Automating Seccomp Filter Generation for Linux Applications

Claudio Canella[1], **Mario Werner**[1], **Daniel Gruss**[1], **Michael Schwarz**[2]

[1]Graz University of Technology [2]CISPA Helmholtz Center for Information Security

Motivation

- Memory safety vulnerabilties are common

Motivation

- Memory safety vulnerabilties are common
- Sandboxing helps in limiting their impact

Motivation

- Memory safety vulnerabilties are common
- Sandboxing helps in limiting their impact
- Linux seccomp: works but hard to do

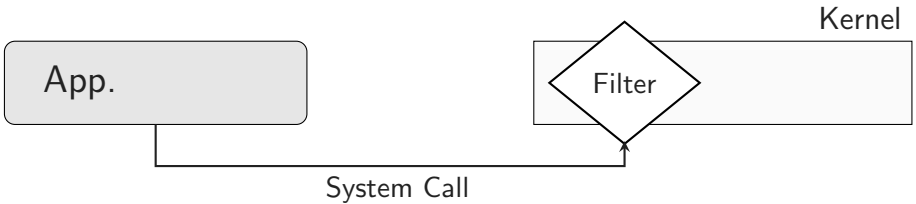
Motivation

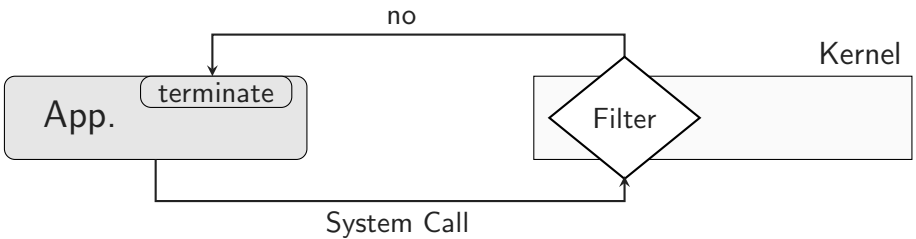
- Memory safety vulnerabilties are common
- Sandboxing helps in limiting their impact
- Linux seccomp: works but hard to do

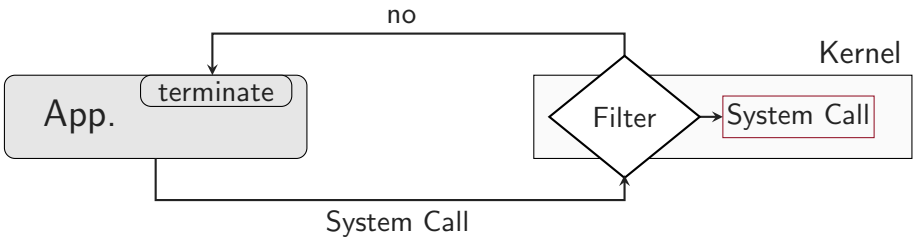
→ Can we automate seccomp sandboxing?

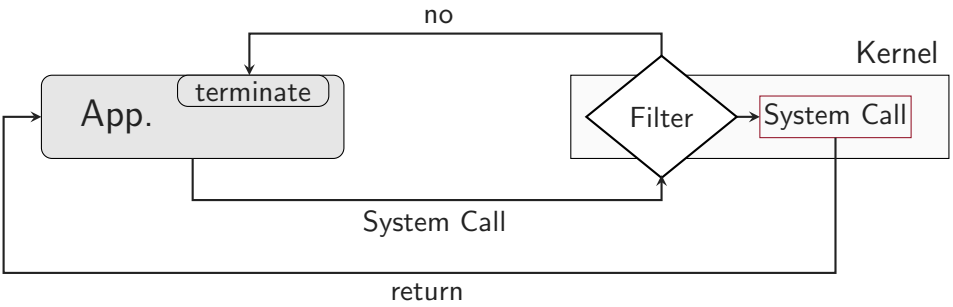

Kernel
App.


Kernel
App.
install
Filter


Kernel
App.
Filter
System Call


no
Kernel
terminate
App.
Filter
System Call


no
Kernel
terminate
App.
Filter
System Call
System Call


no
Kernel
terminate
App.
Filter
System Call
System Call
return

Sample

```c
int main(int argc, char* argv[]) {
    int infd, outfd;
    ssize_t read_bytes;
    char buffer[1024];

    printf("Copying '%s' to '%s'\n", argv[1], argv[2]);
    if((infd = open(argv[1], O_RDONLY)) > 0) {
        if((outfd = open(argv[2], O_WRONLY | O_CREAT, 0644)) > 0) {
            while((read_bytes = read(infd, &buffer, 1024)) > 0)
            write(outfd, &buffer, (ssize_t)read_bytes);
        }
    }
    close(infd);
    close(outfd);
    return 0;
}
```

Syscalls from C Functions?

Challenges

Syscalls from C Functions?

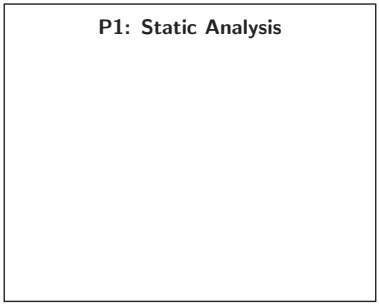
Entire Code Base?

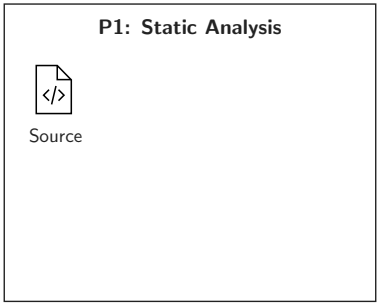
Challenges

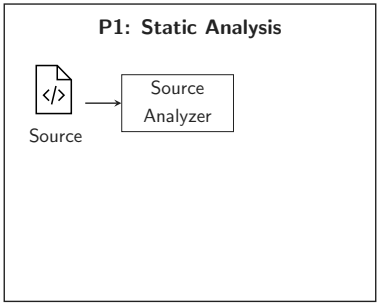
Syscalls from C Functions?

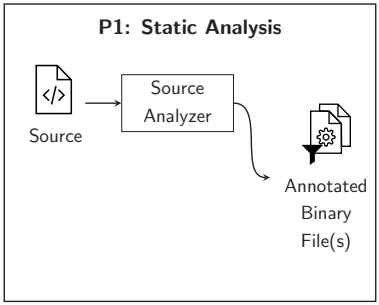
Entire Code Base?

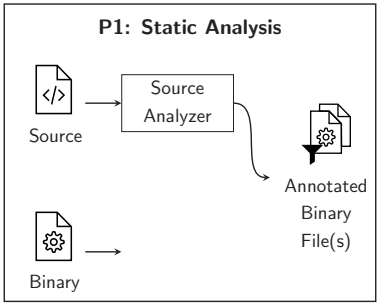
Third-party Libraries?

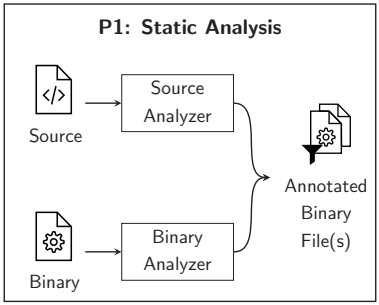

P1: Static Analysis

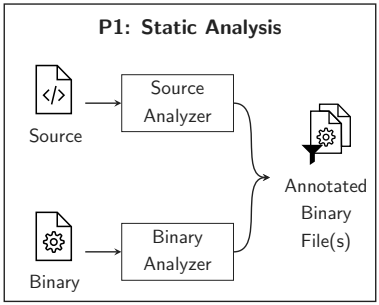

P1: Static Analysis
Source


P1: Static Analysis
Source
Source
Analyzer

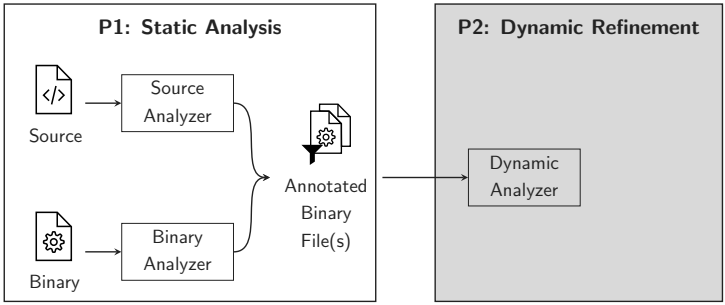

P1: Static Analysis
Source
Source Analyzer
Annotated Binary File(s)

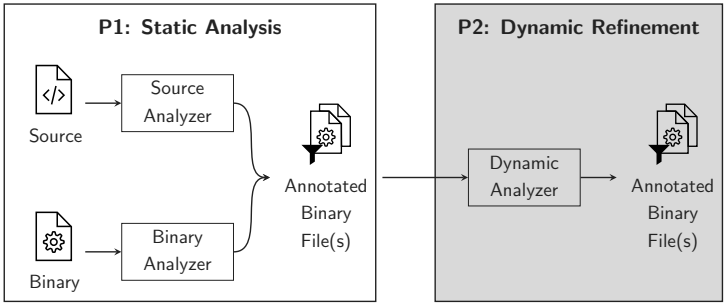

P1: Static Analysis
Source
Source Analyzer
Annotated Binary File(s)
Binary

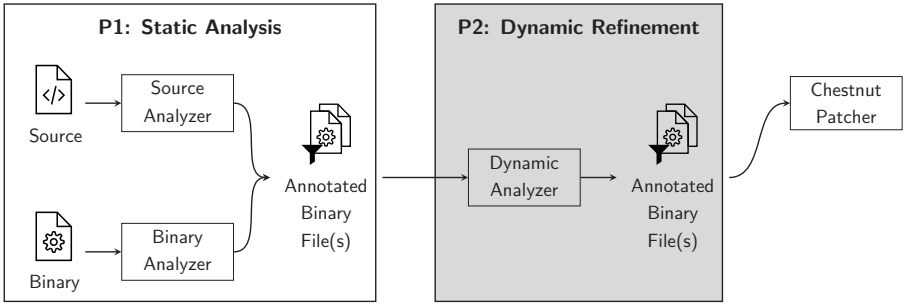

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)

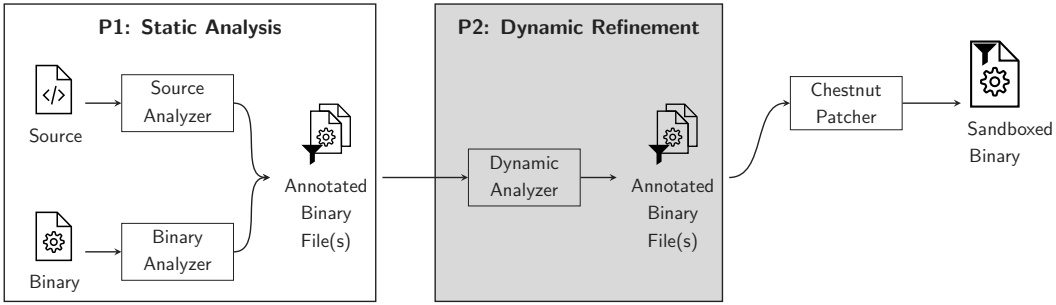


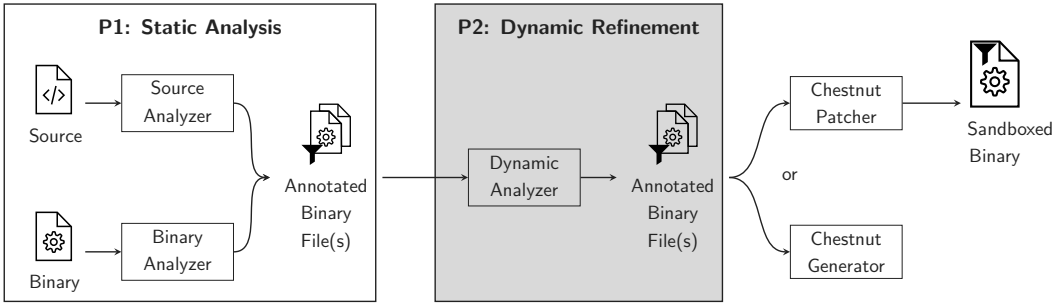
P2: Dynamic Refinement

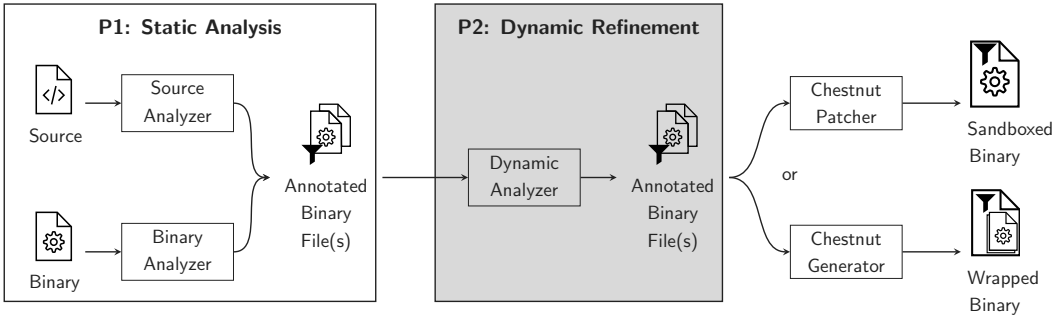

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer


P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)

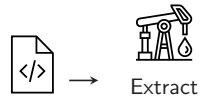

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher


P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher
Sandboxed Binary

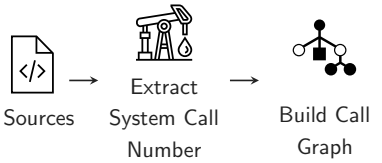

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher
or
Chestnut Generator
Sandboxed Binary

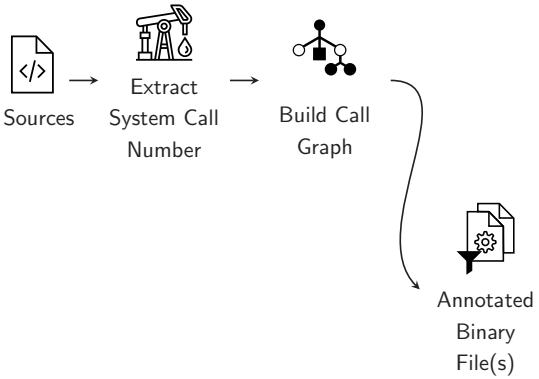

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher
or
Chestnut Generator
Sandboxed Binary
Wrapped Binary

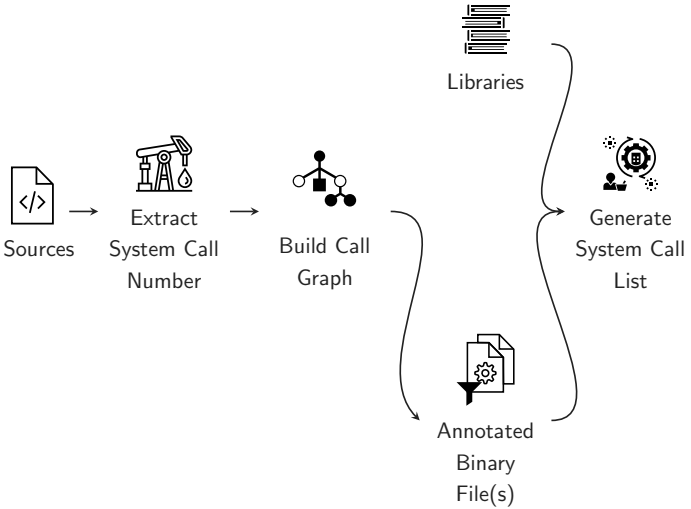
Sources

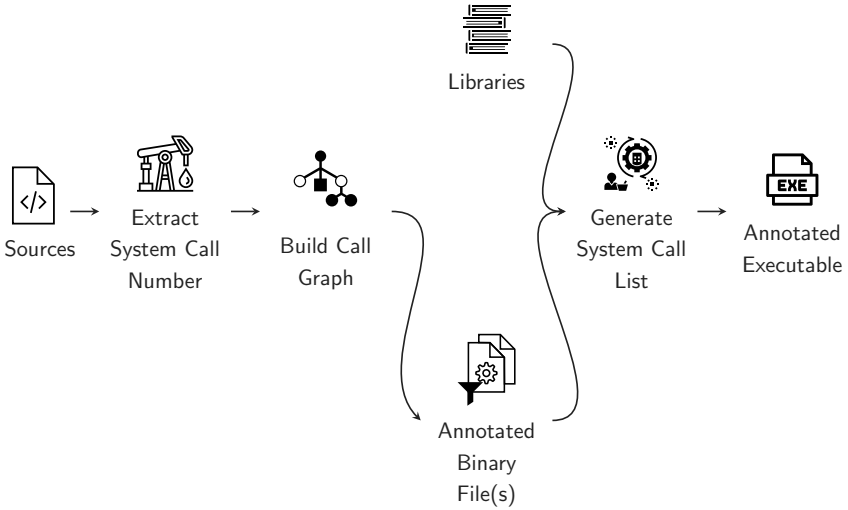

Sources
Extract
System Call
Number

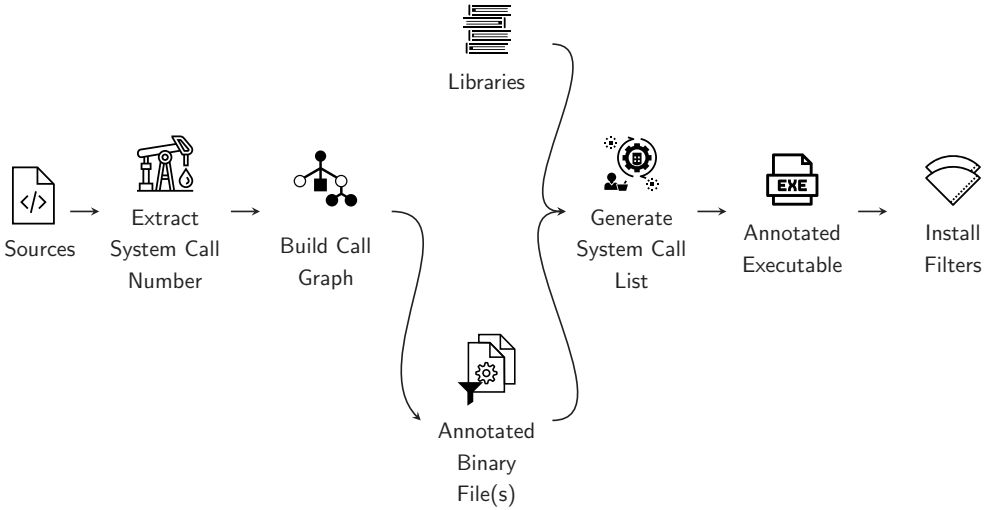
Sourcalyzer




Sources
Extract System Call Number
Build Call Graph
Annotated Binary File(s)


Libraries
Sources
Extract System Call Number
Build Call Graph
Generate System Call List
Annotated Binary File(s)


Libraries
Sources
Extract System Call Number
Build Call Graph
Generate System Call List
EXE
Annotated Executable
Annotated Binary File(s)


Libraries
Sources
Extract System Call Number
Build Call Graph
Annotated Binary File(s)
Generate System Call List
EXE
Annotated Executable
Install Filters

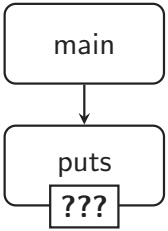
Call Graph Example

Call Graph Example

```
#include<stdio.h>
int main() {
  puts("Hello World!");
}
```

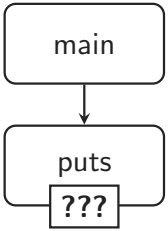
Call Graph Example

```c
#include<stdio.h>
int main() {
  puts("Hello World!");
}
```



Call Graph Example

```c
#include<stdio.h>
int main() {
  puts("Hello World!");
}
```

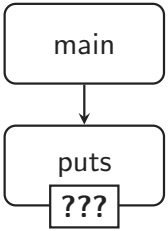


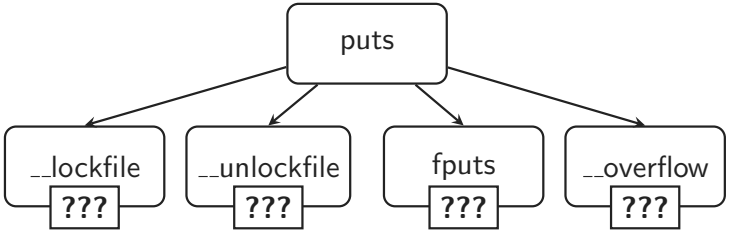

```c
// musl/src/stdio/puts.c
int puts(const char *s) {
  int r; FLOCK(stdout);
  r = -(fputs(s, stdout) < 0 ||
        putc_unlocked('\n', stdout) < 0);
  FUNLOCK(stdout); return r;
}
```

Call Graph Example

```c
#include<stdio.h>
int main() {
  puts("Hello World!");
}
```

```c
// musl/src/stdio/puts.c
int puts(const char *s) {
  int r; FLOCK(stdout);
  r = -(fputs(s, stdout) < 0 ||
        putc_unlocked('\n', stdout) < 0);
  FUNLOCK(stdout); return r;
}
```





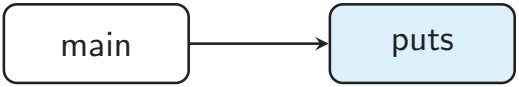
Call Graph Example

```c
#include<stdio.h>
int main() {
  puts("Hello World!");
}
```

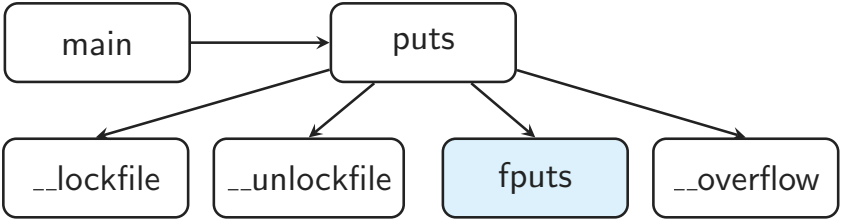
```c
// musl/src/stdio/puts.c
int puts(const char *s) {
  int r; FLOCK(stdout);
  r = -(fputs(s, stdout) < 0 ||
        putc_unlocked('\n', stdout) < 0);
  FUNLOCK(stdout); return r;
}
```

```
{"call_targets": ["__lockfile",
                  "fputs",
                  "__overflow",
                  "__unlockfile"],
"name": "puts",
"type": "i32 (i8*)"}
```

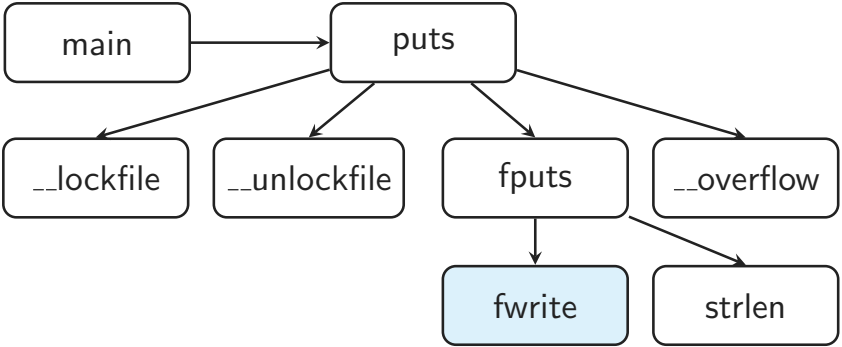
Call Graph Example



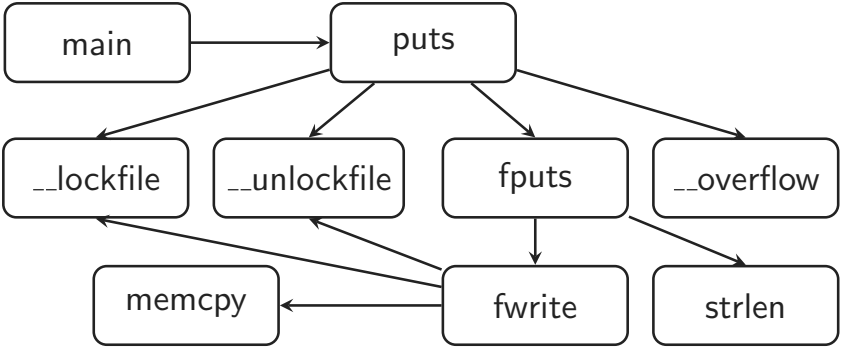
Call Graph Example



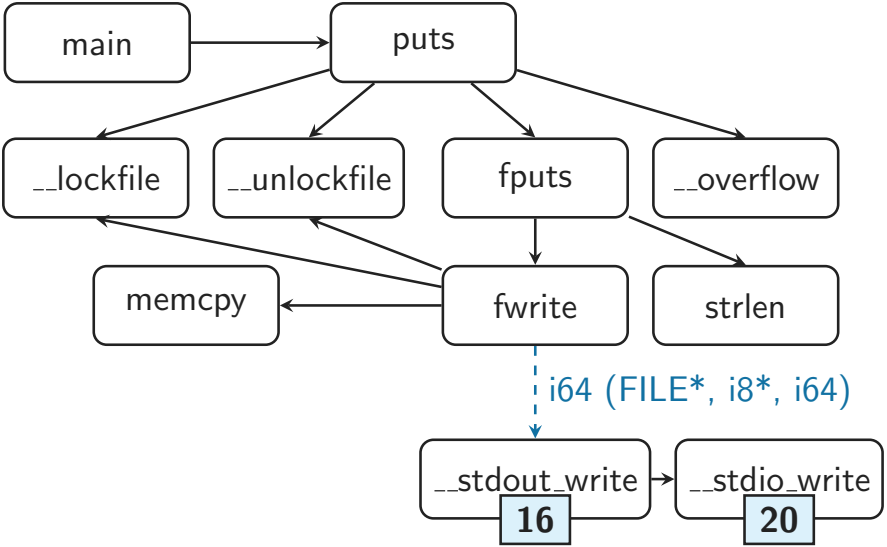
Call Graph Example



Call Graph Example



Call Graph Example



- Extract system calls from existing binaries/libraries

Binalyzer: Overview

- Extract system calls from existing binaries/libraries
- Capstone: disassemble binary

Binalyzer: Overview

- Extract system calls from existing binaries/libraries
- Capstone: disassemble binary
- Anger: build call graph

```
mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
```

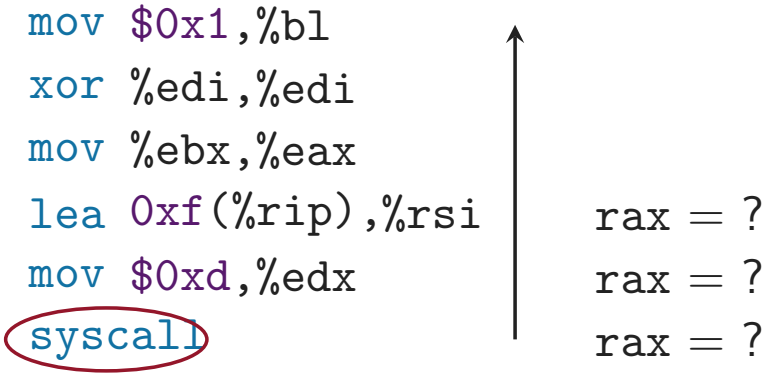
```
mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
```

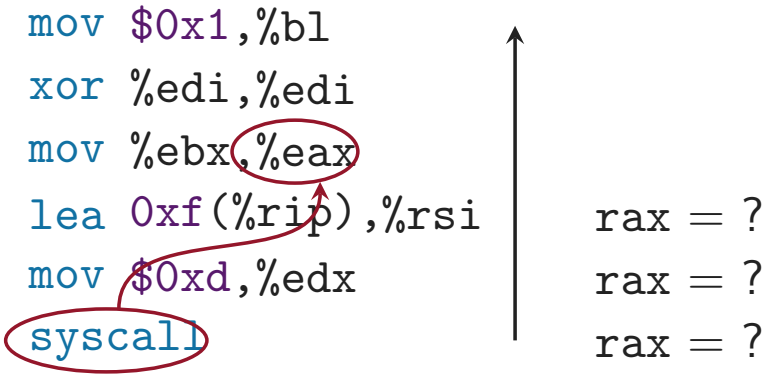
rax = ?

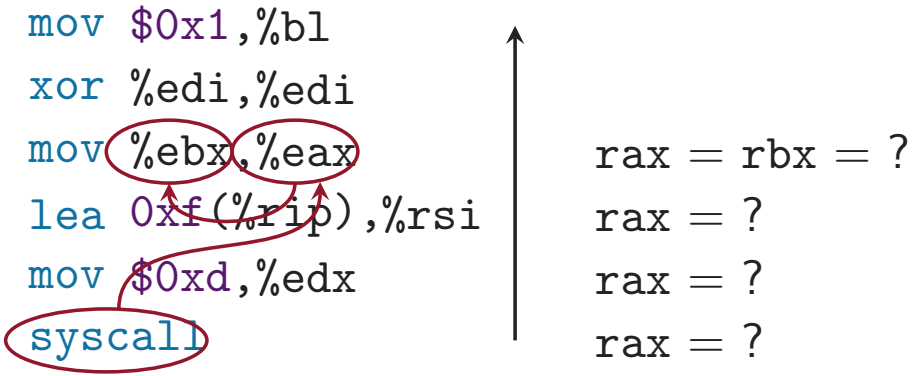
```
mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
```

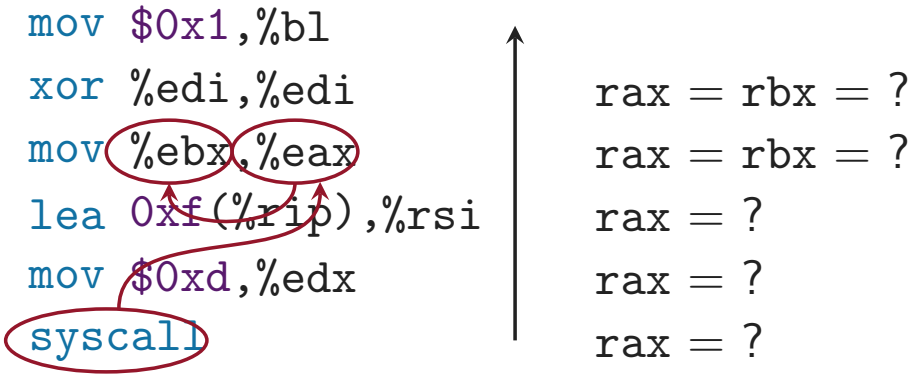
rax = ?

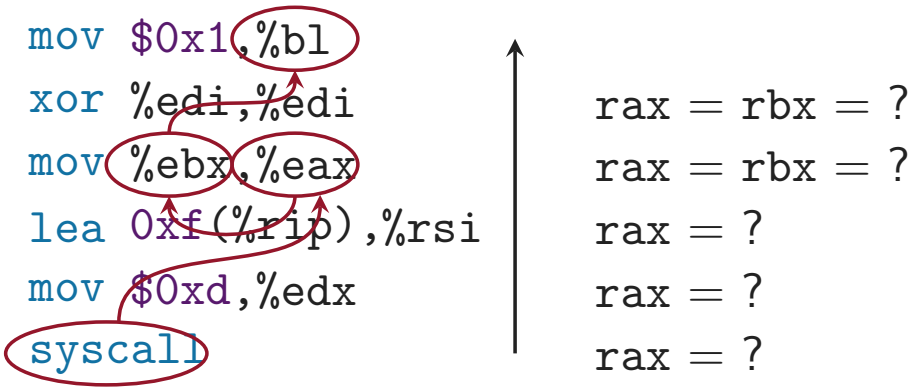
rax = ?

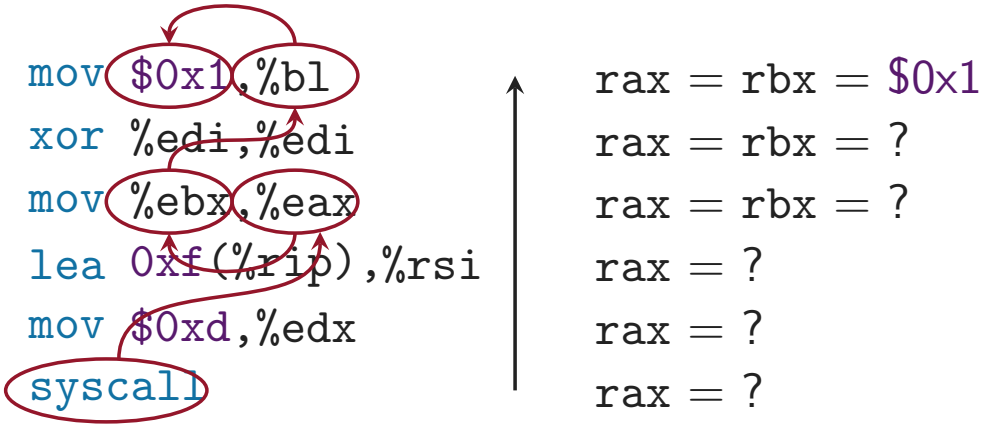

mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = ?
rax = ?
rax = ?

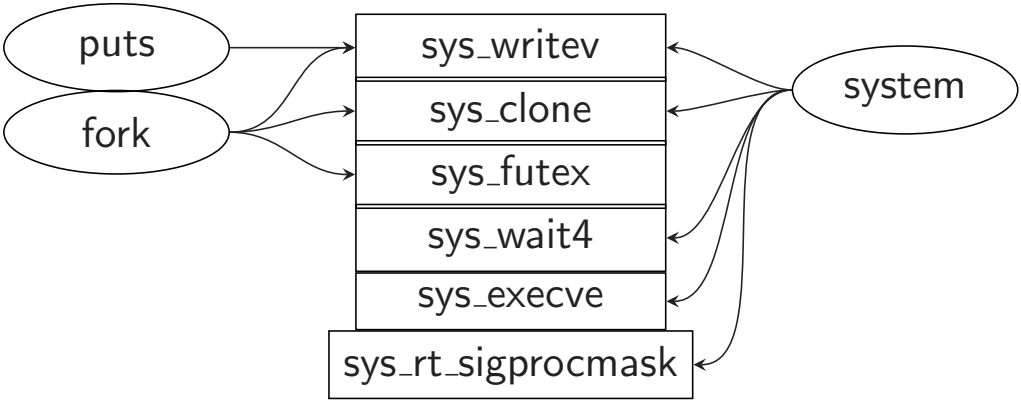

mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = ?
rax = ?
rax = ?

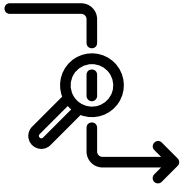

mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = rbx = ?
rax = ?
rax = ?
rax = ?


mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = rbx = ?
rax = rbx = ?
rax = ?
rax = ?
rax = ?


mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = rbx = ?
rax = rbx = ?
rax = ?
rax = ?
rax = ?

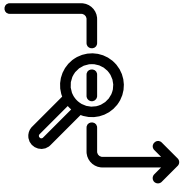

mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = rbx = $0x1
rax = rbx = ?
rax = rbx = ?
rax = ?
rax = ?
rax = ?


puts
fork
sys_writev
sys_clone
sys_futex
sys_wait4
sys_execve
sys_rt_sigprocmask
system

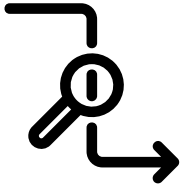


- Strace-like system

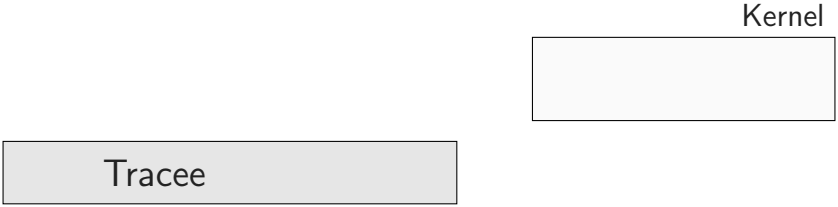
Finalyzer: Overview

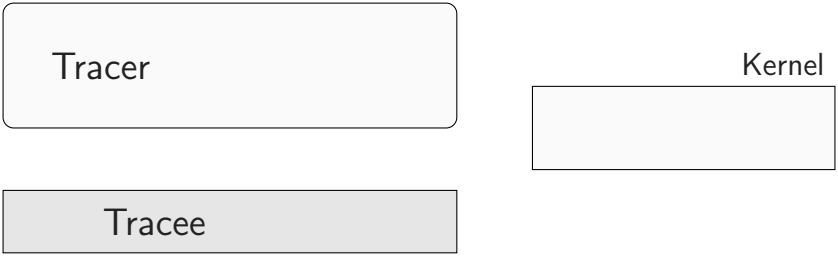


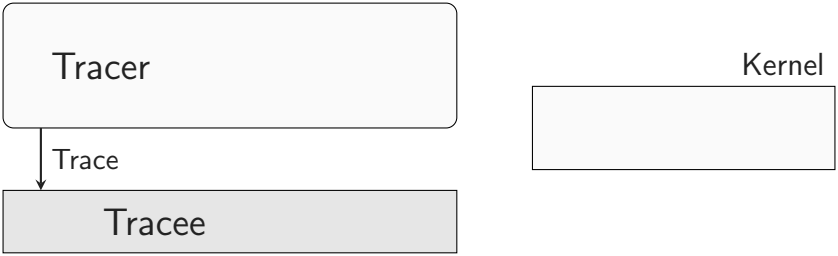
- Strace-like system
- Dynamically trace system calls

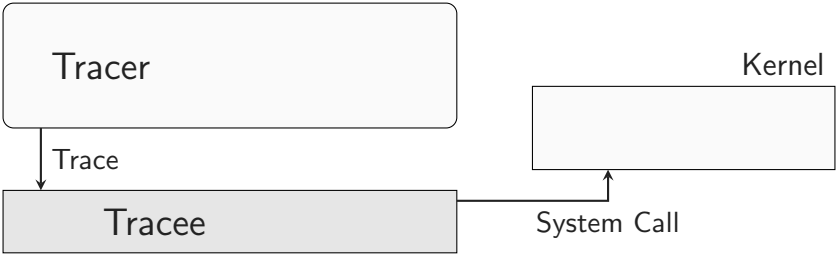


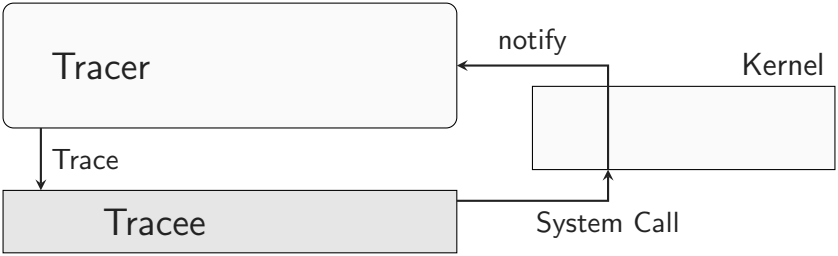
- Strace-like system
- Dynamically trace system calls
- Automatically add missed system calls or optionally remove never-used ones

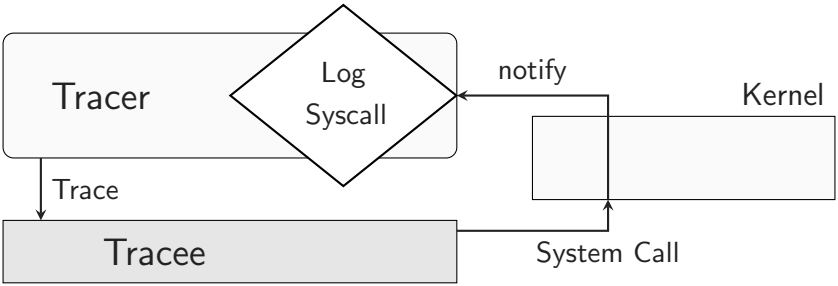

Kernel
Tracee

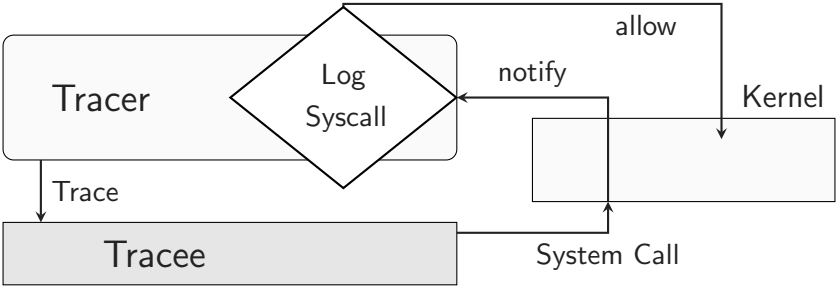

Tracer
Kernel
Tracee

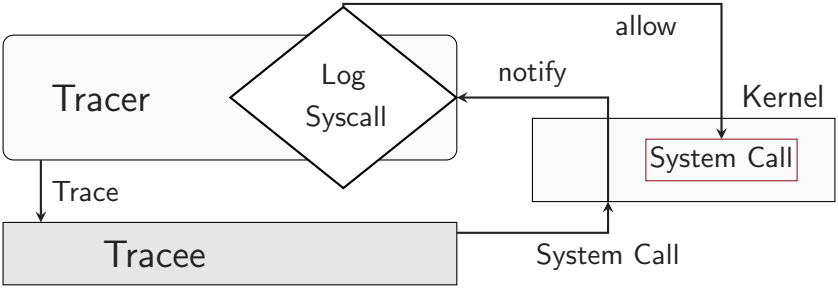

Tracer
Trace
Tracee
Kernel

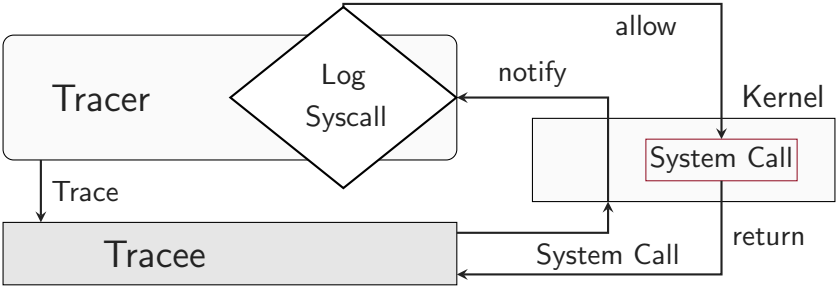

Tracer
Kernel
Trace
Tracee
System Call

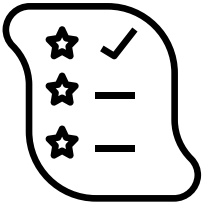

Tracer
notify
Kernel
Trace
Tracee
System Call


Tracer
Log
Syscall
notify
Kernel
Trace
Tracee
System Call

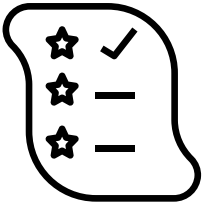

allow
Tracer
Log
Syscall
notify
Kernel
Trace
Tracee
System Call


allow
Tracer
Log
Syscall
notify
Kernel
System Call
Trace
Tracee
System Call

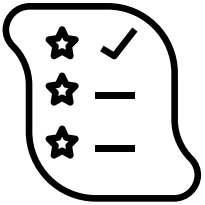

allow
Tracer
Log
Syscall
notify
Kernel
System Call
Trace
Tracee
System Call
return



- Performance, Functional Correctness, and Security



- Performance, Functional Correctness, and Security
- Analyzed Client, Server, and Database applications



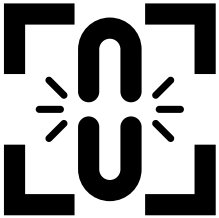
- Performance, Functional Correctness, and Security
- Analyzed Client, Server, and Database applications
- 18 applications

Evaluation: Performance

- Worst Compile Time Overhead (git): 28 % (+19 s)

Evaluation: Performance

- Worst Compile Time Overhead (git): 28 % (+19 s)
- Worst Binary Extraction Time (ffmpeg): 11 min



- Seccomp has inherent performance impact → nothing Chestnut can do about that
- Recent work (Linux 5.11) improved performance

Evaluation: Functional Correctness

- Use application testsuites for checks

Evaluation: Functional Correctness

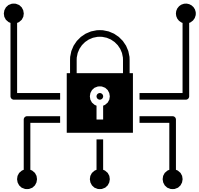
- Use application testsuites for checks
- Code coverage metrics for better estimations of correctness

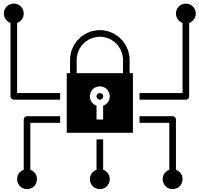
- Use application testsuites for checks
- Code coverage metrics for better estimations of correctness
  - Line coverage: 59-77 %
  - Function coverage: 61-92 %

- Use application testsuites for checks
- Code coverage metrics for better estimations of correctness
  - Line coverage: 59-77 %
  - Function coverage: 61-92 %
- Observed no crashes in tests

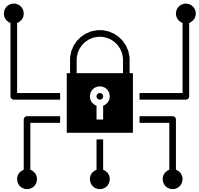
- Use application testsuites for checks
- Code coverage metrics for better estimations of correctness
  - Line coverage: 59-77 %
  - Function coverage: 61-92 %
- Observed no crashes in tests
- 6 month long-term study using nginx: no crashes

- Avg. Number of blocked system calls:

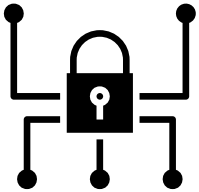




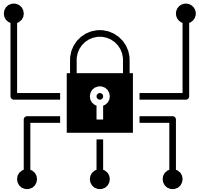
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)



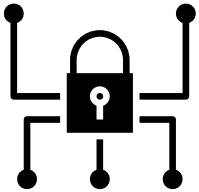
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:



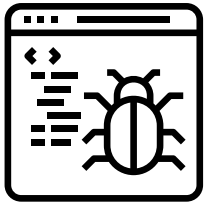
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:
  - : 9 (50 %)
  - : 14 (78 %)



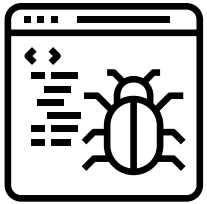
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:
  - : 9 (50 %)
  - : 14 (78 %)
- `mprotect` system calls blocked:



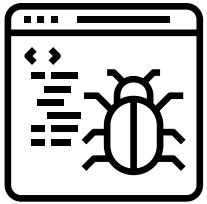
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:
  - : 9 (50 %)
  - : 14 (78 %)
- `mprotect` system calls blocked:
  - : 11 (61 %)
  - : 0 (0 %)



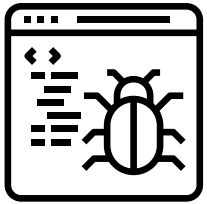
- 175 CVEs extracted from mitre database



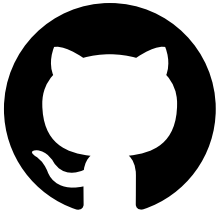
- 175 CVEs extracted from mitre database
- Full CVEs:



- 175 CVEs extracted from mitre database
- Full CVEs:
  - : 64 %
  - : 62 %



- 175 CVEs extracted from mitre database
- Full CVEs:
  - : 64 %
  - : 62 %
- Subvariants:
  - : 75 %
  - : 72 %



You can find our proof-of-concept implementation of Chestnut on:

- `https://github.com/IAIK/Chestnut`

More details in the paper

- More detailed security evaluation
- Information on overapproximation
- More implementation details
- ...

CCSW [Can+21]

Claudio Canella, Mario Werner, Daniel Gruss, Michael Schwarz.
Automating Seccomp Filter Generation for Linux Applications.

- Reduced time-consuming, manual analysis to automated process

- Reduced time-consuming, manual analysis to automated process
- Showed that we can improve overall system security

- Reduced time-consuming, manual analysis to automated process
- Showed that we can improve overall system security
- Chestnut only has small performance impact

Automating Seccomp Filter Generation for Linux Applications

Claudio Canella[1], **Mario Werner**[1], **Daniel Gruss**[1], **Michael Schwarz**[2]

[1]Graz University of Technology [2]CISPA Helmholtz Center for Information Security

References

[Can+21] C. Canella, M. Werner, D. Gruss, and M. Schwarz. Automating Seccomp Filter Generation for Linux Applications. In: CCSW. 2021.

Acknowledgements

This work has been supported by the Austrian Research Promotion Agency (FFG) via the project ESPRESSO, which is funded by the province of Styria and the Business Promotion Agencies of Styria and Carinthia. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 681402). Additional funding was provided by generous gifts from ARM, Intel, and Red Hat. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.