Loria
Laboratoire lorrain de recherche en informatique et ses applications

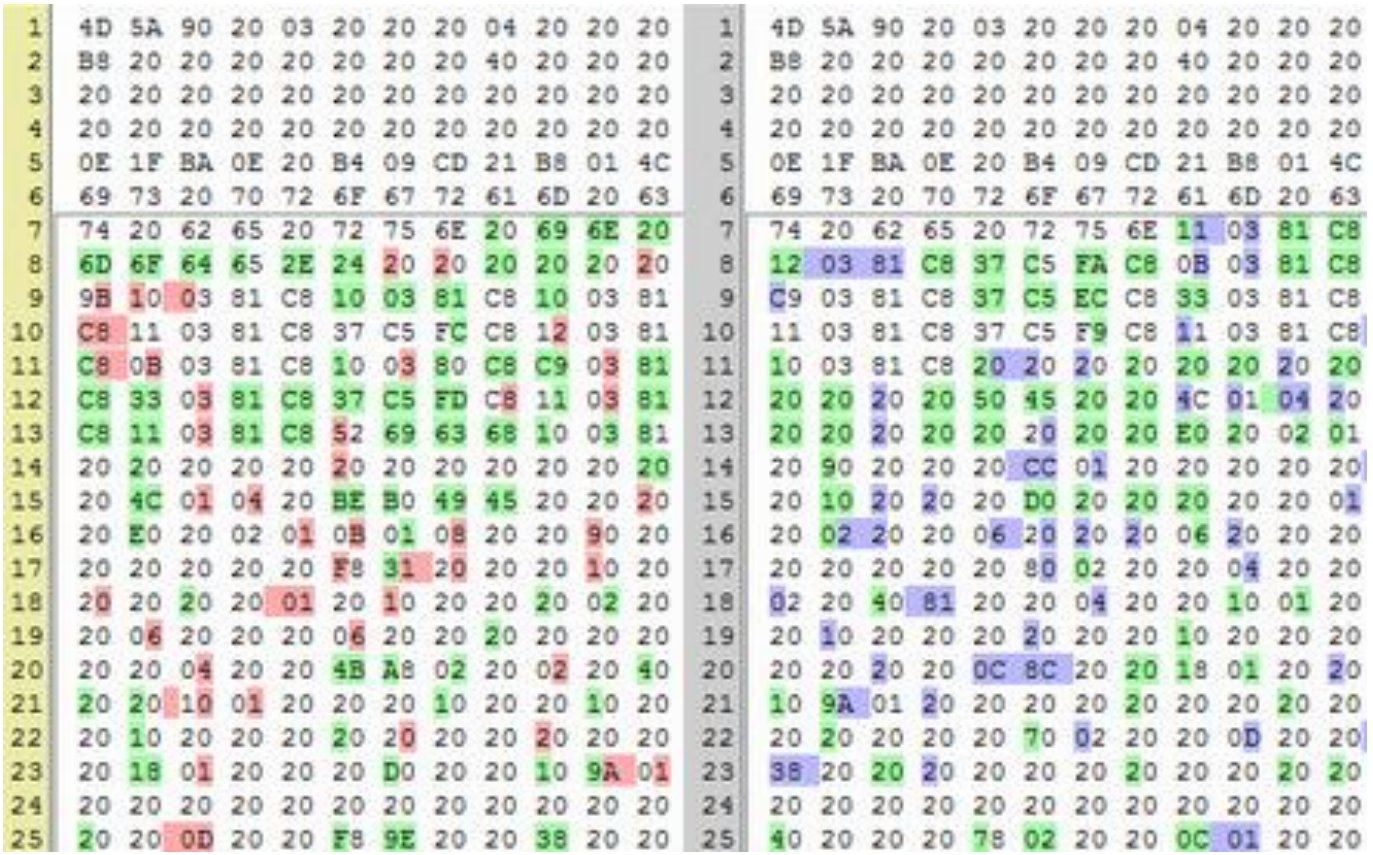
La recherche de clone de programme

Tristan Benoit
Université de Lorraine, CNRS,
LORIA,
F-54000 Nancy, France
tristan.benoit@loria.fr

Jean-Yves Marion
Université de Lorraine, CNRS,
LORIA,
F-54000 Nancy, France
jean-yves.marion@loria.fr

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

Similarité des codes binaires



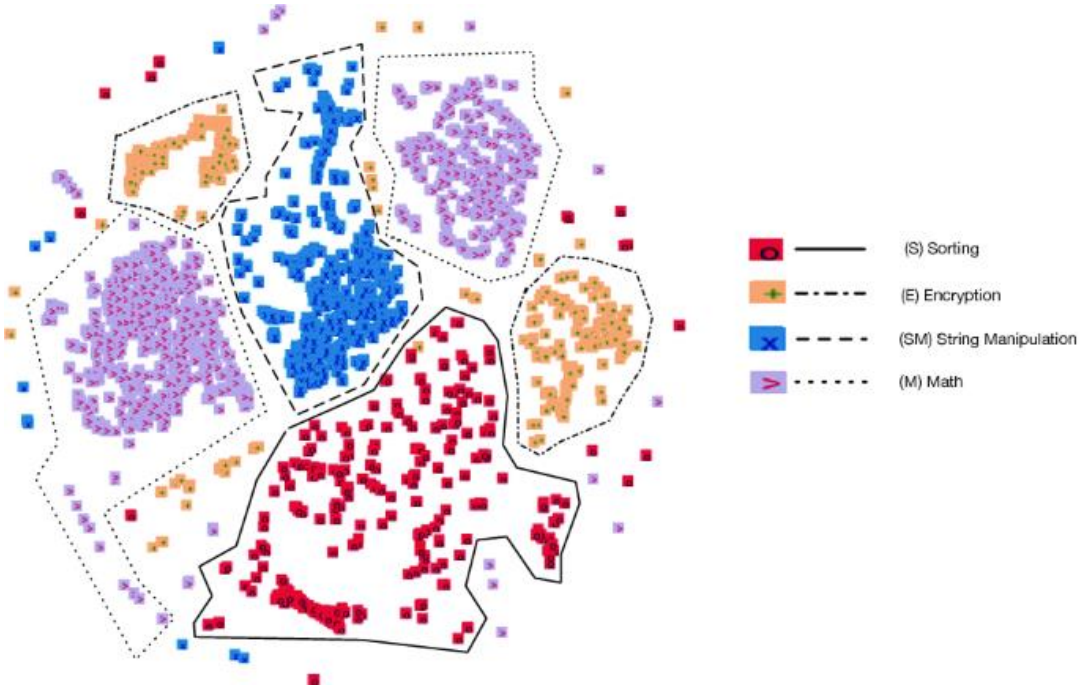
1	4D	5A	90	20	03	20	20	20	04	20	20	20	1	4D	5A	90	20	03	20	20	20	04	20	20	20
2	B8	20	20	20	20	20	20	20	40	20	20	20	2	B8	20	20	20	20	20	20	20	40	20	20	20
3	20	20	20	20	20	20	20	20	20	20	20	20	3	20	20	20	20	20	20	20	20	20	20	20	20
4	20	20	20	20	20	20	20	20	20	20	20	20	4	20	20	20	20	20	20	20	20	20	20	20	20
5	0E	1F	BA	0E	20	B4	09	CD	21	B8	01	4C	5	0E	1F	BA	0E	20	B4	09	CD	21	B8	01	4C
6	69	73	20	70	72	6F	67	72	61	6D	20	63	6	69	73	20	70	72	6F	67	72	61	6D	20	63
7	74	20	62	65	20	72	75	6E	20	69	6E	20	7	74	20	62	65	20	72	75	6E	11	03	81	C8
8	6D	6F	64	65	2E	24	20	20	20	20	20	20	8	12	03	81	C8	37	C5	FA	C8	0B	03	81	C8
9	9B	10	03	81	C8	10	03	81	C8	10	03	81	9	C9	03	81	C8	37	C5	EC	C8	33	03	81	C8
10	C8	11	03	81	C8	37	C5	FC	C8	12	03	81	10	11	03	81	C8	37	C5	F9	C8	11	03	81	C8
11	C8	0B	03	81	C8	10	03	80	C8	C9	03	81	11	10	03	81	C8	20	20	20	20	20	20	20	20
12	C8	33	03	81	C8	37	C5	FD	C8	11	03	81	12	20	20	20	20	50	45	20	20	4C	01	04	20
13	C8	11	03	81	C8	52	69	63	68	10	03	81	13	20	20	20	20	20	20	20	20	E0	20	02	01
14	20	20	20	20	20	20	20	20	20	20	20	20	14	20	90	20	20	20	CC	01	20	20	20	20	20
15	20	4C	01	04	20	BE	B0	49	45	20	20	20	15	20	10	20	20	20	D0	20	20	20	20	20	01
16	20	E0	20	02	01	0B	01	08	20	20	90	20	16	20	02	20	20	06	20	20	20	06	20	20	20
17	20	20	20	20	20	F8	31	20	20	20	10	20	17	20	20	20	20	20	80	02	20	20	04	20	20
18	20	20	20	20	01	20	10	20	20	20	02	20	18	02	20	40	81	20	20	04	20	20	10	01	20
19	20	06	20	20	20	06	20	20	20	20	20	20	19	20	10	20	20	20	20	20	20	10	20	20	20
20	20	20	04	20	20	4B	A8	02	20	02	20	40	20	20	20	20	20	0C	8C	20	20	18	01	20	20
21	20	20	10	01	20	20	20	10	20	20	10	20	21	10	9A	01	20	20	20	20	20	20	20	20	20
22	20	10	20	20	20	20	20	20	20	20	20	20	22	20	20	20	20	20	70	02	20	20	0D	20	20
23	20	18	01	20	20	20	D0	20	20	10	9A	01	23	38	20	20	20	20	20	20	20	20	20	20	20
24	20	20	20	20	20	20	20	20	20	20	20	20	24	20	20	20	20	20	20	20	20	20	20	20	20
25	20	20	0D	20	20	F8	9E	20	20	38	20	20	25	40	20	20	20	78	02	20	20	0C	01	20	20

Compare Suite Pro

Similarité des codes binaires

Les approches de similarité de code binaire identifient des similarités entre des morceaux de code assembleur :

- blocs de base
- fonctions
- programmes entiers



« SAFE: Self-Attentive Function Embeddings for Binary Similarity »
Massarelli, DiLuna, Petroni, Baldoni, Querzoni.
DIMVA 2019.

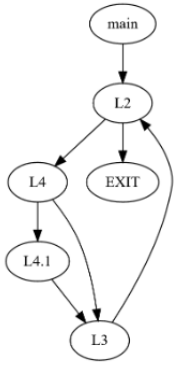
Loria

Similarité des codes binaires

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

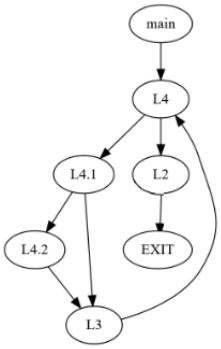
Code source

```
.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

Similarité entre programme

Indice de similarité :
Degré de similarité entre deux programmes.

Nécessaire si :

- Code source original indisponible
- Système hérité
- Programme intégré (Firmware/IoT)
- Logiciel malveillant (Malware)

Applications

2021 Top Malware Strains

« In 2021, the top malware strains included remote access Trojans (RATs), banking Trojans, information stealers, and ransomware. Most of the top malware strains have been in use for more than five years with their respective code bases evolving into multiple variations. The most prolific malware users are cyber criminals, who use malware to deliver ransomware or facilitate theft of personal and financial information. »

Coauthored by:

Loria

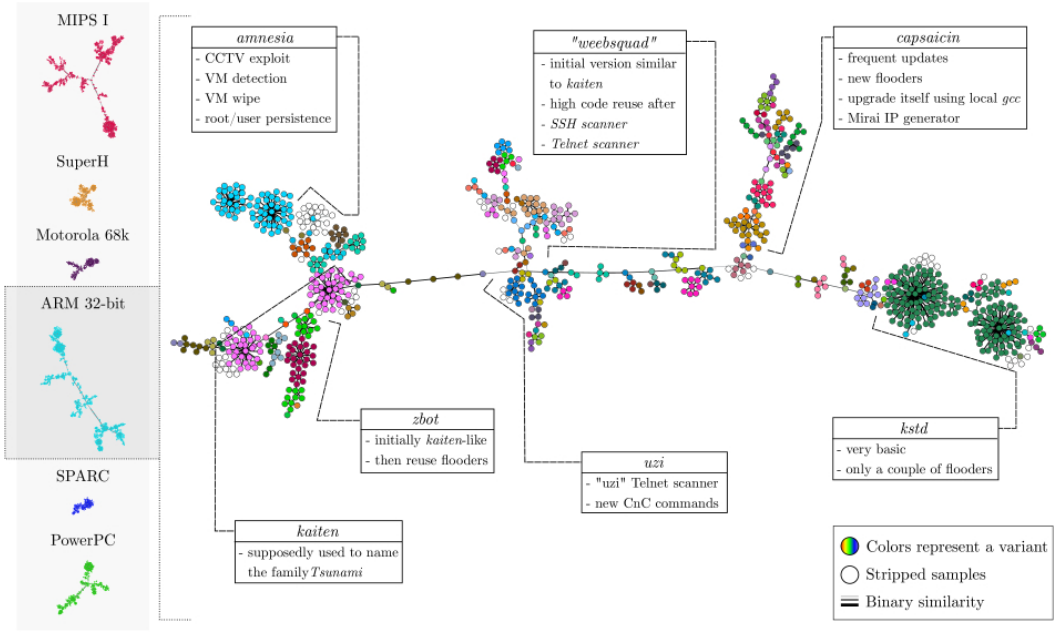
Applications

Étudier les malwares :

- Lignée
- Clustering
- Détection

Détecter :

- Vol de logiciel
- Plagiat



Lignée de *Tsunami* pour *ARM 32*.

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

Applications

Identification de bibliothèques :

- Ingénierie logicielle
- Cybersécurité

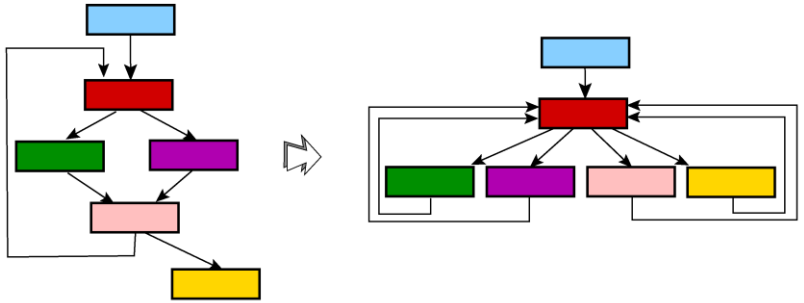
System Tools	Comparison Type	Library Source	Library Database	Detection Targets	Feature	Library Feature Extractor	Target Feature Extractor	Matching Method
BAT	Binary-to -source	Fedora releases 5, 9, 11 and 14	16,085 packages[a]	Firmware (Linux)	String literal	Regular expression	GNU *strings* tool	Direct mapping
OSSPolice	Binary-to -source	C/C++ repositories with more than 100 stars on Github	3,119 repositories with 60,450 versions	Android apps	String literal, exported function	Clang-based parser	Module around pyelftools	Hierarchical matching
LibDX	Binary-to -binary	Fedora release 29	9,537 packages without versions	Various software on multiple platforms	Contents in DATA segment, fuzzy filename, *requires* information	Well-designed accurate extractor		Logic feature block matching

[a] It is not clear how many different packages are in database of BAT, since there are many duplicate items in four Fedora releases.

« LibDX: A Cross-Platform and Accurate System to Detect Third-Party Libraries in Binary Code »
Tang, Luo, Fu, Zhang.
SANER 2020.

Difficultés

- L'efficacité
- La robustesse
- L'absence de symboles
- L'absence d'identificateurs
- Les offuscations



Loria

État de l'art

Approach	Year	Venue	Input
EXEDIFF [25]	1999	WCSSS	P
BMAT [32]	1999	FDO2	P
F2004 [26]	2004	DIMVA	P
DR2005 [27]	2005	SSTIC	P
KKMRV2005 [19]	2005	RAID	P
BMM2006 [20]	2006	DIMVA	P
BINHUNT [28]	2008	ICISC	P
SWPQS2006 [56]	2009	ISSTA	I*
SMIT [16]	2009	CCS	P
IDEA [57]	2010	ESSoS	P
MBC [58]	2012	RACS	P
IBINHUNT [59]	2012	ICISC	P
BEAGLE [22]	2012	ACSAC	P
BINHASH [60]	2012	ICMLA	F
BINJUICE [42]	2013	PPREW	P
BINSLAYER [61]	2013	PPREW	P
RENDEZVOUS [62]	2013	MSR	F
MUTANTX-S [17]	2013	Usenix ATC	P
EXPOSÉ [63]	2013	COMPSAC	P
ILINE [23]	2013	USENIX Sec	P
LKI2013 [64]	2013	RACS	P

I:Instruction B:Bloc F:Fonction P:Programme

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

État de l'art

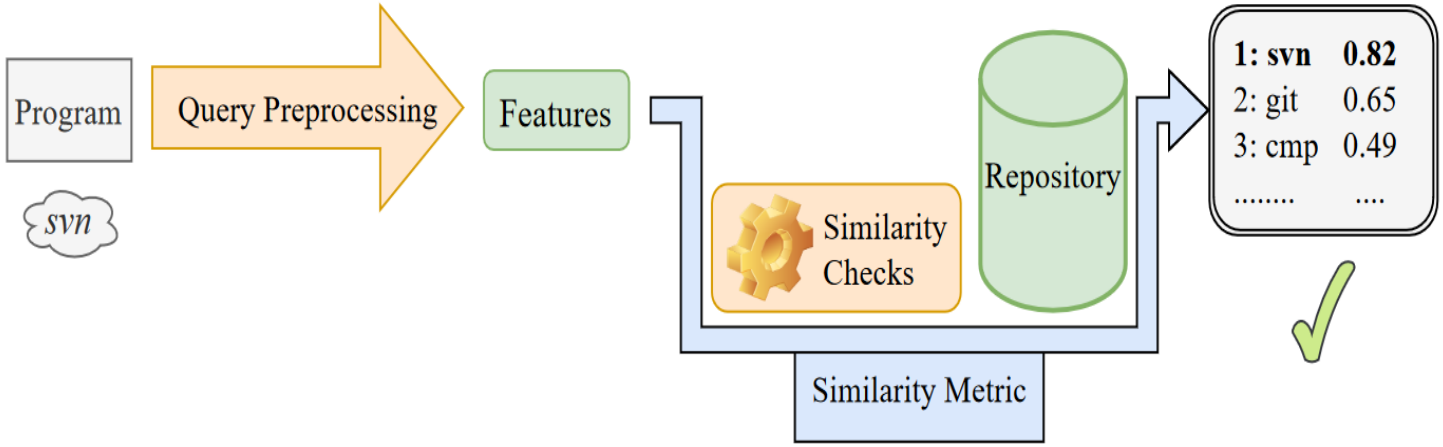
Approach	Year	Venue	Input
TRACY [1]	2014	PLDI	F
BINCLONE [65]	2014	SERE	I*
RMKNHLLP2014 [66]	2014	DIMVA	F*
CXZ2014 [21]	2014	TDSC	P
BLEX [67]	2014	USENIX Sec	F
COP [33], [68]	2014	ESEC/FSE	P
TEDEM [2]	2014	ACSAC	B*
SIGMA [69]	2015	DFRWS	F
MXW2015 [24]	2015	IFIP SEC	P
MULTI-MH [3]	2015	S&P	B*
QSM2015 [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:Bloc F:Fonction P:Programme

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

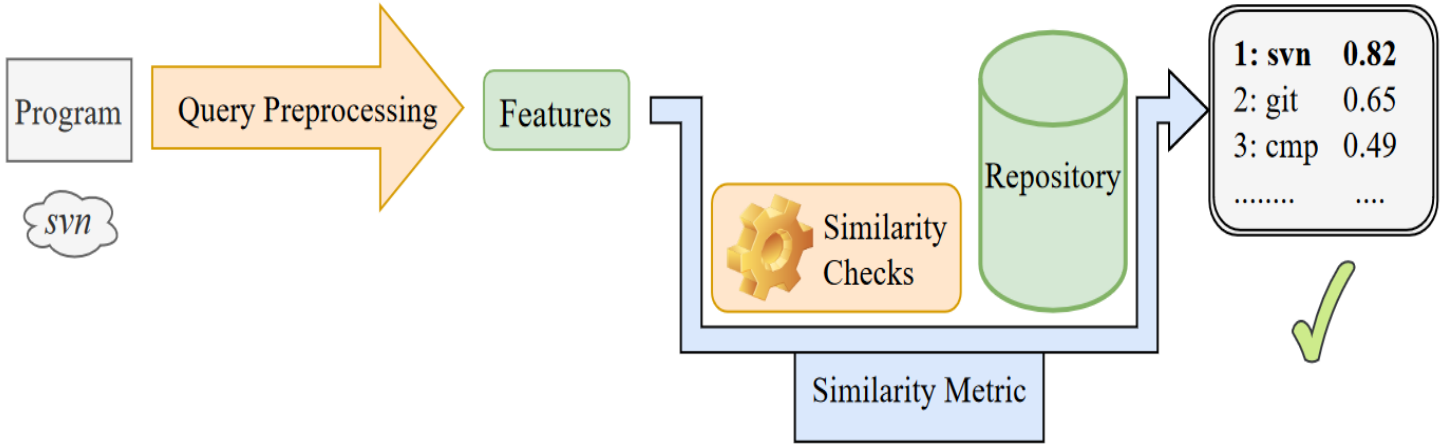
Recherche de clone



Un clone d'un programme est :

- Un programme compilé à partir du même code source mais avec une chaîne de compilation différente.
- Un programme compilé à partir d'une autre version du code source.

Recherche de clone

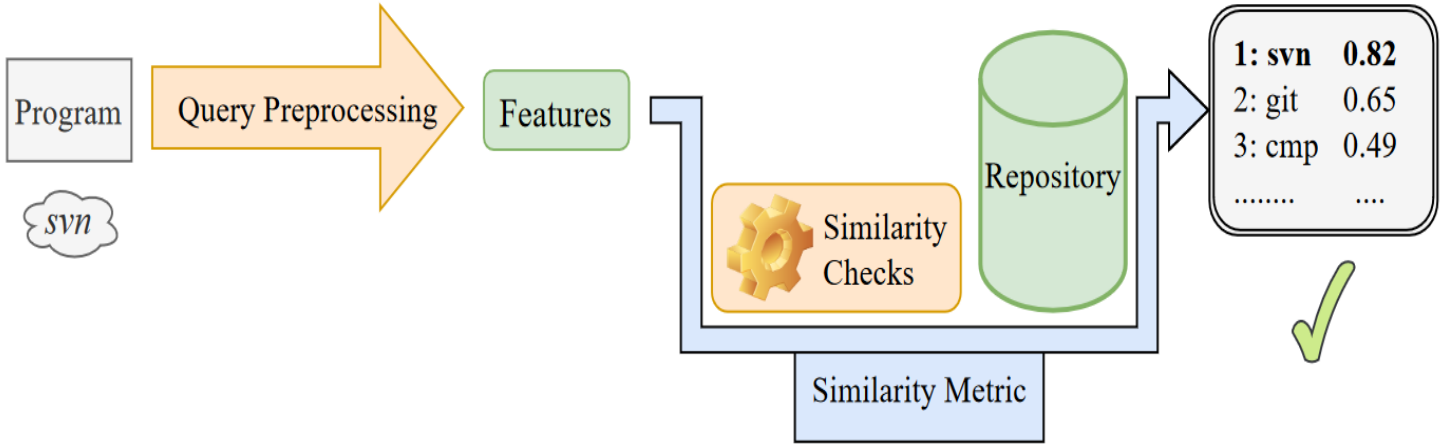


1. Prétraitement de la requête.

Lors de la requête, nous recevons le programme cible P.

Nous pouvons effectuer un prétraitement à cette étape, c'est à dire extraire des caractéristiques pour le reste de la procédure.

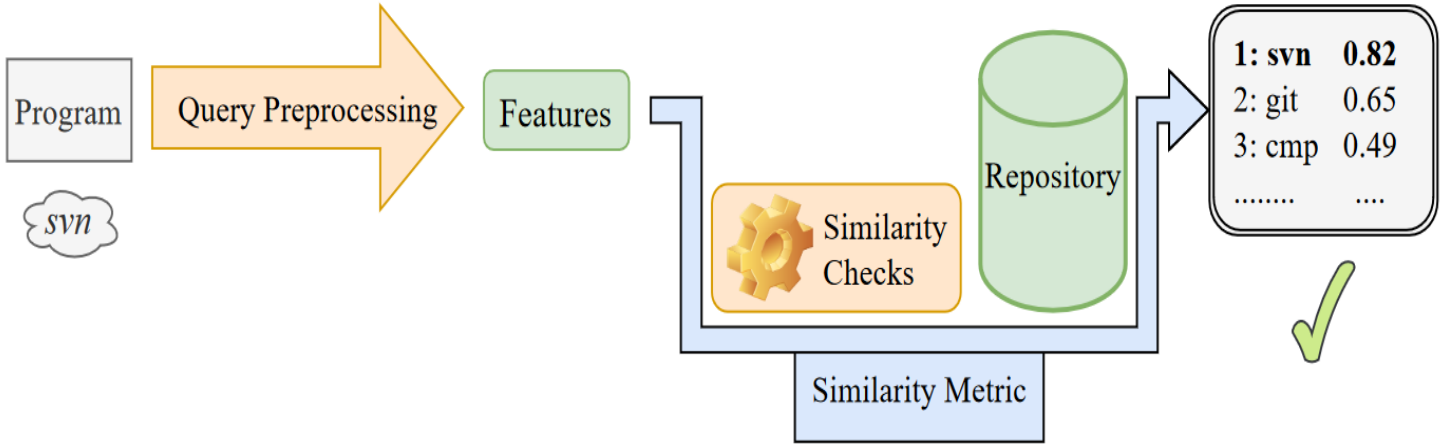
Recherche de clone



2. Calculs de similarité.

Pour chaque programme Q ∈ R, nous effectuons un calcul de similarité avec une métrique de similarité M sur (P, Q) et enregistrons l'indice de similarité M(P, Q).

Recherche de clone



3. Décision.

Le programme Qm ayant l'indice de similarité le plus élevé est considéré comme le plus similaire. La recherche de clone est un succès si Qm est un clone de P, sinon c'est un échec.

Exemple

Dépôt : 1420 bibliothèques.
Codes sources : 20 bibliothèques*.
Chaînes de compilations : 4 optimisations, 5 GCC, 4 Clang, pour les architectures x86-32 et x86-64.
Cibles : Les 20 avec GCC 6.4 -O2 pour x86-32.

*libiconv, coreutils, libtool, gss, gdbm, libtasn1, gsl, libmicrohttpd, osip, readline, gsasl, lightning, recutils, gmp, libunistring et glpk.

Loria

Exemple

Nous considérons 4 méthodes par vectorisation de fonction :
Asm2Vec, Gemini, SAFE et αDiff.
+ LibDB : Identificateurs littéraux et la vectorisation de fonction.

Pour obtenir un indice de similarité entre deux programmes à partir de méthodes par vectorisation de fonctions, nous devons réaliser un couplage entre les fonctions des deux programmes.

$$F(P, P') := - \sum_{x \in embeds(P)} \min_{y \in embeds(P')} \|x - y\|_2$$

Loria

Exemple

Méthode	Score	Temps total d'execution (temps de prétraitement inclus)
Asm2Vec [24] †	0,7	35h
Gemini [105] †	1	17h
SAFE [71] †	0,95	160h
αDiff [67] †	1	140h
LibDB [97] †	1	2h
PSS	1	**26s** (dont 26s de pretraitement)

† Temps d'apprentissage non inclus

- PSS est précis tout en étant rapide

Contributions

- Notre méthode PSS (Program Spectral Similarity).
- Une évaluation comprenant 15 méthodes et plus de 200 000 programmes divers. *Accessible sur github.*
- Des résultats selon lesquelles PSS est un bon compris entre la vitesse et la précision tout en étant très robuste.
- Des enseignements concernant les différentes classes de méthodes pour rechercher des clones.

Loria

PSS

On peut naturellement voir les programmes comme des graphes. La distance d'édition entre les graphes (GED) semble une bonne idée.

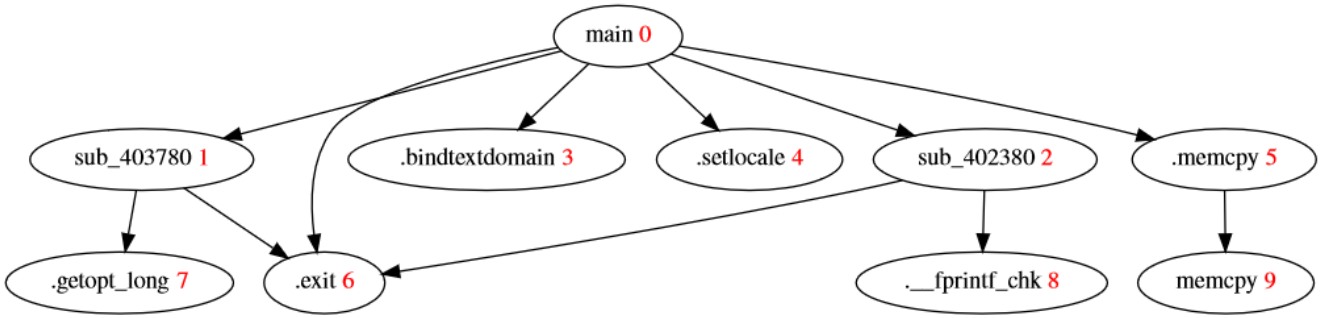
$$GED(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\mathcal{P}(g_1,g_2)} \sum_{i=1}^{k} c(e_i)$$

Opération d'édition de graphe :

- Suppression de sommet
- Suppression d'arête
- Ajout de sommet
- Ajout d'arête

NP-Complet et approximation bipartite en $O(n^3)$.

PSS



Définition 1. *Un graphe non dirigé $G = (V, E)$ de n sommets est représenté par une matrice d'adjacence A de dimension $n \times n$, où $a_{i,j}$ est 1 si $(V_i, V_j) \in E$ et 0 sinon. Soit d_i, le degré du sommet V_i. La matrice laplacienne est définie par :*

$$L_{i,j} := \begin{cases} d_i & si\ i = j\ et\ d_i \neq 0 \\ -1 & si\ i \neq j\ et\ A_{i,j} \neq 0 \\ 0 & sinon \end{cases}$$

Loria

PSS

Définition 2. *Une valeur propre λ et un vecteur propre $\vec{u}$ est une solution à l'équation $(L - \lambda I)\,\vec{u} = \vec{0}$. Le spectre est l'ensemble Λ des valeurs propres $\{\lambda_1(G), \lambda_2(G), \ldots, \lambda_{|G|}(G)\}$ où $\lambda_1(G) \geq \lambda_2(G) \geq \ldots \geq \lambda_{|G|}(G)$ et où $|G|$ est le nombre de sommet de G.*

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

Théorème 1. *Soit $e(G)$ le nombre d'arrêtes de G, alors $\sum_{i=1}^{|G|} \lambda_i(G) = 2e(G)$.*

PSS

Définition 3. *La distance spectrale [54] entre deux graphes G_1, G_2 de même ordre N est* $\sum_1^N |\lambda_i(G_1) - \lambda_i(G_2)|$.

La distance spectrale généralisée est définie par : $\sum_{i=1}^{min(|G1|,|G2|)} |\lambda_i(G_1) - \lambda_i(G_2)|$.

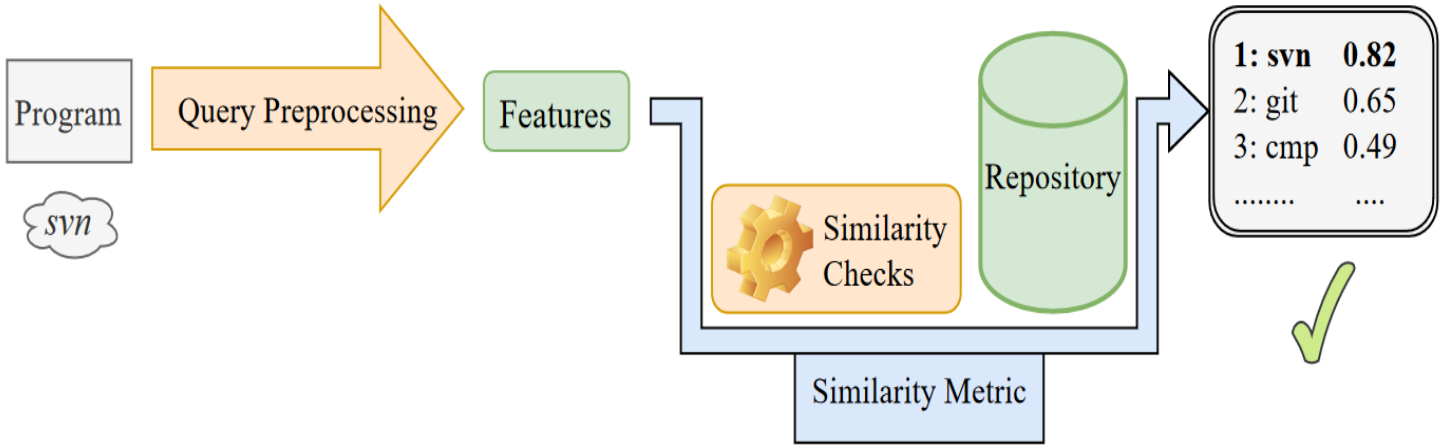
Théorème 2. *Soit G' une copie d'un graphe G auquel on a retiré un sommet de degré r. Pour tout i tel que $1 \leq i \leq n-1$, $\lambda_i(G) \geq \lambda_i(G') \geq \lambda_{i+r}(G)$.*

Corollaire 2.1. *La distance spectrale généralisée entre G et G' est égale à $2r$.*

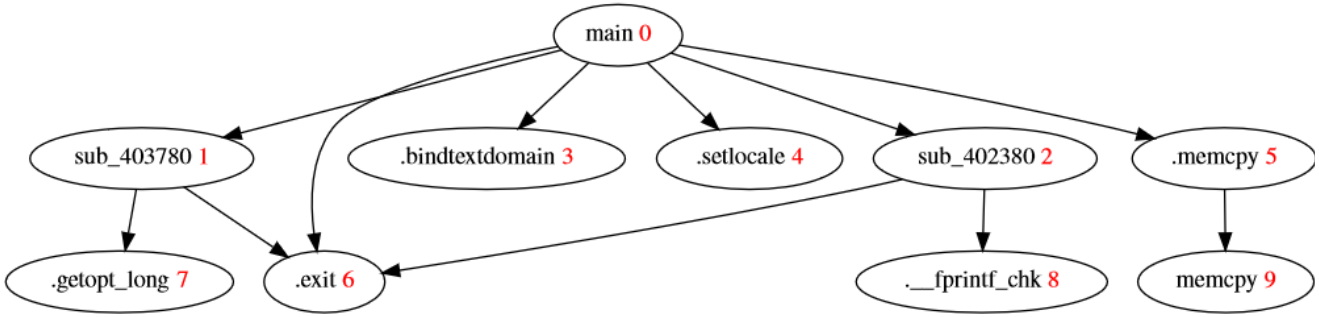
L'algorithme de Lanczos peut calculer le spectre en $O(dn^2)$ où n est le nombre de sommet, et d est le degré moyen du graphe.

PSS

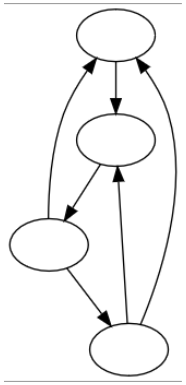
Idée de PSS :
Faire des comparaisons rapides des graphes via la distance spectrale de complexité O(n) après un calcul coûteux mais *unique* du spectre de la cible.

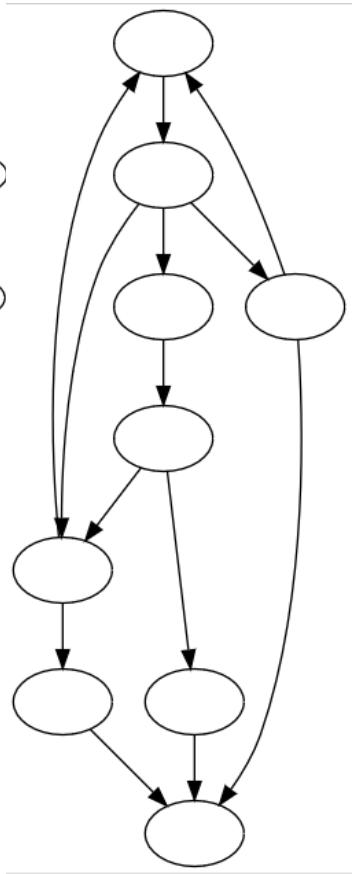


PSS



Autre intuition :
Mêler le graphe d'appel et les graphes de flot de contrôle des fonctions

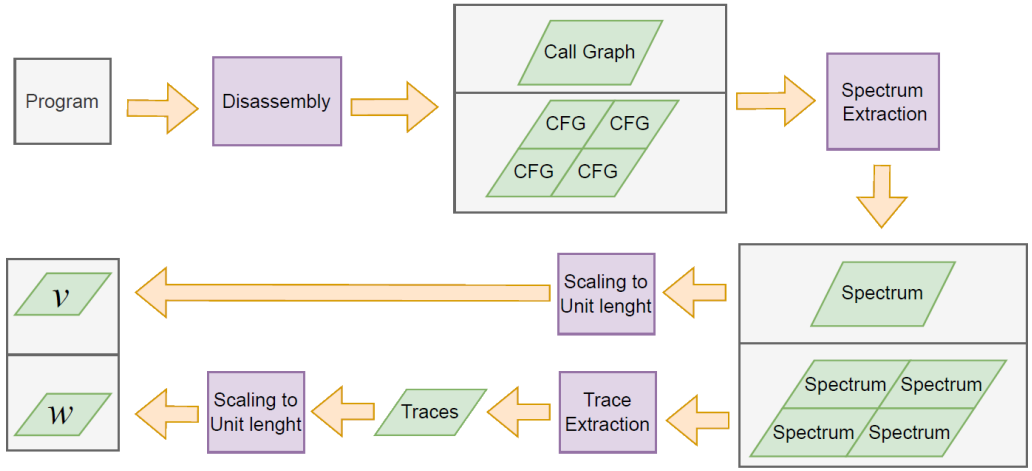




PSS

Le prétraitement de PSS est l'extraction de deux vecteurs clés :

1) Depuis une version *non dirigée* du graphe d'appel, nous calculons son spectre, et nous le normalisons.
2) Nous calculons les nombre d'arêtes de chaque graphe de flot de contrôle des fonctions locales. Nous rangeons ce vecteur par ordre décroissant, comme le spectre, et le normalisons.



PSS

Calcul de l'indice de similarité :

- Une norme 2 entre les vecteurs des programmes.
- Tenant compte de la taille libre de ces vecteurs.
- Avec sqrt(2) comme maximum de la distance entre vecteur.

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

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

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

PSS

Nous proposons PSSO, une version optimisée de PSS.
Au lieu de calculer le spectre complet, nous calculons seulement les 100 plus grandes valeurs propres des graphes d'appels.

L'algorithme de Lanczos peut calculer les k plus grandes valeurs propres en O(kdn) où n est le nombre de sommet et d est le degré moyen du graphe.

Loria

PSS

Si le temps d'exécution d'un calcul de similarité est T(n), le temps d'exécution du prétraitement est PT(n) et la taille du dépôt est D, alors le temps d'une requête est bornée par $D \times T(n) + PT(n)$.

Ainsi, un T(n) plus que linéaire n'est pas assez rapide sur un jeu de données important.

PSS

Complexité en temps des méthodes

Méthode	Classe	Calcul de Similarité†	Prétraitement‡
SMIT [45]	GED	$O(n^4)$	$O(dn)$
CGC [104]	Couplage	$O(n^4)$	$O(dn)$
MutantX-S [44]	**N-gramme**	$O(1)$	$O(i)$
Asm2Vec [24]	Fonction	$O(n^2)$	$O(n)$
Gemini [105]	Fonction	$O(n^2)$	$O(n)$
SAFE [71]	Fonction	$O(n^2)$	$O(n)$
αDiff [67]	Fonction	$O(n^2)$	$O(n)$
LibDX [96]	Chaînes	$O(s)$	$O(s)$
LibDB [97]	Chaînes et fonction	$O(n^2)$	$O(s+n)$
DeepBinDiff [26]	Apprentissage et couplage	$O(n^3m^3)$	Absent
PSS	**Spectral**	$O(n)$	$O(dn^2)$
PSSO	**Spectral**	$O(n)$	$O(dn)$

n : # fonctions, i : # instructions , s : # chaînes de caractères constantes
d : # appels par fonction, m : # blocs de base par fonction
† Entre deux programmes
‡ Une fois pour toute la recherche de clone

Étude systématique

1) Étude exhaustive sur Basique, un jeu de données réduit où toute les méthodes peuvent être évaluées en terme de vitesse, de précision et de robustesse.
2) Étude restreinte sur de grands jeux de données : BinKiT, Windows, IoT. Elle porte encore sur la vitesse, la précision et la robustesse des méthodes.

Étude systématique

Méthode	Classe	A	R	Calcul de similarité	IL	Appr.	Class.
B_{size}	**Référence**			$O(1)$			
D_{size}	**Référence**			$O(1)$			
Shape	**Référence**			$O(1)$			
ASCG [33]	**Spectrale**	×	×	$O(n)$			
ASCFG [33]	**Spectrale**	×	×	$O(1)$			
GED-0 [88]	GED	×	×	$O(n^3)$			
Asm2Vec [24]	Fonction	×		$O(n^2)$		×	
Gemini [105]	Fonction	×		$O(n^2)$		×	×
SAFE [71]	Fonction	×		$O(n^2)$		×	
MutantX-S [44]	**N-gramme**		×	$O(1)$			
DeepBinDiff [26]	Couplage			$O(n^3m^3)$		×	
PSS	**Spectrale**			$O(n)$			
PSS$_O$	**Spectrale**			$O(n)$			
ISO [4]	Couplage		×	$O(n^2)$	×		
CGC [104]	Couplage		×	$O(n^4)$	×		
GED-L [33]	GED	×	×	$O(n^3)$	×		
SMIT [45]	GED		×	$O(n^4)$	×		
αDiff [67]	Fonction	×	×	$O(n^2)$	×	×	
LibDX [96]	Chaînes		×	$O(s)$	×		
LibDB [97]	Chaînes et fonction		×	$O(n^2)$	×	×	×
StringSet	Chaînes			$O(s)$	×		
FunctionSet	**Chaînes**			$O(n)$	×		

A : Adapté pour la recherche de clone de programme, R : Réimplémenté
IL : Des identificateurs littéraux sont utilisés
Appr. : Phase d'apprentissage, Class. : Classification manuelle des mnémoniques

Étude systématique

Dans le travail de Xu et al. (CGC), il y a une méthode simple de décrite. Elle effectue un couplage entre fonctions en utilisant les noms des appels externes et les similarités des opérations machines.

Nous simplifions cette idée, et inventons la métrique de similarité FunctionSet. C'est la similarité de Jaccard entre les ensembles des appels externes des programmes.

$$FunctionSet(a,b) := \frac{|F_a \cap F_b|}{|F_a \cup F_b|}$$

Étude systématique

De même, nous inventons la métrique de similarité StringSet. C'est la similarité de Jaccard entre les ensembles des chaînes littérales des programmes.

$$StringSet(a,b) := \frac{|S_a \cap S_b|}{|S_a \cup S_b|}$$

Étude systématique

Projet	V0	V1	V2	V3
Coreutils	5.93	6.4	7.6	8.30
Binutils	2.25	2.27	2.31	2.35
Findutils	4.233	4.41	4.6	4.7.0
Diffutils	3.1	3.3	3.4	3.6
Bash	4.2	4.3	4.4	5.0
Codeblocks	13.12	16.01	17.12	20.03
Dia	0,94	0,95	0,96	0,97
Geany	1.23	1.27	1.32	1.36
Git	1.9.1	2.7.4	2.17.0	2.25.1
Graphviz	2.36.0	2.38.0	2.40.1	2.42.4
Libsdl2	2.0.10	2.0.14	2.0.2	2.0.8
Lua	5.1.5	5.2.4	5.3.3	5.4.2
Make	3.82	4.1	4.2	4.3
Openssh	6.6p1	7.2p2	7.6p1	8.2p1
Openssl	1.0.1f	1.0.2g	1.1.0g	1.1.1f
Perl	5.18.2	5.22.1	5.26.1	5.30.0
Ruby	1.9.1	2.0.0	2.5	3.0
Subversion	1.14.1	1.8.8	1.9.3	1.9.7
Vlc	2.1.2	2.2.2	3.0.12	3.0.1

Basique

6 sous jeux de données :

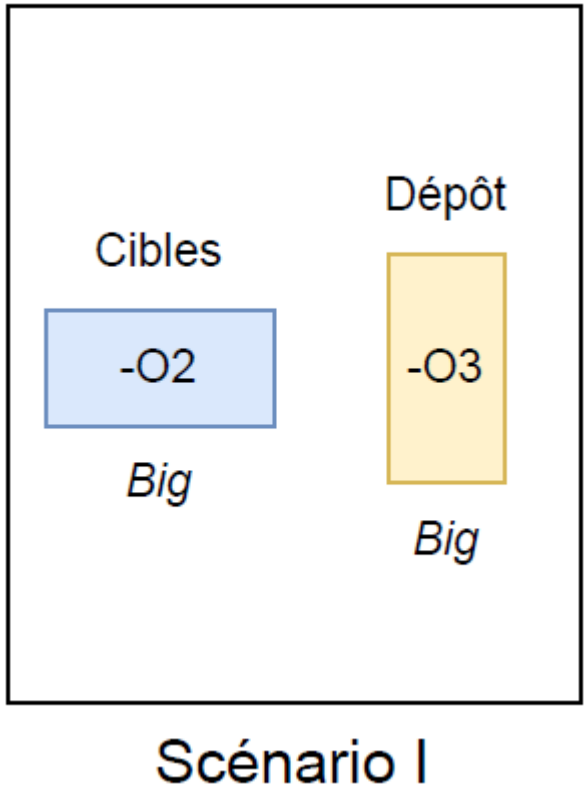
- Coreutils Versions (CV)
- Coreutils Optimisations (CO)
- Utils Versions (UV)
- Utils Optimisations (UO)
- Big Versions (BV)
- Big Optimisations (BO)

Étude systématique

Deux scénarios.

Dans le premier, le dépôt ne contient que des programmes avec une caractéristique différente (optimisation ou version différente).

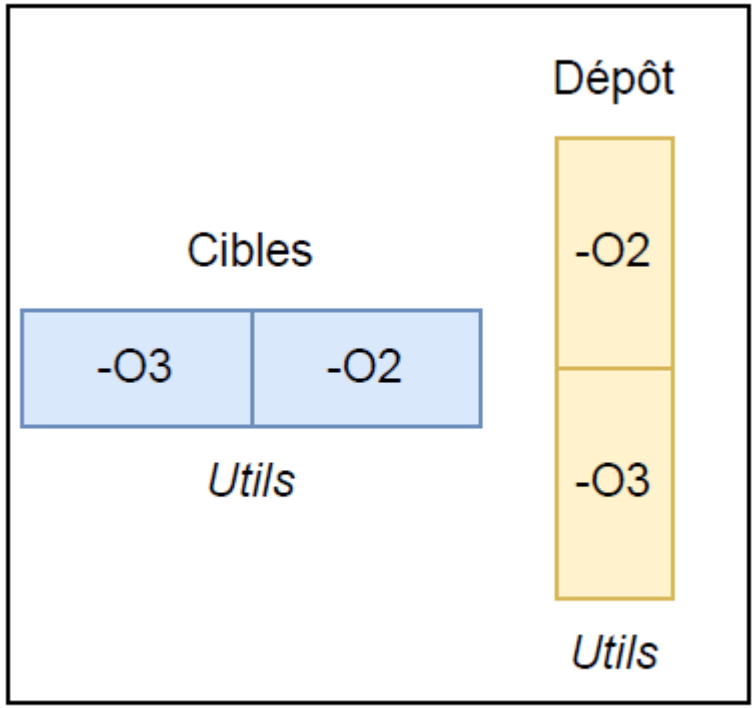
C'est le scénario le plus simple.



Étude systématique

Dans le second, le dépôt contient à la fois des programmes avec une caractéristique différente et des programmes ayant les mêmes caractéristiques.

C'est le scénario difficile à cause des caractéristiques communes entre des programmes différents.



Scénario II

Étude systématique

PSS est rapide, MutantX-S encore plus.
Les méthodes GED, couplage et par fonctions sont lentes.

Temps totaux dans le premier scénario

	CV	UV	BV	CO	UO	BO
B_{size}	2s	0s	0s	3s	0s	0s
D_{size}	2s	0s	0s	3s	0s	0s
Shape	12s	11s	14s	20s	10s	15s
ASCG	16s	2m26s	16m	24s	3m45s	55m
ASCFG	4h14m	25h6m	28h49m	4h35m	22h31m	42h
GED-0	50m	4h13m	20h23m	2h14m	6h29m	47h
Asm2Vec	1h50m	14h16m	34h53m	5h58m	20h24m	63h
Gemini	33m	10h50m	25h23m	2h12m	16h3m	47h
SAFE	9h6m	60h	175h	26h23m	86h	297h
MutantX-S	2s	0s	0s	2s	0s	0s
PSS	22s	2m41s	15m	30s	4m2s	54m
PSSO	45s	2m53s	3m34s	1m2s	2m51s	4m5s
GED-Labels	45m	3h28m	12h28m	1h57m	4h57m	23h20m
SMIT	2h56m	75h	683h	9h59m	285h	2577h
ISO	0s	0s	0s	0s	0s	0s
CGC	1h49m	13h17m	49h	4h9m	18h18m	84h
αDiff	16h1m	57h	161h	40h	79h	287h
LibDX	3s	17s	6s	8s	23s	7s
LibDB	6m41s	4h42m	1h33m	41m	6h33m	2h51m
StringSet	6s	7s	5s	8s	7s	5s
FunctionSet	1s	0s	0s	1s	0s	0s

Étude systématique

Les méthodes par vectorisation de fonction sont plus précises que PSS. PSS reste plus précise que GED, MutantX-S et les couplages.

StringSet et FunctionSet sont devant en utilisant les identificateurs.

Précision dans le premier scénario

	CV	CO	UV	UO	BV	BO	Moyenne
B_{size}	0,06	0,07	0,42	0,37	0,34	0,51	0.29
D_{size}	0,07	0,04	0,37	0,38	0,34	0,51	0.29
Shape	0,05	0,04	0,45	0,29	0,58	0,74	0.36
ASCG	0,15	0,07	0,52	0,36	0,63	0,60	0.39
ASCFG	0,12	0,08	0,07	0,07	0,14	0,12	0.10
GED-0	0,13	0,10	0,59	0,45	0,72	0,84	0.47
Asm2Vec	0,49	0,38	0,75	0,57	0,85	0,77	0.64
Gemini	0,55	0,34	0,57	0,38	0,87	0,59	0.55
SAFE	0,51	0,41	0,75	0,40	0,91	0,64	0.60
MutantX-S	0,43	0,14	0,63	0,36	0,83	0,55	0.49
PSS	0,28	0,25	0,58	0,43	0,78	0,75	0.51
PSSO	0,27	0,25	0,56	0,45	0,77	0,74	0.51
GED-Labels	0,29	0,25	0,55	0,33	0,71	0,79	0.49
SMIT	0,04	0,01	0,09	0,08	0,14	0,11	0.08
ISO	0,01	0,01	0,04	0,05	0,04	0,07	0.04
CGC	0,11	0,12	0,62	0,45	0,58	0,51	0.40
αDiff	0,24	0,25	0,74	0,61	0,85	0,58	0.55
LibDX	0,61	0,69	0,81	0,73	0,86	0,85	0.76
LibDB	0,70	0,46	0,92	0,74	0,91	0,88	0.77
StringSet	0,92	0,96	0,91	0,91	0,97	1.00	0.94
FunctionSet	0,85	0,86	0,90	0,90	0,95	0,96	0.90

Loria

Étude systématique

PSS a l'avantage par rapport à MutantX-S et aux méthodes par vectorisation de fonctions.

LibDX est en tête en utilisant les identificateurs littéraux.

Précision dans le second scénario

	CV	CO	UV	UO	BV	BO	Moyenne
B_{size}	0,05	0,04	0,30	0,25	0,17	0,34	0,19
D_{size}	0,04	0,02	0,27	0,24	0,27	0,40	0,21
Shape	0,03	0,01	0,27	0,12	0,51	0,67	0,27
ASCG	0,10	0,03	0,39	0,30	0,58	0,56	0,32
ASCFG	0,08	0,03	0,06	0,04	0,11	0,10	0,07
GED-0	0,11	0,02	0,35	0,28	0,67	0,77	0,37
Asm2Vec	0,01	≈ 0	0,03	0,02	0,45	0,37	0,15
Gemini	0,15	0,01	0,23	0,05	0,83	0,46	0,29
SAFE	0,08	0,01	0,25	0,06	0,81	0,42	0,27
MutantX-S	0,14	0,01	0,37	0,13	0,79	0,40	0,31
PSS	0,16	0,10	0,36	0,28	0,75	0,65	0,38
PSSO	0,17	0,11	0,36	0,28	0,73	0,65	0,38
GED-Labels	0,16	0,19	0,40	0,23	0,62	0,65	0,37
SMIT	0,02	≈ 0	0,08	0,07	0,14	0,10	0,07
ISO	0,01	≈ 0	0,02	0,04	0,02	0,02	0,01
CGC	0,05	0,04	0,39	0,24	0,51	0,44	0.28
αDiff	0,06	≈ 0	0,32	0,16	0,74	0,43	0,28
LibDX	0,56	0,69	0,55	0,70	0,82	0,85	0,69
LibDB	0,19	0,07	0,41	0,20	0,88	0,79	0,42
StringSet	0,17	0,12	0,36	0,42	0,88	0,90	0,48
FunctionSet	0,40	0,51	0,55	0,52	0,87	0,89	0,62

Loria

Étude systématique

Robustesse

La robustesse d'une métrique de similarité est sa résistance à l'impact de propriétés telles que le niveau d'optimisation ou le niveau de version.

Nous calculons des corrélations rang-bisérial entre (a) le partage d'un niveau d'optimisation ou de version et (b) le classement en terme de similarité calculé par une métrique.

Dans l'idéal : aucune corrélation.

Étude systématique

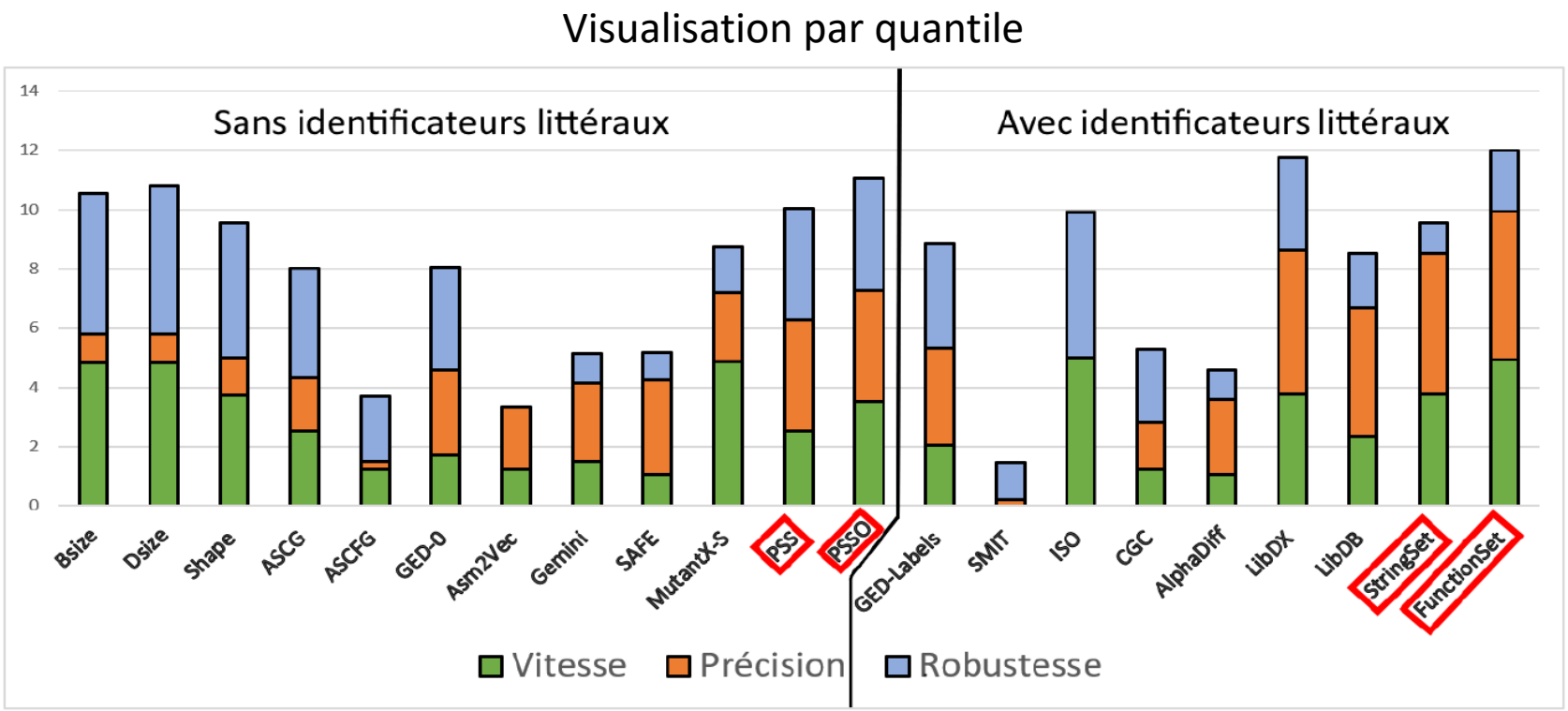
PSS est robuste, contrairement aux méthodes par vectorisation de fonctions et à MutantX-S.
Les méthodes GED sont moins robuste que PSS.
LibDX est parfaitement robuste.

Corrélation moyenne

	CV	CO	UV	UO	BV	BO	Moyenne
B_{size}	0,17	**0,07**	**-0.02**	**0,03**	**-0.03**	**-0.04**	**0,03**
D_{size}	**0,11**	**0,02**	**-0.02**	**0,06**	**-0.04**	**-0.04**	**0,02**
Shape	**0,15**	**0,10**	**-0.03**	**0,06**	**-0.04**	**-0.04**	**0,03**
ASCG	**0,10**	0,19	**-0.01**	**0,08**	**-0.04**	**-0.04**	**0,05**
ASCFG	0,23	0,30	**0,15**	0,17	**-0.02**	**-0.02**	**0,13**
GED-0	0,22	0,25	**-0.02**	**0,05**	**-0.04**	**-0.04**	**0,07**
Asm2Vec	0,99	1.00	0,49	0,65	0,32	0,45	0,65
Gemini	0,76	0,96	0,19	0,37	**-0.04**	**0,06**	0,38
SAFE	0,81	0,98	0,20	0,38	**-0.04**	**0,11**	0,41
MutantX-S	0,39	0,63	**0,05**	0,28	**-0.04**	**0,08**	0,23
PSS	**0,06**	**0,13**	**0,02**	**0,09**	**-0.04**	**-0.02**	**0,04**
PSSO	**0,07**	**0,12**	**0,02**	**0,09**	**-0.04**	**-0.02**	**0,04**
GED-Labels	0,16	0,21	**-0.01**	**0,08**	**-0.04**	**-0.04**	**0,06**
SMIT	**-0.15**	-0.57	**-0.15**	-0.44	**-0.01**	**-0.07**	-0.23
ISO	$\approx$ **0**	$\approx$ **0**	$\approx$ **0**	**0,03**	$\approx$ **0**	**-0.01**	$\approx$ **0**
CGC	0,19	0,32	**0,08**	**0,07**	**-0.04**	**-0.08**	**0,09**
αDiff	0,60	0,93	0,19	0,33	**-0.04**	**0,11**	0,35
LibDX	0,32	**-0.02**	**-0.05**	-0.17	**-0.05**	**-0.04**	$\approx$ **0**
LibDB	0,54	0,46	0,17	0,21	**-0.04**	**-0.03**	0,22
StringSet	0,73	0,86	0,17	0,31	**-0.04**	0,18	0,37
FunctionSet	0,39	0,37	**0,08**	0,22	**-0.04**	**0,02**	0,17

Corrélation absolu moyenne inférieur à 0.16

Étude systématique



➢ PSS est un bon compromis

Étude systématique

Temps totaux pour le premier scénario.

B$_{size}$	$\leq$ 1h30m
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

Méthodes rapides sélectionnées pour l'analyses sur des dépôts larges.
† Temps d'apprentissage non inclus

Étude systématique

Jeu de données : BinKiT (Normal)

Depuis 51 paquet logiciel GNU, 235 codes sources ont été extraits. Ils ont été compilés avec 288 chaînes de compilation différentes pour un total de 67 680 programme.

Le jeu de données couvre :

- 8 architectures (arm, x86, mips, et mipseb, en 32 et 64 bits)
- 9 compilateurs (5 versions de GCC et 4 versions de Clang)
- 4 niveaux d'optimisation de -O0 à -O3.

Étude systématique

Jeu de données : BinKiT (Offuscation)

Le même jeu que précédemment, mais avec Obfuscator-LLVM comme compilateur et 4 offuscations :

1) La substitution d'instruction (SUB)
2) Le bug de flot de contrôle (BCF)
3) L'aplatissement du flot de contrôle (FLA)
4) La combinaison des trois.

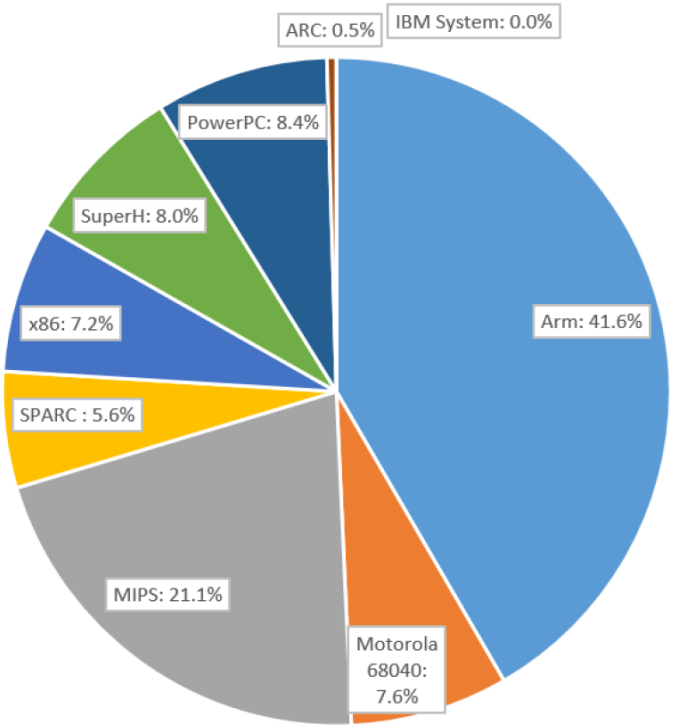
Étude systématique

Jeu de données : IoT

Depuis MalwareBazaar, nous rassemblons 19 959 micrologiciels malveillants soumis entre Mars 2020 et Mai 2022.

Nous décomposons les données en trois familles de clones : 12 357 Mirai, 5 842 Gafgyt, et 1 760 Tsunami.

Architecture des micrologiciels



Étude systématique

Jeu de données : Windows

Nous assemblons un jeu de données de 84 992 programmes sains fonctionnant sous les systèmes d'exploitation Windows. Cela représente plus de 50 Go de programmes bruts.

Nous définissons les clones comme ayant le même nom de programme, et la même plateforme. Ainsi, 49 443 programmes ont un clone.

Distribution des plateformes

Étude systématique

PSS est rapide, sauf sur Windows. PSSO résout ce soucis.
MutantX-S et FunctionSet sont très rapides également, alors que LibDX et StringSet prennent plus de temps.

Temps d'execution, prétraitement significatif dans ()

Jeu de données # Programmes	Basique 950	BinKiT 97 760	IoT 19 959	Windows 84 992
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)
PSSO	0,27 (0.27)	0,30 (0.04)	0,44 (0.14)	2.29 (0.39)
LibDX	0,02	5.09	1.40	12.43
StringSet	0,01	1.40	1.69	18.47
FunctionSet	$< 0,01$	0,10	0,03	2.01

Étude systématique

PSS est aussi précise que MutantX-S sauf sur BinKiT où elle est mieux.
Avec des identificateurs littéraux: StringSet est le plus précis.

Précision (succès moyen)

Jeu de données # Programmes	Basique 950	BinKiT 97 760	IoT 19 959	Windows 84 992
B_{size}	0,17	0,176	0,819	0,196
D_{size}	0,16	0,065	0,787	0,445
Shape	0,19	0,296	0,818	0,388
ASCG	0,24	0,549	0,759	0,444
MutantX-S	0,38	0,365	0,870	0,472
PSS	0,38	0,621	0,863	0,475
PSSO	0,38	0,621	0,862	0,466
LibDX	0,70	0,884	0,707	0,044
StringSet	0,94	0,970	0,922	0,501
FunctionSet	0,87	0,491	0,624	0,426

Étude systématique

Précision sur BinKiT

Catégorie	Niveau d'opt.		Chang. de compilateur			Chang. d'architecture				vs. Offuscation†			
vs	O0 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}	0,07	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	0,07	0,07	0,09	0,04	0,02	0,05	0,03	0,04	0,02	0,01	0,05	0,01
Shape	0,06	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	0,10	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	0,03	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	0,17	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	0,17	0,68	0,78	0,90	0,51	0,44	0,54	0,44	0,66	0,52	0,56	0,82	0,46
LibDX	0,89	0,89	0,89	0,86	0,78	0,87	0,89	0,90	0,88	0,87	0,86	0,86	0,86
StringSet	0,97	0,97	0,97	0,97	0,97	0,96	0,98	0,96	0,97	0,96	0,97	0,96	0,97
FunctionSet	0,53	0,56	0,46	0,68	0,55	0,29	0,02	≈ 0	0,23	0,61	0,61	0,61	0,61

†: Le jeu de données BinKiT ne considère pas d'obsfucation des identificateurs littéraux.

PSS et PSSO sont très robustes, même face à un changement d'architecture et contre les offuscations.
Méthodes avec identificateurs: robustes, à part FunctionSet.

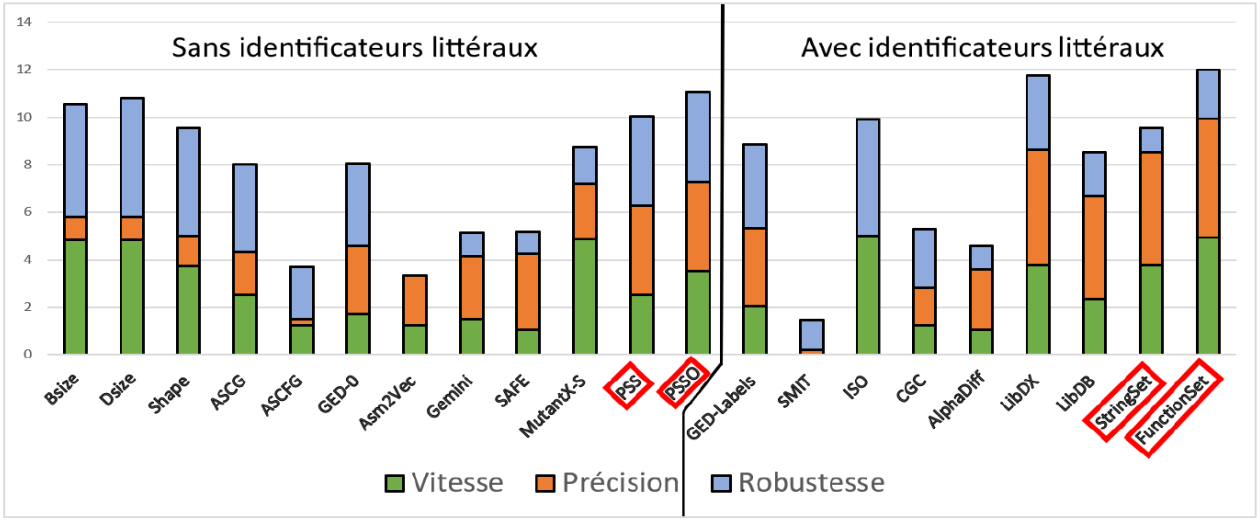
Loria

Étude systématique

Conclusion de notre étude :

Méthode	Vitesse	Précision	Robustesse	Attention à
PSS/PSSO	+	+	+ +	
MutantX-S	+	+	- -	
StringSet	- -	+ +	+ +	Offuscation des chaînes
FunctionSet	+ +	-	-	Offuscation des appels externes
				Bibliothèque statique

Rappel,
sur Basique:



Conclusion

Nous proposons PSS, et sa version optimisée PSSO, qui atteignent un point optimal en termes de vitesse, de précision et de robustesse.

Ce travail ouvre la voie à :

- Une étude des approches spectrales pour d'autres problèmes.
- L'application de hachage par dessus les spectres.
- Des comparaisons plus courantes avec des méthodes simples.

Merci pour votre attention !

Loria

Références A

[A] « SAFE: Self-Attentive Function Embeddings for Binary Similarity ». Massarelli, DiLuna, Petroni, Baldoni, Querzoni. DIMVA 2019.
[B] « The Tangled Genealogy of IoT Malware ». Cozzi, Vervier, Dell'Amico, Shen, Balzarotti. ACSAC 2020.
[C] « LibDX: A Cross-Platform and Accurate System to Detect Third-Party Libraries in Binary Code ». Tang, Luo, Fu, Zhang. SANER 2020.
[D] « A survey of binary code similarity ». Irfan Ul Haq and Juan Caballero. ACM Computing Surveys 2021.

Références B

[4] Bai, Shi, Mu. A malware and variant detection method using function call graph isomorphism. Security and Communication Networks, 2019.
[18] Chung. Spectral graph theory. American Mathematical Society, 1997.
[24] Ding, Fung, Charland. Asm2vec : Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. IEEE Symposium on Security and Privacy, 2019.
[26] Duan, X. Li, J. Wang, H. Yin. Learning program-wide code representations for binary diffing. Network and Distributed System Security Symposium, 2020.
[33] Fyrbiak, Wallat, Reinhard, Bissantz Paar. Graph similarity and its applications to hardware security. IEEE Transactions on Computers, 2020.
[38] Hall. The adjacency matrix, standard laplacian, and normalized laplacian, and some eigenvalue interlacing results. Department of Mathematics and Statistics, Georgia State University, Atlanta, 2010.

Loria

Références B

[44] X. Hu, Bhatkar, Griffin, K. G. Shin. Mutantx-s : Scalable malware clustering based on static features. USENIX Conference on Annual Technical Conference, 2013.
[45] X. Hu, Chiueh, K. G. Shin. Large-scale malware indexing using function call graphs. ACM Conference on Computer and Communications Security, 2009.
[54] Jovanović, Stanić. Spectral distances of graphs. Linear Algebra and its Applications, 2012.
[67] B. Liu, W. Huo, C. Zhang, W. Li, F. Li, A. Piao, W. Zou. αdiff : Cross-version binary code similarity detection with dnn. IEEE/ACM International Conference on Automated Software Engineering, 2018.
[71] Massarelli, Di Luna, Petroni, Querzoni, Baldoni. Function representations for binary similarity. IEEE Transactions on Dependable and Secure Computing, 2021.

Loria

Références B

[96] W. Tang, P. Luo, J. Fu, D. Zhang. Libdx : A cross-platform and accurate system to detect third-party libraries in binary code. SANER, 2020.
[97] W. Tang, Y. Wang, H. Zhang, S. Han, P. Luo, D. Zhang. Libdb : An effective and efficient framework for detecting third-party libraries in binaries. International Conference on Mining Software Repositories, 2022.
[104] M. Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren, N. Zheng. A similarity metric method of obfuscated malware using function-call graph. Journal of Computer Virology and Hacking Techniques, 2013.
[105] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, D. Song. Neural network based graph embedding for cross-platform binary code similarity detection. ACM SIGSAC Conference on Computer and Communications Security, 2017.

Slides supplémentaires