

Daniel Plohmann
Virus Bulletin Conference | London | 2023-10-04

Applied one-to-many code similarity analysis

using  MCRIT

Introduction

\$whoami

- Security Researcher @ Fraunhofer FKIE & University of Bonn



Open Source Analysis Tooling:
IDAscope, ApiScout, SMDA, [MCRIT](#), ...



Data Sets



Botnet Takedowns

[1] <https://github.com/danielplohmann>
[2] <https://dgarchive.caad.fkie.fraunhofer.de>
[3] <https://malpedia.io>

Outline

- Motivation
- MCRIT: System Overview
 - Methodology
 - Framework
 - Use Cases
- Case Studies
- Summary and 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

```
Function name  
sub_10004470  
sub_10004490  
sub_100044D0  
sub_100044F0  
sub_10004520  
sub_10004630  
sub_10004680  
sub_10004710  
sub_10004790  
sub_10004830  
sub_10004850  
sub_100048C0  
sub_10004A30  
sub_10004A80  
sub_10004AE0  
sub_10004BA0  
sub_10004CC0  
sub_10004D00  
sub_10004D50  
sub_10004DD0  
sub_10004E20  
DllMain(x,x,x)  
sub_10004EB0  
sub_10004F20  
sub_10004F80  
sub_10004FC0  
sub_10005100  
sub_10005190  
sub_10005260  
sub_100053E0  
sub_10005560  
sub_10005970  
sub_10005A60  
sub_10005AE0  
sub_10005C20  
sub_10005C40  
sub_10005D20  
sub_10005E10  
sub_10005E80  
sub_10005EC0  
sub_10005F00  
sub_10005F20  
sub_10005F50  
Line 81 of 213  
Graph overview
```

```
sub_10004BA0 proc near  
var_4= dword ptr -4  
arg_0= dword ptr 4  
push ecx  
push ebx  
push ebp  
mov ebp, [esp+0Ch+arg_0]  
push esi  
push edi  
push 20h ; .  
mov eax, [ebp+4]  
lea esi, [ebp+4]  
and al, 1  
or al, 1  
inc esi  
mov [ebp+0], eax  
mov byte ptr [esi-1], 3  
mov byte ptr [esi], 1  
inc esi  
push esi  
call sub_100016B0  
add esp, 8  
push 4  
push 0 ; Time  
call ds:time  
add esp, 4  
cdq  
push edx  
push eax  
call sub_10004CC0  
mov [esi], eax  
add esi, 20h ; .  
add esp, 0Ch  
mov byte ptr [esi], 0  
inc esi  
call ds:rand  
cdq  
mov ecx, 5  
xor edi, edi  
idiv ecx  
lea eax, [esi+2]  
add edx, 2  
lea ebx, [edx+edx*2]  
shl ebx, 1  
test ebx, ebx  
jle short loc_10004C7C
```

```
loc_10004C0E:  
call ds:rand  
xor edx, edx  
mov ecx, 4Bh ; 'K'  
div ecx  
xor eax, eax  
test edi, edi  
mov [esp+14h+var_4], edx  
jle short loc_10004C43
```

```
mov dx, word_10012AA4[edx*2]  
lea ecx, [esi+2]
```

To this day...
unique across 1700+
malware families



```
Function name  
sub_407130  
sub_407160  
sub_4071E0  
sub_4072D0  
sub_407820  
sub_407930  
sub_407D70  
sub_407DB0  
sub_407EF0  
sub_407F10  
sub_407F60  
sub_407FB0  
sub_407FD0  
sub_408060  
sub_408070  
sub_408090  
sub_4080D0  
sub_408130  
sub_408180  
sub_4082E0  
sub_408380  
sub_408BE0  
sub_40C920  
sub_40CDE0  
sub_40D1A0  
sub_40D590  
sub_40DB90  
sub_40DB90  
j_NetApiBufferFree  
j_NetShareEnum  
j_WNetCancelConnection2A  
j_WNetAddConnection2A  
j_ftol  
operator delete(void *)  
j_local_unwind2  
j_alloca_probe  
j_CxxThrowException  
operator new(uint)  
sub_40DEA0  
j_XcptFilter  
j_initterm  
nullsub_1  
j_controlfp  
j_Process32Next  
j_Process32First  
j_CreateToolhelp32Snapshot  
j_URLDownloadToFileA  
Line 151 of 151  
Graph overview
```

```
sub_402560 proc near  
var_4= dword ptr -4  
arg_0= dword ptr 4  
push ecx  
push ebx  
push ebp  
mov ebp, [esp+0Ch+arg_0]  
push esi  
push edi  
push 20h ; .  
mov eax, [ebp+4]  
lea esi, [ebp+4]  
and al, 1  
or al, 1  
inc esi  
mov [ebp+0], eax  
mov byte ptr [esi-1], 3  
mov byte ptr [esi], 1  
inc esi  
push esi  
call sub_408130  
add esp, 0Ch  
push eax  
call ds:time  
add esp, 0Ch  
push eax  
call ds:ntohl  
mov [esi], eax  
add esi, 20h ; .  
mov byte ptr [esi], 0  
inc esi  
call ds:rand  
cdq  
mov ecx, 5  
xor edi, edi  
idiv ecx  
lea eax, [esi+2]  
add edx, 2  
lea ebx, [edx+edx*2]  
shl ebx, 1  
test ebx, ebx  
jle short loc_402633
```

```
loc_4025C5:  
call ds:rand  
xor edx, edx  
mov ecx, 4Bh ; 'K'  
div ecx  
xor eax, eax  
test edi, edi  
mov [esp+14h+var_4], edx  
jle short loc_4025FA
```

```
mov dx, ds:netshort[edx*2]  
lea ecx, [esi+2]
```

```
loc_4025E9:  
[ecx], dx  
cmp [ecx], dx  
jz short loc_402617
```



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

Motivation

■ Code Similarity Analysis

- High potential to help analysts and accelerate analysis
 - Source code reuse identification, library filtering, hunting, label transfer, ...
- Existing solutions mostly
 - limited to 1:1 comparison - e.g. BinDiff, Diaphora, ...
 - abandoned or proprietary

■ My goal with this presentation

- 1) Showcase current status and capabilities of MCRIT
- 2) Get feedback and further ideas from you!

MCRIT

System Overview

MCRIT

Design

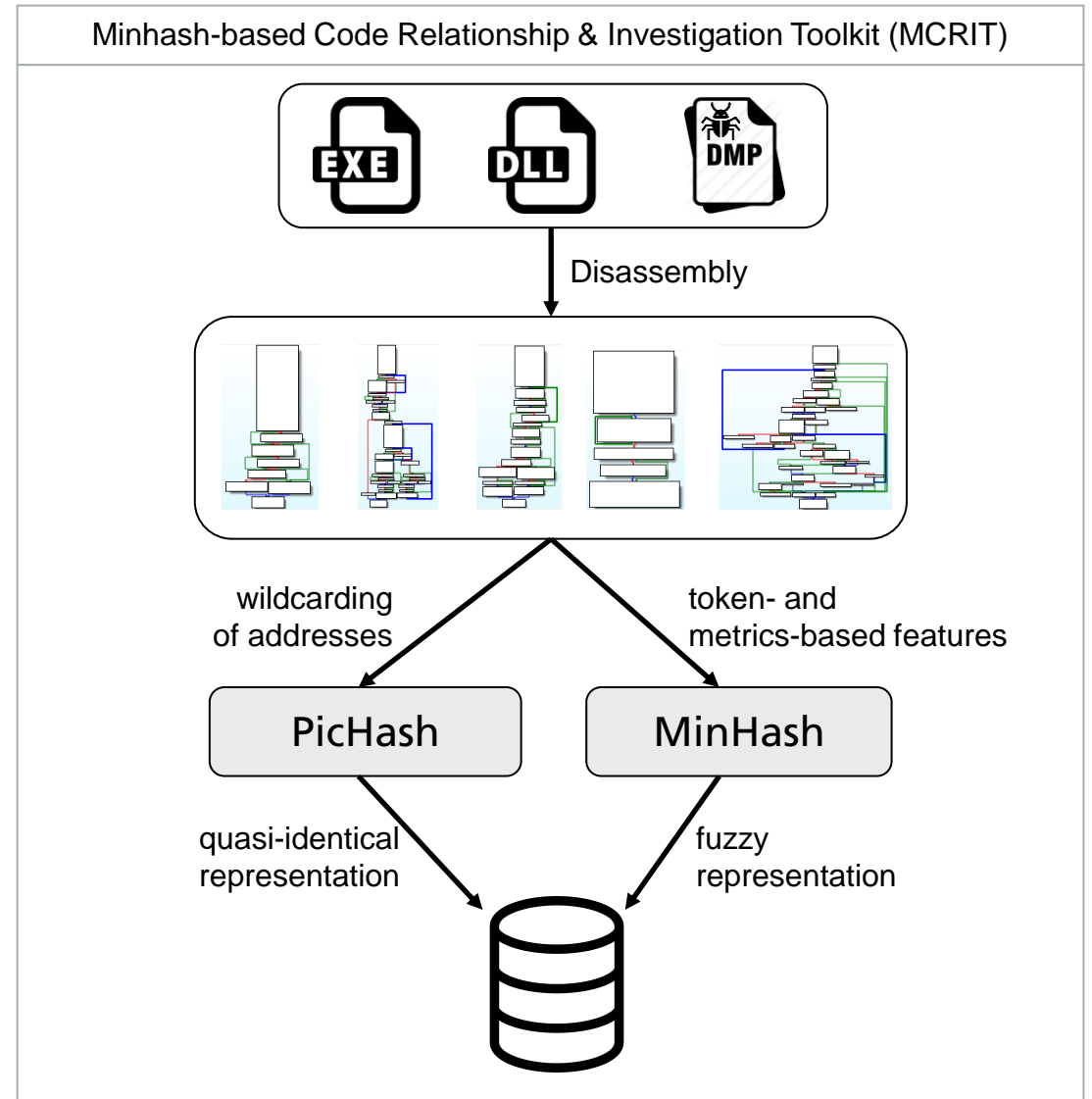
- Minhash-based Code Relationship & Investigation Toolkit (MCRIT)
- Goal: Analyze code sharing and third-party library usage in malware
 - Create open source tools to leverage Malpedia binary corpus
 - Don't reinvent the wheel: **reuse** of **proven techniques** as described in literature
- Requirements:
 - Similarity: reliable, interpretable estimate
 - Scalability: (tens of) millions of functions
 - Efficient representation: (significantly) smaller than indexed code
- Limitation:
 - Limited cross-bitness and no cross-architecture for now

Code Recovery and Similarity Analysis

MCRIT: Overview

■ MCRIT

- Combines quasi-identical and fuzzy code representation
- Basic 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

MCRIT

PicHash: Functions

```

; Attributes: bp-based frame
sub_162401 proc near
var_134= byte ptr -134h
var_A8= byte ptr -0A8h
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 134h
push    ebx
xor     ebx, ebx
push    esi
cmp     [ebp+arg_0], ebx
jz     short loc_162454

cmp     ds:byte_176FCC, 1
jbe     short loc_162454

lea     eax, [ebp+var_134]
push    eax
call    sub_15967A
mov     esi, 0A0h
push    esi
lea     eax, [ebp+var_A8]
push    eax
mov     eax, [ebp+arg_0]
call    sub_16E723
push    esi
push    [ebp+arg_0]
push    3
push    offset aTyap ; "Tyap"
push    offset word_1775D8
call    sub_171A8C
mov     b1, al

loc_162454:
push    ds:dword_1775D4
call    sub_170FBF
mov     al, b1
pop     ebx
leave
retn    4
sub_162401 endp
    
```

- 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 81EC34010000 ...“)
```

-> **8806641384121875405**

```
pichash(„55 8BEC 81EC1C010000 ...“)
```

-> **10270976525648996728**

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

```

; Attributes: bp-based frame
; int __stdcall sub_40F0E0(BYTE *lpData)
sub_40F0E0 proc near

var_11C= byte ptr -11Ch
var_90= byte ptr -90h
lpData= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 11Ch
push    ebx
xor     ebx, ebx
cmp     [ebp+lpData], ebx
jz     short loc_40F131

cmp     byte_43432C, 1
jbe     short loc_40F131

lea     eax, [ebp+var_11C]
push    eax
call    sub_412BE1
lea     eax, [ebp+var_90]
push    eax
mov     eax, [ebp+lpData]
mov     ebx, 0A0h
call    sub_42B569
push    ebx ; cbData
push    [ebp+lpData] ; lpData
push    3 ; dwType
push    offset word_4339C0 ; lpValueName
push    offset word_4339D8 ; lpSubKey
call    sub_42E82B
mov     b1, al

loc_40F131:
; hMutex
push    dword_4339D4
call    sub_42D05E
mov     al, b1
pop     ebx
leave
retn    4
sub_40F0E0 endp ; sp-analysis failed
    
```

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

MCRIT

PicHash: Basic Blocks

```

; Attributes: bp-based frame
sub_162401 proc near
var_134= byte ptr -134h
var_A8= byte ptr -0A8h
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 134h
push    ebx
xor     ebx, ebx
push    esi
cmp     [ebp+arg_0], ebx
jz      short loc_162454
    
```

```

cmp     ds:byte_176FCC, 1
jbe     short loc_162454
    
```

```

lea     eax, [ebp+var_134]
push    eax
call   sub_15967A
mov     esi, 0A0h
push    esi
lea     eax, [ebp+var_A8]
push    eax
mov     eax, [ebp+arg_0]
call   sub_16E723
push    esi
push    [ebp+arg_0]
push    3
push    offset aTyap ; "Tyap"
push    offset word_1775D8
call   sub_171A8C
mov     b1, a1
    
```

```

loc_162454:
push    ds:dword_1775D4
call   sub_170FBF
mov     a1, b1
pop     ebx
leave
retn    4
sub_162401 endp
    
```

■ 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 ...“)
-> 620962970
    
```

```

pichash(„55 8BEC 81EC1C010000 ...“)
-> 2827249019
    
```

```

pichash(„8D85CCFEFFFF 50 E8?? ...“)
-> 847901973
    
```

```

pichash(„8D85E4FEFFFF 50 E8?? ...“)
-> 640583737
    
```

■ Good for recognizing partial code reuse (fractions and candidates)

```

pichash(„FF35????????? E8???? ...“)
-> 463987246
    
```

```

pichash(„FF35????????? E8???? ...“)
-> 463987246
    
```

```

; Attributes: bp-based frame
; int __stdcall sub_40F0E0(BYTE *lpData)
sub_40F0E0 proc near

var_11C= byte ptr -11Ch
var_90= byte ptr -90h
lpData= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 11Ch
push    ebx
xor     ebx, ebx
cmp     [ebp+lpData], ebx
jz      short loc_40F131
    
```

```

cmp     byte_43432C, 1
jbe     short loc_40F131
    
```

```

lea     eax, [ebp+var_11C]
push    eax
call   sub_412BE1
lea     eax, [ebp+var_90]
push    eax
mov     eax, [ebp+lpData]
mov     ebx, 0A0h
call   sub_42B569
push    ebx ; cbData
push    [ebp+lpData] ; lpData
push    3 ; dwType
push    offset word_4339C0 ; lpValueName
push    offset word_4339D8 ; lpSubKey
call   sub_42E82B
mov     b1, a1
    
```

```

loc_40F131:
; hMutex
push    dword_4339D4
call   sub_42D05E
mov     a1, b1
pop     ebx
leave
retn    4
sub_40F0E0 endp ; sp-analysis failed
    
```

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

MCRIT

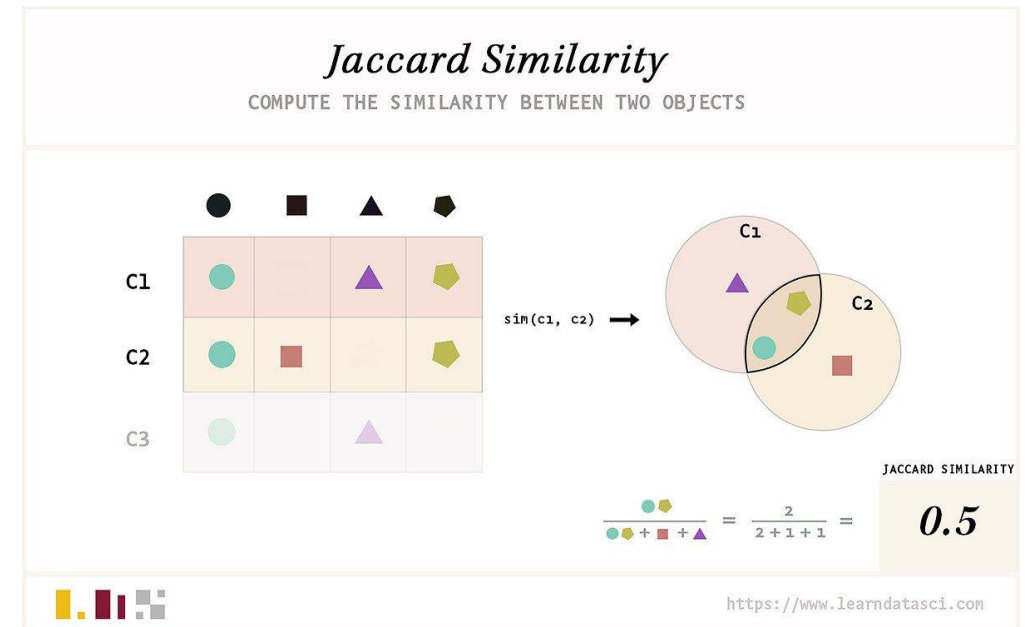
MinHashing

■ 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.

13 [3] <https://www.learn datasci.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

Example Function:

```

push esi
mov esi, 0x023C598C
push esi
call dword ptr [0x02391294]
push 0x14
pop eax
push dword ptr [0x023C5978]
mov word ptr [0x023C5980], ax
mov eax, dword ptr [esp+0x10]
mov dword ptr [0x023C5988], eax
call dword ptr [0x023B9085]
xor eax, eax
cmp dword ptr [esp+0x8], eax
jz 0x023B4830
    
```

```

push dword ptr [esp+0x8]
push eax
or eax, 0xFFFFFFFF
call 0x023B92C5
    
```

```

push esi
mov dword ptr [0x023C5978], eax
call dword ptr [0x0239129C]
pop esi
ret 0x8
    
```

Metrics-based Features

Feature	Value	LogBucket Triplets
max_block_size	14	12 14 16
num_calls	4	3 4 6
num_ins_C	7	4 6 8
num_ins_S	9	6 8 10
num_ins_A_rel	8	6 8 10
num_ins_M_rel	17	14 16 20
stack_size	0	-1 0 1

Token-based Features:

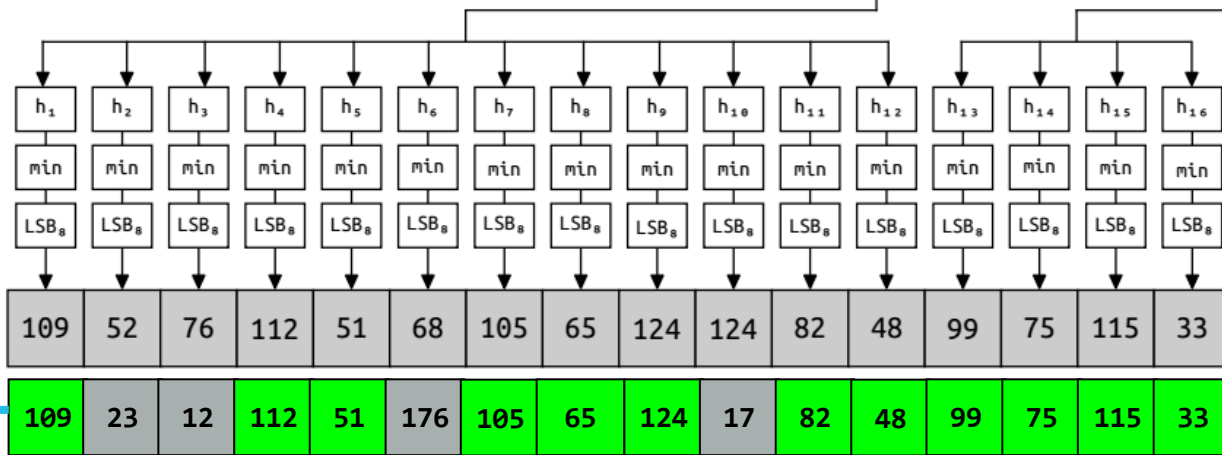
Instruction 3-grams (first 5)

```

push esi - mov esi, 0x023C598C - push esi
mov esi, 0x023C598C - push esi - call dword ptr [0x02391294]
push esi - call dword ptr [0x02391294] - push 0x14
call dword ptr [0x02391294] - push 0x14 - pop eax
push 0x14 - pop eax - push dword ptr [0x023C5978]
    
```

converted to escaped 3-perms:

- M REG, CONST - S REG - S REG
- C PTR - M REG, CONST - S REG
- C PTR - S CONST - S REG
- C PTR - S CONST - S REG
- S CONST - S PTR - S REG



12/16 = 75%

MCRIT

Querying the System

Query with

- Basic Block
- Function
- Sample



Function A	Offset A	Offset B	Function B	Family B	Sample B	Score
4459063	0x406770	0x406770	4990660	win_romeos	4498	100
4459063	0x406770	0x10006b50	6362927	win_romeos	5710	92
4459063	0x406770	0x10006b50	5238246	win_romeos	4730	92
4459063	0x406770	0x405f00	6681224	win_op_blockbuster	5954	89
4459063	0x406770	0x405ff0	1816610	win_op_blockbuster	1859	89
4459063	0x406770	0x10003490	1257856	win_badcall	1288	65
4459063	0x406770	0x10004600	1135454	win_hardrain	1159	62
4459063	0x406770	0x344600	889162	win_hardrain	909	62

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.

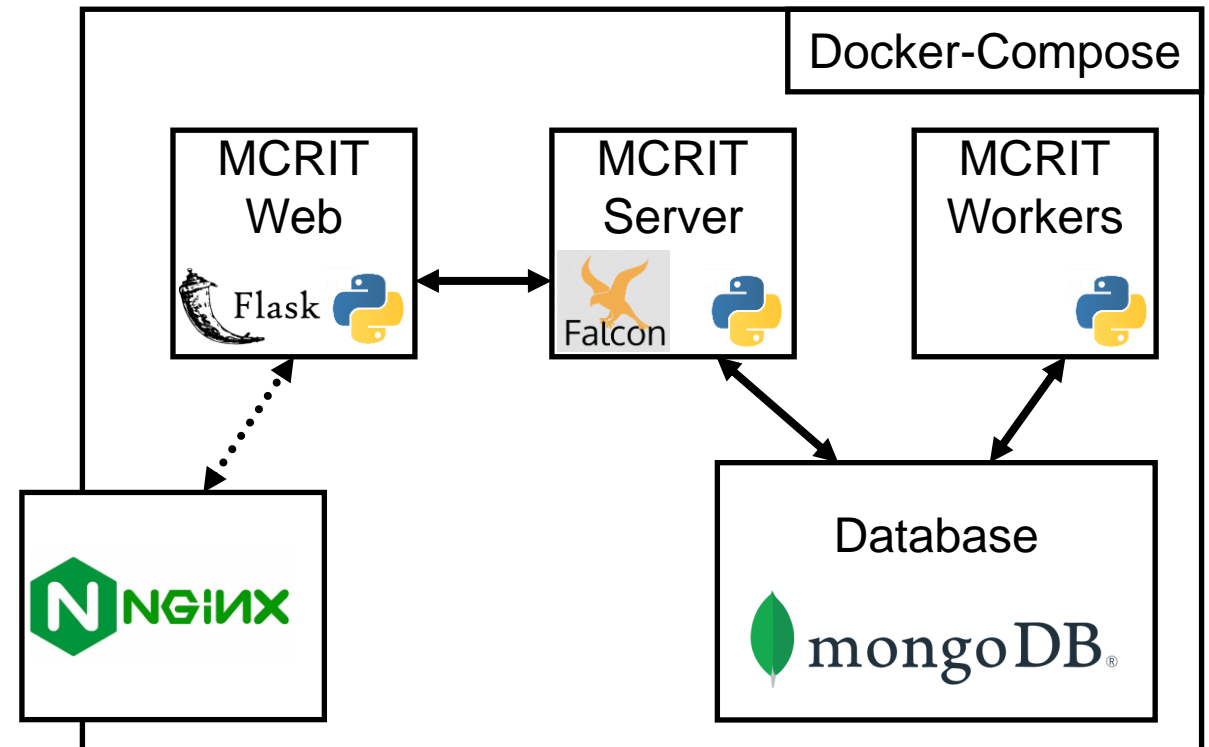
Family	Version	Score	SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct	Frequency	Uniq	
win_romeos	2014-07-07-alfa	4498	4ec0214b	eff542ac8e...0x00400000	32	311	247	247	71	98 97	47	59	21.49%
win_hardrain		909	c3b1af35	2cc3b5f2df...0x00340000	32	289	138	103	66	50 37	13	13	0.00%
win_op_blockbuster	2015-04-08	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] eff542ac8e37db48821cb4e5a7d95c044fff27557763de3a891b40eb52cc55 @ 0x406770 | win_romeos

MCRIT

Dockerized 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

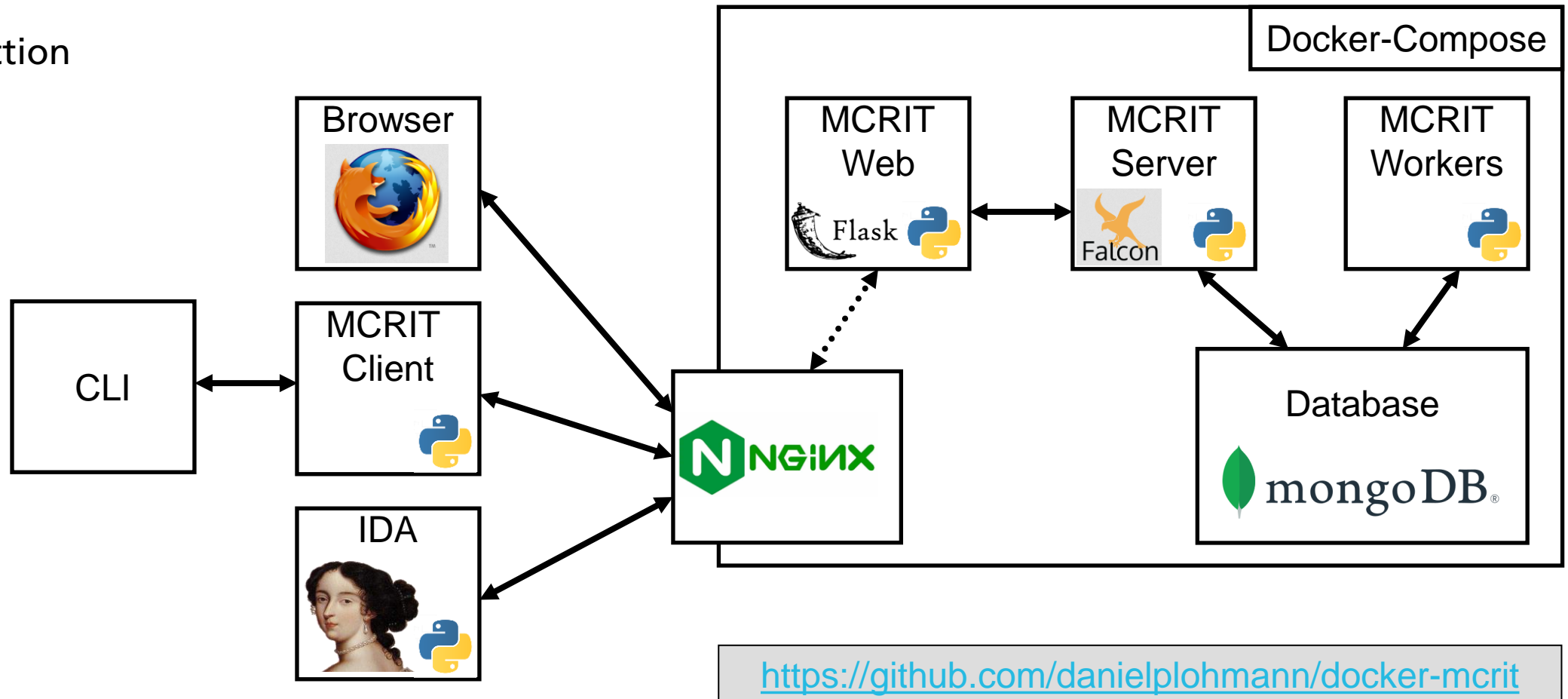




<https://github.com/danielplohmann/docker-mcrit>

MCRIT

Interaction with MCRIT





































■ Interaction





Explore Analyze Data pnx

Overview of Families

* ^	Family	Samples	Functions	Library	
0	Unnamed	38	55170	+	   
1	msvcrt	327	310944	☑	   
2	win.wastedlocker	10	1620	+	   
3	win.isfb	118	74792	+	   
4	win.juicy_potato	1	1688	+	   
5	win.lyposit	2	1095	+	   
6	win.horus_eyes_rat	0	0	+	   
7	win.bumblebee	23	157017	+	   
8	win.cobra	53	14954	+	   

Compare Samples 1vsN

wannacry

* ^	SHA256	Family	Version	Filename	Bitness	Functions	Library
626	e458d473	win.wannacryptor	vt-2017-05-05	0345782378ee7a8b48c2...7366_dump_0x00400000	32	922	✖ ⓘ
1380	2be58051	win.wannacryptor	2017-02-09	3e6de9e2baacf9309496...eed9_dump_0x00400000	32	450	✖ ⓘ
4092	ca29de1d	win.wannacryptor	2017-03-19	ca29de1dc8817868c93e...1f52ba469c8_unpacked	32	907	✖ ⓘ
4805	d181360a	win.wannacryptor	vt-2017-05-12	b9c5d4339809e0ad9a00...1c25_dump_0x00400000	32	926	✖ ⓘ
4955	d36a4116	win.wannacryptor	vt-2017-05-12	ed01ebfbc9eb5bbea545...41aa_dump_0x00400000	32	926	✖ ⓘ
5828	6611cc9e	win.wannacryptor	2017-03-19	ca29de1dc8817868c93e...69c8_dump_0x00400000	32	590	✖ ⓘ

<< < 1 > >>

Force rematch

Minhash
Matching:

Standard

Compare

Best Family Matches

total: 5, showing: 1 - 5 (filtered: 966)

Filter results to (nonlib) direct score

Filter results to (nonlib) frequency score

only show families with unique matches

exclude own family

filter clear

* SHA256	Version	* SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct	Frequency	Uniq
ca29de1d	2017-03-19	4092	ca29de1dc8...8_unpacked	32	907	161	114	14	69 68	34 34	13.93%
3d3c3bf0	2017-01-30	3490	5793b30729...0x00400000	32	1288	76	62	8	35 34	9 9	0.00%
6386ae55	2013-03-07	4357	afe3dd68bd...0x00400000	32	337	48	37	5	31 31	8 8	0.00%
31e27637		5695	a1c483b0ee...0x002c0000	32	186	30	24	3	11 11	7 7	0.00%
d594b683		2923	93e13ffd2a...0x006e0000	32	166	58	44	3	27 27	7 7	0.00%

<< < 1 > >>

Compare

MCRIT

New Features in Release 1.2

- Version 1.0 published in April 2023 at Botconf
 - Matching fully implemented, basic WebUI
- Version 1.2 – Virus Bulletin Release
 - Focus on usability improvements
 - Polishing of the WebUI – filters, ...
 - Significant extension of the IDA plugin
 - Just-in-time matching information for currently shown function
 - Import of matching and label information
 - Prototype LinkHunt feature that makes use of ICFG relationship information

Use Cases and Case Studies

Example Use Cases

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



Current stats of my demo instance:

num_samples	7372
num_families	1767
num_functions	9637676
num_pichashes	2083507

8 Cores, 32GB RAM

Use Cases and Case Studies

Malware Family Identification

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.

Query Sample

Drop file or click here to import

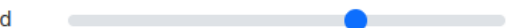
45c317_lockbit_green.exe

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

Best Family Matches

total: 1070, showing: 1 - 10

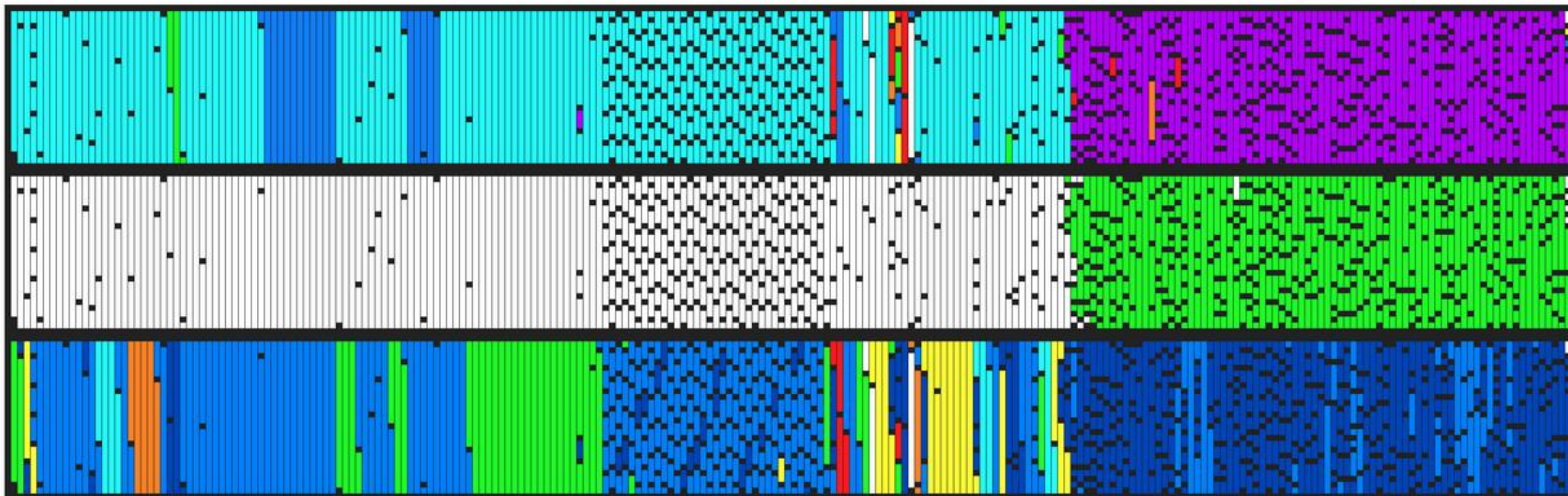
✱	Version	✱	SHA256	Filename	Bitness	FNs	Min#	Pic#	Lib	Direct	Frequency		
win.meow ▼		2172 ▼	222e2b91	222e2b91f5...3_unpacked	32	702	461	126	220	66	71	41	53
win.conti ▼	2021-02-04	5041 ▼	a5751a46	a5751a4676...6_unpacked	32	736	516	183	256	59	58	28	36
win.scarecrow ▼		6199 ▼	bcf49782	bcf49782d7...a_unpacked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

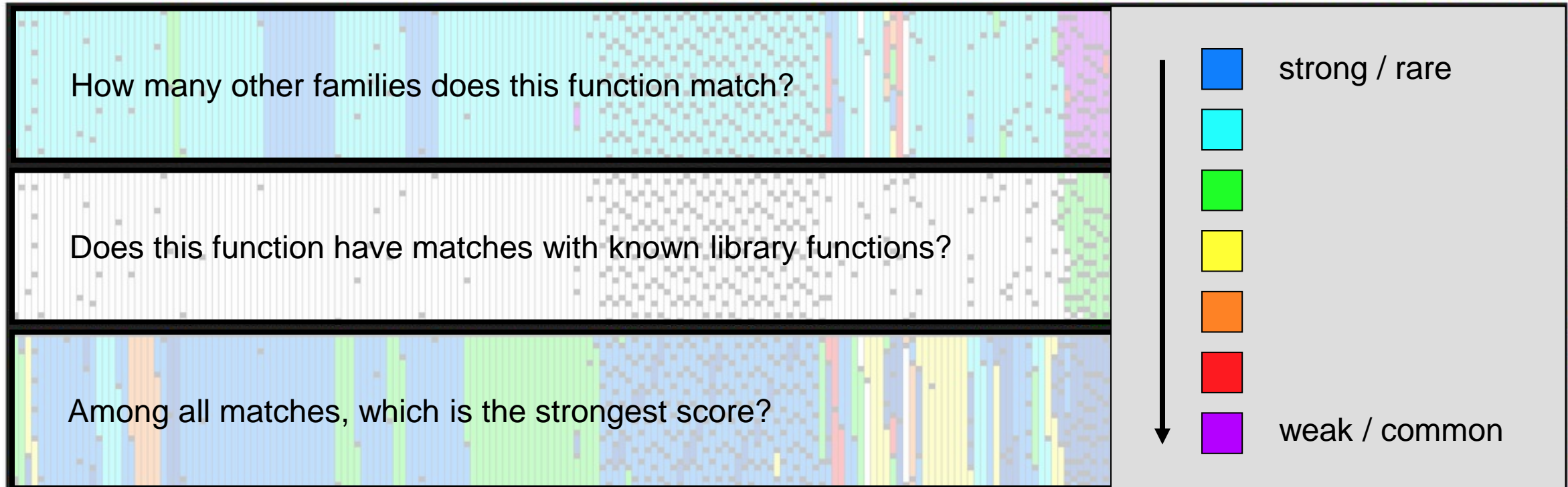


Best Family Matches

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.

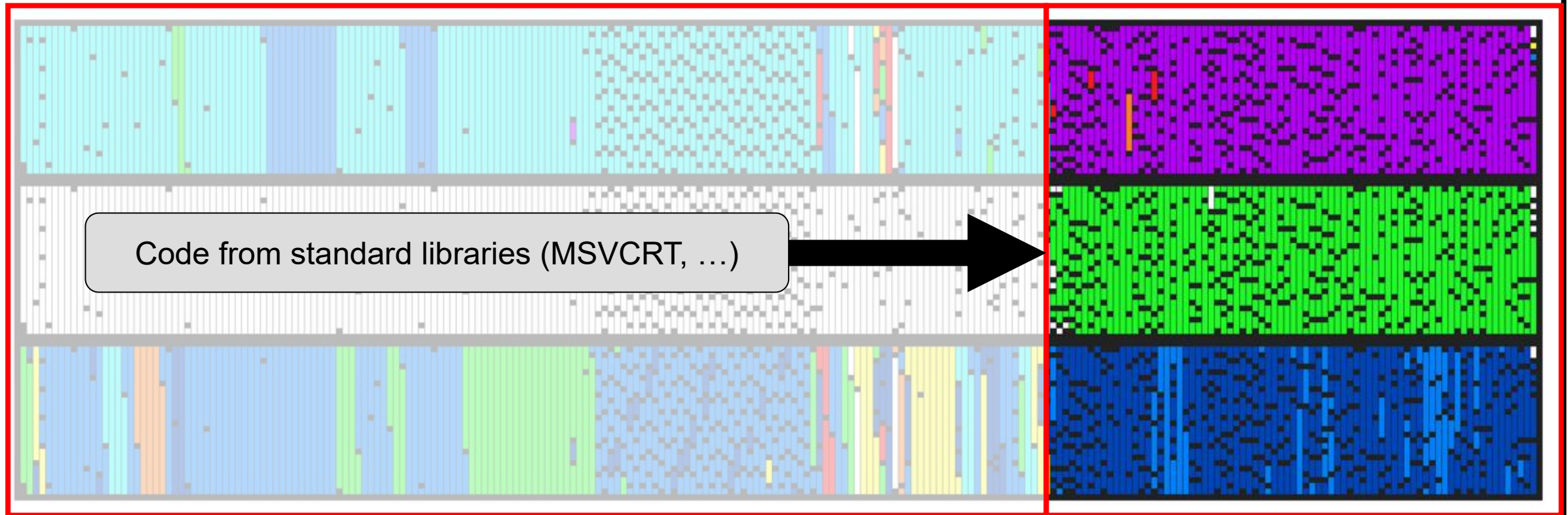


Best Family Matches

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.

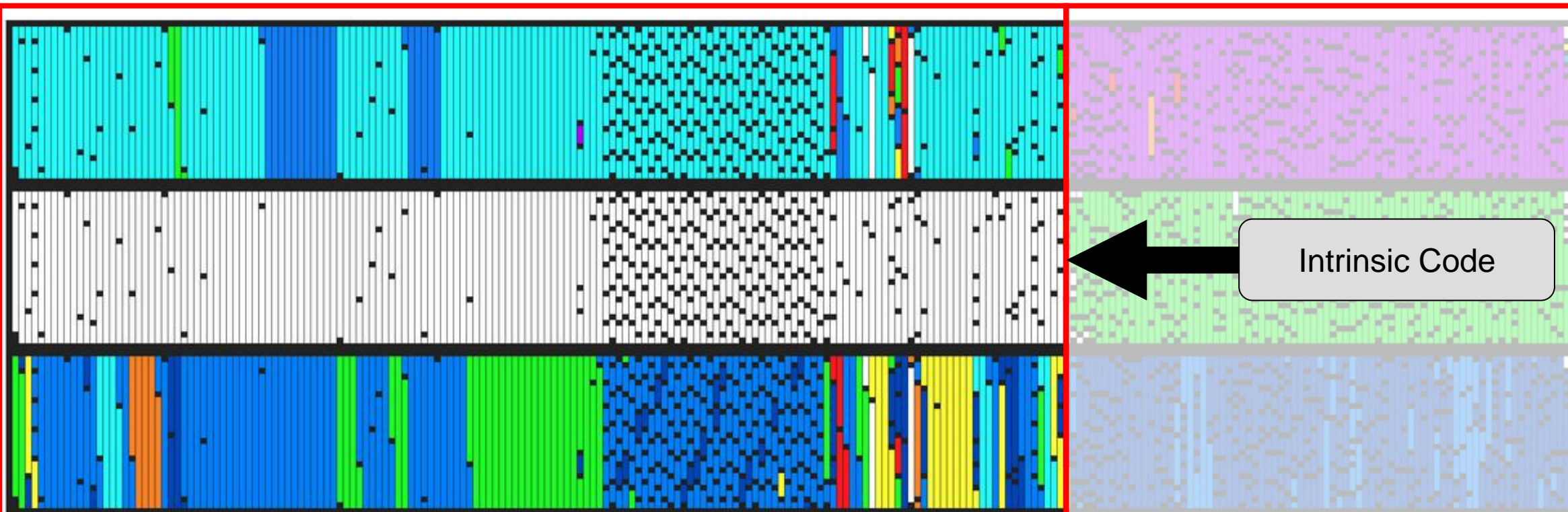


Best Family Matches

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.



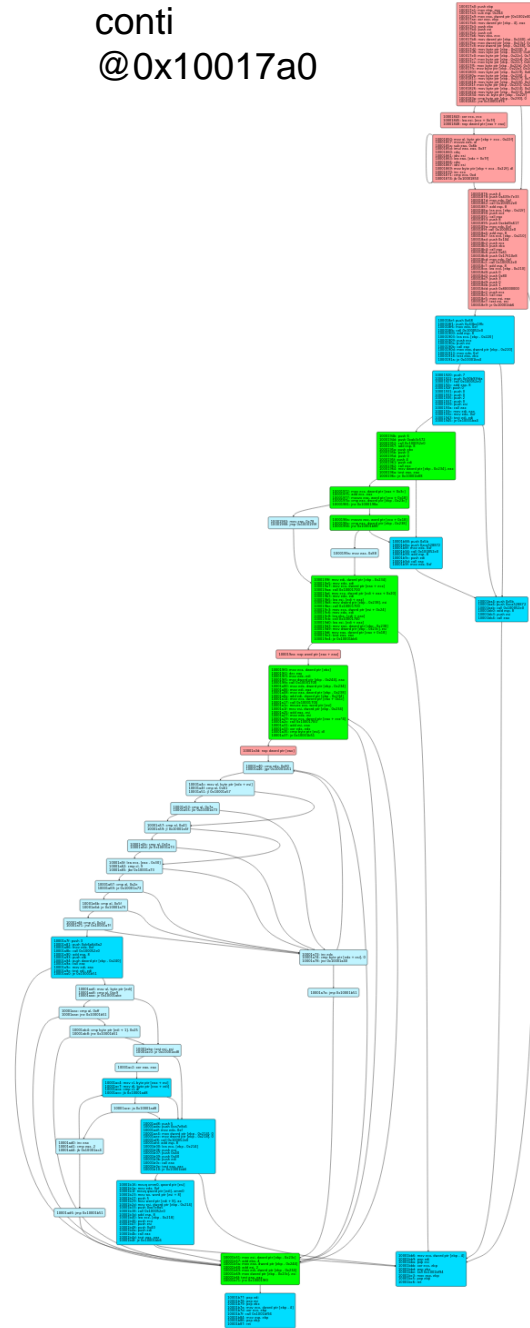
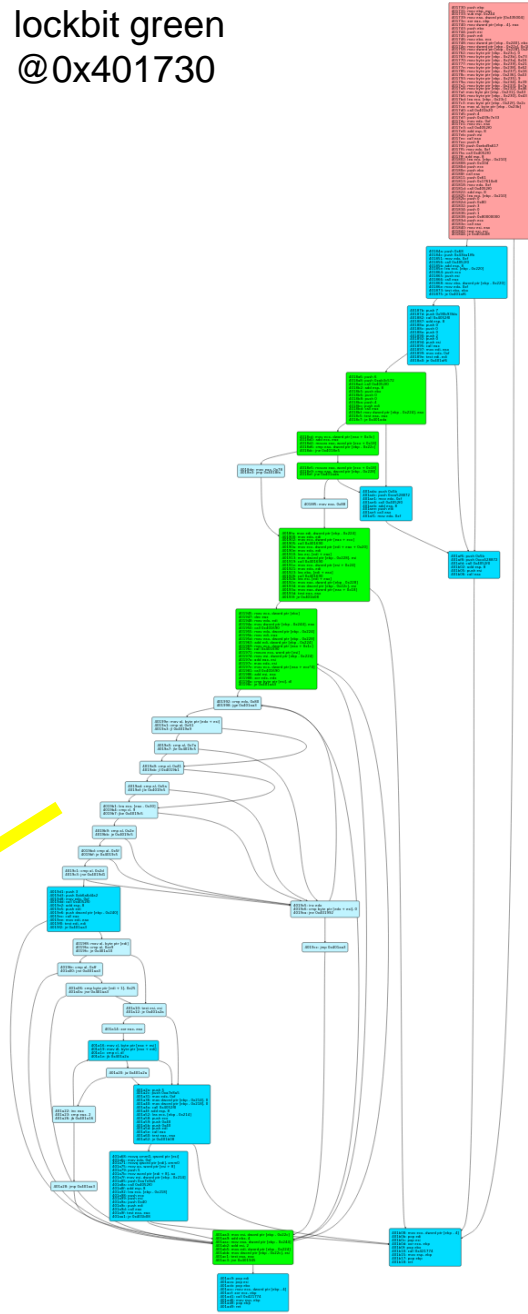
Best Family Match

total: 1070, showing: 1 - 10

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

lockbit green
@0x401730

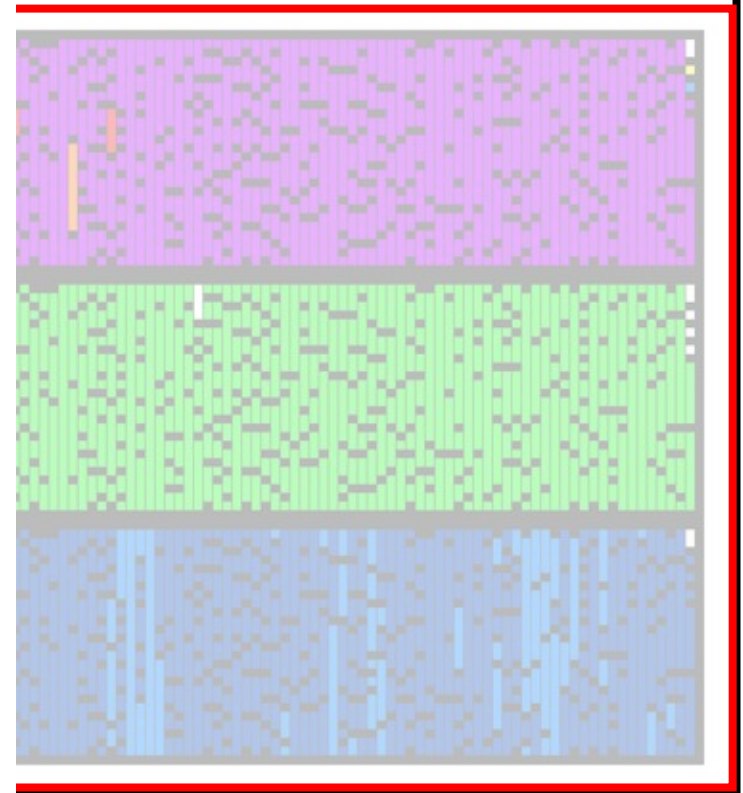
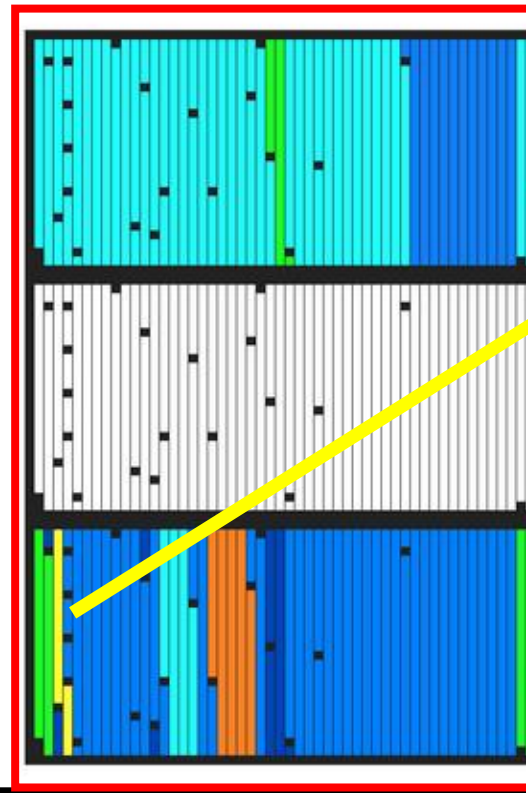
conti
@0x10017a0



[go to top](#)

Min#	Pic#	Lib	Direct	Frequency
461	126	220	66	71
516	183	256	59	58
582	271	334	61	51

Showing: foreign family match frequ



Use Cases and Case Studies

Clustering

Example Use Cases

Clustering: KEYPLUG and Friends

- APThursday @ FKIE
 - Volunteer / Enthusiast Group looking at APT activity
- VT Retrohunted shellcode loader used in samples publicly reported
 - Found / unpacked 13 hits
- Question:
 - What families are these samples?



[1] <https://go.recordedfuture.com/hubfs/reports/cta-2023-0330.pdf>

[2] <https://www.mandiant.com/resources/blog/apt41-us-state-governments>

Example Use Cases

Clustering: KEYPLUG and Friends

- Running them through MCRIT
- Labeling of clusters / samples via other sources



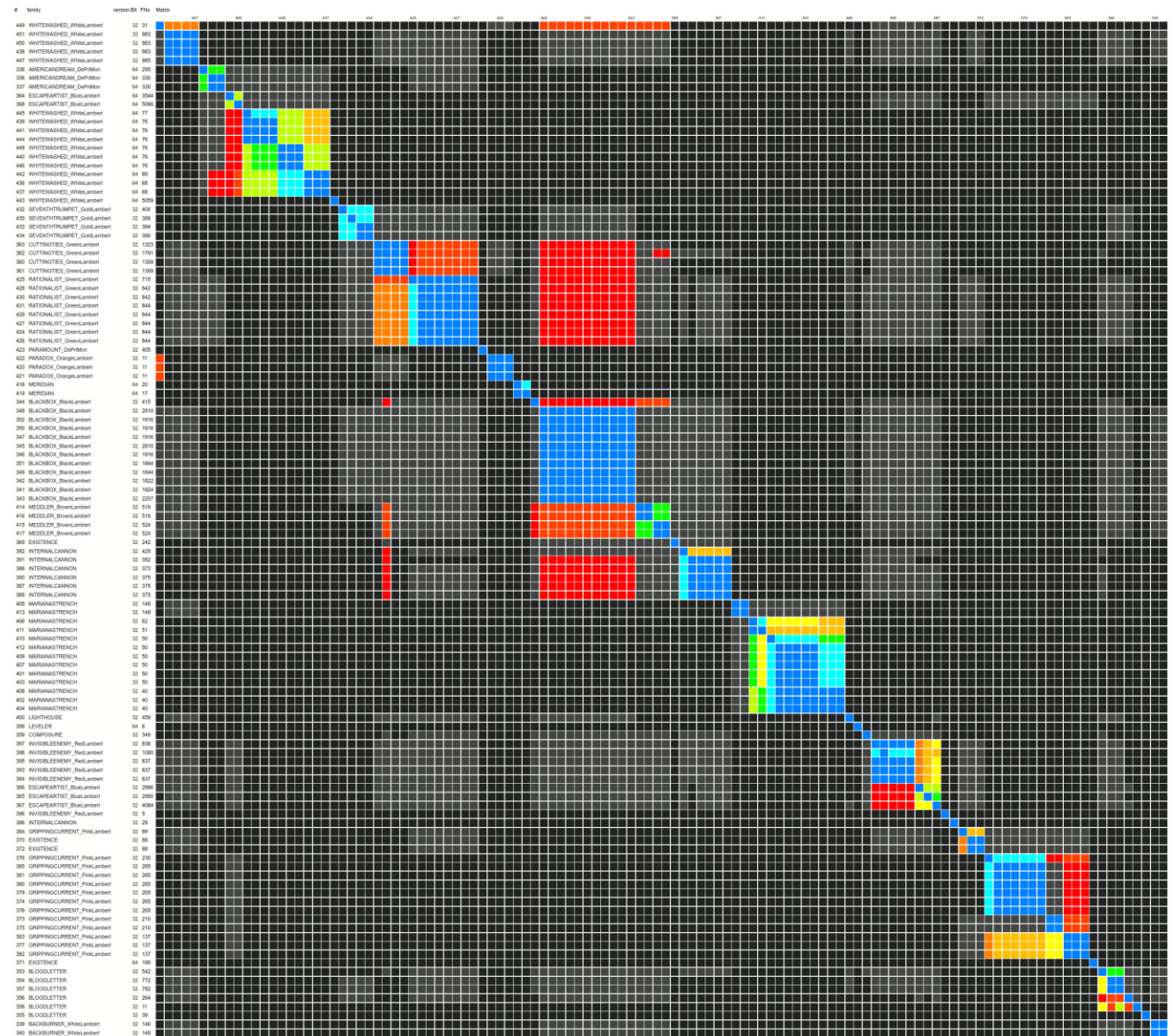
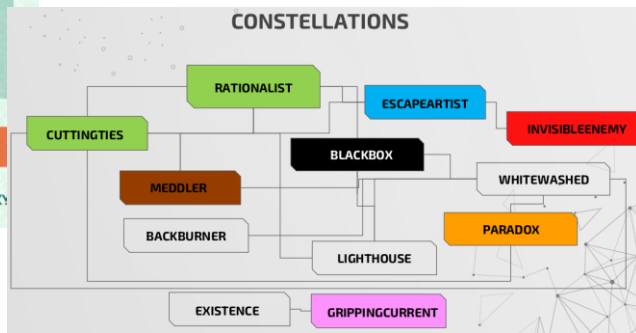
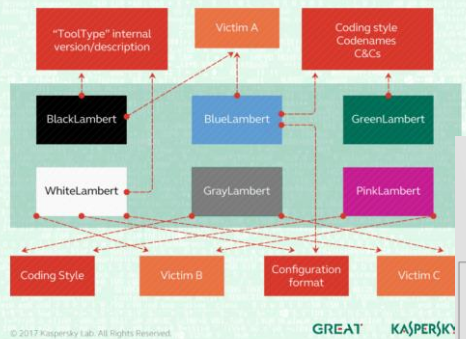
Example Use Cases

Clustering: „Bright Constellation“

- Oh colorful Lamberts...
 - Write-ups by various institutions
 - revisited in Sep 2022 by Greg Lesnewich who shared this data set with me (THX!)
- Experiment: Cluster verification using MCRIT

An overview of connections between the Lambert families

Different Lambert families share code, data formats command and control servers and victims.

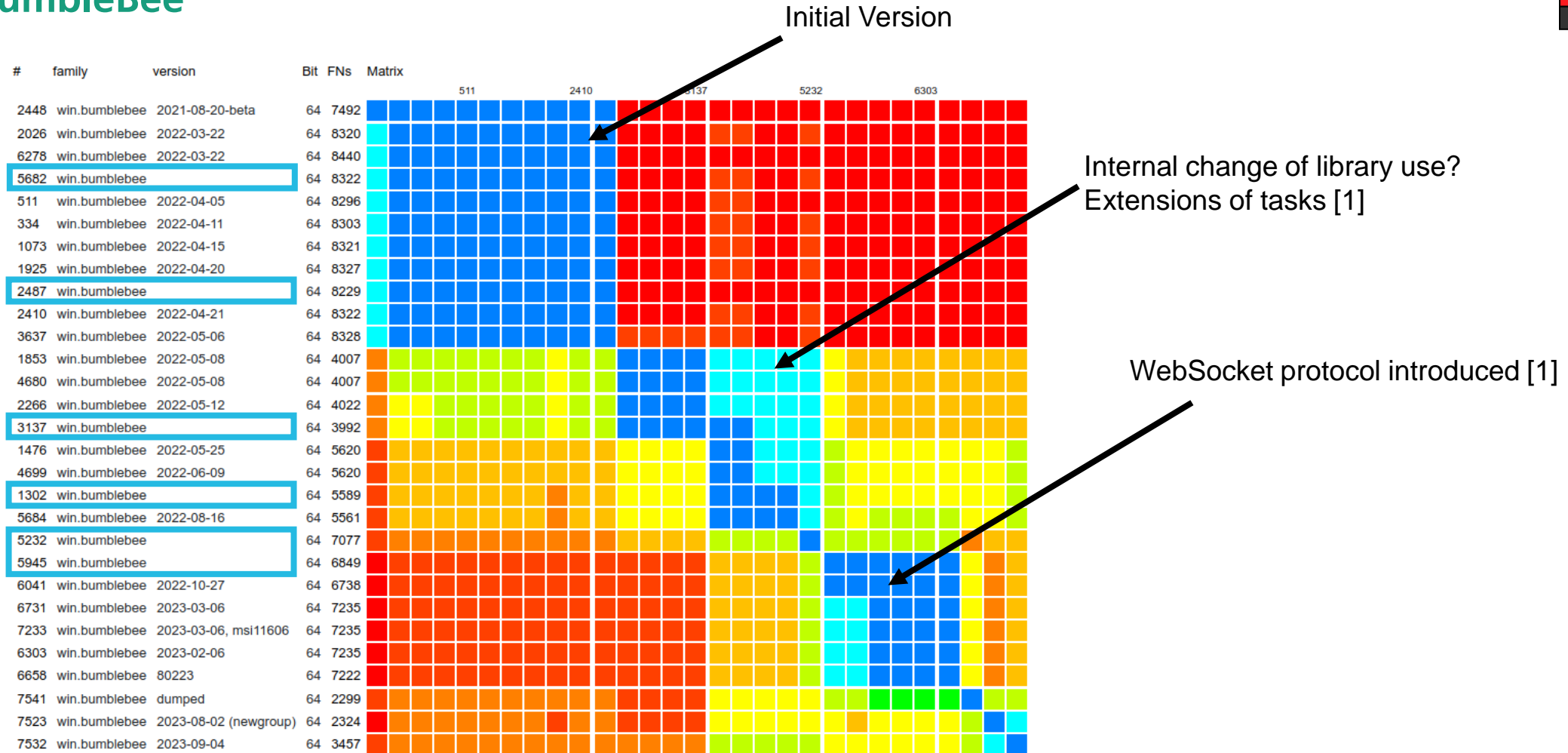


■ MCRIT Processing time: ~35min.

[1] <https://securelist.com/unraveling-the-lamberts-toolkit/77990/>
 [2] https://www.youtube.com/watch?v=aaV7UieJ_I4

Example Use Cases

Clustering: BumbleBee



■ MCRIT Processing time: ~20min

[1] "Tracking Bumblebee's Development". Suweera De Souza, 2023.

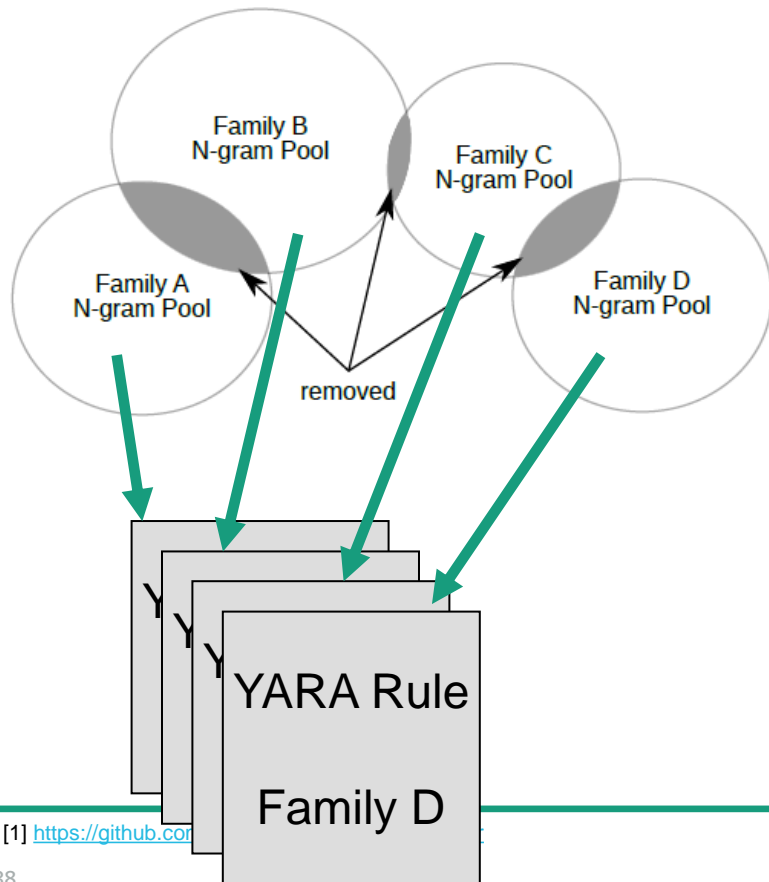
Use Cases and Case Studies

Isolation of Unique Family Code

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.

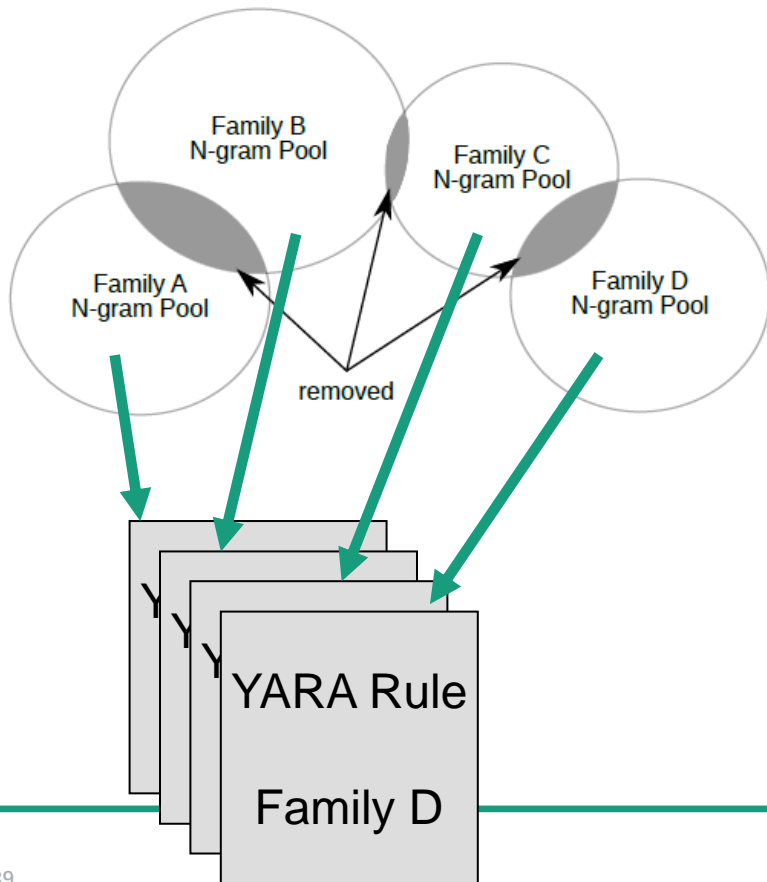
Sample ID	Total Blocks	Characteristic Blocks	Unique Blocks
1111	1104	965 (87.41%)	0 (0.00%)
1342	1178	1056 (89.64%)	567 (48.13%)
1343	1106	967 (87.43%)	1 (0.09%)
1519	746	548 (73.46%)	68 (9.12%)
1968	8020	3423 (42.68%)	188 (2.34%)
2217	867	743 (85.70%)	266 (30.68%)
3729	1190	1041 (87.48%)	32 (2.69%)
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%)

[1] <https://github.com>

Example Use Cases

Isolation of Unique Family Code

- Essentially like YARA-Signator, but with basic blocks



Explore Unique Blocks

Filter blocks to

filter

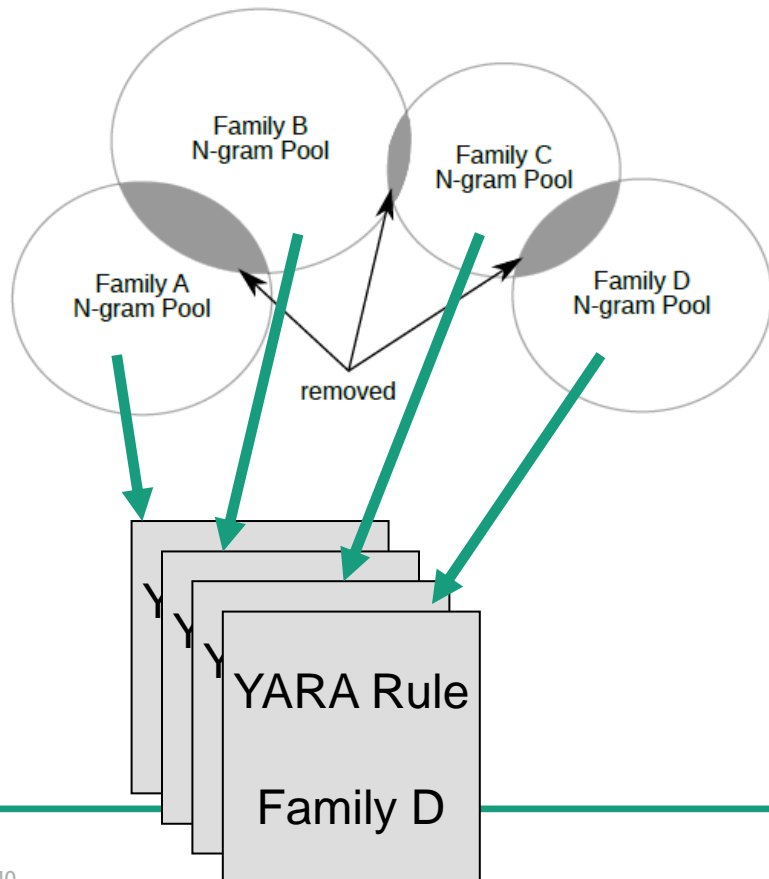
total: 10028, showing: 1 - 100

Score	PicBlockHash	Samples	Instructions	Function ID	Block
97.14	0xcadb40567ec2656	19 / 19	6	470589	<pre> /* picblockhash: 0xcadb40567ec2656 * 6a09 push 9 * ff35fc8d4100 push dword ptr [0x418dfc] * ff1554244100 call dword ptr [0x412454] * ff35fc8d4100 push dword ptr [0x418dfc] * ff1588244100 call dword ptr [0x412488] * eb38 jmp 0x41157f */ { 6a09 ff35?????? ff15?????? ff35?????? ff15?????? eb?? } </pre>
83.16	0x1bb7e1ed48b1d751	15 / 19	7	470361	<pre> /* picblockhash: 0x1bb7e1ed48b1d751 * 53 push ebx * 53 push ebx * 56 push esi * 68ed524000 push 0x4052ed * 53 push ebx * 53 push ebx * ffd7 call edi */ { 53 53 56 68?????? 53 53 ffd7 } </pre>
80.30	0x38bb1b656ac38756	15 / 19	6	470428	<pre> /* picblockhash: 0x38bb1b656ac38756 * 6890374100 push 0x413790 * 6874374100 push 0x413774 * ffd7 call edi * 50 push eax * ffd6 call esi * a3a48a4100 mov dword ptr [0x418aa4], eax */ { 68?????? 68?????? ffd7 50 ffd6 a3?????? } </pre>

Example Use Cases

Isolation of Unique Family Code

- Essentially like YARA-Signator, but with basic blocks



Proposed YARA rule

Copy rule to clipboard! [🔗](#)

```
rule mcrif_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]
    * 037d0800 | 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: 0x3fc01ffe2622434d - 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
    */
  }
```

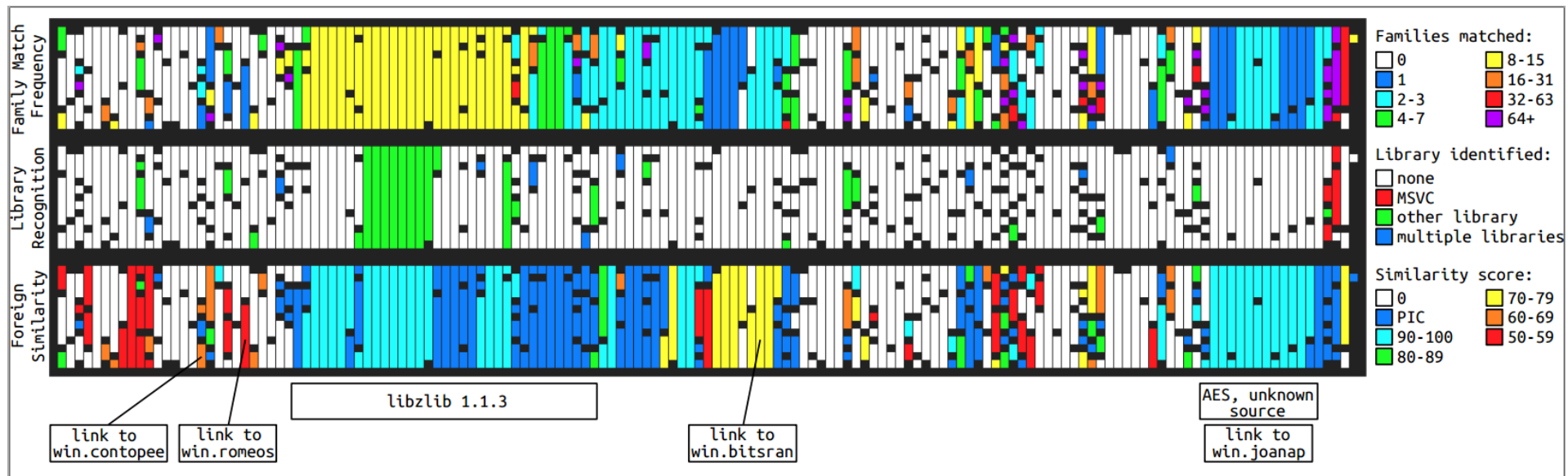
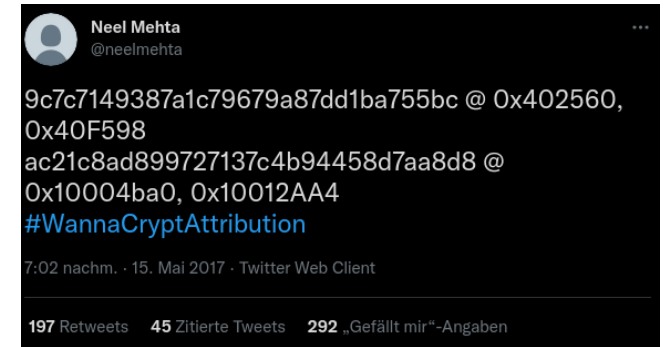

Use Cases and Case Studies

Lead Generation

Example Use Cases

Lead Generation: Reconstructing the WannaCry Hunt

- 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.

Example Use Cases

Lead Generation: Reconstructing the WannaCry Hunt

Function Matches

selection: 49, showing: 1 - 49 (filtered: 64916)

Filter results to max score (0-100)

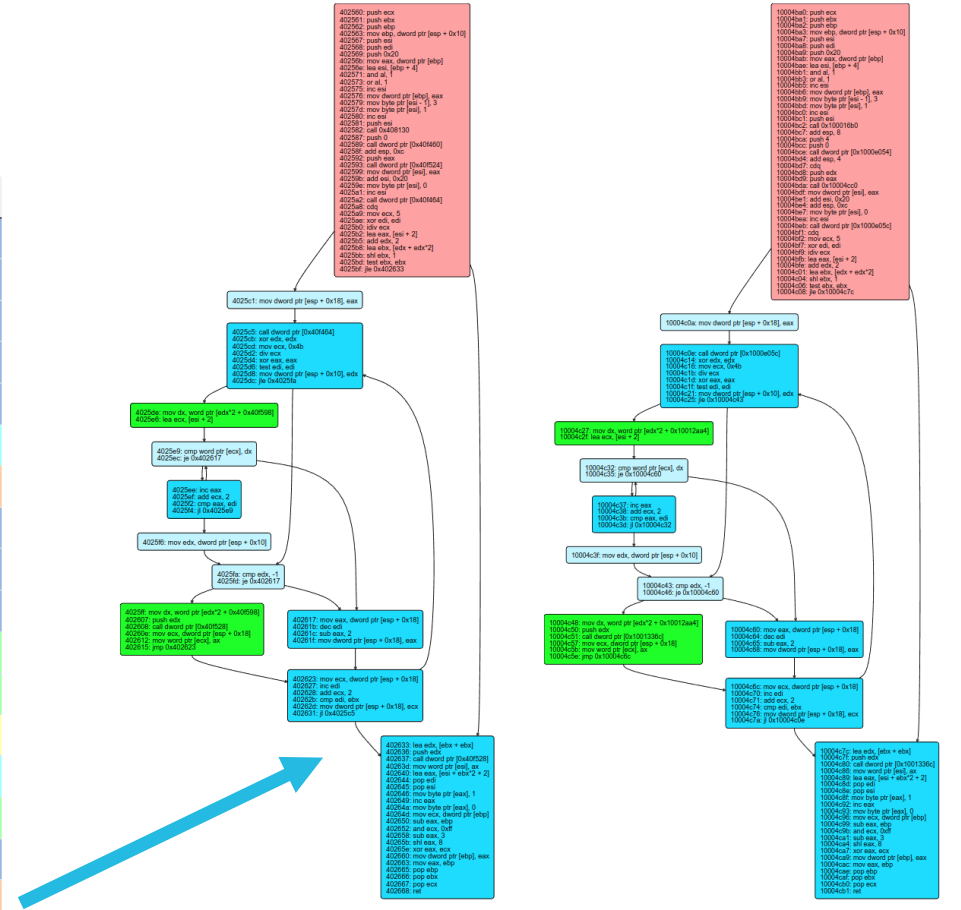
Filter to offset

exclude functions with library hits exclude PIC hits unique only

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

Function CFGs

Show Loop Boundaries Enable Tooltip



Example Use Cases

Lead Generation: New LinkHunt Feature

■ LinkHunt

- Consider ICFG relationship of functions in a matched sample
 - Do multiple connected functions match another family?
 - If yes, highlight such clusters
- Otherwise rank functions by combined
 - Matching score
 - Size
 - Position in binary (front)


Example Use Cases

Lead Generation: Recent Lazarus Reporting


ESET RESEARCH

Lazarus luring employees with trojanized coding challenges: The case of a Spanish aerospace company

While analyzing a Lazarus attack luring employees of an aerospace company, ESET researchers discovered a publicly undocumented backdoor

 Peter Kálnai

29 Sep 2023 • 29 min. read



ESET researchers have uncovered a Lazarus attack against an aerospace company in Spain, where the group deployed several tools, most notably a publicly undocumented backdoor we named LightlessCan. Lazarus operators obtained initial access to the company's network last year after a successful spearphishing campaign, masquerading as a recruiter for Meta – the company behind Facebook, Instagram, and WhatsApp.

The fake recruiter contacted the victim via LinkedIn Messaging, a feature within the LinkedIn professional social networking platform, and sent two coding challenges required as part of a hiring process, which the victim downloaded and executed on a company device. The first challenge is a very basic project that displays the text "Hello, World!"; the second one prints a Fibonacci sequence – a series of numbers in which each number is the sum of the two preceding ones. ESET Research was able to reconstruct the initial access steps and analyze the toolset used by Lazarus thanks to cooperation with the affected aerospace company.

In this blogpost, we describe the method of infiltration and the tools deployed during this Lazarus attack. We will also present some of our findings about this attack at the [Virus Bulletin conference](#) on October 4, 2023.

Execution chain 1: miniBlindingCan

One of the payloads downloaded and executed by NickelLoader is miniBlindingCan, a simplified version of the group's flagship BlindingCan RAT. It was reported for the first time by Mandiant in September 2022, under the name AIRDRY.V2.

* SHA256	Version	SHA256	Filename	Bitness	FNS	Min#	Pic#	Lib	Direct	Frequency	Uniq
win.blindingcan	2020-05-19	3689	58027c80 58027c80c6...d_unpacked	64	348	200	188	182	51	15	9 5 1.37%
win.snatchcrypto		986	bde53bd6 156d33cd77...0180000000	64	2087	221	206	209	52	8	8 2 0.40%
win.neddnlloader		7368	e05344f6 454734dca5...6_unpacked	64	263	165	155	150	39	16	8 7 2.04%
win.cloudburst		6367	e1ecf0f7 e1ecf0f7bd...0_unpacked	64	2996	218	207	211	53	5	7 0 0.00%
win.banpolmex		7490	21c83fe2 21c83fe249...f_unpacked	64	2165	220	212	210	53	8	7 1 0.00%
Unnamed		7428	333b4da6 333b4da636...218b52c979	64	5070	218	207	209	52	8	7 1 0.00%
win.soul	soulsearcher	1208	844b9ce5 1af5252cad...0180000000	64	634	219	213	209	51	8	7 1 0.00%
win.unidentified_087		795	1af5252c 1af5252cad...f_unpacked	64	635	219	213	209	51	8	7 1 0.00%
win.bookcodesrat		7277	7695e619 7695e619cb...e_unpacked	64	800	219	213	208	51	8	7 1 0.00%
win.pocodown	2018-07-04	4028	e4a99f7b b40909ac0b...0180000000	64	9890	221	206	213	50	7	7 1 0.00%

■ MCRIT Processing time:


- Disassembly: 6sec -> 950 functions
- Matching: 21sec

[win.blindingcan \(Back to overview\)](#)




aka: AIRDRY, ZetaNile
Actor(s): Lazarus Group

[win.snatchcrypto \(Back to overview\)](#)



Actor(s): Lazarus Group

[win.neddnlloader \(Back to overview\)](#)



Actor(s): Lazarus Group



[1] <https://www.welivesecurity.com/en/eset-research/lazarus-luring-employees-trojanized-coding-challenges-case-spanish-aerospace-company/>

[2] f69198afcd237af265f1f48203d0db8f5f07cdf53a0f410e4d49f77434b53cff

Example Use Cases

Lead Generation: LinkHunt in Action

Link Clusters

These are clusters of matches with another single family where all functions have a direct code CFG relationship (calls, jumps, ...) among them. Bold matches are also unique with this other family.

Rank	Clusterscore	Best Linkscore	Family	Cluster Size	Unique Matches	Functions
1	163.93	90.58	⚡(472) win.blindingcan	3	1	0x18000be80 ⚡ 0x180002d14 ⚡ 0x1800027d0 ⚡
2	112.67	73.84	⚡(27) win.neddnloader	2	0	0x180001000 ⚡ 0x180001444 ⚡

Best Individual Links

Family-unique matches in bold.

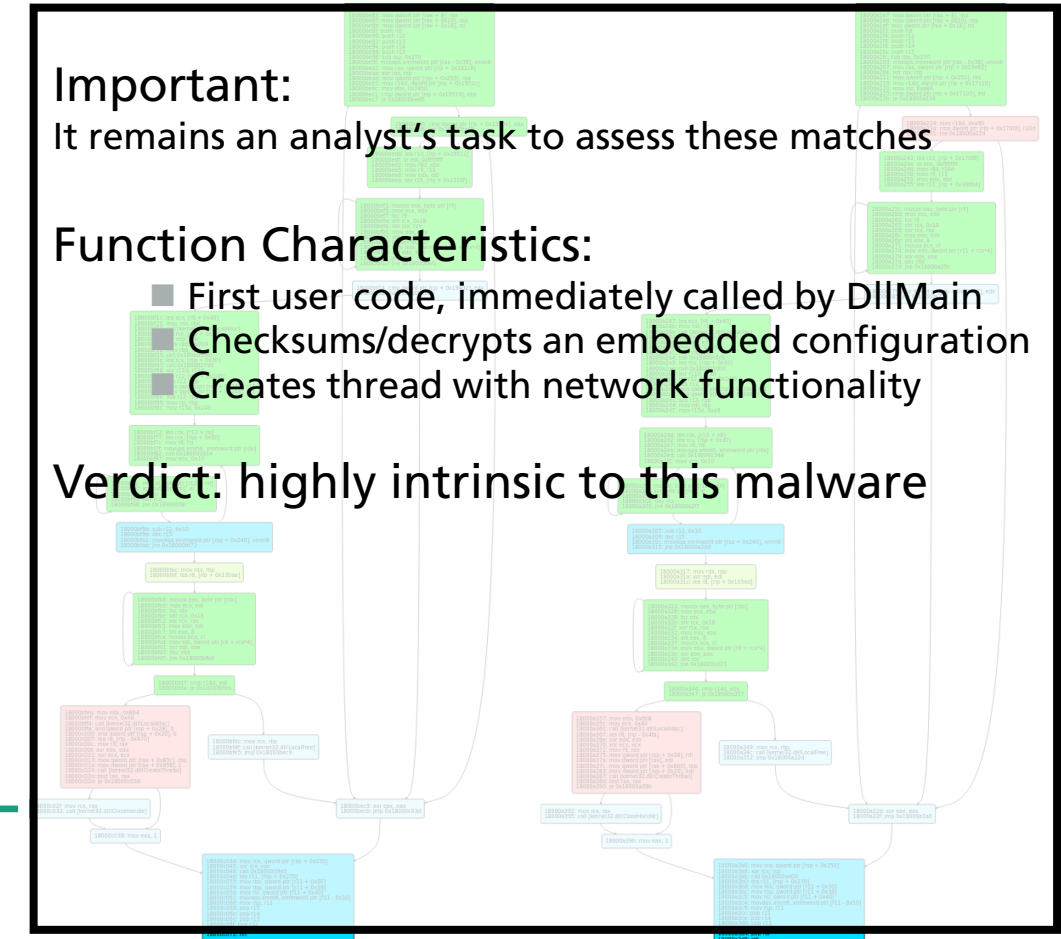
Rank	Linkscore	Function ID	Offset	num_bytes	Family	Sample ID	Function ID	Match Score
1	90.58	9595275	0x1800027d0	1347	⚡(472) win.blindingcan	3689	4087967 ⚡	100.00
2	90.53	9595279	0x180003e38	414	⚡(472) win.blindingcan	3689	4087998 ⚡	100.00
3	90.39	9595321	0x18000b5e8	381	⚡(472) win.blindingcan	3689	4088036 ⚡	100.00
4	86.20	9595275	0x1800027d0	1347	⚡(27) win.neddnloader	7368	9197730 ⚡	93.75
5	81.88	9595313	0x18000a7d0	171	⚡(472) win.blindingcan	3689	4088045 ⚡	100.00
6	80.70	9595319	0x18000b45c	154	⚡(472) win.blindingcan	3689	4088034 ⚡	100.00
7	75.99	9595285	0x180005330	79	⚡(474) win.spyder	1417	1376061 ⚡	100.00
8	75.99	9595285	0x180005330	79	⚡(27) win.neddnloader	7368	9197740 ⚡	100.00
9	73.84	9595257	0x180001444	1041	⚡(27) win.neddnloader	7368	9197762 ⚡	76.56
10	71.60	9595324	0x18000be80	499	⚡(472) win.blindingcan	3689	4088049 ⚡	73.44
11	67.41	9595294	0x180005dd0	255	⚡(34) win.colony	480	443267 ⚡	71.88
12	66.18	9595257	0x180001444	1041	⚡(1710) win.httpsuploader	7280	8991609 ⚡	65.62
13	64.44	9595303	0x180006b2c	142	⚡(191) win.easynight	4869	5407333 ⚡	78.12
14	63.03	9595294	0x180005dd0	255	⚡(1040) win.pocodown	4028	4442855 ⚡	65.62
15	59.59	9595285	0x180005330	79	⚡(1737) win.unidentified_106	7366	9190548 ⚡	76.56
16	57.88	9595303	0x180006b2c	142	⚡(1652) win.r77	6741	7855204 ⚡	68.75
17	57.88	9595303	0x180006b2c	142	⚡(0) Unnamed	7550	9583145 ⚡	68.75

Function Comparison: 9595324 vs. 4088049

Architecture	intel	intel
Binweight	499	498
Family ID	1791	472
Family Name	winid.lazarus	win.blindingcan
Sample ID	7560	3689
Short Sha256	f69198af	58027c80
Function ID	9595324	4088049
Function Name		
Num Blocks	18	18
Num Instructions	126	128
Offset	0x18000be80	0x18000a1e4
PicHash	0xb38855d9b62981d5 (1 ⚡, 1 ⚡, 1 ⚡)	0xdf6b227d8d2df497 (1 ⚡, 1 ⚡, 1 ⚡)
Matching Score	73.44	

Function CFGs

Show Cycles Show Loops Show Loop Boundaries Enable Tooltip



Use Cases and Case Studies

IDA Plugin, Label Transfer, Reference Data

Example Use Cases

IDA Plugin

The screenshot displays the IDA Pro interface with the following components:

- Left Panel (Call Graph):** Shows a function call graph for `sub_140001890`. The graph includes nodes for `loc_1400018C0`, `loc_14000193C`, `loc_14000192C`, `loc_1400018FD`, `loc_14000191C`, `loc_14000192C`, `loc_1400018F3`, and `loc_14000188E`.
- Central Panel:**
 - Activity Info:** 2023-09-29T20-24-24Z - Success! Received all remote FamilyEntries. Remote server: <https://mcr.it.malpedia.io/api> -- 1.1.7 - No statistics. Remote sample: 7526 (tmp.lazarus_downloaded --)
 - Metadata:** SHA256: 4574549877c7c1613517a01f25160082db11149d64c54ef003e60bbe6305923; Architecture: intel; Bitness: 64 bit; ImageBase: 0x140001000; Functions: 820 (leaf: 172, recursive: 0); Instructions: 44081; Code Size: 176629 bytes; Family: tmp.lazarus_downloaded; Version: NO; Library: NO.
 - Block Scope / Function Scope / Function Overview:** Matches for Function: 0x140001890 - 8 families, 22 samples, 3 functions (19 filtered).
 - Function Matches Table:**

ID	SHA256	Sample	Family	Version	Plc#	Score	Lib
1	9450718 6c1f388b	7518	tmp.lazarus_downloaded		YES	100	NO
2	9455657 45745498	7526	tmp.lazarus_downloaded		YES	100	NO
3	9252235 f603713b	7445	win.webbytea		NO	100	NO
 - Names from Matched Functions Table:**

ID	Score	user	Function Label
- Right Panel (Assembly View):** Shows assembly code for `sub_140001890` with instructions highlighted in green and blue. The code includes instructions like `mov byte ptr [rsp+0x18], r8b`, `cmp dword ptr [rsp+0x20], 0`, and `movzx eax, byte ptr [rsp+0x26]`.

Example Use Cases

Label Transfer

The screenshot displays the Immunity Debugger interface for a sample named MCRIT4IDA v0.1. The main window shows assembly code for the `init_cnc_packet` function, with a control flow graph overlaid. The graph highlights a branch from `loc_4025C5` to `loc_402633`. The `loc_4025C5` block contains the following assembly:

```

loc_4025C5:
call    ds:rand
xor     edx, edx
mov     ecx, 4Bh ; 'K'
div     ecx
xor     eax, eax
test    edi, edi
mov     [esp+14h+var_4], edx
jle     short loc_4025FA

```

The `loc_402633` block contains:

```

jle     short loc_402633

```

On the right, the 'Function Matches' panel is open, showing a table of matches for the function `0x402560`:

ID	SHA256	Sample	Family	Version	Pic#	Min#	Lib
1 1158	3e6de9e2	2	win.wannacry		YES	100	NO
2 85	2c4e5ba7	0	win.contopee		NO	67	NO

Below the table, the 'Names from Matched Functions' section shows a list of function labels:

ID	Score	user	Function Label
1 1158	100	anonymous	init_cnc_packet

The 'init_cnc_packet' label is highlighted in blue in the original image.

Example Use Cases

Reference Data

- MCRIT-Data [1]
 - Parsed *.lib/obj files from various sources
 - Long-term goal: cover commonly encountered statically linked code
 - Currently: compilers (MSVC, MinGW, Golang, Nim, Rust) and libraries (aPLib, zlib)
 - 890k functions **with symbols**
 - Ready to use with MCRIT via Import functionality

[1] <https://github.com/danielplohmann/mcrit-data>

Summary & Outlook

Summary and Outlook

MCRIT

- Minhash-based Code Relationship & Investigation Toolkit (MCRIT)
 - A framework for quasi-identical and fuzzy 1:n code matching
 - Use Cases: Code identification & library filtering, hunting, label transfer, ...
 - Open Source, convenient deployment via Docker [1]

- Outlook
 - Scalability improvements
 - Further usability refinement based on user feedback

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

Thank you for your attention!

Dr. Daniel Plohmann

daniel.plohmann@fkie.fraunhofer.de



@push_pnx