FN_FUZZY: FAST MULTIPLE BINARY DIFFING TRIAGE WITH IDA

Takahiro Haruyama

Threat Analysis Unit

Carbon Black

WHO AM I?

- Takahiro Haruyama (@cci_forensics)
  - Senior Threat Researcher with Carbon Black's Threat Analysis Unit (TAU)
  - Reverse-engineering cyber espionage malware linked to PRC/Russia/DPRK
  - Past public research presentations
    - malware research (Winnti/PlugX), anti-forensic analysis, memory forensics

OVERVIEW

- Background
- fn_fuzzy
- Evaluation
- Wrap-up

4

BACKGROUND

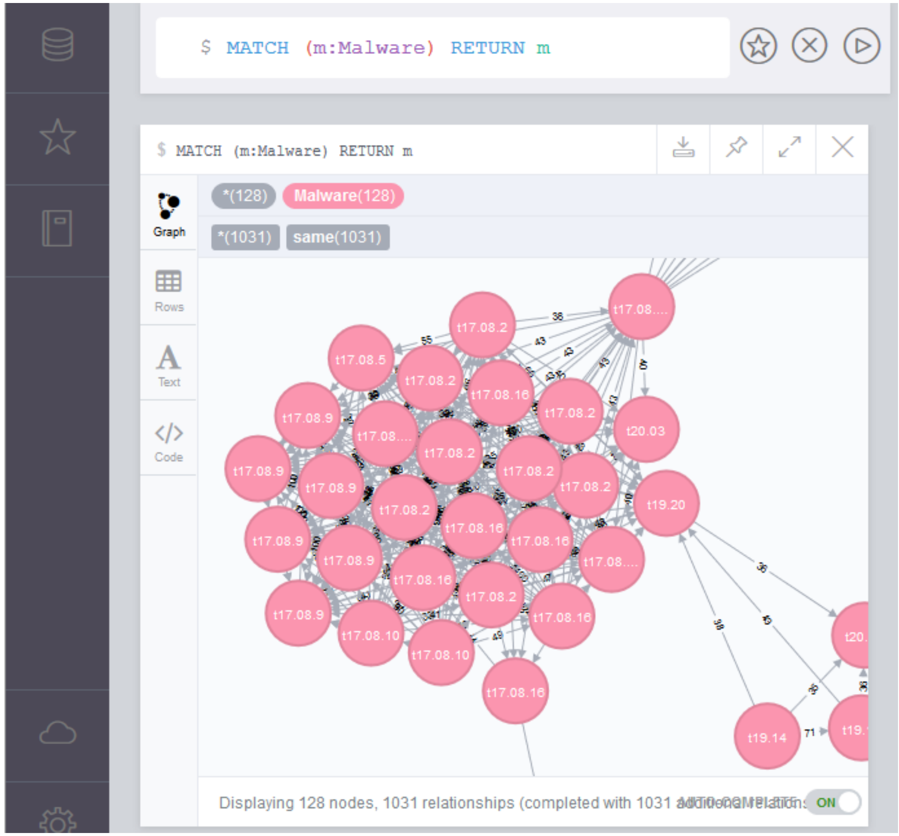
BACKGROUND

- IDA Pro is the de facto disassembler for malware reverse engineers
  - save findings into the database files (IDBs)
  - import them when analyzing new malware variants
- Which is the most similar & analyzed IDB to be imported?
  - A lot of IDBs
  - Some of them were analyzed a few years ago ☹

RELATED BINARY DIFFING TOOLS

- Impfuzzy-based binary diffing for PE-formatted executables
  - impfuzzy for Neo4j
- Function-level binary diffing with IDA
  - one on one comparison
    - BinDiff
    - Diaphora
    - BinGrep
  - one to many comparison
    - BinDiff automation tool
    - Kam1n0

IMPFUZZY FOR NEO4J



```
$ MATCH (m:Malware) RETURN m
```

```
$ MATCH (m:Malware) RETURN m
```

Displaying 128 nodes, 1031 relationships (completed with 1031 additional relationships)

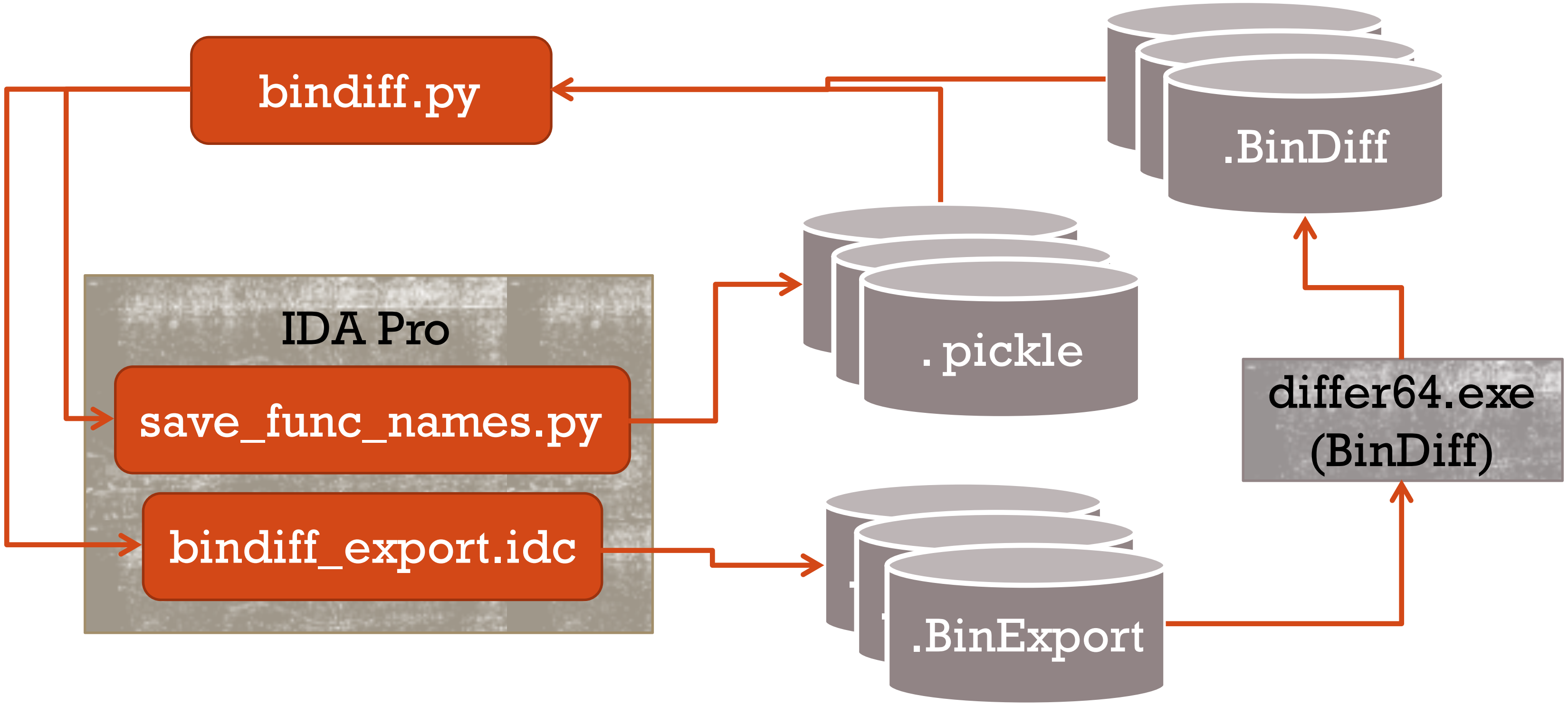
- Published by JPCERT [1]
- impfuzzy
  - ssdeep value of API function names in PE import section
- Neo4j visualizes malware clustering based on impfuzzy values quickly
- Not available for
  - Mac/Linux malware
  - malware resolving API function addresses dynamically
- Not sure which sample is most-analyzed

FUNCTION-LEVEL BINARY DIFFING: ONE-ON-ONE SAMPLE COMPARISON

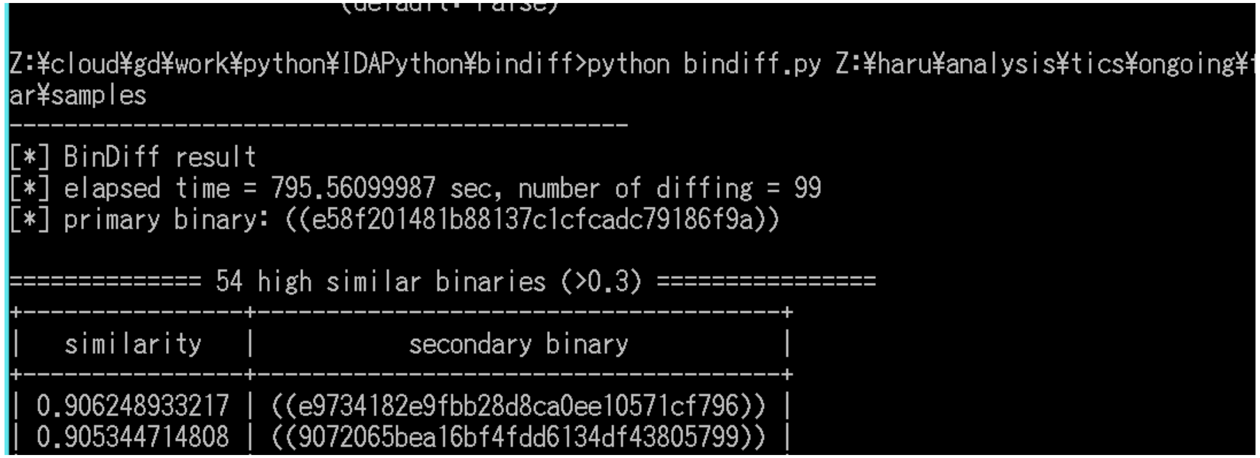
- BinDiff [2]
  - widely-used IDA Pro plugin
- Diaphora [3]
  - IDAPython script supporting psuedo-code diffing
  - the development is very active
- BinGrep [4]
  - IDAPython script providing multiple candidates for each function
- All tools compare binaries one-on-one

FUNCTION-LEVEL BINARY DIFFING: ONE-TO-MANY SAMPLE COMPARISON (BINDIFF AUTOMATION TOOL)

- My wrapper script for BinDiff 4.2



FUNCTION-LEVEL BINARY DIFFING: ONE-TO-MANY SAMPLE COMPARISON (BINDIFF AUTOMATION TOOL, CONT.)

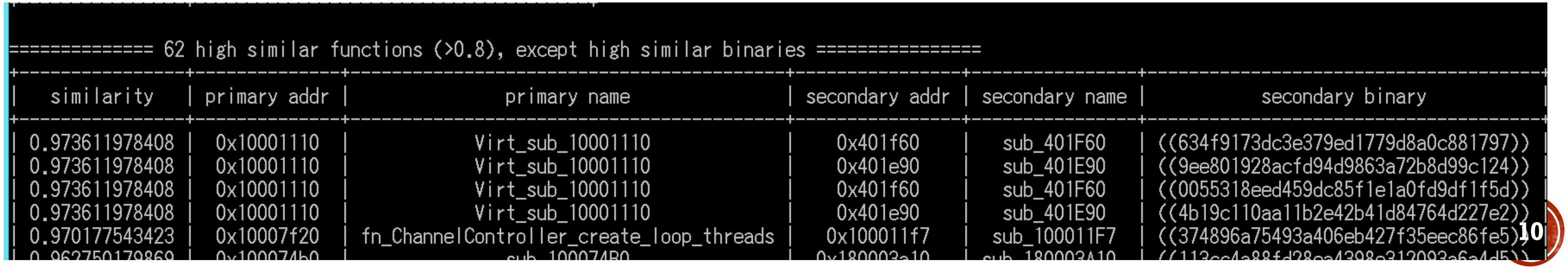


```
                         (default: False)

Z:¥cloud¥gd¥work¥python¥IDAPython¥bindiff>python bindiff.py Z:¥haru¥analysis¥tics¥ongoing¥t
ar¥samples
---------------------------------------------
[*] BinDiff result
[*] elapsed time = 795.56099987 sec, number of diffing = 99
[*] primary binary: ((e58f201481b88137c1cfcadc79186f9a))

============= 54 high similar binaries (>0.3) ================
+----------------+--------------------------------------+
|   similarity   |            secondary binary          |
+----------------+--------------------------------------+
| 0.906248933217 | ((e9734182e9fbb28d8ca0ee10571cf796)) |
| 0.905344714808 | ((9072065bea16bf4fdd6134df43805799)) |
```

...



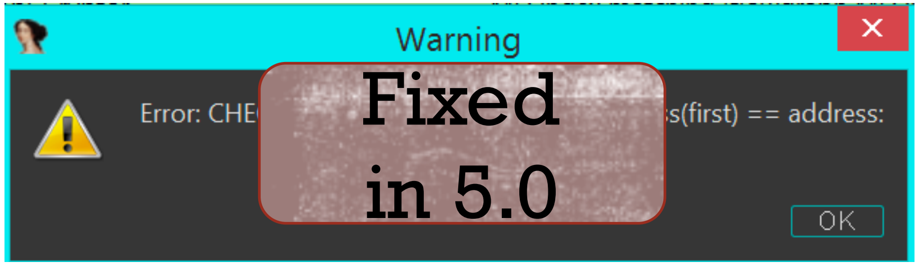
```
+----------------+--------------------------------------+

============= 62 high similar functions (>0.8), except high similar binaries ================
+----------------+--------------+--------------------------------------+----------------+----------------+--------------------------------------+
|   similarity   | primary addr |             primary name             | secondary addr | secondary name |           secondary binary           |
+----------------+--------------+--------------------------------------+----------------+----------------+--------------------------------------+
| 0.973611978408 |  0x10001110  |           Virt_sub_10001110          |    0x401f60    |   sub_401F60   | ((634f9173dc3e379ed1779d8a0c881797)) |
| 0.973611978408 |  0x10001110  |           Virt_sub_10001110          |    0x401e90    |   sub_401E90   | ((9ee801928acfd94d9863a72b8d99c124)) |
| 0.973611978408 |  0x10001110  |           Virt_sub_10001110          |    0x401f60    |   sub_401F60   | ((0055318eed459dc85f1e1a0fd9df1f5d)) |
| 0.973611978408 |  0x10001110  |           Virt_sub_10001110          |    0x401e90    |   sub_401E90   | ((4b19c110aa11b2e42b41d84764d227e2)) |
| 0.970177543423 |  0x10007f20  | fn_ChannelController_create_loop_threads |  0x100011f7  |  sub_100011F7  | ((374896a75493a406eb427f35eec86fe5)) |
| 0.962750179869 |  0x100074b0  |             sub_100074B0             |   0x180003a10  |  sub_180003A10 | ((113cc4a88fd28ca4398a312093a6a4d5)) |
```

- 99 samples comparison on my analysis VM
  - 795 secs
  - 300 secs if .BinExport ready

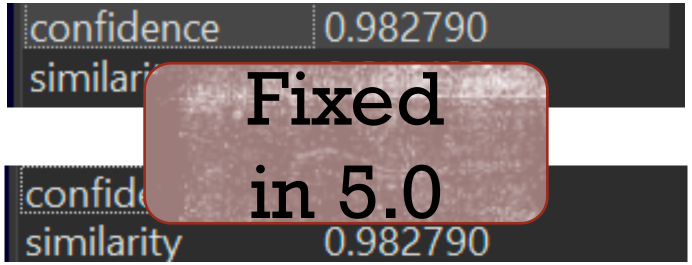
FUNCTION-LEVEL BINARY DIFFING: ONE-TO-MANY SAMPLE COMPARISON (BINDIFF AUTOMATION TOOL, CONT.)

- The wrapper is not scalable for hundreds or thousands samples
- BinDiff is closed-source software
  - multiple functions importing error (4.3)
  - confidence/similarity swapped after saving&loading .BinDiff (4.3 or before)
  - saved .BinDiff file loading error (5.0) <- NEW!



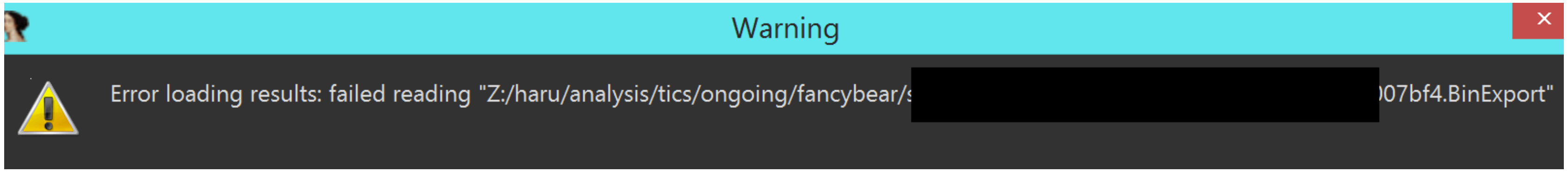
Error: CHE s(first) == address:

Fixed in 5.0


confidence	0.982790
similari	

confide	
similarity	0.982790

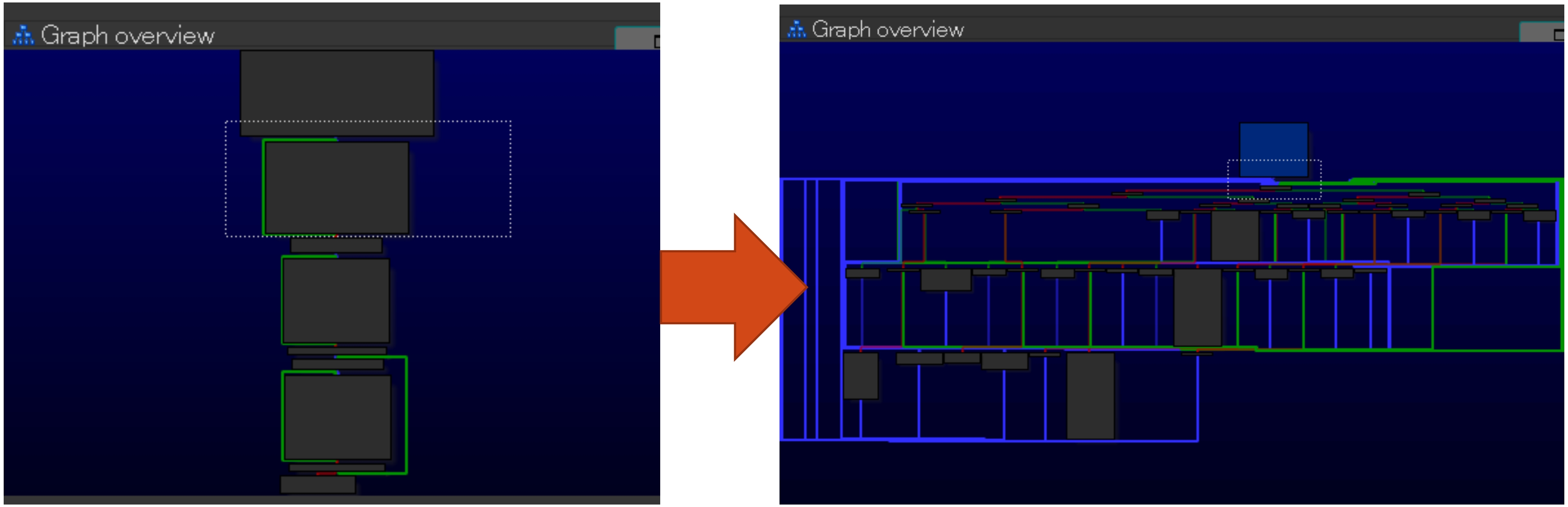
Fixed in 5.0



Error loading results: failed reading "Z:/haru/analysis/tics/ongoing/fancybear/s[redacted]07bf4.BinExport"

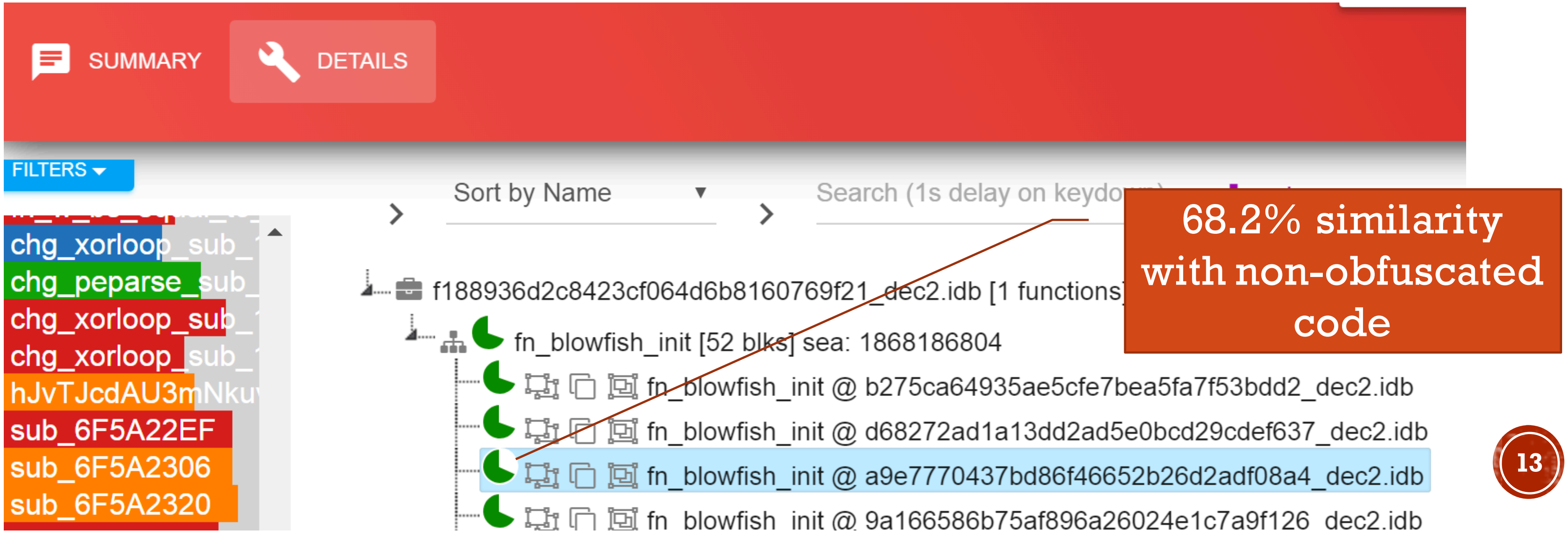
FUNCTION-LEVEL BINARY DIFFING: ONE-TO-MANY SAMPLE COMPARISON (KAM1NO) [5]

- Scalable assembly management and analysis platform with IDAPython plugin
  - Asm2Vec analysis engine has high accuracy (>0.8) for all options applied in O-LLVM
- I tested APT10 malware obfuscated by an unknown obfuscating compiler [13]



FUNCTION-LEVEL BINARY DIFFING: ONE-TO-MANY SAMPLE COMPARISON (KAM1N0, CONT.)

- Kam1n0 could detect original functions of the highly-obfuscated one!
- But 20 samples comparison takes over 1 hour
  - Kam1n0 requires high-spec machines



```
f188936d2c8423cf064d6b8160769f21_dec2.idb [1 functions]
  fn_blowfish_init [52 blks] sea: 1868186804
    fn_blowfish_init @ b275ca64935ae5cfe7bea5fa7f53bdd2_dec2.idb
    fn_blowfish_init @ d68272ad1a13dd2ad5e0bcd29cdef637_dec2.idb
    fn_blowfish_init @ a9e7770437bd86f46652b26d2adf08a4_dec2.idb
    fn_blowfish_init @ 9a166586b75af896a26024e1c7a9f126_dec2.idb
```

68.2% similarity with non-obfuscated code

MOTIVATION

- Function-level binary diffing to identify the most similar & analyzed IDB from large ones then import the findings
  - get the comparison result quickly
    - e.g., less than 1 minute for hundreds or thousands comparison
  - not require high-spec machines
    - simpler tool to work on the analysis VM of the laptop

15

FN_FUZZY

BASIC CONCEPT

- fn_fuzzy calculates two kinds of fuzzy hashes for each function
  - ssdeep [6] hash value of code bytes
  - Machoc [7] hash value of call flow graph
- All hashes are saved into one database file then used for comparison
  - On IDA, we can import function names and prototypes from multiple IDBs at one time
    - Structure type information will be imported automatically as needed

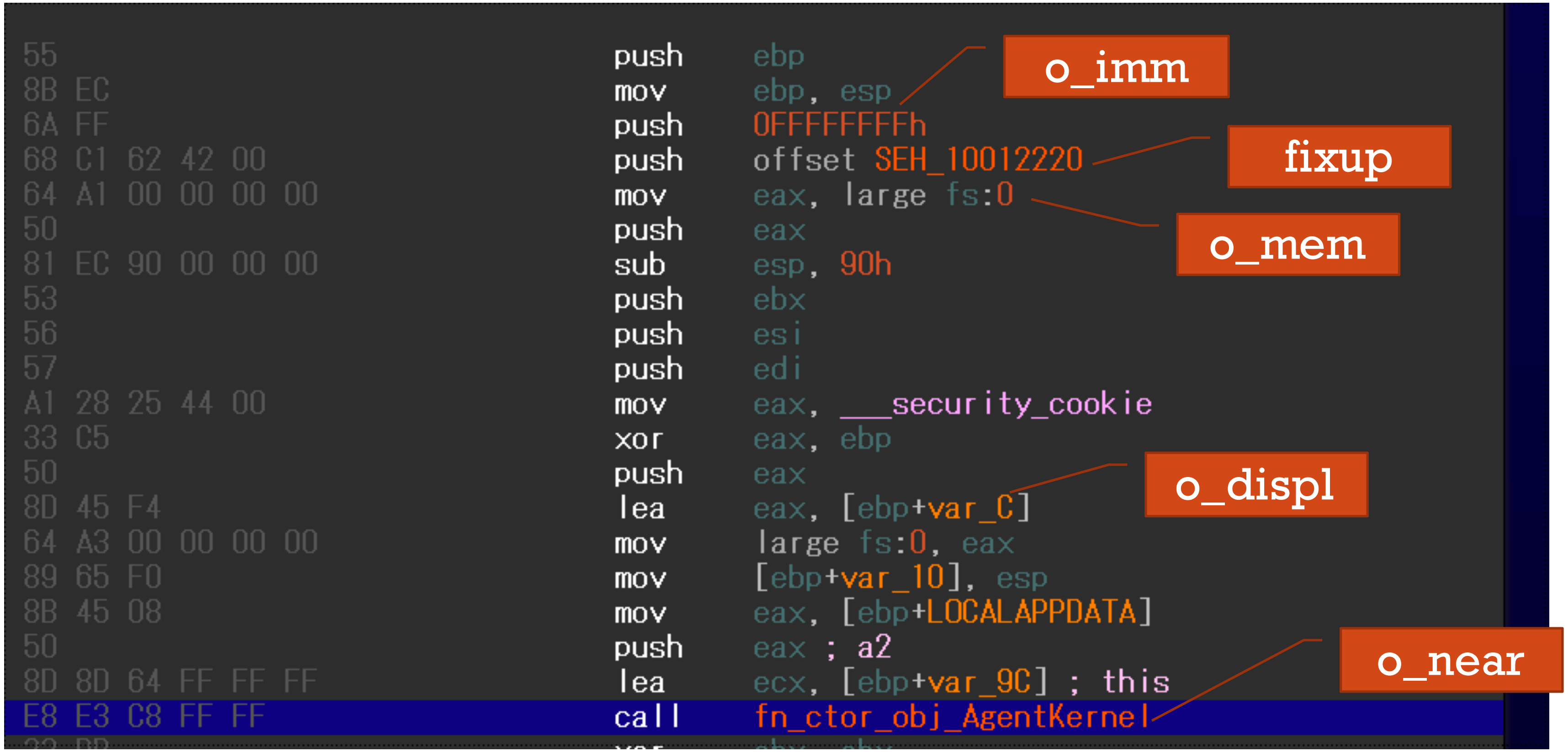
SSDEEP HASH VALUE OF CODE BYTES: WHY SSDEEP?

- de facto standard
  - originally from spam email detection algorithm, but not limited to text data
- speed
  - twice as fast as TLSH [8]
- other fuzzy hashes require minimum size
  - e.g., 512 bytes in sdhash [9]
  - ssdeep doesn’t define the minimum size

SSDEEP HASH VALUE OF CODE BYTES: GENERIC CODE BYTES EXTRACTION

- I've used the modified version of yara_fn.py [10] to define a yara rule based on generic code bytes of a function
  - calculate fixup (relocation) size correctly
  - exclude not only fixup bytes but also following operand type values
    - o_mem, o_imm, o_displ, o_near, o_far
- I reuse it for ssdeep hash calculation

SSDEEP HASH VALUE OF CODE BYTES: GENERIC CODE BYTES EXTRACTION (CONT.)

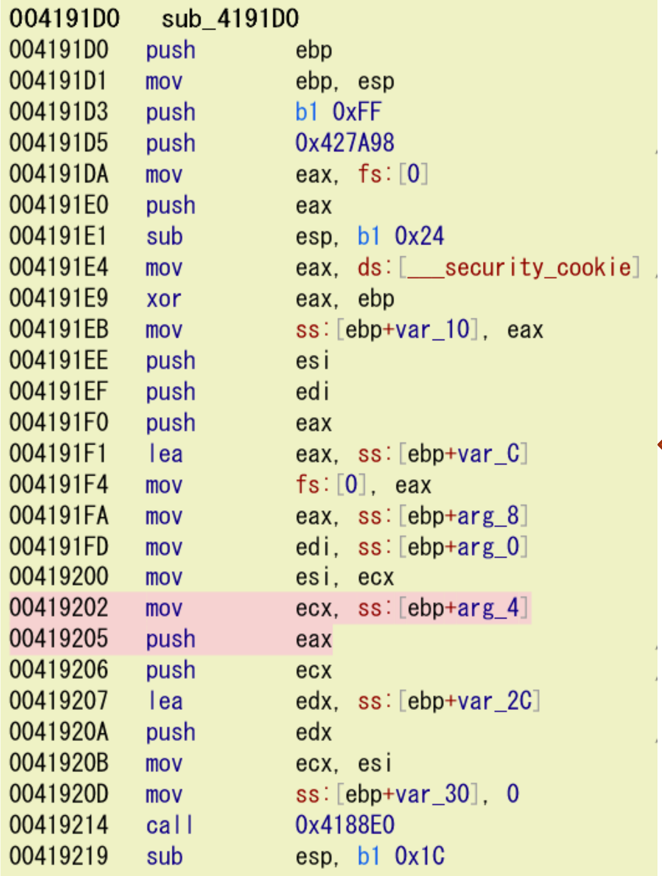


```
55                    push    ebp
8B EC                 mov     ebp, esp
6A FF                 push    0FFFFFFFFh
68 C1 62 42 00        push    offset SEH_10012220
64 A1 00 00 00 00     mov     eax, large fs:0
50                    push    eax
81 EC 90 00 00 00     sub     esp, 90h
53                    push    ebx
56                    push    esi
57                    push    edi
A1 28 25 44 00        mov     eax, ___security_cookie
33 C5                 xor     eax, ebp
50                    push    eax
8D 45 F4              lea     eax, [ebp+var_C]
64 A3 00 00 00 00     mov     large fs:0, eax
89 65 F0              mov     [ebp+var_10], esp
8B 45 08              mov     eax, [ebp+LOCALAPPDATA]
50                    push    eax ; a2
8D 8D 64 FF FF FF     lea     ecx, [ebp+var_9C] ; this
E8 E3 C8 FF FF        call    fn_ctor_obj_AgentKernel
33 DB                 xor     ebx, ebx
```

{ 55 8B EC 6A ?? 68 ?? ?? ?? ?? 64 A1 ?? ?? ?? ?? 50 81 EC ?? ?? ?? ?? 53 56 57 A1 ?? ?? ?? ?? 33 C5 50 8D 45 ?? 64 A3 ?? ?? ?? ?? 89 65 ?? 8B 45 ?? 50 8D 8D ?? ?? ?? ?? E8 }

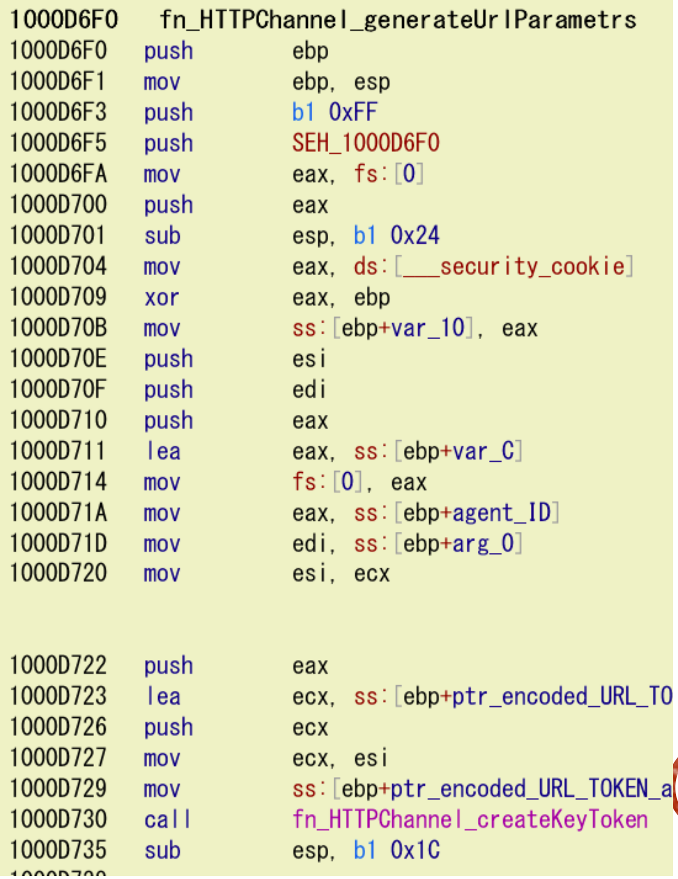
MACHOC HASH VALUE OF CALL FLOW GRAPH: PURPOSE

- The ssdeep score for small data sometimes drops sharply
- fn_fuzzy calculates Machoc hash values of call flow graphs to correct abnormal ssdeep score



```
004191D0    sub_4191D0
004191D0    push    ebp
004191D1    mov     ebp, esp
004191D3    push    b1 0xFF
004191D5    push    0x427A98
004191DA    mov     eax, fs:[0]
004191E0    push    eax
004191E1    sub     esp, b1 0x24
004191E4    mov     eax, ds:[___security_cookie]
004191E9    xor     eax, ebp
004191EB    mov     ss:[ebp+var_10], eax
004191EE    push    esi
004191EF    push    edi
004191F0    push    eax
004191F1    lea     eax, ss:[ebp+var_C]
004191F4    mov     fs:[0], eax
004191FA    mov     eax, ss:[ebp+arg_8]
004191FD    mov     edi, ss:[ebp+arg_0]
00419200    mov     esi, ecx
00419202    mov     ecx, ss:[ebp+arg_4]
00419205    push    eax
00419206    push    ecx
00419207    lea     edx, ss:[ebp+var_2C]
0041920A    push    edx
0041920B    mov     ecx, esi
0041920D    mov     ss:[ebp+var_30], 0
00419214    call    0x4188E0
00419219    sub     esp, b1 0x1C
```

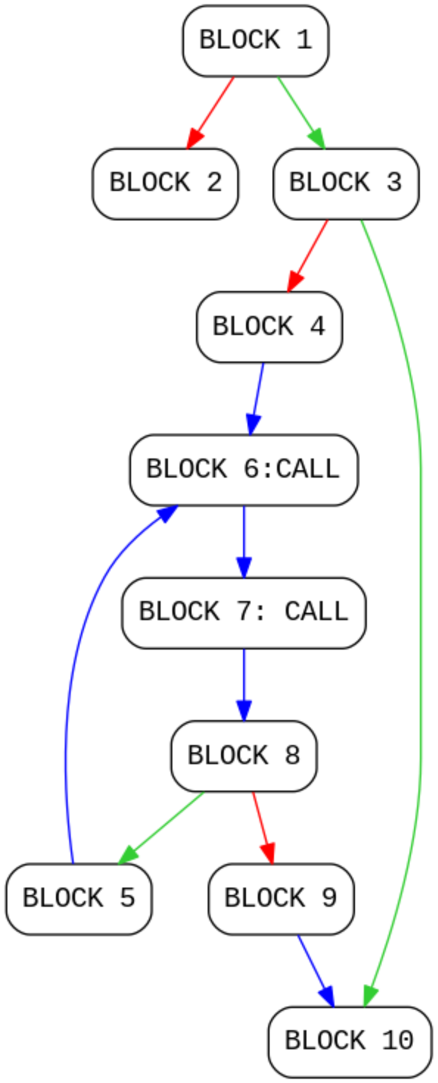
ssdeep score: 33



```
1000D6F0    fn_HTTPChannel_generateUrlParametrs
1000D6F0    push    ebp
1000D6F1    mov     ebp, esp
1000D6F3    push    b1 0xFF
1000D6F5    push    SEH_1000D6F0
1000D6FA    mov     eax, fs:[0]
1000D700    push    eax
1000D701    sub     esp, b1 0x24
1000D704    mov     eax, ds:[___security_cookie]
1000D709    xor     eax, ebp
1000D70B    mov     ss:[ebp+var_10], eax
1000D70E    push    esi
1000D70F    push    edi
1000D710    push    eax
1000D711    lea     eax, ss:[ebp+var_C]
1000D714    mov     fs:[0], eax
1000D71A    mov     eax, ss:[ebp+agent_ID]
1000D71D    mov     edi, ss:[ebp+arg_0]
1000D720    mov     esi, ecx


1000D722    push    eax
1000D723    lea     ecx, ss:[ebp+ptr_encoded_URL_TO
1000D726    push    ecx
1000D727    mov     ecx, esi
1000D729    mov     ss:[ebp+ptr_encoded_URL_TOKEN_a
1000D730    call    fn_HTTPChannel_createKeyToken
1000D735    sub     esp, b1 0x1C
```

MACHOC HASH VALUE OF CALL FLOW GRAPH: WHAT'S MACHOC HASH?



1:2,3;
2:;
3:4,10;
4:6;
5:6;
6:c,7;
7:c,8;
8:5,9;
9:10;
10:;

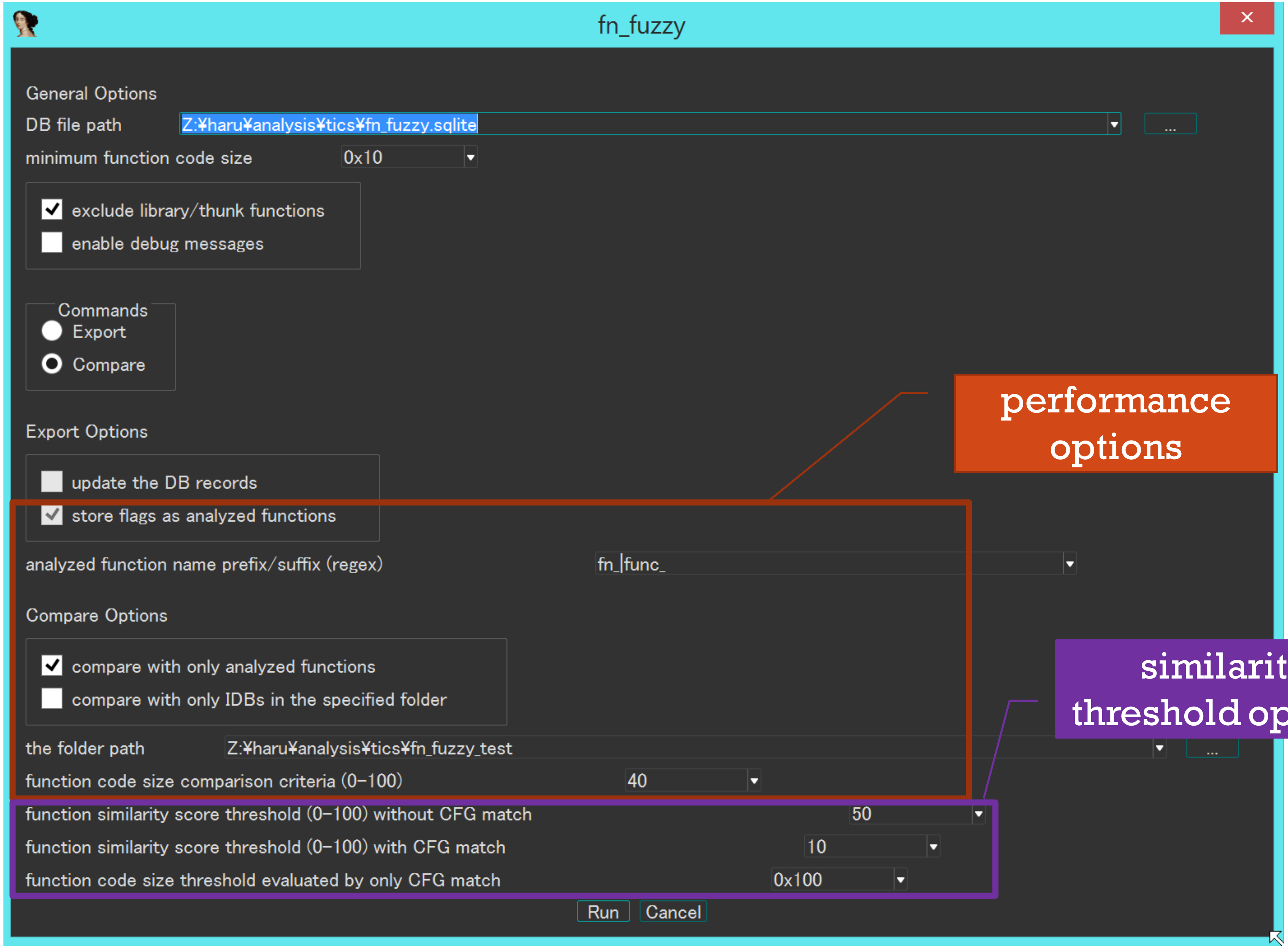
0x1014997f

- Simple fuzzy hash mechanism based on the Call Flow Graph (CFG) of a function
- Each basic block is numbered and translated to a string
  - NUMBER:[c,][DST, ...];
- The concatenated string is hashed to produce a 32 bits output
  - fn_fuzzy uses Murmurhash3 [11]

IMPLEMENTATION

- IDAPython and the wrapper scripts
  - fn_fuzzy.py
    - IDAPython script to export/compare hashes of one binary on IDA
  - cli_export.py
    - python wrapper script to export hashes of multiple binaries
- Required python packages: mmh3, python-idb [12]
- Supported IDB version
  - generated by IDA 6.9 or later due to SHA256 API usage
    - ida_netnode.cvar.root_node.supstr(ida_nalt.RIDX_IDA_VERSION)

DEMO: EXECUTION OPTIONS DIALOG



General Options

DB file path Z:¥haru¥analysis¥tics¥fn_fuzzy.sqlite

minimum function code size 0x10

☑ exclude library/thunk functions

□ enable debug messages

Commands

○ Export

● Compare

Export Options

□ update the DB records

☑ store flags as analyzed functions

analyzed function name prefix/suffix (regex) fn_|func_

Compare Options

☑ compare with only analyzed functions

□ compare with only IDBs in the specified folder

the folder path Z:¥haru¥analysis¥tics¥fn_fuzzy_test

function code size comparison criteria (0-100) 40

function similarity score threshold (0-100) without CFG match 50

function similarity score threshold (0-100) with CFG match 10

function code size threshold evaluated by only CFG match 0x100

performance options

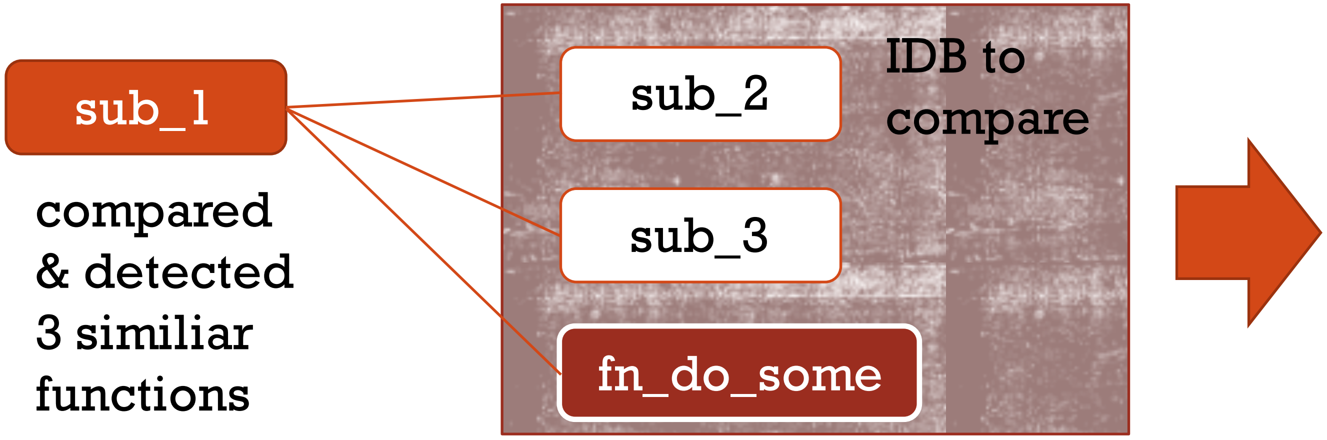
similarity threshold options

FN_FUZZY.PY: PERFORMANCE OPTIONS

- ssdeep hash comparison computation
  - We compare y hashes against the database containing x hashes = O(xy) :(
  - e.g., x = 317,576 hashes from 733 samples
- Performance options
  - *compare with only analyzed functions*
    - Analyzed flag info is added based on the renamed function name prefix/suffix in export command
  - *compare with only IDBs in the specified folder*
    - Specify the folder path
  - *function code size comparison criteria (0-100)*
    - Each hash comparison only targets hashes with similar size (40 = comparison with 60%-140% size hashes)

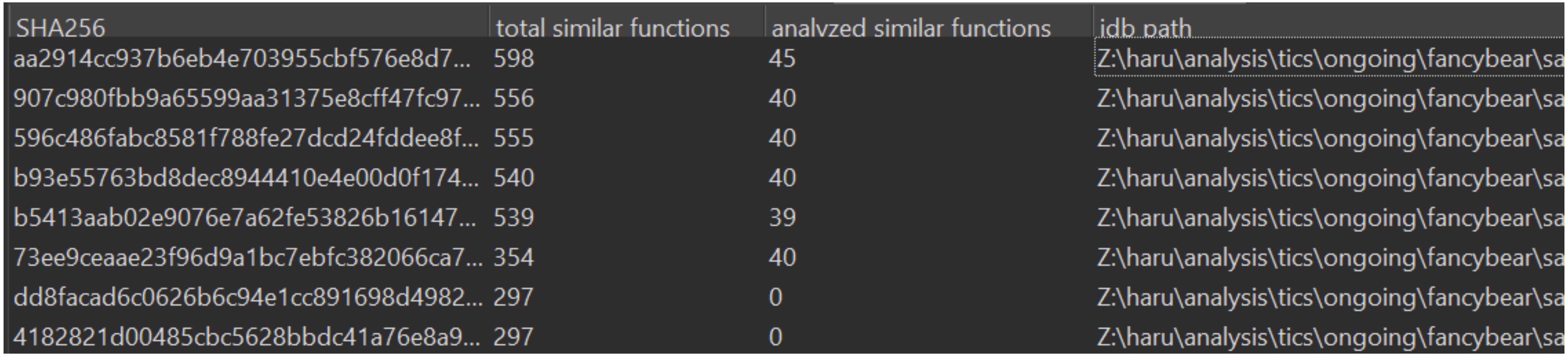
DEMO: SUMMARY TAB

- fn_fuzzy counts multiple similar functions per each function comparison



compared & detected 3 similiar functions

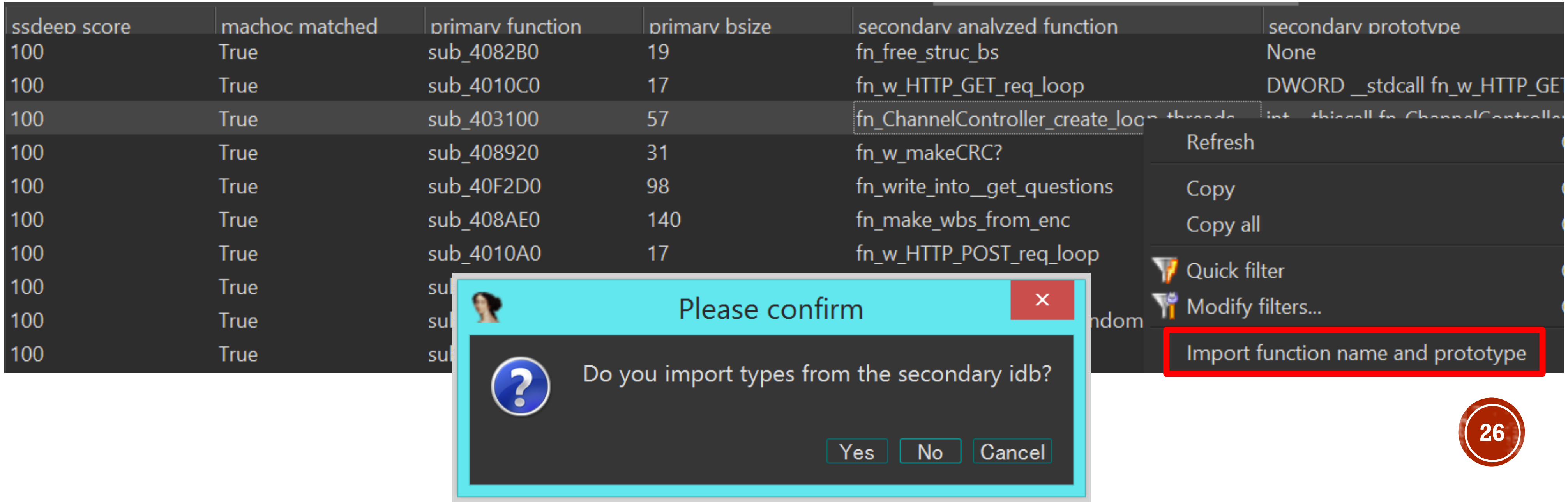
sub_1 comparison:
total += 3
analyzed += 1



SHA256	total similar functions	analyzed similar functions	idb path
aa2914cc937b6eb4e703955cbf576e8d7...	598	45	Z:\haru\analysis\tics\ongoing\fancybear\sa
907c980fbb9a65599aa31375e8cff47fc97...	556	40	Z:\haru\analysis\tics\ongoing\fancybear\sa
596c486fabc8581f788fe27dcd24fddee8f...	555	40	Z:\haru\analysis\tics\ongoing\fancybear\sa
b93e55763bd8dec8944410e4e00d0f174...	540	40	Z:\haru\analysis\tics\ongoing\fancybear\sa
b5413aab02e9076e7a62fe53826b16147...	539	39	Z:\haru\analysis\tics\ongoing\fancybear\sa
73ee9ceaae23f96d9a1bc7ebfc382066ca7...	354	40	Z:\haru\analysis\tics\ongoing\fancybear\sa
dd8facad6c0626b6c94e1cc891698d4982...	297	0	Z:\haru\analysis\tics\ongoing\fancybear\sa
4182821d00485cbc5628bbdc41a76e8a9...	297	0	Z:\haru\analysis\tics\ongoing\fancybear\sa

DEMO: SIMILARITIES WITH [SHA256] TAB

- fn_fuzzy displays primary and secondary functions one on one
  - analyzed & the highest score function selected
- Right-click->"Import function name and prototype"
  - If the structure type is not found, we can import the type info



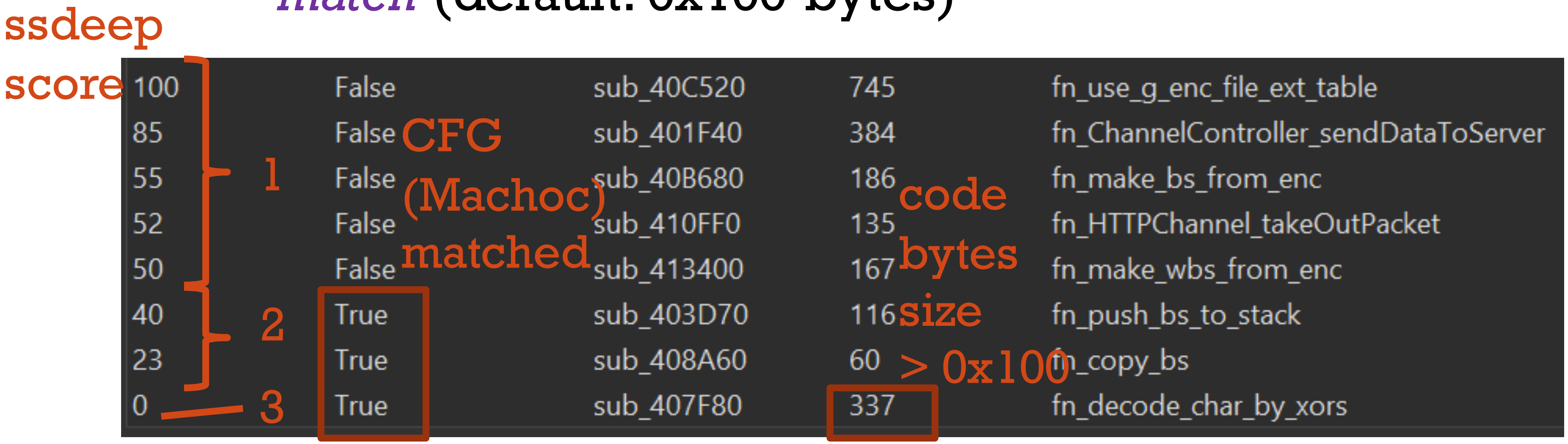
ssdeep score	machoc matched	primary function	primary bsize	secondary analyzed function	secondary prototype
100	True	sub_4082B0	19	fn_free_struc_bs	None
100	True	sub_4010C0	17	fn_w_HTTP_GET_req_loop	DWORD __stdcall fn_w_HTTP_GE
100	True	sub_403100	57	fn_ChannelController_create_loop_threads	int __thiscall fn_ChannelControlle
100	True	sub_408920	31	fn_w_makeCRC?	
100	True	sub_40F2D0	98	fn_write_into__get_questions	
100	True	sub_408AE0	140	fn_make_wbs_from_enc	
100	True	sub_4010A0	17	fn_w_HTTP_POST_req_loop	
100	True	su			
100	True	su		ndom	
100	True	su			

Please confirm

Do you import types from the secondary idb?

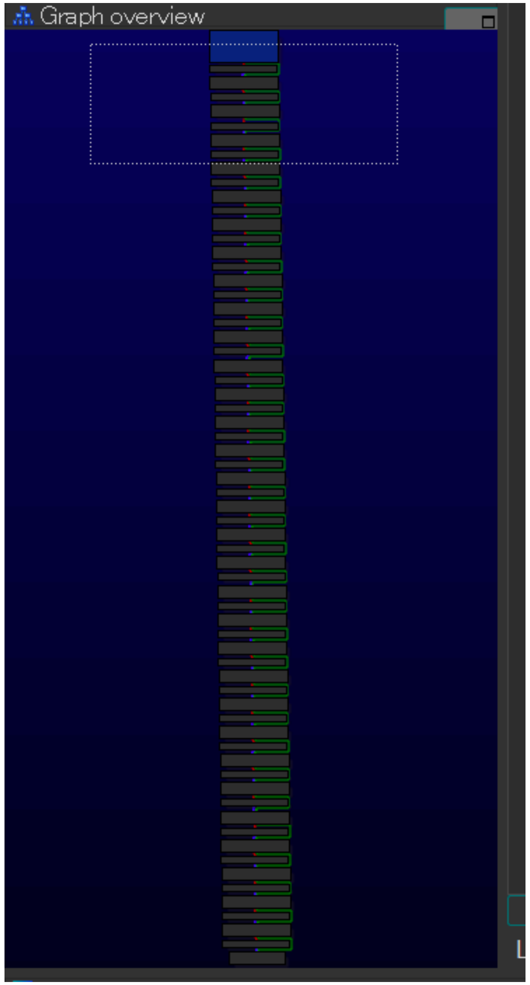
FN_FUZZY.PY: SIMILARITY THRESHOLD OPTIONS

- fn_fuzzy detects similar functions matching with one of following conditions
  1. *function similarity score threshold (0-100) without CFG match* (default: 50)
  2. *function similarity score threshold (0-100) with CFG match* (default: 10)
  3. *function code size threshold evaluated by only CFG match* (default: 0x100 bytes)



100	False	sub_40C520	745	fn_use_g_enc_file_ext_table
85	False	sub_401F40	384	fn_ChannelController_sendDataToServer
55	False	sub_40B680	186	fn_make_bs_from_enc
52	False	sub_410FF0	135	fn_HTTPChannel_takeOutPacket
50	False	sub_413400	167	fn_make_wbs_from_enc
40	True	sub_403D70	116	fn_push_bs_to_stack
23	True	sub_408A60	60	fn_copy_bs
0	True	sub_407F80	337	fn_decode_char_by_xors

CONDITION 3: SSDEEP SCORE 0 BUT CFG (MACHOC) MATCHED?



- e.g., Fancy Bear XAgent variant with a polymorphic deobfuscation function
  - the arithmetic logics and immediate values are changed per sample
  - but the CFG is the exactly same
- The condition may also detect similarities between different architecture samples

EVALUATION

PERFORMANCE

- 733 IDBs tested on the same analysis VM
- Export
  - cli_export.py with -ear options
  - about 2 hours
- Compare
  - compare a C++ sample including 900 functions with the DB
    - default options and values
  - about 20-30 secs (analyzed functions only)
  - about 3 minutes (all functions)

ACCURACY1: UPDATED VARIANT

- tested Fancy Bear XAgent samples
  - sample A: AgentKernel module ID 0x3303
  - sample B: AgentKernel module ID 0x4401
- compare sample B IDB with sample A IDB
  - sample A IDB contains 69 analyzed functions
- BinDiff vs. fn_fuzzy
  - manually checked the results
    - BinDiff: similarity > 0.7
    - fn_fuzzy: default similarity threshold options

ACCURACY1: UPDATED VARIANT (CONT.)

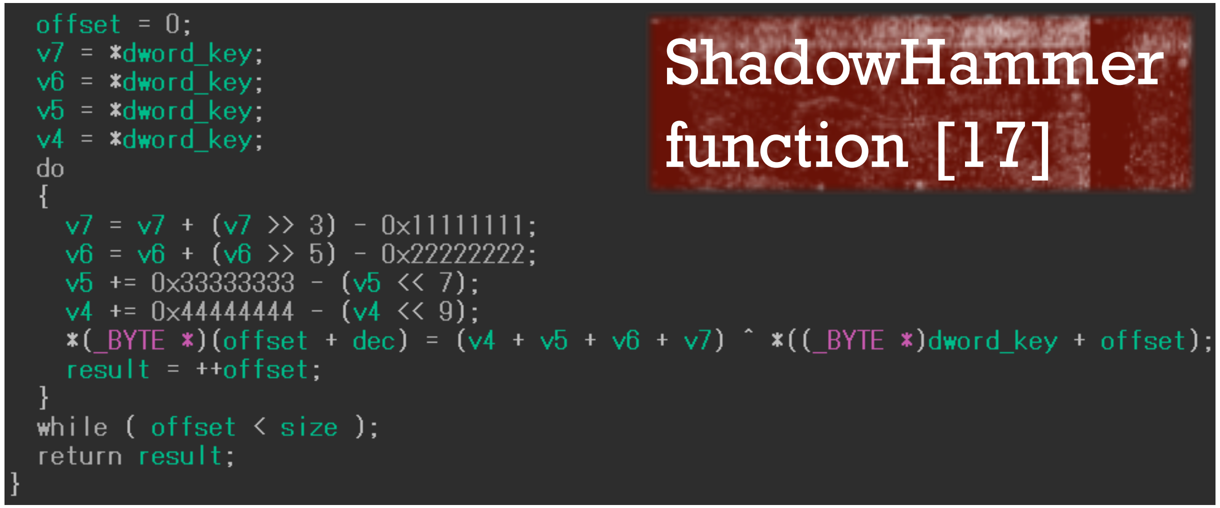
- BinDiff is better than fn_fuzzy
- causes about false negatives
  - BinDiff doesn't accept duplicated matching for secondary functions (4/7)
    - If one match is incorrect, the other will be incorrect too
  - fn_fuzzy
    - exclude small function whose generic code bytes < 0x10 (6/15)
    - can't detect obfuscated functions (2/15)
    - exclude non-library function due to incorrect FLIRT sig (1/15)

item	BinDiff	fn_fuzzy
total detected similar functions	42	35
false positives	1	2
false negatives against functions that the other one could detect	7	15

ACCURACY2: OBFUSCATED VARIANT

- tested APT10 ANEL samples
  - sample A: ANEL 5.2.2 rev2
    - 94 analyzed functions
  - sample B: ANEL 5.4.1
    - heavily-obfuscated with compiler-level obfuscations [13]
- BinDiff detected 3 similar functions
- fn_fuzzy could not find at all
  - 1 function found by changing "function code size comparison criteria" option from 40 to 60
  - Some functions are not obfuscated but CFGs are changed due to more call instructions
    - Machoc hash calculation splits a basic block by them

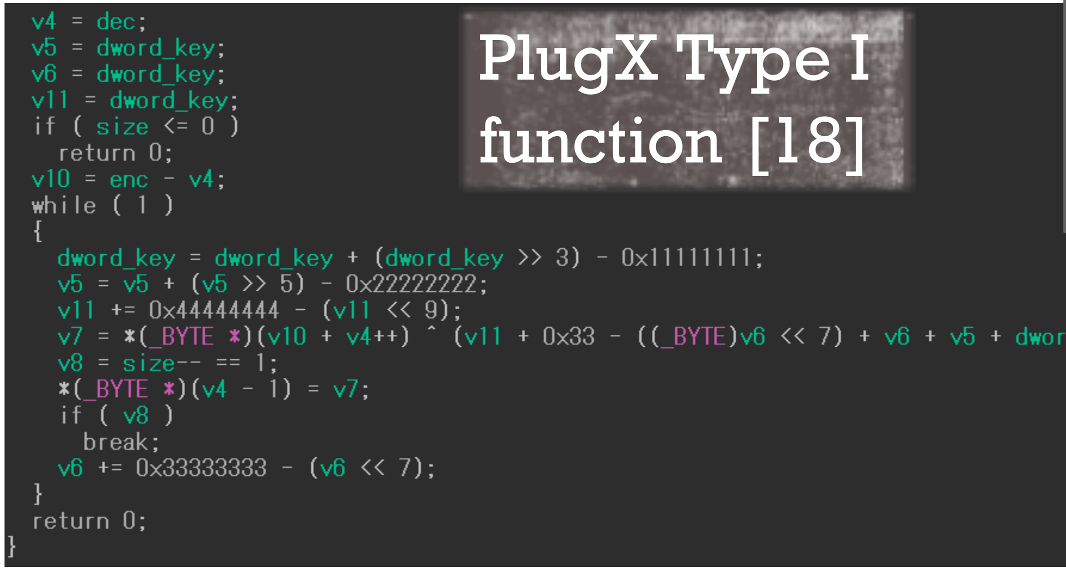
ACCURACY3: UNIQUE DECODING FUNCTION



ShadowHammer function [17]

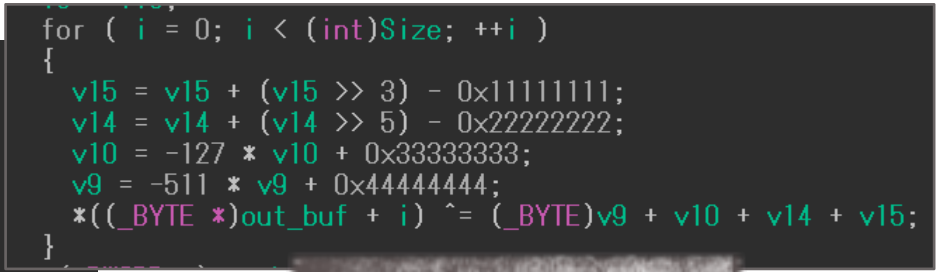
```
  offset = 0;
  v7 = *dword_key;
  v6 = *dword_key;
  v5 = *dword_key;
  v4 = *dword_key;
  do
  {
    v7 = v7 + (v7 >> 3) - 0x11111111;
    v6 = v6 + (v6 >> 5) - 0x22222222;
    v5 += 0x33333333 - (v5 << 7);
    v4 += 0x44444444 - (v4 << 9);
    *(_BYTE *)(offset + dec) = (v4 + v5 + v6 + v7) ^ *((_BYTE *)dword_key + offset);
    result = ++offset;
  }
  while ( offset < size );
  return result;
}
```

- The similar functions from old 2 binaries can be detected?



PlugX Type I function [18]

```
  v4 = dec;
  v5 = dword_key;
  v6 = dword_key;
  v11 = dword_key;
  if ( size <= 0 )
    return 0;
  v10 = enc - v4;
  while ( 1 )
  {
    dword_key = dword_key + (dword_key >> 3) - 0x11111111;
    v5 = v5 + (v5 >> 5) - 0x22222222;
    v11 += 0x44444444 - (v11 << 9);
    v7 = *(_BYTE *)(v10 + v4++) ^ (v11 + 0x33 - ((_BYTE)v6 << 7) + v6 + v5 + dword_key);
    v8 = size-- == 1;
    *(_BYTE *)(v4 - 1) = v7;
    if ( v8 )
      break;
    v6 += 0x33333333 - (v6 << 7);
  }
  return 0;
}
```



```
  for ( i = 0; i < (int)Size; ++i )
  {
    v15 = v15 + (v15 >> 3) - 0x11111111;
    v14 = v14 + (v14 >> 5) - 0x22222222;
    v10 = -127 * v10 + 0x33333333;
    v9 = -511 * v9 + 0x44444444;
    *((_BYTE *)out_buf + i) ^= (_BYTE)v9 + v10 + v14 + v15;
  }
```

Part of Winnti function

ACCURACY3: SIMILAR DECODING FUNCTION (CONT.)

- All couldn’t detect the similarities
  - PlugX Type I function
    - different code bytes and CFG
  - Part of Winnti function
    - just a small part of the function
- A new algorithm may be required…

	fn_fuzzy	BinDiff	Diaphora	Kam1n0
PlugX Type I detected?	No	No	No	No output after 18 hours Binary Composition
Winnti detected?	No	No	No	

WRAP-UP

WRAP-UP

- fn_fuzzy is a fast and light-weight binary diffing tool for large IDBs
    - BinDiff is still better in accuracy but fn_fuzzy provides a high-speed comparison
    - The code is on GitHub [16]
- Future work
    - extract more generic code bytes
        - exclude function prologue/epilogue (e.g., is_prolog_insn)
    - IDA microcode-based fuzzy hashing
        - combine with HexRaysDeob [14][15] for defeating compiler-level obfuscations

REFERENCES

- [1] https://blogs.jpcert.or.jp/en/2017/03/malware-clustering-using-impfuzzy-and-network-analysis---impfuzzy-for-neo4j-.html
- [2] https://www.zynamics.com/bindiff.html
- [3] https://github.com/joxeankoret/diaphora
- [4] https://github.com/hada2/bingrep
- [5] https://github.com/McGill-DMaS/Kam1n0-Community
- [6] https://ssdeep-project.github.io/ssdeep/index.html
- [7] https://github.com/ANSSI-FR/polichombr/blob/dev/docs/MACHOC_HASH.md
- [8] https://github.com/trendmicro/tlsh
- [9] http://roussev.net/sdhash/sdhash.html

REFERENCES (CONT.)

- [10] https://www.carbonblack.com/2019/04/05/cb-threat-intelligence-notification-hunting-apt28-downloaders/
- [11] https://pypi.org/project/mmh3/
- [12] https://github.com/williballenthin/python-idb
- [13] https://www.carbonblack.com/2019/02/25/defeating-compiler-level-obfuscations-used-in-apt10-malware/
- [14] https://github.com/RolfRolles/HexRaysDeob
- [15] https://github.com/carbonblack/HexRaysDeob
- [16] https://github.com/TakahiroHaruyama/ida_haru/tree/master/fn_fuzzy
- [17] https://securelist.com/operation-shadowhammer/89992/
- [18] https://www.blackhat.com/docs/asia-14/materials/Haruyama/Asia-14-Haruyama-I-Know-You-Want-Me-Unplugging-PlugX.pdf