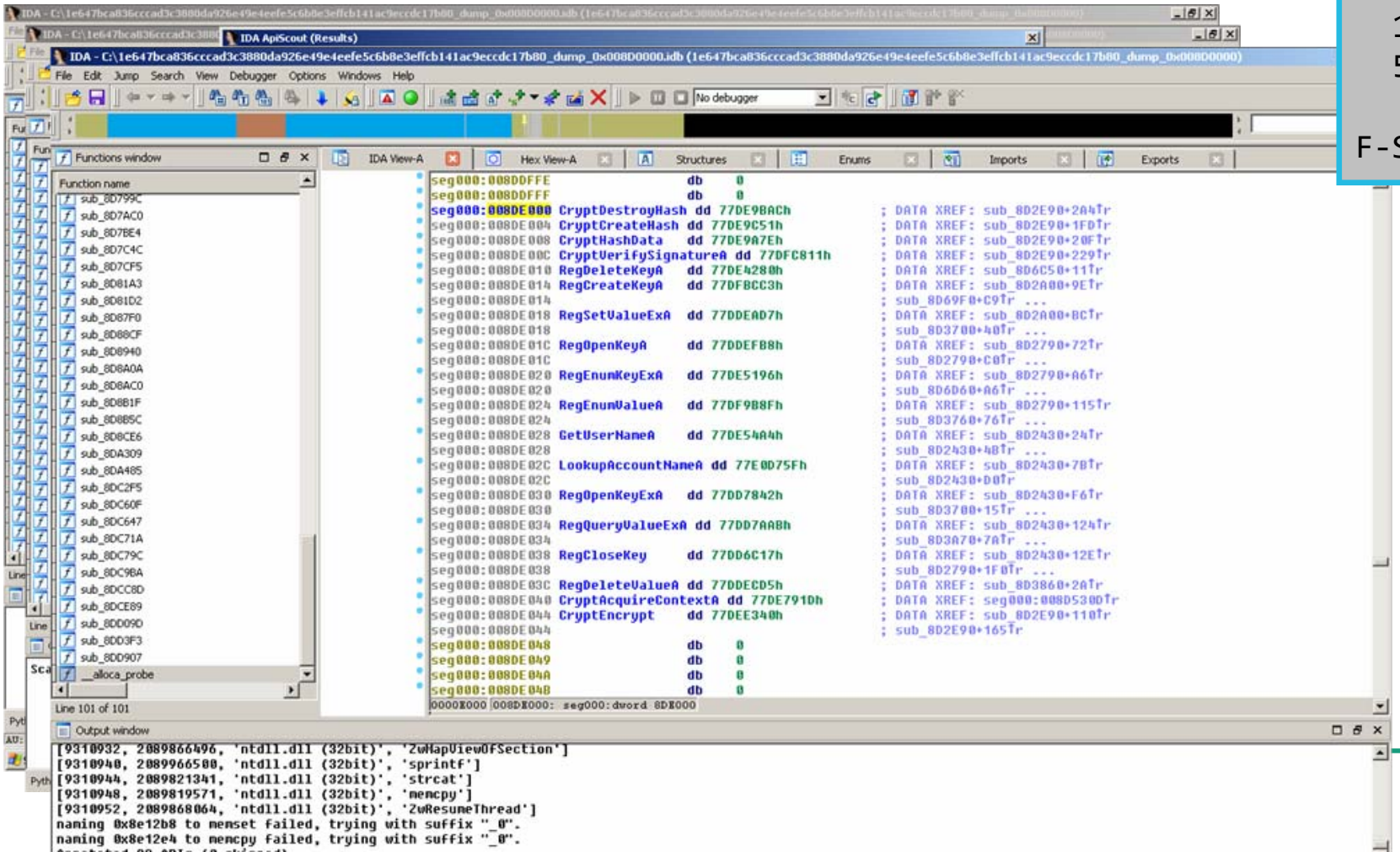
Windows API Usage Recovery

ApiScout Methodology

Accuracy Evaluation:

Sample:
  15 Windows Bins
  5,367 API Imports

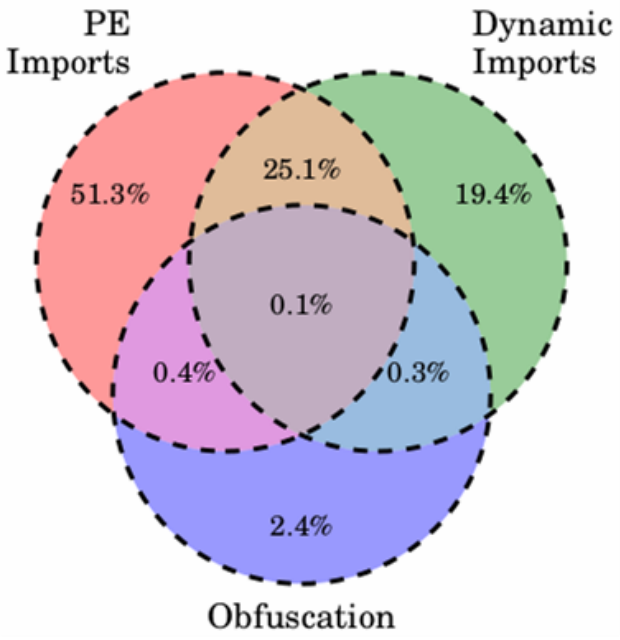
F-Score: 0.995



Fraunhofer FKIE

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
- 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)*

Covered by ApiScout [1]

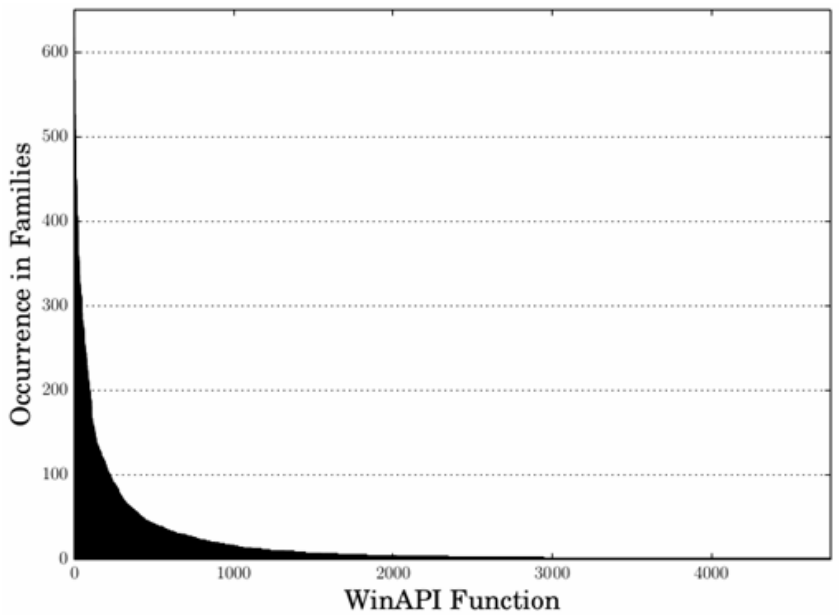
[1] https://github.com/danielplohmann/apiscout

Fraunhofer FKIE

Windows API Usage Recovery

Occurrence Frequency of Individual WinAPI Functions

- Occurrence frequency per Windows API function



- Discounting .NET and API-obfuscated families
  - Only 3 API functions > 90% (CloseHandle, Sleep, WriteFile)
  - Only 48 API functions > 50%
  - API function at position 150 appears in 21.73%
  - 4,392 (92.52%) of API functions <= 10%
- API compositions are highly specific per family
  - 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)

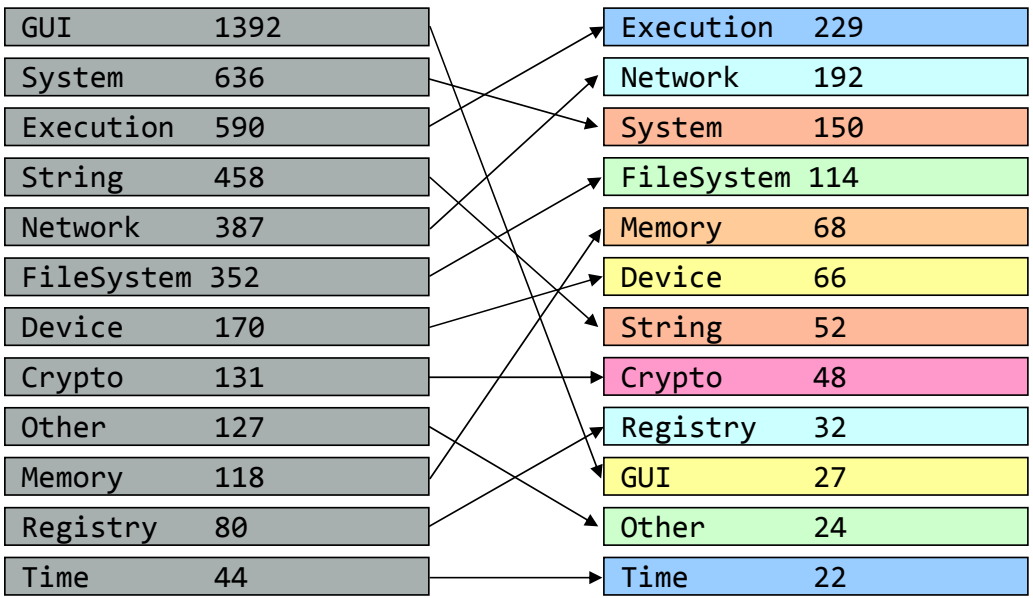
- Define: API Context Groups
  - Manually labelled ~4.500 APIs, primary (12) and secondary class (115)

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

WinAPI Reference Vector

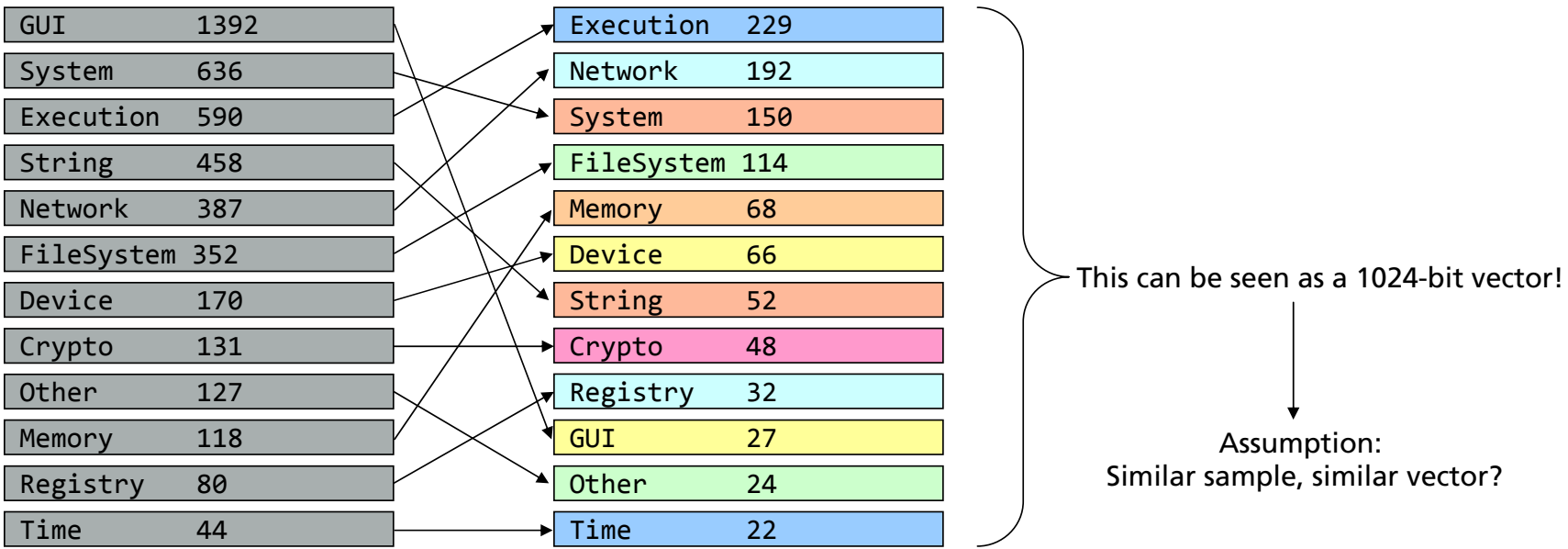
- Define: API Context Groups
  - Reduce this set to 1024 WinAPIs (~80% hierarchy, ~20% based on domain knowledge)



Windows API Usage Recovery

WinAPI Reference Vector

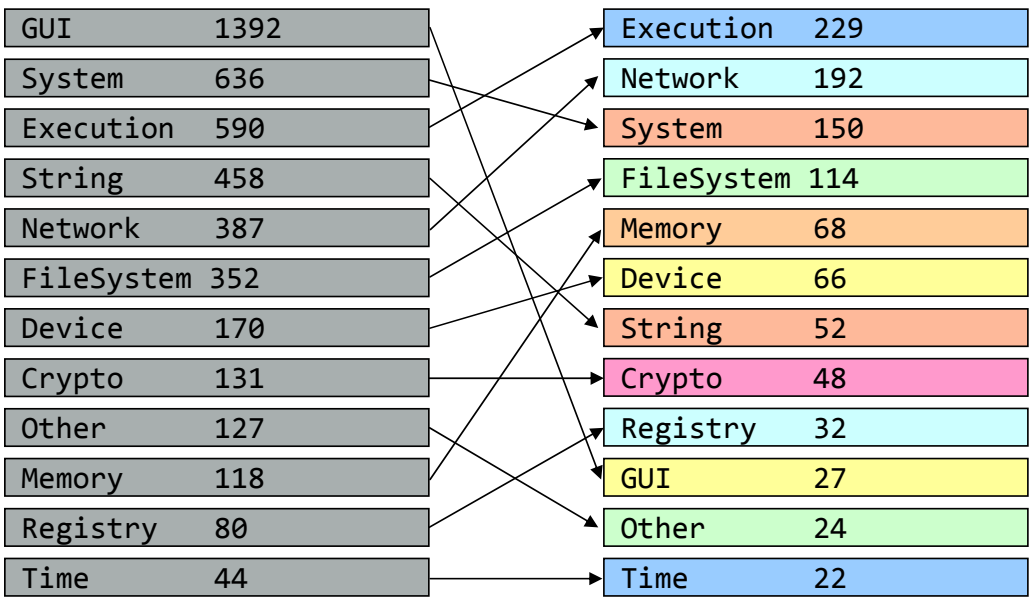
- Define: API Context Groups
  - Reduce this set to 1024 WinAPIs (~80% hierarchy, ~20% based on domain knowledge)
  - Vector yields 90% coverage (mean) for APIs found by ApiScout for ~600 malware families

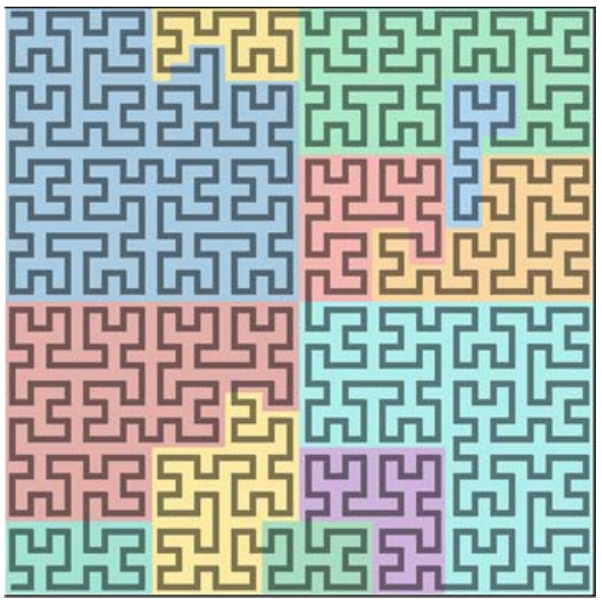


Fraunhofer FKIE

ApiScout
WinAPI Reference Vector

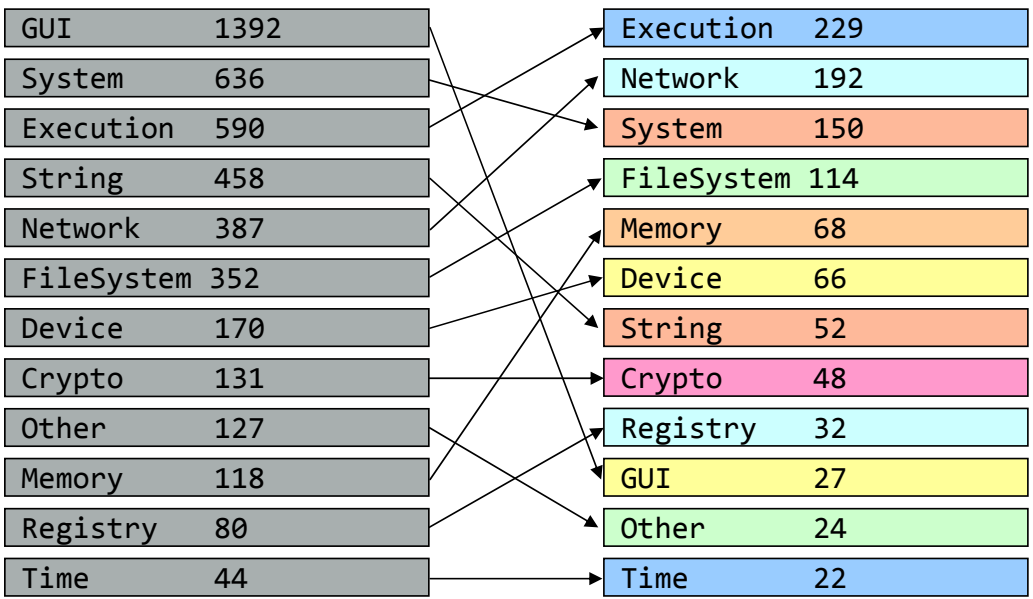
- Visualize Vectors:
  - Hilbert Curve to ensure neighboring of contexts

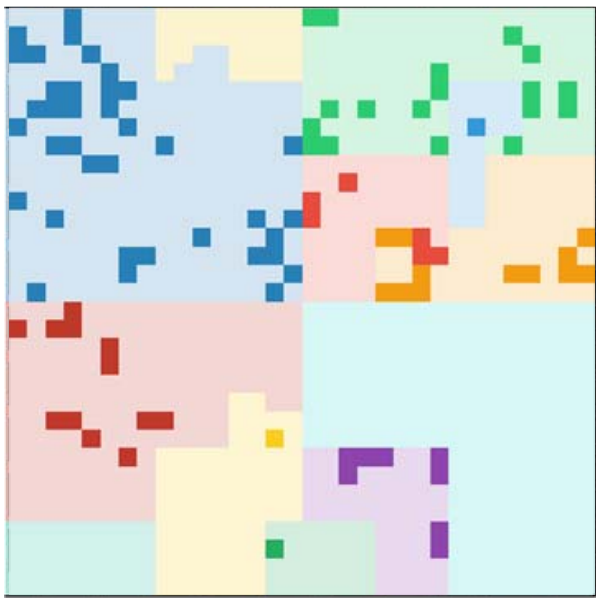




ApiScout
WinAPI Reference Vector

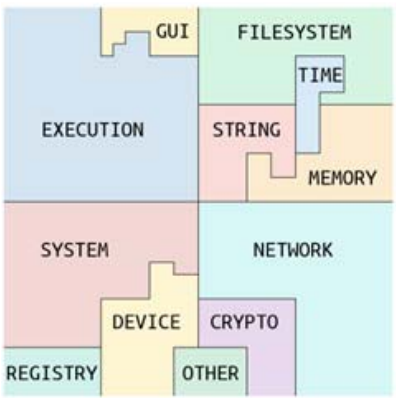
- Visualize Vectors:
  - Hilbert Curve to ensure neighboring of contexts



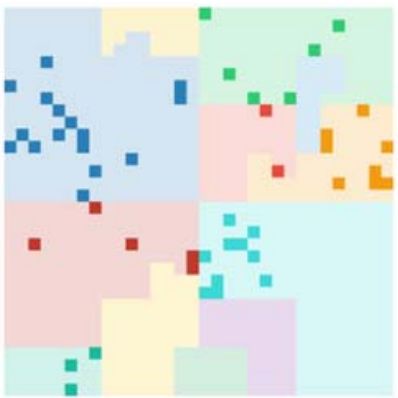


ApiScout
WinAPI Reference Vector

- Some Examples



(a) Semantic Categories


(b) DELoader


(c) Zeus

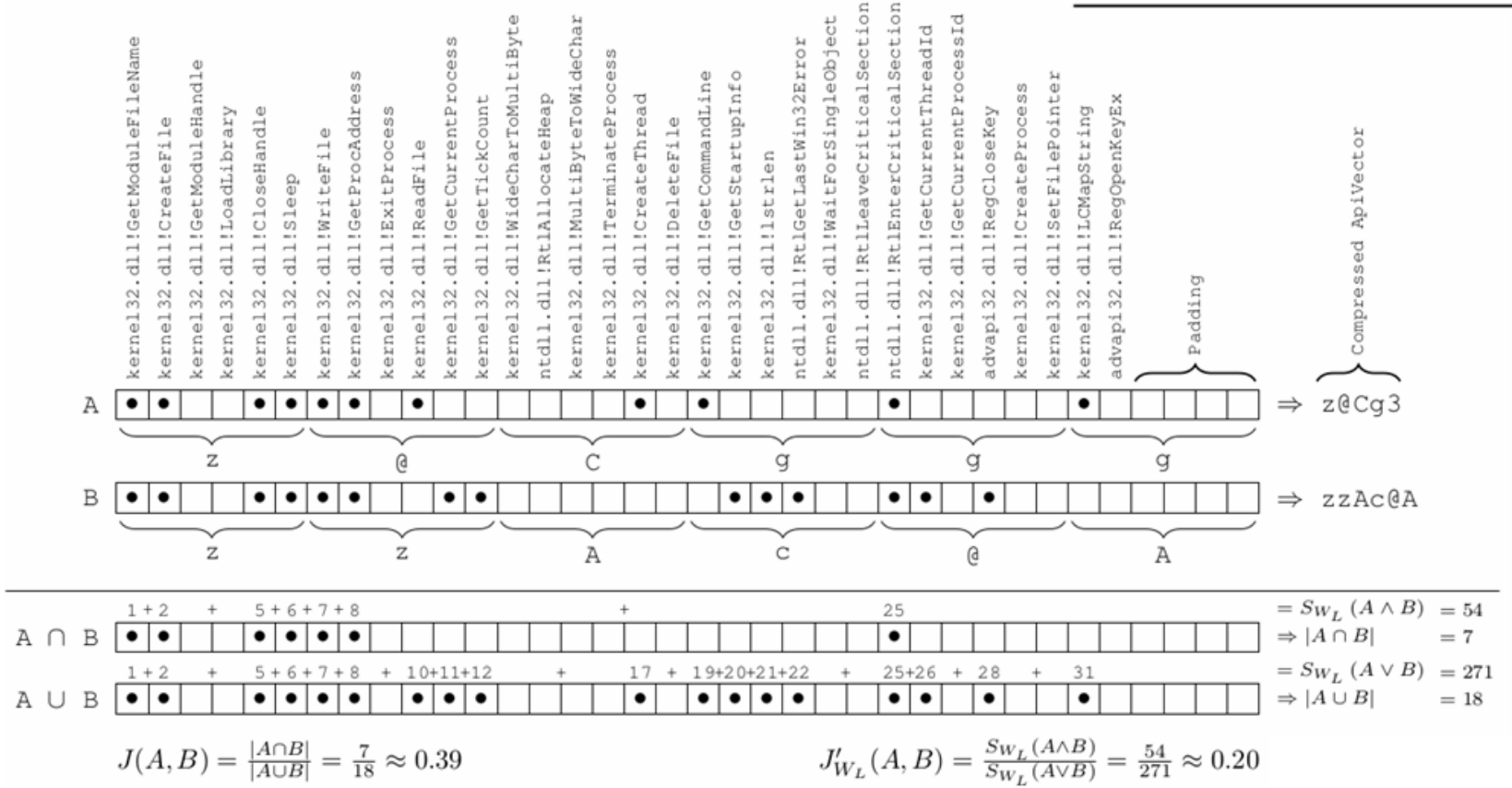

(d) Citadel


(e) DarkComet

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	_



Fraunhofer FKIE

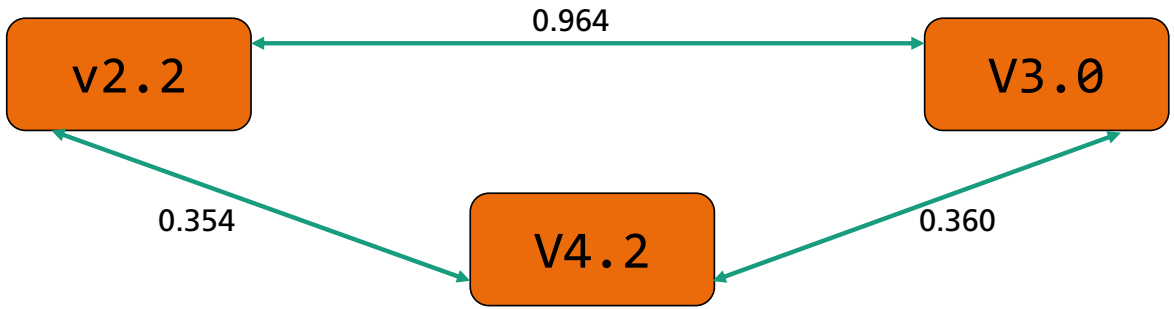
Windows API Usage Analysis

Comparison of ApiVectors

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

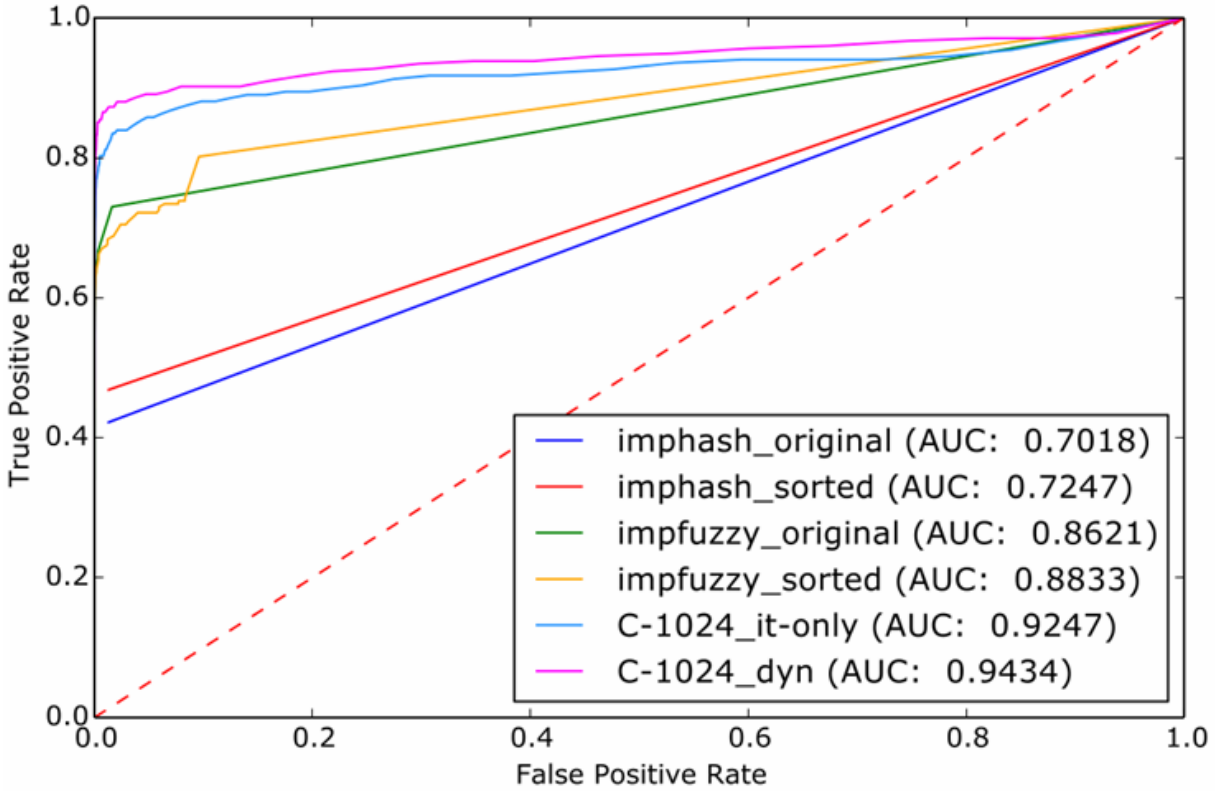
```
A42gA28KA13    CAAMA16BABAAJAECAxMAACkAAQUA7CJBCgAgUBA3 kQCBAHJSRjU^q-*}_pb__N,_^?
A42gA28KA13    CAAMA16BABAAJAEAAxMAACkAAQUA7CJBCgAAUBA3 kQCBAHJSRjU^q-*}_pL__N,._^?
A41BA29CA4IA9gCA9gA8Q  BAAJAEAABMA3 gAAQA8 QJRCgAgUBAAHkQARCDIADDBGAqQAgCcGOIOp,f?
```

TeslaCrypt 2.2, 3.0, 4.2



Windows API Usage Analysis

Evaluation of Matching Performance

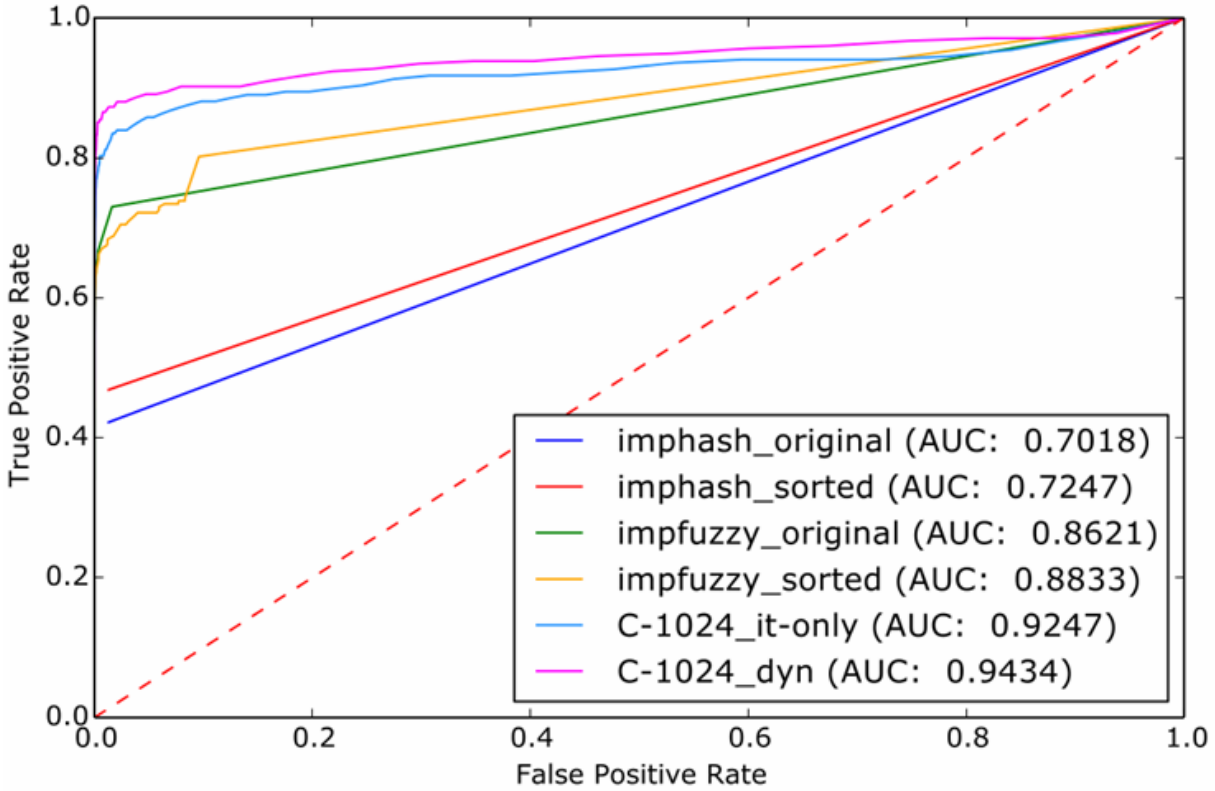


- Data set: Malpedia (2018-05-17)
  - 673 families, 1854 samples
- Comparison with ImpHash, ImpFuzzy
  - Mean Fingerprint sizes:
    - ImpHash: 32 bytes
    - ImpFuzzy: 54.4 bytes
    - ApiVector: 74.3 bytes
  - ApiVector: recoverable info
- Performance @ Thresholds
  - T: 0.18 – 90.18% TPR, 9.45% FPR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

Fraunhofer FKIE

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 to be presented @ MISP Summit / hack.lu [4]
  - Malpedia :)
- Ideally used as post-processing for sandboxing
  - Cuckoo plugin?

[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/

Malpedia Usage Example:

Code-based Similarity Analysis

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]
  - Work in progress*, already silently released on GitHub [2]*
  - Built on top of Capstone [1]
  - „SMDA is a minimalist recursive disassembler library that is optimized for accurate Control Flow Graph (CFG) recovery from memory dumps."
- Tool: MCRIT
  - „MinHash-based Code Relationship Identification Toolkit"
  - Work in progress, will be released later this year

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

Code-based Similarity Analysis

SMDA

- Malpedia uses memory dumps as normalization
  - Available tools (IDA, Binary Ninja, radare2, …) not suited / optimized for this

```
/home/pnx/smda $ python3 analyze.py citadel_dump_0x00140000
                 -o citadel_disasm.json

      now analyzing /home/pnx/citadel_dump_0x02390000
      2018-09-24 11:30:57,299: smda.common.ApiResolver
        - loaded 57315 exports from 134 DLLs (Windows XP Professional).

->  3.86s |   873 Func (status: ok)

/home/pnx/smda $ cat citadel_disasm.json
```


```
{
 "architecture": "intel",
 "base_addr": 1310720,
 "bitness": 32,
 "execution_time": 3.86346,
 "filename": "citadel_dump_0x00140000",
 "message": "Analysis finished regularly.",
 "sha256": "2c4166c81c31fd5de7fe205ea12c74d8bb394005337fa844afc23e4707a3d42f",
 "status": "ok",
 "summary": {
  "num_api_calls": 1146,
  "num_basic_blocks": 10686,
  "num_disassembly_errors": 56,
  "num_function_calls": 3979,
  "num_functions": 873,
  "num_instructions": 53324,
  "num_leaf_functions": 114,
  "num_recursive_functions": 7
 },
 "timestamp": "2018-09-24T09-31-01",
 "version": "1.0.1",
 "xcfg": {
  "1318184": {
   "apirefs": {},
   "blockrefs": {},
   "blocks": {
    "1318184": [
     [
      1318184,
      "55",
      "push",
      "ebp"
     ],
[...]
```


- Example results:

	Method	FNC	BB	INS	TPR
■	Manual Reference:	793	10,264	52,121	99.99%
■	SMDA vs. dump:	873	10,686	53,324	98.36%
■	IDA vs. dump:	658	9,071	46,266	85.62%
■	*IDA vs. clean unpacked:*	*788*	*10,209*	*51,794*	*99.37%*

Fraunhofer FKIE

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

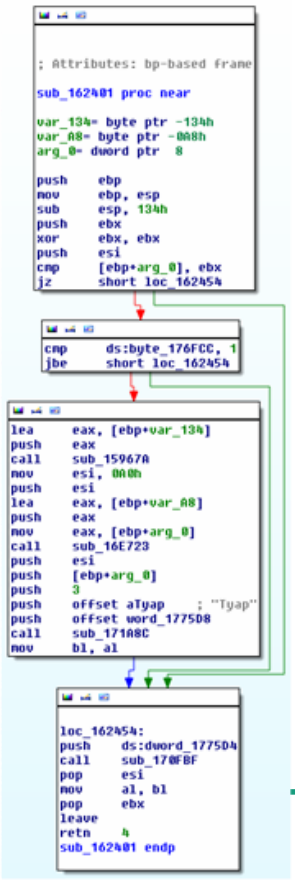
MinHash 101

- MinHash procedure:
  - Extract a range of descriptive features („shingles“) for each object
  - Hash them n times (with each hash function seeded differently)
  - Select the minimum hash value for each of the n groups
  - The resulting sequence of n values is considered as the function‘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

Statistics:
- „num_ins:33“
- „num_blocks:4“
- „num_calls:4“
- …

Mnemonic N-grams:
- „push-mov-sub-push“
- „mov-sub-push-xor“,
- „sub-push-xor-push“,
- …

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

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

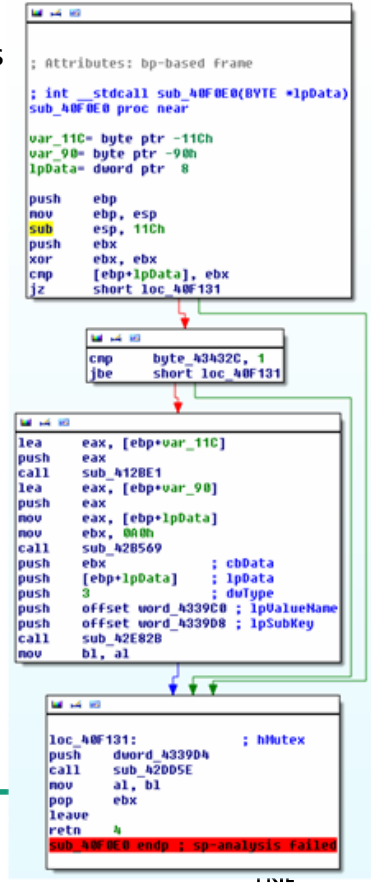
Fingerprint B : [77, 99, 82, 73, 84, 45, 121, 97, 112, 33]

Same shingles for all functions

Statistics:
- „num_ins:30“
- „num_blocks:4“
- „num_calls:4“
- …

Mnemonic N-grams:
- „push-mov-sub-push“
- „mov-sub-push-xor“,
- „sub-push-xor-cmp“
- …

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 takes ~2min on this laptop (i5, 8GB RAM).

- Comparison: BinDiff
  - All samples vs. each other (50*49/2 = 1,225 pairs)
  - Runtime: ~60min

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 (banding)
  - This results in 29,596,574 matches above threshold (0.7)

- 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
  - Basically, on this laptop.

Code-based Similarity Analysis

MCRIT

- Next steps
  - Improve matching quality
    - Tweak / verify against multiple ground truth data sets
    - Goodware / libraries with different compilers available
  - Make it usable
    - REST API
    - integrations with other analysis tools (IDA, Binja, r2, …?)
  - Extensive evaluation on Malpedia data set

- Hosted service along Malpedia?

Summary

Summary
Code Cartographer's Diary

- **The Malpedia Vision: A curated, free, high-quality malware corpus for research**
  - Vetted + community-driven
- 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: Code-based similarity analysis has huge potential

Thank You for Your Attention!

Daniel Plohmann
daniel.plohmann@fkie.fraunhofer.de


@push_pnx
@malpedia

Fraunhofer
FKIE