Fraunhofer FKIE

Fraunhofer-Institut für Kommunikation, Informationsverarbeitung und Ergonomie FKIE

Daniel Plohmann, Manuel Blatt, and Daniel Enders | 2023-04-13

MCRIT: The MinHash-based Code Relationship & Investigation Toolkit

Introduction
$whoami

Fraunhofer FKIE

universitätbonn

- Security Researcher @ Fraunhofer FKIE & University of Bonn
- Research Scope:
  - Analysis of malicious software (malware) / reverse engineering / analysis automation
- Past Appearences at Botconf:
  - Laura Guevara, Daniel Plohmann
    2014 - Semantic Exploration of Binaries
  - Daniel Plohmann
    2015 - DGArchive: A deep dive into domain generating algorithms
  - Daniel Plohmann, Martin Clauß, Steffen Enders, Elmar Padilla
    2017 - Malpedia: A Collaborative Effort to Inventorize the Malware Landscape
  - Daniel Plohmann, Steffen Enders, Elmar Padilla
    2018 - Code Cartographer's Diary
  - Felix Bilstein, Daniel Plohmann
    2019 - YARA-Signator: Automated Generation of Code-based YARA Rules

Outline

- Motivation
- MCRIT: System Overview
  - Methodology
  - Framework
- Use Cases
- Outlook

Motivation

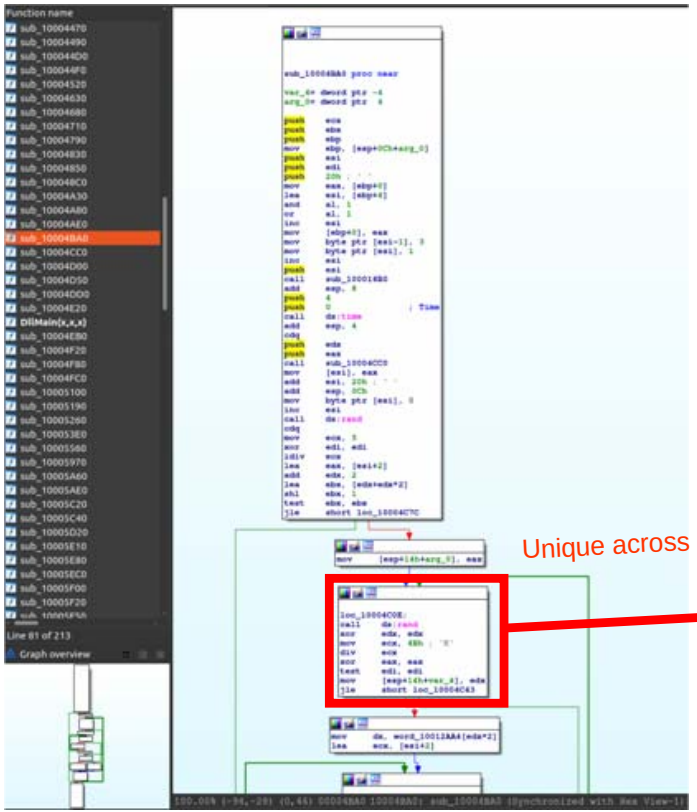
Motivation

Anzeige im Hauptbahnhof in Chemnitz am Freitag: Forderung von Lösegeld Foto: P. Götzelt/ dpa

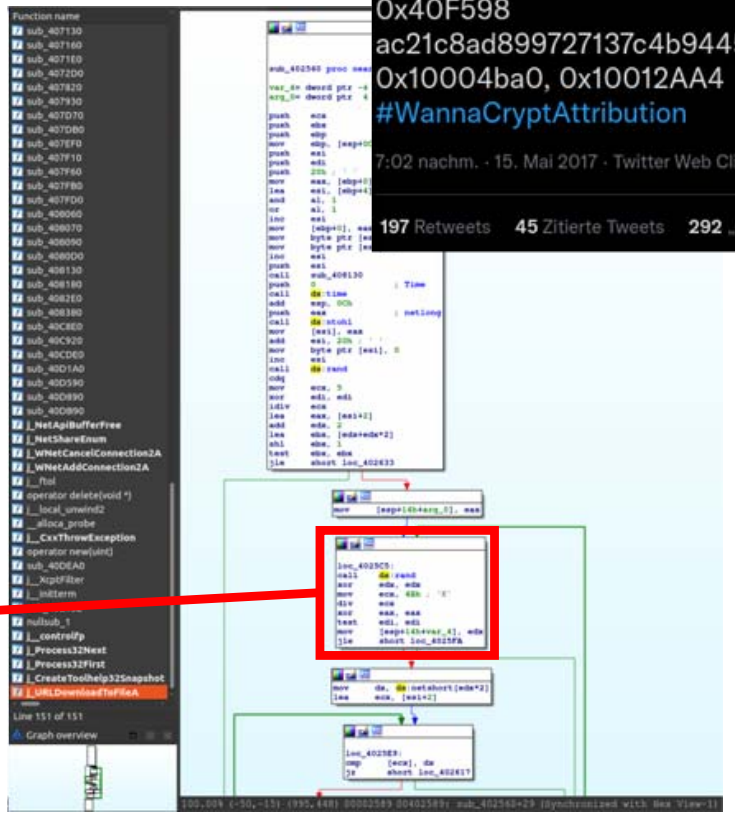
- Infamous WannaCry
- Ransomware attack using wormable exploit (EternalBlue)
  - Attack started on May 12th 2017
  - 230k affected systems in ~8 hours
  - Quickly disrupted due to a lucky registration of killswitch domain
- Impact
  - UK NHS disrupted (£100m damage)
  - Nissan, Renault, Telefonica, FedEx, DB, …
- Attack Attribution?

[1] https://en.wikipedia.org/wiki/WannaCry_ransomware_attack
[2] https://twitter.com/neelmehta/status/864164081116225536

Motivation



Unique across all of Malpedia



[1] https://twitter.com/neelmehta/status/864164081116225536

Fraunhofer FKIE

Motivation

- Code Similarity Analysis
  - High potential to help analysts and accelerate analysis
    - Code identification, library filtering, hunting, label transfer, …
  - Existing solutions mostly limited to
    - 1:1 comparison
    - Proprietary
- Let’s see what we can do. :)

MCRIT

System Overview

MCRIT
Design

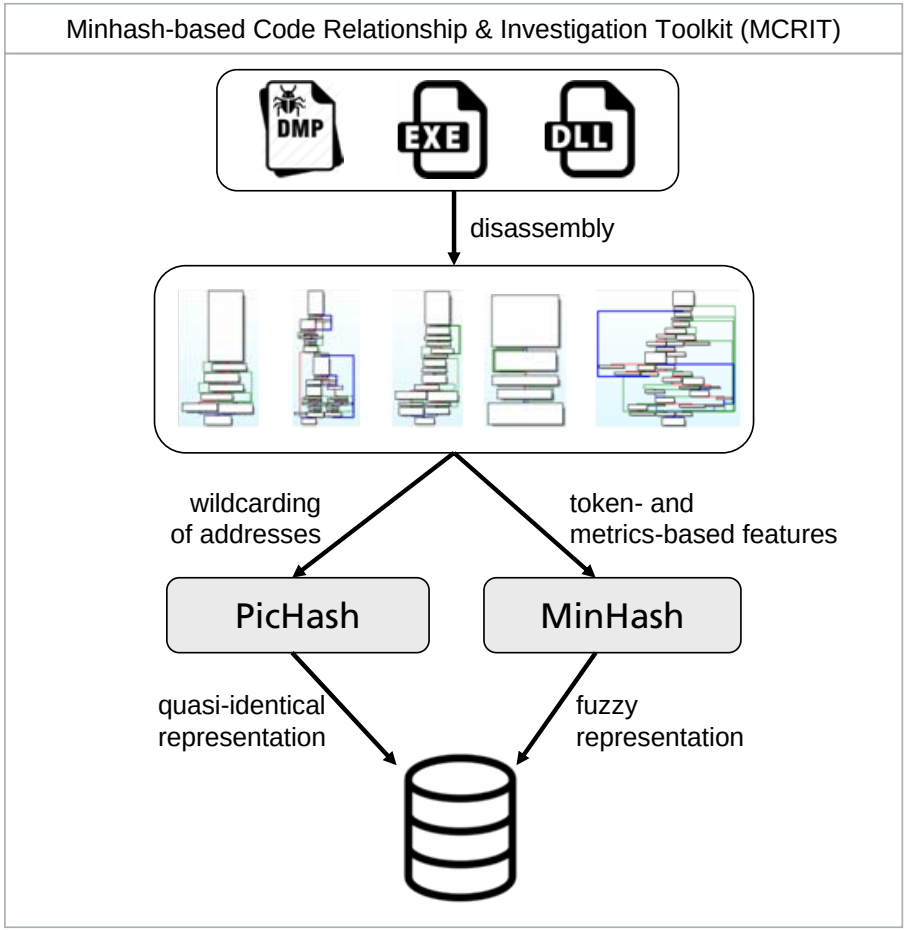
- Goal: Analyze code sharing and third-party library usage in malware
  - Create tools to leverage Malpedia binary corpus
  - Don't reinvent the wheel: reuse of proven techniques as described in literature

- Requirements:
  - Similarity: reliable, interpretable estimate
  - Scalability: (tens of) millions of functions
  - Efficient representation: (significantly) smaller than indexed code
  - No cross-bitness or cross-architecture (*Malpedia at the time was 95% 32bit Intel*)

Code Recovery and Similarity Analysis

MCRIT: Approach

- Initial Observations
  - Haq et al. [1] survey:
    50+ works on code similarity since 2010
  - Only 17 with 10+ malware samples
  - Only 1 analyzed FOSS usage in malware (Alrabaee et al. [2])
- MCRIT
  - Combines quasi-identical and fuzzy code representation
  - Block & Function-level similarity
  - Efficient 1:n matching via
    - Hashmaps
    - Locality-Sensitive Hashing (LSH)

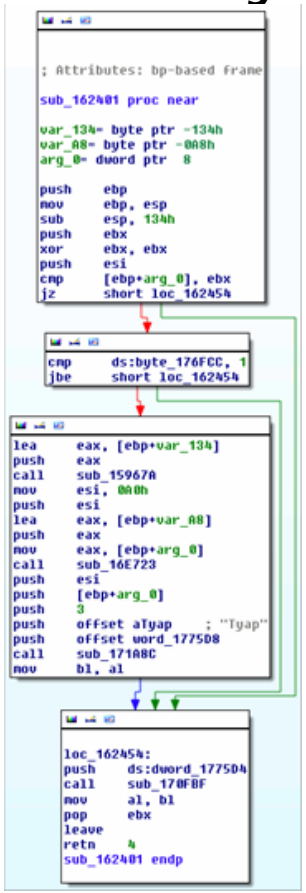


Project Co-Authors: Paul Hordiienko, Steffen Enders, Manuel Blatt, Daniel Enders

[1] I. U. Haq and J. Caballero, "A survey of binary code similarity," In: Arxiv.org Computers & Security, 2019.
[2] S. Alrabaee, P. Shirani, L. Wang, and M. Debbabi, "FOSSIL: A Resilient and Efficient System for Identifying FOSS Functions in Malware Binaries," In: ACM Trans. Priv. Secur., vol. 21, 2018.

MCRIT

PIC Hashing



- Quasi-Identical: Position Independent Code (PIC) Hashing
  - On **function level** (original method as introduced by Cohen and Havrilla [1])
  - On basic block level (more granularity, almost the same speed)

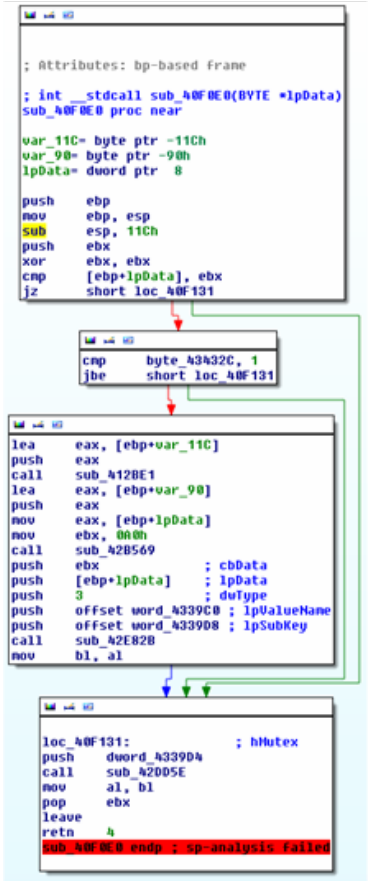
pichash(„55 8BEC 81EC**34**010000 …“)

-> **8806641384121875405**

pichash(„55 8BEC 81EC**1C**010000 …“)

-> **10270976525648996728**

- Good for recognizing statically linked code (often binary identical)

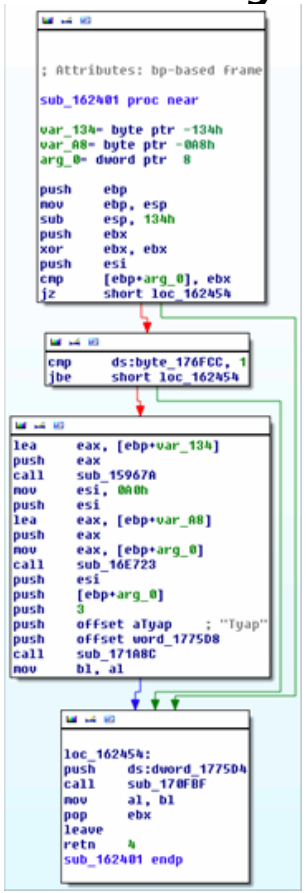


[1] C. Cohen and J. Havrilla, “Function Hashing for Malicious Code Analysis“, tech. rep., SEI, CMU, 2009.

Fraunhofer FKIE

MCRIT

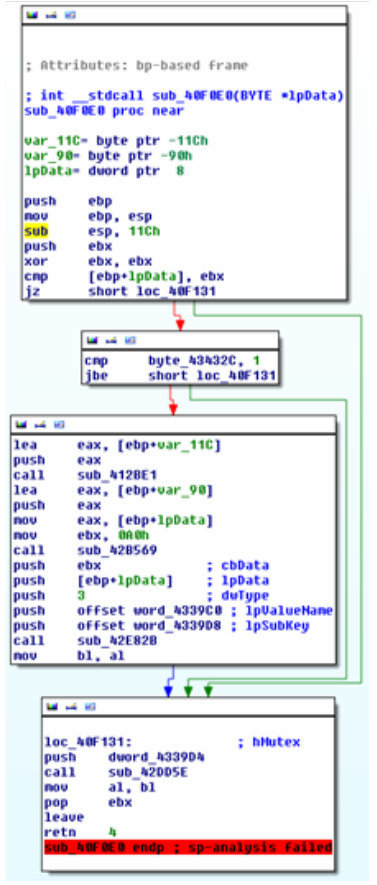
PIC Hashing



- Quasi-Identical: Position Independent Code (PIC) Hashing
  - On function level (original method as introduced by Cohen and Havrilla [1])
  - On **basic block level, size 4+** (more granularity, almost the same speed)

`pichash(„55 8BEC 81EC34010000 …“)`	`pichash(„55 8BEC 81EC1C010000 …“)`
`-> 620962970`	`-> 2827249019`
`pichash(„8D85CCFEFFFF 50 E8?? …“)`	`pichash(„8D85E4FEFFFF 50 E8?? …“)`
`-> 847901973`	`-> 640583737`
`pichash(„FF35???????? E8???? …“)`	`pichash(„FF35???????? E8????  …“)`
`-> 463987246`	`-> 463987246`

- Good for recognizing partial code reuse (candidates)



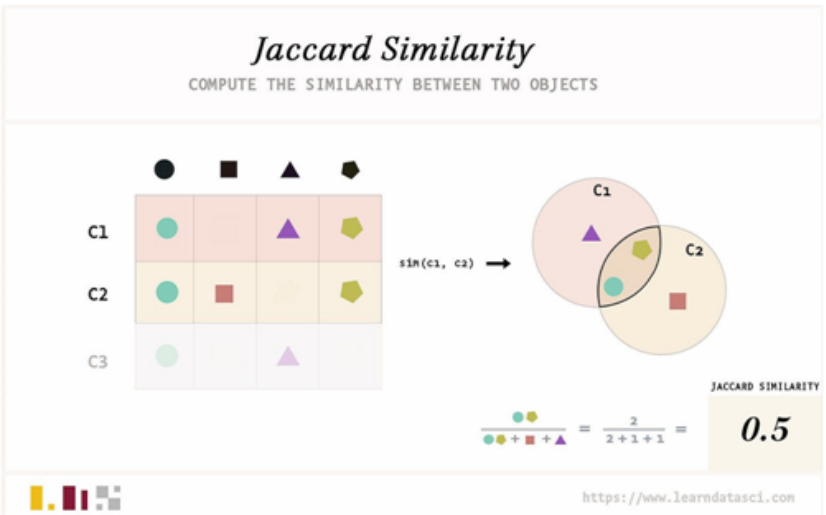
[1] C. Cohen and J. Havrilla, “Function Hashing for Malicious Code Analysis“, tech. rep., SEI, CMU, 2009.

Fraunhofer FKIE

MinHash 101

- MinHashing
  - „Min-wise independent permutations“ - Locality Sensitive Hashing (LSH) scheme [1]
  - Fast estimation of set similarity -> approximation of Jaccard similarity coefficient
  - Scalability: O(log *n*) for single lookups

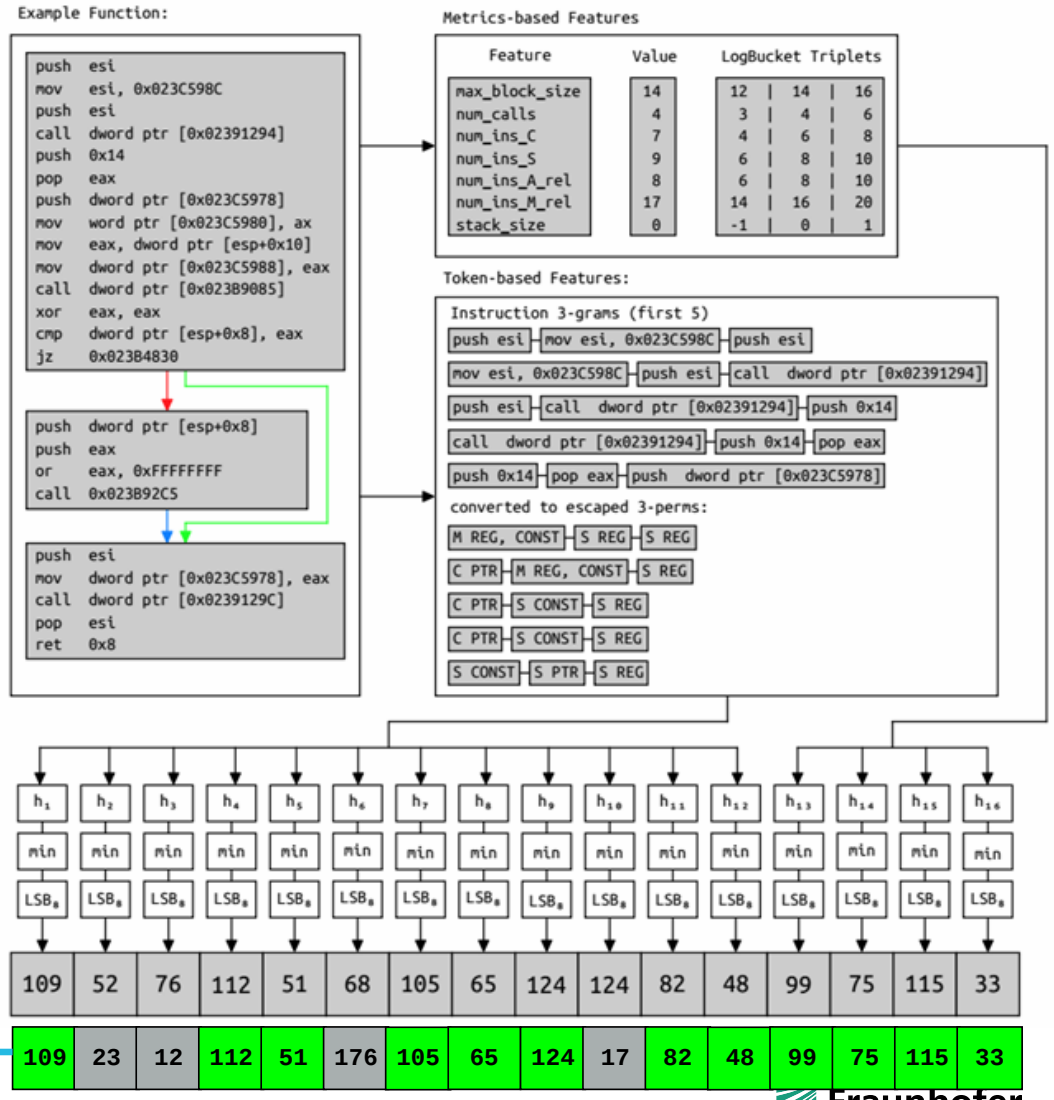
- Use cases:
  - text documents / websites (duplicates, plagiarism)
  - genome sequencing
  - code similarity! [2]



[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.
[3] https://www.learndatasci.com/glossary/jaccard-similarity/

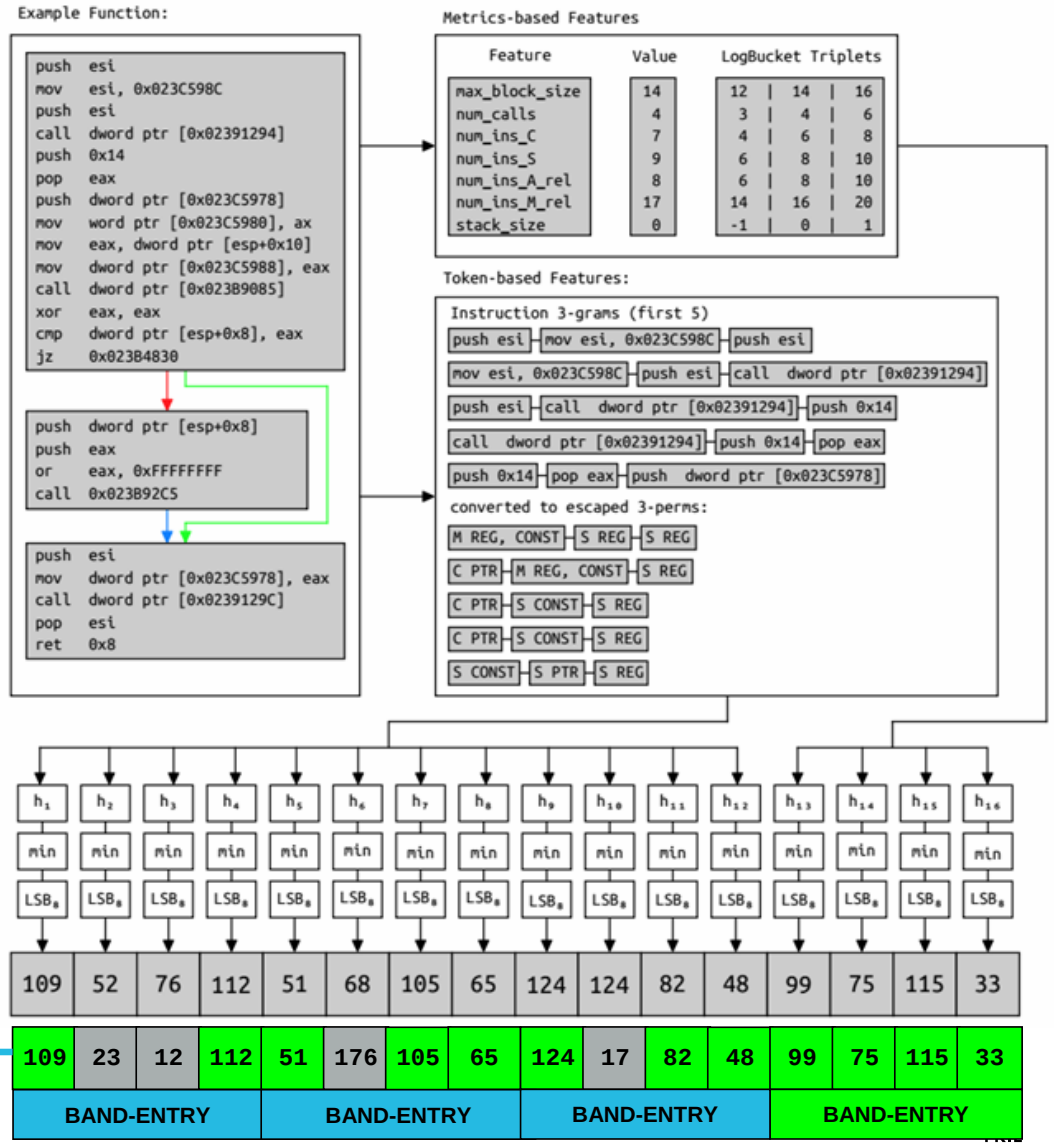
MCRIT
MinHash Composition

- Token-based features:
  - Instruction 3-grams
  - Abstract semantically
  - Convert to 3-perms (sorted)
- Metrics-based features:
  - Numerically describe structure of a function
  - Normalize & Quantize for fuzzy matching
- MinHash matching:
  - Count same values in same position



MCRIT
MinHash Composition

- Token-based features:
  - Instruction 3-grams
  - Abstract semantically
  - Convert to 3-perms (sorted)
- Metrics-based features:
  - Numerically describe structure of a function
  - Normalize & Quantize for fuzzy matching
- Candidate Identification:
  - Additionally index subsequences from signatures into buckets

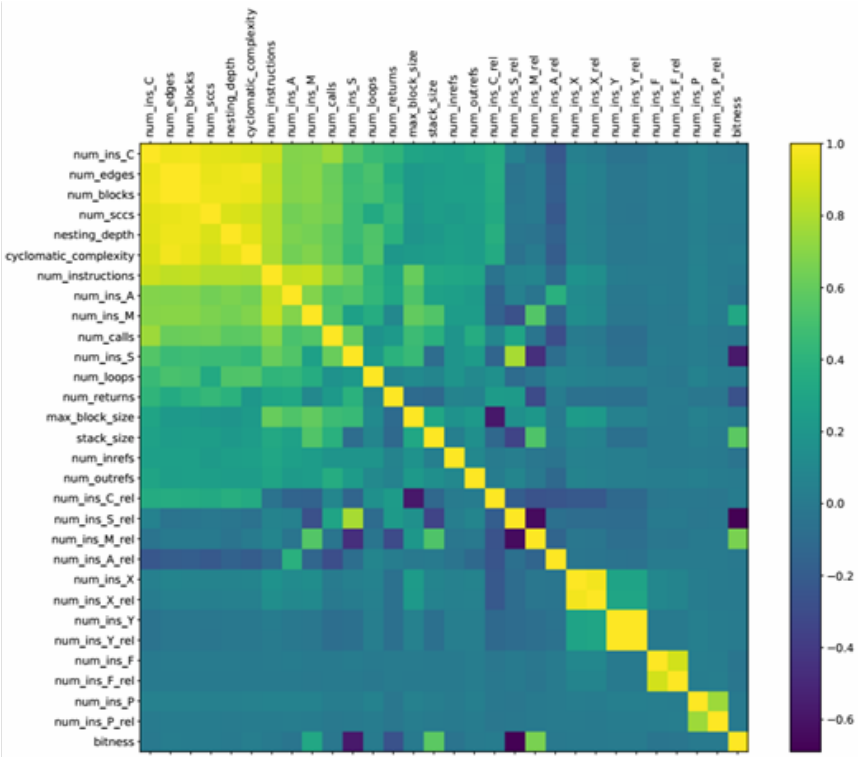


12/16 = 75%

MinHash Feature Engineering

feature-name	space	min	25%	50%	75%	max	avg	sd	ρ_{32bit}	ρ_{64bit}	ρ_{same}	ρ_{diff}
cyclomatic_complexity	232	0	2	4	9	561	8.223	13.767	0.961	0.953	0.958	0.923
max_block_size	554	2	9	12	17	6,669	22.173	100.989	0.946	0.966	0.963	0.805
nesting_depth	65	0	2	4	7	101	4.871	4.777	0.973	0.966	0.970	0.960
num_blocks	339	1	4	9	18	817	15.417	22.605	0.974	0.939	0.954	0.933
num_calls	148	0	1	3	7	380	6.308	10.260	0.980	0.980	0.980	0.952
num_edges	480	0	4	11	25	1,376	21.640	36.096	0.969	0.949	0.955	0.929
num_inrefs	114	0	0	0	1	224	1.018	3.907	0.396	0.861	0.605	0.303
num_ins_A	516	0	3	6	12	3,929	16.244	83.531	0.985	0.963	0.981	0.688
num_ins_C	495	0	7	15	32	1,343	26.784	38.942	0.980	0.971	0.976	0.964
num_ins_F	14	0	0	0	0	30	0.003	0.161	0.972	-	0.972	-0.000
num_ins_M	691	0	6	14	33	2,811	29.725	60.236	0.980	0.970	0.981	0.384
num_ins_P	25	0	0	0	0	40	0.012	0.373	0.606	0.693	0.655	0.471
num_ins_S	301	0	2	6	15	932	13.854	24.591	0.947	0.946	0.965	0.006
num_ins_X	261	0	0	0	0	5,897	2.569	66.481	0.893	0.957	0.934	0.637
num_ins_Y	39	0	0	0	0	280	0.161	4.015	0.967	0.989	0.982	0.929
num_ins_A_rel	83	0	9	13	17	86	14.465	9.469	0.914	0.944	0.954	0.106
num_ins_C_rel	97	0	26	33	40	100	33.017	12.931	0.909	0.964	0.952	0.424
num_ins_F_rel	13	0	0	0	0	27	0.004	0.236	0.972	-	0.972	-0.000
num_ins_M_rel	100	0	18	31	46	99	32.194	16.829	0.935	0.939	0.975	-0.648
num_ins_P_rel	13	0	0	0	0	13	0.007	0.207	0.629	0.716	0.677	0.418
num_ins_S_rel	80	0	3	11	30	86	17.022	16.306	0.907	0.963	0.976	-0.339
num_ins_X_rel	97	0	0	0	0	99	1.068	7.057	0.887	0.957	0.933	0.636
num_ins_Y_rel	45	0	0	0	0	84	0.141	2.501	0.980	0.990	0.987	0.917
num_instructions	1,293	10	23	48	100	6,743	89.718	182.991	0.950	0.961	0.973	0.771
num_loops	36	0	0	0	1	100	0.380	1.139	0.957	0.962	0.961	0.894
num_outrefs	88	0	0	0	0	128	0.692	2.843	0.886	0.886	0.886	0.810
num_returns	56	0	1	2	3	232	2.197	3.249	0.973	0.930	0.956	0.682
num_sccs	262	1	4	8	15	555	12.672	16.598	0.973	0.941	0.954	0.931
stack_size	389	0	0	8	48	4,092	44.652	143.470	0.952	0.912	0.971	-0.292

Table 6.9.: Results for the metrics-based feature evaluation. Qualified features highlighted yellow, selected features highlighted green. ρ lists the Spearman rank correlation for function pairs of 32bit and 64bit as well as same and difference bitness only.

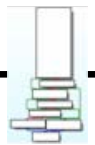


Full details in my PhD thesis

[1] „Classification, Characterization, and Contextualization of Windows Malware using Static Behavior and Similarity Analysis“, D. Plohmann, 2022.

MCRIT
Querying the System

- Query with
  - Basic Block
  - Function
  - Sample



	offset	num_bytes				Score
4459063	0x406770	150	win.hardrain	909	889162	62
4459063	0x406770	150	win.hardrain	1159	1135454	62
4459063	0x406770	150	win.op_blockbuster	1859	1816610	89
4459063	0x406770	150	win.op_blockbuster	5954	6681224	89
4459063	0x406770	150	win.romeos	4498	4990660	100
4459063	0x406770	150	win.romeos	4730	5238246	92
4459063	0x406770	150	win.romeos	5710	6362927	92
4459063	0x406770	150	win.badcall	1288	1257856	65

For every function, we know how many families we match.

1. We can use this to weigh by occurrence frequency when aggregating to sample matches.
2. We can identify unique matches into just one family and use this as further indication for identity.

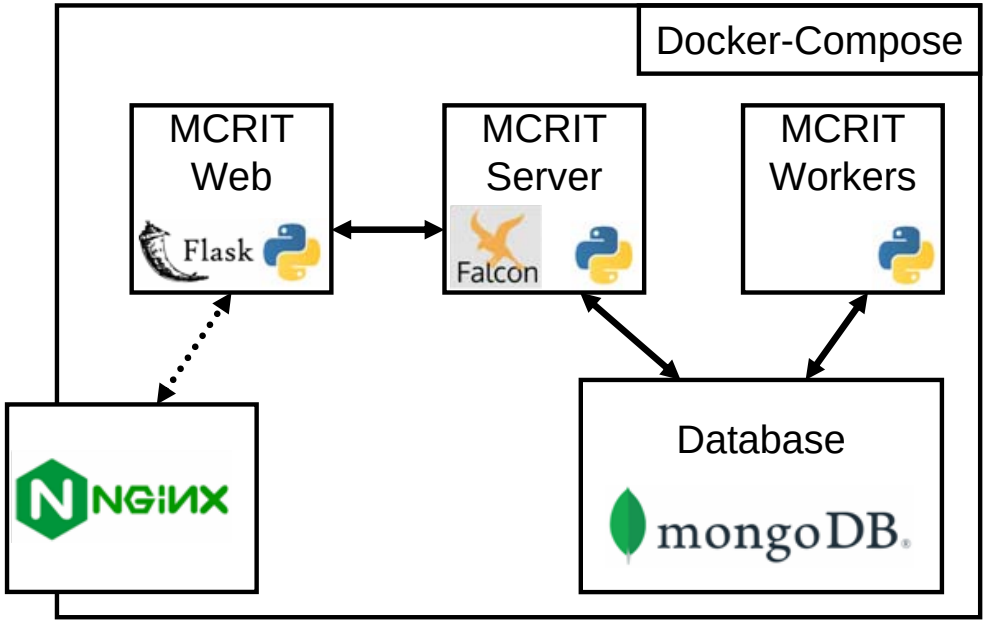
	Version		SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct	Frequency		Uniq
win.romeos	2014-07-07-alfa	4498	4ec0214b	eff542ac8e...0x00400000	32	311	247	247	71	98	97	47	59	21.49%
win.hardrain		909	c3b1af35	2cc3b5f2df...0x00340000	32	289	138	103	66	50	37	13	13	0.00%
win.op_blockbuster	2015-04-08	1859	2f629c3c	2f629c3c65...3_unpacked	32	341	159	122	70	56	43	12	12	0.00%
win.keymarble	2017-04-12	4543	f19cd9ef	e23900b00f...0x00400000	32	448	168	137	70	53	38	9	8	0.00%

[1] eff542ac8e37db48821cb4e5a7d95c044fff27557763de3a891b40ebeb52cc55 @ 0x406770 | win.romeos |

Fraunhofer FKIE

MCRIT
Setup

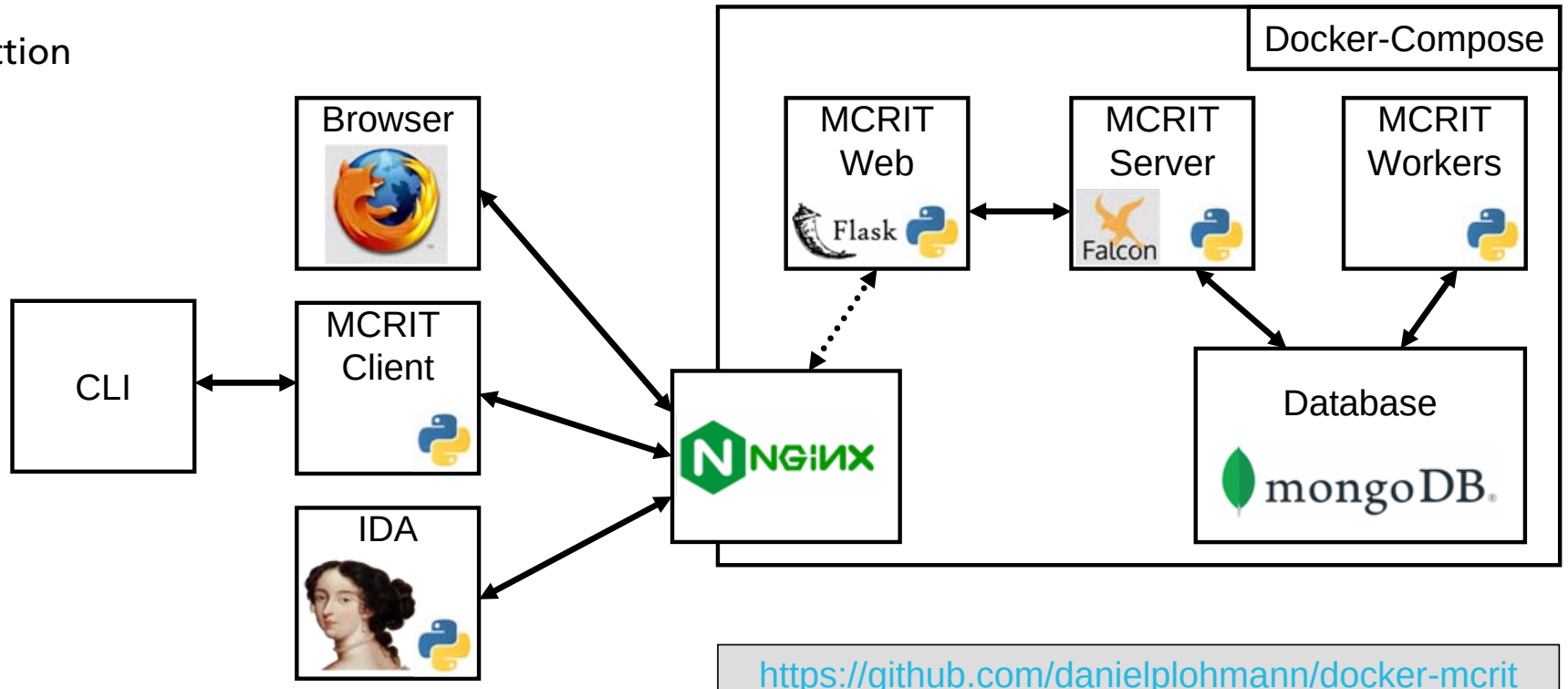
- Database
- MCRIT Server
  - Core of the system
  - Enable access to stored content (API)
  - Create matching jobs
- MCRIT Workers
  - Process jobs from the queue
- MCRIT Web
  - Expose service functionality in a user interface
  - User management
  - API forwarding to MCRIT server



https://github.com/danielplohmann/docker-mcrit

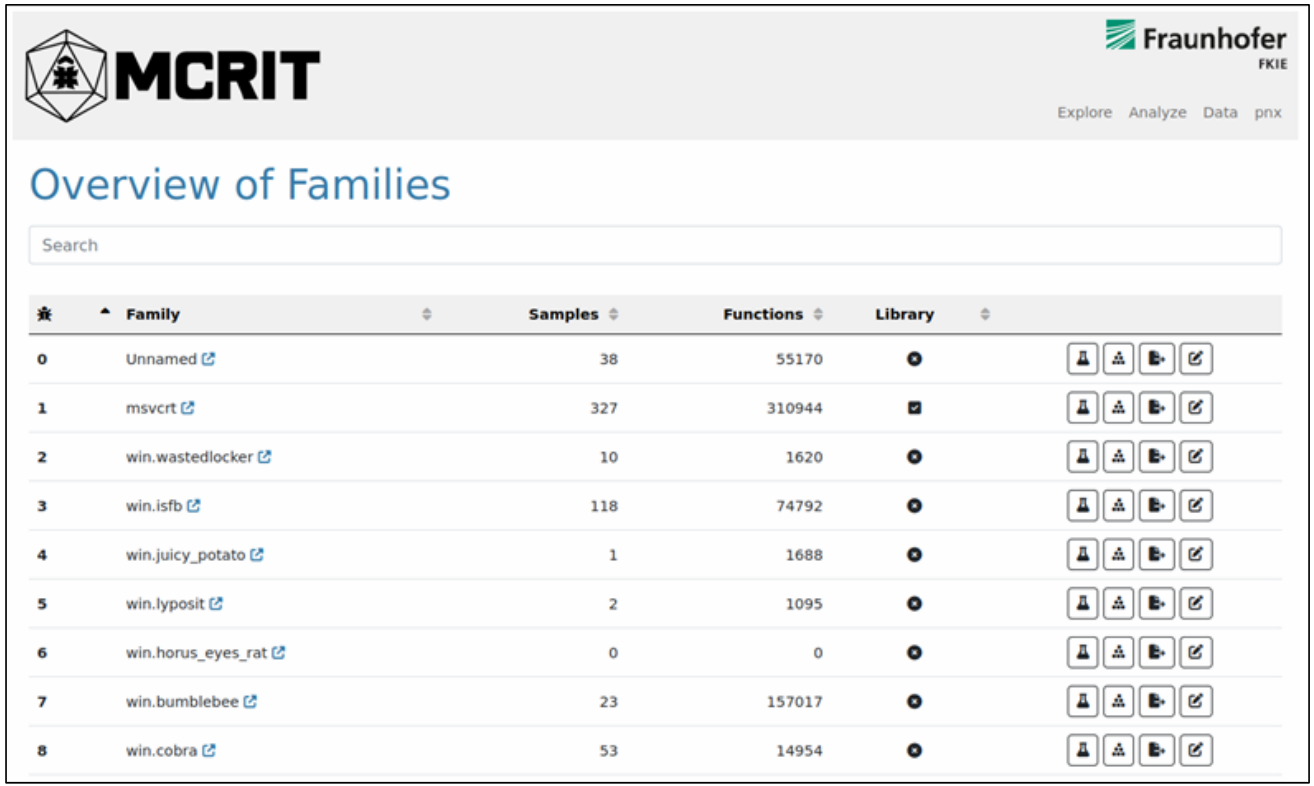
MCRIT
Setup

- Interaction

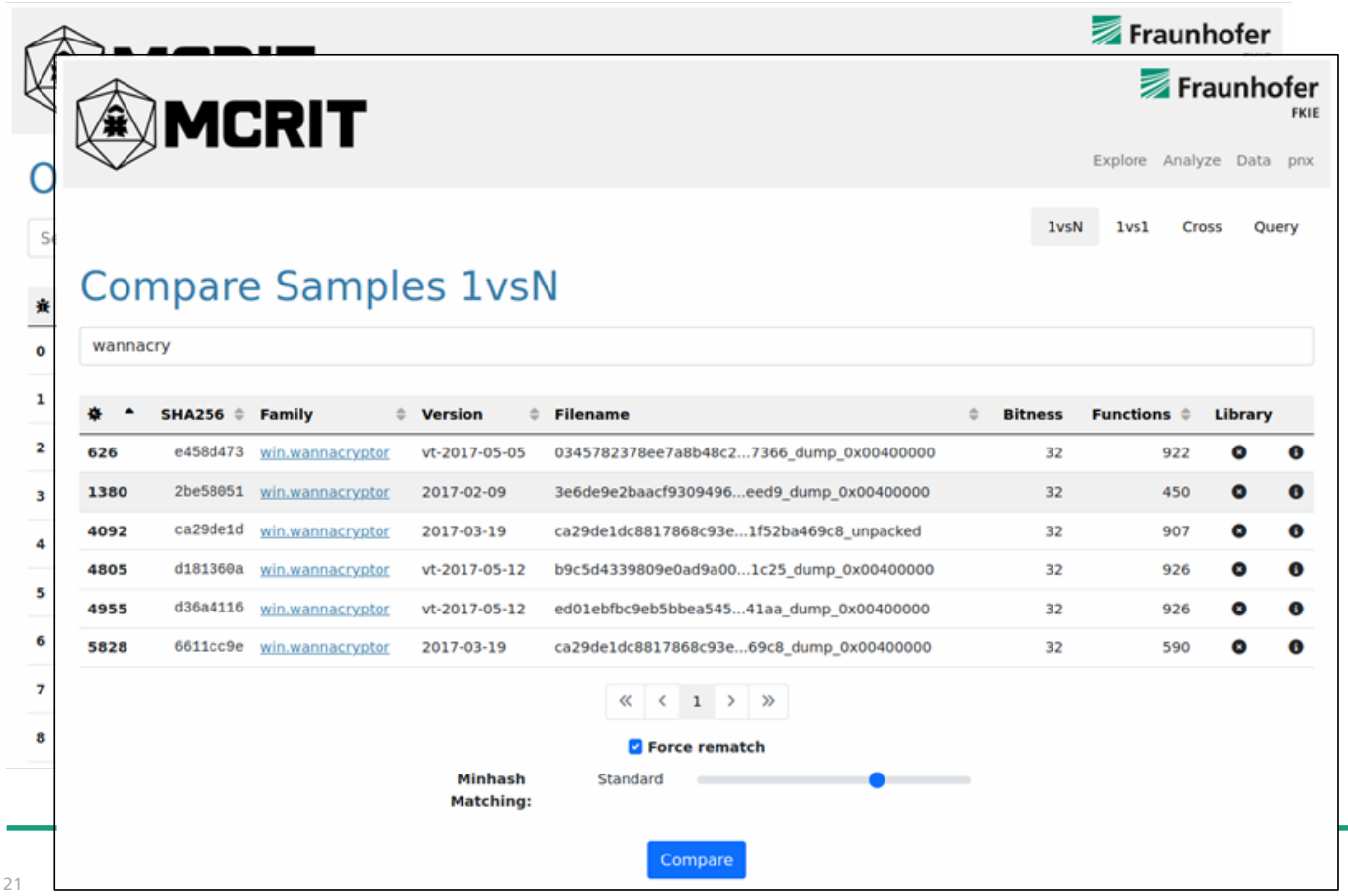


https://github.com/danielplohmann/docker-mcrit

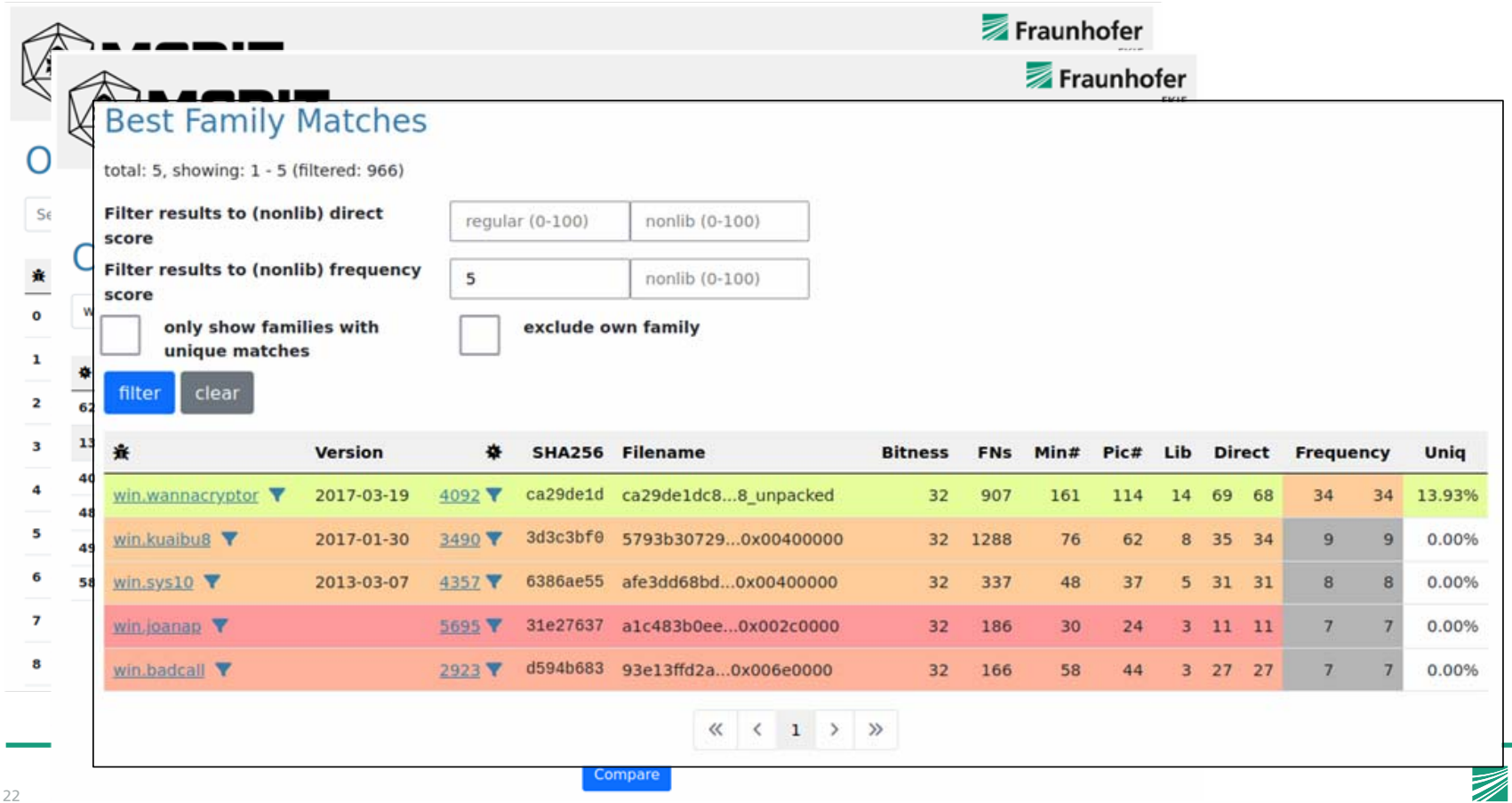
MCRIT
Setup: WebUI



MCRIT
Setup: WebUI



MCRIT
Setup: WebUI



Use Cases

Example Use Cases

- Malware family identification and library code differentiation
- Isolation of unique family code
- Lead generation for discovering potentially unknown links
- Label Transfer

Example Use Cases

Malware Family Identification and Library Code Differentiation

LockBit ransomware goes 'Green,' uses new Conti-based encryptor

By Lawrence Abrams — February 1, 2023 05:48 PM 0

The LockBit ransomware gang has again started using encryptors based on other operations, this time switching to one based on the leaked source code for the Conti ransomware.

Since its launch, the LockBit operation has gone through numerous iterations of its encryptor, starting with a custom one and moving to LockBit 3.0 (aka LockBit Black), which is derived from the BlackMatter gang's source code.

This week, cybersecurity collective VX-Underground first reported that the ransomware gang is now using a new encryptor named 'LockBit Green,' based on the leaked source code of the now-disbanded Conti gang.

num_samples	6243
num_families	1440
num_functions	6995154
num_pichashes	1464810

Query Sample

Drop file or click here to import

45c317_lockbit_green.exe

Unmapped

Dumped

Minhash Matching: Standard

Submit

Disassembly + Matching: 35sec

[1] https://www.bleepingcomputer.com/news/security/lockbit-ransomware-goes-green-uses-new-conti-based-encryptor/
[2] lockbit green: 45c317200e27e5c5692c59d06768ca2e7eeb446d6d495084f414d0f261f75315

Fraunhofer FKIE

Best Family Matches

total: 1070, showing: 1 - 10

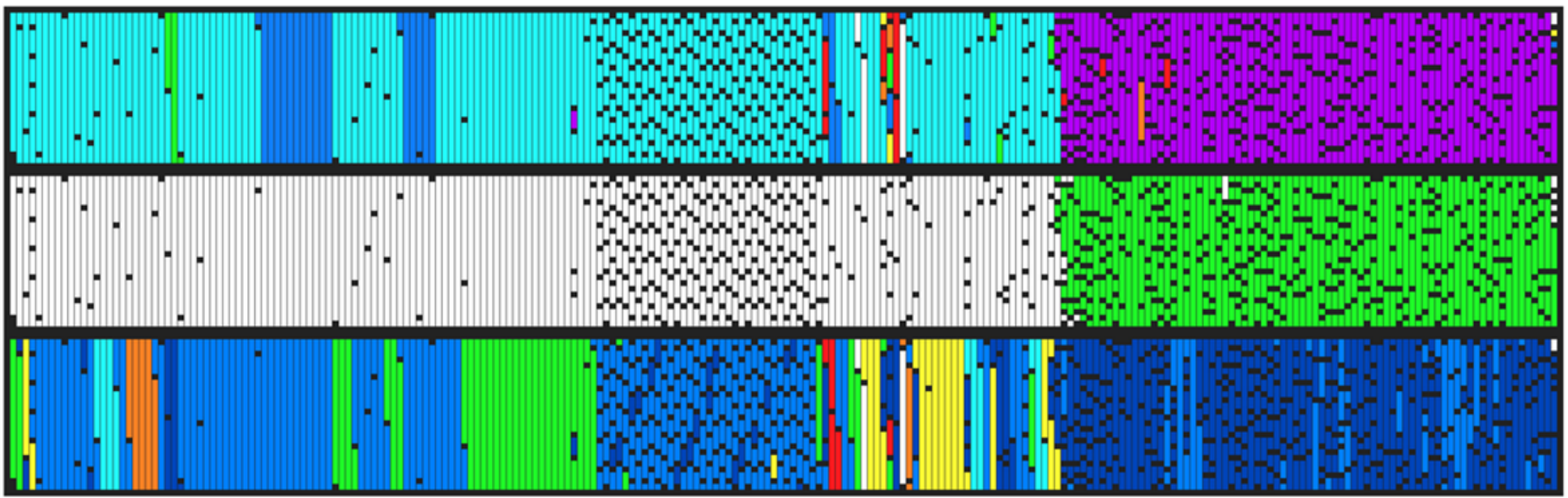
	Version		SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct		Frequency	
win.meow		2172	222e2b91	222e2b91f5...3_unpacked	32	702	461	126	220	66	71	41	53
win.conti	2021-02-04	5041	a5751a46	a5751a4676...6_unpacked	32	736	516	183	256	59	58	28	36
win.scarecrow		6199	bcf49782	bcf49782d7...a_unpacked	32	653	582	271	334	61	51	27	32
win.lockergoga	2019-03-18	4517	edae201c	c97d9bbc80...0x00400000	32	7847	369	311	352	26	1	3	0
win.void		1460	2fd1863e	2fd1863eb3...c_unpacked	32	7123	366	312	351	26	1	3	0
win.bandook		4792	fabce973	fabce973a9...7_unpacked	32	3229	363	306	349	25	1	3	0

Best Family Matches

total: 1070, showing: 1 - 10

	Version		SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct		Frequency	
win.meow		2172	222e2b91	222e2b91f5...3_unpacked	32	702	461	126	220	66	71	41	53
win.conti	2021-02-04	5041	a5751a46	a5751a4676...6_unpacked	32	736	516	183	256	59	58	28	36
win.scarecrow		6199	bcf49782	bcf49782d7...a_unpacked	32	653	582	271	334	61	51	27	32

Showing: foreign family match frequency, library matches, best foreign family match scores.

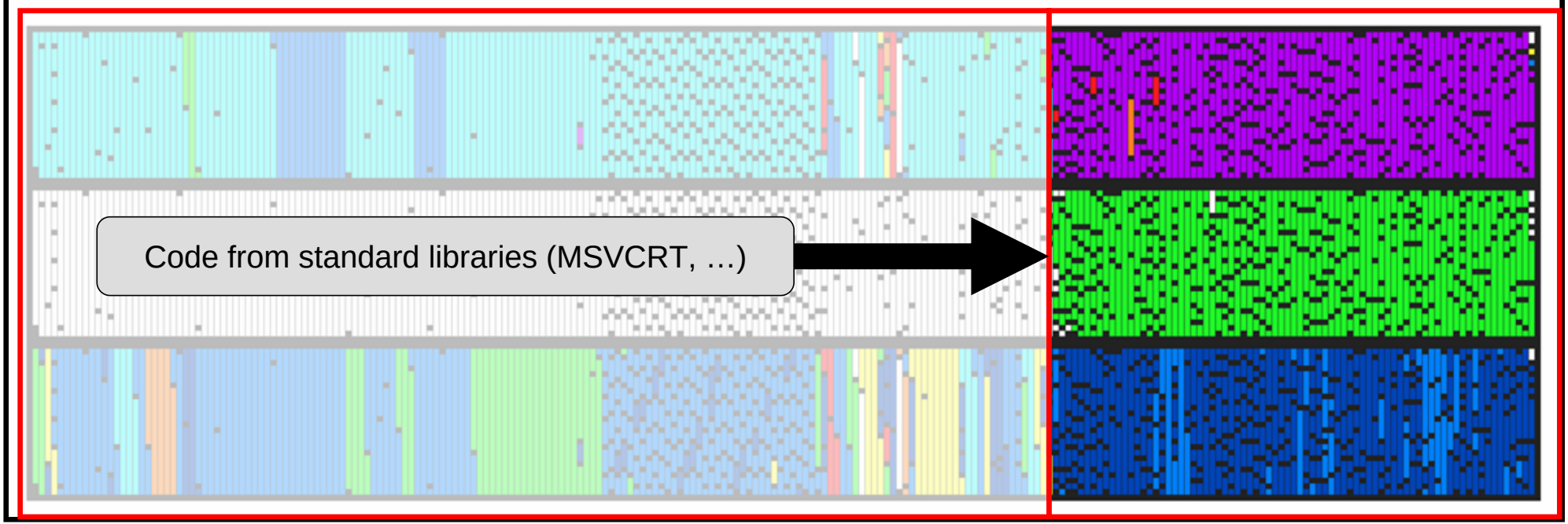


Best Family Matches

total: 1070, showing: 1 - 10

	Version		SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct		Frequency	
win.meow		2172	222e2b91	222e2b91f5...3_unpacked	32	702	461	126	220	66	71	41	53
win.conti	2021-02-04	5041	a5751a46	a5751a4676...6_unpacked	32	736	516	183	256	59	58	28	36
win.scarecrow		6199	bcf49782	bcf49782d7...a_unpacked	32	653	582	271	334	61	51	27	32

Showing: foreign family match frequency, library matches, best foreign family match scores.

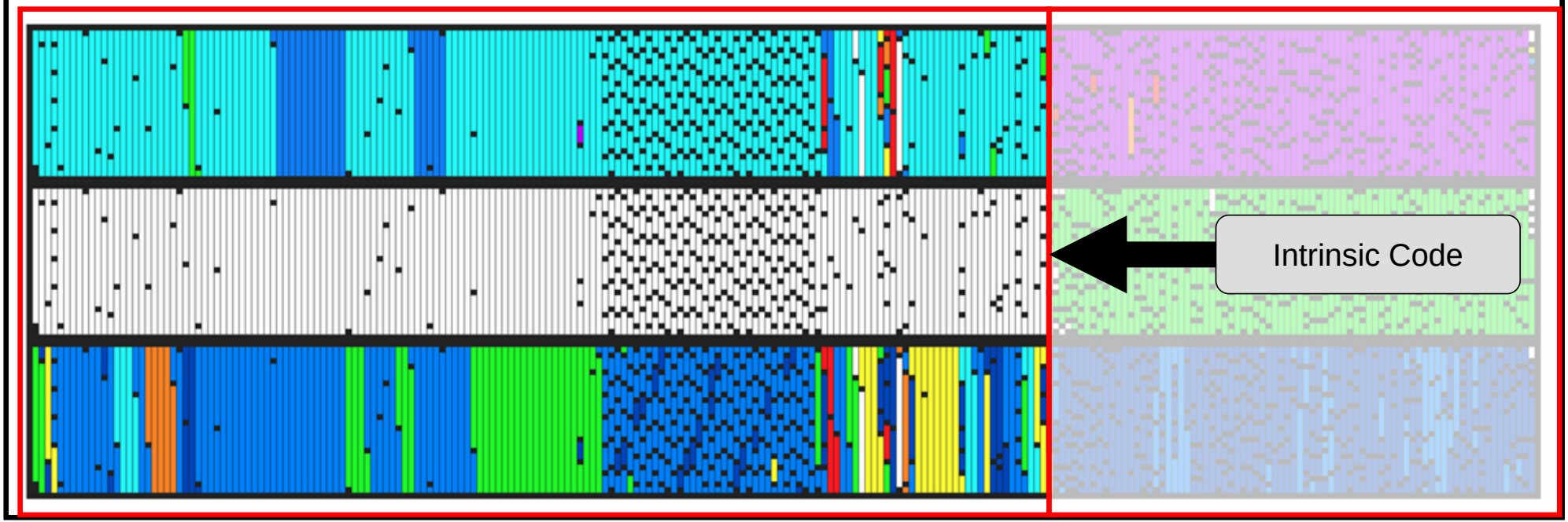


Best Family Matches

total: 1070, showing: 1 - 10

	Version		SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct		Frequency	
win.meow		2172	222e2b91	222e2b91f5...3_unpacked	32	702	461	126	220	66	71	41	53
win.conti	2021-02-04	5041	a5751a46	a5751a4676...6_unpacked	32	736	516	183	256	59	58	28	36
win.scarecrow		6199	bcf49782	bcf49782d7...a_unpacked	32	653	582	271	334	61	51	27	32

Showing: foreign family match frequency, library matches, best foreign family match scores.



[1] conti: a5751a46768149c5ddf318fd75afc66b3db28a5b76254ee0d6ae27b21712e266

Function CFGs

Show Cycles | Show Loops | Show Loop Boundaries | Enable Tooltip

go to top

lockbit green
@0x401730

conti
@0x10017a0

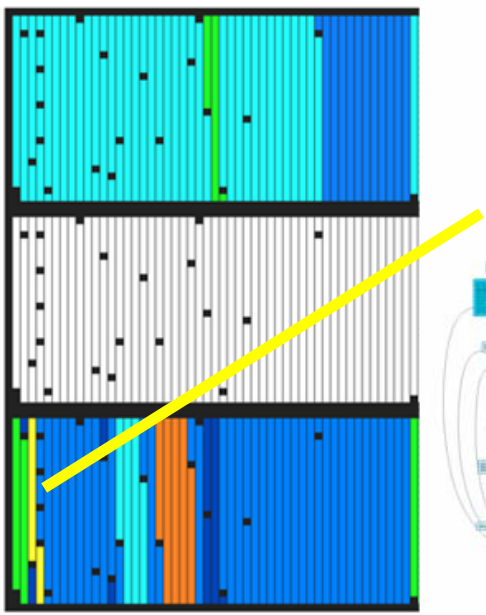
Best Family Match

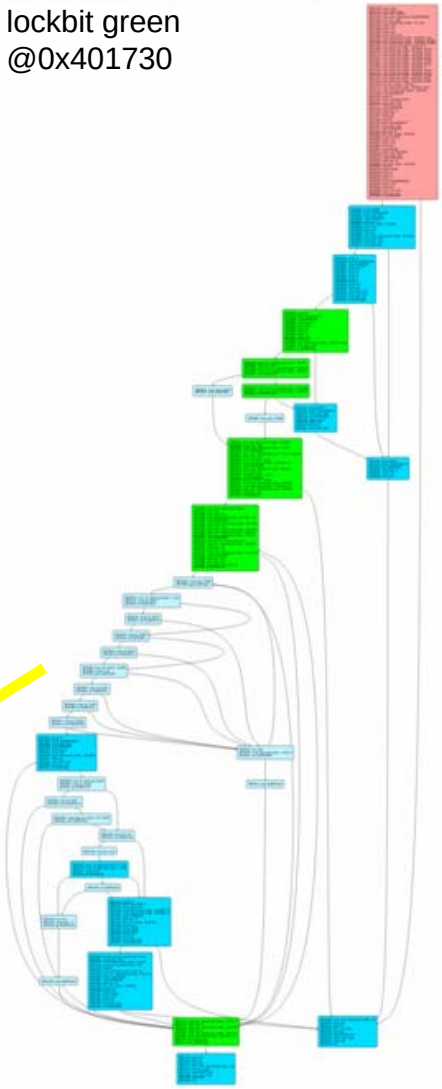
total: 1070, showing: 1 - 10

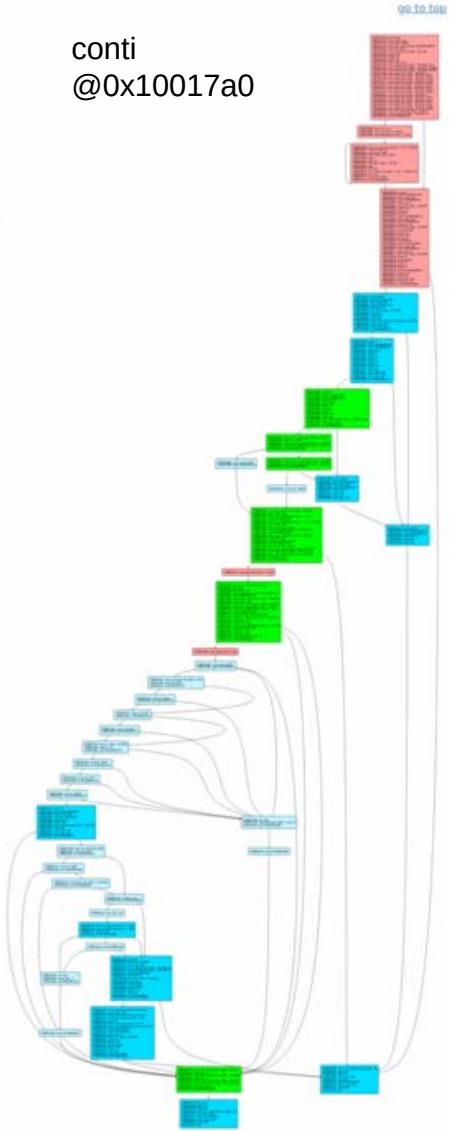
	Version
win.meow	
win.conti	2021-02-
win.scarecrow	

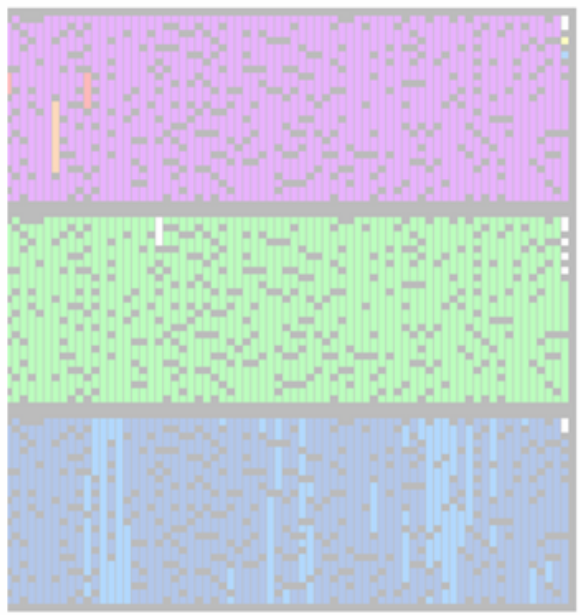
Min#	Pic#	Lib	Direct		Frequency	
461	126	220	66	71	41	53
516	183	256	59	58	28	36
582	271	334	61	51	27	32

Showing: foreign family match freque



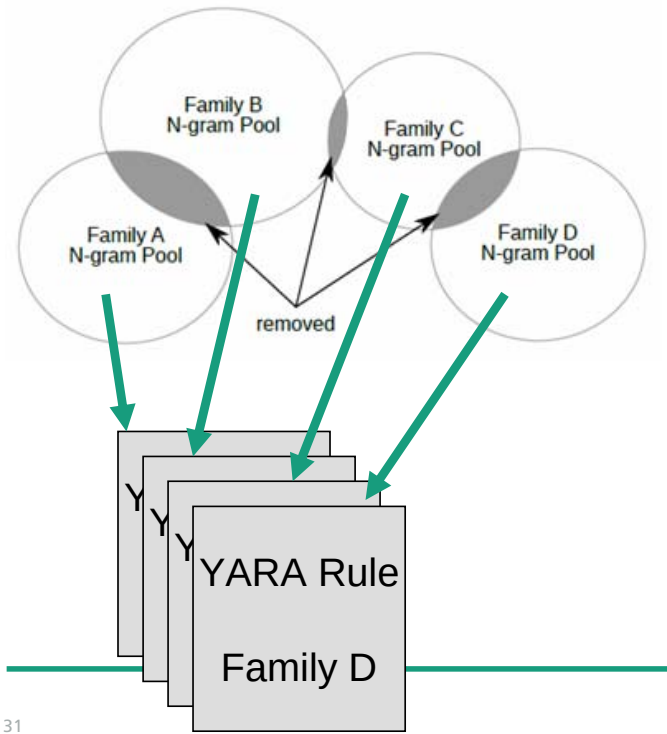






Example Use Cases
Isolation of Unique Family Code

- Essentially like YARA-Signator, but with basic blocks



Unique Block Isolation Report

Job ID	639358a4e0ff5413a77221e4
Family	win.remcos
Samples	19
Unique Blocks	10028
Has a YARA rule?	True, covers: 19 samples
YARA rule covers all?	True

Statistics | Unique Blocks | YARA Rule

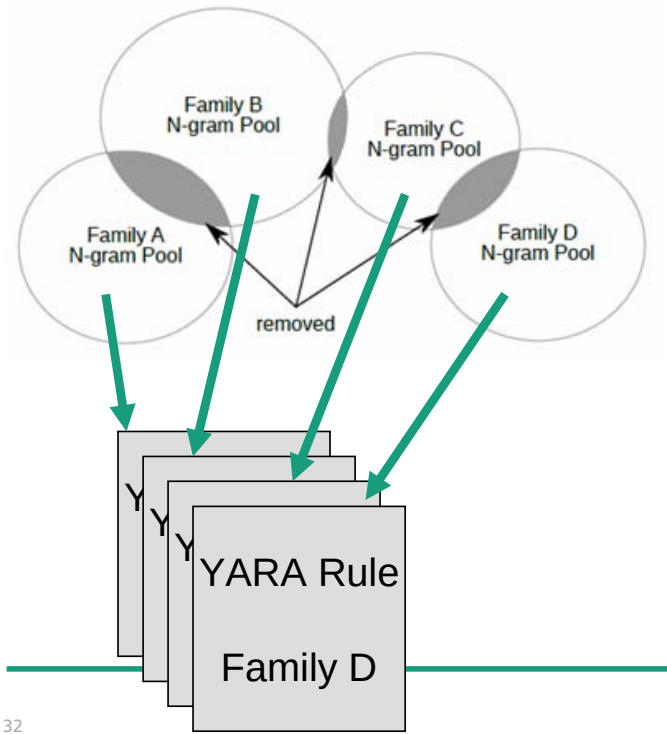
Block Statistics across Samples

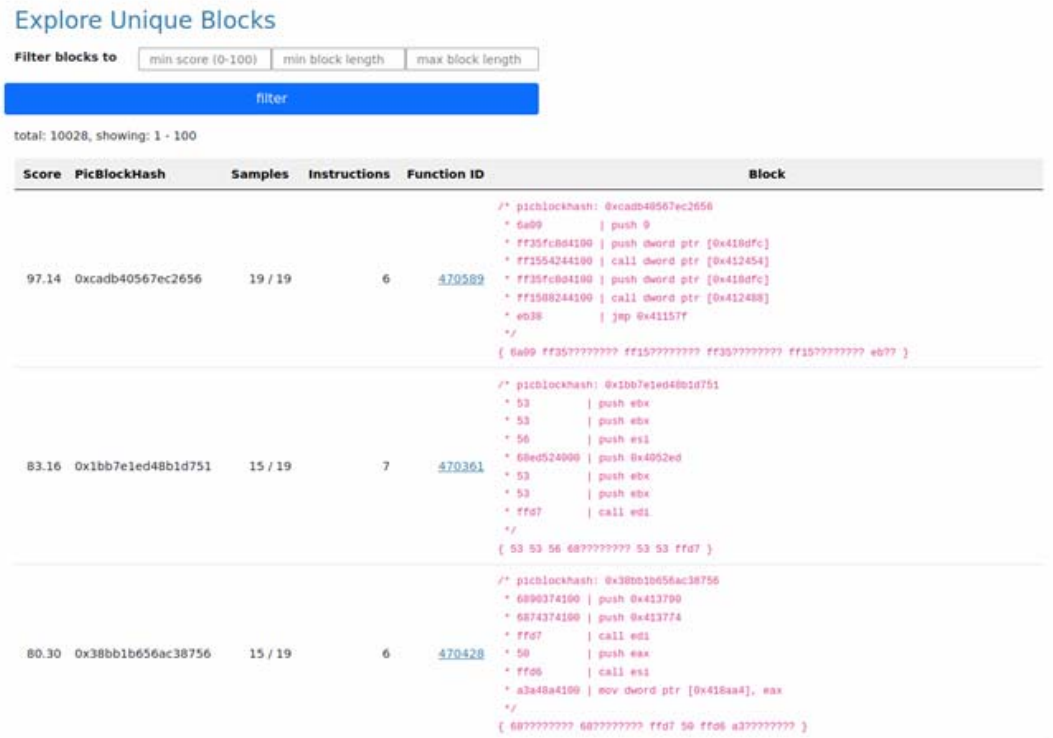
Characteristic blocks are basic blocks only found in this collection of samples (versus rest of the whole data set), unique blocks are only found in the specific sample.

Sample ID	Total Blocks	Characteristic Blocks	Unique Blocks
1111	1104	965 (87.41%)	0 (0.00%)
1342	1178	1056 (89.64%)	567 (48.13%)
1343	1106	967 (87.43%)	1 (0.09%)
1519	746	548 (73.46%)	68 (9.12%)
1968	8020	3423 (42.68%)	188 (2.34%)
2217	867	743 (85.70%)	266 (30.68%)
3729	1190	1041 (87.48%)	32 (2.69%)
4501	8307	3937 (47.39%)	0 (0.00%)
4561	8315	3938 (47.36%)	0 (0.00%)
4575	1299	1137 (87.53%)	74 (5.70%)
4683	876	745 (85.05%)	49 (5.59%)

Example Use Cases
Isolation of Unique Family Code

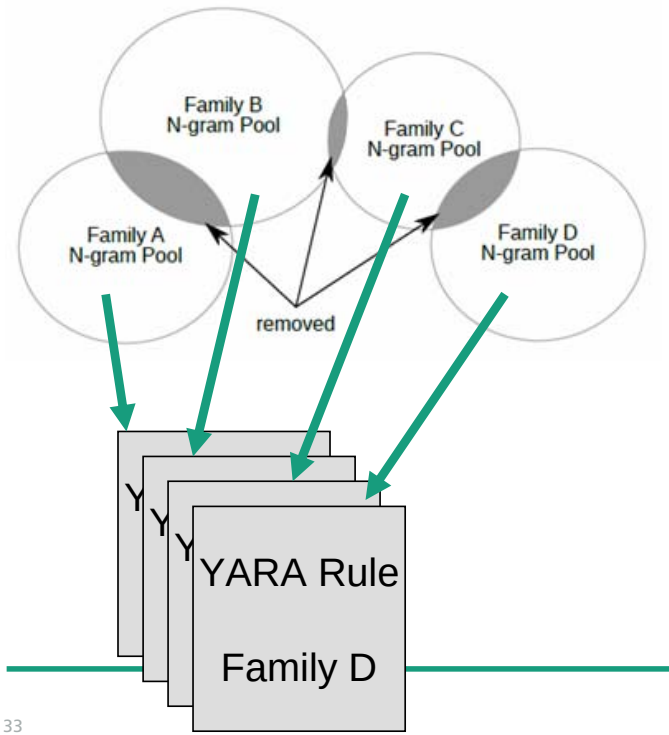
- Essentially like YARA-Signator, but with basic blocks





Example Use Cases
Isolation of Unique Family Code

- Essentially like YARA-Signator, but with basic blocks



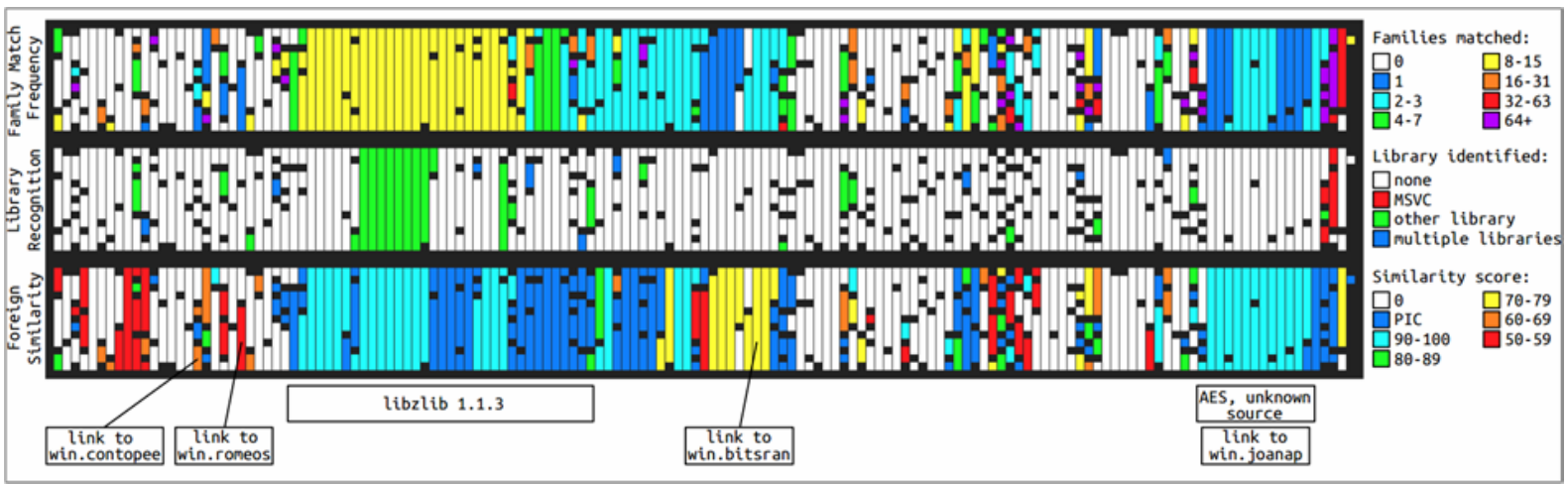
Proposed YARA rule

Copy rule to clipboard!

```
rule mcrit_639358a4e0ff5413a77221e4 {
    meta:
        author = "MCRIT YARA Generator"
        description = "Code-based YARA rule composed from potentially unique basic blocks for the selected set of samples/family."
        date = "2023-03-29"
    strings:
        // Rule generation selected 20 picblocks, covering 19/19 input sample(s).
        /* picblockhash: 0x1bb7e1ed48b1d751 - coverage: 15/19 samples.
         * 53          | push ebx
         * 53          | push ebx
         * 56          | push esi
         * 68ed524000  | push 0x4052ed
         * 53          | push ebx
         * 53          | push ebx
         * ffd7        | call edi
         */
        $blockhash_0x1bb7e1ed48b1d751 = { 53 53 56 68???????? 53 53 ffd7 }

        /* picblockhash: 0x2e684020db1f147c - coverage: 14/19 samples.
         * ff7508      | push dword ptr [ebp + 8]
         * 6a00        | push 0
         * 6800040000  | push 0x400
         * ff15cc204100 | call dword ptr [0x4120cc]
         * 8d4d08      | lea ecx, [ebp + 8]
         * 51          | push ecx
         * 50          | push eax
         * ff15e48d4100 | call dword ptr [0x418de4]
         * 837d0800    | cmp dword ptr [ebp + 8], 0
         * 7504        | jne 0x410ae5
         */
        $blockhash_0x2e684020db1f147c = { ff7508 6a00 6800040000 ff15???????? 8d4d08 51 50 ff15???????? 837d0800 75?? }

        /* picblockhash: 0x38bb1b656ac38756 - coverage: 15/19 samples.
         * 6890374100  | push 0x413790
         * 6874374100  | push 0x413774
         * ffd7        | call edi
         * 50          | push eax
         * ffd6        | call esi
         * a3a48a4100  | mov dword ptr [0x418aa4], eax
         */
        $blockhash_0x38bb1b656ac38756 = { 68???????? 68???????? ffd7 50 ffd6 a3???????? }

        /* picblockhash: 0x3fc81ffe2622434d - coverage: 14/19 samples.
         * ff7508      | push dword ptr [ebp + 8]
         * ff36        | push dword ptr [esi]
         * ff15ac234100 | call dword ptr [0x4123ac]
         * 59          | pop ecx
         * 85c0        | test eax, eax
         * 59          | pop ecx
         * 741f        | je 0x40f2dc
```

Example Use Cases
Lead Generation for Discovering Potentially Unknown Links

- May 15th 2017 - Tweet by Neel Mehta (Google) with hashes + offsets
  - Earlier version of WannaCry sharing „rare“ code with Contopee
- Identification of similar functions with appearance across few families
  - Potential reuse of non-public code as an indicator for relationship



[1] https://twitter.com/neelmehta/status/864164081116225536
[2] „Classification, Characterization, and Contextualization of Windows Malware using Static Behavior and Similarity Analysis“, D. Plohmann, 2022.

Fraunhofer FKIE

Example Use Cases

Lead Generation for Discovering Potentially Unknown Links

Function Matches

selection: 49, showing: 1 - 49 (filtered: 128)

Filter results to 65 | max score (0-100) | 3

exclude functions with library hits

exclude PIC hits

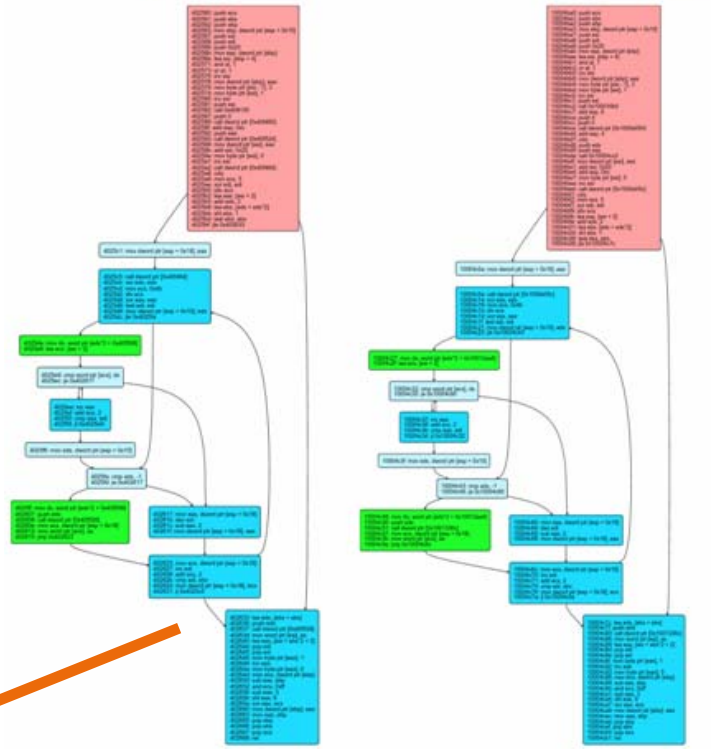
filter clear

Function ID	offset	num_bytes	Matched Families	Matched Samples	Matched Functions	Best Score	Min	Pic	Lib	Uniq
1344938	0x401000	58	1	5	5	100	5	5	0	☑
1344939	0x401040	173	1	3	3	100	3	3	0	☑
1344941	0x401120	252	1	3	3	100	3	3	0	☑
1344942	0x401220	69	1	3	3	100	3	3	0	☑
1344948	0x401360	61	1	5	5	100	5	3	0	☑
1344953	0x401790	278	1	5	5	90	5	0	0	☑
1344955	0x4018d0	183	1	5	5	68	5	0	0	☑
1344961	0x401dd0	42	1	5	5	100	5	5	0	☑
1344965	0x401e40	145	1	5	5	100	5	5	0	☑
1344966	0x401ee0	190	1	5	5	100	5	5	0	☑
1344967	0x401fa0	181	1	5	5	84	5	0	0	☑
1344968	0x402060	344	1	5	5	84	5	0	0	☑
1344969	0x4021c0	77	1	5	5	78	5	0	0	☑
1344970	0x402210	451	1	5	5	95	5	0	0	☑
1344971	0x4023e0	214	1	5	5	82	5	0	0	☑
1344975	0x402500	55	3	7	7	100	7	7	0	●
1344977	0x402560	265	1	1	1	67	1	0	0	☑

win.contopee
win.op_blockbuster
win.romeos

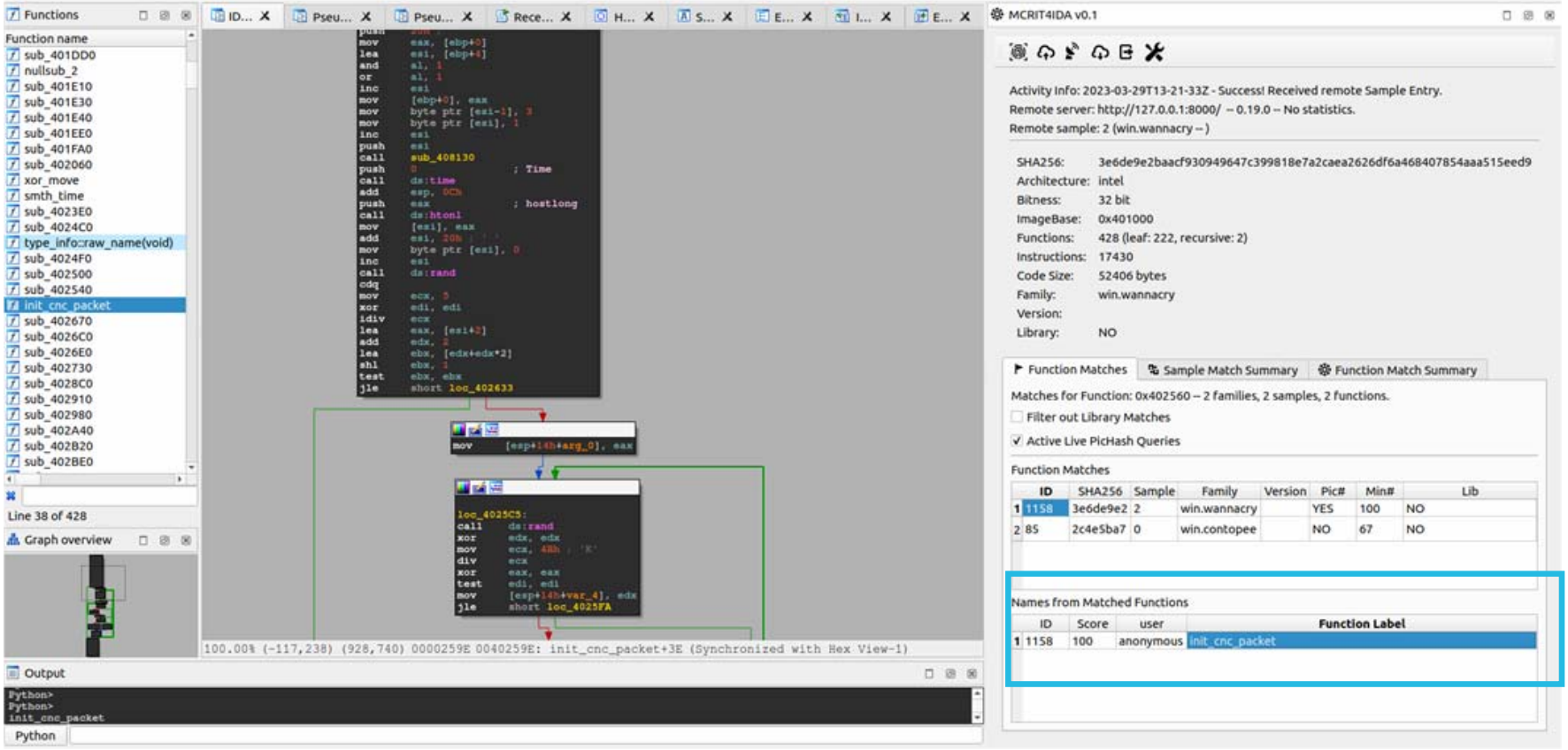
Function CFGs

Show Cycles | Show Loops | Show Loop Boundaries | Enable Tooltip



Fraunhofer FKIE

Example Use Cases
Label Transfer



Summary

Limitations
Minhash-based Code Relationship & Investigation Toolkit (MCRIT)

- Version released today is a first version
  - Fully functional, but needs some usability improvements
- Data exchange
  - Basic import / export, looking to improve reference data distribution
- Architecture support
  - x86/x64 only, not optimized for cross-bitness
- Matching / Search
  - Currently only PicHash and MinHash, may add further options (WinAPI, strings, PE/ELF meta data, …)

[1] https://github.com/danielplohmann/docker-mcrit

Summary
Minhash-based Code Relationship & Investigation Toolkit (MCRIT)

- MCRIT
  - A framework for quasi-identical and fuzzy 1:n code matching
- Variety of Use Cases
  - Code identification & library filtering, hunting, label transfer, …
- Full Open Source Release
  - Convenient deployment via Docker [1]

MCRIT: The MinHash-based Code Relationship & Investigation Toolkit

Daniel Plohmann[1], Manuel Blatt[1], and Daniel Enders[1]
[1]Fraunhofer FKIE

This paper was presented at Botconf 2023, Strasbourg, 11-14 April 2023, www.botconf.eu
It is published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr/ojs
It is shared under the CC BY license http://creativecommons.org/licenses/by/4.0/.

Abstract

As the number of malware attacks continually rises, malware analysts are facing an ever-increasing workload. The growing complexity of malware families and the sheer volume of new threats make it challenging for analysts to keep up with their analysis tasks.

Code similarity analysis offers high potential in this regard, helping analysts to orient themselves and to speed up analysis. While being a very active research field with many recent publications, only few of these focus on malware or support immediate practical usage, as they are rarely accompanied by public code releases.

In this paper, we present the MinHash-based Code Relationship & Investigation Toolkit (MCRIT). MCRIT is intended to serve as a framework for code similarity analysis, mainly focusing on One-to-Many (1:N) comparisons and with the ability to recognize and filter out library code. We publish MCRIT as open source, including a dockerized setup for easy deployment.

Keywords: malware analysis, reverse engineering, code similarity.

1 Introduction

As the information security is constantly evolving, malware analysts continue to face a constant influx of samples from a wide array of malware families. Recent phenomena such as the frequent rebranding observable for multiple ransomware strains or the occasional retooling of APTs make it increasingly hard to keep track.

Code similarity analysis is a means with high potential in this regard to help analysts navigate the space and accelerate their analysis. It has become a popular research topic in recent years, easily explained by its multi-dimensional use in security context, such as vulnerability discovery and malware analysis. Especially when diving deep into malware families, focusing on their internal evolution and relationship among each other, there is a crucial need to have this capability available.

In 2019, Haq and Caballero [1] published a survey, listing about 50 academic works on the topic that have been proposed since 2010. Sadly, many of the presented approaches have not been evaluated on larger malware data sets and few are focused on use cases like malware clustering, lineage analysis, or library recognition. They are also not tailored towards immediate practical usage, as they are rarely accompanied by public code releases. Other established tools like BinDiff [2] and Diaphora [3] focus only on direct comparison of single files with each other.

With this paper, we present and publish MCRIT, the MinHash-based Code Relationship & Investigation Toolkit. MCRIT is intended to serve as an open source framework for code similarity analysis, mainly focusing on One-to-Many (1:N) comparisons and with the ability to recognize and filter out library code. MCRIT was originally developed since 2018 as a pure research-oriented evaluation framework [4] with support for quasi-identical and fuzzy code matching. Over the last two years, it was extended towards becoming applicable in practical malware research context, with internal improvements to efficiency and also the addition of a user friendly web front-end and dockerized deployment.

We envision four primary use cases for MCRIT. First, the toolkit is intended to aid with malware family and library code differentiation. Being able to dis-

Daniel Plohmann, Manuel Blatt, Daniel Enders, MCRIT: The MinHash-based Code Relationship & Investigation Toolkit 1

[1] https://github.com/danielplohmann/docker-mcrit

Thank you for your attention!

Dr. Daniel Plohmann
daniel.plohmann@fkie.fraunhofer.de
 @push_pnx

Fraunhofer
FKIE