(Un)Supervised Learning for Malware with quickSpan

Thomas Given-Wilson

Cisco- Ecole Polytechnique Symposium,
Paris, France

10th April 2018

Goals

Supervised Learning:

- Detect existing and new variants of malware
- Use behaviour (not YARA), here SCDGs
- Middle line-of-defence

Unsupervised Learning:

- Group unknown programs
- Use behaviour, again SCDGs
- Aid analysis

SCDG Toy Example

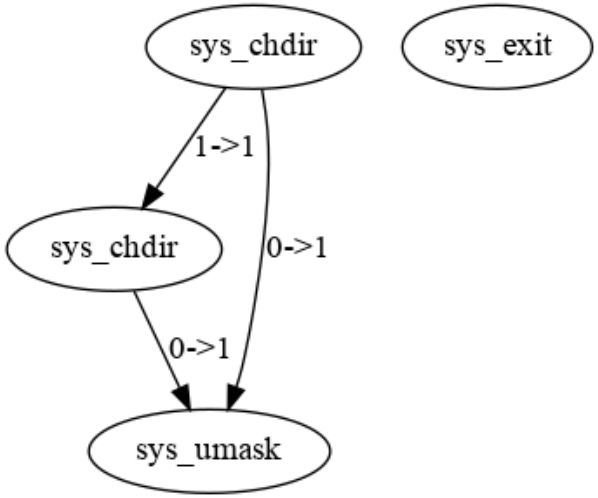
1 - Binary

```
1  section .data
2    newdir db "/tmp"
3
4  section .text
5    global _start
6
7  _start:
8    mov rdi, newdir    ; 1st arg
9    mov rax, 80        ; sys_chdir
10   syscall
11   push rax           ; saved for later
12   mov rdi, newdir    ; 1st arg
13   mov rax, 80        ; sys_chdir
14   syscall
15   pop rdi            ; pop the return value of 1st call to chdir
16   add rdi, rax       ; add return value of second chdir call as umask
17   mov rax, 95        ; sys_umask
18   syscall            ;
19   mov rdi, 0
20   mov rax, 60        ; exit (0)
21   syscall
```

2 - Trace


```
{
  "name": "sys_chdir",
  "arguments": [ {"value": [6291692]} ], "return": "syscall_stub_1_64"
},
{
  "name": "sys_chdir",
  "arguments": [ {"value": [6291692]} ], "return": "syscall_stub_2_64",
},
{
  "name": "sys_umask",
  "arguments": [ {"value": ["syscall_stub_1_64", "syscall_stub_2_64"]} ],
  "return": "syscall_stub_3_64"
},
{
  "name": "sys_exit",
  "arguments": [ {"value": [0]} ]
}
```


3 - Graph

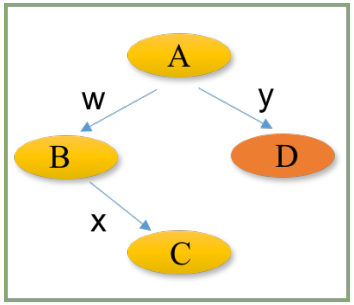


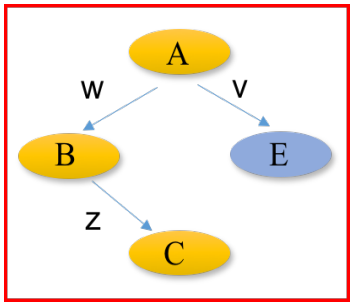
Common Subgraphs: gSpan Algorithm

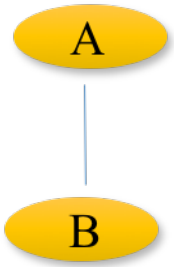
Find common sub-graphs:

- Builds set of (sub-)graphs
- That meet "support" (% of graph set), here support=1.0
- Example set of graphs

  (Note: ignores edge direction and edge labels here.)







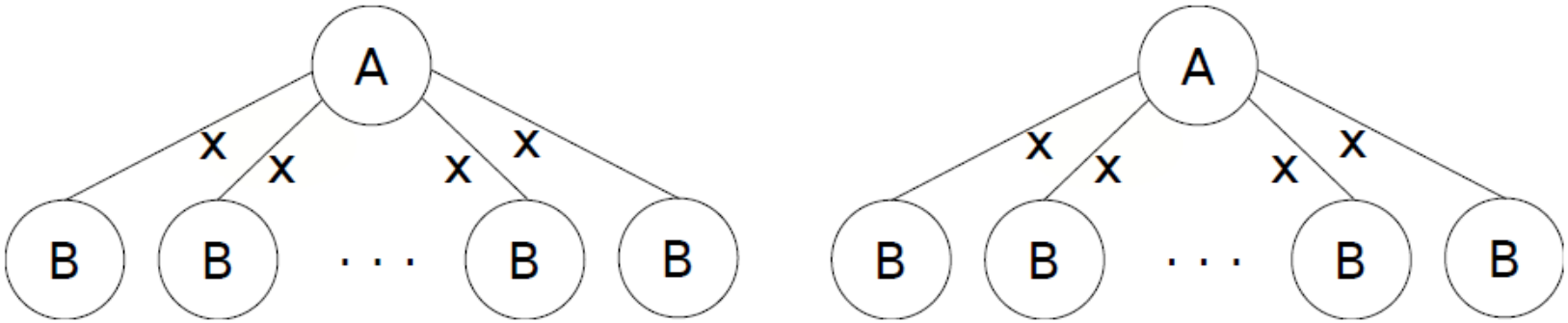
gSpan (Implementation) Problems

Algorithm:

- Scales poorly with graph size
- Vulnerable to pathological cases

Implementation (gBolt [1]):

- Ignores edge direction
- Runs till completion/failure
- Memory exhaustion crashes



1 - https://github.com/Jokeren/DataMining-gSpan

quickSpan: Improved gSpan Implementation

Extended and improved parallel version of gBolt:

- *Edge direction*
- *(Safe) Time out*
- *Memory safety (safe failure)*
- *Minimum and maximum output size*
- *Fast exit on "success"*
- *Thread control*
- Removes duplicate sub-graphs
- Incremental output
- "Best" output
- …

quickSpan: Comparison

Performs very well overall!

Data sets where quickSpan is not clear winner:

- AIDS
- Cancer

Note: All data sets are public and gathered from graph mining papers. (except Pathological).

Data set name		Support	gBolt	GLP	quickSpan	quickSpan[1]	SF
BrainNet	ADHD	0.10	0.27	1.77	**0.11**	0.13	–
		0.15	0.27	0.14	0.10	**0.03**	–
		0.20	0.28	0.07	0.08	**0.02**	–
		0.50	0.13	0.07	0.07	**0.02**	–
	Hyper/Impulsive	0.10	14.12	–	**11.48**	117.54	–
		0.15	**0.67**	54.06	**0.67**	1.44	–
		0.20	0.39	2.47	**0.19**	0.44	–
		0.50	0.19	0.11	0.07	**0.06**	–
	Gender	0.10	0.34	11.16	**0.16**	0.52	–
		0.15	0.30	1.35	**0.10**	0.12	–
		0.20	0.30	0.10	0.11	**0.05**	–
		0.50	0.16	0.05	0.08	**0.03**	–
Chemical 340		0.10	0.37	1.06	**0.18**	0.21	0.23
		0.15	0.34	0.68	0.14	**0.12**	0.14
		0.20	0.39	0.37	0.13	**0.09**	0.13
		0.50	0.28	0.18	0.10	**0.06**	**0.06**
Mutagen		0.10	0.40	0.70	**0.23**	0.28	–
		0.15	0.40	0.40	**0.16**	0.24	–
		0.20	0.37	0.32	**0.15**	0.19	–
		0.50	0.22	0.13	**0.12**	0.15	–
NCI	AIDS	0.10	**5.73**	90.96	5.75	15.15	–
		0.15	**5.21**	48.11	5.36	11.39	–
		0.20	4.84	34.45	**4.78**	9.75	–
		0.50	**4.19**	4.33	**4.19**	5.16	–
	AIDS Active	0.10	0.59	18.62	**0.36**	1.20	2.53
		0.15	0.45	4.70	**0.28**	0.58	0.96
		0.20	0.43	1.52	**0.25**	0.26	0.53
		0.50	0.32	0.29	0.18	**0.11**	0.12
	AIDS + Cancer	0.10	3.34	57.28	**3.32**	8.37	–
		0.15	3.10	29.51	**3.05**	6.32	–
		0.20	**2.80**	20.66	2.86	5.26	–
		0.50	2.55	6.78	**2.47**	3.13	–
	Cancer	0.10	**19.05**	363.91	20.44	89.04	–
		0.15	**15.11**	170.93	15.50	51.63	–
		0.20	**13.60**	108.72	14.04	38.19	–
		0.50	**10.42**	29.90	10.77	14.99	–
Social	DBLP	0.10	0.84	1.51	0.80	**0.77**	–
		0.15	0.84	1.48	0.85	**0.78**	–
		0.20	0.85	1.51	0.82	**0.75**	–
		0.50	0.82	1.50	0.89	**0.75**	–
	Twitter	0.10	**1.61**	3.01	**1.61**	1.62	–
		0.15	1.66	3.06	1.61	**1.52**	–
		0.20	1.63	3.06	1.60	**1.49**	–
		0.50	1.70	3.04	1.78	**1.48**	–
Pathological 2		1.00	!	!	84.54(24.16)	**81.21(23.38)**	–

Supervised Classification Using gSpan

Learning:

1. Given graphs from malware family Fi:
   obtain canonical representation gi =
   gSpan(Fi, learning_parameters)

Classification:

1. Given sample s
   For each gi:
   calculate similarity using sim(s,gi) =
   max_size(gSpan({s,gi}, classification_parameters))
2. Classification result r = max (sim(s,gi))
3. If r > threshold then classify as “malware”
   otherwise classify as “cleanware” …(or “unknown”)

Classification Example

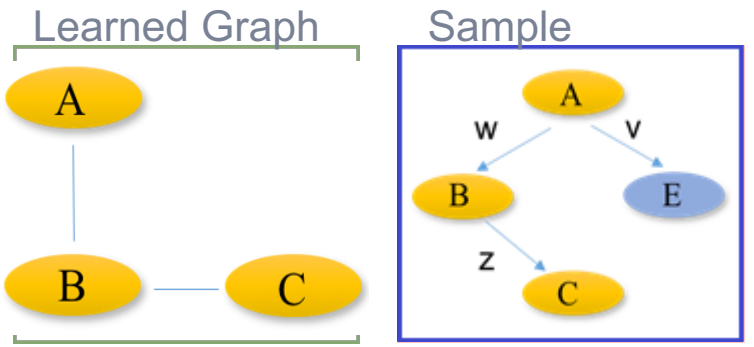
Learning:

- Find common subgraphs
- Take largest 1

Classification:

- Find common sub-graphs
- Test threshold (0.25)?

  size(AB)/size(ABC) = 2/3 > 0.25 => true, classify as “malware”



Mirai Case Study: Data Set

Mirai:

- Collected from IoT honeypot (6 April to 14 August 2017)
- Selected only X86 32-bit ELF

Cleanware:

- Static-get distribution[1]

After (attempted) SCDG extraction:

- 504 Mirai SCDGs
- 942 clean SCDGs

1 - http://s.minos.io/

Mirai Case Study: Experiments

Best parameters:

- Undirected
- Learning time 2500
- Support 0.7
- Number of graphs 50
- Threshold 0.32

	Mirai samples	Clean samples
Detected Mirai	98.09 (97.12%)	0 (0%)
Detected Clean	2.91 (2.88%)	101 (100%)

Accuracy	Precision	$F_{0.5}$ score
98.56%	100.00%	99.41%

Conclusions:

- Zero false positives!
- Low false negatives.
- Reasonable classification time (1.56s)

Compared with YARA:

- Comparable/better $F_{0.5}$ score (YARA: 84.38 to 98.61%)
- Higher cost

Unsupervised Learning: Clustering

- Exploit gSpan for clustering graphs
  - Base algorithm by Seeland et al.
  - Allows overlap of clusters
- Apply this to SCDGs
  - Cluster by common behaviour
  - Should correlate with families
- Aim is to cluster suspected malware samples
  - Complement to classification
  - Weaker correlation => show new families/evolution

Seeland Clustering Algorithm

Find clusters in size ordered graphs using gSpan:

For each graph g in the sequence:

    clustered = false

    For each cluster C in clusters:

        If gspan(C+g,1.0,*min_size*) != {} then

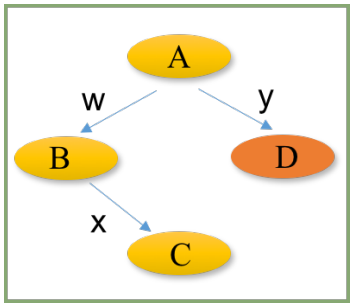
            C += g

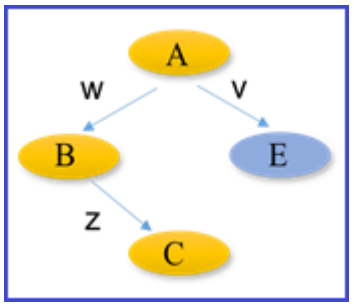
            clustered = true

    If clustered == false

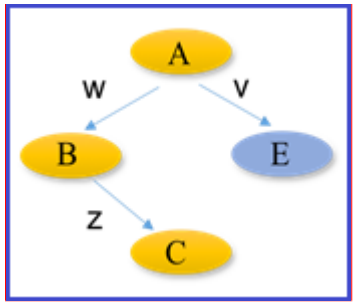
        clusters.append({g})

Cluster 1

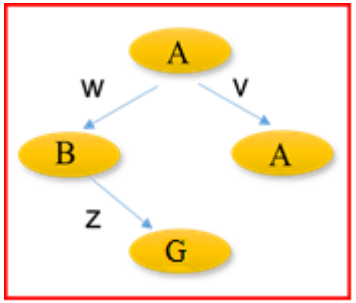




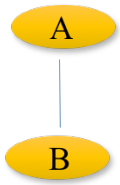
New graph..



Cluster 2



Run gSpan with Cluster 1 …



not common enough, not added. clustered.

Practical Clustering

The min_size parameter is unclear, possibilities:

- Fixed/absolute (base algorithm):
  - User must specify/know the optimal size
  - Size is chosen uniformly for all graph sizes
- Percentage of graph:
  - User must specify/know the percentage that is optimal
  - Percentage automatically adjusts for graph size
- Computation based:
  - User doesn't need to specify
  - Automatically adjusts for graph size

The sorting by size is sub-optimal:

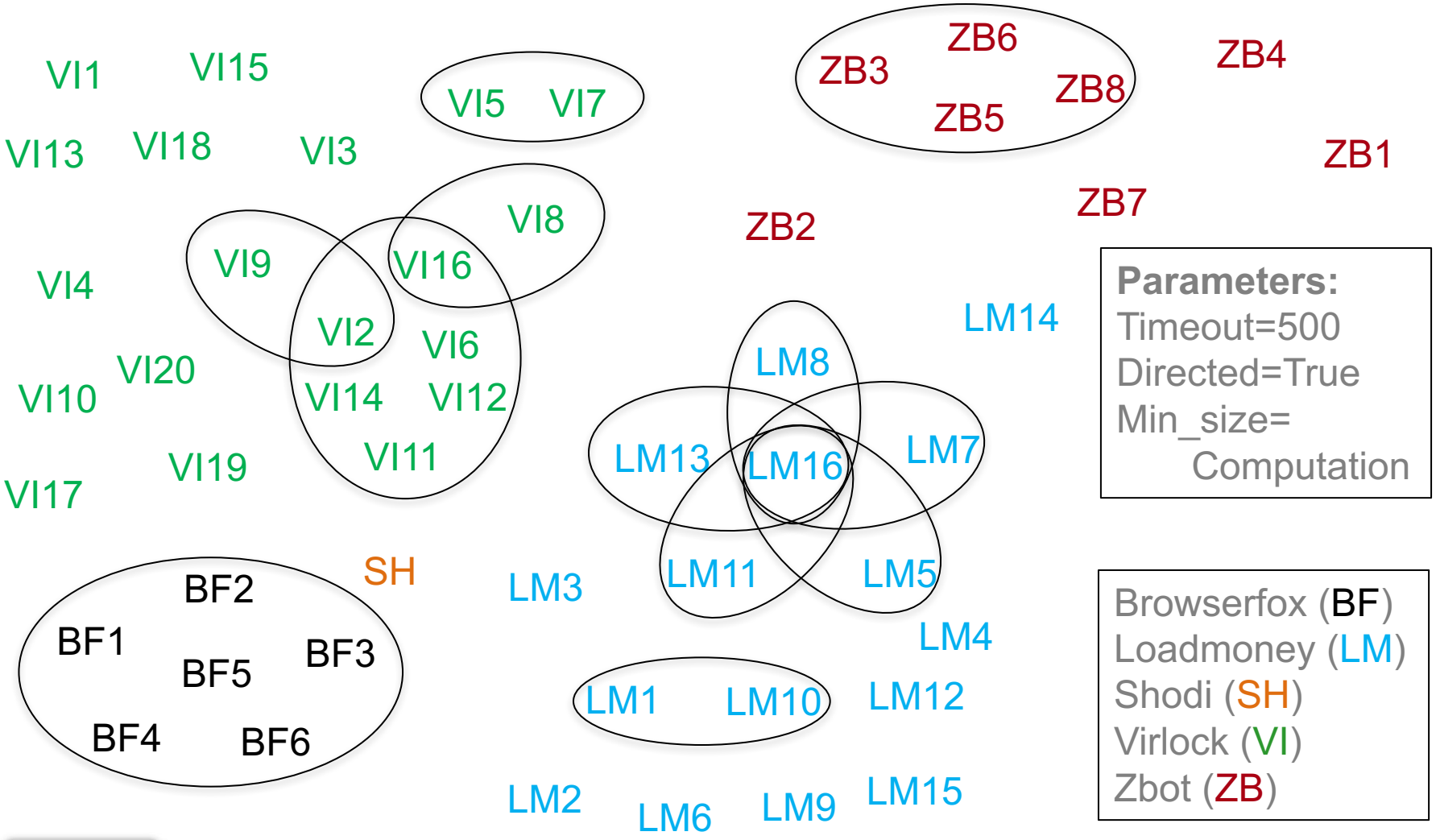
- Instead we use random ordering

Initial Experiment Configuration

Small sample experiment with:

- 5 families (1-20 graphs each):
  - Browserfox 6 graphs
  - Loadmoney 16 graphs
  - Shodi 1 graph
  - Virlock 20 graphs
  - Zbot 8 graphs
- quickSpan parameters:
  - Timeout
  - Direction
  - min_size

Experimental Results



Conclusions

gSpan:

- Good graph mining algorithm
- quickSpan adds performance and quality of life features
- Useful for both supervised and unsupervised learning

Supervised Learning:

- Good results on Mirai
- Comparable with YARA
- Highly tuneable

Unsupervised Learning:

- Good results for correctness of clustering
- Highly tuneable