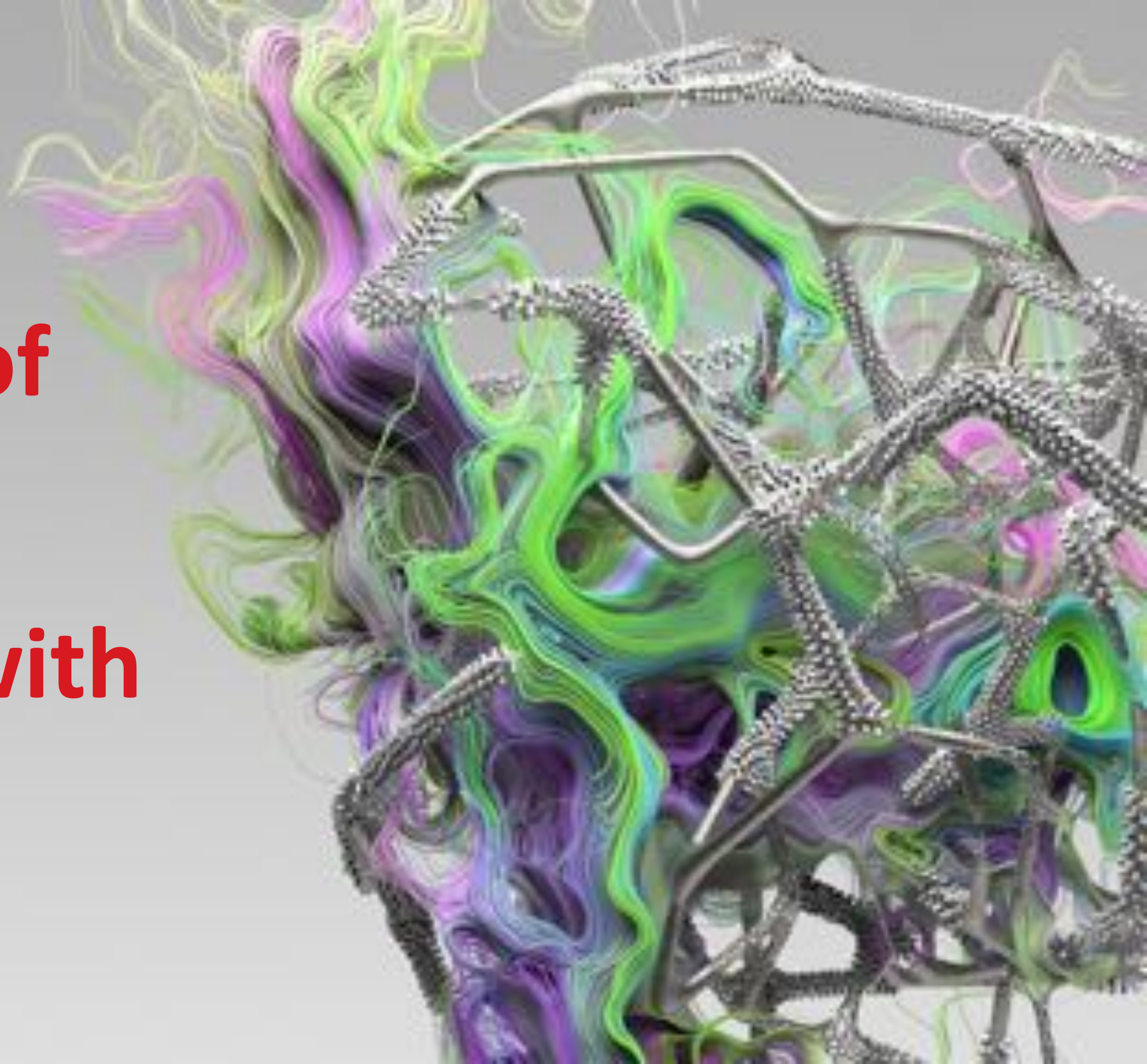
What Species of this Fish is? Malware Classification with Graph Hash



Chia-Ching Fang
Shih-Hao Weng

About Us

- Chia-Ching Fang
  - Over a decade of experience in malware analysis, malicious document analysis, and vulnerability assessment
  - Focus on targeted attacks and threat intelligence now
- Shih-Hao Weng
  - Focus on targeted attack investigation, incident response, and threat solution research for more than 15 years

Agenda

- Motivation
- Related Toolsets / Works
- Methodology
- Demo
- Evaluation
- Limitation
- Conclusion

Motivation

- Malware classification
- Share cyber security intelligence
  – Share IoC with some information that better than file checksum, such as MD5, SHA family

Related Toolsets / Works

Taxonomy	Toolsets / Works
Cryptographic Hash	MD5, SHA Family
Fuzzy Hash	tlsh, ssdeep
Feature-based	imphash
Graph-based	BinDiff
Hybrid	impfuzzy (Feature-based + Fuzzy Hash)

Cryptographic Hash

- Not for classification
- Message digest
- Ex. MD5, SHA256

Fuzzy Hash

- CTPH, Context Triggered Piecewise Hashing
- Match inputs that have homologies
- For digital forensics in the beginning
- Ex. tlsh, ssdeep

imphash

- imphash = f_{MD5} (IAT of Executable)
  - IAT, Import Address Table
  - Executable file feature => Partial content of executable
  - Powered by Madiant

impfuzzy

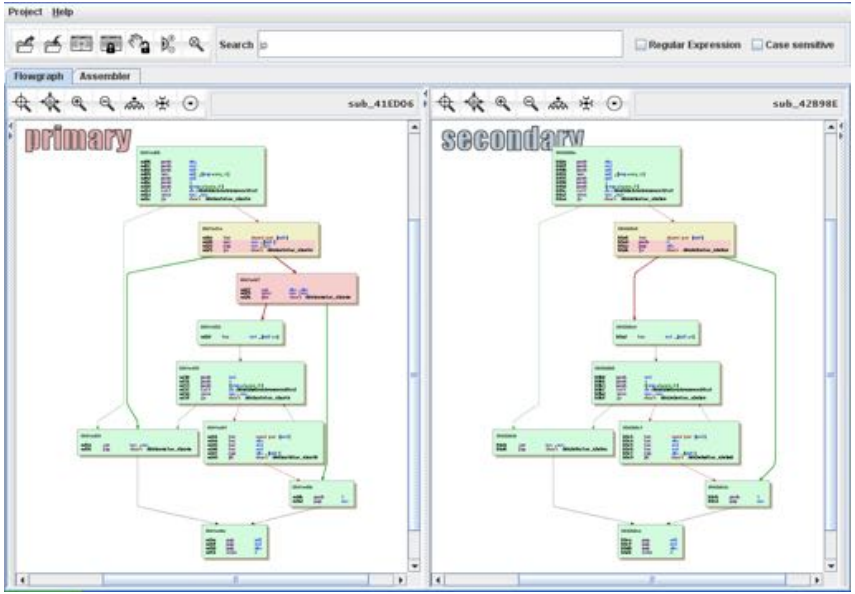
- impfuzzy = f_{ssdeep} (IAT of Executable)
  – Hybrid – Feature-based + Fuzzy Hash
  – Powered by Shusei Tomonaga, JP/CERTCC

Graph-based Similarity Analysis

- From graph point of view
- Call graph of executable

Bindiff

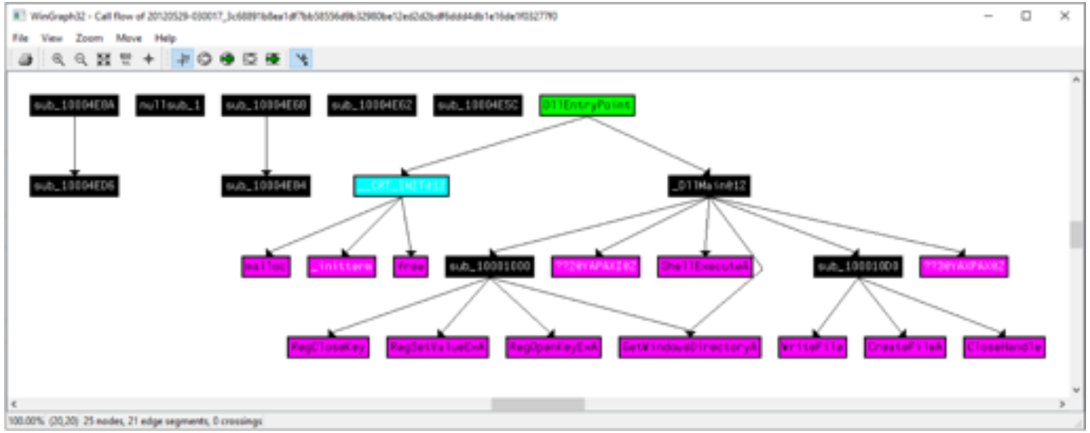
- Very detail information about what similarity in which parts of two executable files
- Vulnerability Analysis / Patch Analysis / Exploit Development

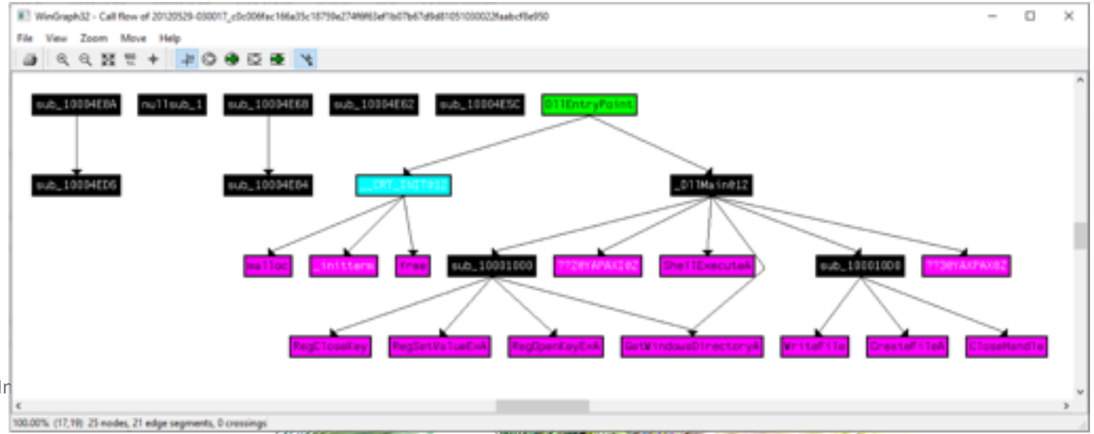


When Using BinDiff …

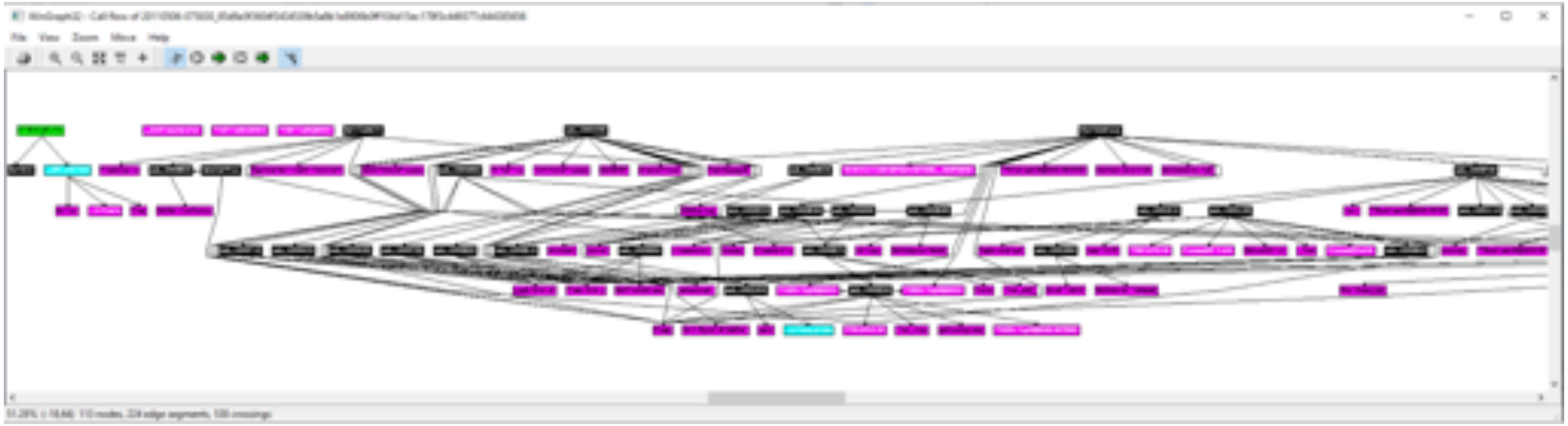
- Only process two files at the same time
- Performance
  - That’s because it does not only do graph comparison, but also disassembly comparison.
- How to scale it?

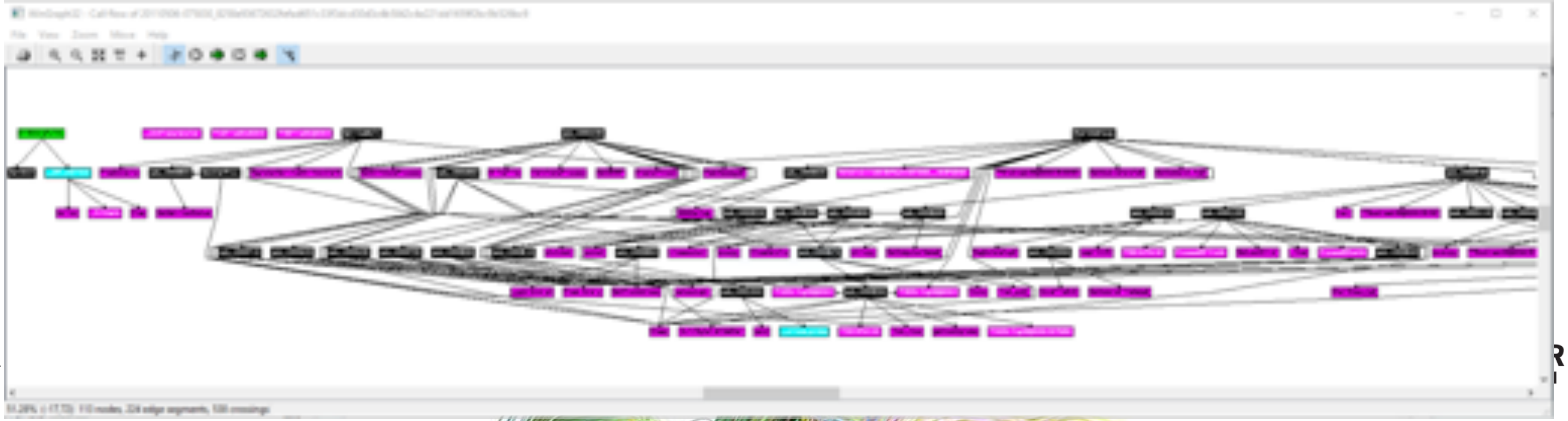
Comparing Call Graphs Task 1



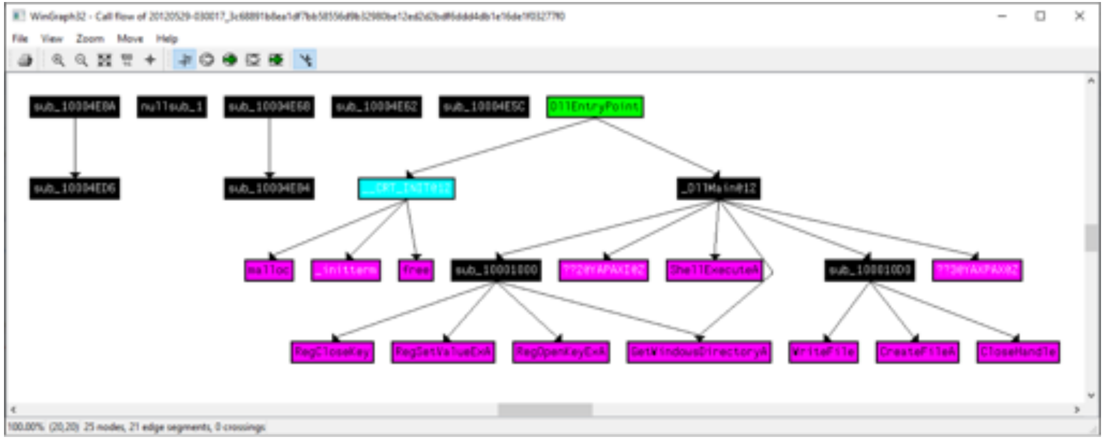


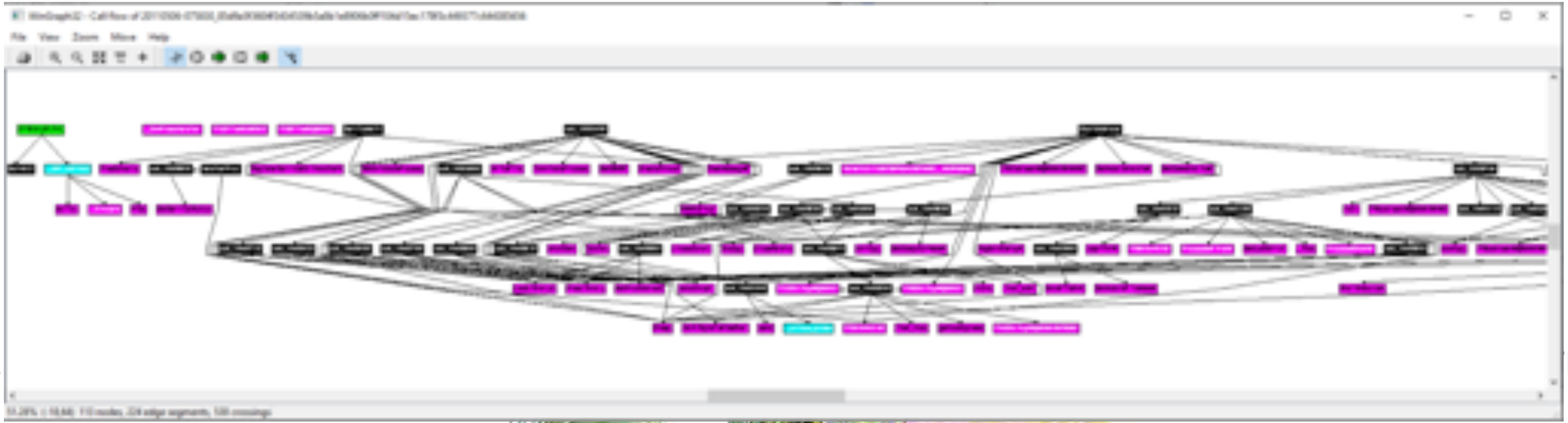
Comparing Call Graphs Task 2





Comparing Call Graphs Task 3





What If There Is Something That Could …

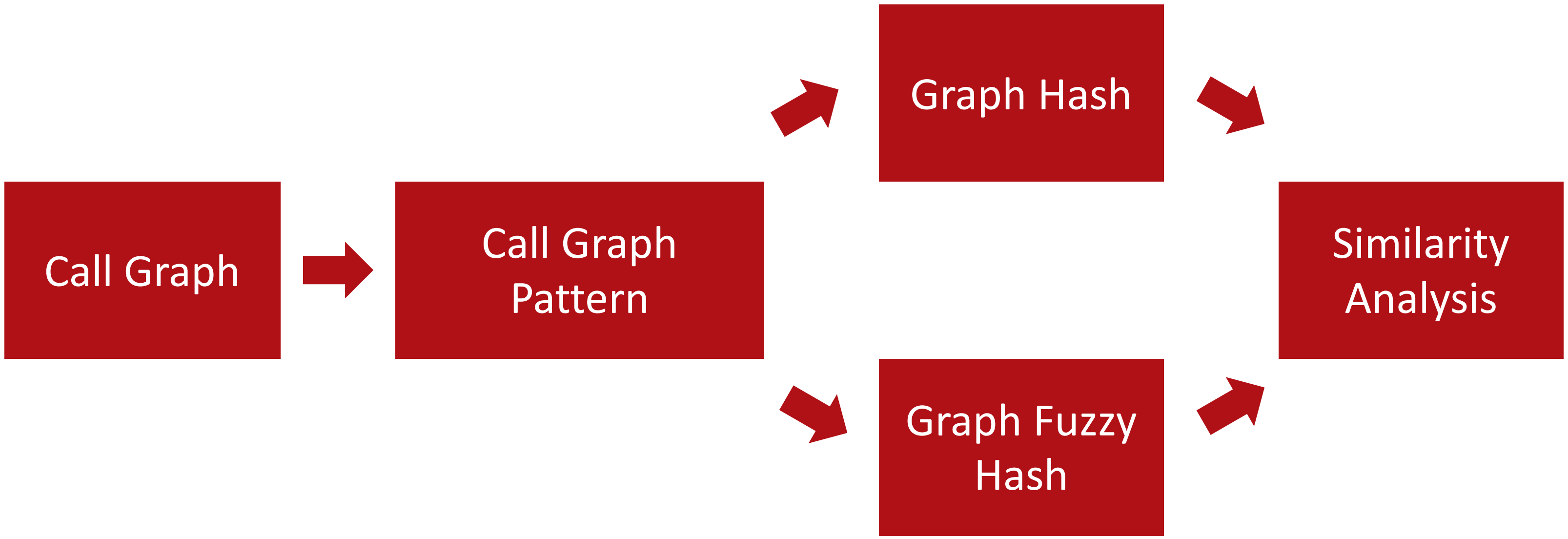
- Present a call graph of a executable
- Not Graph, but binary
- Calculate cryptographic hash of it
- Calculate fuzzy hash of it

Call Graph Pattern (CGP)

Our Methodology

- Hybrid
- CGP is a graph-based pattern
- $f_{\text{Crypto Hash}}$ (CGP)
- $f_{\text{Fuzzy Hash}}$ (CGP)

Methodology Flow



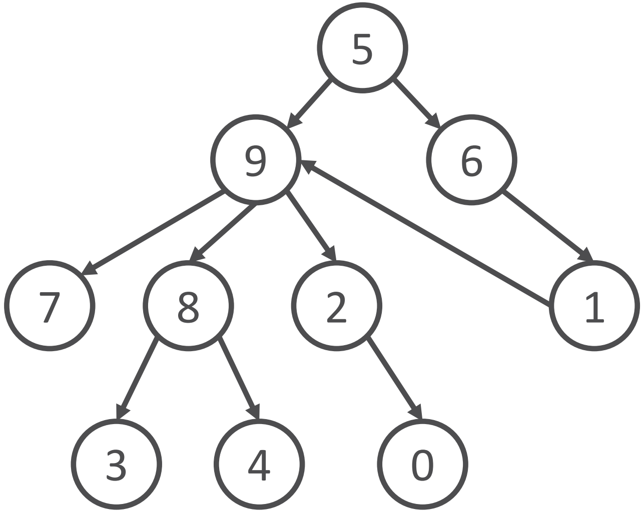
Call Graph

Call Graph / Flow Graph

- Call Graph := {Vertices, Edges}
- Vertices := Functions
- Edges := Vertex A goes to Vertex B (Function A calls Function B)
  - Focus on from one function to other functions

Abstract Call Graph

- Vertices := {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Edges := {1, 9} {2, 0} {5, 9} {5, 6} {6, 1} {8, 3} {8, 4} {9, 7} {9, 8} {9, 2}



Vertices (Functions)

Function name	Segment	Start	Length
sub_401000	.text	00401000	0000009A
StartAddress	.text	004010A0	000000DC
sub_401180	.text	00401180	00000281
sub_401410	.text	00401410	000000DC
sub_4014F0	.text	004014F0	000000EB
sub_4015E0	.text	004015E0	000000C6
sub_4016B0	.text	004016B0	00000094
sub_401750	.text	00401750	0000002F
sub_401780	.text	00401780	000000C3
sub_401850	.text	00401850	0000015A
_main	.text	004019B0	00000A16
Process32NextW	**.text**	**004023F0**	**00000006**
Process32FirstW	**.text**	**004023F6**	**00000006**
CreateToolhelp32Snapshot	**.text**	**004023FC**	**00000006**
operator delete(void *)	.text	00402402	00000006
operator new(uint)	.text	00402408	00000006
__alloca_probe	.text	00402410	0000002F
_except_handler3	.text	00402440	00000006
start	.text	00402446	00000110
_XcptFilter	.text	00402556	00000006
_initterm	.text	0040255C	00000006
__setdefaultprecision			
sub_402574			
nullsub_1			
_controlfp			

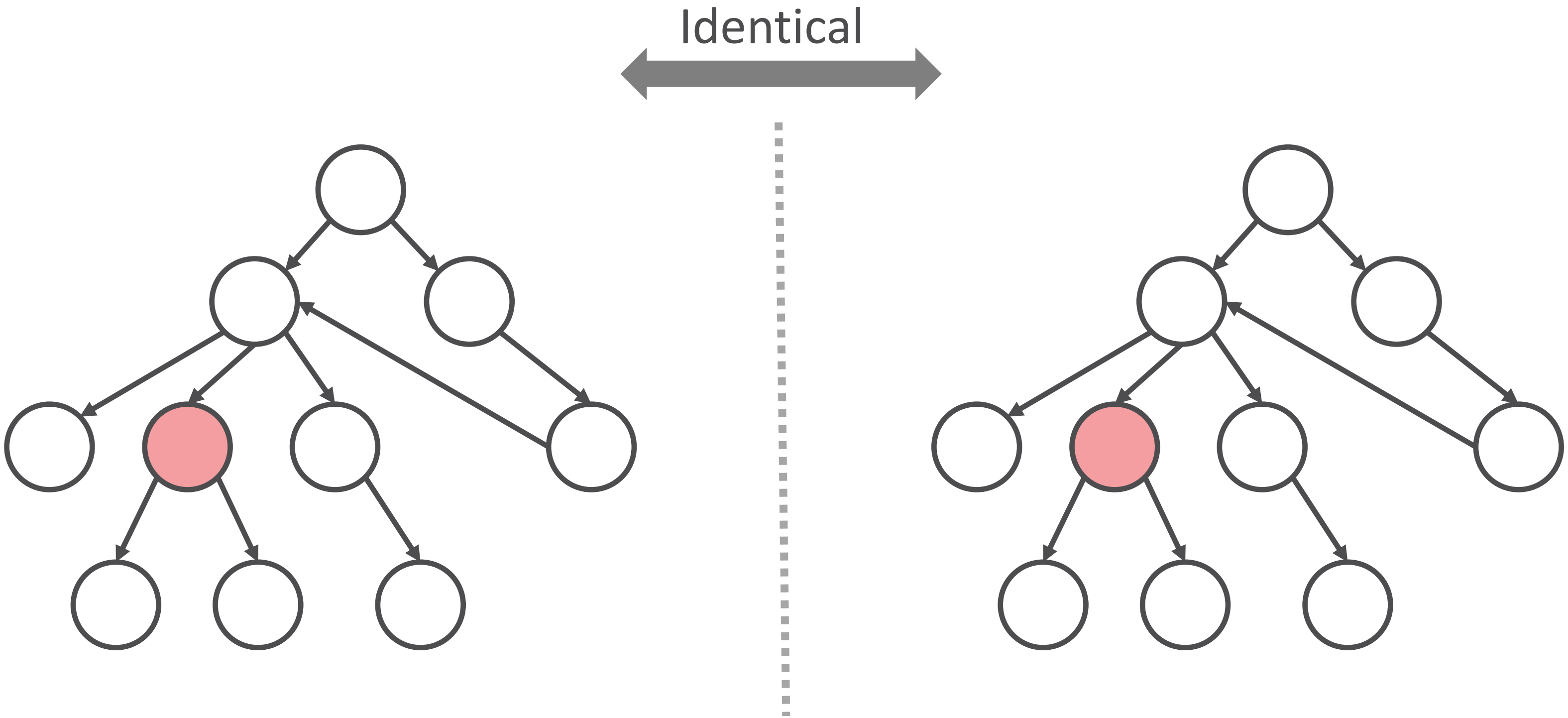
Functions

+

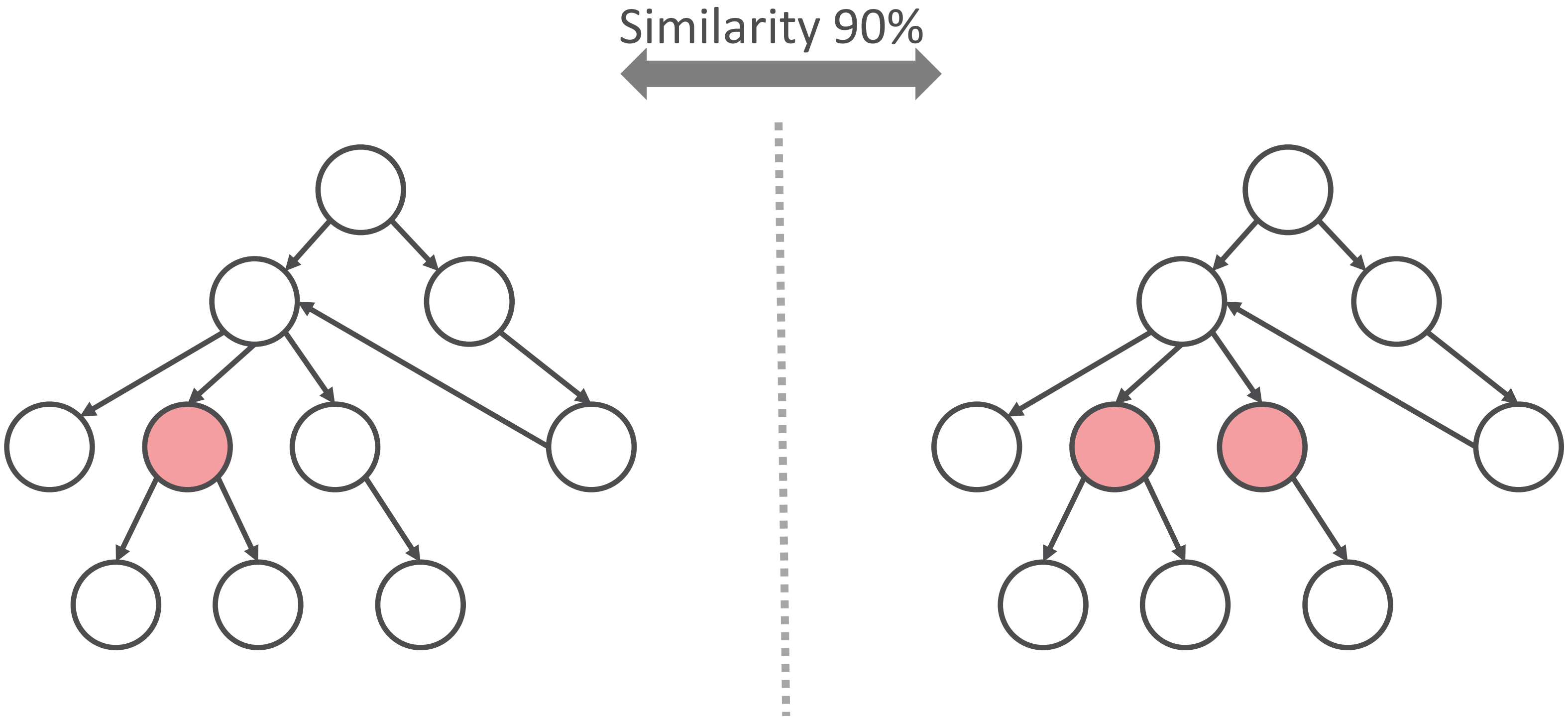
Imported Functions

Address	Ordinal	Name	Library
00403000		RegCloseKey	ADVAPI32
00403004		RegOpenKeyA	ADVAPI32
00403008		RegDeleteValueA	ADVAPI32
0040300C		RegOpenKeyExA	ADVAPI32
00403010		RegQueryValueExA	ADVAPI32
00403018		CreateThread	KERNEL32
0040301C		GetOEMCP	KERNEL32
00403020		CreateProcessW	KERNEL32
00403024		GetSystemDirectoryW	KERNEL32
00403028		GetStartupInfoW	KERNEL32
0040302C		CreatePipe	KERNEL32
00403030		Process32NextW	KERNEL32
00403034		Process32FirstW	KERNEL32
00403038		WriteFile	KERNEL32
0040303C		GetLastError	KERNEL32
00403040		MoveFileExA	KERNEL32
00403044		GetTickCount	KERNEL32
00403048		GetVersionExW	KERNEL32
0040304C		DeleteFileW	KERNEL32
00403050		CreateFileW	KERNEL32
00403054		CloseHandle	KERNEL32
			KERNEL32
			KERNEL32
			KERNEL32
			KERNEL32

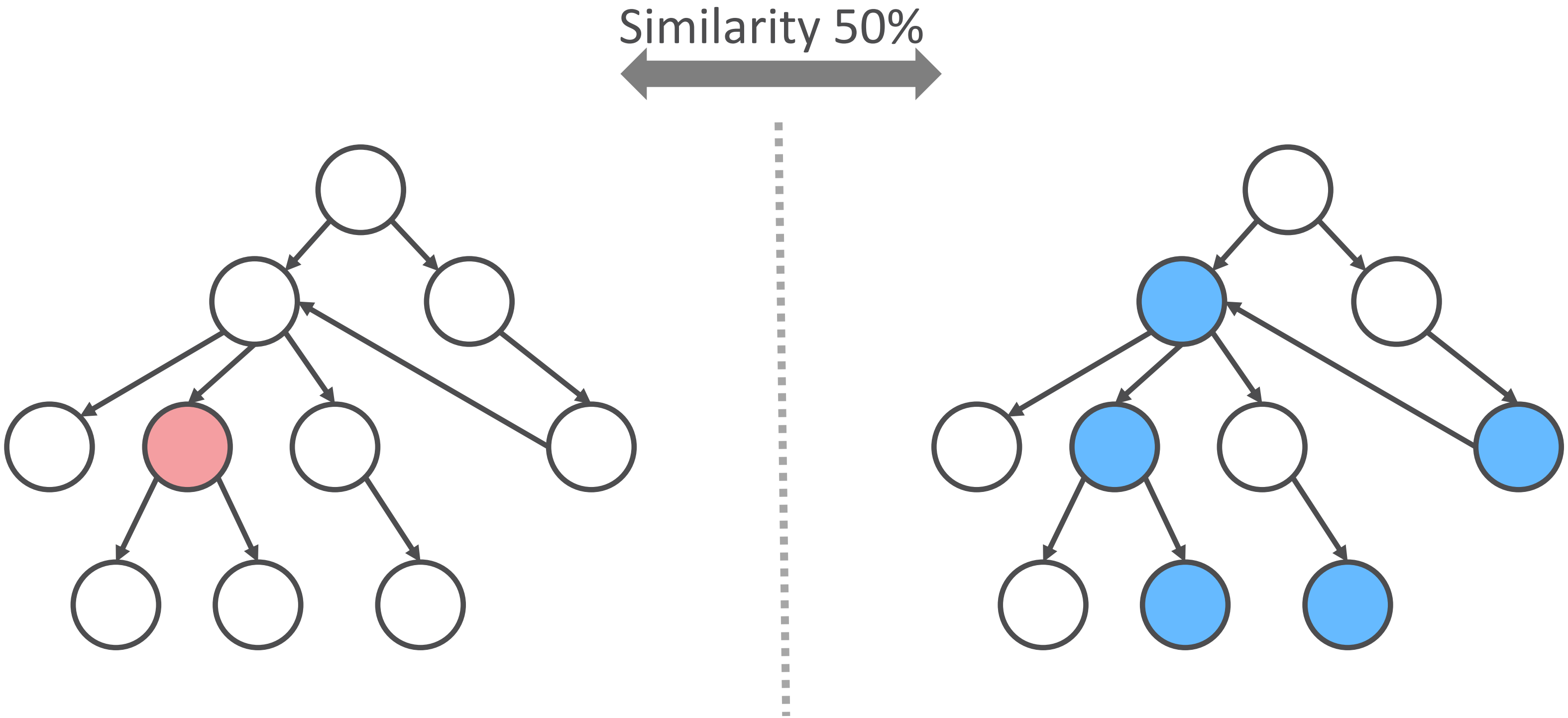
Assign Value to Vertex - Color Vertex (1)



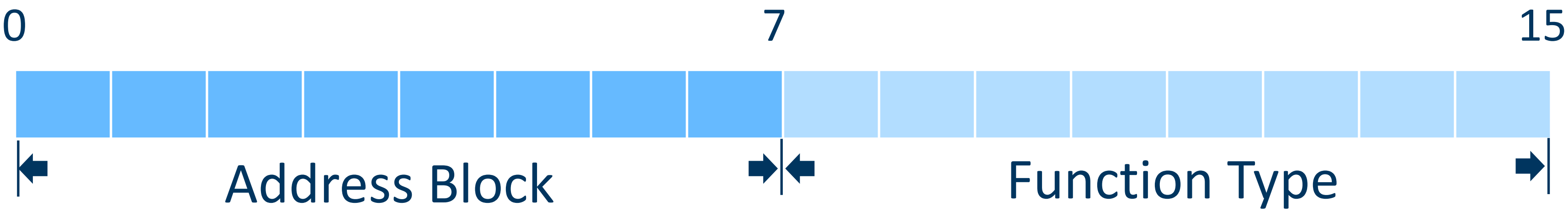
Color Vertex (2)



Color Vertex (3)



One Vertex Value



Address Block := {0 … 15}

Function Type := {0 … 4}

TREND MICRO

Function Types

Function Type	Definition	Value
Regular Function	With full disassembly and isn't library function or imported function	0
Library Function	Well known library function	1
Imported Function	From a dynamic link library	2
Thunk Function	Forwarding its work via an unconditional jump	3
Invalid Function	Invalid function	4

Address Blocks

- Divide whole linear address space into 16 address blocks
- Calculate which address block that each function locates according to its starting address

Edges (Relationship Between Functions)

- Relationship that one function calls other functions

```
.text:0040107D        push    ecx              ; s
.text:0040107E        call    ds:send
.text:00401084        mov     [esp+10h+arg_C], eax
```

```
.text:004010CA        push    1F4h             ; dwMill
.text:004010CF        call    ebx ; Sleep
.text:004010D1        mov     ecx, 400h
```

```
.text:00401160        push    edx              ; s
.text:00401161        call    sub_401000
.text:00401166        add     esp, 10h
```

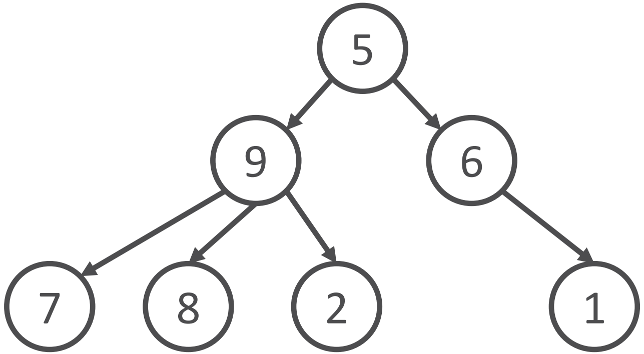
```
{0x401410 => 0x4023fc}
{0x401410 => 0x4023f6}
{0x401410 => 0x401000}
{0x401410 => 0x4023f0}
{0x401410 => 0x403054}
{0x4016b0 => 0x402440}
{0x402446 => 0x402440}
{0x402446 => 0x403074}
{0x402446 => 0x403078}
{0x402446 => 0x40307c}
{0x402446 => 0x403080}
{0x402446 => 0x402577}
{0x402446 => 0x402574}
{0x402446 => 0x403084}
{0x402446 => 0x402562}
{0x402446 => 0x40255c}
{0x402446 => 0x40308c}
{0x402446 => 0x403090}
{0x402446 => 0x4019b0}
{0x402446 => 0x4030bc}
```

Call Graph Traversal Strategy

- Start with root vertex
  - Root vertex is a vertex that has no parent.
- Depth-first Search (DFS)

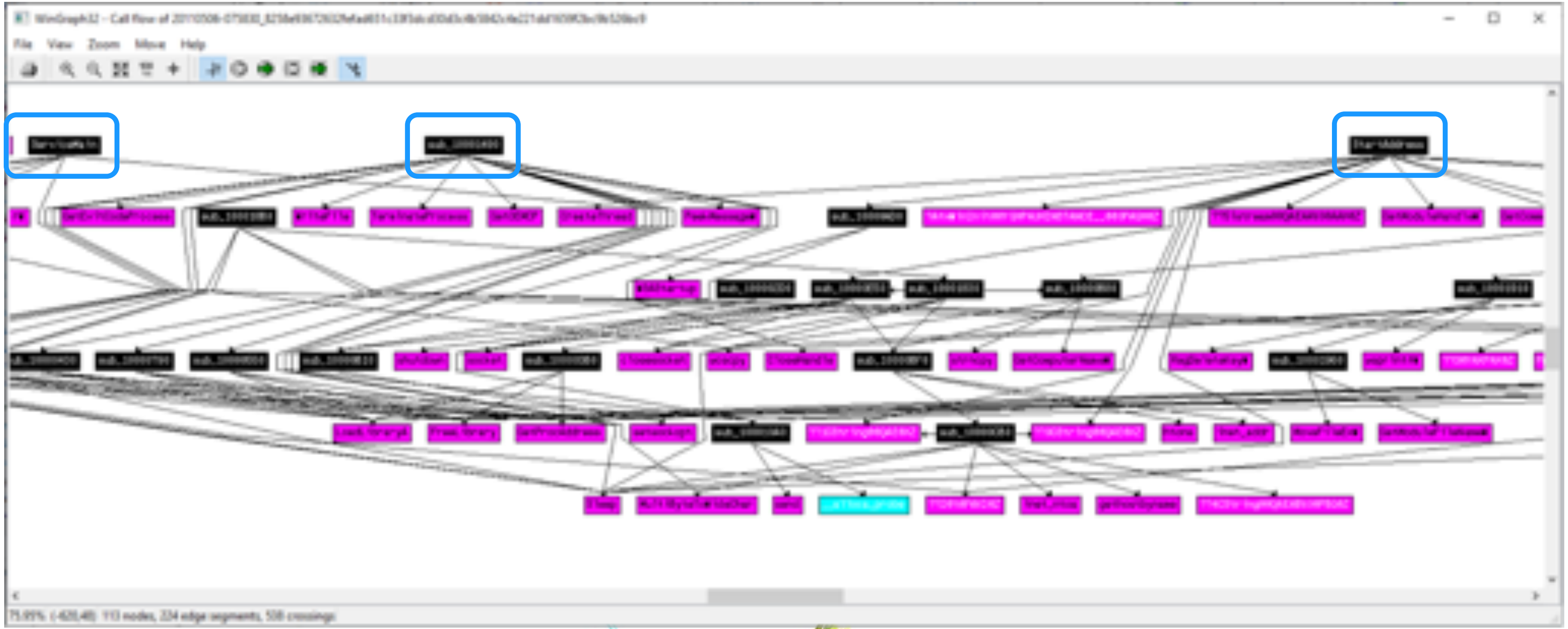
Simple Traversal Example

- Vertices := {1, 2, 5, 6, 7, 8, 9}
- Edges := {5, 9} {5, 6} {6, 1} {9, 7} {9, 8} {9, 2}
- Root := {5}



5 9 7 8 2 6 1

Multiple Root Vertices



Multiple Root Vertices Example

- Windows service DLL
- Exports := {ServiceMain, DllEntryPoint}
- Root Vertices := {ServiceMain, DllEntryPoint}

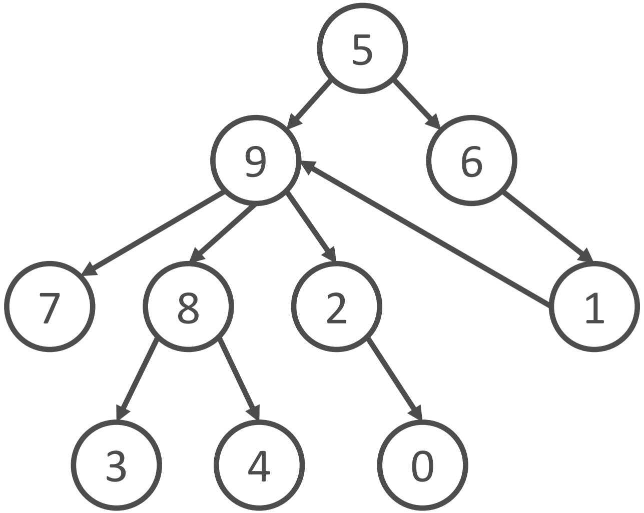
Name	Address	Ordinal
ServiceMain	10000830	1
DllEntryPoint	10001B0A	[main entry]

Function Reuse

- For code reuse
- Avoid redundancy
- Reusing function means visiting reused function vertex and its child vertices more than one time
- Keep only the visited vertex in CGP, without its child vertices

Reused Function Call Graph Example

- Vertices := {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Edges := {1, 9} {2, 0} {5, 9} {5, 6} {6, 1} {8, 3} {8, 4} {9, 7} {9, 8} {9, 2}
- Root := {5}
- Reused Function := {9}



5 9 7 8 3 4 2 0 6 1 9 ~~7 8 3 4 2 0~~

Call Graph Pattern

```
Offset:   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 00 03 00 00 05 02 00 00 00 01 05 03 05 02 00 01
00000010: 05 03 05 02 05 01 05 01 05 02 05 03 05 02 00 00
00000020: 00 01 05 03 05 02 00 01 05 03 05 02 05 01 00 00
00000030: 00 01 05 03 00 01 05 03 05 01 01 00 05 02 01 00
00000040: 01 00 05 02 05 02 05 02 05 02 01 00 01 00 05 02
00000050: 05 02 05 03 05 02 05 02 05 02 05 02 05 03 05 03
00000060: 02 00 05 02 02 00 05 02 02 00 05 02 05 02 05 02
00000070: 02 00 02 00 05 02 05 01 05 02 05 03 05 03 05 02
00000080: 05 03 05 02 05 02 05 02 05 03 05 02 05 02 05 02
00000090: 02 00 05 02 05 02 05 02 00 00 05 02 05 02 05 02
000000a0: 02 00 02 00 03 00 05 01 05 02 05 02 04 00 05 02
000000b0: 05 02 04 00 05 02 05 02 02 00 02 00 02 00 05 02
000000c0: 05 02 05 02 03 00 05 02 03 00 05 02 05 02 03 00
000000d0: 03 00 01 00 05 02 05 02 05 02 05 01 05 03 05 02
000000e0: 05 02 05 02 05 02 05 03 05 02 03 00 05 02 05 01
000000f0: 05 02 03 00 05 02 05 02 05 02 04 00 00 00 05 02
00000100: 05 02 05 02 01 00 05 02 05 02 05 02 05 02 05 02
00000110: 05 02 05 02 05 02 05 02 05 02 00 00 05 02 05 02
00000120: 05 02 01 00 05 02 05 02 05 02 05 01 03 00 01 00
00000130: 05 02 05 02 05 02 05 02 05 02 01 00 05 02 05 02
00000140: 05 02 03 00 05 03 05 02 05 03 05 02 04 00 05 02
00000150: 03 00 05 02 05 02 05 02 05 02 05 02 05 00 05 02
00000160: 05 02 05 00 05 02 05 02 05 02 05 02 05 02 05 02
00000170: 05 02 05 02 05 02 05 02 05 02 05 02 05 01 05 01
00000180: 05 02 05 02 05 03 05 02 05 02 01 00
```

Vertex

Development Environment

- IDA Pro 7.2
- IDApython
- MD5
- ssdeep

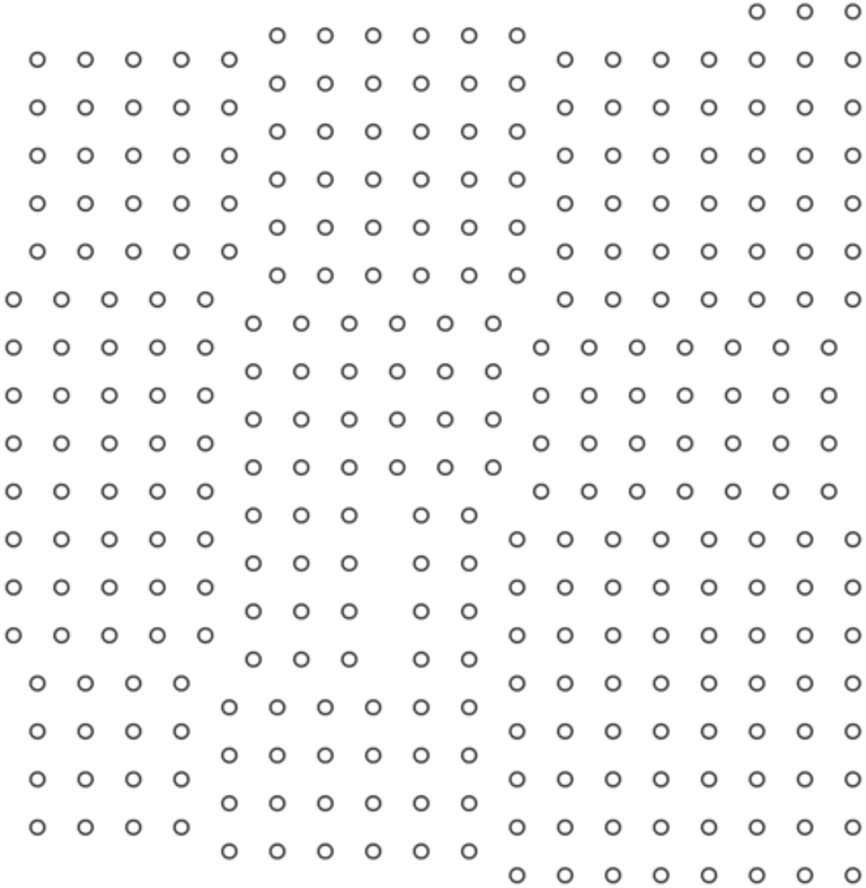
Demo

Evaluation

- Operation Orca
  - Long term cyber espionage
  - Most targets are East Asia countries
  - We disclosed it in 2017

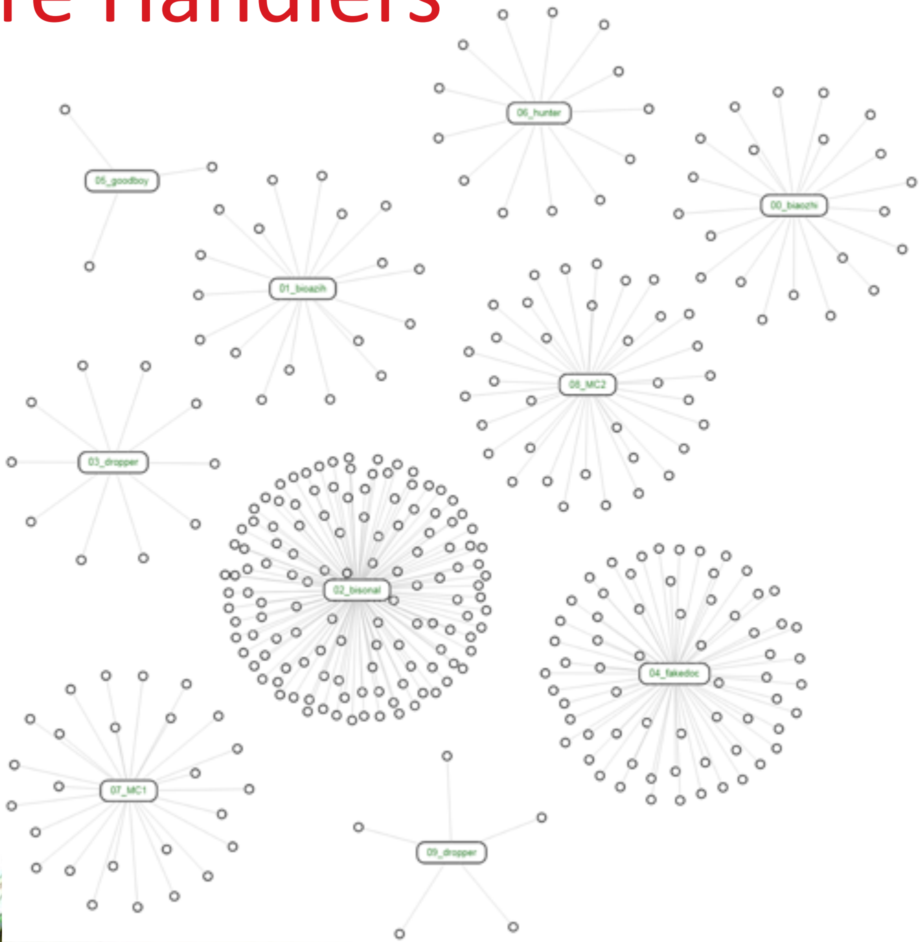
Orca Raw Samples

- 322 distinct samples



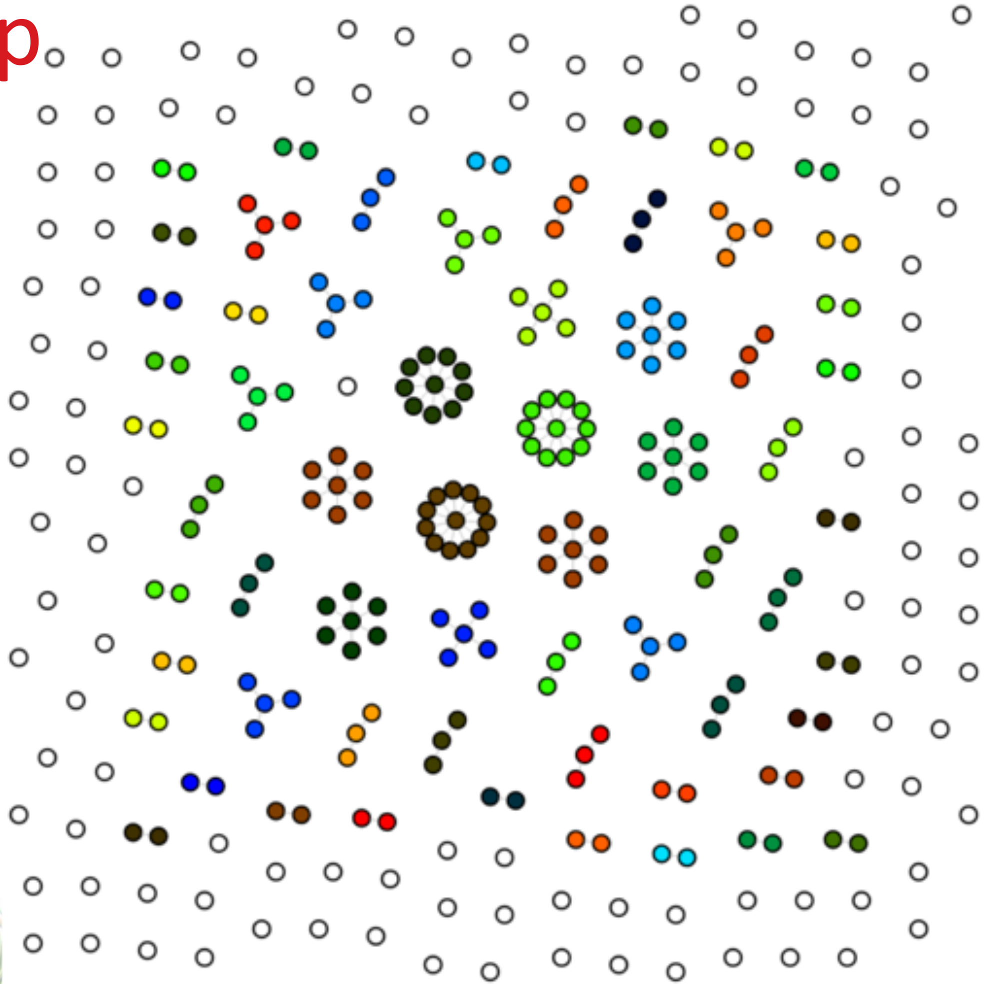
10 Families by Malware Handlers

- 10 Families
- Based on token, communication protocol or C2 used by malware



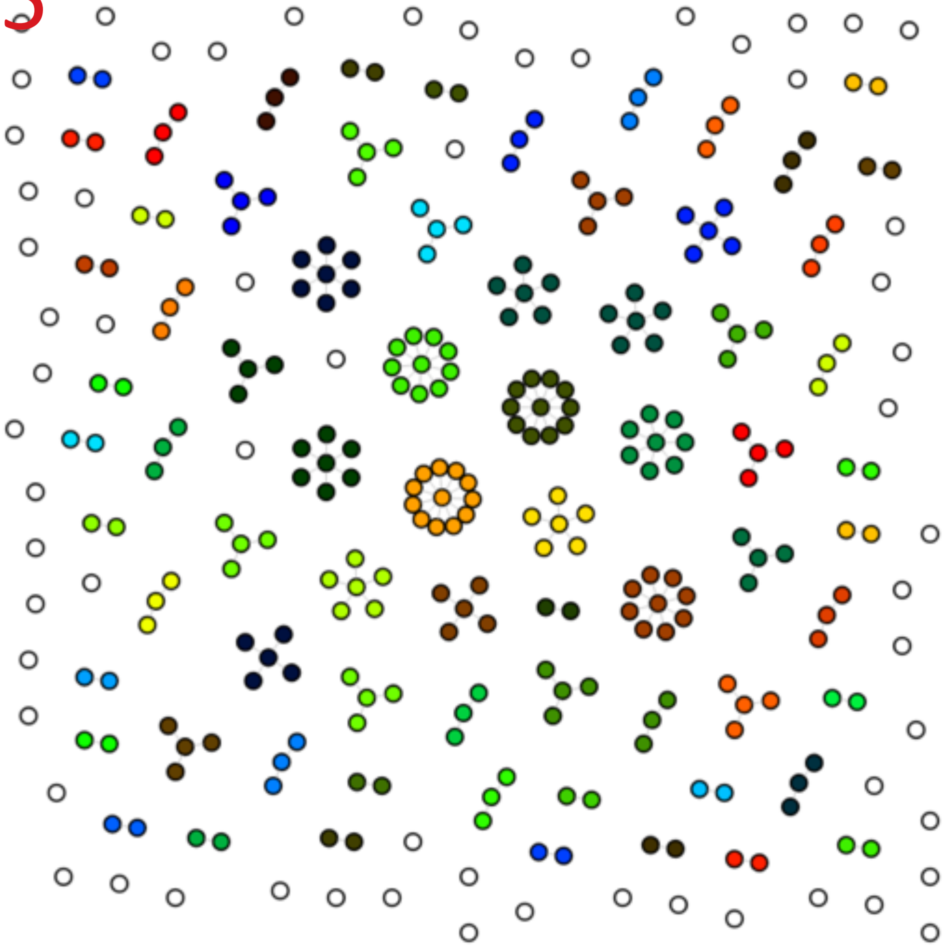
Groups by File ssdeep

- Set ssdeep similarity as 85%
- 211/322 (66%) samples could be grouped
- 62 groups



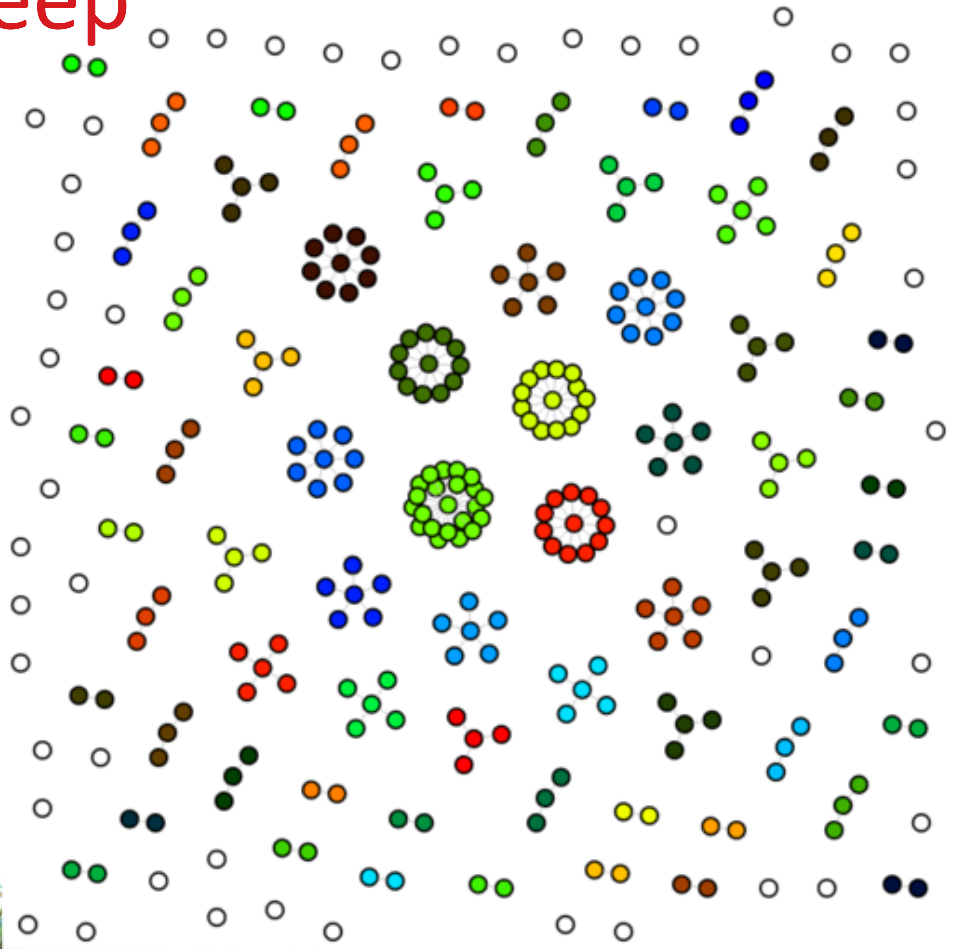
Groups by Graph MD5

- 260/322 (81%) samples could be grouped
- 71 groups



Groups by Graph ssdeep

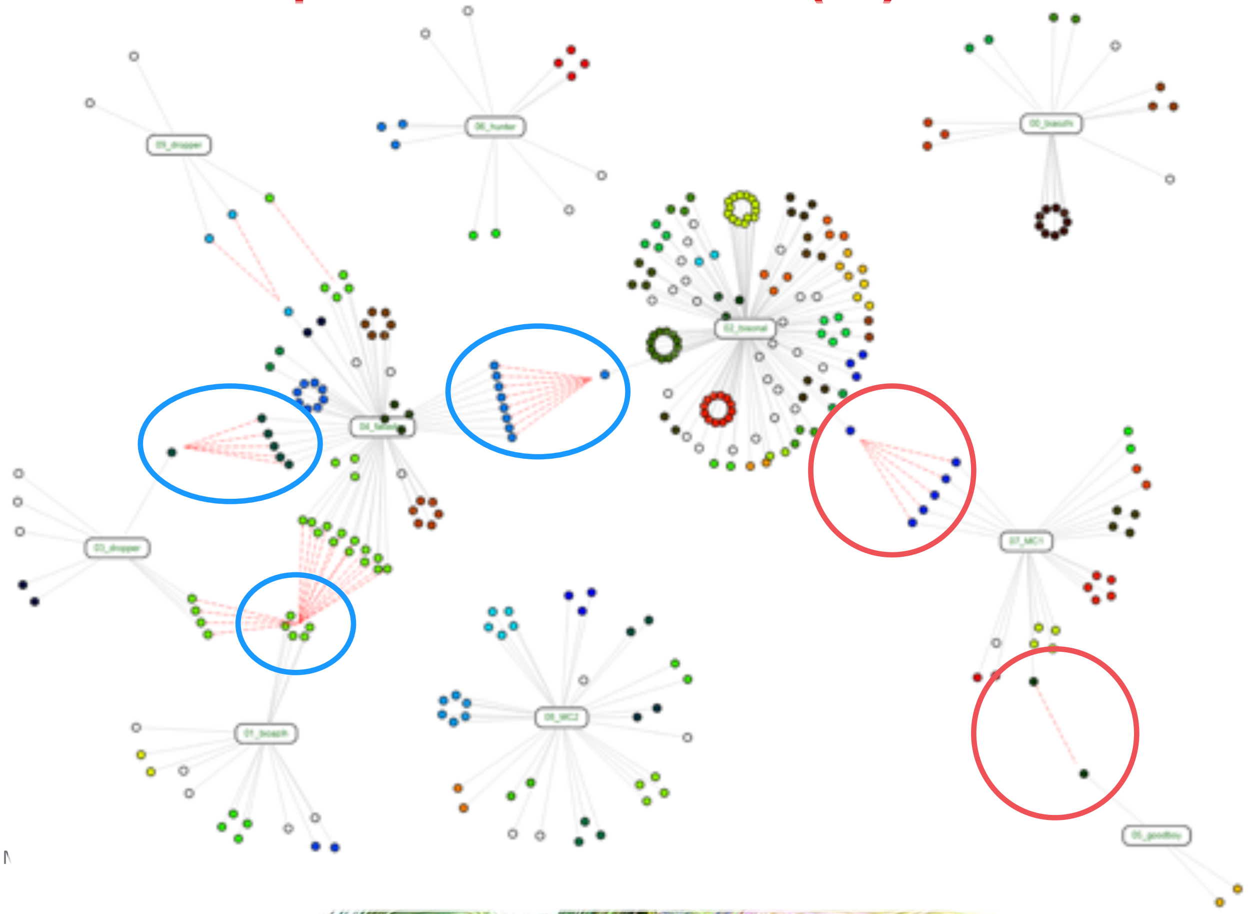
- Set ssdeep similarity as 85%
- 274/322 (85%) samples could be grouped
- 67 groups



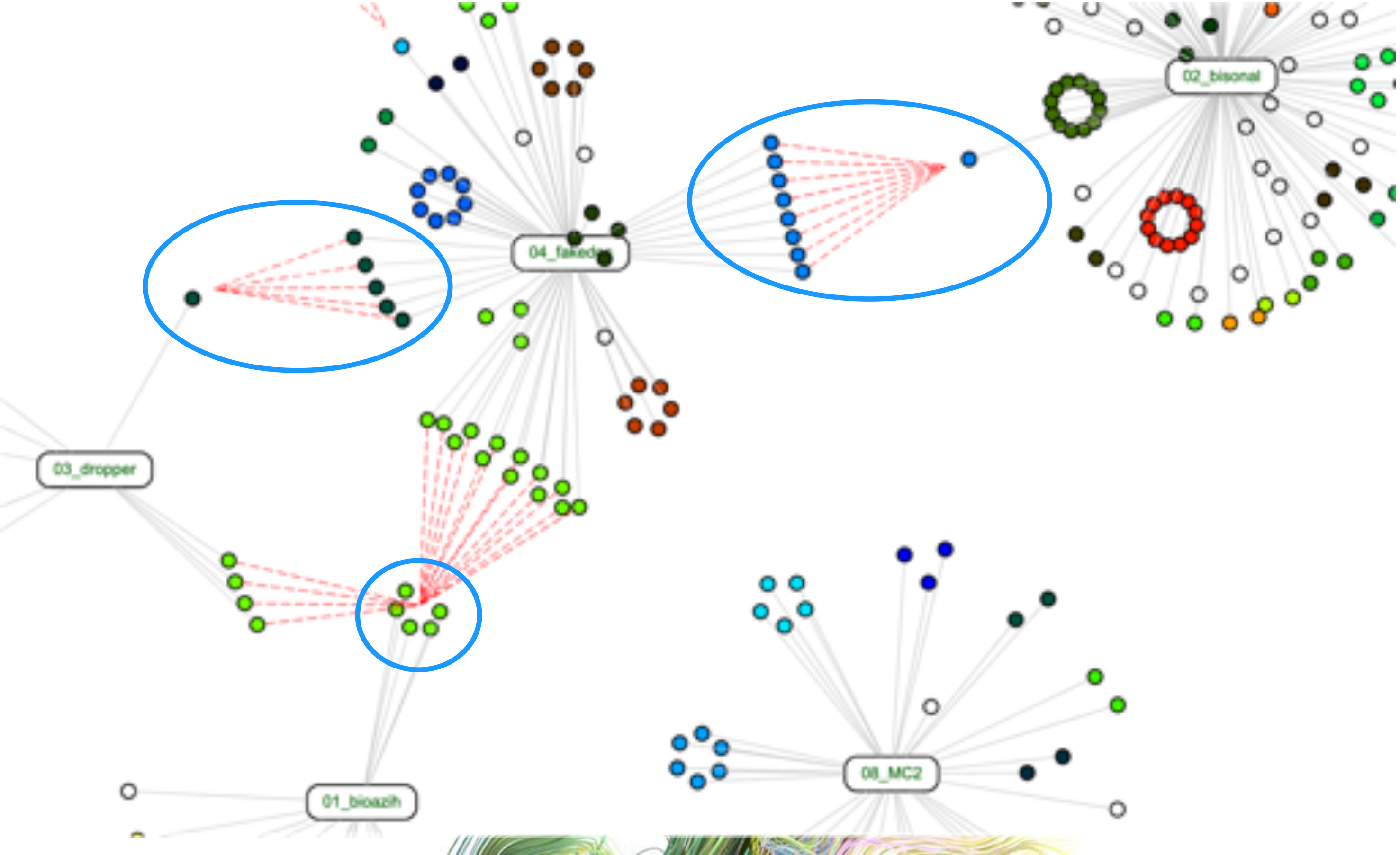
Comparison

	Grouping Rate	vs File ssdeep (GR)	Groups
Graph MD5	81% (260/322)	+15%	71
Graph ssdeep	85% (274/322)	+19%	67
File ssdeep	66% (211/322)	--	62
Malware Handler	100% (322/322)	--	10

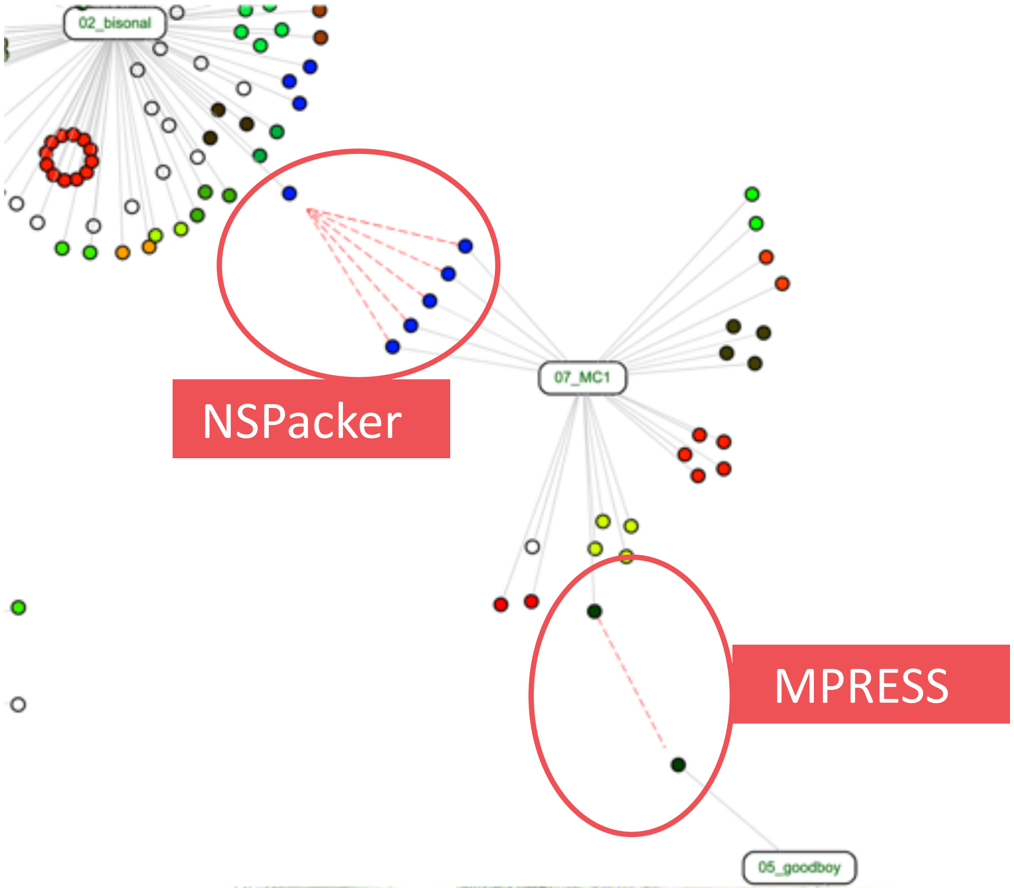
Graph ssdeep vs Families (1)



Graph ssdeep vs Families (2)



Graph ssdeep vs Families (3)



Accuracy Test

- Calculate graph MD5 and graph ssdeep of 10,150 APT samples
- Compare if there are samples classified as the groups of Orca samples
- Only 1 sample from Orca and 2 samples from 10,150 APT samples are classified as the same group
- That’s because these three files share the same packer

Limitation

- Not so good for packers or simple structure executables
  - In some situations, CGP could recognize some packer routines.
- Lean on IDA Pro right now

Future Work

- Benign files test
- ELF and Mach-O files test
  - We have tested on 50 ~ 60 samples of ELF and Mach-O files
  - Work fine so far
- Plugin for Radare2 or Ghidra

Publishing Plan and Schedule

- Publish PoC as open source
- Under internal review
- ASAP
- Update info on @0xvico

Special Thanks

- Kenney Lu
- Serena Lin
- Tunyi Huang

Thank You All

- Chia-Ching Fang
  - vico_fang@trendmicro.com
  - @0xvico
- Shih-Hao Weng
  - shihhao_weng@trendmicro.com

References (1)

- MD5, https://en.wikipedia.org/wiki/MD5
- SHA Family, https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
- Context Triggered Piecewise Hashing, https://www.forensicswiki.org/wiki/Context_Triggered_Piecewise_Hashing
- tlsh, https://github.com/trendmicro/tlsh
- ssdeep, https://ssdeep-project.github.io
- imphash, https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html

References (2)

- BinDiff, https://www.zynamics.com/bindiff.html
- binexport, https://github.com/google/binexport
- impfuzzy, https://blog.jpcert.or.jp/2016/05/classifying-mal-a988.html
- IDA Pro, https://www.hex-rays.com/
- The IDA Pro Book 2nd Edition, http://www.idabook.com/
- Operation Orca, https://www.virusbulletin.com/conference/vb2017/abstracts/operation-orca-cyber-espionage-diving-ocean-least-six-years

