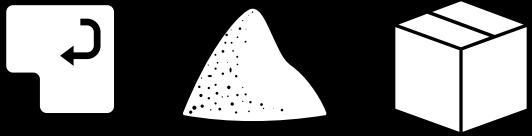
black hat

ENTER SANDBOX



Claudio Canella, Mario Werner, Michael Schwarz

Claudio Canella

PhD candidate @ Graz University of Technology

@cc0x1f

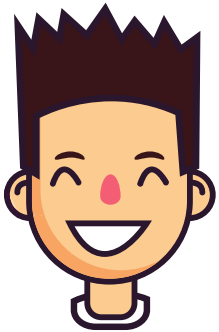
claudio.canella@iaik.tugraz.at

Mario Werner

PhD @ Graz University of Technology
(when performing this research)

Now: Hardware Design Engineer @ NXP Semiconductors

https://we.rner.at/



Michael Schwarz

Faculty @ CISPA Helmholtz Center for Information Security

@misc0110

michael.schwarz@cispa.saarland

- Memory safety vulnerabilties are common

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

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

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

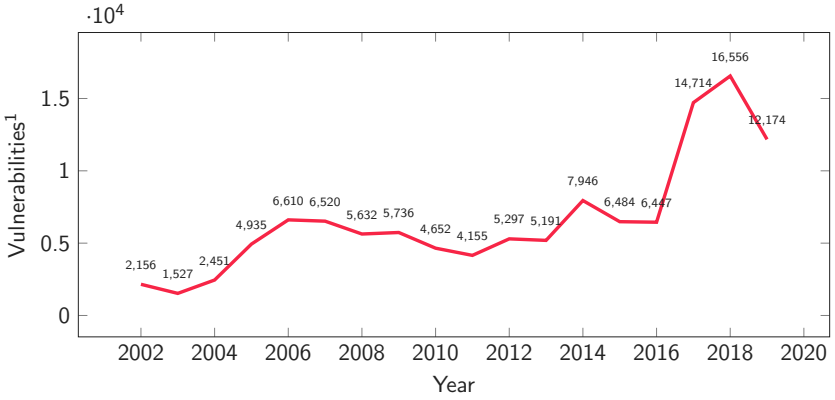
→ Can we automate seccomp sandboxing?

A few months ago...

```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;
}
```

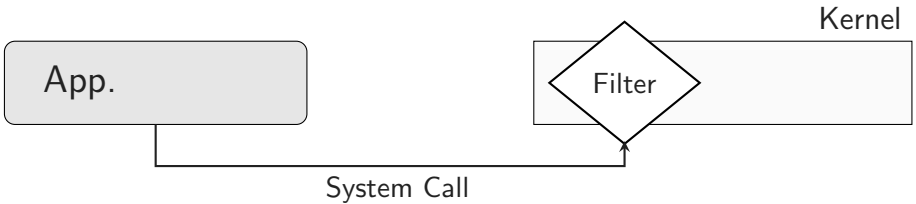
CVEs

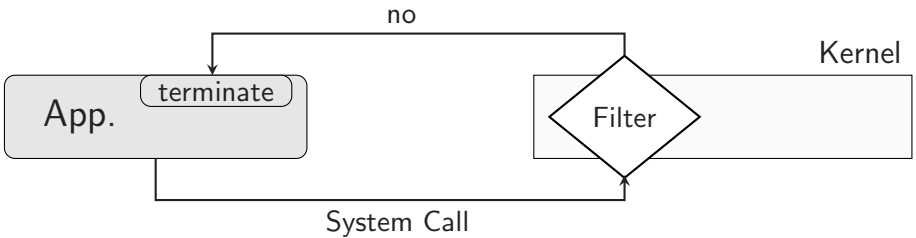


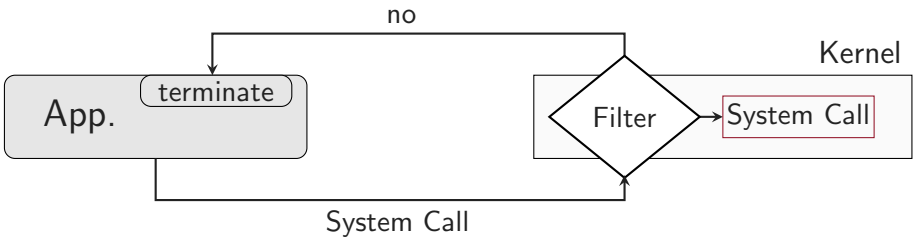
[1]Source: `http://www.cvedetails.com/vulnerabilities-by-types.php`

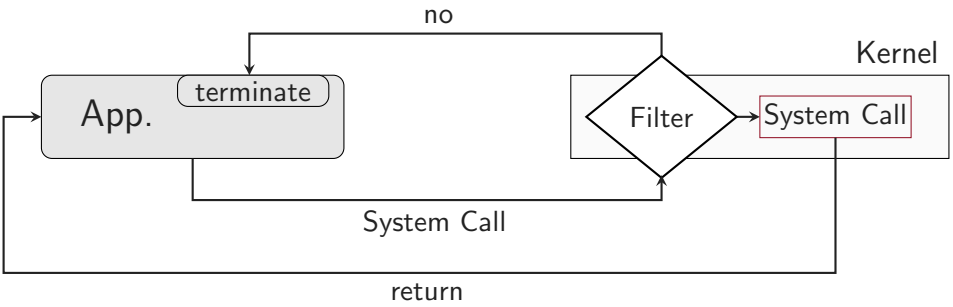

Kernel
App.


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

```
 1 struct sock_filter filter [] = {
 2  BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, nr))),
 3  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_write, 0, 1),
 4  BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
 5  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_open, 0, 3),
 6  BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, args[1]))),
 7  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, O_RDONLY, 0, 1),
 8  BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
 9  BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL)
10 };
11 struct sock_fprog prog = {
12     .len = (unsigned short)(sizeof(filter) / sizeof (filter[0])),
13     .filter = filter,
14 };
15
16 printf("Configuring seccomp\n");
17 prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0);
18 prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, &prog);
```

```
prctl(PR_SET_NO_NEW_PRIVS, 1);
prctl(PR_SET_DUMPABLE, 0);
scmp_filter_ctx ctx;
ctx = seccomp_init(SCMP_ACT_KILL);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(rt_sigreturn), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(openat), 1,
  SCMP_A2(SCMP_CMP_EQ, O_RDONLY));
seccomp_load(ctx);
```

Syscalls from C Functions?

Syscalls from C Functions?

Entire Code Base?

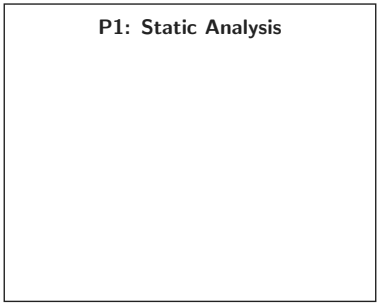
Syscalls from C Functions?

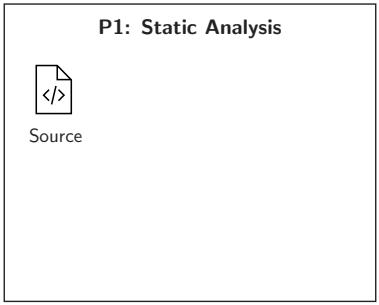
Entire Code Base?

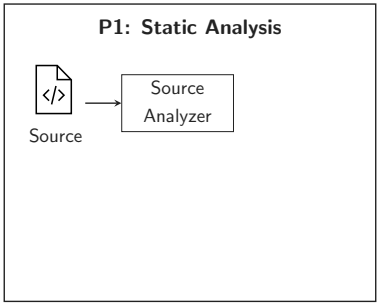
Third-party Libraries?

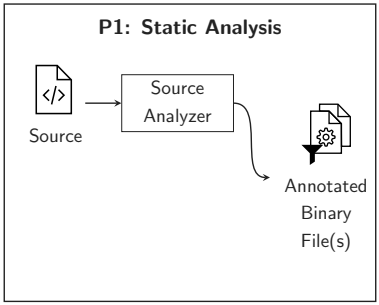
Months of research and engineering later

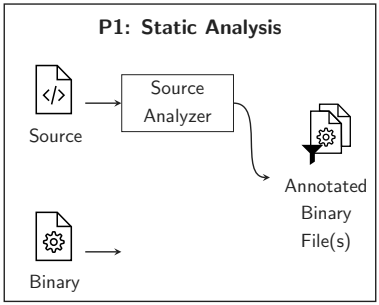
Exit lite
Enter tight
Let's withstand
We're off to never exploit-land

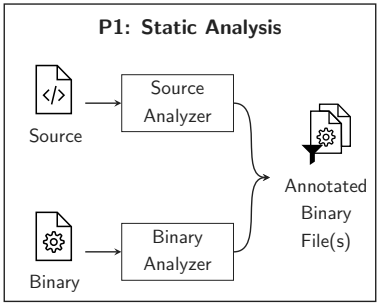

P1: Static Analysis

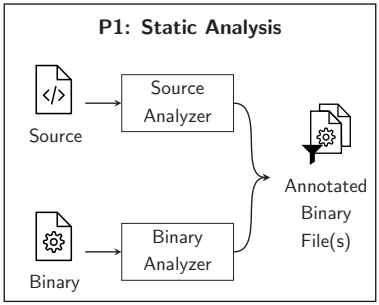

P1: Static Analysis
Source

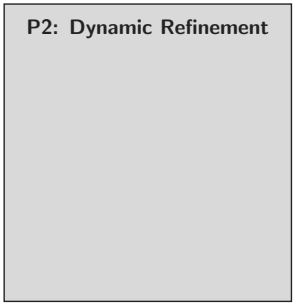

P1: Static Analysis
Source
Source Analyzer

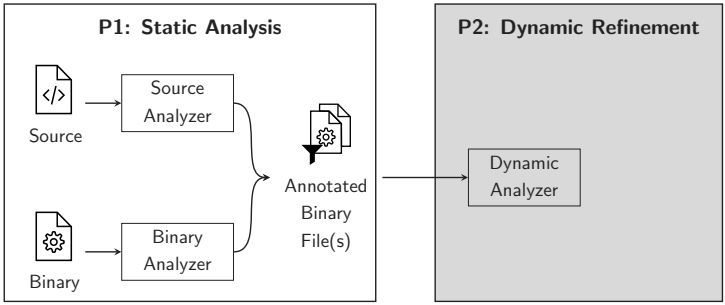

P1: Static Analysis
Source
Source Analyzer
Annotated Binary File(s)

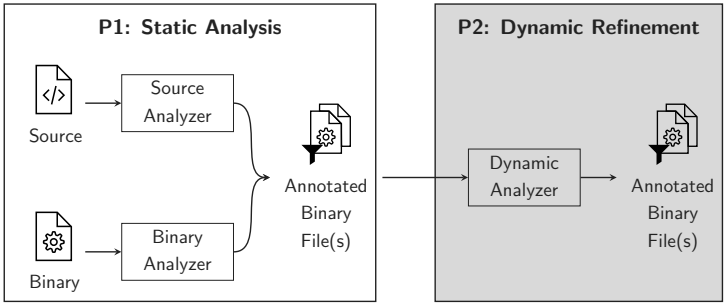

P1: Static Analysis
Source
Source Analyzer
Annotated Binary File(s)
Binary

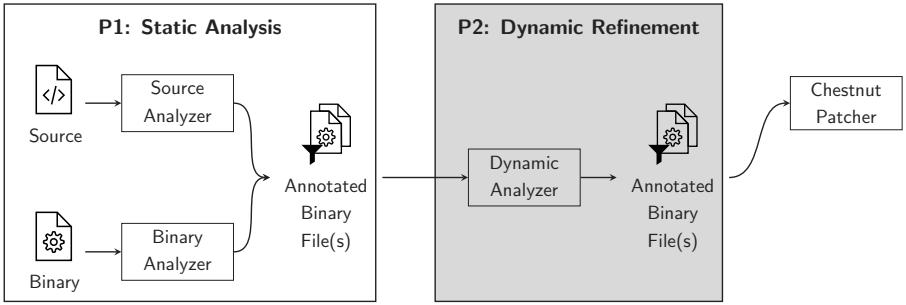

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)

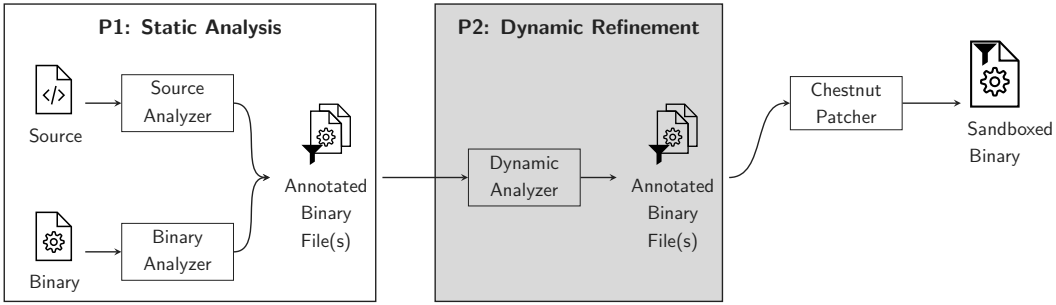

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)

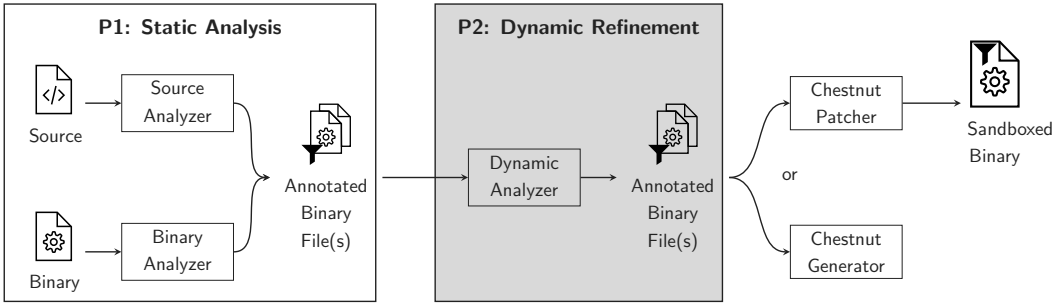

P2: Dynamic Refinement

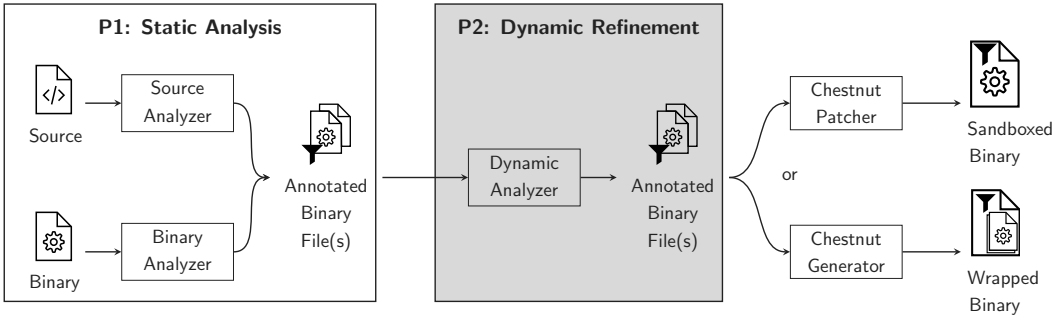

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer


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

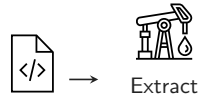

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher

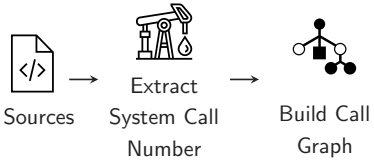

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher
Sandboxed Binary

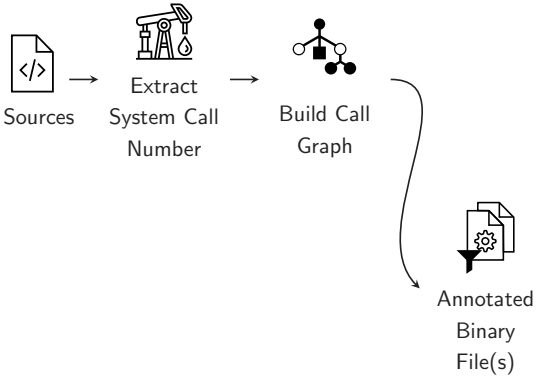

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher
or
Chestnut Generator
Sandboxed Binary

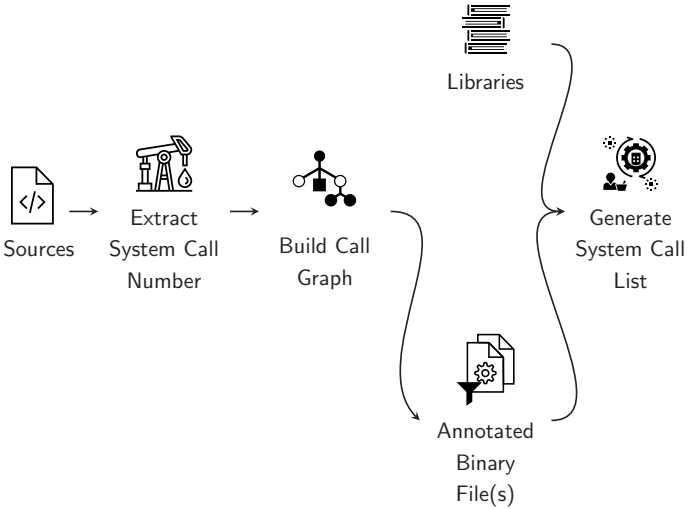

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)
Chestnut Patcher
or
Chestnut Generator
Sandboxed Binary
Wrapped Binary

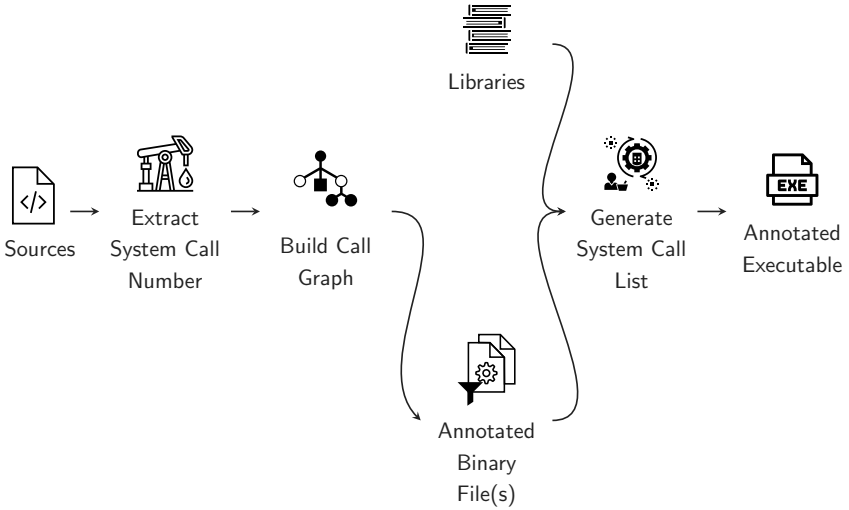
Sources

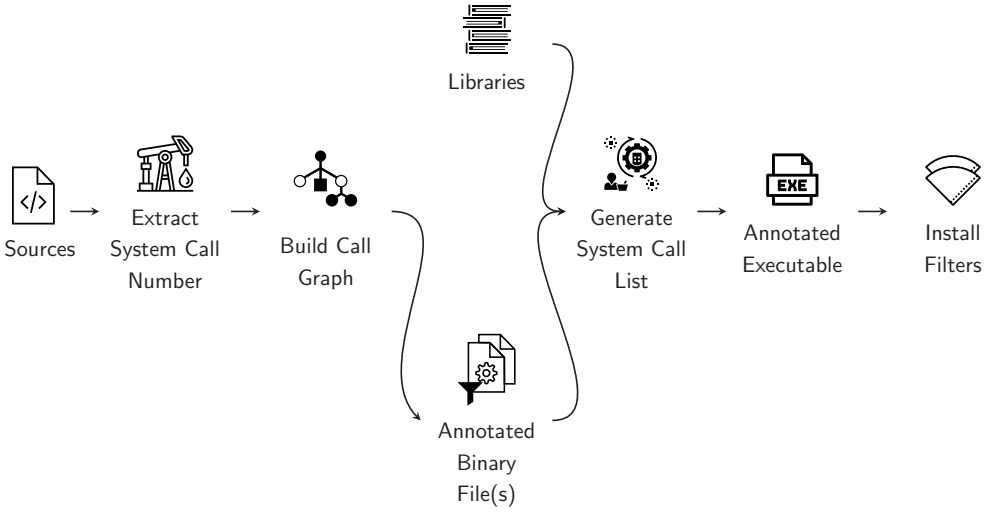

Sources
Extract
System Call
Number


Sources
Extract
System Call
Number
Build Call
Graph


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
Annotated Binary File(s)
Generate System Call List
EXE
Annotated Executable


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

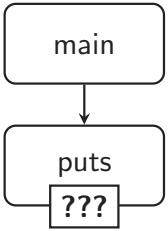
Call Graph Example

Call Graph Example

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

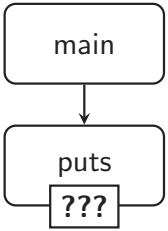
Call Graph Example

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



Call Graph Example

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

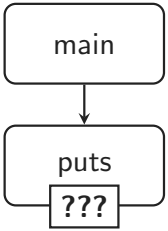


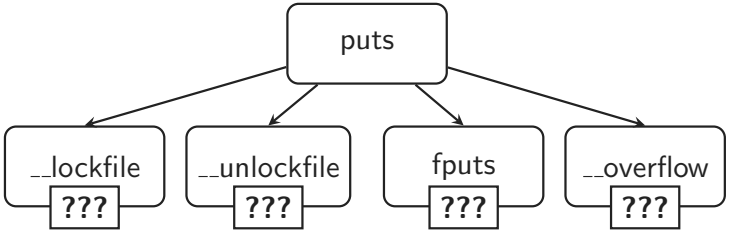

```c
// musl/src/stdio/puts.c
int puts(const char *s) {
  int r; FLOCK(stdout);
  r = -(fputs(s, stdout) < 0 ||
        putc_unlocked('\n', stdout) < 0);
  FUNLOCK(stdout); return r;
}
```

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;
}
```



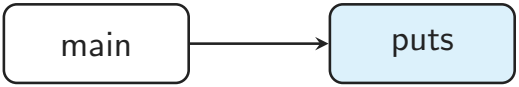


Call Graph Example

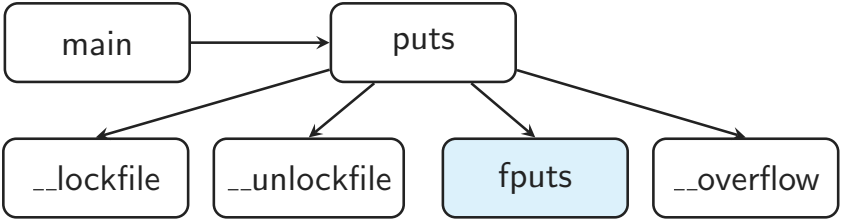
```c
#include<stdio.h>
int main() {
  puts("Hello World!");
}
```

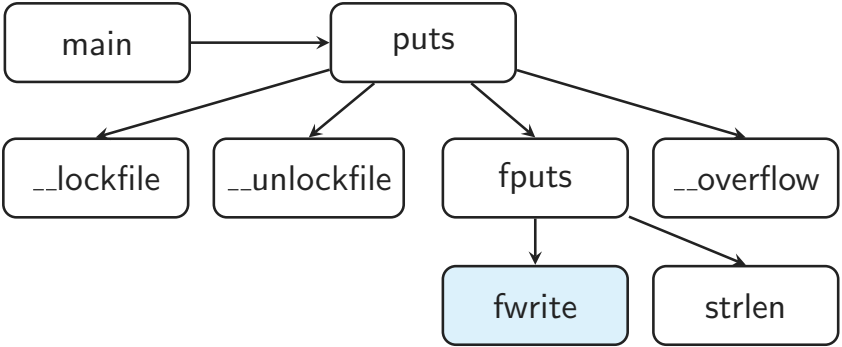
```c
// musl/src/stdio/puts.c
int puts(const char *s) {
  int r; FLOCK(stdout);
  r = -(fputs(s, stdout) < 0 ||
        putc_unlocked('\n', stdout) < 0);
  FUNLOCK(stdout); return r;
}
```

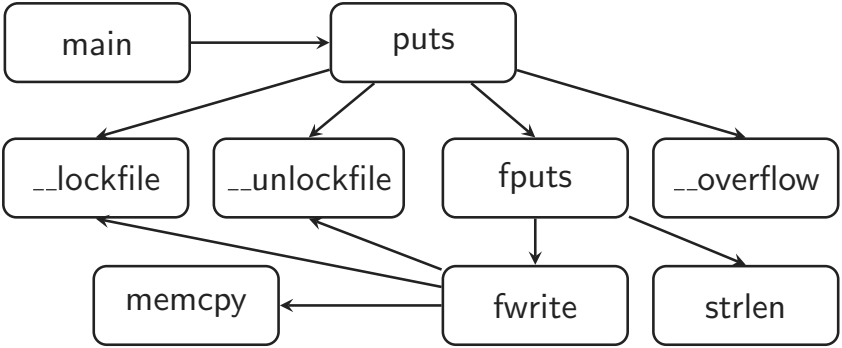
```
{"call_targets": ["__lockfile",
                  "fputs",
                  "__overflow",
                  "__unlockfile"],
"name": "puts",
"type": "i32 (i8*)"}
```

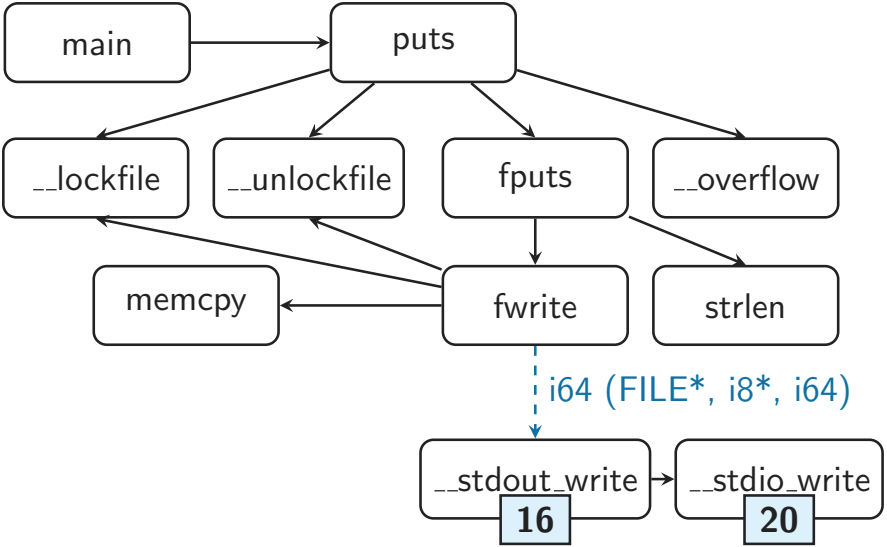

main
puts

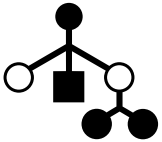
Call Graph Example



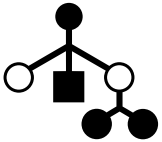

main
puts
__lockfile
__unlockfile
fputs
__overflow
fwrite
strlen


main
puts
__lockfile
__unlockfile
fputs
__overflow
memcpy
fwrite
strlen


main
puts
__lockfile
__unlockfile
fputs
__overflow
memcpy
fwrite
strlen
i64 (FILE*, i8*, i64)
__stdout_write
16
__stdio_write
20



- Extract and serialize meta data during compilation:
  - Function names with signatures
  - Partial call graphs (direct calls)
  - Indirectly called function signatures
  - *hasAddressTaken* information
  - Syscall information



- Extract and serialize meta data during compilation:
  - Function names with signatures
  - Partial call graphs (direct calls)
  - Indirectly called function signatures
  - *hasAddressTaken* information
  - Syscall information
- Generate complete call graph approximation at link time
- Utilize link-time garbage collection to prune the graph/syscalls
- Use function-signature heuristic to resolve indirect calls

→ No time-consuming whole-program analysis required

```
/tmp/ffmpeg ls
Changelog     configure          COPYING.LGPLv2.1  ffbuild     ffprobe_g   libavdevice    libavutil      LICENSE.md          parse_elf.py  tests
compat        CONTRIBUTING.md    COPYING.LGPLv3    ffmpeg      fftools     libavfilter    libpostproc    MAINTAINERS         presets       tools
config.asm    COPYING.GPLv2      CREDITS           ffmpeg_g    INSTALL.md  libavformat    libswresample  Makefile            README.md
config.h      COPYING.GPLv3      doc               ffprobe     libavcodec  libavresample  libswscale     nginx_sandbox.mp4   RELEASE
/tmp/ffmpeg
```

- Extract system calls from existing binaries/libraries

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

- 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
```

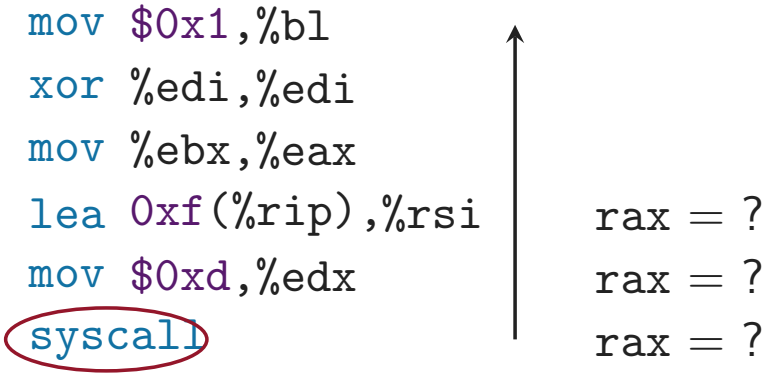
```
mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
```

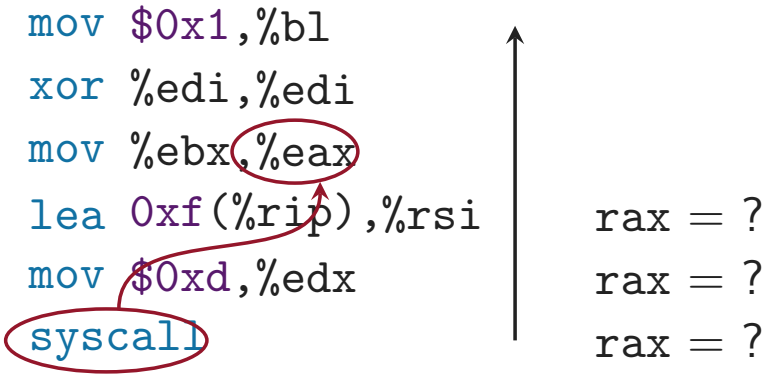
rax = ?

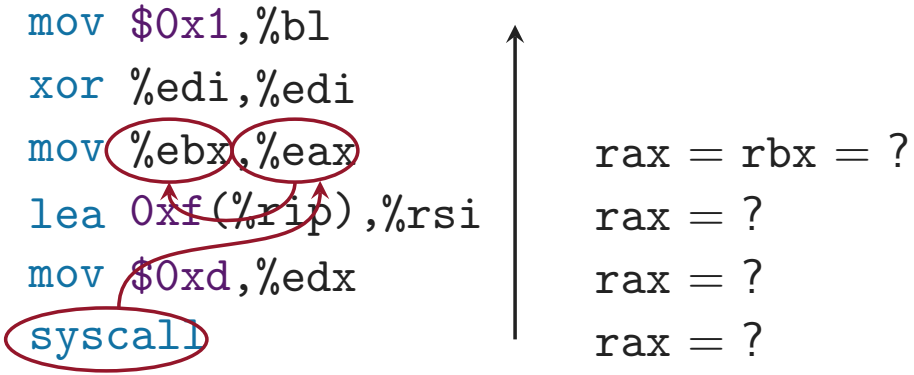
```
mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
```

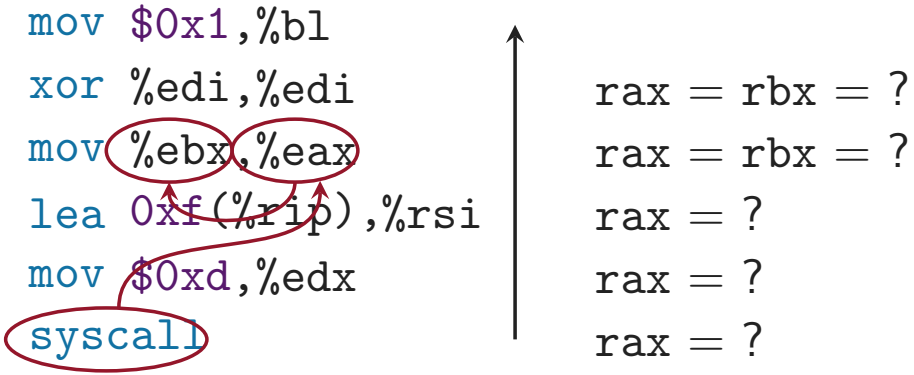
rax = ?

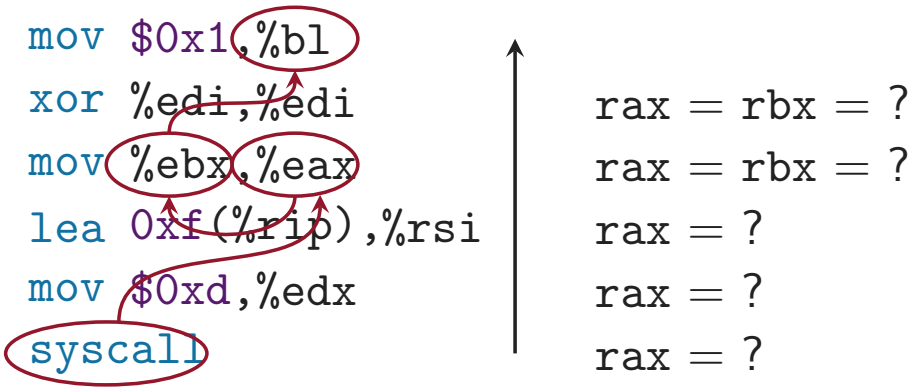
rax = ?

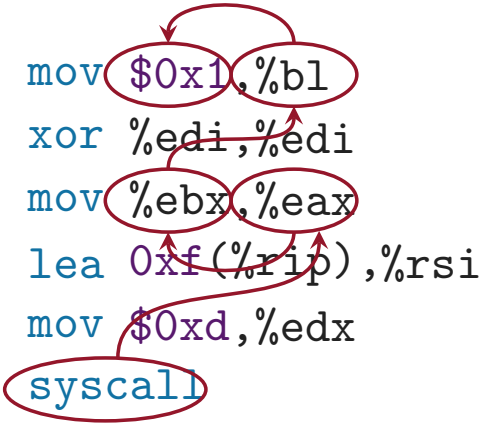

mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = ?
rax = ?
rax = ?


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

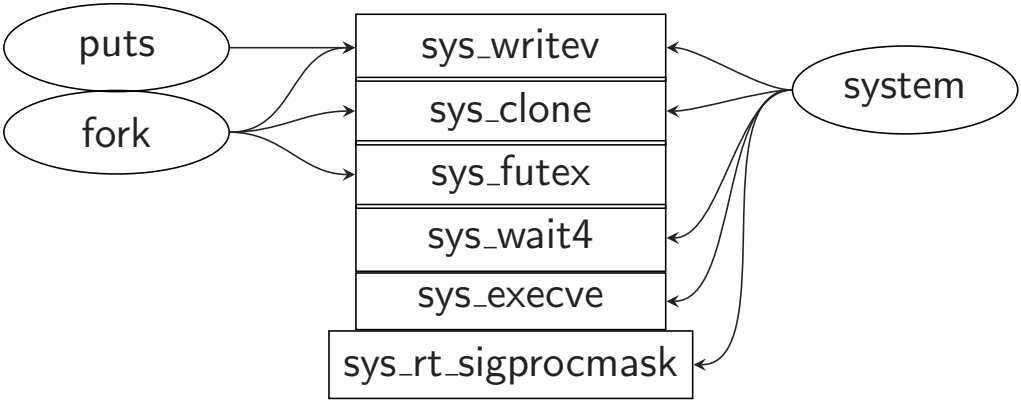

mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = rbx = ?
rax = ?
rax = ?
rax = ?


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

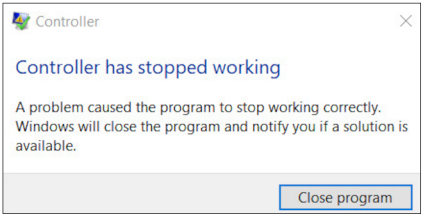

mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
rax = rbx = ?
rax = rbx = ?
rax = ?
rax = ?
rax = ?

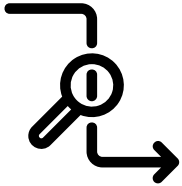


$$\begin{aligned}
\texttt{rax} &= \texttt{rbx} = \texttt{\$0x1} \\
\texttt{rax} &= \texttt{rbx} = ? \\
\texttt{rax} &= \texttt{rbx} = ? \\
\texttt{rax} &= ? \\
\texttt{rax} &= ? \\
\texttt{rax} &= ?
\end{aligned}$$

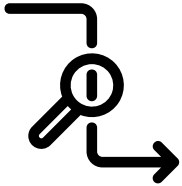

puts
fork
sys_writev
sys_clone
sys_futex
sys_wait4
sys_execve
sys_rt_sigprocmask
system

```
✘ /tmp/redis ls
00-RELEASENOTES  CONTRIBUTING  deps      INSTALL   MANIFESTO     README.md   runtest          runtest-moduleapi  sentinel.conf  tests   utils
BUGS             COPYING       dump.rdb  Makefile  parse_elf.py  redis.conf  runtest-cluster  runtest-sentinel   src            TLS.md
 /tmp/redis python parse_elf.py parse-file src/redis-server
```

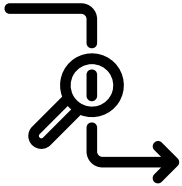

Controller
Controller has stopped working
A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.
Close program



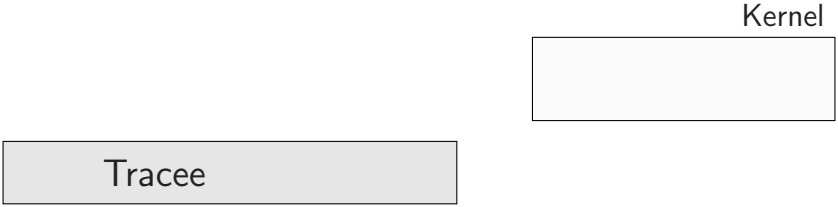
- Strace-like system

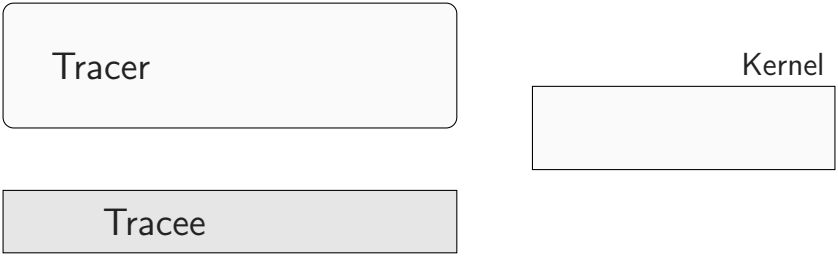


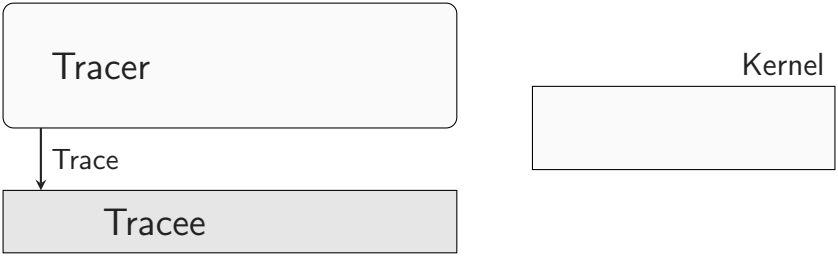
- Strace-like system
- Dynamically trace system calls

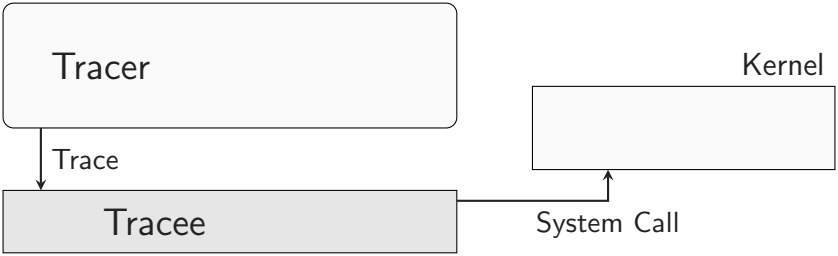


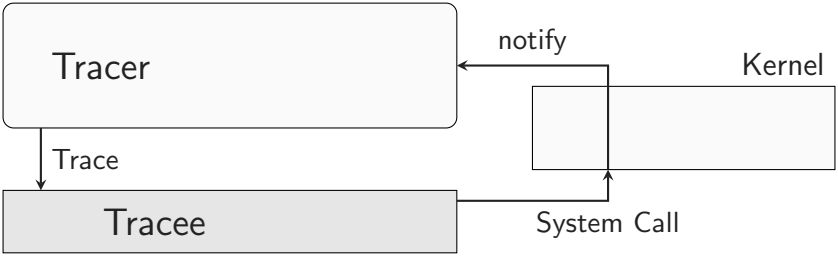
- Strace-like system
- Dynamically trace system calls
- Automatically add missed system calls or optionally remove never-used ones

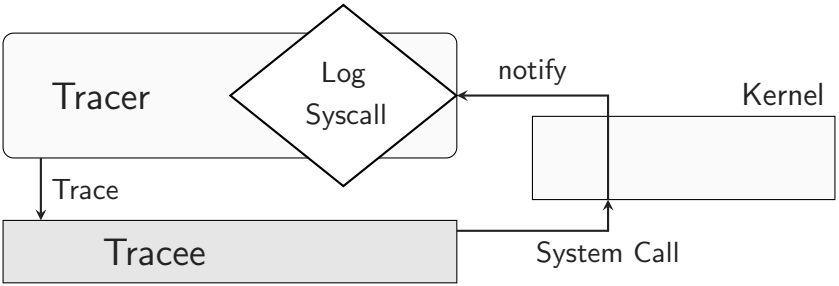

Kernel
Tracee

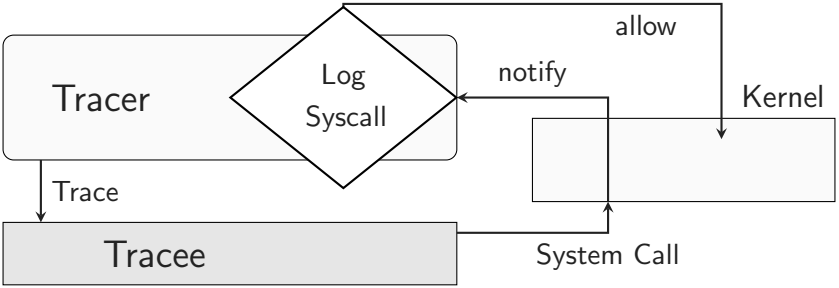

Tracer
Kernel
Tracee

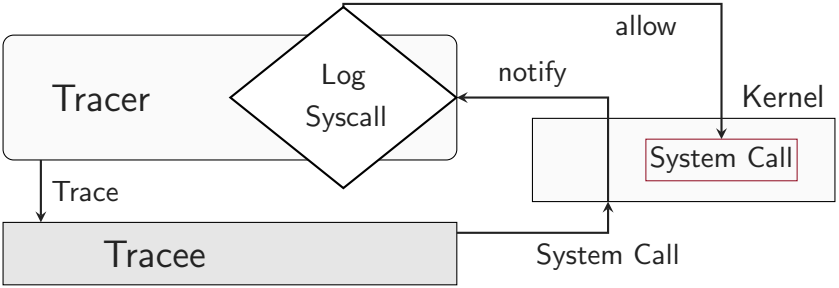

Tracer
Trace
Tracee
Kernel

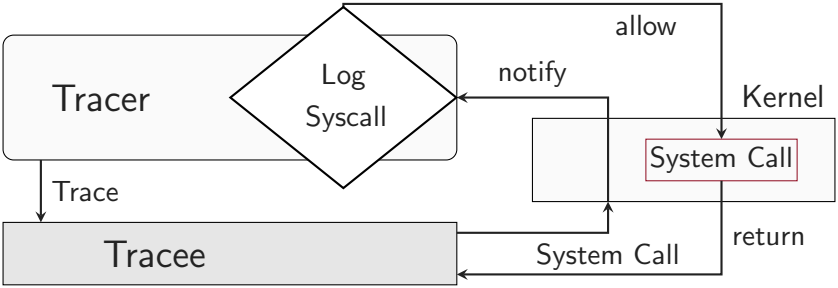

Tracer
Kernel
Trace
Tracee
System Call


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

- libchestnut takes care of setting up seccomp filters

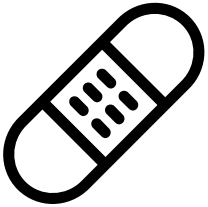
- libchestnut takes care of setting up seccomp filters
- Provides Finalyzer as well

- libchestnut takes care of setting up seccomp filters
- Provides Finalyzer as well
- Contains constructor that runs before *main* function

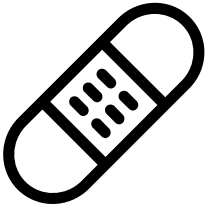
- libchestnut takes care of setting up seccomp filters
- Provides Finalyzer as well
- Contains constructor that runs before *main* function
- Sourcalyzer automatically links it and libseccomp

- libchestnut takes care of setting up seccomp filters
- Provides Finalyzer as well
- Contains constructor that runs before *main* function
- Sourcalyzer automatically links it and libseccomp

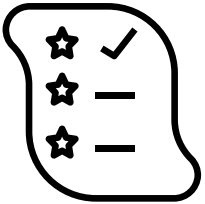
→ What about Binalyzer?



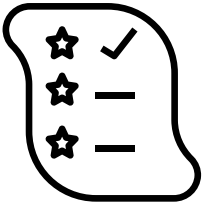
- ChestnutPatcher: directly patch binary



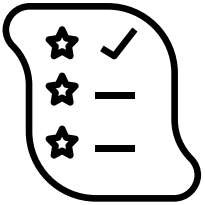
- ChestnutPatcher: directly patch binary
- ChestnutGenerator: create wrapper program that launches binary



- Performance, Functional Correctness, and Security



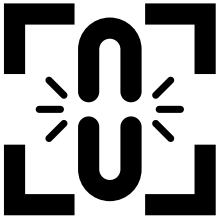
- Performance, Functional Correctness, and Security
- Analyzed Client, Server, and Database applications



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

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

- 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

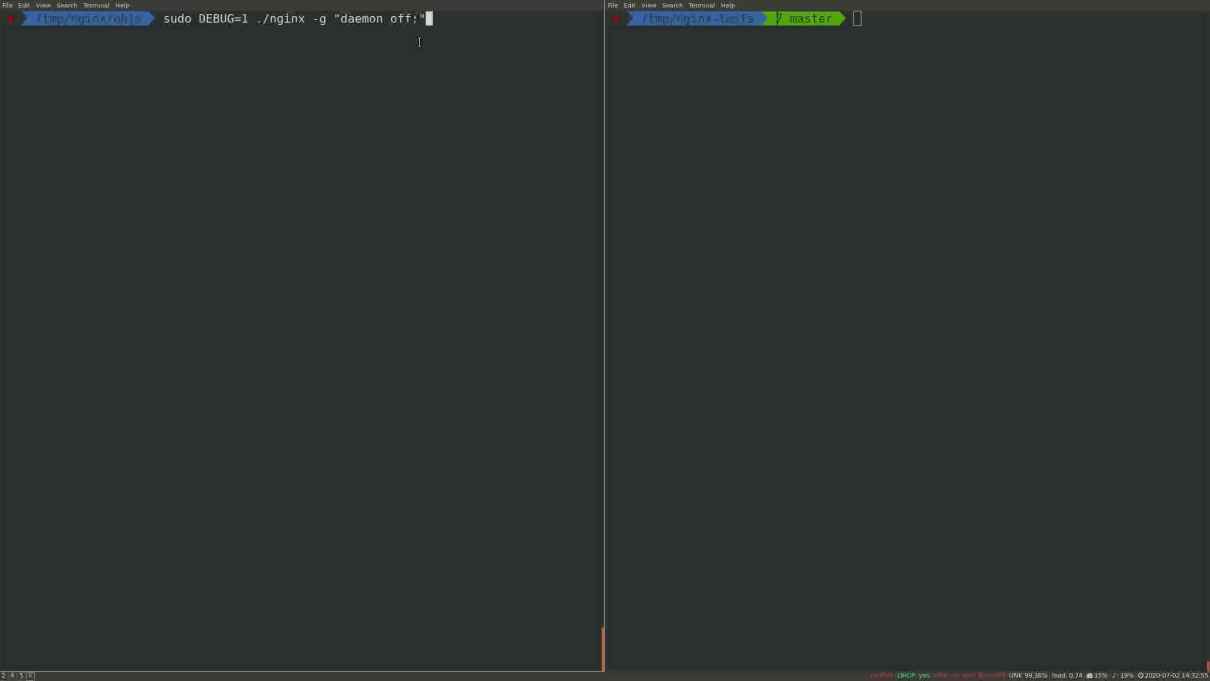
- Use application testsuites for checks

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

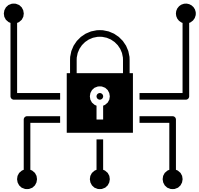
- Use application testsuites for checks
- Code coverage metrics for better estimations of correctness
  - Line coverage: 59-77 %
  - Function coverage: 61-92 %

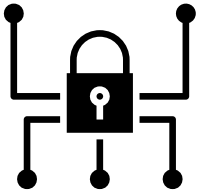
- Use application testsuites for checks
- Code coverage metrics for better estimations of correctness
  - Line coverage: 59-77 %
  - Function coverage: 61-92 %
- Observed no crashes in tests

- 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

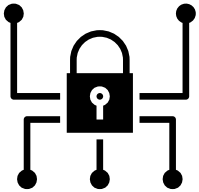

File Edit View Search Terminal Help
/tmp/nginx/objs
sudo DEBUG=1 ./nginx -g "daemon off;"
File Edit View Search Terminal Help
/tmp/nginx-tests
master

- Avg. Number of blocked system calls:

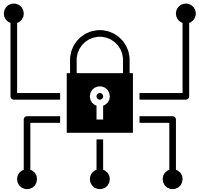




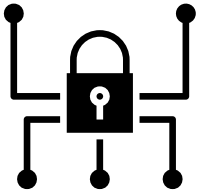
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)



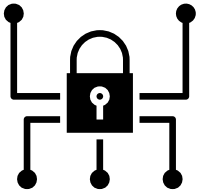
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:



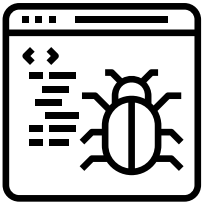
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:
  - : 9 (50 %)
  - : 14 (78 %)



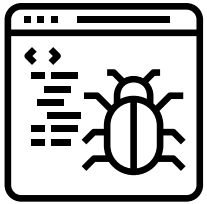
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:
  - : 9 (50 %)
  - : 14 (78 %)
- `mprotect` system calls blocked:



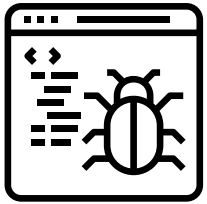
- Avg. Number of blocked system calls:
  - : 302 (87 %)
  - : 288 (83 %)
- `exec` system calls blocked:
  - : 9 (50 %)
  - : 14 (78 %)
- `mprotect` system calls blocked:
  - : 11 (61 %)
  - : 0 (0 %)



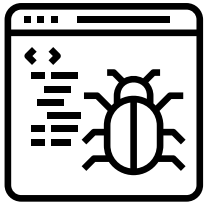
- 175 CVEs extracted from mitre database



- 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 %

- Use exisiting code to exploit a program

- Use exisiting code to exploit a program
- Jumps to parts of functions (so called gadgets)

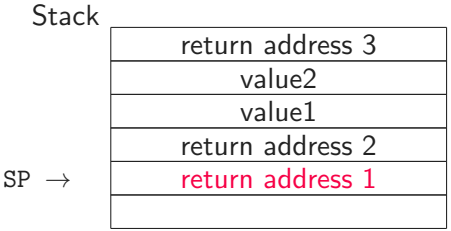
- Use exisiting code to exploit a program
- Jumps to parts of functions (so called gadgets)
- These *gadgets* are assembler instructions followed by a `ret`

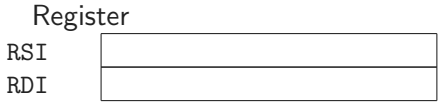
- Use exisiting code to exploit a program
- Jumps to parts of functions (so called gadgets)
- These *gadgets* are assembler instructions followed by a `ret`
  - `pop RDI; retq`

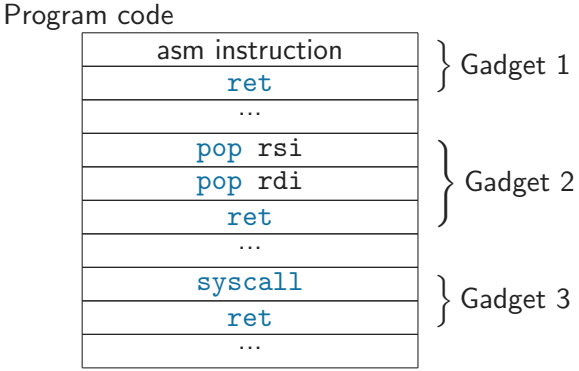
- Use exisiting code to exploit a program
- Jumps to parts of functions (so called gadgets)
- These *gadgets* are assembler instructions followed by a `ret`
  - `pop RDI; retq`
  - `syscall; retq`

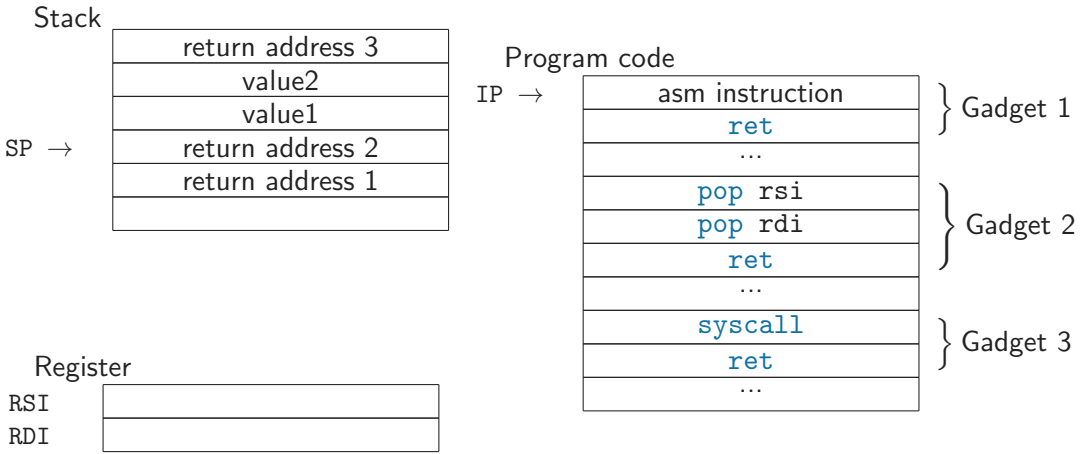
- Use exisiting code to exploit a program
- Jumps to parts of functions (so called gadgets)
- These *gadgets* are assembler instructions followed by a `ret`
  - `pop RDI; retq`
  - `syscall; retq`
  - `add RSP, 8; retq`

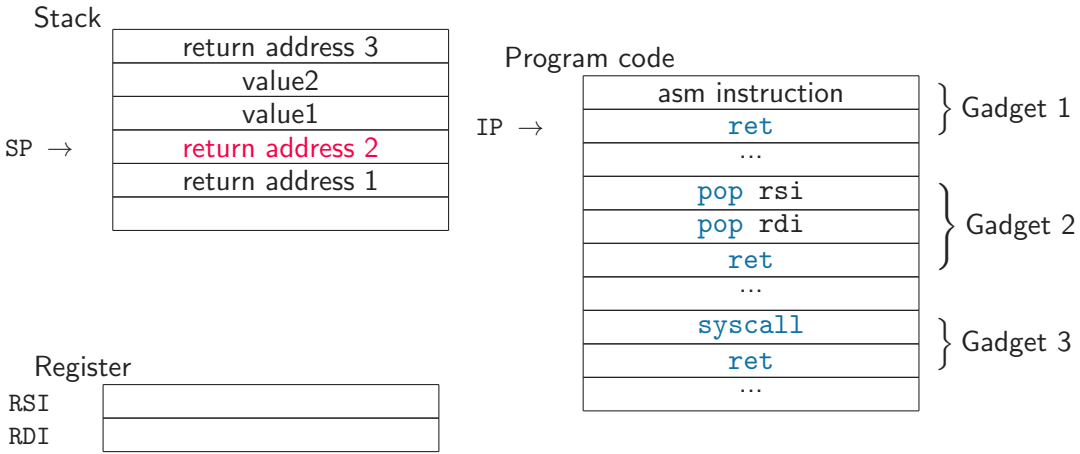
- Use exisiting code to exploit a program
- Jumps to parts of functions (so called gadgets)
- These *gadgets* are assembler instructions followed by a `ret`
  - `pop RDI; retq`
  - `syscall; retq`
  - `add RSP, 8; retq`
- Gadgets are chained together for an exploit
- Overwrite the stack with gadget addresses and parameters

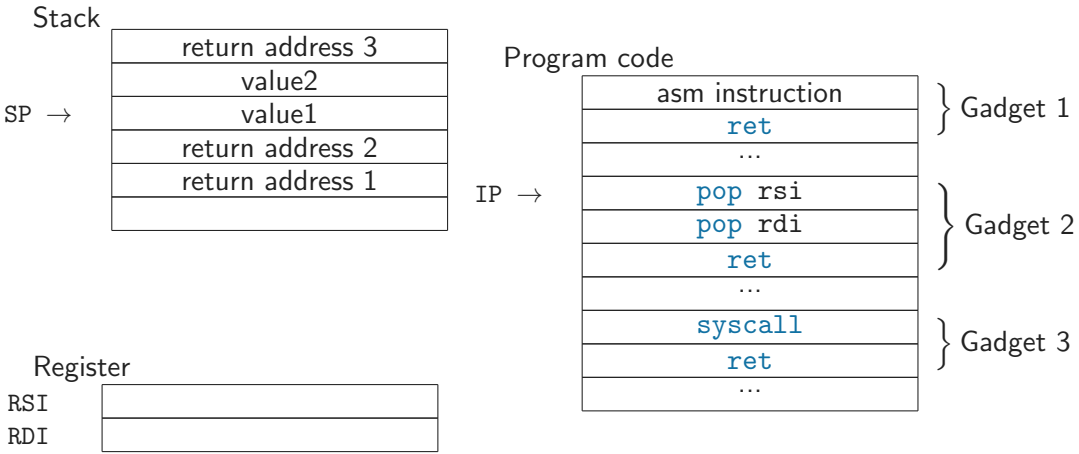

Stack
return address 3
value2
value1
return address 2
SP →
return address 1

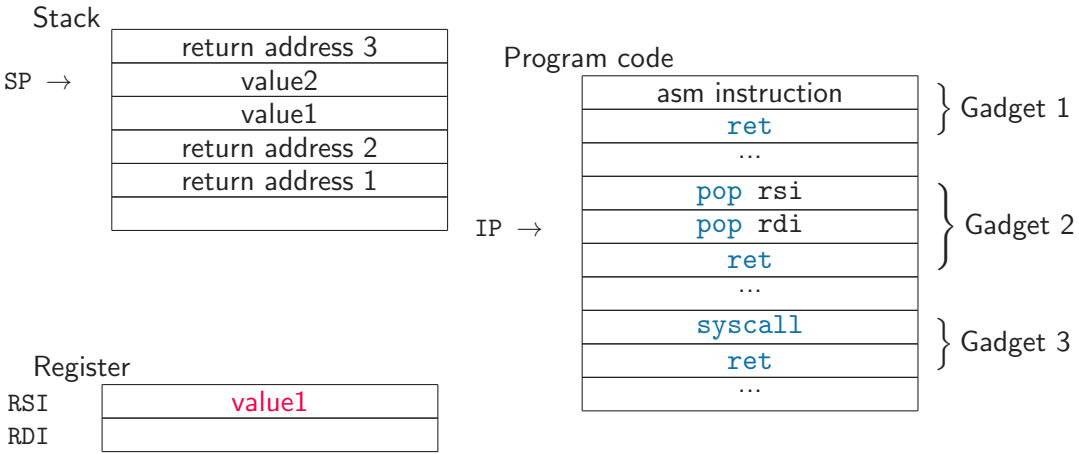

Register
RSI
RDI

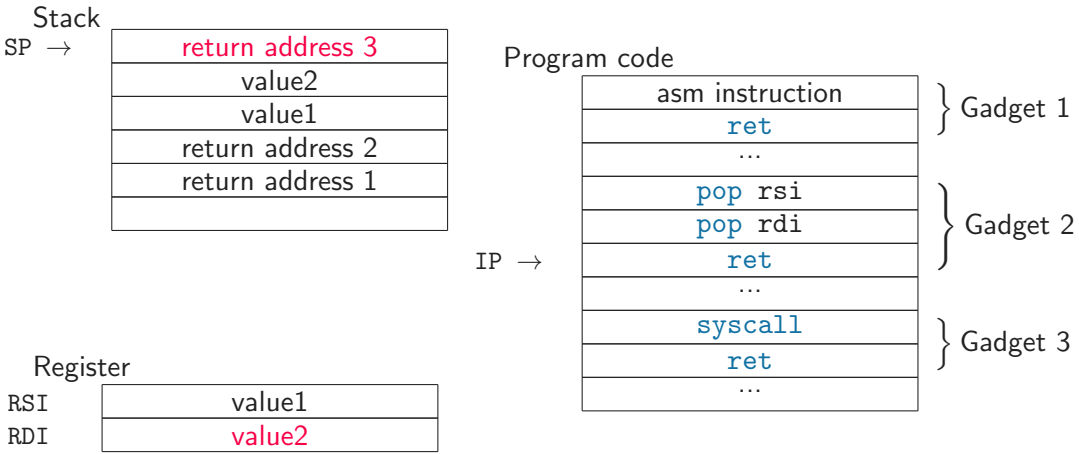

Program code
asm instruction
ret
...
pop rsi
pop rdi
ret
...
syscall
ret
...
Gadget 1
Gadget 2
Gadget 3

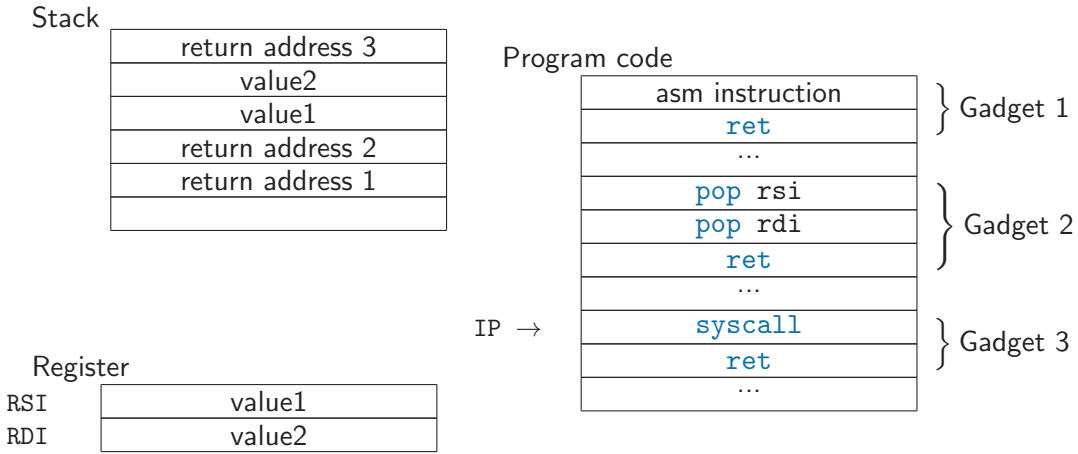

Stack
return address 3
value2
value1
SP →
return address 2
return address 1
Program code
IP →
asm instruction
ret
...
pop rsi
pop rdi
ret
...
syscall
ret
...
Gadget 1
Gadget 2
Gadget 3
Register
RSI
RDI

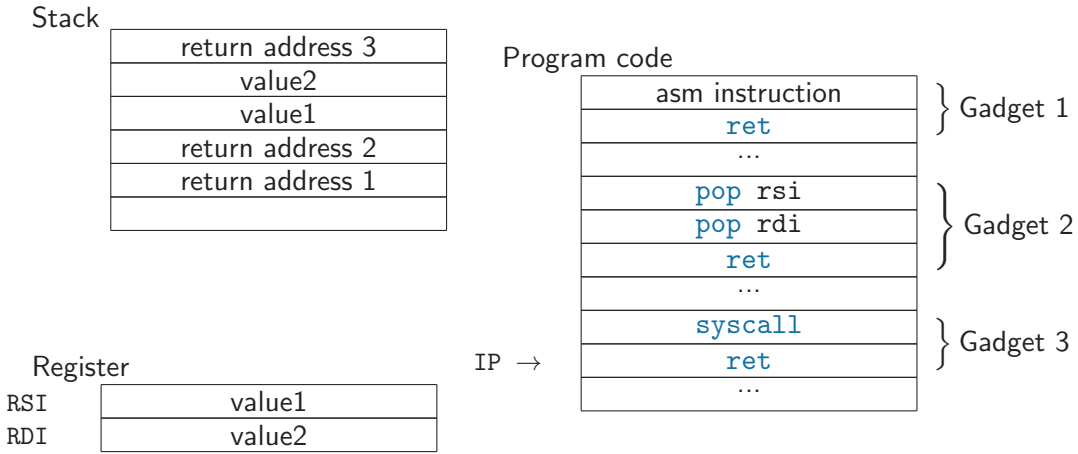

Stack
return address 3
value2
value1
SP →
return address 2
return address 1
Program code
IP →
asm instruction
ret
...
pop rsi
pop rdi
ret
...
syscall
ret
...
Gadget 1
Gadget 2
Gadget 3
Register
RSI
RDI


Stack
SP →
return address 3
value2
value1
return address 2
return address 1
Program code
IP →
asm instruction
ret
...
pop rsi
pop rdi
ret
...
syscall
ret
...
Gadget 1
Gadget 2
Gadget 3
Register
RSI
RDI


Stack
SP →
return address 3
value2
value1
return address 2
return address 1
Program code
IP →
asm instruction
ret
...
pop rsi
pop rdi
ret
...
syscall
ret
...
Gadget 1
Gadget 2
Gadget 3
Register
RSI
RDI
value1


Stack
SP →
return address 3
value2
value1
return address 2
return address 1
Program code
asm instruction
ret
...
pop rsi
pop rdi
ret
...
syscall
ret
...
IP →
Gadget 1
Gadget 2
Gadget 3
Register
RSI
RDI
value1
value2


Stack
return address 3
value2
value1
return address 2
return address 1
Program code
asm instruction
ret
...
pop rsi
pop rdi
ret
...
IP →
syscall
ret
...
Gadget 1
Gadget 2
Gadget 3
Register
RSI
value1
RDI
value2


Stack
return address 3
value2
value1
return address 2
return address 1
Program code
asm instruction
ret
...
pop rsi
pop rdi
ret
...
syscall
ret
...
Gadget 1
Gadget 2
Gadget 3
IP →
Register
RSI
value1
RDI
value2

Gadgets are often unintended

- Consider the byte sequence `05 5a 5e 5f c3`

Gadgets are often unintended

- Consider the byte sequence `05 5a 5e 5f c3`
- It disassembles to

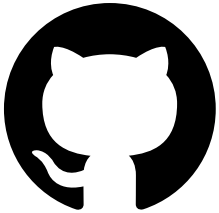
Gadgets are often unintended

- Consider the byte sequence `05 5a 5e 5f c3`
- It disassembles to
  `add eax, 0xc35f5e5a`
- However, if we skip the first byte, it disassembles to

Gadgets are often unintended

- Consider the byte sequence `05 5a 5e 5f c3`
- It disassembles to
  ```
  add eax, 0xc35f5e5a
  ```
- However, if we skip the first byte, it disassembles to
  ```
  pop rdx
  pop rsi
  pop rdi
  ret
  ```
- This property is due to non-aligned, variable-width opcodes

```
/tmp/rop_demo ls
exploit.sh  Makefile  name  name.c  test
/tmp/rop_demo
```



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

- `https://github.com/chestnut-sandbox/Chestnut`

More details in the paper

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

arXiv [Can+20]

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

- Reduced time-consuming, manual analysis to automated process

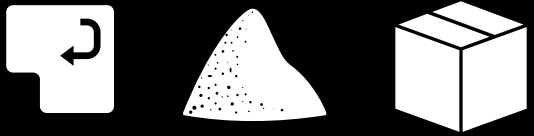
- Reduced time-consuming, manual analysis to automated process
- Showed that we can improve overall system security

- Reduced time-consuming, manual analysis to automated process
- Showed that we can improve overall system security
- Demonstrated functional correctness using testsuites and a long-term study

- Reduced time-consuming, manual analysis to automated process
- Showed that we can improve overall system security
- Demonstrated functional correctness using testsuites and a long-term study
- Chestnut only has small performance impact

black hat

ENTER SANDBOX



https://github.com/chestnut-sandbox/Chestnut

Claudio Canella (@cc0x1f), Mario Werner (we.rner.at), Michael Schwarz (@misc0110)

References

C. Canella, M. Werner, D. Gruss, and M. Schwarz. Automating Seccomp Filter Generation for Linux Applications. In: arXiv:2012.02554 (2020).