Copyright©2017 NTT Corp. All Rights Reserved.

Innovative R&D by NTT

Similarity Calculation Method for Binary Executables

Asuka Nakajima
NTT Secure Platform Laboratories, Tokyo, Japan

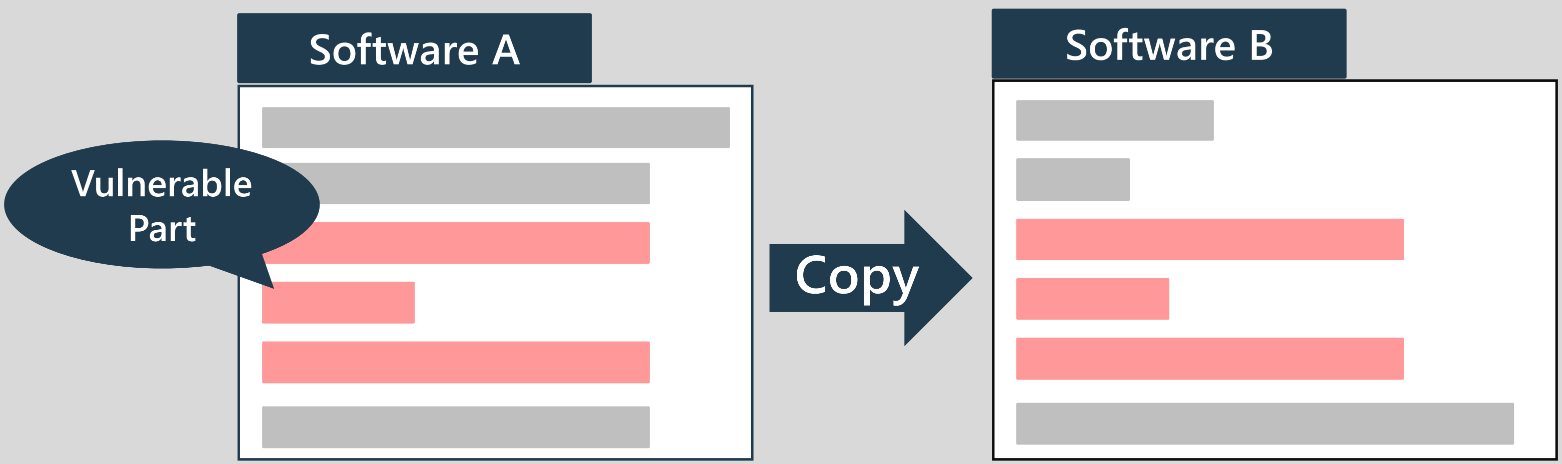
Dagstuhl Seminar 17281, July, 2017

whoami

NTT Confidential

- **Asuka Nakajima**
  - Researcher at NTT Secure Platform Laboratories
  - Reverse Engineering / Vulnerability Discovery
  - Organizer of SECCON CTF / Founder of “CTF for GIRLS” ☺

- **Detection of Code Clone Vulnerability**



Calculate the similarity between vuln func and target binary func

Outline of Today's Talk :
Similarity Calculation Method for Binary Executables

NTT Confidential

1. Background

- Software Similarity Calculation on Malware Analysis Field

2. A Survey on Similarity Calculation Method for Binary Executables

- Overview
- Taxonomy of Program Features
- Challenges
- Research Map

3. About My Research

- Gapped Code Clone Detection in Binary Executables

4. State-of-the-Art Research

- GitZ

Copyright©2016 NTT corp. All Rights Reserved.

NTT Confidential

1. Background

- Software Similarity Calculation on Malware Analysis Field

2. A Survey on Similarity Calculation Method for Binary Executables

- Overview
- Taxonomy of Program Features
- Challenges
- Research Map

3. About My Research

- Gapped Code Clone Detection in Binary Executables

4. State-of-the-Art Research

- GitZ

Copyright©2016 NTT corp. All Rights Reserved.

Background [1/2]

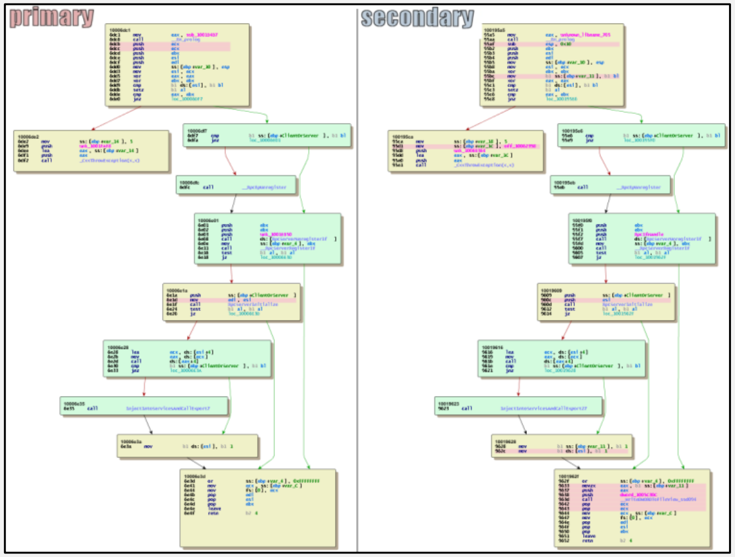
NTT Confidential

Software Similarity Calculation in Malware Analysis Field

"program similarity is a key sub-problem in malware analysis"

[1]Andrew Walenstein and Arun Lakhotia, **"The Software Similarity Problem in Malware Analysis."** In Proceedings Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 10 pp., Dagstuhl, Germany, July 2006.

- **Subspecies**
  - Version update
  - Toolkit-generated malware
  - Metamorphic / Polymorphic malware
- **Code Reuse**
  - e.g. Stuxnet/Duqu
  - Open sourced malware (e.g. Mirai)



▲ **Stuxnet vs Duqu**

https://www.welivesecurity.com/wp-content/media_files/2_1.png

Copyright©2016 NTT corp. All Rights Reserved.

Background [2/2]

NTT Confidential

- **Related Research Area**
  - Copyright Infringement Detection
  - Source Code Plagiarism Detection
  - Vulnerability Discovery
  - Software Evolution Analysis

Similarity Calculation Method Has Become Highly Sophisticated

It can even identify similar function in other binary across different **architecture** or **compilers** or **compilation options** or **OS**

Copyright©2016 NTT corp. All Rights Reserved.

Outline of Today's Talk :
Similarity Calculation Method for Binary Executables

1. Background
- Software Similarity Calculation on Malware Analysis Field

2. A Survey on Similarity Calculation Method for Binary Executables
- Overview
- Taxonomy of Program Features
- Challenges
- Research Map

3. About My Research
- Gapped Code Clone Detection in Binary Executables

4. State-of-the-Art Research
- GitZ

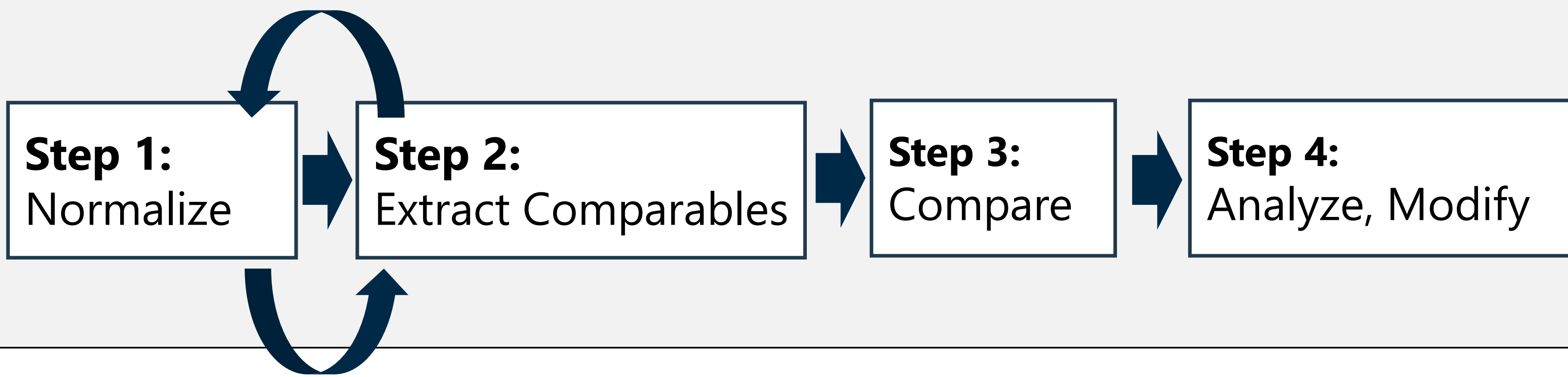
Copyright©2016 NTT corp. All Rights Reserved.

Overview

NTT Confidential

- **Type of Similarity Calculation Method**
  - Dynamic Analysis Based Method
    - Memory Usage Pattern, API call pattern, etc
  - **Static Analysis Based Method**
    - Control Flow Graph, Instruction, etc

Overview of Static Analysis Based Method[1]



Copyright©2016 NTT corp. All Rights Reserved.

Program Features

Category	Example
Graph-based feature	Control Flow Graph
	Data Flow Graph
	Call Graph
Tree-based feature	Abstract Syntax Tree
	S-Expression
Text-based feature	String
	Instructions(opcode/operand)
	API Name
Metric-based feature	Complexity
Hybrid	Graph-based + Text-based
Other	Basic block I/O sampling

Copyright©2016 NTT corp. All Rights Reserved.

Challenges in Similarity Calculation on Binary Executables

Category	Example
Compiler Difference Compiler Optimization Difference	1. Basic Block Reordering
	2. Instruction Reordering
	3. Function Inlining/Outlining
	4. Loop Unrolling
	5. Register Reassignment
Software Evolution	1. Patch
	2. Update
	3. Refactoring
Obfuscation/Metamorphism	1. Dead Code-Insertion
	2. Instruction Substitution
	3. Opaque Predicate Insertion
Encryption and Packing	UPX, etc
Architecture Difference (e.g. IoT Malware)	x86/x86-64/ARM/MIPS/PPC/etc

Research Map of
Static Binary Similarity Calculation [1/3]

~2005	Zeng Wang et.al '00(BMAT)[2]	Halvar Flake '04 [3]	Tomas Dullien '05[4]	Ginger Myles et.al '05 (K-gram)[5]		
~2009	Andreas Sæbjørnsen et.al '09 [8]	Debin Gao et.al '08 (BinHunt)[7]	Seokwoo Choi et.al '07[6]			
2010						
2011	Emily R. Jacobsom et.al '11 [9]	Armijn Hemel et.al '11 (BAT)[10]	Silvio Cesare et.al '11[11]	Jiyong Jang et.al '11 (BitShred)[12]		
2012	Jiang Ming et.al '12 (iBinHunt)[13]					
2013	Beng Heng Ng, et.al '13 (Exposé)[14]	Wei Ming Khoo et.al '13 (Rendezvous)[15]	Martial Bourquin et.al '13 (BinSlayer)[16]	Jiyong Jang et.al '13 (ILINE)[17]	Ming Xu et.al '13 [18]	Arun Lakhotia et.al '13 (BinJuice) [19]
2014	Yaniv David et.al '14 (Tracelet)[20]	Lannan Luo et.al '14 (CoP) [21]	Jannik Pewny et.al '14 (TEDEM) [22]	Mohammad Reza Farhadi et.al '14 (BinClone) [23]		
2015	Jannik Pewny et.al '15 (Multi-MH)[24]	Saed Alrabaee et.al '15 (SIGMA)[25]				
2016	Yaniv David et.al '16 (Esh)[26]	Steven H.H. Ding et.al '16 (Kam1n0) [27]	Mahinthan Chandramohan et.al '16 (BinGo)[28]	Sebastian Eschweiler et.al '16 (discovRE)[29]	Asuka Nakajima et.al '16[30]	Qian Feng et.al '16 [31]
2017	Yaniv David et.al '17 (GitZ)[32]	He Huang et.al '17 (BinSequence)[33]	Qian Feng et.al '17 (XMATCH)[34]			

Software Evolution Analysis (Binary Diffing)

Malware Analysis

Copyright Infringement / Plagiarism Detection

Vulnerability Discovery

General Purpose Binary Similarity Analysis

Copyright©2016 NTT corp. All Rights Reserved.

Research Map of Static Binary Similarity Calculation [2/3]

#		Compiler					Evolution	Architecture	Obfuscation			Encryption Packer
		1	2	3	4	5			1	2	3	
1	Zeng Wang et.al '00(BMAT) [2]	○	△	×	△	○	○	×	×	△	×	×
2	Halvar Flake '04 [3]	△	○	×	△	○	○	△	○	○	×	×
3	Tomas Dullien'05[4]	○	○	×	○	○	○	△	○	○	×	×
4	Ginger Myles et.al '05 (K-gram)[5]	○	○	×	○	○	○	△	×	×	×	×
5	Seokwoo Choi et.al '07 [6]	○	○	×	○	○	△	○	○	○	○	×
6	Debin Gao et.al '08 (BinHunt) [7]	△	○	×	○	○	○	×	○	○	×	×
7	Andreas Sæbjørnsen et.al '09 [8]	×	△	×	△	○	○	×	△	△	△	×
8	Emily R. Jacobsom et.al '11 [9]	○	○	△	○	△	△	×	○	△	△	×
9	Armijn Hemel et.al '11 (BAT) [10]	○	○	×	△	○	△	○	△	△	△	×
10	Silvio Cesare et.al '11[11]	△	○	△	△	○	○	○	○	○	×	△
11	Jiyong Jang et.al '11 (BitShred) [12]	△	×	△	△	×	○	×	△	△	△	△
12	Jiang Ming et.al '12 (iBinHunt) [13]	○	○	○	○	○	○	×	○	○	△	×
13	Beng Heng Ng et.al '13 (Exposé)[14]	○	△	△	○	○	○	×	○	○	×	×
14	Wei Ming Khoo et.al '13(Rendezvous)[15]	△	△	×	△	○	△	×	△	×	×	×
15	Martial Bourquin et.al '13 (BinSlayer)[16]	△	○	×	△	○	○	○	○	○	×	×
16	Jiyong Jang et.al '13 (ILINE)[17]	○	△	△	△	○	○	×	×	○	×	×
17	Ming Xu et.al '13 [18]	○	○	×	△	○	△	×	×	○	×	×

Copyright©2016 NTT corp. All Rights Reserved.

Research Map of Static Binary Similarity Calculation [3/3]

Innovative R&D by NTT

#		Compiler					Evolution	Architecture	Obfuscation			Encryption Packer
		1	2	3	4	5			1	2	3	
18	Arun Lakhotia et.al '13 (BinJuice) [19]	○	○	△	△	○	△	×	△	△	△	×
19	Yaniv David et.al '14 (Tracelet)[20]	○	△	×	×	○	○	×	×	×	×	×
20	Lannan Luo et.al '14 (CoP) [21]	△	○	△	○	○	○	×	○	○	△	×
21	Jannik Pewny et.al '14 (TEDEM) [22]	△	○	×	△	○	○	×	○	○	×	×
22	Mohammad Reza Farhadi et.al '14 (BinClone) [23]	×	△	×	△	○	△	×	△	△	△	×
23	Jannik Pewny et.al '15 (Multi-MH)[24]	○	○	×	○	○	△	○	○	○	△	×
24	Saed Alrabaee et.al '15 (SIGMA)[25]	△	○	×	△	○	○	×	△	△	×	×
25	Yaniv David et.al '16 (Esh)[26]	○	△	×	△	○	△	○	×	×	×	×
26	Steven H.H. Ding et.al '16 (Kam1n0) [27]	○	△	△	△	○	○	×	△	△	×	×
27	Mahinthan Chandramohan et.al '16 (BinGo)[28]	○	○	○	○	○	△	○	○	○	×	×
28	Sebastian Eschweiler et.al '16 (discovRE)[29]	△	○	×	△	○	○	○	×	×	×	×
29	Asuka Nakajima et.al '16[30]	×	△	△	△	○	○	×	△	△	△	×
30	Qian Feng et.al '16 [31]	△	○	×	△	○	△	○	△	△	×	×
31	Yaniv David et.al '17 (GitZ)[32]	○	△	×	△	○	△	○	×	×	×	×
32	He Huang et.al '17 (BinSequence)[33]	△	△	×	△	○	○	×	△	△	×	×
33	Qian Feng et.al '17 (XMATCH)[34]	○	○	△	△	○	△	○	○	○	×	×

Research Map of Static Binary Similarity Calculation [3/3]

#		Compiler					Evolution	Architecture	Obfuscation			Encryption Packer
		1	2	3	4	5			1	2	3	
18	Arun Lakhotia et.al '13 (BinJuice) [19]	○	○	△	△	○	△	×	△	△	△	×
19	Yaniv David et.al '14 (Tracelet)[20]	○	△	×	×	○	○	×	×	×	×	×
20	Lannan Luo et.al '14 (CoP) [21]	△	○	△	○	○	○	×	○	○	△	×
21	Jannik Pewny et.al '14 (TEDEM) [22]	△	○	×	△	○	○	×	○	○	×	×
22	Mohammad Reza Farhadi et.al '14 (BinClone) [23]	×	△	×	△	○	△	×	△	△	△	×
23	Jannik Pewny et.al '15 (Multi-MH)[24]	○	○	×	○	○	△	○	○	○	△	×
24	Saed Alrabaee et.al '15 (SIGMA)[25]	△	○	×	△	○	○	×	△	△	×	×
25	Yaniv David et.al '16 (Esh)[26]	○	△	×	△	○	△	○	×	×	×	×
26	Steven H.H. Ding et.al '16 (Kam1n0) [27]	○	△	△	△	○	○	×	△	△	×	×
27	Mahinthan Chandramohan et.al '16 (BinGo)[28]	○	○	○	○	○	△	○	○	○	×	×
28	Sebastian Eschweiler et.al '16 (discovRE)[29]	△	○	×	△	○	○	○	×	×	×	×
29	Asuka Nakajima et.al '16[30]	×	△	△	△	○	○	×	△	△	△	×
30	Qian Feng et.al '16 [31]	△	○	×	△	○	△	○	△	△	×	×
31	Yaniv David et.al '17 (GitZ)[32]	○	△	×	△	○	△	○	×	×	×	×
32	He Huang et.al '17 (BinSequence)[33]	△	△	×	△	○	○	×	△	△	×	×
33	Qian Feng et.al '17 (XMATCH)[34]	○	○	△	△	○	△	○	○	○	×	×

Outline of Today's Talk :
Similarity Calculation Method for Binary Executables

1. Background

- Software Similarity Calculation on Malware Analysis Field

2. A Survey on Similarity Calculation Method for Binary Executables

- Overview
- Taxonomy of Program Features
- Challenges
- Research Map

3. About My Research

- Gapped Code Clone Detection in Binary Executables

4. State-of-the-Art Research

- GitZ

Copyright©2016 NTT corp. All Rights Reserved.

About My Research:
Gapped Code Clone Detection in Binary Executables

NTT Confidential

- **Background & Motivation**
  - Discover Code Clone Vulnerability in Binary Executables
    - Windows, Adobe Reader, etc
  - Method that can discover Gapped Code Clone Vulnerability
    - Source Code Modification (add multiple lines, I/O Change)

It can be also applied to malware analysis field

Copyright©2016 NTT corp. All Rights Reserved.

About My Research:
Gapped Code Clone Detection in Binary Executables

Innovative R&D by NTT

NTT Confidential

System Overview

Input

Binaryfile
- Unpatched vuln
- Target Binary File

Step1: Disassemble

Step2: Normalization

Step3: Similarity Calculation

- Needleman-Wunsh Algorithm
- Applied "Affine Gap Penalty"

Output

Vulnerability Candidate

System Image

Vuln part
push REG
mov REG REG
mov REG VAL
call MEM
・・・

Similarity Calculation

Target Binary
mov REG REG
push REG
mov REG REG
push REG
push REG
mov REG MEM
mov REG MEM
lea REG MEM
・・・

Similarity

N%

About My Research:
Gapped Code Clone Detection in Binary Executables

Innovative R&D by NTT

NTT Confidential

LCS (Global Alignment)

String1 (source)	String2 (dest)
mov REG REG	push REG REG
mov REG REG	push REG REG
call MEM	call MEM
test REG REG	test REG REG
	jmp MEM
	xor REG REG
	pop REG
	pop REG
	・
	・

Search all similar part between two given string

Smith-Waterman (Local Alignment)

String1 (source)	String2 (dest)
mov REG REG	push REG REG
mov REG REG	push REG REG
call MEM	call MEM
test REG REG	test REG REG
	jmp MEM
	xor REG REG
	pop REG
	pop REG
	・
	・

Search similar region between two given strings

Needleman-Wunsch (Semi-Global Alignment)

String1 (source)	String2 (dest)
mov REG REG	mov REG REG
mov REG REG	push REG REG
call MEM	push REG REG
test REG REG	call MEM
	test REG REG
	jmp MEM
	xor REG REG
	pop REG
	pop REG
	・

Search the region (in string2) that best matches to string1

Needleman-Wunsch is most suitable

NTT

Copyright©2016 NTT corp. All Rights Reserved.

Approach: Disassemble & Normalization

Innovative R&D by NTT

NTT Confidential

1 Disassemble

- Binary File(unpatched vuln)
- Target Binary File

2 Normalization (Operand)

Different assembly(operand) will be generated even the source code is same※

Before	After
Immediate val	VAL
Memory	MEM
Register	REG

mov eax ecx ➡ mov REG REG

※Example

Original	Copy
shr rdx,1 lea rdi, [rdx+0x4] call 3f3d0	shr rdx,1 lea rdi, [rdx+0x4] call 41d630

Original	Copy

xor ebx, ebx
add rsp, 38h
mov eax, ebx
pop rbx
pop rbp
pop r12
pop r13
retn
	xor r12d, r12d
	add rsp, 38h
	mov eax, r12d
	pop rbx
	pop rbp
	pop r12
	pop r13
	retn

NTT

Copyright©2016 NTT corp. All Rights Reserved.

Approach: Similarity Calculation [1/3]

NTT Confidential

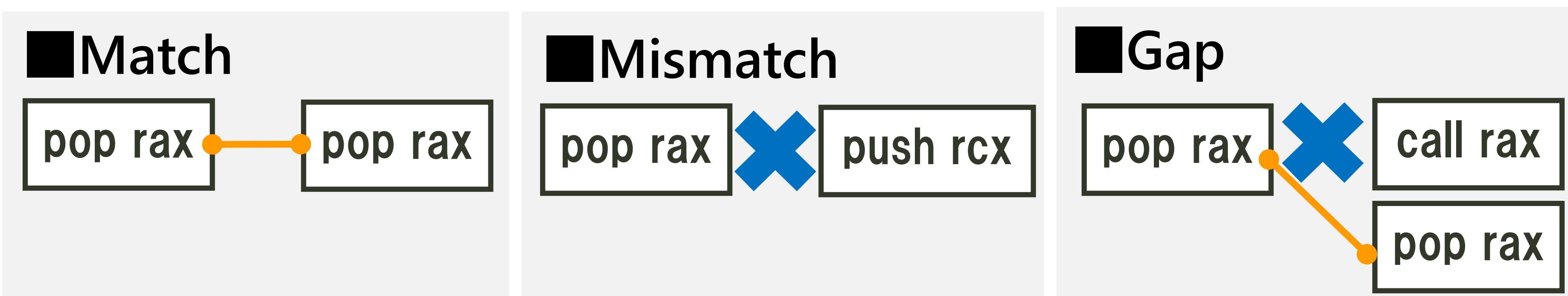
$$Similarity = \frac{Score\ of\ Most\ Similar\ Part}{Maximum\ Score(All\ Matched\ Case)}$$

● Score Calculation

Needleman-Wunsch(Normal Gap)	
match	+2point
mismatch	–2point
gap	–1point

Needleman-Wunsch (AffineGap)	
match	+2point
mismatch	-2point
open gap※	-3point
extended gap	-0.5point

Distinct the Gap



※Open gap：The first gap of multiple gaps

Copyright©2016 NTT corp. All Rights Reserved.

Approach: Similarity Calculation [3/3]

Innovative R&D by NTT

NTT Confidential

Affine Gap penalty can mitigate the significant score drop due to the source code modification

Source Code

```
int main(int argc, char* argv[]){

  if(argc !=2){
    printf("Usage:%s <your name>¥n", argv[0]);
    return 1;
  }

  printf("Argument:%d,%s¥n",argc,argv[1]);
  printf("Hello World! %s¥n", argv[1]);

  return 0;
}
```

Adding 1L Source Code = Adding 8L Assembly Code

Assembly

```
push   ebp
mov    ebp, esp
and    esp, 0xfffffff0
sub    esp, 0x10
cmp    DWORD PTR [ebp+0x8], 0x2
je     0x8048448 <main+43>
mov    eax, DWORD PTR [ebp+0xc]
mov    eax, DWORD PTR [eax]
mov    DWORD PTR [esp+0x4], eax
mov    DWORD PTR [esp], 0x8048520
call   0x80482f0 <printf@plt>
mov    eax, 0x1
jmp    0x8048484 <main+103>
mov    eax, DWORD PTR [ebp+0xc]
add    eax, 0x4
mov    eax, DWORD PTR [eax]
mov    DWORD PTR [esp+0x8], eax
mov    eax, DWORD PTR [ebp+0x8]
mov    DWORD PTR [esp+0x4], eax
mov    DWORD PTR [esp], 0x8048536
call   0x80482f0 <printf@plt>
mov    eax, DWORD PTR [ebp+0xc]
add    eax, 0x4
mov    eax, DWORD PTR [eax]
mov    DWORD PTR [esp+0x4], eax
mov    DWORD PTR [esp], 0x8048546
call   0x80482f0 <printf@plt>
mov    eax, 0x0
leave
ret
```

- Normal gap

$$22 \times 2 = 44$$
$$8 \times -1 = -8$$

Total36p

- Affine Gap

$$22 \times 2 = 44$$
$$1 \times -3 = -3$$
$$7 \times -0.5 = -3.5$$

Total37.5p

NTT

Copyright©2016 NTT corp. All Rights Reserved.

Evaluation

[GOAL] Evaluate the validity of the approach

- Calculate the similarity between original and copied binary

Vuln1 (CVE-2008-4314) Original → Vuln1 (CVE-2008-4314) Copy

?%

Vuln2 (CVE-2008-5023) Original → Vuln2 (CVE-2008-5023) Copy

- Calculate the similarity between original and dataset binary

Vuln1 (CVE-2008-4314) Original

?%

Vuln2 (CVE-2008-5023) Original

→ Dataset(432 binary)

Ubuntu12.04
/bin,/usr/lib
(x86-64/ELF)

[score setting] Match2p, Mismatch -2p, Opengap-3p, Extendedgap-0.5p

NTT

Copyright©2016 NTT corp. All Rights Reserved.

Case1: CVE-2008-4316 (Source Code)

Original
[Glib]

```
g_base64_encode (const guchar *data,gsize len){
  gchar *out;
  gint state = 0, outlen;
  gint save = 0;

  g_return_val_if_fail (data != NULL, NU
  g_return_val_if_fail (len > 0, NULL);

  out = g_malloc (len * 4 / 3 + 4);
  outlen = g_base64_encode_step (data, len, FALSE, out, &state, &save);
  outlen += g_base64_encode_close (FALSE, out + outlen, &state, &save);
  out[outlen] = '¥0';
  return (gchar *) out;
}
```

2 lines are deleted

Copy
[Seahorse]

```
seahorse_base64_encode (const guchar *data,gsize len){
  gchar *out;
  gint state = 0, outlen;
  gint save = 0;

  out = g_malloc (len * 4 / 3 + 4);
  outlen = seahorse_base64_encode_step (data, len, FALSE, out, &state, &save);
  outlen += seahorse_base64_encode_close (FALSE,out + outlen,&state,&save);
  out[outlen] = '¥0';
  return (gchar *) out;
}
```

Copyright©2016 NTT corp. All Rights Reserved.

Case2: CVE-2008-5023 (Source Code)

Original
[Firefox]

```
PRBool nsXBLBinding::AllowScripts(){
  PRBool result;
  mPrototypeBinding->GetAllowScripts(&result);
  …
  nsCOMPtr<nsIDocument> ourDocument;
  mPrototypeBinding->XBLDocumentInfo()->GetDocument(getter_AddRefs(ourDocument));
  PRBool canExecute;
  nsresult rv = mgr->CanExecuteScripts(cx, ourDocument->NodePrincipal(), &canExecute);
  return NS_SUCCEEDED(rv) && canExecute;
}
```

Copy
[Seamonkey]

```
PRBool nsXBLBinding::AllowScripts(){
  PRBool result;
  mPrototypeBinding->GetAllowScripts(&result);
  …
  nsCOMPtr<nsIDocument> ourDocument;
  mPrototypeBinding->XBLDocumentInfo()->GetDocument(getter_AddRefs(ourDocument));
  nsIPrincipal* principal = ourDocument->GetPrincipal();
  if (!principal) {
    return PR_FALSE;
  }
  PRBool canExecute;
  nsresult rv = mgr->CanExecuteScripts(cx, principal, &canExecute);
  return NS_SUCCEEDED(rv) && canExecute;
}
```

4 lines are added &
1 line is modified

Copyright©2016 NTT corp. All Rights Reserved.

Evaluation 1

NTT Confidential

CVE-ID	Original	Copy	Similarity (unpatched)	Similarity (patched)	Max similarity (Dataset)
CVE-2008-4316	Glib	Seahorse	60.7%	11.5%	9.2%
CVE-2008-5023	Firefox	Seamonkey	68.8%	38.0%	9.7%

- The extracted part was the copied vulnerable part
- Similarity between the dataset was maximum 9.7%

Detected codeclone vulnerability in binary executables, even there was source code modification

Copyright©2016 NTT corp. All Rights Reserved.

Evaluation 2 [1/2]

Innovative R&D by NTT

[GOAL] Detect codeclone vulnerability from real world software product

21 Vulnerabilities		
CVE-2015-1635	CVE-2010-0028	CVE-2015-1793
CVE-2014-0301	CVE-2008-4250	CVE-2015-1790
CVE-2013-5058	CVE-2008-4028	CVE-2015-1789
CVE-2013-0030	CVE-2007-1794	CVE-2015-0292
CVE-2011-2005	CVE-2007-0024	CVE-2015-0288
CVE-2011-0658	CVE-2006-4691	CVE-2015-0287
CVE-2010-0816	CVE-2006-0021	CVE-2015-0286

- 14 vulnerabilities from Windows
- 7 vulnerabilities from OpenSSL

?%

40945 binary files

Windows XP.
Windows Vista,
Windows 7
Windows 8.1
Windows Server
Virus Total(NSRL)

[Score setting]match2p,mismatch-2p,opengap-3p,extendedgap-0.5p
[Threshold]　20%

NTT

Copyright©2016 NTT corp. All Rights Reserved.

Evaluation [2/2]

NTT Confidential

- Candidate of codeclone vulnerability

CVE-ID	Original	Copy	Similarity	Result
CVE-2008-4250	netapi32.dll (5.1.2600.2952)	netlogon.dll (5.2.3790.1830)	37.7%	☺
CVE-2011-0658	oleaut32.dll (5.2.3790.4202)	olepro32.dll (6.1.7601.17514)	75.1%	☹ Deadcode
CVE-2015-1789	libeay32.dll (0.9.8.31)	JunosPulseVpnBg.dll (1.0.0.206)	43.9%	☺
CVE-2015-1793	libeay32.dll (1.0.1.15)	JunosPulseVpnBg.dll (1.0.0.206)	39.0%	☹ No attack vector

Copyright©2016 NTT corp. All Rights Reserved.

CVE-2008-4520 (MS08-067)

- **Details**
  - It was real codeclone BoF vulnerability !
    - → Vulnerability which was used by Conficker Worm
  - [original] netapi32.dll [copy] netlogon.dll

Original

```
.text:5925A180 ; int __stdcall CanonicalizePathName(wchar_t *Source, wc
.text:5925A180 _CanonicalizePathName@20 proc near      ; CODE XREF: Ne
.text:5925A180
.text:5925A180 var_420          = dword ptr -420h
.text:5925A180 var_41C          = dword ptr -41Ch
.text:5925A180 Dest             = word ptr -418h
.text:5925A180 var_4            = dword ptr -4
.text:5925A180 Source           = dword ptr  8
.text:5925A180 Str              = dword ptr  0Ch
.text:5925A180 arg_8            = dword ptr  10h
.text:5925A180 arg_C            = dword ptr  14h
.text:5925A180 arg_10           = dword ptr  18h
.text:5925A180
.text:5925A180 ; FUNCTION CHUNK AT .text:59268864 SIZE 000000AB BYTES
.text:5925A180
.text:5925A180                  mov     edi, edi
.text:5925A182                  push    ebp
.text:5925A183                  mov     ebp, esp
.text:5925A185                  sub     esp, 420h
.text:5925A18B                  mov     eax, ___security_cookie
.text:5925A190                  push    ebx
.text:5925A191                  mov     ebx, [ebp+Str]
.text:5925A194                  mov     [ebp+var_4], eax
.text:5925A197                  mov     eax, [ebp+arg_8]
.text:5925A19A                  push    esi
.text:5925A19B                  mov     esi, [ebp+Source]
.text:5925A19E                  push    edi
.text:5925A19F                  mov     [ebp+var_41C], eax
.text:5925A1A5                  mov     eax, [ebp+arg_10]
```

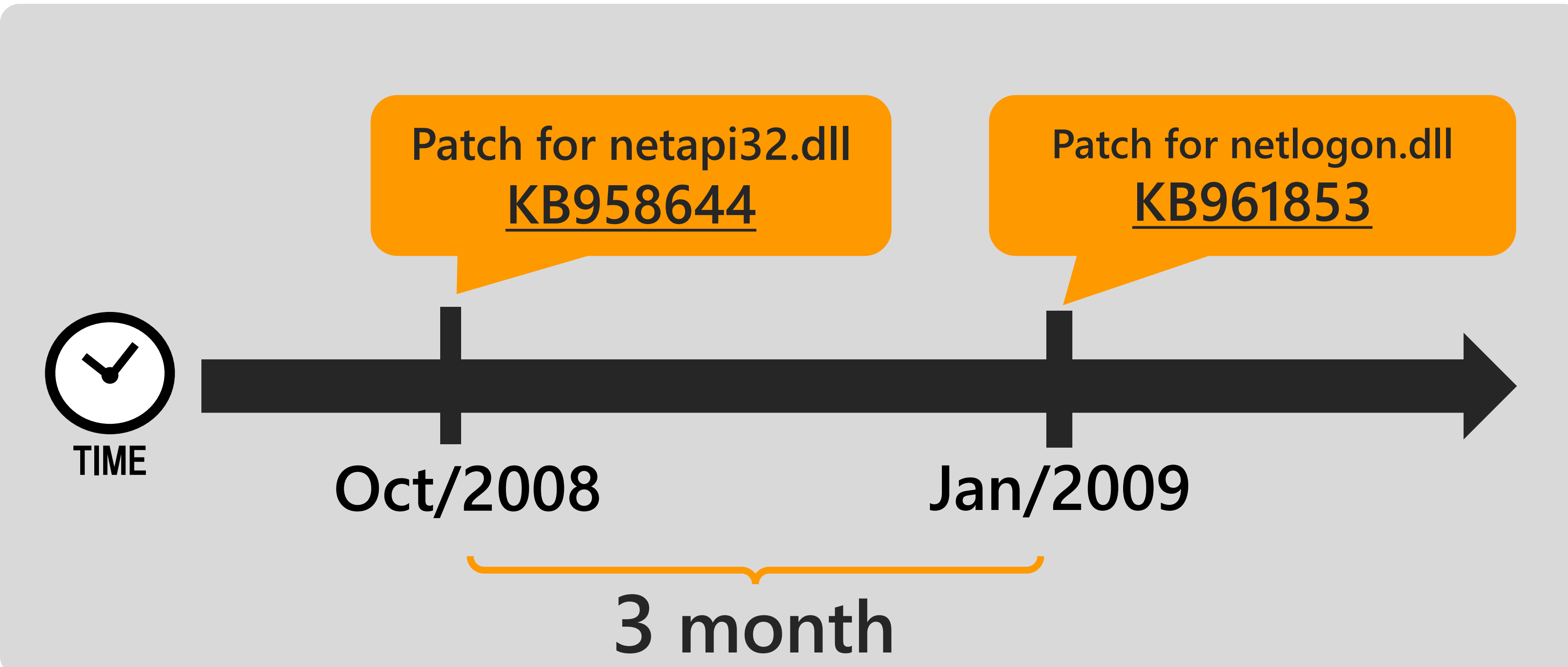
Copy

```
.text:7423C933 ; int __stdcall CanonicalizePathName(wchar_t *Str, wchar_t *
.text:7423C933 _CanonicalizePathName@20 proc near      ; CODE XREF: NetpwPa
.text:7423C933
.text:7423C933 var_420          = dword ptr -420h
.text:7423C933 var_41C          = dword ptr -41Ch
.text:7423C933 Dest             = word ptr -418h
.text:7423C933 var_4            = dword ptr -4
.text:7423C933 Str              = dword ptr  8
.text:7423C933 Source           = dword ptr  0Ch
.text:7423C933 arg_8            = dword ptr  10h
.text:7423C933 arg_C            = dword ptr  14h
.text:7423C933 arg_10           = dword ptr  18h
.text:7423C933
.text:7423C933                  mov     edi, edi
.text:7423C935                  push    ebp
.text:7423C936                  mov     ebp, esp
.text:7423C938                  sub     esp, 420h
.text:7423C93E                  mov     eax, ___security_cookie
.text:7423C943                  push    ebx
.text:7423C944                  mov     [ebp+var_4], eax
.text:7423C947                  mov     eax, [ebp+arg_8]
.text:7423C94A                  push    esi
.text:7423C94B                  mov     esi, [ebp+Str]
.text:7423C94E                  mov     [ebp+var_41C], eax
.text:7423C954                  mov     eax, [ebp+arg_10]
.text:7423C957                  xor     ebx, ebx
.text:7423C959                  cmp     esi, ebx
.text:7423C95B                  push    edi
.text:7423C95C                  mov     edi, [ebp+Source]
.text:7423C95F                  mov     [ebp+var_420], eax
.text:7423C965                  jz      short loc_7423C9CD
.text:7423C967                  push    esi             ; Str
.text:7423C968                  call    ds:__imp__wcslen
```

CVE-2008-4520 (MS08-067)

NTT Confidential

- Distribution of patch



Patch distribution date differs three month a part

Copyright©2016 NTT corp. All Rights Reserved.

Outline of Today’s Talk :
Similarity Calculation Method for Binary Executables

NTT Confidential

Copyright©2016 NTT corp. All Rights Reserved.

Innovative R&D by NTT
NTT Confidential

"Similarity of Binaries through re-Optimization" (GitZ) [32]

Yaniv David, Nimrod Partush, Eran Yahav, PLDI, June, 2017

- **Background/Motivation**
  - Develop Cross-{compiler, optimization, architecture} binary code similarity method
- **Key Idea**
  - Strands
  - Out-of-context re-optimization

NTT
Copyright©2016 NTT corp. All Rights Reserved.

State-of-the-art Research

Similarity of Binaries through re-Optimization (GitZ)

NTT Confidential

- **System Overview**
  - **Step1: Split the basic block assembly to “strand”**
  - Step2: Lift assembly to IR
  - Step3: Canonical
  - Step4: Normalization

Basic block(top) and extracted strands (bottom)

```
1  mov   rbx, 0x147
2  lea   r15, [rax+1]
3  add   rbx, r15
4  sub   r13, r15
5  cmp   r13, -2
```

Basic Block

```
1  mov   rbx, 0x147
2  lea   r15, [rax+1]
3  add   rbx, r15
```

Strand 1

```
2  lea   r15, [rax+1]
4  sub   r13, r15
5  cmp   r13, -2
```

Strand 2

...hts Reserved.

State-of-the-art Research

Similarity of Binaries through re-Optimization (GitZ)

NTT Confidential

System Overview

- Step1: Split the basic block assembly to “strand”
- **Step2: Lift assembly to IR**
- **Step3: Canonical**
- **Step4: Normalization**

(i) Assembly → (ii) Lifted → (iii) Canonical → (iv) Canonical & Normalized

(A) ARM-64 GCC 4.8 -O0

```
mov    x0,  x20
add    x0,  x0, 1
sub    x21, x21, x0
cmn    x21, 2
```

```
t3  = load i64, i64* x20
t4  = sext i8 0 to i64
t5  = shl  i64  t3, t4
t6  = or   i64  0,  t5
store i64  t6,  i64* x0
t17 = load i64, i64* x0
t18 = add  i64  t17, 1
store i64  t18, i64* x0
t38 = load i64, i64* x21
...
```

```
t3  = load i64, i64* x20
t18 = add  i64  t3,  1
store i64  t18, i64* x0
t38 = load i64, i64* x21
t42 = sub  i64  t38, t18
store i64  t42, i64* x21
t57 = add  i64  t42, 2
ret   i64  t57
```

(B) X86-64 icc 15.0.3 -O3

```
lea    r15, [rax+1]
sub    r13, r15
cmp    r13, -2
```

```
t18 = load i64, i64* rax
t19 = add  i64  t18, 1
store i64  t19, i64* r15
t23 = load i64, i64* r13
t24 = load i64, i64* r15
t25 = sub  i64  t23, t24
store i64  t25, i64* r13
t37 = load i64, i64* r13
t38 = sub  i64  t37, -2
...
```

```
t18 = load i64, i64* rax
t19 = add  i64  t18, 1
store i64  t19, i64* r15
t23 = load i64, i64* r13
t25 = sub  i64  t23, t19
store i64  t25, i64* r13
t38 = add  i64  t25, 2
ret   i64  t38
```

(iv) Canonical & Normalized:

```
t0  = load i64, i64* r0
t1 = add  i64  t0,  1
store i64  t1, i64* r1
t2 = load i64, i64* r2
t3 = sub  i64  t2, t1
store i64  t3, i64* r2
t4 = add  i64  t3, 2
ret   i64  t4
```


Copyright©2016 NTT corp. All Rights Reserved.

State-of-the-art Research

Similarity of Binaries through re-Optimization (GitZ)

NTT Confidential

Scenario	#	Queries	Targets	CROC	FPr
Cross-*	1	*	*	.977	.03
Cross-Arch,Opt	2	C_{ARM} -O*	C_{x64} -O*	.963	.01
	3	C_{x64} -O*	C_{ARM} -O*		
Cross-Opt, Version	4	gcc$_{x64}$ 4.{6,8,9} -O*	gcc$_{x64}$ 4.{6,8,9} -O*	.999	.001
	5	icc$_{x64}$ {14,15} -O*	icc$_{x64}$ {14,15} -O*	.999	.001
	6	CLang$_{x64}$ 3.{4,5} -O*	CLang$_{x64}$ 3.{4,5} -O*	1	0
	7	gcc$_{ARM}$ 4.8 -O*	gcc$_{ARM}$ 4.8 -O*	1	0
	8	CLang$_{ARM}$ 4.0 -O*	CLang$_{ARM}$ 4.0 -O*	1	0
Cross-Comp x86_64	9	C_{x64} -Os	C_{x64} -Os	.992	.001
	10	C_{x64} -O0	C_{x64} -O0	.992	.001
	11	C_{x64} -O1	C_{x64} -O1	.986	.002
	12	C_{x64} -O2	C_{x64} -O2	.992	.001
	13	C_{x64} -O3	C_{x64} -O3	.992	.001
Cross-Comp AArch64	14	C_{ARM} -Os	C_{ARM} -Os	.988	.002
	15	C_{ARM} -O0	C_{ARM} -O0	.995	.001
	16	C_{ARM} -O1	C_{ARM} -O1	.999	.001
	17	C_{ARM} -O2	C_{ARM} -O2	.995	.001
	18	C_{ARM} -O3	C_{ARM} -O3	.998	.001
Cross-Arch	19	C_{x64} -Os	C_{ARM} -Os	.969	.006
	20	C_{ARM} -Os	C_{x64} -Os		
	21	C_{x64} -O0	C_{ARM} -O0	.977	.004
	22	C_{ARM} -O0	C_{x64} -O0		
	23	C_{x64} -O1	C_{ARM} -O1	.960	.006
	24	C_{ARM} -O1	C_{x64} -O1		
	25	C_{x64} -O2	C_{ARM} -O2	.965	.005
	26	C_{ARM} -O2	C_{x64} -O2		
	27	C_{x64} -O3	C_{ARM} -O3	.975	.004
	28	C_{ARM} -O3	C_{x64} -O3		

Table 2. Accuracy and FP rate for different derivatives of the All v. All experiment.

ıt©2016 NTT corp. All Rights Reserved.

Questions?

Copyright©2016 NTT corp. All Rights Reserved.

Reference

Innovative R&D by NTT

NTT Confidential

[1]A. Walenstein and A. Lakhotia, "The Software Similarity Problem in Malware Analysis." In Proceedings Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 10 pp., Dagstuhl, Germany, July 2006.

[2] Z. Wang, K. Pierce and S. McFarling, "BMAT -- A Binary Matching Tool for Stale Profile Propagation", Journal of Instruction-Level Parallelism, 2, 2000.

[3] H. Flake, "Structural Comparison of Executable Objects", DIMVA 2004,161-173.

[4] T. Dullien and R. Rolles "Graph-based comparison of Executable Objects" , SSTIC'05, Rennes, France, June 2005 .

[5] G. Myles and C. Collberg, "k-gram Based Software Birthmarks", In Proceedings of the 2005. ACM Symposium on Applied Computing, ACM, 314-318, 2005.

[6] S. Choi, H. Park, H. Lim, and T. Han, "A Static Birthmark of Binary Executables Based on API Call Structure", ASIAN, 2-16, 2007.

[7]D. Gao, M. K. Reiter, and D. Song, "BinHunt: Automatically Finding Semantic Differences in Binary Programs", ICICS '08 Proceedings of the 10th International Conference on Information and Communications Security , 238 – 255, Birmingham, UK, October, 2008.

[8] A. Sæbjørnsen, J. Willcock, T. Panas, D. Quinlan and Z. Su "Detecting Code Clones in Binary Executables", In Proceedings of the Eighteenth International. Symposium on Software Testing and Analysis, ISSTA. '09, pages 117–128, New York, NY, USA, 2009.

NTT

Copyright©2016 NTT corp. All Rights Reserved.

Reference

NTT Confidential

[9] E. R. Jacobson, N. Rosenblum and B. P. Miller, "Labeling Library Functions in Stripped Binaries", In Proceedings of the 10th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools, PASTE 2011, pp.1-8, 2011.

[10] A. Hemel, K. T. Kalleberg and R. Vermaas, " Finding Software License Violations Through Binary Code Clone Detection", MSR '11, Waikiki, Honolulu, HI, USA, May, 2011.

[11] S. Cesare and Y. Xiang. "Malware Variant Detection Using Similarity Search over Sets of Control Flow Graphs", In Proceedings of the International Conference on Trust, Security and Privacy in Computing and Communications(TrustCom), 2011.

[12] J. Jang, D. Brumley and S. Venkataraman,"BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis", CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

[13] J. Ming, M. Pang, D. Gao, "iBinHunt: Binary Hunting with Inter-Procedural Control Flow", In Proceedings of the 15th Annual International Conference on Information Security and Cryptology, Seoul, Korea, November, 2012

[14] B. H. Ng and A Prakash, "Exposé : Discovering Potential Binary Code Re-Use", COMPSAC, IEEE 37th Annual, pp. 492–501, 2013

[15] W. M. Khoo, A. Mycroft and R. Anderson, "Rendezvous: A Search Engine for Binary Code", In: Proceedings of the 10th Working Conference on Mining Software Repositories, *MSR* 2013, pp.329-338, 2013

Copyright©2016 NTT corp. All Rights Reserved.

Reference

Innovative R&D by NTT
NTT Confidential

[16] M. Bourquin, A King and E Robbins, "BinSlayer: accurate comparison of binary executables", PPREW '13, Jan, 2013,Rome, Italy,

[17] J. Jang, M. Woo and D. Brumley, "Towards Automatic Software Lineage Inference", In the Proceedings of the 22nd USENIX Security Symposium, August, 2013, Washington, D.C., USA

[18] M. Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren and N. Zheng, "A similarity metric method of obfuscated malware using function-call graph" , Journal in Computer Virology archive Volume 9 Issue 1, February 2013, Pages 35-47.

[19] A. Lakhotia, M. D. Preda and R. Giacobazzi "Fast Location of Similar Code Fragments Using Semantic 'Juice'", PPREW '13 Jan 26, 2013, Rome, Italy.

[20] Y. David, E. Yahav, "Tracelet-Based Code Search in Executables", PLDI '14, June, 2014, Edinburgh, United Kingdom.

[21] L. Luo, J Ming, D. Wu, P. Liu, S. Zhu "Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection", SIGSOFT FSE, 2014, 389-400.

[22] J. Pewnyy, F. Schustery, C. Rossowz, L. Bernhardy and T. Holz, "Leveraging Semantic Signatures for Bug Search in Binary Programs", ACSAC December 8-12, 2014, New Orleans, LA, USA

[23] M. R. Farhadi, B. C. M. Fung, P. Charland and M. Debbabi "BinClone: Detecting Code Clones in Malware", SERE 2014, 78-87.

[24] J. Pewny, B. Garmany, R. Gawlik, C. Rossow, T. Holz,"Cross-Architecture Bug Search in Binary Executables", In Proceedings of the 2015 IEEE Symposium on Security and Privacy, Pages 709-724

NTT

Copyright©2016 NTT corp. All Rights Reserved.

Reference

NTT Confidential

[25] S. Alrabaee, P. Shirani, L. Wang and M. Debbabi, "SIGMA: A Semantic Integrated Graph Matching Approach for identifying reused functions in binary code", Digital Investigation 12, S61-S71, 2015

[26] Y. David, N. Partush and E. Yahav, "Statistical Similarity of Binaries", In Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, Pages 266-280, 2016

[27] S. H. H. Ding, B. C. M. Fung and P. Charland, "Kam1n0: MapReduce-based Assembly Clone Search for Reverse Engineering", *KDD '16 August , 2016, San Francisco, CA, USA*

[28] M. Chandramohan, Y. Xue, Z. Xu, Y. Liu, C. Y. Cho and T. H. B. Kuan, "BinGo: Cross-Architecture Cross-OS Binary Search", FSE'16, November , 2016, Seattle, WA, USA

[29] S. Eschweiler, K. Yakdan, E. Gerhards-Padilla, "discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code", NDSS '16, February 2016, San Diego, CA, USA

[30]

[31] Q. Fengy, R. Zhouy, C. Xuy, Y. Chengy, B. Testay, and H. Yin, "Scalable Graph-based Bug Search for Firmware Images", CCS'16, October, 2016, Vienna, Austria

[32] Y. David, N. Partush and E. Yahav, "Similarity of Binaries through re-Optimization", In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation , Pages 79-94, 2017

Copyright©2016 NTT corp. All Rights Reserved.

Reference

[33] H. Huang, A. M. Youssef and M. Debbabi, “BinSequence: Fast, Accurate and Scalable Binary Code Reuse Detection”, ASIA CCS ’17, April 02-06, 2017, Abu Dhabi, United Arab Emirates

[34] Q. Feng, M. Wang , M. Zhang, R. Zhou, A. Henderson and Heng Yin, “Extracting Conditional Formulas for Cross-Platform Bug Search”, ASIA CCS ’17, April 02-06, 2017, Abu Dhabi, United Arab Emirates

Copyright©2016 NTT corp. All Rights Reserved.