Author: Benjamin Selyem

System Call Sandboxing

Responsible Professor: Alexios Voulimeneas

1. Introduction & Background

- **System call:** Talk to hardware through kernel (Figure 1)
- **Sandboxing:** Restrict system calls to minimal required set (Figure 2)
- **Problem:** Which calls to block and which ones to allow?
- **Solution:** Analyse applications, find out which calls are needed
- **Gap:** Static vs Dynamic analysis & Single vs Multi phase model (Figure 3)

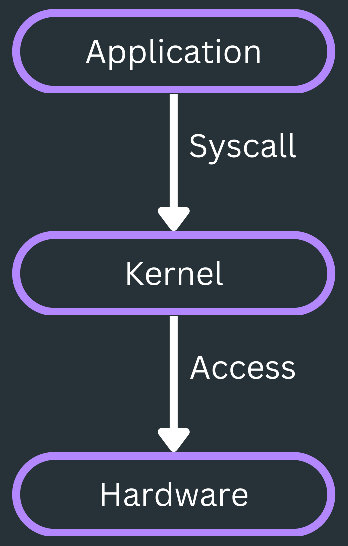


Figure 1: Regular syscall flow

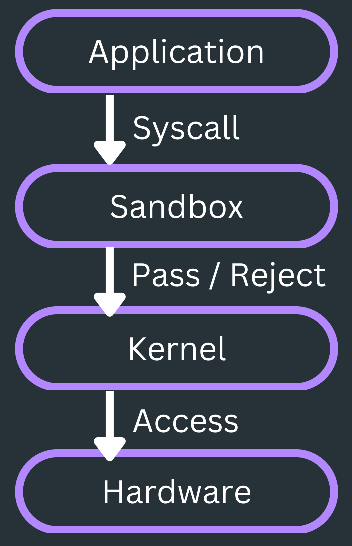


Figure 2: Sandboxed flow

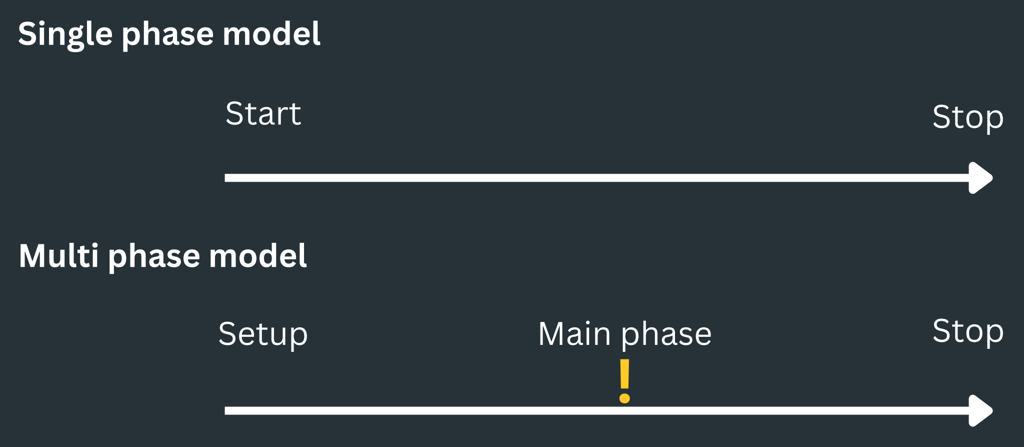
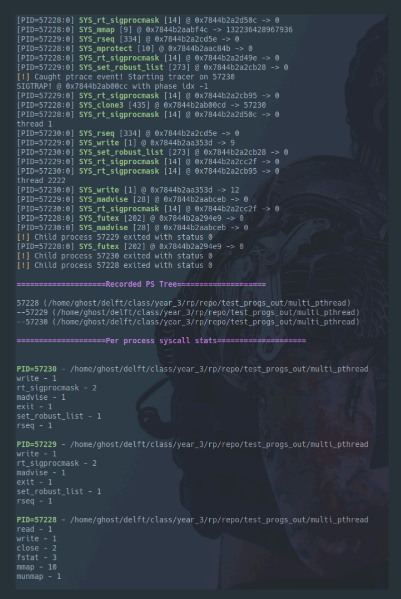


Figure 3: Single phase & multi phase model



```
[PID=57228:0] SYS_rt_sigprocmask [14] @ 0x7844b2a2d50c -> 0
[PID=57228:0] SYS_mmap [9] @ 0x7844b2aabf4c -> 132236428967936
[PID=57229:0] SYS_rseq [334] @ 0x7844b2a2cd5e -> 0
[PID=57228:0] SYS_mprotect [10] @ 0x7844b2aac84b -> 0
[PID=57228:0] SYS_rt_sigprocmask [14] @ 0x7844b2a2d49e -> 0
[PID=57229:0] SYS_set_robust_list [273] @ 0x7844b2a2cb28 -> 0
[!] Caught ptrace event! Starting tracer on 57230
SIGTRAP! @ 0x7844b2ab00cc with phase idx -1
[PID=57229:0] SYS_rt_sigprocmask [14] @ 0x7844b2a2cb95 -> 0
[PID=57228:0] SYS_clone3 [435] @ 0x7844b2ab00cd -> 57230
[PID=57228:0] SYS_rt_sigprocmask [14] @ 0x7844b2a2d50c -> 0
thread 1
[PID=57230:0] SYS_rseq [334] @ 0x7844b2a2cd5e -> 0
[PID=57229:0] SYS_write [1] @ 0x7844b2aa353d -> 9
[PID=57230:0] SYS_set_robust_list [273] @ 0x7844b2a2cb28 -> 0
[PID=57229:0] SYS_rt_sigprocmask [14] @ 0x7844b2a2cc2f -> 0
[PID=57230:0] SYS_rt_sigprocmask [14] @ 0x7844b2a2cb95 -> 0
thread 2222
[PID=57230:0] SYS_write [1] @ 0x7844b2aa353d -> 12
[PID=57229:0] SYS_madvise [28] @ 0x7844b2aabceb -> 0
[PID=57230:0] SYS_rt_sigprocmask [14] @ 0x7844b2a2cc2f -> 0
[PID=57228:0] SYS_futex [202] @ 0x7844b2a294e9 -> 0
[PID=57230:0] SYS_madvise [28] @ 0x7844b2aabceb -> 0
[!] Child process 57229 exited with status 0
[PID=57228:0] SYS_futex [202] @ 0x7844b2a294e9 -> 0
[!] Child process 57230 exited with status 0
[!] Child process 57228 exited with status 0

====================Recorded PS Tree====================

57228 (/home/ghost/delft/class/year_3/rp/repo/test_progs_out/multi_pthread)
--57229 (/home/ghost/delft/class/year_3/rp/repo/test_progs_out/multi_pthread)
--57230 (/home/ghost/delft/class/year_3/rp/repo/test_progs_out/multi_pthread)

====================Per process syscall stats====================


PID=57230 - /home/ghost/delft/class/year_3/rp/repo/test_progs_out/multi_pthread
write - 1
rt_sigprocmask - 2
madvise - 1
exit - 1
set_robust_list - 1
rseq - 1

PID=57229 - /home/ghost/delft/class/year_3/rp/repo/test_progs_out/multi_pthread
write - 1
rt_sigprocmask - 2
madvise - 1
exit - 1
set_robust_list - 1
rseq - 1

PID=57228 - /home/ghost/delft/class/year_3/rp/repo/test_progs_out/multi_pthread
read - 1
write - 1
close - 2
fstat - 3
mmap - 10
munmap - 1
```

Figure 4: The dynamic tracer

2. Research questions & Contributions

1. How can dynamic analysis method be used to identify the used system calls?
2. What is the runtime and accuracy of single phase model static analysis tools?
3. What is the runtime and accuracy of multi phase model static analysis tools?

- Dynamic analysis tool
- Analysis of various static analysis solutions
- Comparison of dynamic vs static and single phase vs multi phase approaches

3. Dynamic tracer

- Gather list of system calls used, using ptrace
- Traces multiple processes & threads
- Records process structure and system calls (Figure 4)
- Supports multi phase tracing
- Requires exploration of many program states

4. Experiment setup

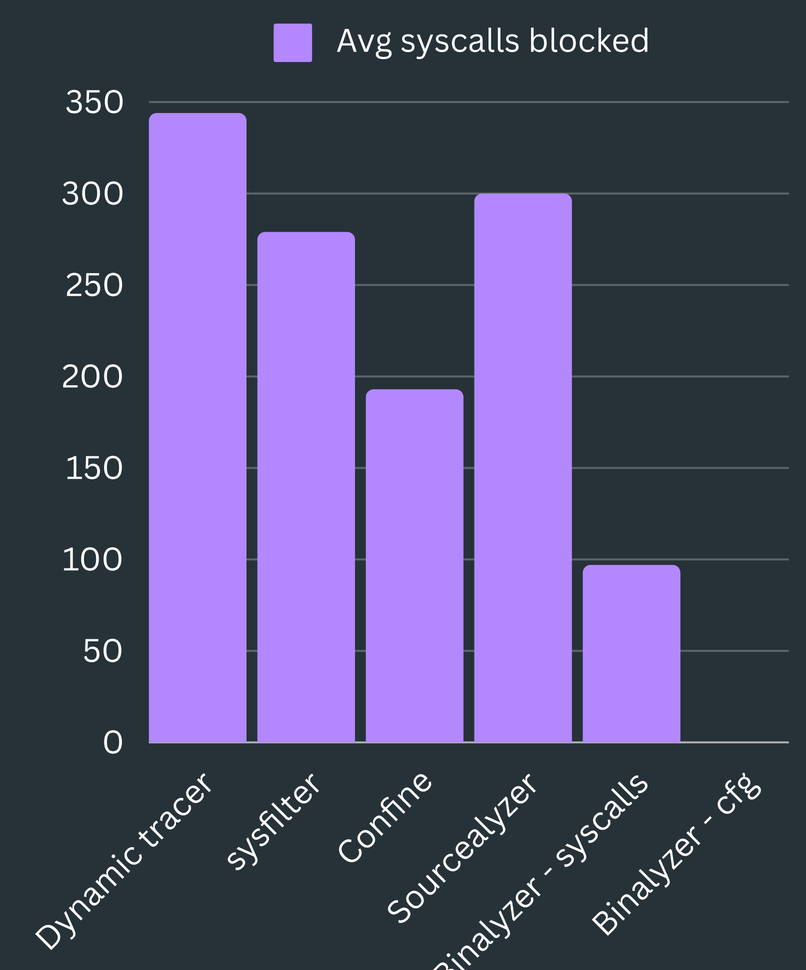
Test programs: ls (v8.28), sqlite3 (v3.22), redis (v4.0.9)

Environment: Ubuntu 18.04 Docker container

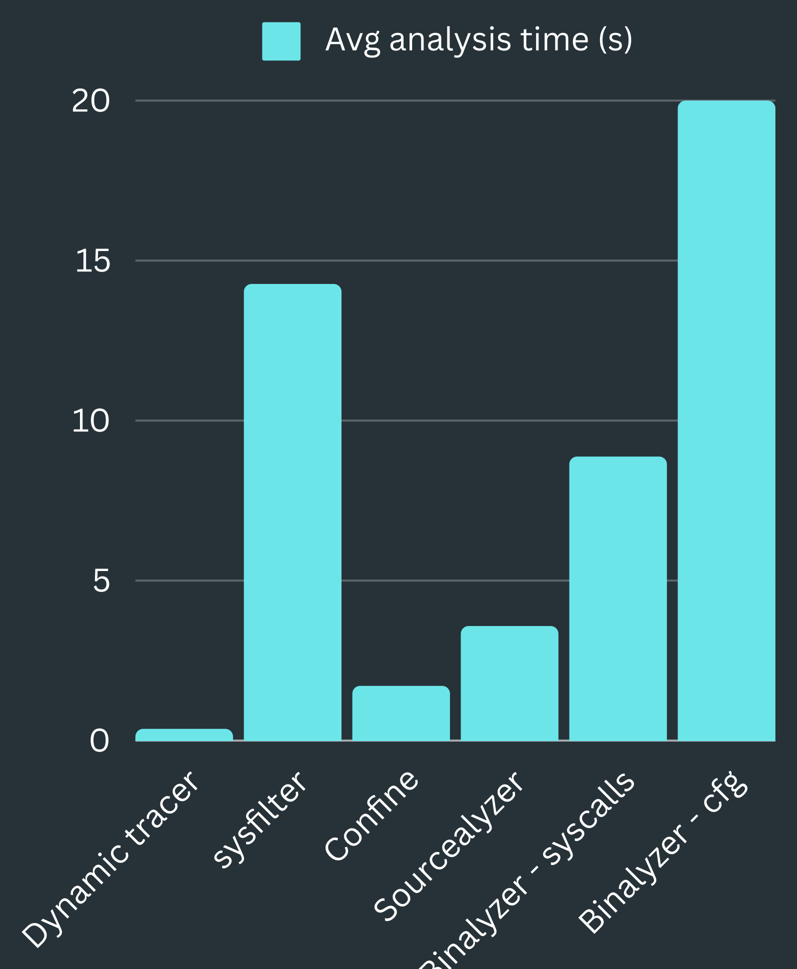
Analysis tools: Chestnut [1], Confine [4], temporal-specialization [3], sysfilter [2]

Measurements: Runtime, Accuracy

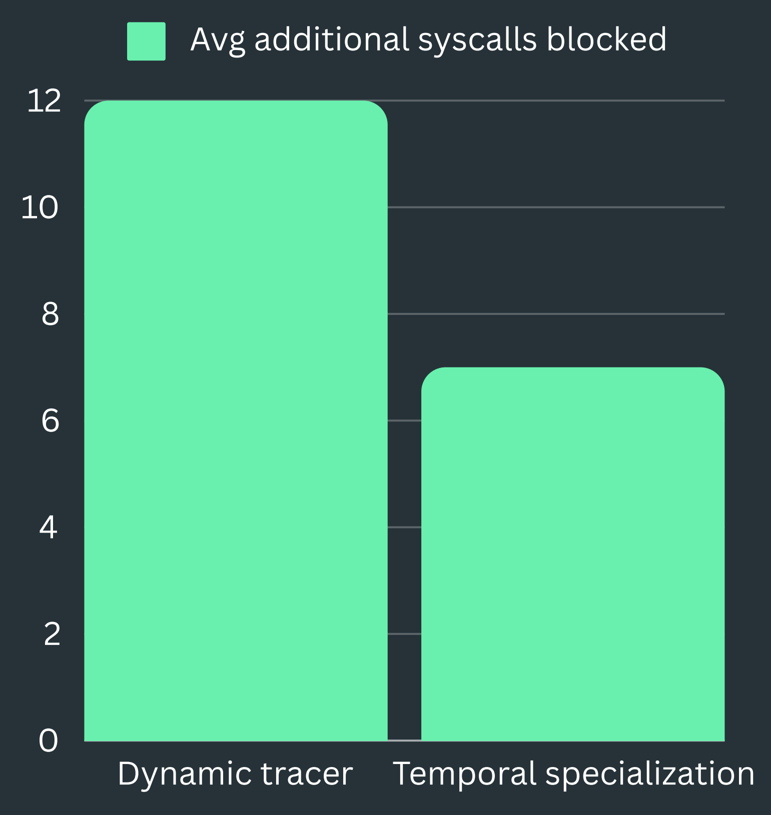
5. Results & Discussion



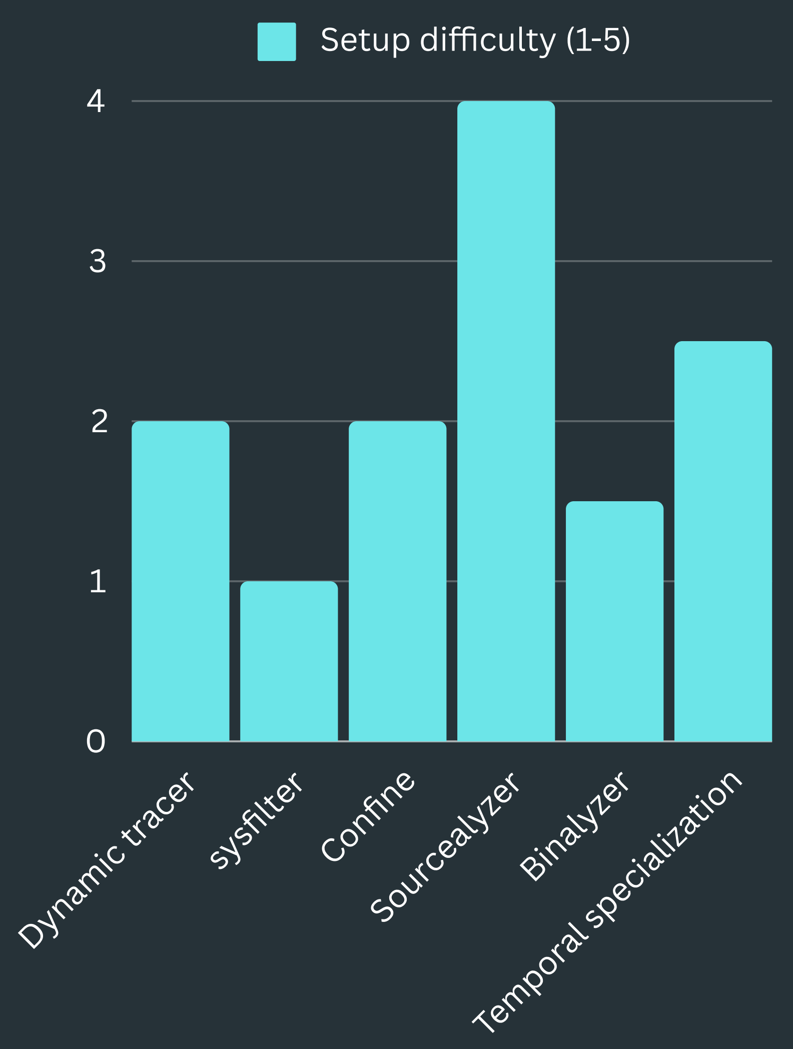
Dynamic tracer blocks **14%** more syscalls, than Sourecalyzer on average



Dynamic tracer is **4.5x** faster than Confine on average



Multi phase analysis blocks **11** more syscalls during the second phase than single phase analysis on average



Sourcealyzer is hardest to set up due to little setup instructions and a lot of compiling

6. Conclusion

Dynamic analysis

⊕ Allows for custom usage profile

⊕ Fast analysis times

⊖ Requires extensive test cases

ⓘ Second step analysis to reduce amount of incorrectly blocked system calls

Static analysis

⊕ Good amount of blocked system calls, with slight underestimation

⊖ Slower analysis time as programs scale

⊖ Difficult setup and maintainability

ⓘ Pre-computing CFG and parallelism

Multi phase model

⊕ Better security through more fine-grained filters

⊖ Need to determine transition point manually

ⓘ Instruction level transition points

ⓘ Multiple transition points

Possible future work

ⓘ Investigate if programs lose any functions after applying the filters

ⓘ Expand the set of tested programs

References

[1] Claudio Canella, Mario Werner, Daniel Gruss, and Michael Schwarz. Automating seccomp filter generation for linux applications. In Proceedings of the 2021 on Cloud Computing Security Workshop, CCSW '21, page 139–151, New York, NY, USA, 2021. Association for Computing Machinery.
[2] Nicholas DeMarinis, Kent Williams-King, Di Jin, Rodrigo Fonseca, and Vasileios P. Kemerlis. sysfilter: Automated system call filtering for commodity software. In International Symposium on Recent Advances in Intrusion Detection, 2020.
[3] Seyedhamed Ghavamnia, Tapti Palit, Shachee Mishra, and Michalis Polychronakis. Temporal system call specialization for attack surface reduction. In Proceedings of the 29th USENIX Conference on Security Symposium, SEC'20, USA, 2020. USENIX Association.
[4] Seyedhamed Ghavamnia, Tapti Palit, Azzedine Benameur, and Michalis Polychronakis. Confine: Automated system call policy generation for container attack surface reduction. In International Symposium on Recent Advances in Intrusion Detection, 2020