Universitatea
Politehnica
București

Facultatea de
Automatică și
Calculatoare

Catedra de
Calculatoare

Malsharp: Malicious Application Analysis

Bachelor Thesis, July 2013

Autor

Cristian Condurache

Conducător științific

As.dr.ing. Laura Gheorghe

Motivation & Goals

- Why?
  - Popularity of Linux based OSs
  - Use in embedded systems

Motivation & Goals

- Why?
  - Popularity of Linux based OSs
  - Use in embedded systems
- How?
  - Malsharp: malicious behavior pattern mining

Malware Detection

- Signature-based
  - Problem: fails to detect new malware, obfuscation
- Behavior-based
  - Problem: behavior patterns require manual identification

Malspec Mining Algorithm

- Input: a malware sample and a set of benign programs

Malspec Mining Algorithm

- Input: a malware sample and a set of benign programs
- Output: a malicious behavior pattern

Malspec Mining Algorithm

- Input: a malware sample and a set of benign programs
- Output: a malicious behavior pattern
- Creates graphs for sample malware and benign programs
  - A node represents a system call
  - An edge is an argument dependency

Malspec Mining Algorithm

- Input: a malware sample and a set of benign programs
- Output: a malicious behavior pattern
- Creates graphs for sample malware and benign programs
  - A node represents a system call
  - An edge is an argument dependency
- Computes malware specifications as “difference” between graphs
  - Maximal common subgraph algorithm
  - Complement graph
  - Minimal transversal

Dependency Graph (1)

- Initial nodes

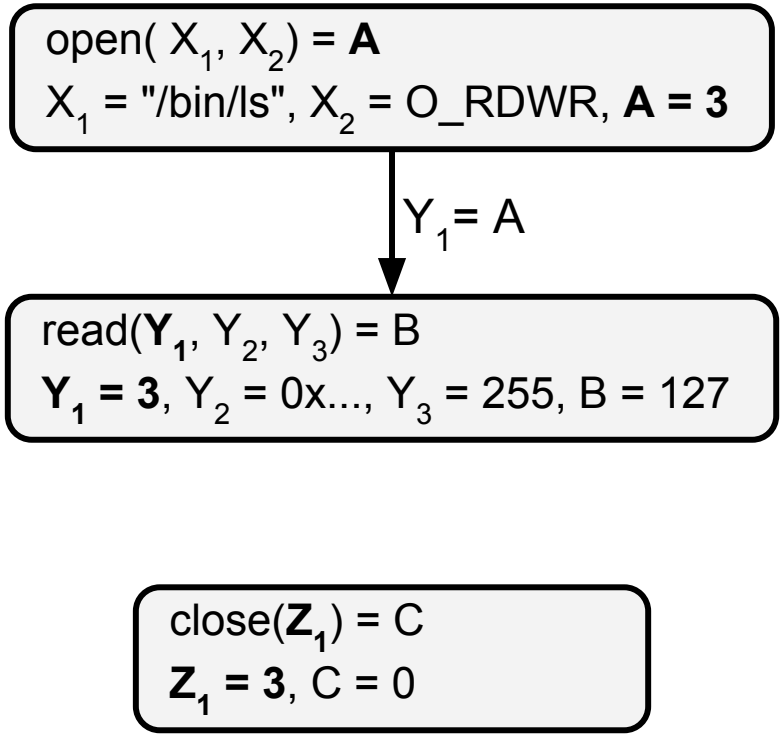
open(X_1, X_2) = **A**
X_1 = "/bin/ls", X_2 = O_RDWR, **A = 3**

read(**Y_1**, Y_2, Y_3) = B
Y_1 = 3, Y_2 = 0x..., Y_3 = 255, B = 127

close(**Z_1**) = C
Z_1 = 3, C = 0

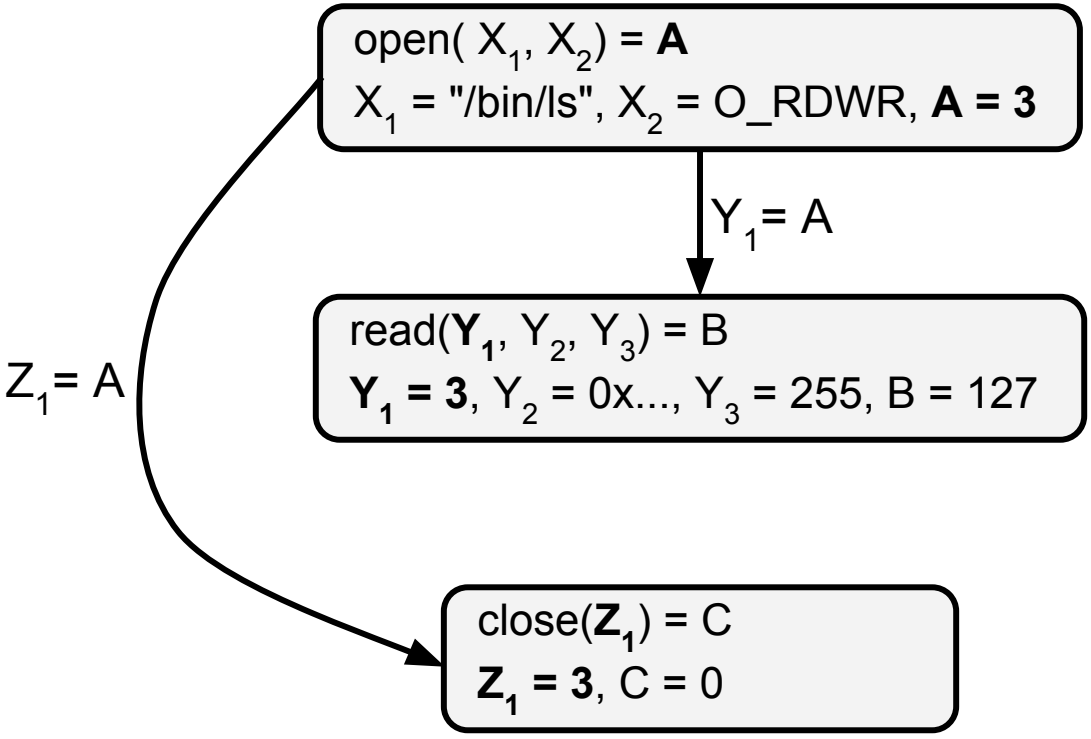
Dependency Graph (2)

- Adding a dependency edge between open and read

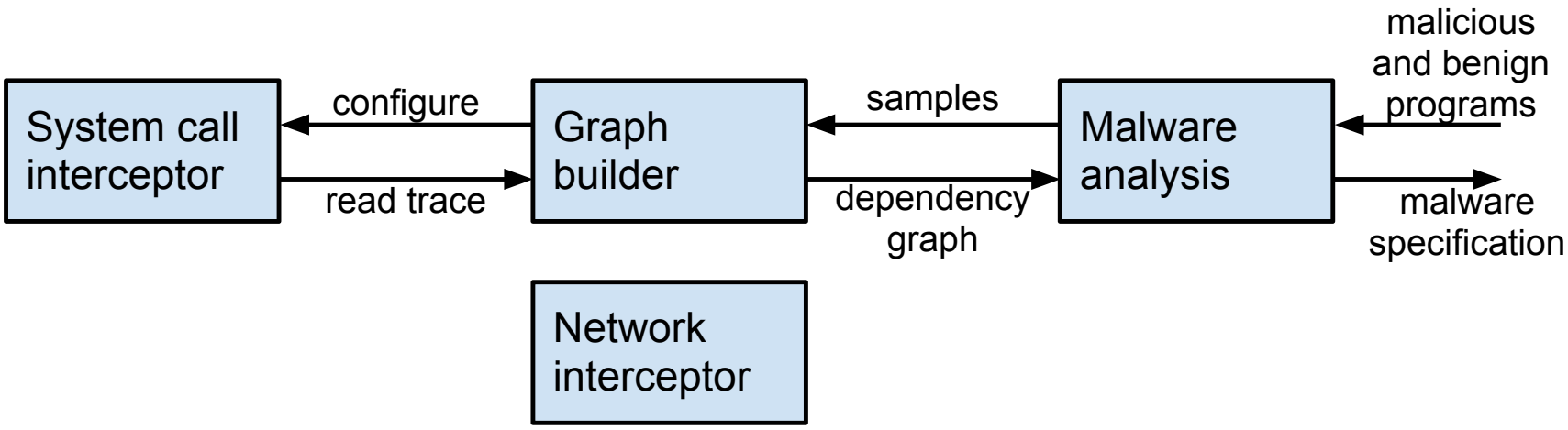


Dependency Graph (3)

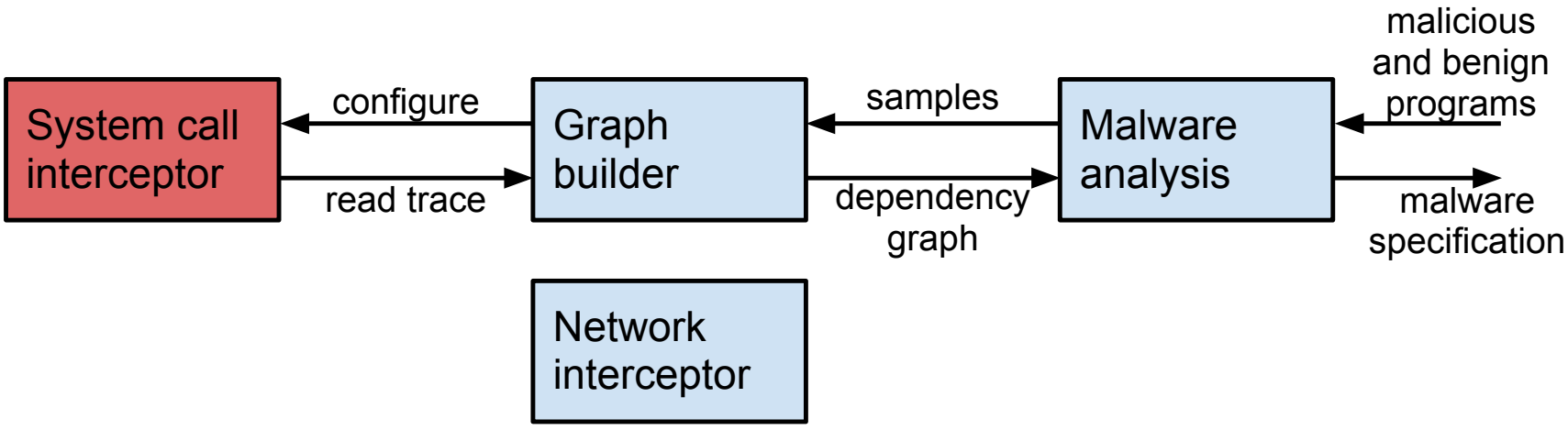
- Adding a dependency edge between open and close



Malsharp Architecture

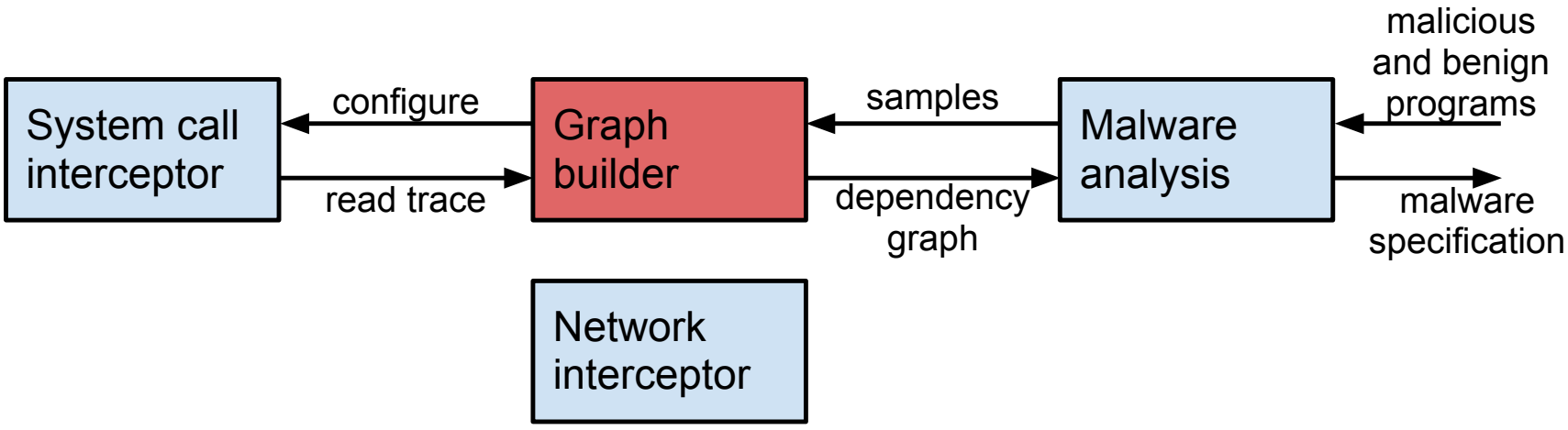


Implementation (1)



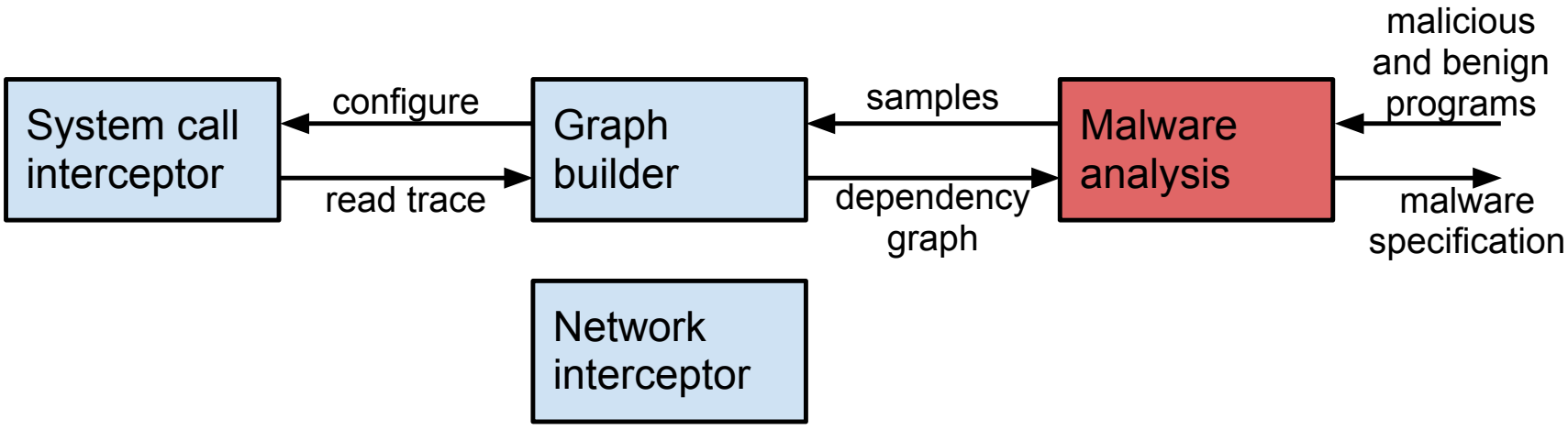
- System Call Interceptor Driver (SCID)
  - Logs execution trace for a process
  - Kernel module, registers by using miscdevice
  - Controlled via the ioctl system call

Implementation (2)

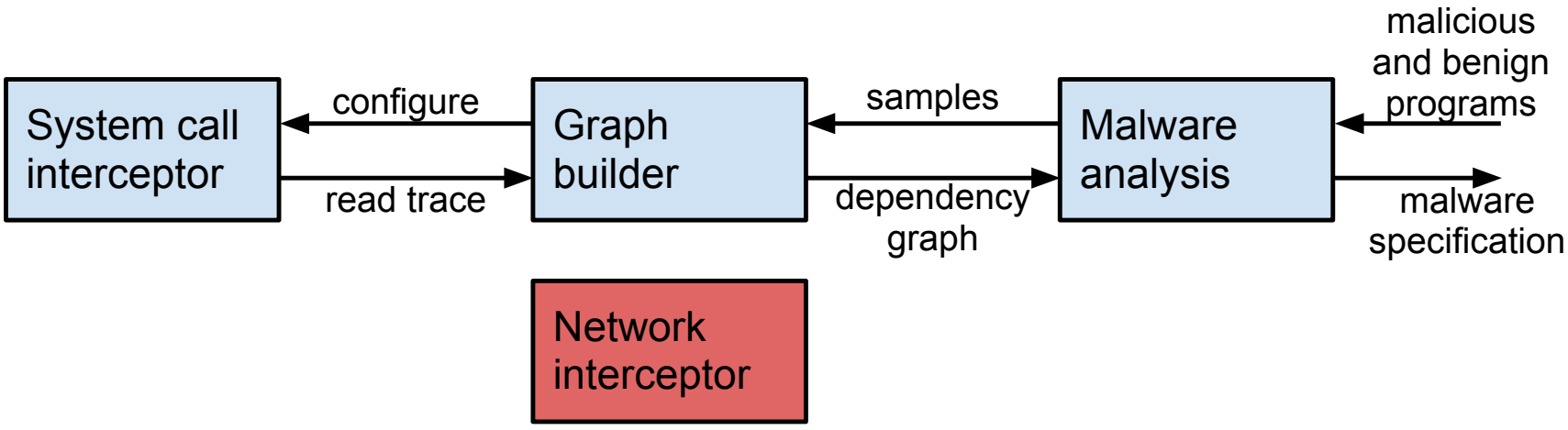


- Graph Builder
  - Runs each program
  - Reads execution traces from SCID
  - Finds argument dependencies
  - Node aggregation: a read of 1000 bytes is equivalent to 1000 individual reads of 1 byte

Implementation (3)



- Malware Analysis
  - Uses the graph builder for each program
  - Applies the malspec mining algorithm



- Network Interceptor (NI)
  - Used for monitoring malware networking activity
  - Netfilter hooks to monitor traffic
  - Can be configured to monitor specific protocols

Evaluation (1)

- Virtual machine, snapshots
- Bridged network access
- Revert to snapshot before each test

Evaluation (1)

- Virtual machine, snapshots
- Bridged network access
- Revert to snapshot before each test

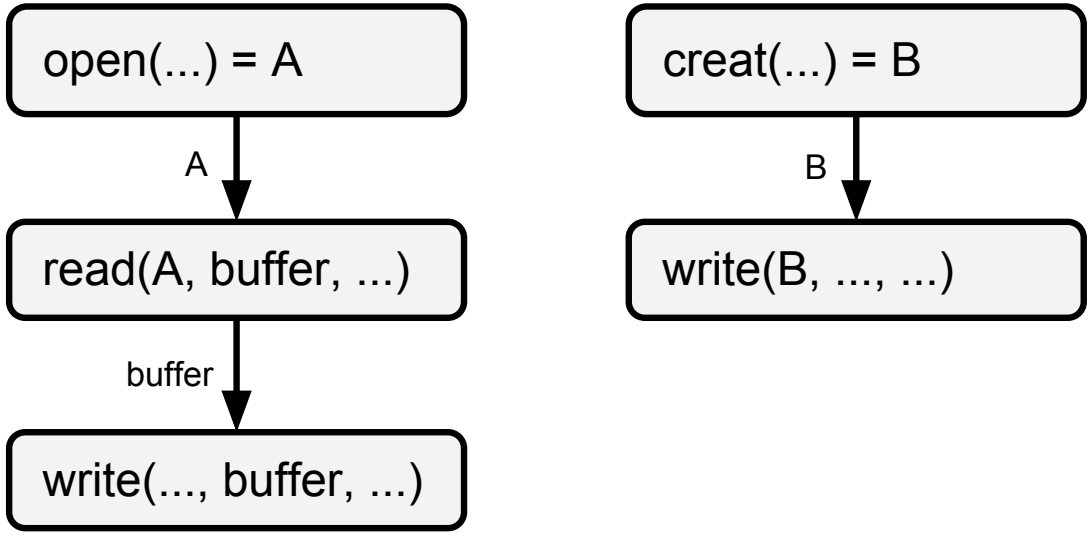
- Monitored two sets of system calls

Evaluation (1)

- Virtual machine, snapshots
- Bridged network access
- Revert to snapshot before each test
- Monitored two sets of system calls
- A set of 20 well-known malware samples
  - Viruses: Virus.Linux.Rike.1627, Virus.Linux.Osf.8759
  - Backdoors: Backdoor.Linux.CGI, Backdoor.Linux.Phobi.1

Evaluation (2)

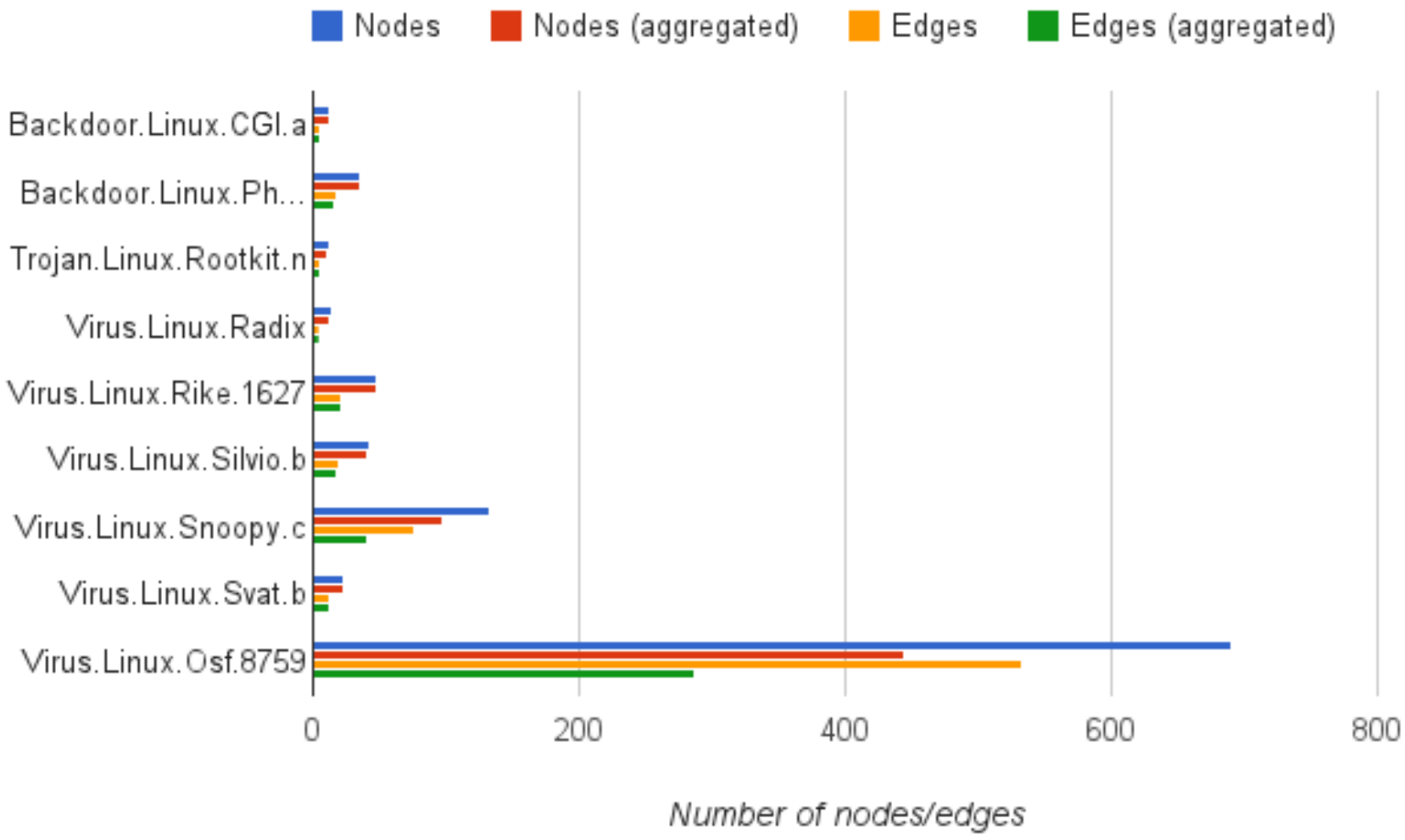
- Execution traces, graphs successfully built
- Malware patterns identified, 3-5 nodes
- Results for Virus.Linux.Radix



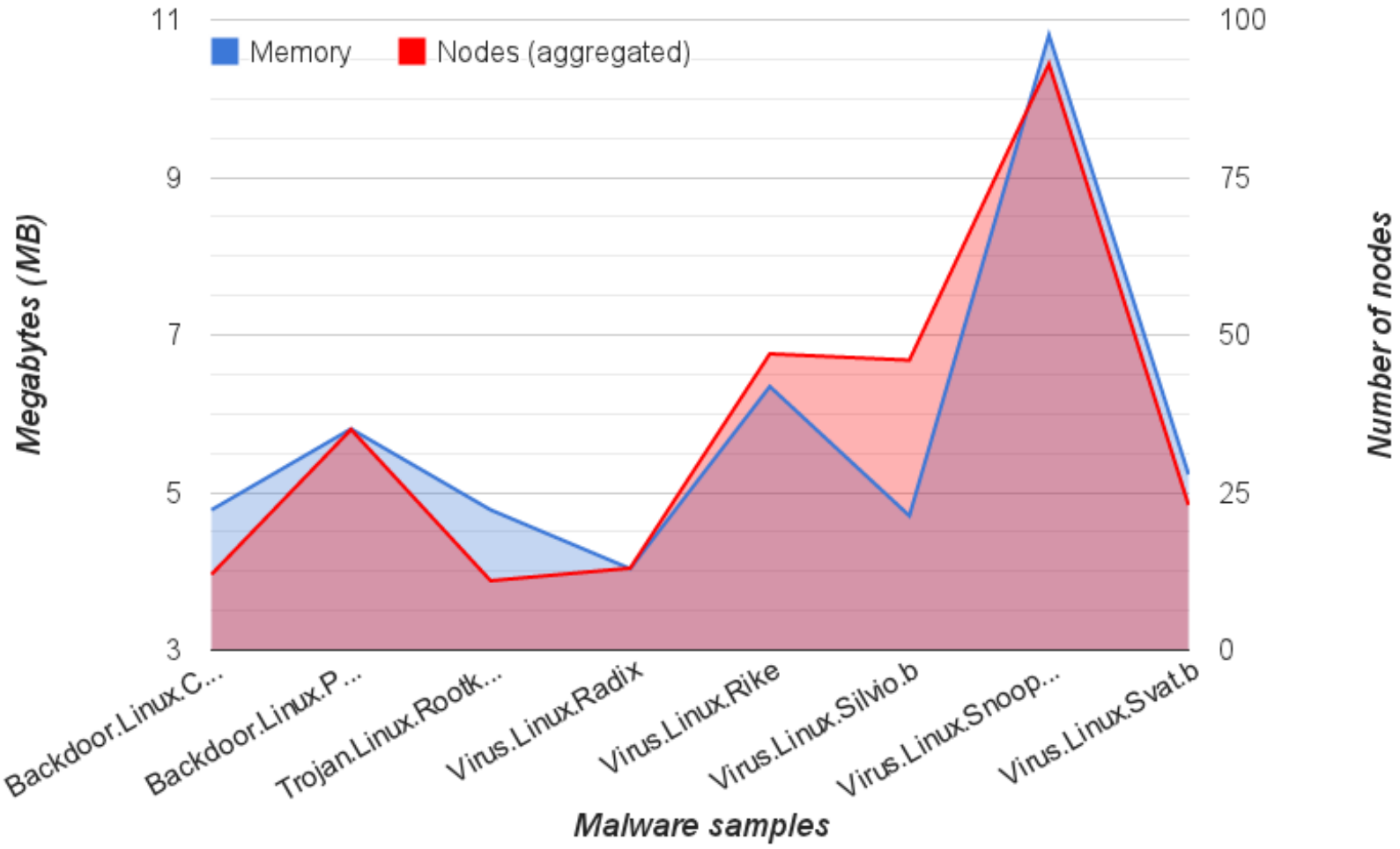
Evaluation (3)

Sample	Nodes	Edges	Malspec	Time	Observed behavior
Backdoor.Linux.CGI.a	13	5	1	0.827s	open, read, close
Backdoor.Linux.Phobi.1	35	16	1	0.641s	open, read, close
Trojan.Linux.Rootkit.n	14	5	1	1.044s	open, read, close
Virus.Linux.Radix	22	6	1	0.673s	open, read, write; creat, write
Virus.Linux.Svat.b	25	12	0	3.495s	replaces stdio.h

Node aggregation results



Memory usage



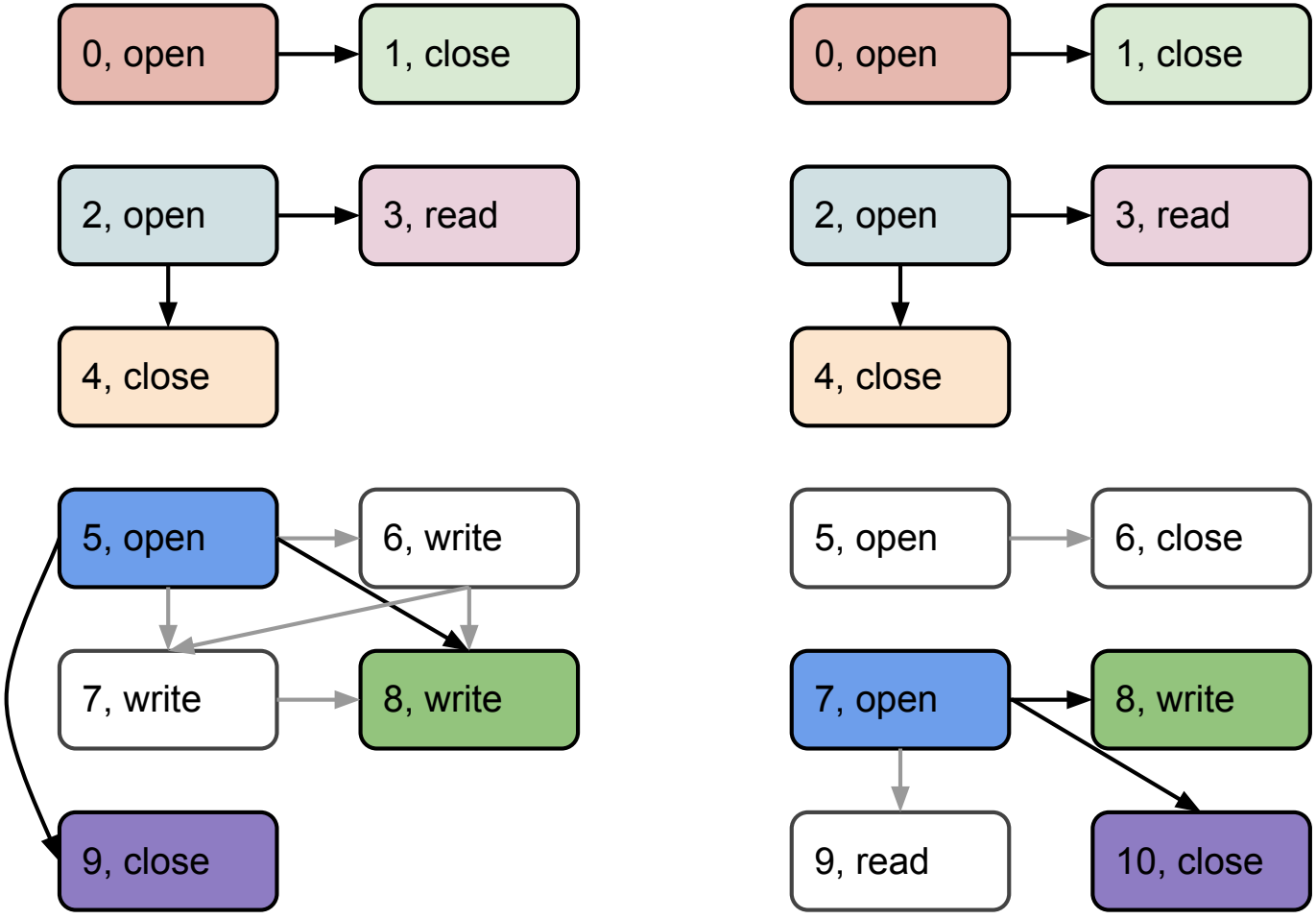
Conclusions

- Proof of concept for a Linux malware behavior miner
- Analysis detected malicious patterns
- Node aggregation successfully reduced total number of nodes
- Possible future improvements:
  - Additional pruning: node ordering strategies
  - Adding other types of dependency edges

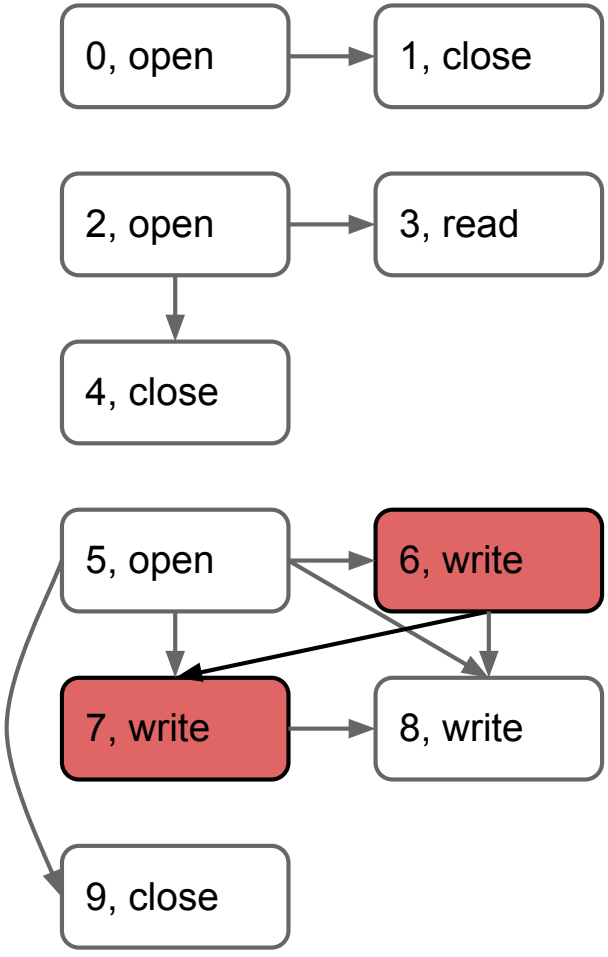
Test programs

	program_test	diff_test
1	open(...) = fd1	open(...) = fd1
2	close(fd1)	close(fd1)
3	open(...) = fd2	open(...) = fd2
4	read(fd2, ...)	read(fd2, ...)
5	close(fd2)	close(fd2)
6	open(...) = fd3	open(...) = fd3
7	write(fd3, ...)	close(fd3)
8	write(fd3, ...)	open(...) = fd4
9	write(fd3, ...)	write(fd4, ...)
10	close(fd3)	read(fd4, ...)
11	-	close(fd4)

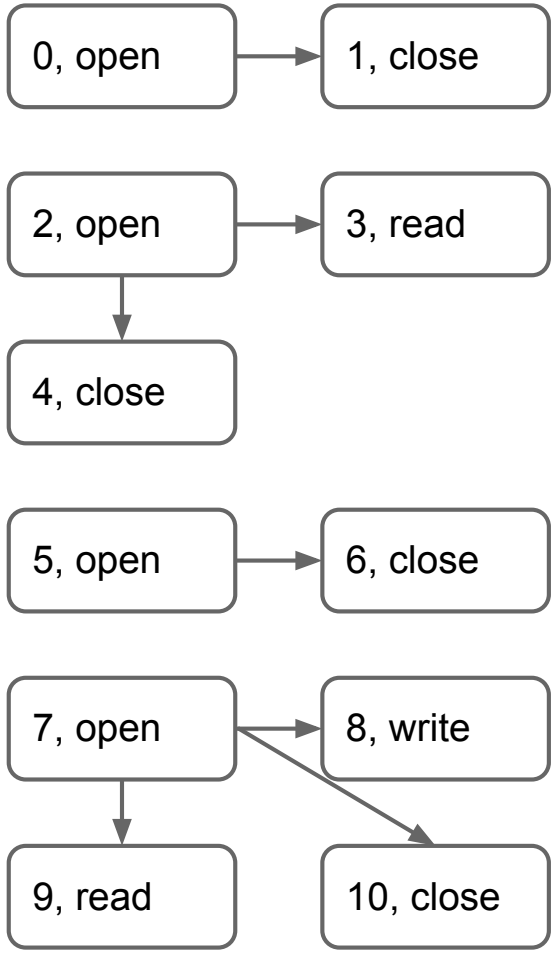
Maximal common edge set



Complement and minimal transversal



(a)



(b)