Scalable Program Clone Search through Spectral Analysis

Tristan Benoit
benoit.tristan.info@gmail.com
Université de Lorraine, CNRS, LORIA
Nancy, France

Jean-Yves Marion
jean-yves.marion@loria.fr
Université de Lorraine, CNRS, LORIA
Nancy, France

Sébastien Bardin
sebastien.bardin@cea.fr
CEA LIST, Université Paris-Saclay
Saclay, France

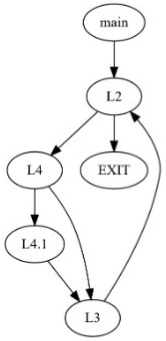
ESEC/FSE 2023
Sun 3 - Sat 9 December 2023
San Francisco, California, United States

Program Clones

```c
#include <cstdio>
int main()
{
    int n, answer = 0;
    scanf("%d", &n);
    while(n)
    {
        if(n % 2 == 1)
        {
            answer++;
        }
        n /= 2;
    }
    return answer;
}
```

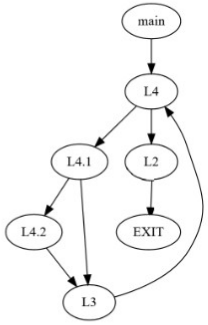
Source code

```
.LC0:
        .string "%d"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 0
        lea     rax, [rbp-8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    scanf
        jmp     .L2
.L4:
        mov     eax, DWORD PTR [rbp-8]
        cdq
        shr     edx, 31
        add     eax, edx
        and     eax, 1
        sub     eax, edx
        cmp     eax, 1
        jne     .L3
        add     DWORD PTR [rbp-4], 1
.L3:
        mov     eax, DWORD PTR [rbp-8]
        mov     edx, eax
        shr     edx, 31
        add     eax, edx
        sar     eax
        mov     DWORD PTR [rbp-8], eax
.L2:
        mov     eax, DWORD PTR [rbp-8]
        test    eax, eax
        jne     .L4
        mov     eax, DWORD PTR [rbp-4]
        leave
        ret
```



GCC 4.8.5 -O0

```
.LC0:
        .string "%d"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 0
        lea     rax, [rbp-8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
.L4:
        mov     eax, DWORD PTR [rbp-8]
        test    eax, eax
        je      .L2
        mov     eax, DWORD PTR [rbp-8]
        cdq
        shr     edx, 31
        add     eax, edx
        and     eax, 1
        sub     eax, edx
        cmp     eax, 1
        jne     .L3
        add     DWORD PTR [rbp-4], 1
.L3:
        mov     eax, DWORD PTR [rbp-8]
        mov     edx, eax
        shr     edx, 31
        add     eax, edx
        sar     eax
        mov     DWORD PTR [rbp-8], eax
        jmp     .L4
.L2:
        mov     eax, DWORD PTR [rbp-4]
        leave
        ret
```



GCC 9.3.0 -O0

- same source code but different toolchains / optimizations / architecture
- same source code but obfuscation
- (slightly) different version of the source code

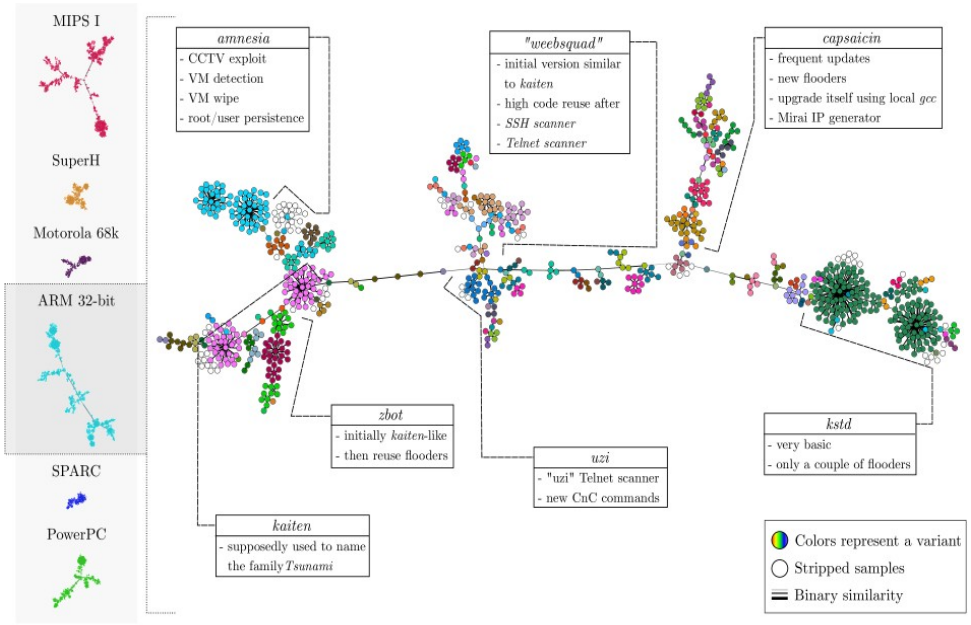
Applications

Studying Malware
- Genealogy
- Clustering
- Detection

Detecting
- Software Theft
- Plagiarism

Identifying libraries
- Software Engineering
- Cybersecurity (1day)

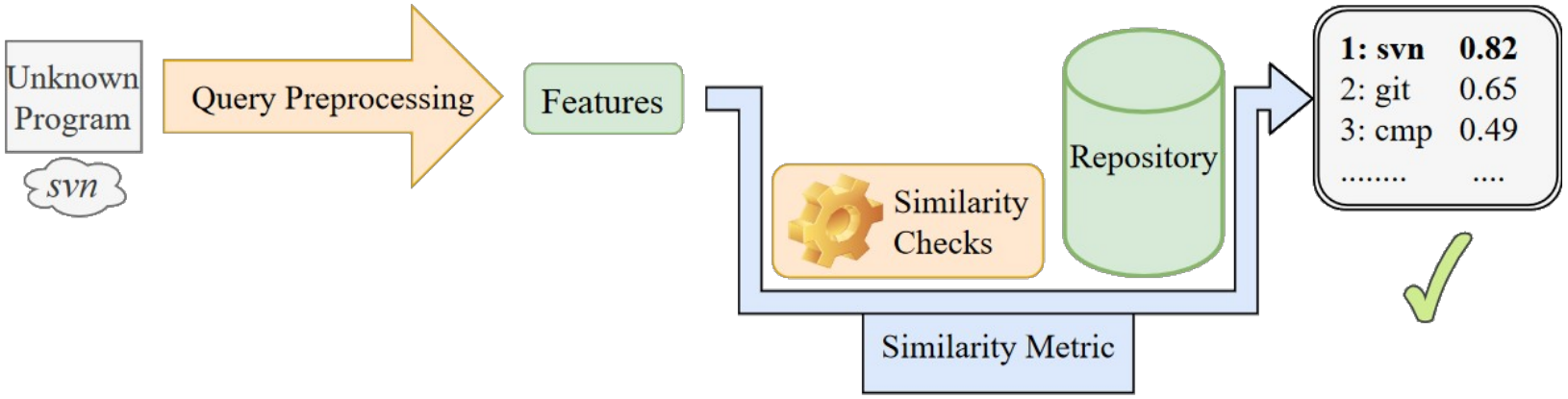
Malware – supply chain – firmware – closed programs



Tsunami for ARM 32 genealogy.

« The Tangled Genealogy of IoT Malware »
Cozzi, Vervier, Dell'Amico, Shen, Balzarotti.
ACSAC 2020

Program Clone Search



1) Preprocessing
We extract features from **target program P.**

2) Similarity Checks
For each program Q in **repository** R, compute the **similiarity index** between P and Q.

3) Decision
We **pick the program** Qm with the **highest similarity index**.
The clone search is a success if Qm is indeed a clone of P.

A Blooming Field

Approach	Year	Venue	Input
TRACY [1]	2014	PLDI	F
BINCLONE [65]	2014	SERE	I*
RMKNHLLP2014 [66]	2014	DIMVA	F*
CXZ2014 [21]	2014	TDSC	P
BLEX [67]	2014	USENIX Sec	F
COP [33], [68]	2014	ESEC/FSE	P
TEDEM [2]	2014	ACSAC	B*
SIGMA [69]	2015	DFRWS	F
MXW2015 [24]	2015	IFIP SEC	P
MULTI-MH [3]	2015	S&P	B*
QSM2015 [70]	2015	SANER	F
DISCOVRE [4]	2016	NDSS	F
MOCKINGBIRD [29]	2016	SANER	F
ESH [5]	2016	PLDI	F
TPM [71]	2016	TrustCom	P
BINDNN [72]	2016	SecureComm	F
GENIUS [6]	2016	CCS	F
BINGO [7]	2016	FSE	F
KLKI2016 [18]	2016	JSCOMPUT	P
KAM1N0 [73]	2016	SIGKDD	B*

Approach	Year	Venue	Input
BINSEQUENCE [8]	2017	ASIACCS	F
XMATCH [9]	2017	ASIACCS	F
CACOMPARE [74]	2017	ICPC	F
SPAIN [30]	2017	ICSE	P
BINSIGN [75]	2017	IFIP SEC	F
GITZ [10]	2017	PLDI	F
BINSHAPE [76]	2017	DIMVA	F
BINSIM [77]	2017	USENIX Sec	T
KS2017 [31]	2017	ASE	T
IMF-SIM [78]	2017	ASE	F
GEMINI [12]	2017	CCS	F
FOSSIL [79]	2018	TOPS	F
FIRMUP [13]	2018	ASPLOS	F
BINARM [14]	2018	DIMVA	F
αDIFF [15]	2018	ASE	P
VULSEEKER [11]	2018	ASE	F
RLZ2019 [80]	2019	BAR	B
INNEREYE [81]	2019	NDSS	B*
ASM2VEC [82]	2019	S&P	F
SAFE [83]	2019	DIMVA	F

I:Instruction B:Basic Block F:Function P:Program

« A survey of binary code similarity » Irfan Ul Haq and Juan Caballero. ACM Computing Surveys 2021

A Blooming Field

- Precision
- Robustness (cross-compilers, cross-archi., light obfuscation)
- Speed/Scale (size of code, size of repository)

CoP [33], [68]	2014	ESEC/FSE	P
TEDEM [2]	2014	ACSAC	B*
SIGMA [69]	2015	DFRWS	F
MXW2015 [24]	2015	IFIP SEC	P
MULTI-MH [3]	2015	S&P	B*
QSM2015 [70]	2015	SANER	F
DISCOVRE [4]	2016	NDSS	F
MOCKINGBIRD [29]	2016	SANER	F
ESH [5]	2016	PLDI	F
TPM [71]	2016	TrustCom	P
BINDNN [72]	2016	SecureComm	F
GENIUS [6]	2016	CCS	F
BINGO [7]	2016	FSE	F
KLKI2016 [18]	2016	JSCOMPUT	P
KAM1N0 [73]	2016	SIGKDD	B*

GITZ [10]	2017	PLDI	F
BINSHAPE [76]	2017	DIMVA	F
BINSIM [77]	2017	USENIX Sec	T
KS2017 [31]	2017	ASE	T
IMF-SIM [78]	2017	ASE	F
GEMINI [12]	2017	CCS	F
FOSSIL [79]	2018	TOPS	F
FIRMUP [13]	2018	ASPLOS	F
BINARM [14]	2018	DIMVA	F
αDIFF [15]	2018	ASE	P
VULSEEKER [11]	2018	ASE	F
RLZ2019 [80]	2019	BAR	B
INNEREYE [81]	2019	NDSS	B*
ASM2VEC [82]	2019	S&P	F
SAFE [83]	2019	DIMVA	F

I:Instruction B:Basic Block F:Function P:Program

« A survey of binary code similarity » Irfan Ul Haq and Juan Caballero. ACM Computing Surveys 2021

A Blooming Field

- Precision
- Robustness (cross-compilers, cross-archi., light obfuscation)
- Speed/Scale (size of code, size of repository)

CoP [33], [68]	2014	ESEC/FSE	P	GitZ [10]	2017	PLDI	F
Tedem [2]	2014	ACSAC	B*	BinShape [76]	2017	DIMVA	F
Sigma [69]	2015	DFRWS	F	BinSim [77]	2017	USENIX Sec	T
MXW2015 [24]	2015	IFIP SEC	P	KS2017 [31]	2017	ASE	T
							F
							F
							F
							F
							F
							P
							F
							B
BINGO [7]	2016	FSE	F	INNEREYE [81]	2019	NDSS	B*
KLKI2016 [18]	2016	JSCOMPUT	P	Asm2Vec [82]	2019	S&P	F
Kam1n0 [73]	2016	SIGKDD	B*	SAFE [83]	2019	DIMVA	F

- **Heavy focus on function level similarities**
- Robustness of ML embeddings uncertain
- Graph Edit Distance and Matchings abandonned

I:Instruction B:Basic Block F:Function P:Program

« A survey of binary code similarity » Irfan Ul Haq and Juan Caballero. ACM Computing Surveys 2021

A case for program-level clone search

Claim:
Clone search is essentially at the level of programs (not functions)

- Malware studies, software theft, library identification
- A single function is not the right focus

Function-level methods do not scale to program-level

- Gemini: 17h for 20 clone searches on 1420 libraries
- SAFE: 160h for the same 20 clone searches

GED and Matching methods do not scale with code size

- SMIT: 43h to compute a single similarity index between Geany and the cp command
- DeepBinDiff: 10min for basic block matching on small binaries from core utils

A case for program-level clone search

Goal: a program-level clone search approach

- Scalable
- Reasonably precise and robust (cross archi / compilers, etc.)
- Do not rely on symbols and strings

Contributions

- PSS (Program Spectral Similarity), a similarity metric between **programs** that is **fast** and **accurate** while being very **robust** even against (light) **obfuscations**
  - **key insight: spectral graph analysis**
  - **Dedicated adaptations and optimization (PSSO)**

No learning

- An evaluation covering **14** methods and over **200,000 programs** from Linux, Windows and IoT malware

- **Insights** about how **classes** of methods perform

Try it!

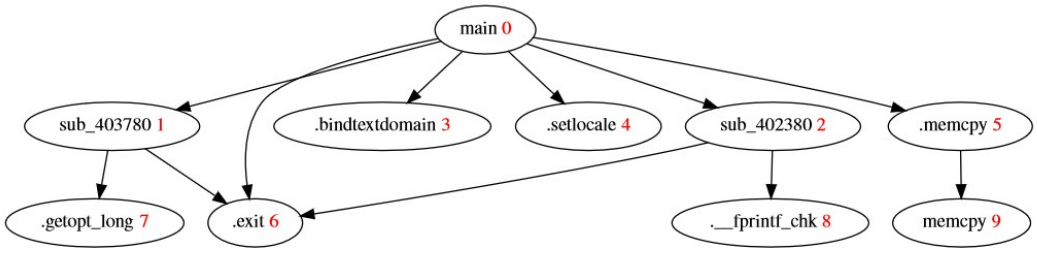
Available at:
https://doi.org/10.5281/zenodo.8289599

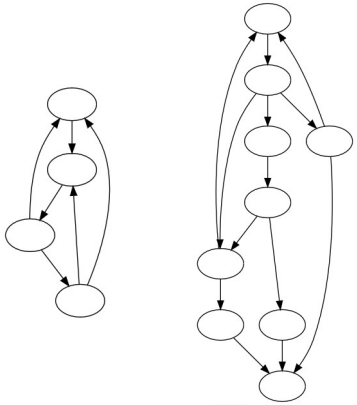
Loria

PSS

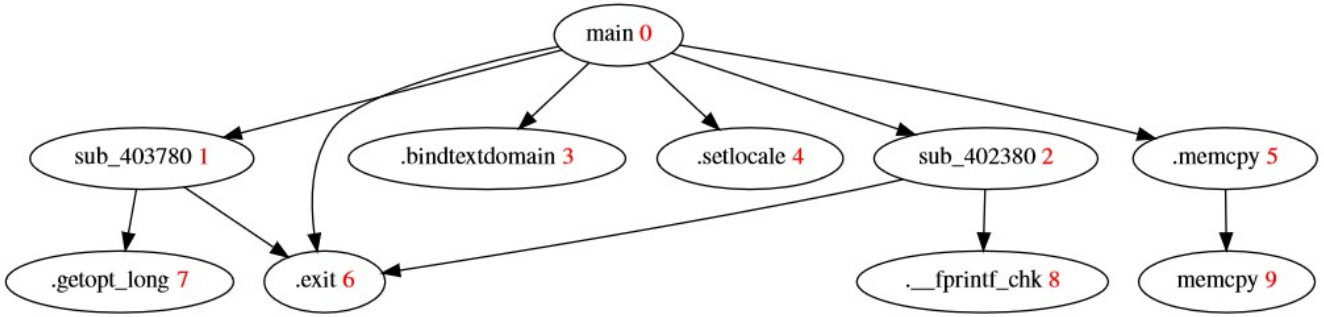
Ideas

- Performing quick comparisons with the **spectral distance** between graph representations of programs.
- Reducing the one-time computation of the spectrum by using only the spectrum of the call graph.
- Incorporating information about each function CFG.





PSS



Definition *An eigenvalue λ and an eigenvector $\vec{u}$ is a solution to the equation : $(L - \lambda I)\,\vec{u} = \vec{0}$. The spectrum is the set Λ of eigenvalues $\{\lambda_1(G), \lambda_2(G), \ldots, \lambda_{|G|}(G)\}$ where $\lambda_1(G) \geq \lambda_2(G) \geq \ldots \geq \lambda_{|G|}(G)$ and where $|G|$ is the number of vertices of G.*

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad L = \begin{bmatrix} 6 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & 3 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ -1 & 0 & 3 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \Lambda \approx \begin{bmatrix} 7,10 \\ 4,52 \\ 3,41 \\ 2,54 \\ 1,72 \\ 1 \\ 0,72 \\ 0,58 \\ 0,40 \\ 0 \end{bmatrix}$$

PSS

Definition *The spectral distance [54] between two graphs G_1, G_2 of same order n is* $\sum_1^N |\lambda_i(G_1) - \lambda_i(G_2)|$.

The generalized spectral distance is defined by : $\sum_{i=1}^{min(|G1|,|G2|)} |\lambda_i(G_1) - \lambda_i(G_2)|$.

Theorem *Let G' be a copy of a graph G except for a removed vertex of degree r. For all i such that $1 \leq i \leq n-1$, $\lambda_i(G) \geq \lambda_i(G') \geq \lambda_{i+r}(G)$.*

Corollary *The generalized spectral distance between G and G' is equal to à $2r$.*

The Lanczos algorithm calculate the full spectrum in $O(dn^2)$ where n is the number of vertices, and d is the average degree of the graph.

PSS

1) From an undirected version of the **call graph**, we compute its spectrum ***v*** and **normalize it**.

2) For each function, we compute its **number of edges.** We gather it as a vector ***w***, sort it in descending order, and **normalize it**.

Similarity index between programs **in O(n)**:

$$simCG(P_0, P_1) := \sqrt{2} - \sqrt{\sum_{i=0}^{min(|\vec{v_0}|,|\vec{v_1}|)} (v_{0,i} - v_{1,i})^2}$$

$$simCFG(P_0, P_1) := \sqrt{2} - \sqrt{\sum_{i=0}^{min(|\vec{w_0}|,|\vec{w_1}|)} (w_{0,i} - w_{1,i})^2}$$

$$PSS(P_0, P_1) := \frac{simCG(P_0, P_1) + simCFG(P_0, P_1)}{2\sqrt{2}}$$

The PSSO optimization

We propose PSSO, an optimized version of PSS.
Instead of computing the whole spectrum, we compute only the **100 greatest eigenvalues** of the call graph.

> Lanczos algorithm can compute the k greatest eigenvalues in O(k.d.n) where n is the number of vertices and d the average degree of the graph.

Theoretical complexities

Method	Class	Similarity Check†	Preprocesing‡
SMIT [45]	GED	$O(n^4)$	$O(dn)$
CGC [104]	Matching	$O(n^4)$	$O(dn)$
MutantX-S [44]	**N-gram**	$O(1)$	$O(i)$
Asm2Vec [24]	Function ML	$O(n^2)$	$O(n)$
Gemini [105]	Function ML	$O(n^2)$	$O(n)$
SAFE [71]	Function ML	$O(n^2)$	$O(n)$
αDiff [67]	Function ML	$O(n^2)$	$O(n)$
LibDX [96]	Literal	$O(s)$	$O(s)$
LibDB [97]	Literal and Function ML	$O(n^2+s)$	$O(s+n)$
DeepBinDiff [26]	Matching and ML	$O(n^3m^3)$	None
PSS	**Spectral**	$O(n)$	$O(dn^2)$
PSSO	**Spectral**	$O(n)$	$O(dn)$

n : # functions, i : # instructions , s : # string literals

d : average # call per functions, m : # basic blocks per function

† Between two programs

‡ Once for the whole clone search

Preliminary Study

Total time for all clone searches on the Basic dataset

$\mathbf{B}_{size}$	$\leq$ 1h30m
$\mathbf{D}_{size}$	$\leq$ 1h30m
Shape	$\leq$ 1h30m
ASCG	$\leq$ 1h30m
MutantX-S	$\leq$ 1h30m
PSS	$\leq$ 1h30m
PSSO	$\leq$ 1h30m
LibDX	$\leq$ 1h30m
StringSet	$\leq$ 1h30m
FunctionSet	$\leq$ 1h30m

ASCFG	42h
GED-0	81h
GED-L	46h
SMIT	3634h
CGC	171h
Asm2vec †	141h
Gemini †	102h
SAFE †	655h
αDiff †	642h
LibDB †	16h

Learning time not included

- **10 Methods selected for comparison**

Focus of the talk:
the 7 methods that do not rely on strings

Experiments

BinKiT

From 51 GNU packages, 235 source codes have been extracted. They have been compiled with 416 different toolchains for a total of 97,760 programs.

It cover:

- 8 architectures (arm, x86, mips, and mipseb, for 32/64 bits)
- 9 compilers (5 versions of GCC and 4 versions of Clang)
- 4 optimization levels from -O0 to -O3
- 4 obfuscations from Obfuscator-LLVM (SUB, BCF, FLA, all 3)

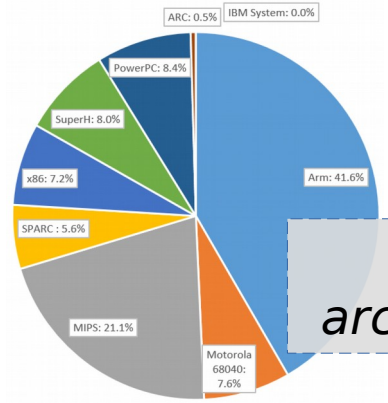
« Revisiting Binary Code Similarity Analysis using Interpretable Feature Engineering and Lessons Learned ».
Dongkwan Kim, Eunsoo Kim, Sang Kil Cha, Sooel Son, Yongdae Kim
IEEE Transactions on Software Engineering. 2022

Experiments

IoT Malware

19,959 IoT malwares from MalwareBazaar (2020-2022)
3 clones family:

- 12,357 mirai
- 5842 gafgyt
- 1760 tsunami



8 major architectures

Windows

84,992 programs for Windows
Clones share:
1) filename 2) target platform

49,443 programs have a clone.

15 target platforms

Experiments

Clone search average runtime (preprocessing)

Dataset # Programs	Basic 1K	BinKiT 96K	IoT 20K	Windows 85K
B_{size}	< 0.01	0.11	0.14	0.63
D_{size}	< 0.01	0.11	0.14	0.63
Shape	0.02	0.05	0.06	0.31
ASCG	1.42 (1.42)	0.37 (0.21)	0.25 (0.06)	17.68 (16.60)
MutantX-S	< 0.01	0.57	0.64	3.00
PSS	1.41 (1.41)	0.49 (0.21)	0.39 (0.05)	19.17 (16.95)
PSS_O	0.27 (0.27)	0.30 (0.04)	0.44 (0.14)	2.29 (0.39)

PSS is fast, except on Windows.
PSSO solves this issue.
MutantX-S is also very fast.

Experiments

Average success

Dataset	Basic	BinKit	IoT	Windows
B_{size}	0.17	0.166	0.819	0.196
D_{size}	0.16	0.062	0.787	0.445
Shape	0.19	0.297	0.818	0.389
ASCG	0.24	0.554	0.759	0.444
MutantX-S	0.38	0.354	0.870	0.472
PSS	0.38	0.619	0.863	0.475
PSS_O	0.38	0.619	0.862	0.466

PSS/PSSO are as accurate as MutantX-S, except on BinKiT on which they are better.

Experiments

Average success on BinKiT

Category	Optimization level						Cross-compiler			Cross-architecture				vs. Obfuscation†			
vs.	O0 O1	O0 O2	O0 O3	O1 O2	O1 O3	O2 O3	gcc-4 gcc-8	clang-4 clang-7	clang gcc	arm mips	arm x86	mips x86	32 64	bcf	fla	sub	all
B_{size}	0.04	0.04	0.07	0.19	0.11	0.21	0.11	0.45	0.07	0.03	0.10	0.04	0.04	0.04	0.01	0.08	0.01
D_{size}	0.03	0.03	0.03	0.06	0.05	0.07	0.07	0.09	0.04	0.02	0.05	0.03	0.04	0.02	0.01	0.05	0.01
Shape	0.19	0.07	0.06	0.17	0.11	0.33	0.38	0.65	0.16	0.04	0.16	0.04	0.19	0.25	0.27	0.48	0.23
ASCG	0.40	0.12	0.10	0.43	0.24	0.68	0.78	0.91	0.46	0.08	0.46	0.06	0.59	0.54	0.64	0.78	0.48
MutantX-S	0.04	0.03	0.03	0.43	0.36	0.64	0.67	0.80	0.14	0.02	0.01	0.01	0.06	0.09	0.03	0.54	0.01
PSS	0.54	0.23	0.17	0.59	0.38	0.70	0.79	0.91	0.51	0.39	0.55	0.39	0.66	0.53	0.57	0.82	0.46
PSS_O	0.53	0.24	0.17	0.60	0.39	0.68	0.78	0.90	0.51	0.44	0.54	0.44	0.66	0.52	0.56	0.82	0.46

PSS and PSSO are very robust, even in cross-compiler/cross architecture scenarios and against obfuscations.

Conclusion

Method	speed	precision	robust.
ASCG [27]	+	-	+
MutantX-S [33]	+	+	--
PSS/PSS_O	+/++	+	+

+ Insigts about strings based methods in the paper

Limitations

1) Hard to deal with changes in function call graphs

2) Evaluated mostly on C/C++ programs

Thank you for your attention!