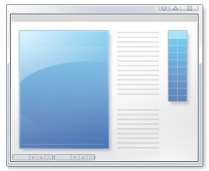
Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization

Steven H. H. Ding
Data Mining and Security Lab
School of Information Studies
McGill University
Montreal, Canada

Benjamin C. M. Fung
Data Mining and Security Lab
School of Information Studies
McGill University,
Montreal, Canada

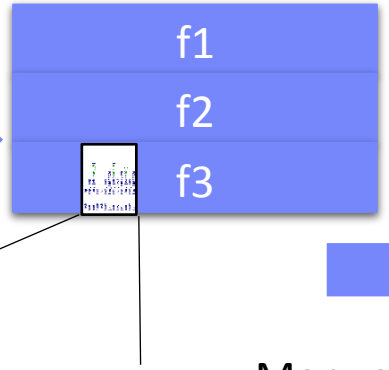
Philippe Charland
Mission Critical Cyber Security Section
Defence R&D Canada – Valcartier
Quebec, Canada

Reverse engineering



A binary file

Disassemble




```
LDR     R3, [R11,#sct]
LDR     R2, [R3,#0xC]
LDR     R3, [R11,#applet_no]
CMP     R2, R3
BEQ     loc_DFD0
LDR     R3, [R11,#sct]
LDR     R3, [R3]
STR     R3, [R11,#sct]
loc_DFC0
LDR     R3, [R11,#sct]
CMP     R3, #0
BNE     loc_DFA0
```

Manual analysis

Reverse engineer

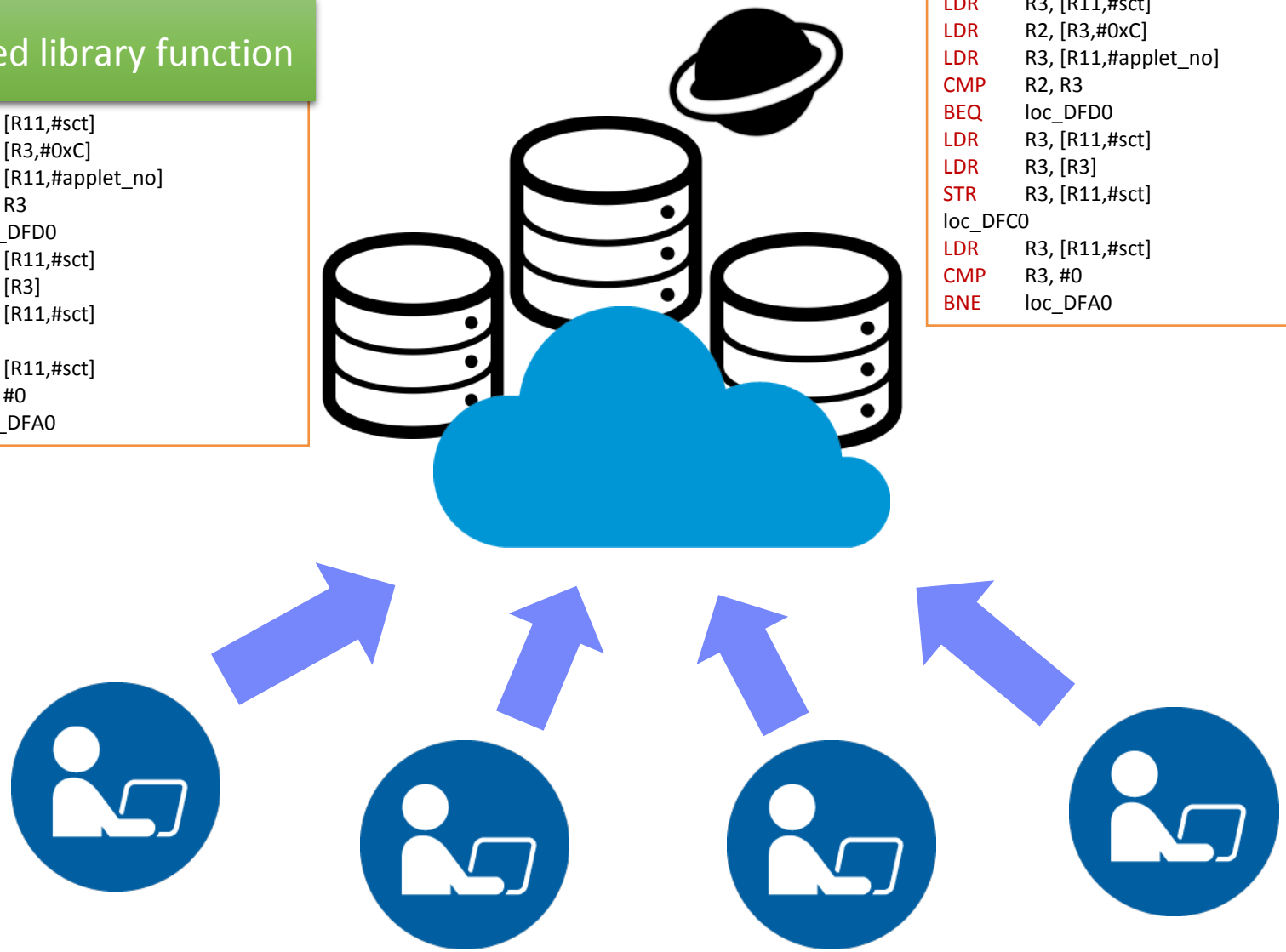
Did anyone analyze something similar before?

Is it a library function?

With Kam1n0

Labeled library function

```
LDR     R3, [R11,#sct]
LDR     R2, [R3,#0xC]
LDR     R3, [R11,#applet_no]
CMP     R2, R3
BEQ     loc_DFD0
LDR     R3, [R11,#sct]
LDR     R3, [R3]
STR     R3, [R11,#sct]
loc_DFC0
LDR     R3, [R11,#sct]
CMP     R3, #0
BNE     loc_DFA0
```



Commented assembly function

```
LDR     R3, [R11,#sct]
LDR     R2, [R3,#0xC]
LDR     R3, [R11,#applet_no]
CMP     R2, R3
BEQ     loc_DFD0
LDR     R3, [R11,#sct]
LDR     R3, [R3]
STR     R3, [R11,#sct]
loc_DFC0
LDR     R3, [R11,#sct]
CMP     R3, #0
BNE     loc_DFA0
```

Type I: Exact clone

```
0x1FE69C0   PUSH ebp
0x1FE69C1   MOV ebp, esp
0x1FE69C3   MOV ecx, [ebp+arg_0]
0x1FE69C6   PUSH ebx
0x1FE69C7   MOV ebx, [ebp+arg_8]
0x1FE69CA   PUSH esi
0x1FE69CB   MOV esi, ecx
0x1FE69CD   AND ecx, 0FFFFh
0x1FE69D3   SHR esi, 10h
0x1FE69D6   CMP ebx, 1
0x1FE69D9   +JNZ loc_1FE6A0C
```

```
0x1FE69C0   PUSH ebp
0x1FE69C1   MOV ebp, esp
0x1FE69C3   MOV ecx, [ebp+arg_0]
0x1FE69C6   PUSH ebx
0x1FE69C7   MOV ebx, [ebp+arg_8]
0x1FE69CA   PUSH esi
0x1FE69CB   MOV esi, ecx
0x1FE69CD   AND ecx, 0FFFFh
0x1FE69D3   SHR esi, 10h
0x1FE69D6   CMP ebx, 1
0x1FE69D9   +JNZ loc_1FE6A0C
```

Type II: Syntactically equivalent

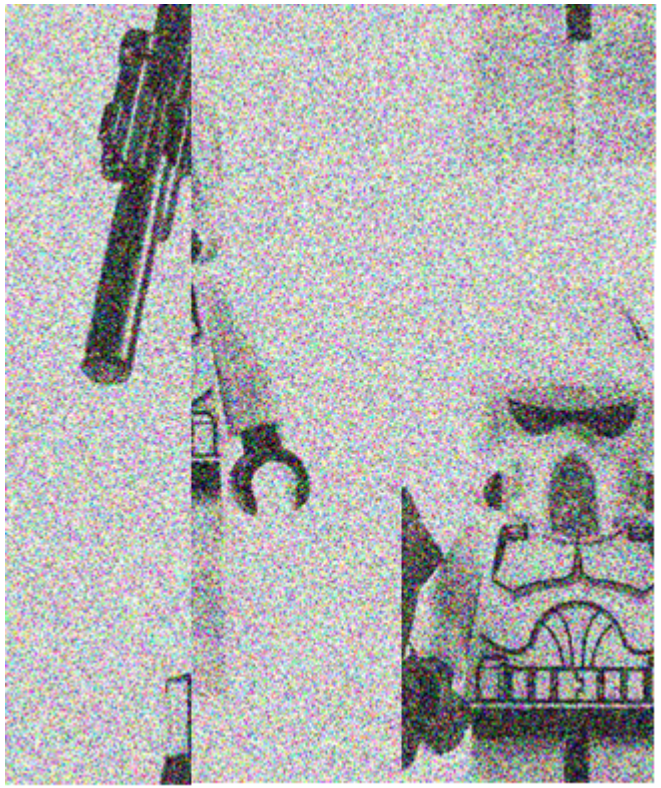
```
0x1FE05B0   PUSH ebp
0x1FE05B1   MOV ebp, esp
0x1FE05B3   MOV ecx, [ebp+arg_0]
0x1FE05B6   PUSH ebx
0x1FE05B7   MOV ebx, [ebp+arg_8]
0x1FE05BA   PUSH esi
0x1FE05BB   MOV esi, ecx
0x1FE05BD   AND ecx, 0FFFFh
0x1FE05B3   SHR esi, 10h
0x1FE05B6   CMP ebx, 1
0x1FE05B9  +JNZ loc_1FE05BC
```

```
0x1FE69C0+  PUSH ebp
0x1FE69C1+  MOV ebp, esp
0x1FE69C3+  MOV eax, [ebp+msg_0]
0x1FE69C6+  PUSH edx
0x1FE69C7+  MOV edx, [ebp+msg_1]
0x1FE69CA+  PUSH esi
0x1FE69CB+  MOV esi, eax
0x1FE69CD+  AND eax, 0FFFFh
0x1FE69D3+  SHR esi, 10h
0x1FE69D6+  CMP edx, 1
0x1FE69D9+ +JNZ loc_1FE6A0C
```

Type III: Minor modification

```
0x1FE05B0    PUSH ebp
0x1FE05B1    MOV ebp, esp


0x1FE05B7    MOV ebx, [ebp+arg_8]
0x1FE05BA    PUSH esi
0x1FE05BB    MOV esi, ecx
0x1FE05BD    AND ecx, 0FFFFh
0x1FE05B3    MOV eax, ecx
0x1FE05B6    SHR esi, 10h
0x1FE05B9    CMP ebx, 1
0x1FE05C1   +JNZ loc_1FE05BC
```

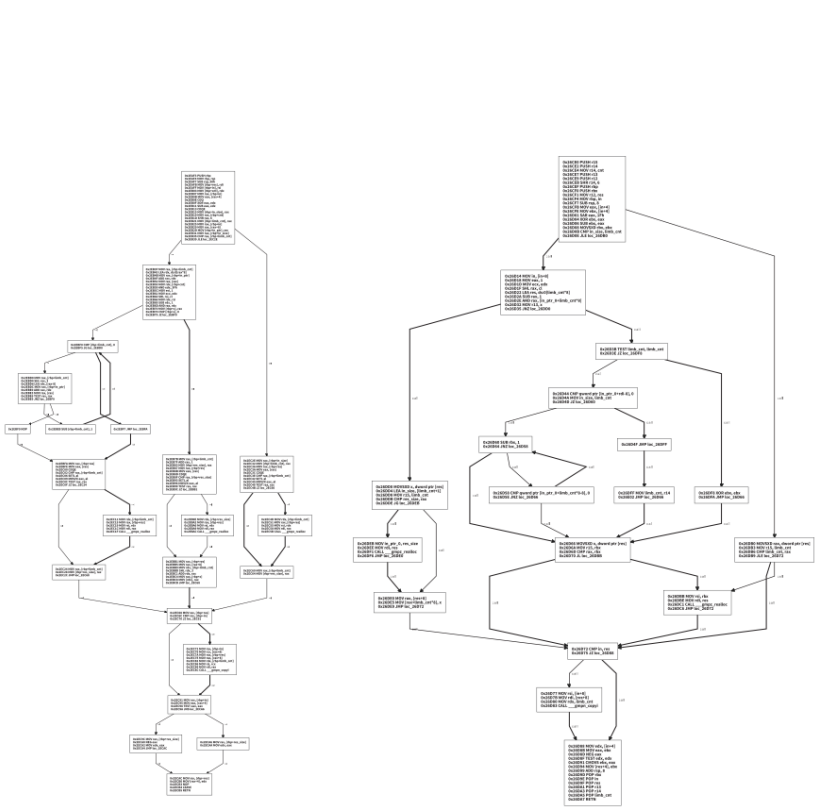
```
0x1FE69C0    PUSH ebp
0x1FE69C1    MOV ebp, esp
0x1FE69C3    MOV eax, [ebp+msg_0]
0x1FE69C6    PUSH edx
0x1FE69C7    MOV edx, [ebp+msg_1]
0x1FE69CA    PUSH esi
0x1FE69CB    MOV esi, eax
0x1FE69CD    AND eax, 0FFFFh


0x1FE69D3    SHR esi, 10h
0x1FE69D6    CMP edx, 1
0x1FE69D9   +JNZ loc_1FE6A0C
```

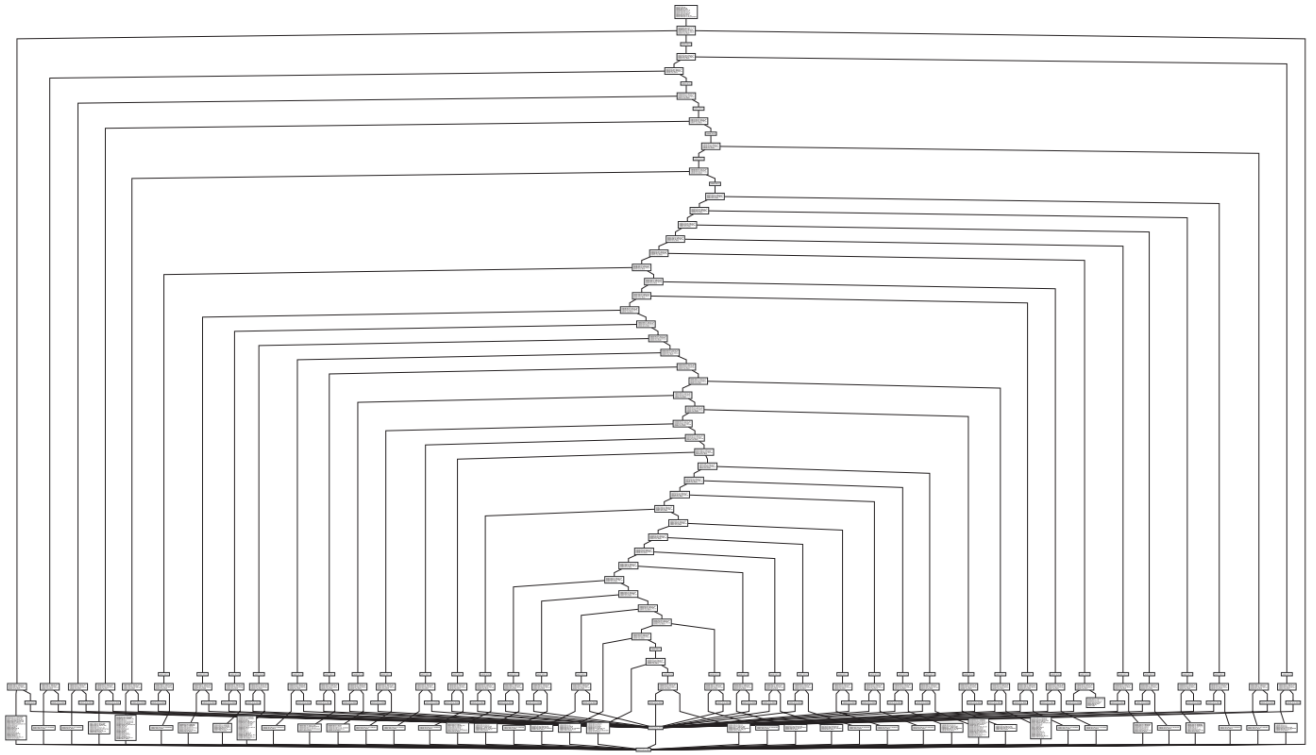


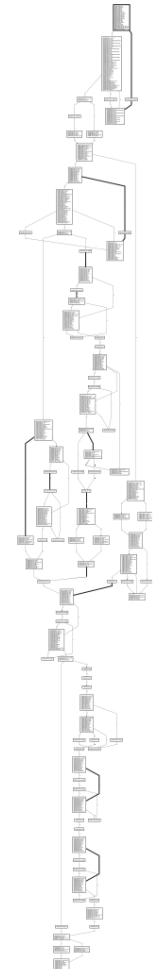
clone

original

Obfuscation and Optimization - Challenges







Obfuscation and Optimization - Problems

- P1: The relationships among assembly tokens
  - xmm0 (SSE) register vs. SSE operations such as movaps
  - *fclose* vs. *fopen*.
  - *strcpy* vs. *memcpy*.

- P2: Token combination weights
  - Reverse engineers look for ‘interesting pattern’. (higher weight)
  - Regular, random, or repeated pattern is not interesting. (lower weight)

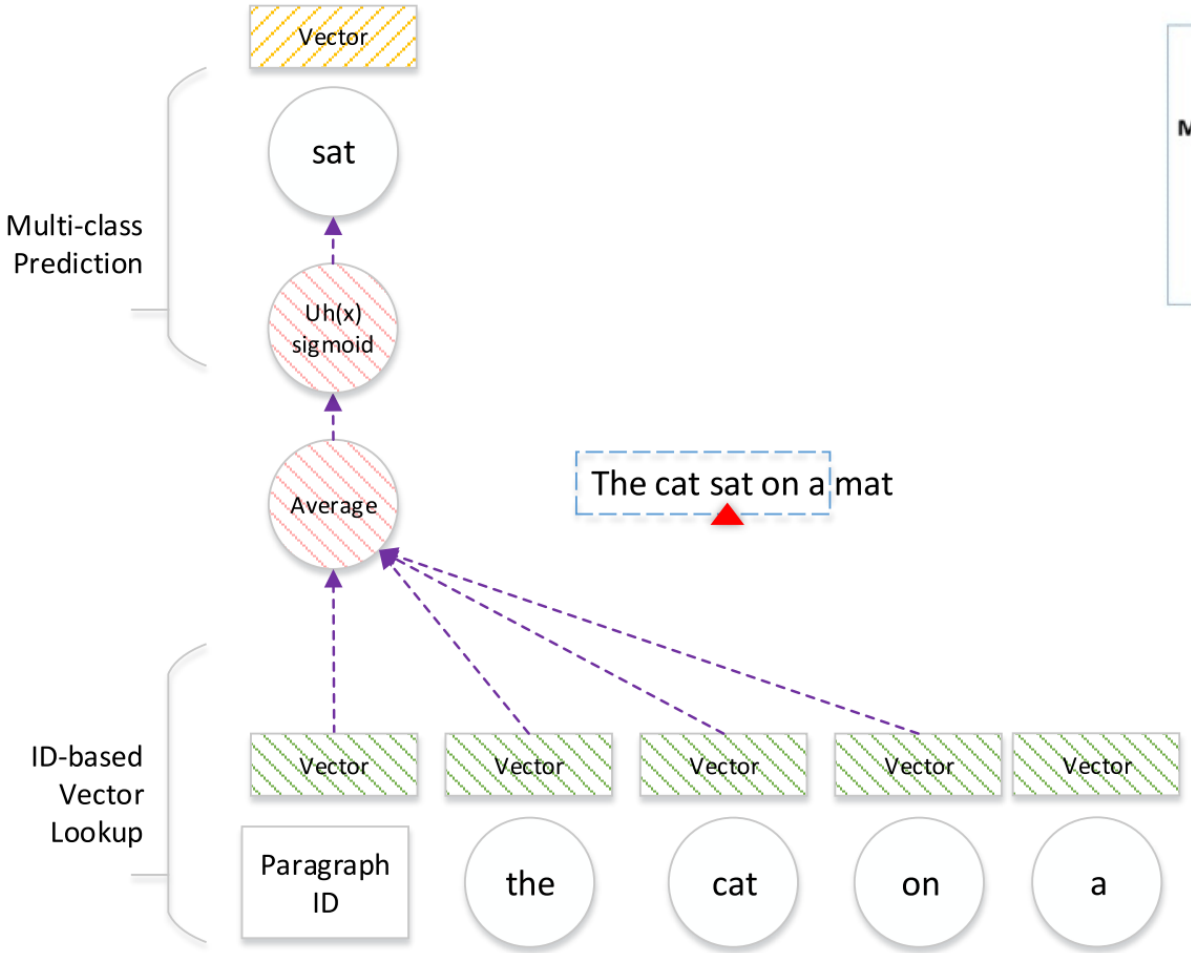
- Sound so familiar in NLP!

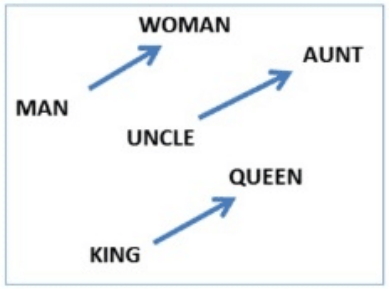
Learning English

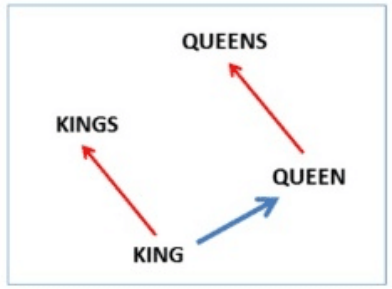
1) The cat ____ on the mat.

A: food
B: sat
C: sitting
D: is speaking

Paragraph Vector (p2vec):







king – man + woman = queen

bad - good = maniacal_killer *

Figure 4: The PV-DM model.

* Example collected from Andreas Mueller@amuellerml

Asm2Vec:

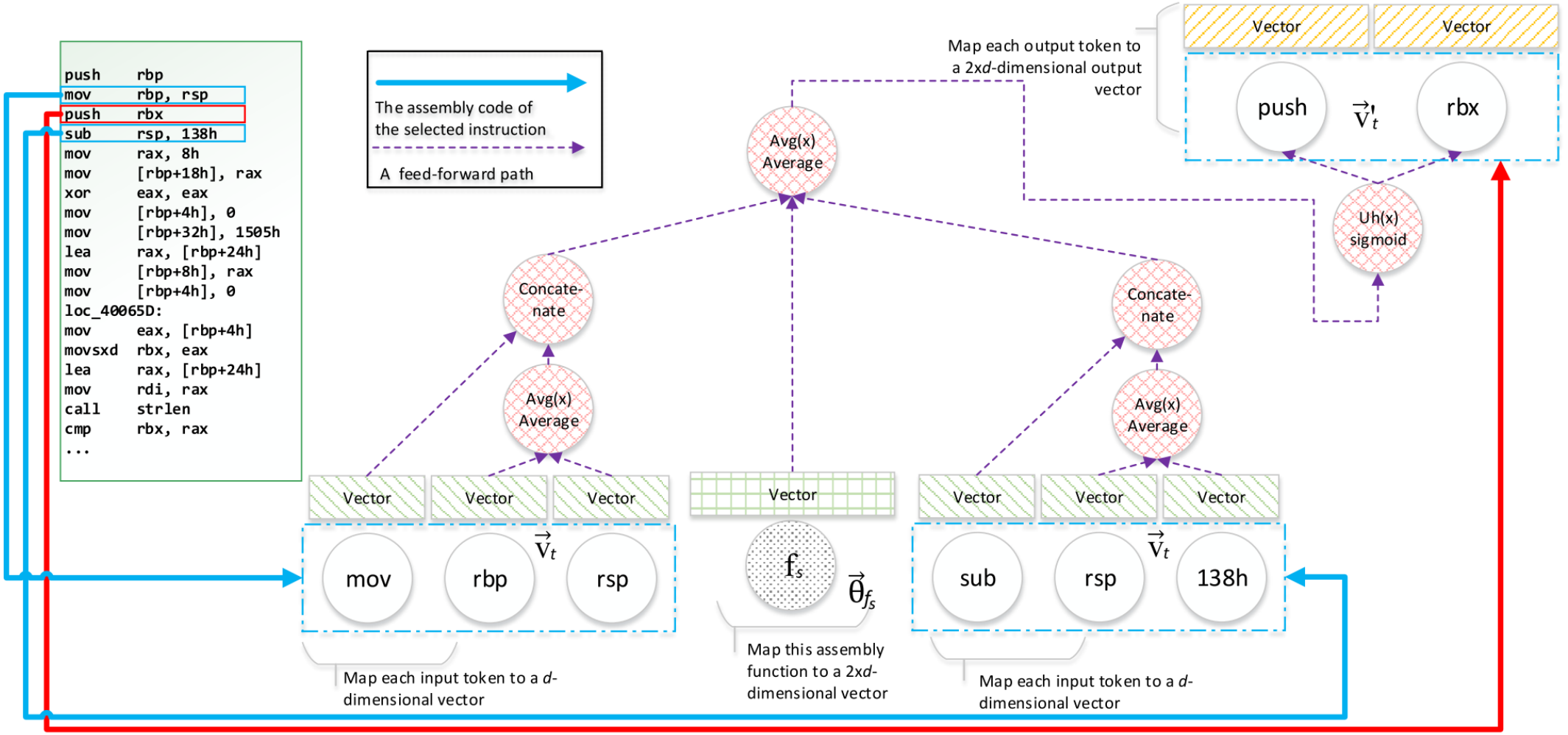
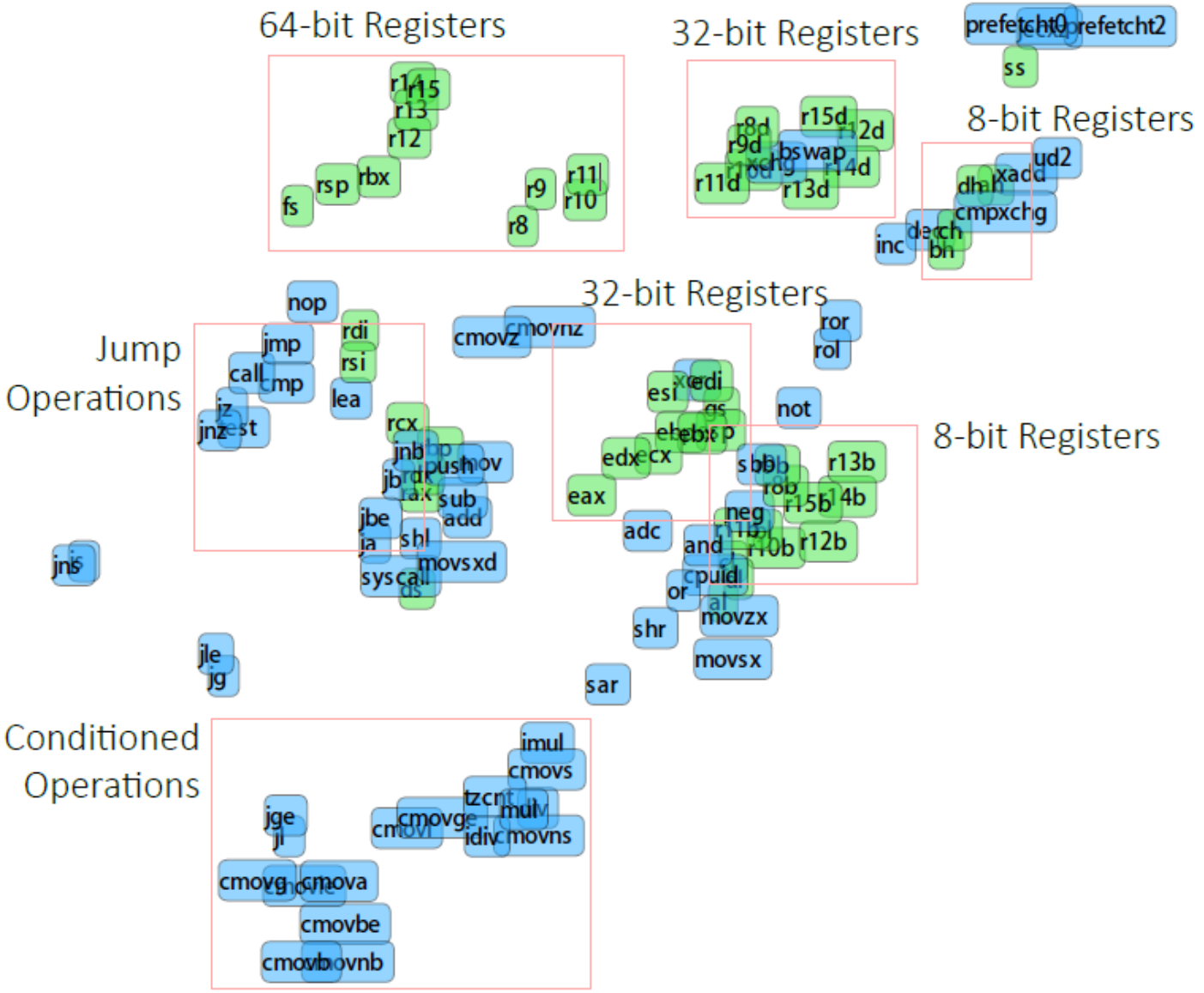
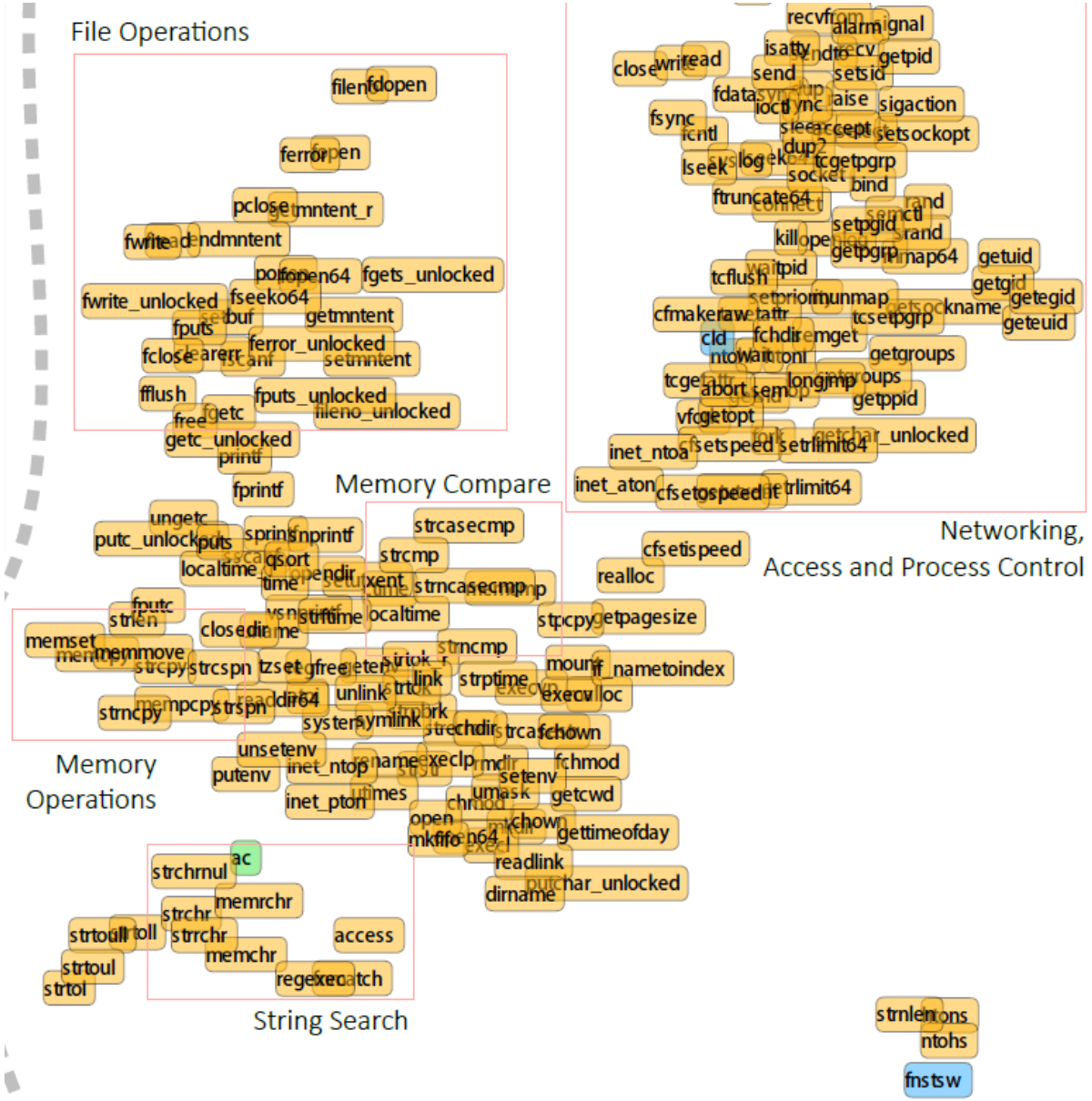


Figure 5: The proposed *Asm2Vec* neural network model for assembly code.

T-SNE Visualization



T-SNE Visualization



Evaluation (Quantitative)

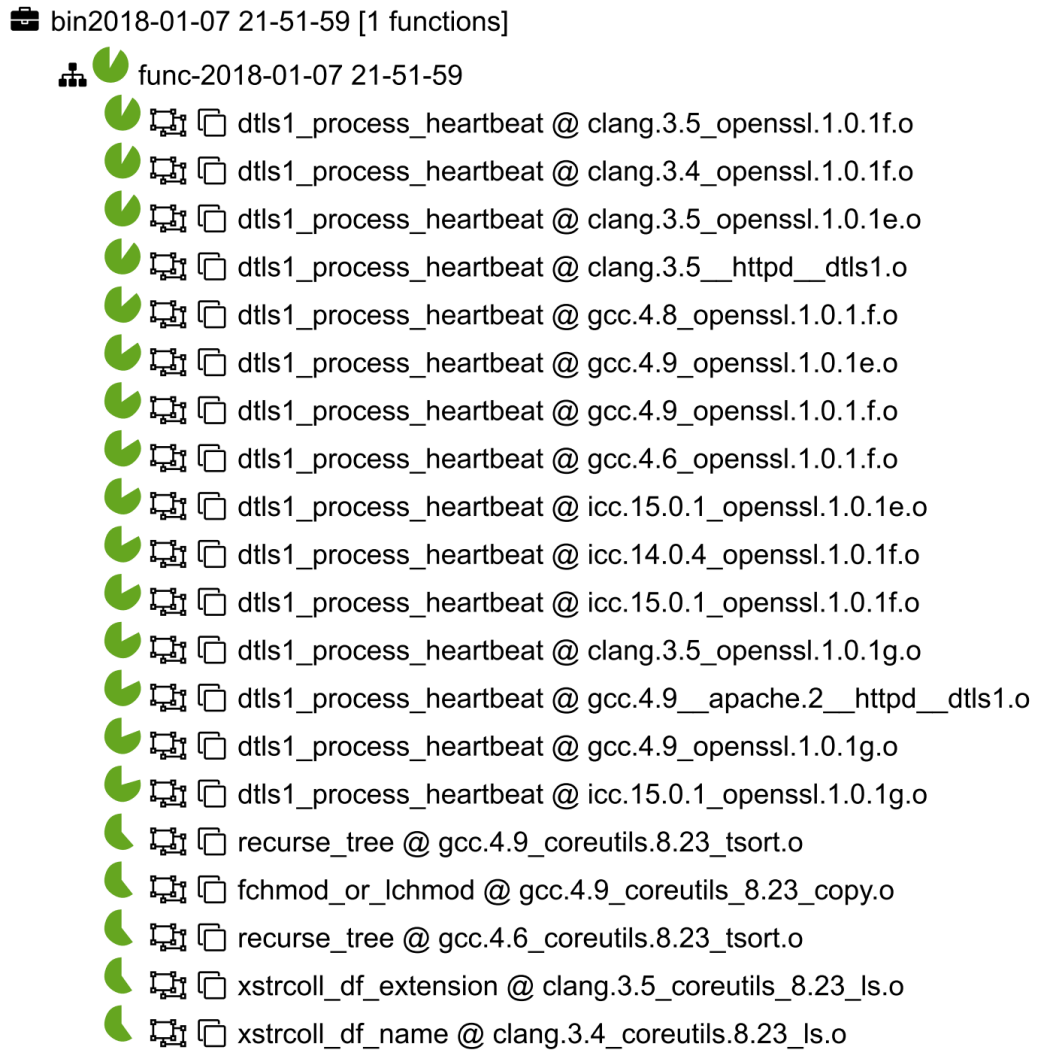
Compiler optimization O0 and O3												
Baselines	BusyBox	CoreUtils	Libgmp	ImageMagick	Libcurl	LibTomCrypt	OpenSSL	SQLite	zlib	PuTTYgen	Avg.	p
BinGo†	Ø	.317	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	.317	○
CACompare†	.844	Ø	Ø	**.893**	.794	Ø	**.795**	Ø	Ø	.717	.808	○
Composite	.013	.031	.019	.017	.004	.005	.007	.004	.036	.127	.026	●
Constant	.239	.128	.101	.610	.369	.258	.270	.182	.360	.439	.296	●
Graphlet	.017	.008	.049	.010	.023	.011	.009	.014	.029	.016	.019	●
Graphlet-C	.018	.020	.012	.022	.027	.001	.034	.012	.065	.102	.031	●
Graphlet-E	.021	.011	.075	.019	.017	.003	.018	.019	.051	.058	.029	●
MixedGram	.016	.033	.019	.018	.011	.005	.007	.006	.036	.116	.028	●
MixedGraph	.034	.028	.062	.024	.039	.015	.023	.030	.064	.097	.042	●
n-gram	.011	.029	.012	.021	.011	.010	.005	.003	.036	.129	.027	●
n-perm	.017	.029	.021	.021	.011	.006	.007	.005	.036	.129	.028	●
FuncSimSearch	.008	.019	.323	.039	.036	.030	.220	.011	.054	.040	.078	●
PV(DM/DBOW)	.745	.677	.760	.802	.792	.821	.759	.758	.713	.615	.744	●
Asm2Vec*	**.856**	**.781**	**.763**	.837	**.850**	**.921**	.792	**.776**	**.722**	**.788**	**.809**	○

Evaluation (Quantitative)

	O-LLVM - Bogus Control Flow Graph (BCF)					O-LLVM - Instruction Substitution (SUB)				
Baselines	Libgmp	ImageMagick	LibTomCrypt	OpenSSL	Avg.	Libgmp	ImageMagick	LibTomCrypt	OpenSSL	Avg.
Composite	.226	.224	.312	.246	.252	.620	.675	.600	.766	.665
Constant	.130	.592	.412	.318	.363	.173	.622	.492	.360	.412
Graphlet	.003	.005	.033	.007	.012	.198	.158	.411	.308	.269
Graphlet-C	.112	.118	.165	.124	.130	.626	.572	.539	.585	.581
Graphlet-E	.026	.011	.050	.014	.025	.454	.216	.286	.271	.307
MixedGram	.220	.234	.375	.303	.283	.585	.642	.563	.743	.633
MixedGraph	.011	.007	.049	.014	.020	.356	.325	.495	.488	.416
n-gram	.134	.134	.295	.195	.190	.466	.516	.513	.670	.541
n-perm	.233	.224	.374	.274	.276	.557	.624	.558	.736	.619
FuncSimSearch	.109	.022	.029	.027	.047	.685	.442	.699	.330	.539
PV(DM/DBOW)	.784	.870	.842	.768	.816	.935	.968	.964	.958	.956
Asm2Vec *	**.802**	**.920**	**.933**	**.883**	**.885**	**.940**	**.960**	**.981**	**.961**	**.961**
	O-LLVM - Control Flow Flattening (FLA)					O-LLVM - SUB+FLA+BCF				
Baselines	Libgmp	ImageMagick	LibTomCrypt	OpenSSL	Avg.	Libgmp	ImageMagick	LibTomCrypt	OpenSSL	Avg.
Composite	.138	.129	.052	.027	.086	.219	.226	.015	.009	.117
Constant	.105	.480	.215	.209	.252	.137	.591	.173	.159	.265
Graphlet	.000	.002	.000	.000	.000	.000	.005	.000	.000	.001
Graphlet-C	.003	.008	.000	.001	.003	.107	.124	.000	.000	.058
Graphlet-E	.001	.002	.000	.000	.001	.020	.012	.000	.000	.008
MixedGram	.148	.143	.075	.036	.101	.221	.234	.018	.010	.121
MixedGraph	.003	.003	.000	.000	.002	.006	.010	.000	.000	.004
n-gram	.095	.093	.059	.030	.069	.154	.144	.013	.007	.079
n-perm	.133	.126	.055	.033	.087	.224	.222	.018	.008	.118
FuncSimSearch	.095	.001	.004	.008	.027	.110	.025	.003	.008	.037
PV(DM/DBOW)	**.852**	**.938**	.786	.763	.835	.780	.873	.639	.595	.722
Asm2Vec *	.772	.920	**.890**	**.795**	**.844**	**.854**	**.880**	**.830**	**.690**	**.814**

Evaluation (Case Studies)

Vulnerability retrieval



		ESH [18]			Asm2Vec		
Vulnerability	CVE	FP	ROC	CROC	FP	ROC	CROC
Heartbleed	2014-0160	0	1	1	0	1	1
Shellshock	2014-6271	3	0.999	0.996	0	1	1
Venom	2015-3456	0	1	1	0	1	1
Clobberin' Time	2014-9295	19	0.993	0.956	0	1	1
Shellshock #2	2014-7169	0	1	1	0	1	1
ws-snmp	2011-0444	1	1	0.997	0	1	1
wget	2014-4877	0	1	1	0	1	1
ffmpeg	2015-6826	0	1	1	0	1	1

Evaluation (Case Studies)

Searching with Obfuscation Options in Tigress									
Name	Heartbleed	ShellshocK	Venom	Clobberin' Time	Shellshock #2	ws-snmp	wget	ffmpeg	avg.
CVE	2014-0160	2014-6271	2015-3456	2014-9295	2014-7169	2014-4877	2014-4877	2015-6826	
# of Positives (k)	15	9	6	10	3	7	3	7	
Encode Literal	100%	77.8%	100%	100%	100%	100%	100%	100%	97.2%
Virtualization	0%	0%	100%	20%	100%	0%	66.7%	0%	35.8%
JIT Execution	53.3%	0%	83.3%	30%	33.3%	0%	0%	100%	37.5%

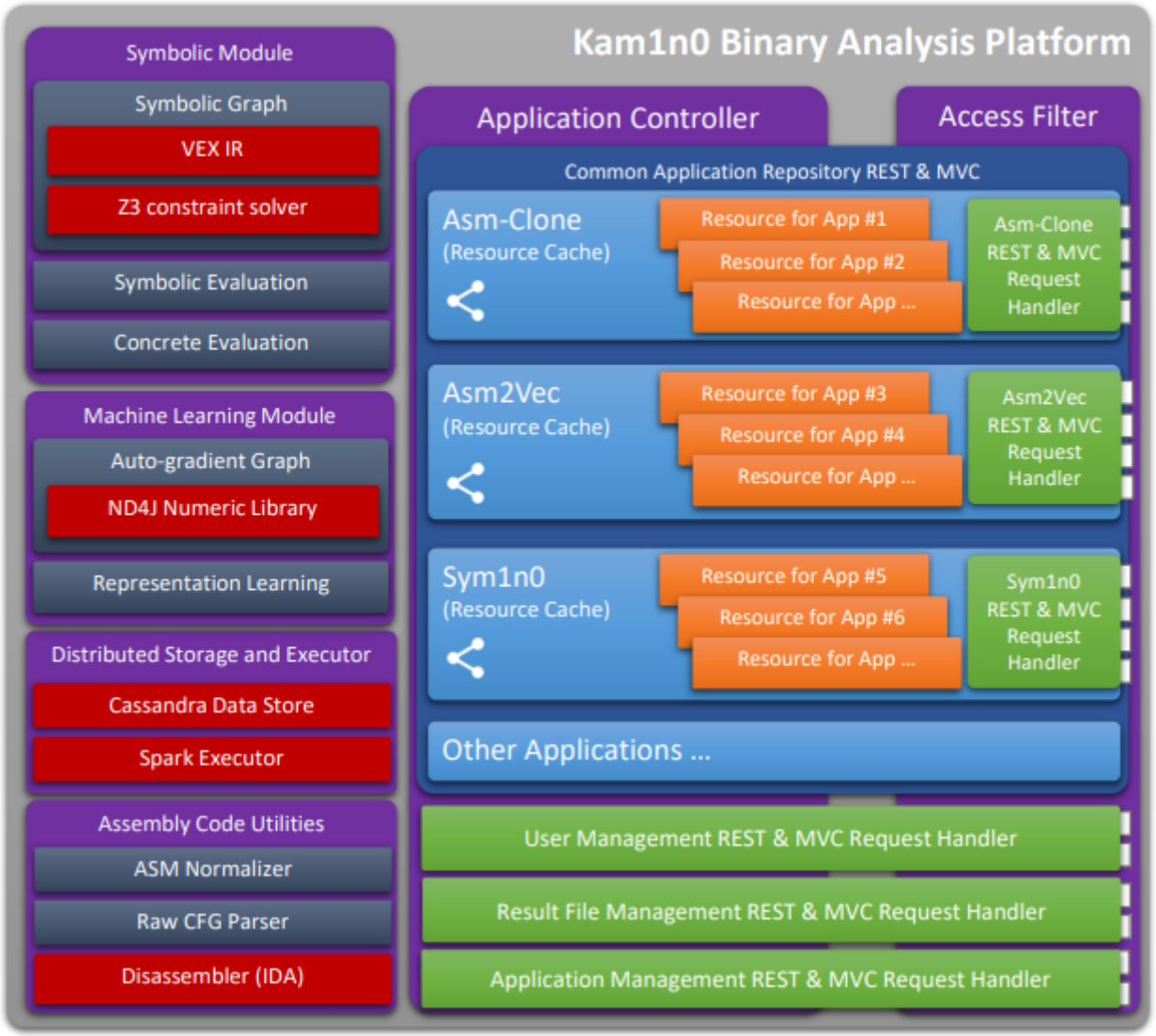
TABLE 4: True Positive Rate (TPR) of the top-k results searching the obfuscated vulnerable function against the dataset in [18]. k is chosen as the number of ground-truth clones in the dataset. For example, *Venom CVE 2015-3456-4877* has 6 variants in the dataset. By inspecting the top-6 results from *Asm2Vec* we recovered 100% (6/6) for the query with literals encoded, 100% (6/6) for the virtualized query, and 83.3% (5/6) for JIT-transformed query. After applying all the options at the same time, *Asm2Vec* cannot recover any true positives.

Asm2Vec (IEEE S&P19)

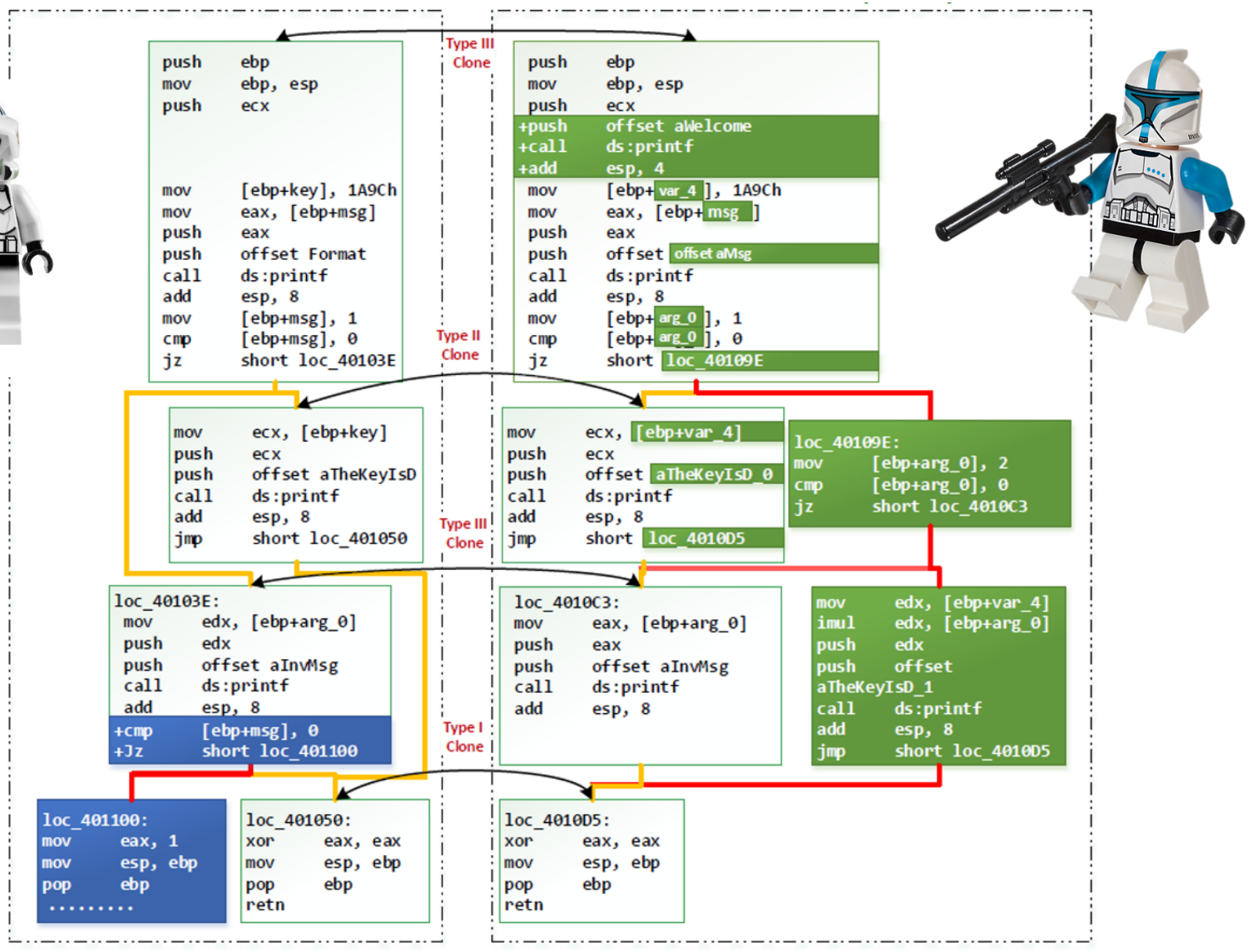
+ Against obfuscation and optimization.
+ Even better than the most recent dynamic approach.
+ Static approach: efficient and scalable.

- Binary differing (interpretability?)
- Static approach: cannot recognize jump table, etc.
- Assembly code come from **the same processor family**.

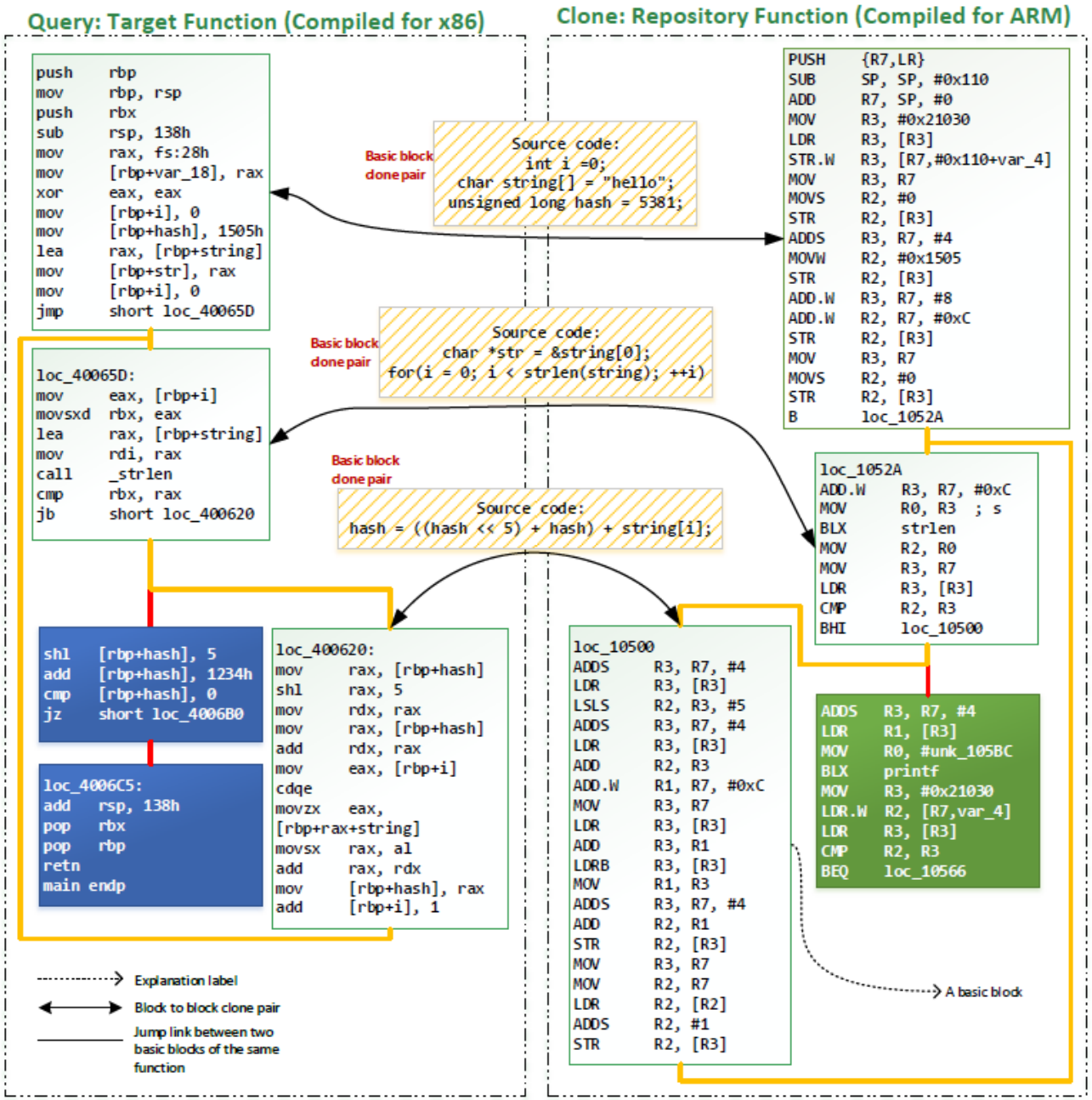
The Kam1n0 2.x Binary Analysis Platform



Subgraph clone



Sym1n0


Query: Target Function (Compiled for x86)
Clone: Repository Function (Compiled for ARM)
push rbp
mov rbp, rsp
push rbx
sub rsp, 138h
mov rax, fs:28h
mov [rbp+var_18], rax
xor eax, eax
mov [rbp+i], 0
mov [rbp+hash], 1505h
lea rax, [rbp+string]
mov [rbp+str], rax
mov [rbp+i], 0
jmp short loc_40065D
Basic block done pair
Source code:
int i =0;
char string[] = "hello";
unsigned long hash = 5381;
PUSH {R7,LR}
SUB SP, SP, #0x110
ADD R7, SP, #0
MOV R3, #0x21030
LDR R3, [R3]
STR.W R3, [R7,#0x110+var_4]
MOV R3, R7
MOVS R2, #0
STR R2, [R3]
ADDS R3, R7, #4
MOVW R2, #0x1505
STR R2, [R3]
ADD.W R3, R7, #8
ADD.W R2, R7, #0xC
STR R2, [R3]
MOV R3, R7
MOVS R2, #0
STR R2, [R3]
B loc_1052A
Basic block done pair
Source code:
char *str = &string[0];
for(i = 0; i < strlen(string); ++i)
loc_40065D:
mov eax, [rbp+i]
movsxd rbx, eax
lea rax, [rbp+string]
mov rdi, rax
call _strlen
cmp rbx, rax
jb short loc_400620
loc_1052A
ADD.W R3, R7, #0xC
MOV R0, R3 ; s
BLX strlen
MOV R2, R0
MOV R3, R7
LDR R3, [R3]
CMP R2, R3
BHI loc_10500
Basic block done pair
Source code:
hash = ((hash << 5) + hash) + string[i];
shl [rbp+hash], 5
add [rbp+hash], 1234h
cmp [rbp+hash], 0
jz short loc_4006B0
loc_400620:
mov rax, [rbp+hash]
shl rax, 5
mov rdx, rax
mov rax, [rbp+hash]
add rdx, rax
mov eax, [rbp+i]
cdqe
movzx eax, [rbp+rax+string]
movsx rax, al
add rax, rdx
mov [rbp+hash], rax
add [rbp+i], 1
loc_10500
ADDS R3, R7, #4
LDR R3, [R3]
LSLS R2, R3, #5
ADDS R3, R7, #4
LDR R3, [R3]
ADD R2, R3
ADD.W R1, R7, #0xC
MOV R3, R7
LDR R3, [R3]
ADD R3, R1
LDRB R3, [R3]
MOV R1, R3
ADDS R3, R7, #4
ADD R2, R1
STR R2, [R3]
MOV R3, R7
MOV R2, R7
LDR R2, [R2]
ADDS R2, #1
STR R2, [R3]
ADDS R3, R7, #4
LDR R1, [R3]
MOV R0, #unk_105BC
BLX printf
MOV R3, #0x21030
LDR.W R2, [R7,var_4]
LDR R3, [R3]
CMP R2, R3
BEQ loc_10566
loc_4006C5:
add rsp, 138h
pop rbx
pop rbp
retn
main endp
A basic block
Explanation label
Block to block clone pair
Jump link between two basic blocks of the same function

Thank you. Questions?