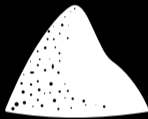


ENTER SANDBOX



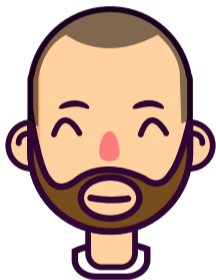


Claudio Canella

PhD candidate @ Graz University of Technology

 @cc0x1f

 claudio.canella@iaik.tugraz.at

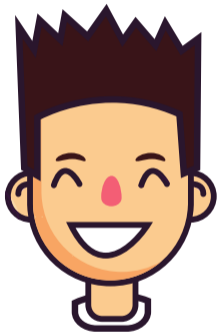


Mario Werner

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

Now: Hardware Design Engineer @ NXP Semiconductors

 <https://we.rner.at/>



Michael Schwarz

Faculty @ CISA Helmholtz Center for Information Security

🐦 @misc0110

✉ michael.schwarz@cisa.saarland



- **Memory safety vulnerabilities** are common



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



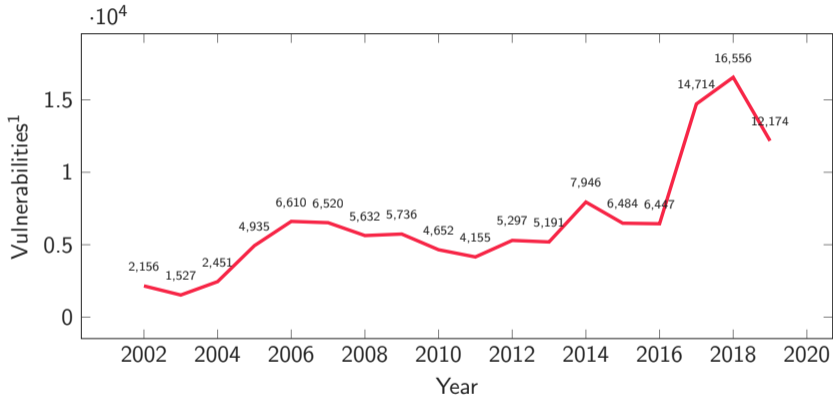
- Memory safety vulnerabilities are common
- Sandboxing helps in limiting their impact
- Linux seccomp: works but hard to do



- **Memory safety vulnerabilities** are common
 - Sandboxing helps in **limiting** their **impact**
 - Linux seccomp: works but hard to do
- Can we **automate** seccomp sandboxing?

A few months ago...

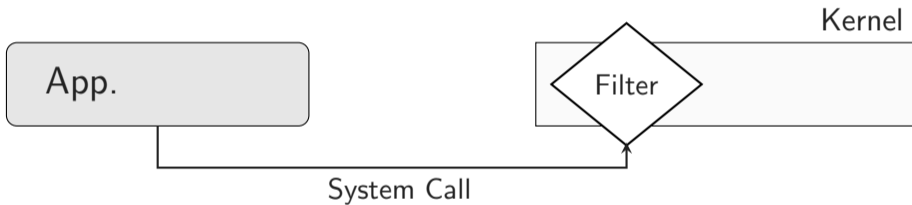
```
1 int main(int argc, char* argv[]) {
2     int infd, outfd;
3     ssize_t read_bytes;
4     char buffer[1024];
5
6     printf("Copying '%s' to '%s'\n", argv[1], argv[2]);
7     if((infd = open(argv[1], O_RDONLY)) > 0) {
8         if((outfd = open(argv[2], O_WRONLY | O_CREAT, 0644)) > 0) {
9             while((read_bytes = read(infd, &buffer, 1024)) > 0)
10                write(outfd, &buffer, (ssize_t)read_bytes);
11         }
12     }
13     close(infd);
14     close(outfd);
15     return 0;
16 }
```

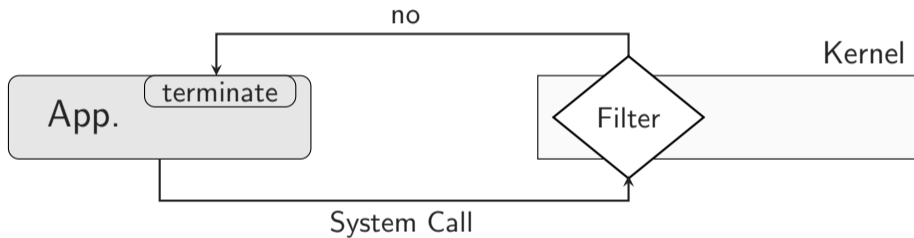


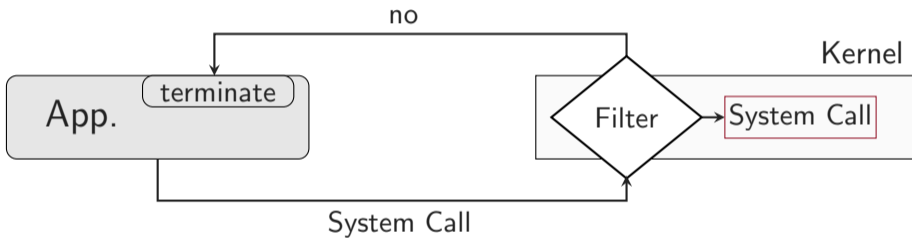
¹Source: <http://www.cvedetails.com/vulnerabilities-by-types.php>

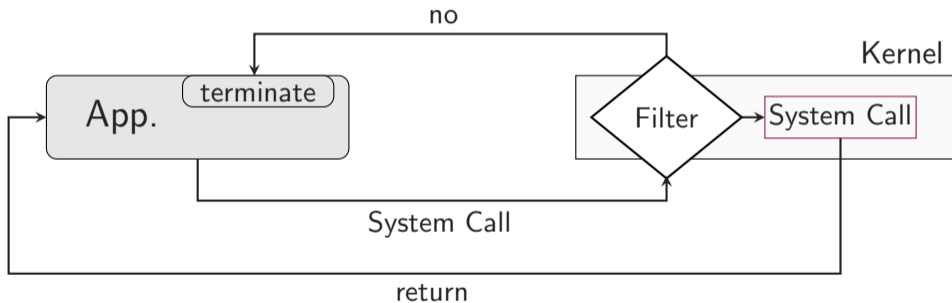












```
1 struct sock_filter filter [] = {
2   BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, nr))),
3   BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_write, 0, 1),
4   BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
5   BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_open, 0, 3),
6   BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, args[1]))),
7   BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 0_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);
```

```
1 prctl(PR_SET_NO_NEW_PRIVS, 1);
2 prctl(PR_SET_DUMPABLE, 0);
3 scmp_filter_ctx ctx;
4 ctx = seccomp_init(SCMP_ACT_KILL);
5 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(rt_sigreturn), 0);
6 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
7 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
8 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0);
9 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(openat), 1,
10   SCMP_A2(SCMP_CMP_EQ, 0_RDONLY));
11 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

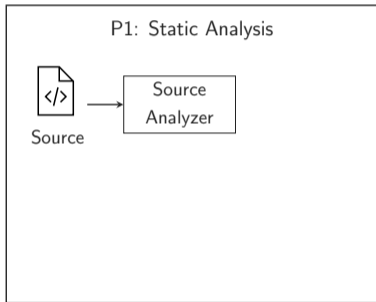
We're off to never exploit-land

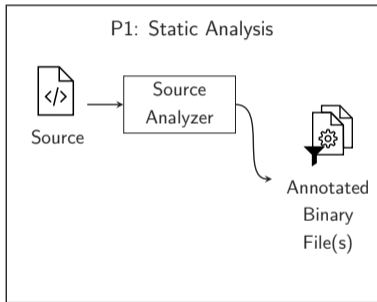
P1: Static Analysis

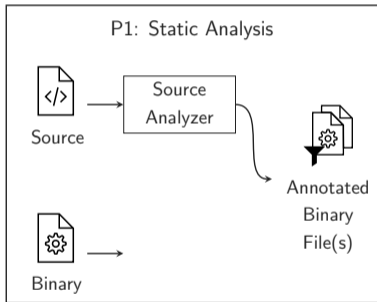
P1: Static Analysis

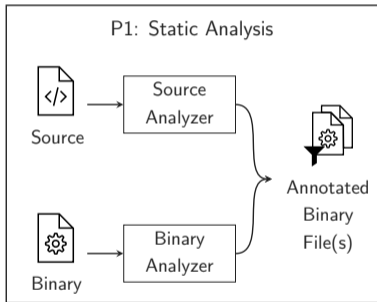


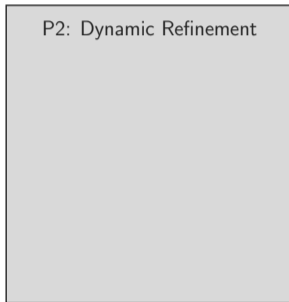
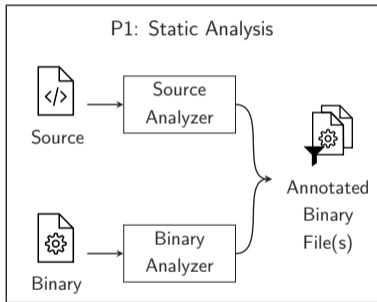
Source

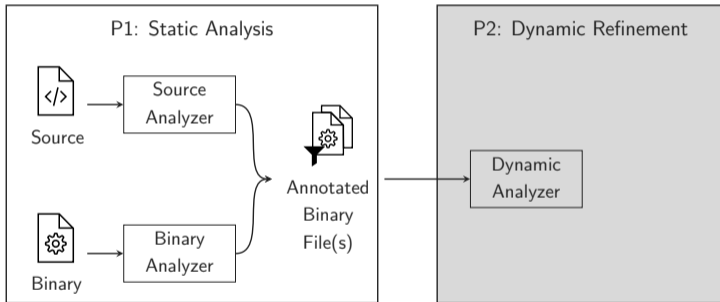


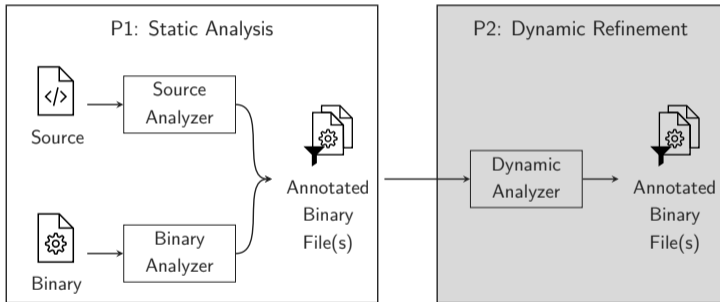


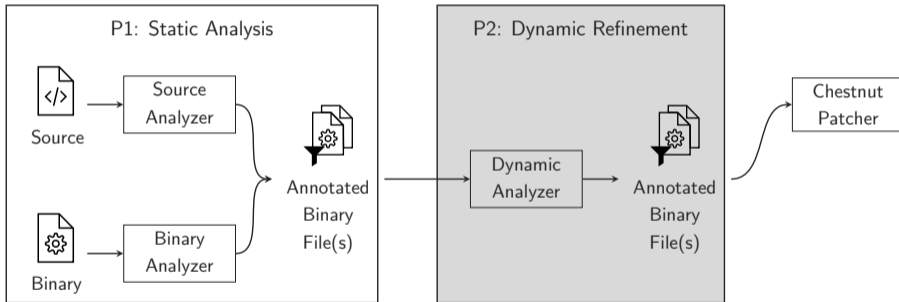


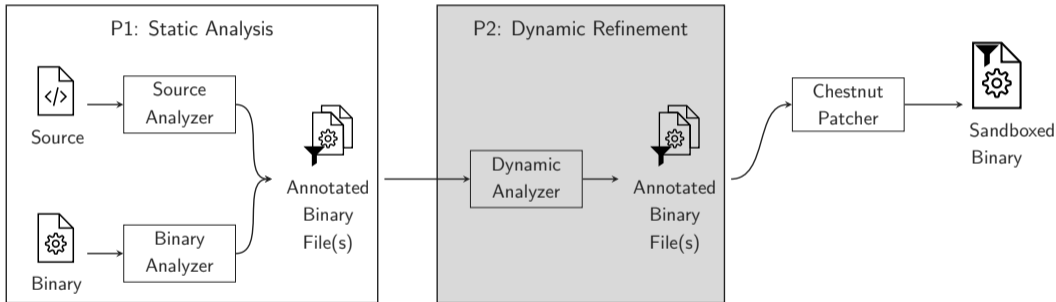


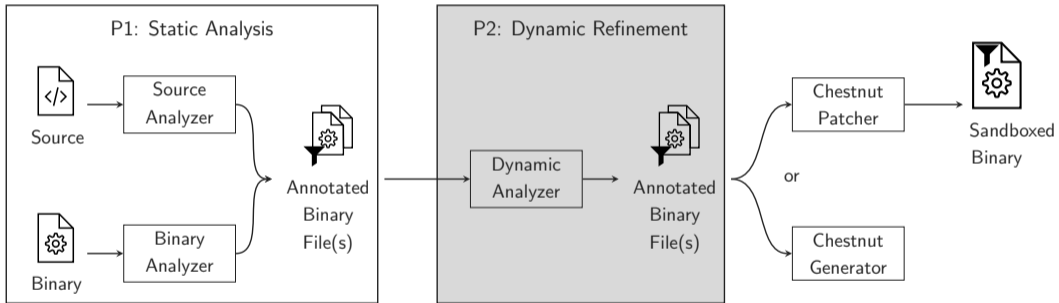


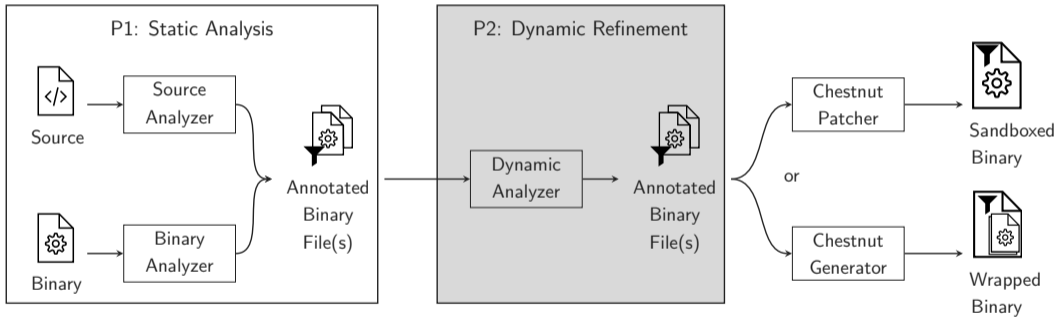






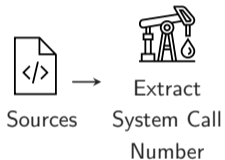


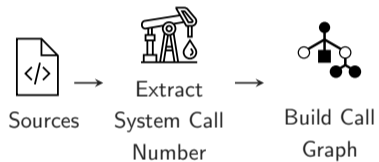


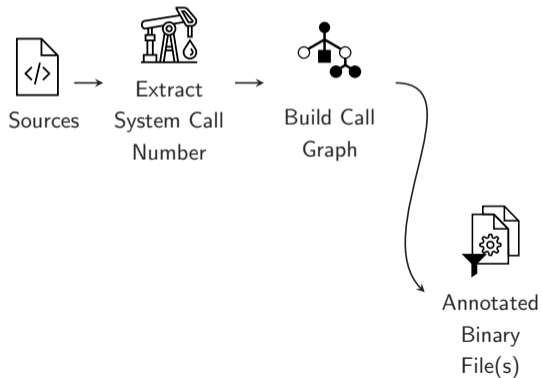


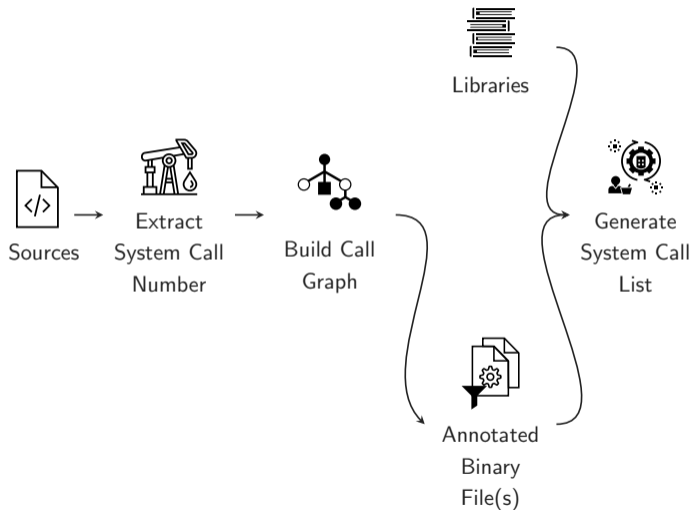


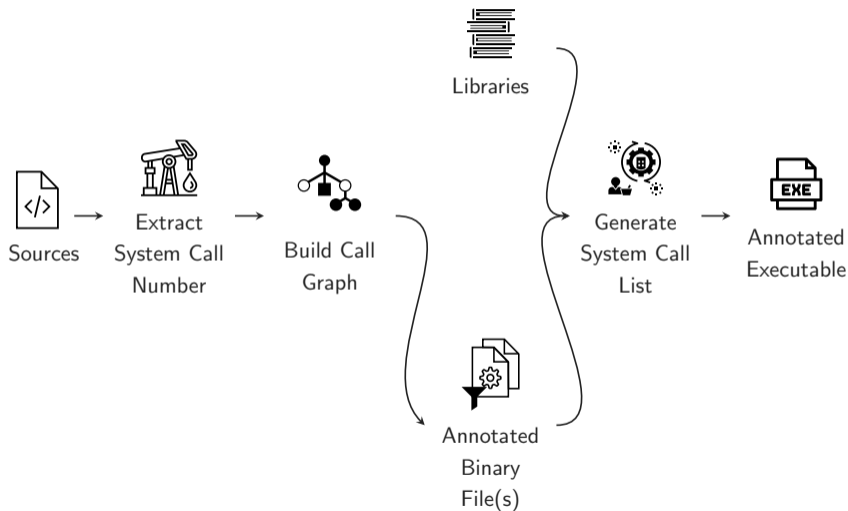
Sources

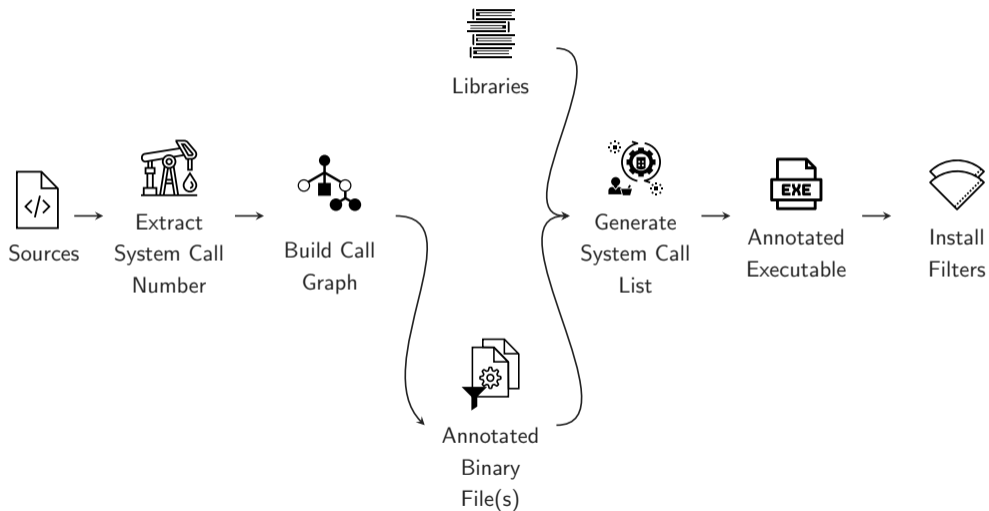






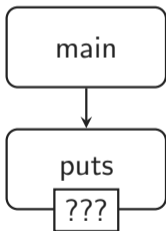






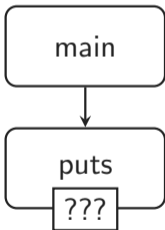

```
#include <stdio.h>
int main() {
    puts("Hello World!");
}
```

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



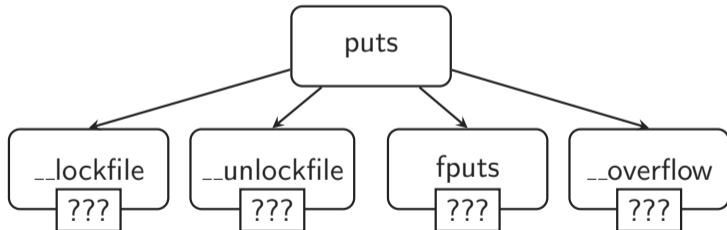
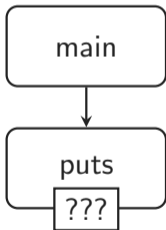
```
#include <stdio.h>
int main() {
    puts("Hello World!");
}
```

```
// 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;
}
```




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

```
// 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;
}
```

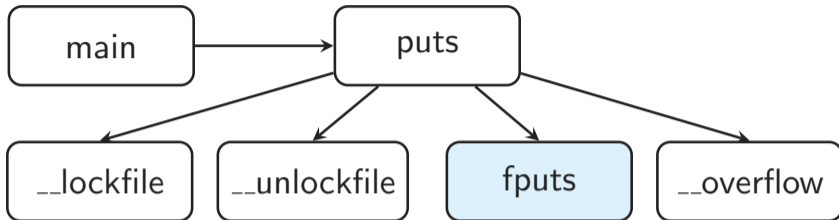


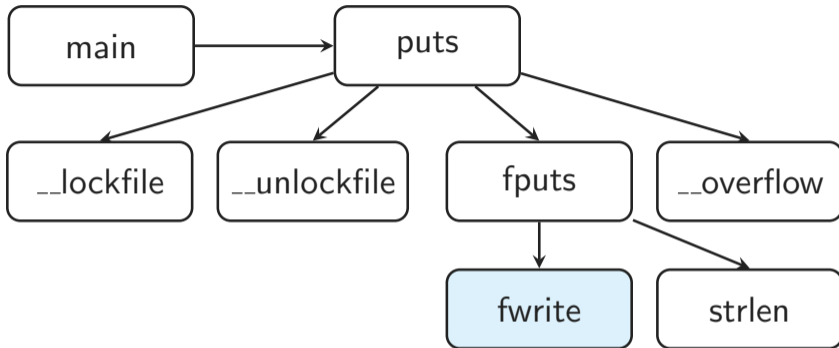
```
#include<stdio.h>
int main() {
    puts("Hello World!");
}
```

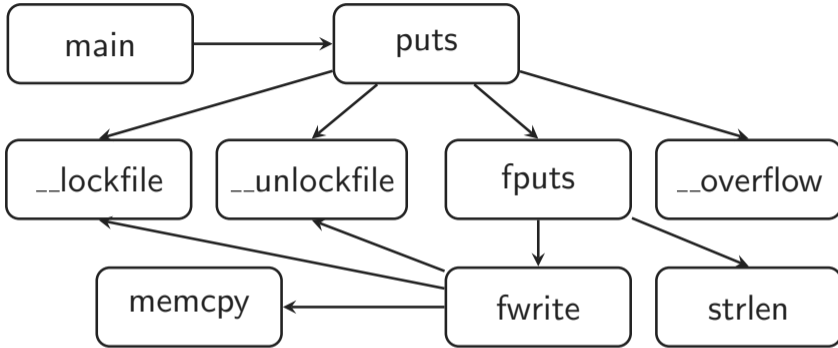
```
// musl/src/stdio/puts.c
int puts(const char *s) {
    int r; FLOCK(stdout);
    r = -(fputs(s, stdout) < 0 ||
        putc_unlocked('\n', stdout) < 0);
    FUNLOCK(stdout); return r;
}
```

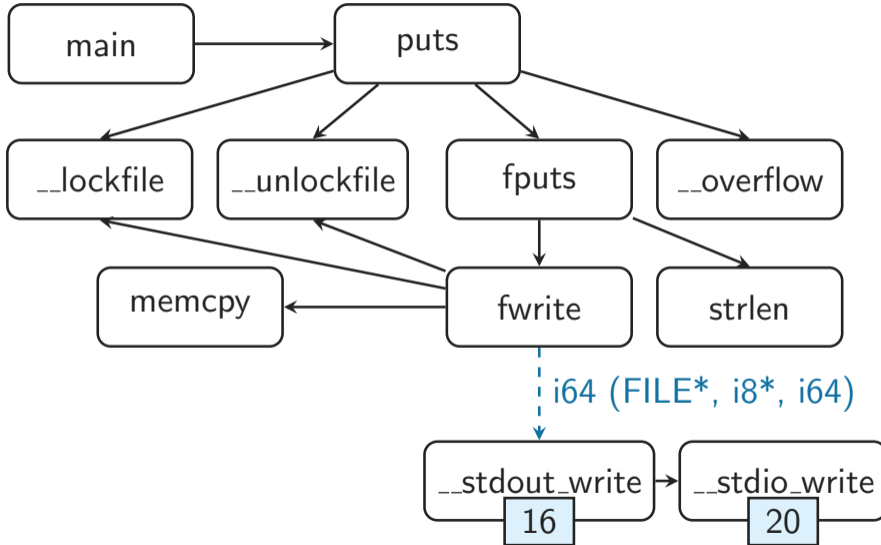
```
{"call_targets": ["__lockfile",
                 "fputs",
                 "__overflow",
                 "__unlockfile"],
 "name": "puts",
 "type": "i32 (i8*)"}
```

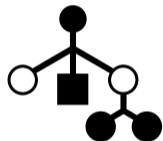




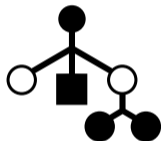








- Extract and serialize meta data during compilation:
 - Function names with signatures
 - Partial call graphs (direct calls)
 - Indirectly called function signatures
 - *hasAddressTaken* information
 - Syscall information



- 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
```

I



- 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
```



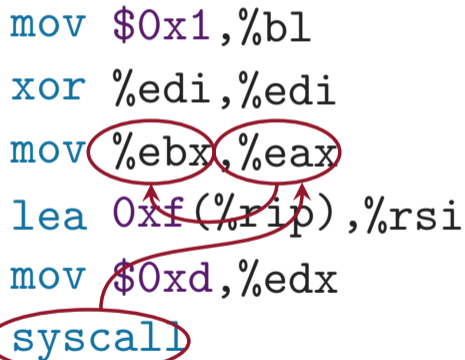
```
rax = ?  
rax = ?  
rax = ?
```

```
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
```



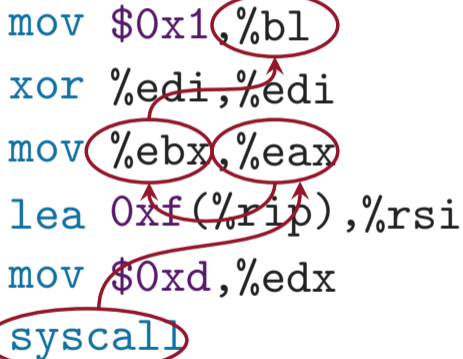
```
rax = rbx = ?
rax = ?
rax = ?
rax = ?
```

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



```
rax = rbx = ?  
rax = rbx = ?  
rax = ?  
rax = ?  
rax = ?
```

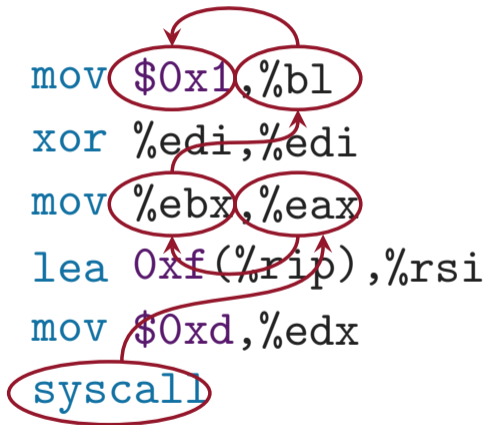
```
mov $0x1,%bl
xor %edi,%edi
mov %ebx,%eax
lea 0xf(%rip),%rsi
mov $0xd,%edx
syscall
```



↑

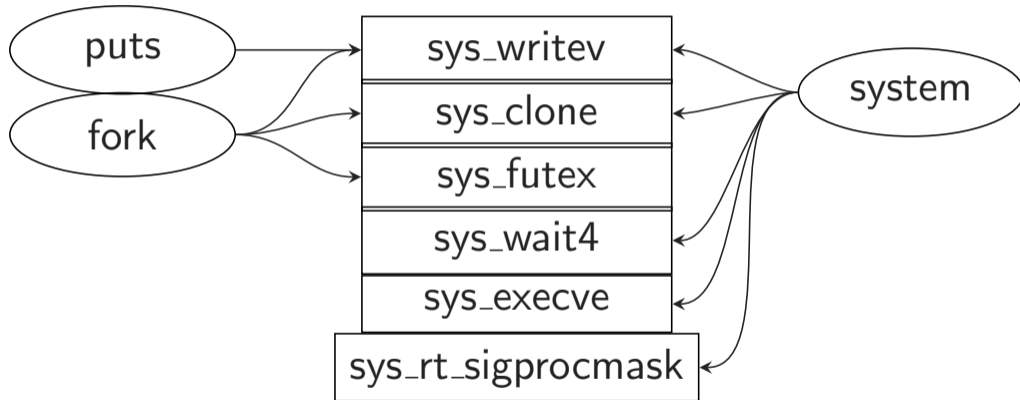
```
rax = rbx = ?
rax = rbx = ?
rax = ?
rax = ?
rax = ?
```

```
mov $0x1,%b1  
xor %edi,%edi  
mov %ebx,%eax  
lea 0xf(%rip),%rsi  
mov $0xd,%edx  
syscall
```



↑

```
rax = rbx = $0x1  
rax = rbx = ?  
rax = rbx = ?  
rax = ?  
rax = ?  
rax = ?
```




```
~/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



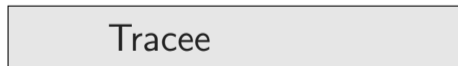
- **Strace-like** system
- Dynamically trace system calls

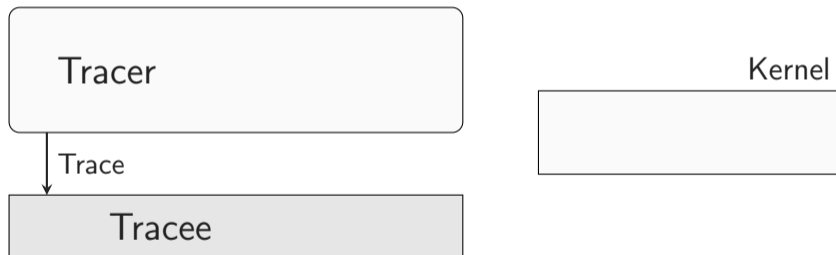


- **Strace-like** system
- Dynamically trace system calls
- Automatically add **missed system calls** or optionally remove **never-used** ones

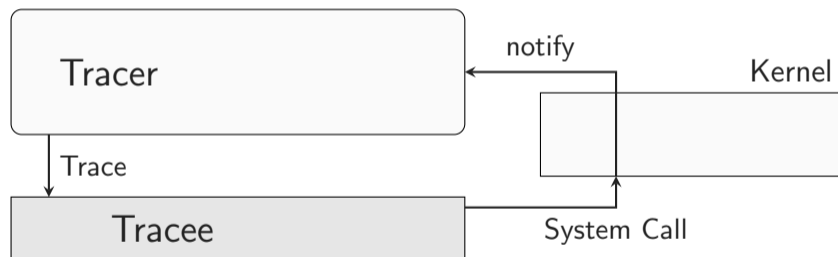
Tracee

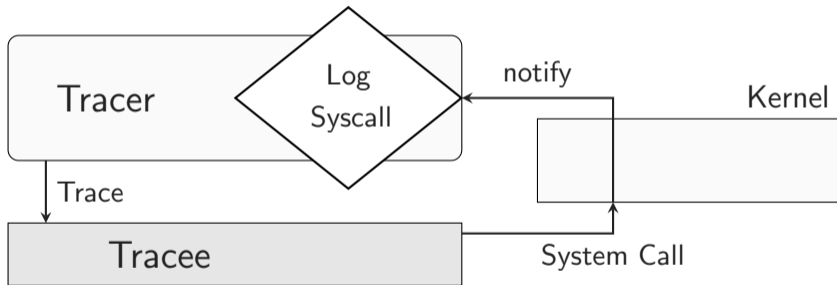
Kernel

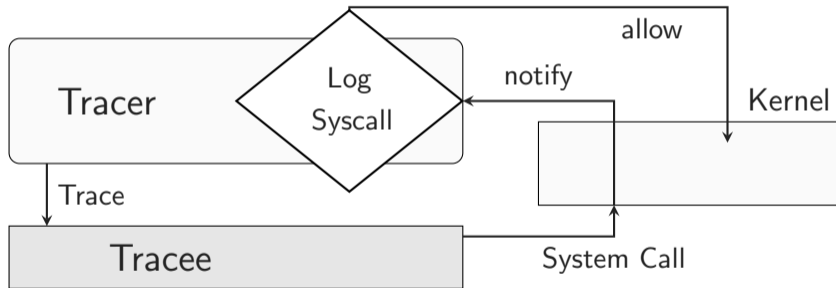


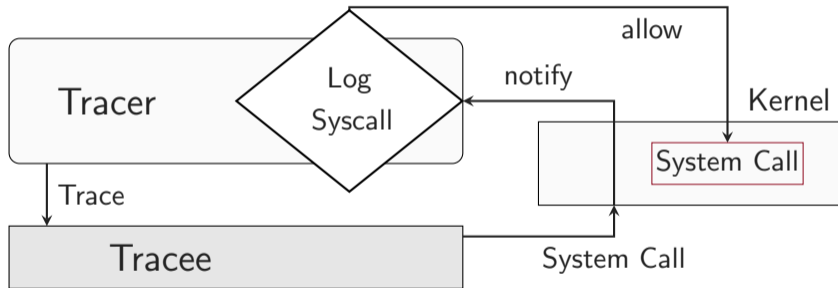


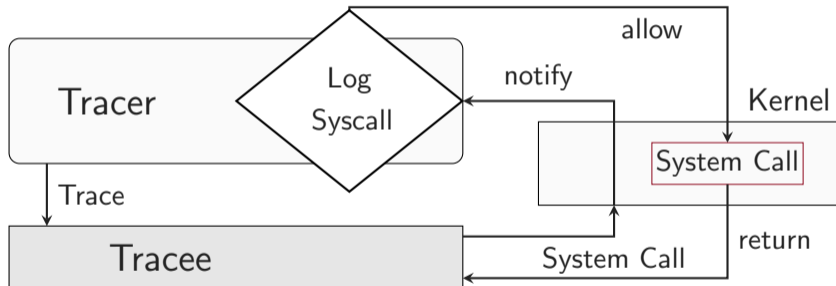














- libchestnut takes care of **setting up seccomp filters**



- 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**



- 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**


```
x ~/tmp/nginx/ohjs sudo DEBUG=1 ./nginx -g "daemon off;"
```

```
|
```

```
x ~/tmp/nginx-logs 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 %)

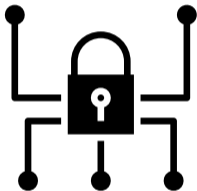
: **288** (83 %)

- **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:

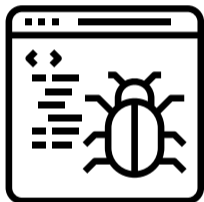
 : 9 (50 %)

 : 14 (78 %)

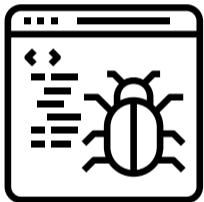
- **mprotect** system calls blocked:

 : 11 (61 %)

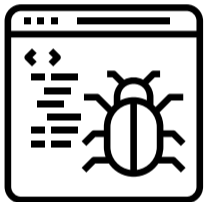
 : 0 (0 %)




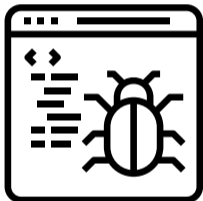
- 175 CVEs extracted from **mitre database**







- 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 existing code to exploit a program





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



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



- Use existing 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`



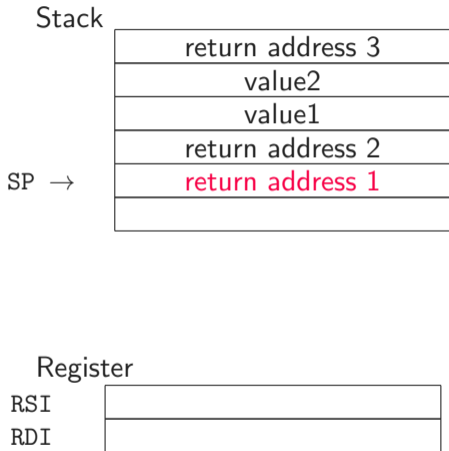
- Use existing code to exploit a program
- Jumps to parts of functions (so called **gadgets**)
- These *gadgets* are assembler **instructions followed by a ret**
 - `pop RDI; retq`
 - `syscall; retq`



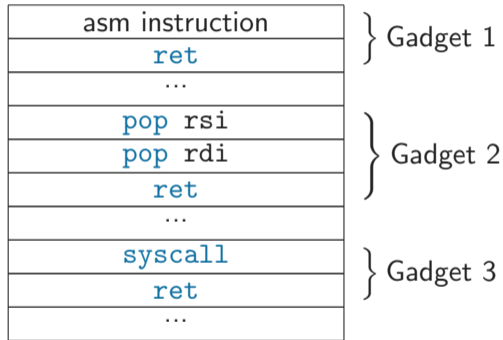
- Use existing code to exploit a program
- Jumps to parts of functions (so called **gadgets**)
- These *gadgets* are assembler **instructions followed by a ret**
 - `pop RDI; retq`
 - `syscall; retq`
 - `add RSP, 8; retq`

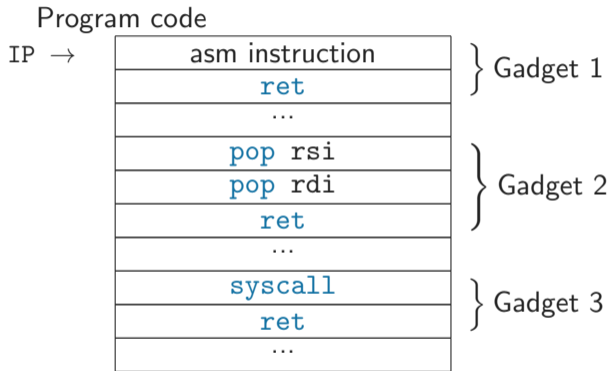
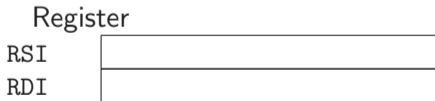
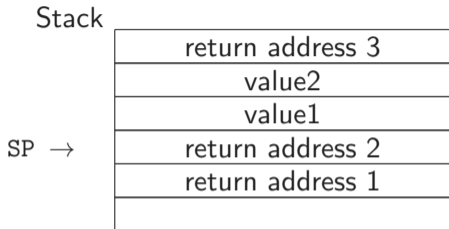


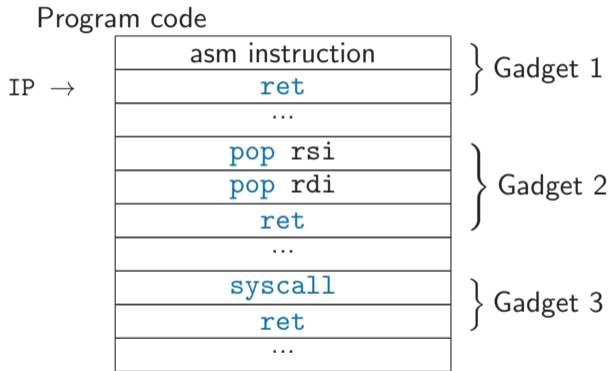
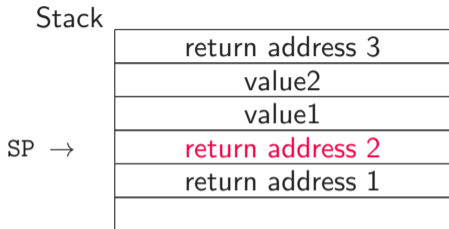
- Use existing code to exploit a program
- Jumps to parts of functions (so called **gadgets**)
- These *gadgets* are assembler **instructions followed by a ret**
 - `pop RDI; retq`
 - `syscall; retq`
 - `add RSP, 8; retq`
- Gadgets are chained together for an exploit
- Overwrite the **stack** with **gadget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

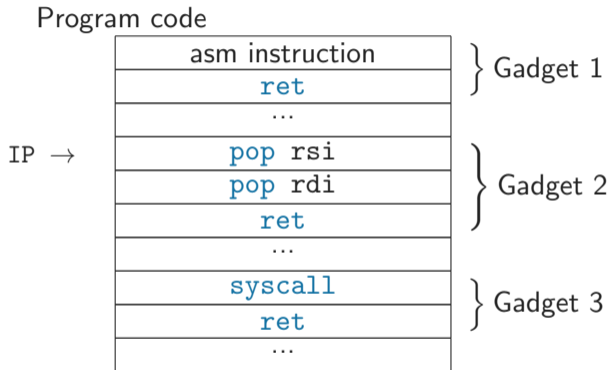
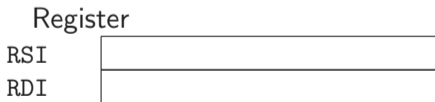
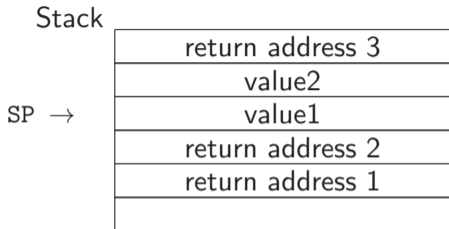


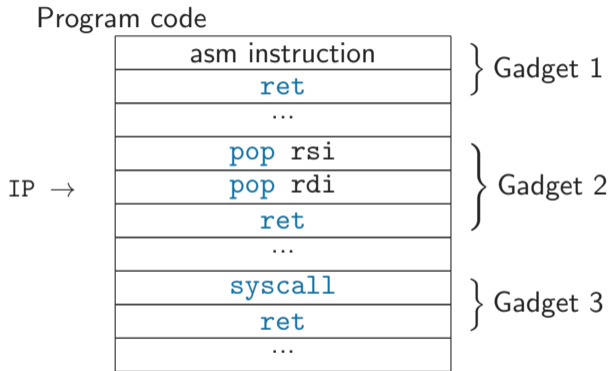
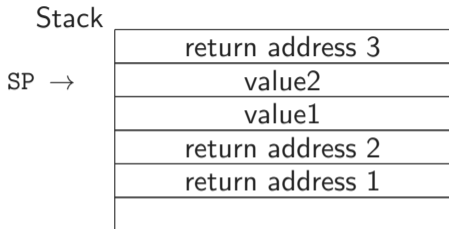
Program code

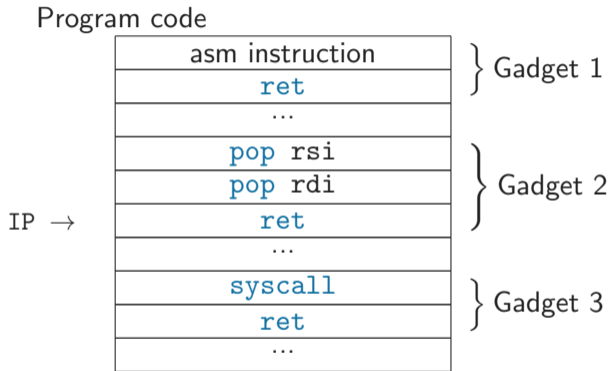
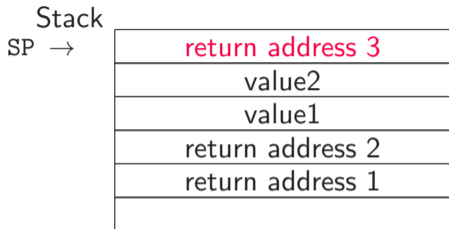












Stack

return address 3
value2
value1
return address 2
return address 1

Register

RSI	value1
RDI	value2

Program code

asm instruction	} Gadget 1
<code>ret</code>	
...	} Gadget 2
<code>pop rsi</code>	
<code>pop rdi</code>	
<code>ret</code>	
...	} Gadget 3
<code>syscall</code>	
<code>ret</code>	
...	

IP →

Stack

return address 3
value2
value1
return address 2
return address 1

Register

RSI	value1
RDI	value2

Program code

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

IP →

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**

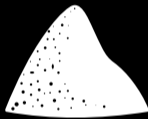



- Reduced time-consuming, manual analysis to **automated process**
- Showed that we can **improve** overall **system security**
- Demonstrated functional correctness using **testsuites and a long-term study**



- 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**

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).