How we almost secured our projects by writing more tests

Alessio Greggi – **Conf42** 2024

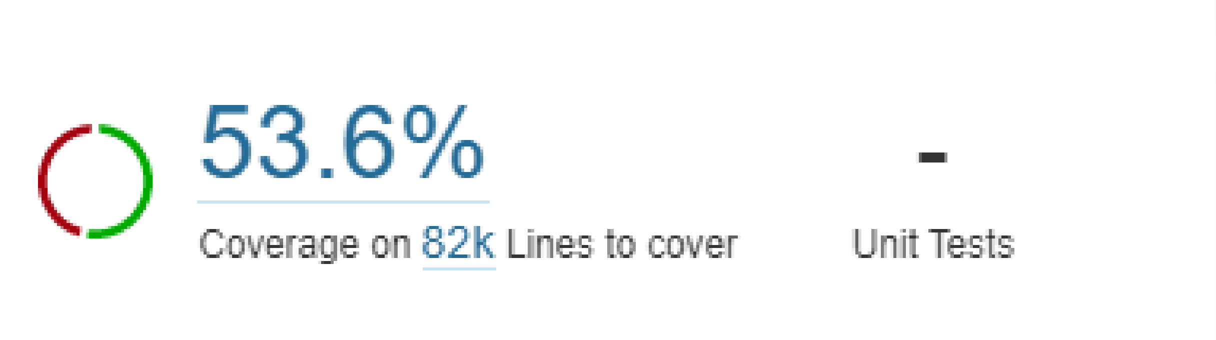
$ whoami

- Alessio Greggi
- Software Engineer
- Cat food opener for my furry friend

- Passionate about reading and taking long walks
-
```
$ cat {github,linkedin,twitter}.com | uniq
alegrey91
```

What is Code Coverage

- A metric that can help you understand how much of your source is tested
- Mostly used when writing unit-tests



53.6%

Coverage on 82k Lines to cover

-

Unit Tests

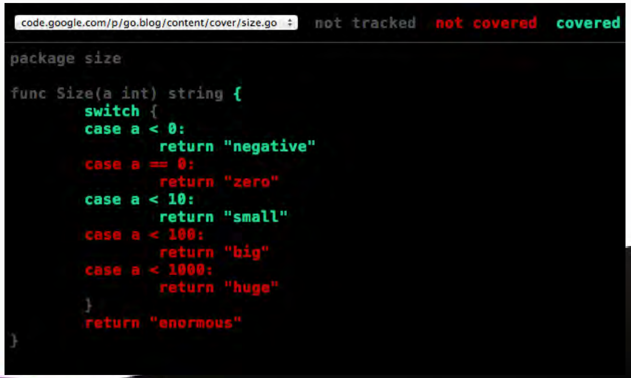
Code Coverage with Go

- First time introduced in version 1.2 for **unit-tests**

  https://tip.golang.org/doc/go1.2#cover
- The story continues with version 1.20 with support for **integration-tests** https://go.dev/blog/integration-test-coverage
- Sensitively increased coverage percentage of projects

```
go test -coverprofile=coverage.out -cover -v ./...

go tool cover -html=coverage.out -o coverage.html
```

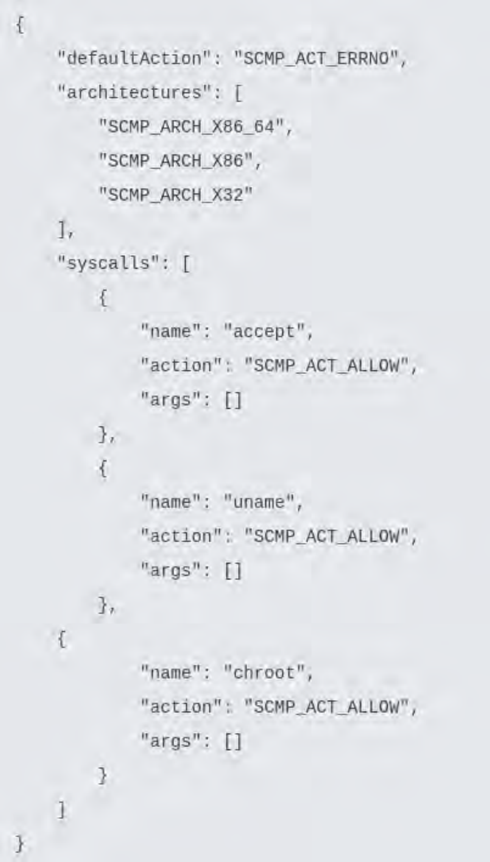


```
package size

func Size(a int) string {
        switch {
        case a < 0:
                return "negative"
        case a == 0:
                return "zero"
        case a < 10:
                return "small"
        case a < 100:
                return "big"
        case a < 1000:
                return "huge"
        }
        return "enormous"
}
```

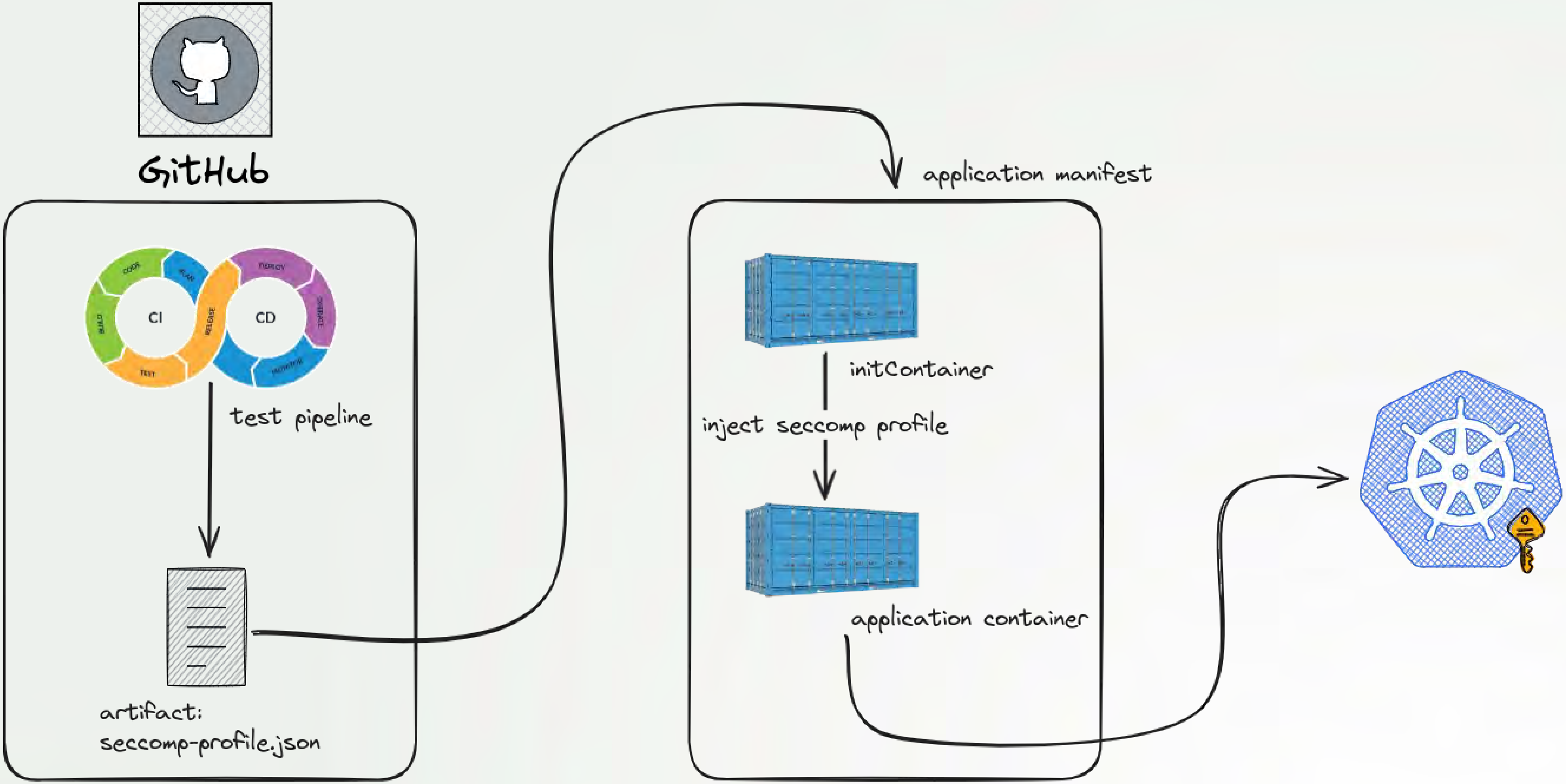
What is a Seccomp Profile

- It’s a security feature of the Linux kernel
- Rules are defined in a file and referred to as a seccomp profile
- Extensively used in the **Kubernetes** ecosystem (default profile)



```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
        "SCMP_ARCH_X86_64",
        "SCMP_ARCH_X86",
        "SCMP_ARCH_X32"
    ],
    "syscalls": [
        {
            "name": "accept",
            "action": "SCMP_ACT_ALLOW",
            "args": []
        },
        {
            "name": "uname",
            "action": "SCMP_ACT_ALLOW",
            "args": []
        },
    {
            "name": "chroot",
            "action": "SCMP_ACT_ALLOW",
            "args": []
        }
    ]
}
```

Seccomp profile as artifact



GitHub

test pipeline

artifact:
seccomp-profile.json

application manifest

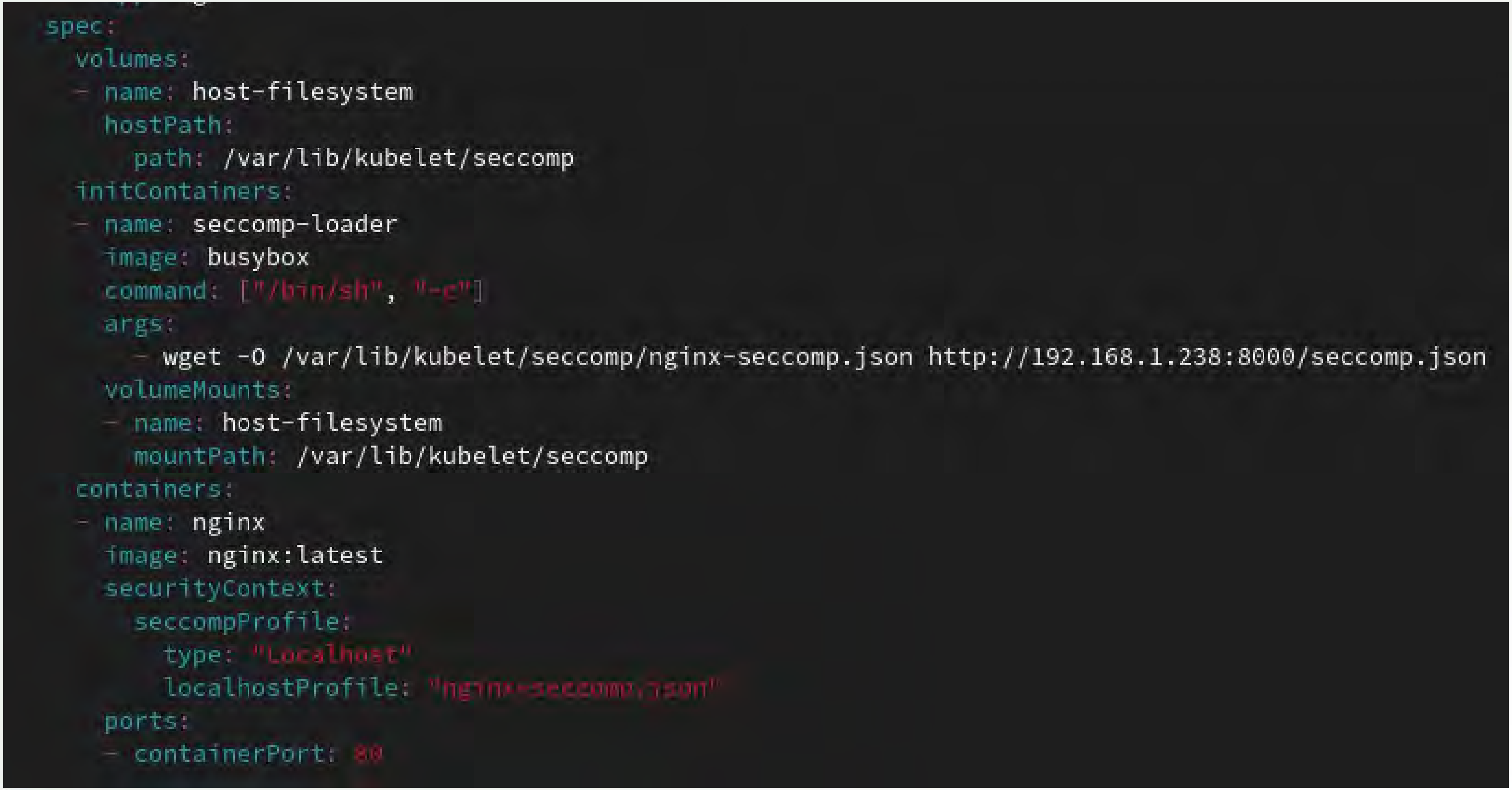
initContainer

inject seccomp profile

application container

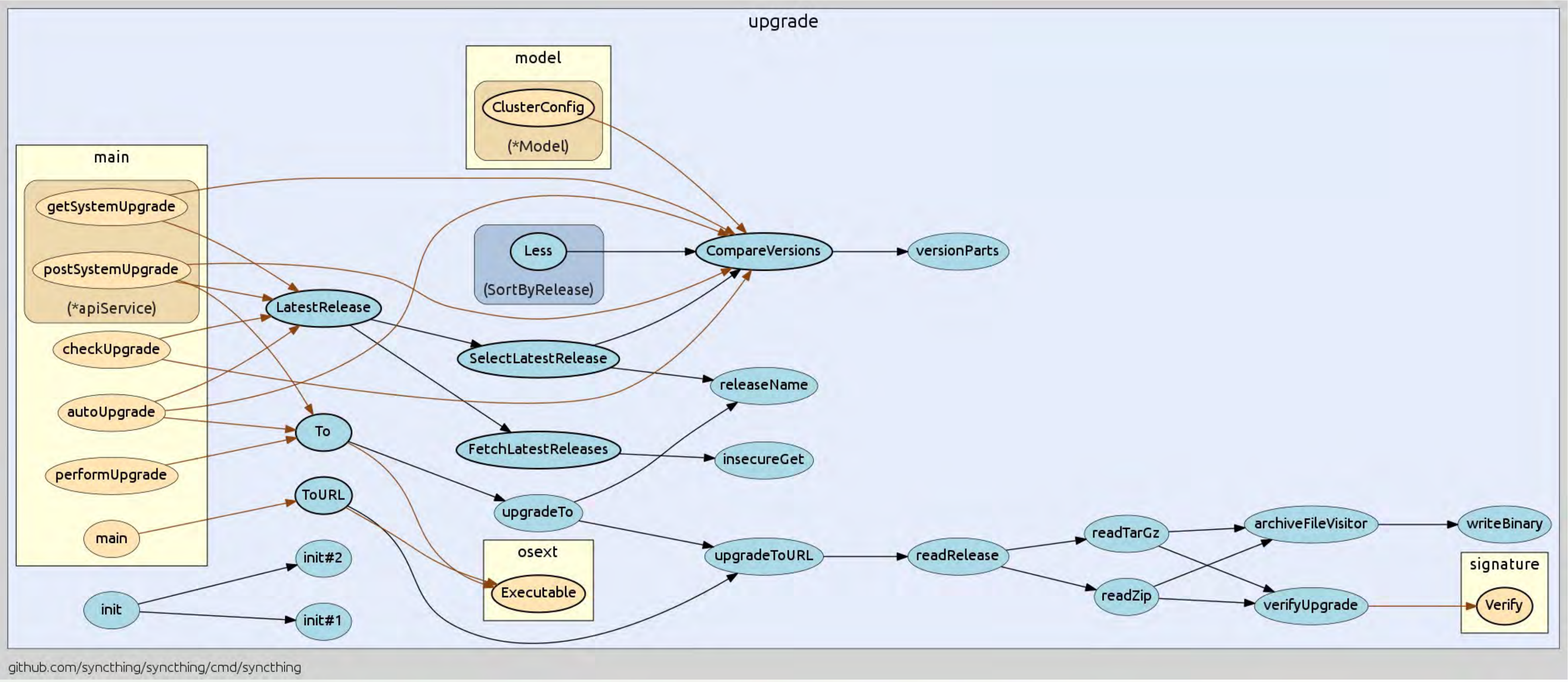
```
securityContext:
  seccompProfile:
    type: Localhost
    localhostProfile: "path/to/injected.json"
```

Seccomp profile as artifact



```yaml
spec:
  volumes:
  - name: host-filesystem
    hostPath:
      path: /var/lib/kubelet/seccomp
  initContainers:
  - name: seccomp-loader
    image: busybox
    command: ["/bin/sh", "-c"]
    args:
      - wget -O /var/lib/kubelet/seccomp/nginx-seccomp.json http://192.168.1.238:8000/seccomp.json
    volumeMounts:
    - name: host-filesystem
      mountPath: /var/lib/kubelet/seccomp
  containers:
  - name: nginx
    image: nginx:latest
    securityContext:
      seccompProfile:
        type: "Localhost"
        localhostProfile: "nginx-seccomp.json"
    ports:
    - containerPort: 80
```

Extracting the syscalls



Credits: Go-callvis
https://github.com/ondrajz/go-callvis/tree/master/examples

Extracting the syscalls (integration-tests)

- Build the binary
- Provide scripts that check for expected results
- Run the binary along with some tracing tool (**strace/perf/...**)
- Collecting executed syscalls
- This allow us to collect most of the syscalls used in the program

Extracting the syscalls (unit-tests)

- A bit more complicated..
- `go test` command **compile** and **run** the test binary all at once
  (no `strace go test .`)
- The test binary could include “noise” not related to our syscalls
  (no `strace ./test-binary`)

Harpoon

- Idea: use **eBPF** to define a tracepoint that starts when a `uprobe` attached to the function is triggered and stops when the `uretprobe` returns
- Previously `iovisor/gobpf` (`bcc` project)
- Currently using `aquasecurity/libbpfgo`

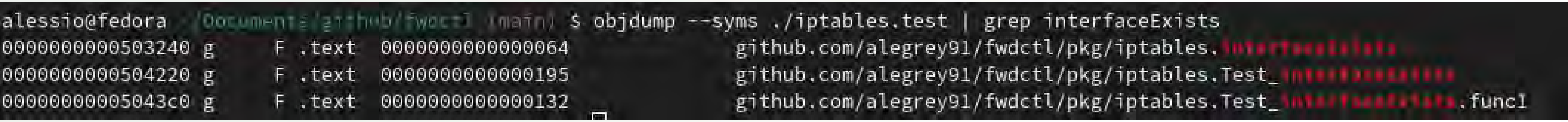
Harpoon

- Build the test binary first:

```
go test -c ./pkg/example
```

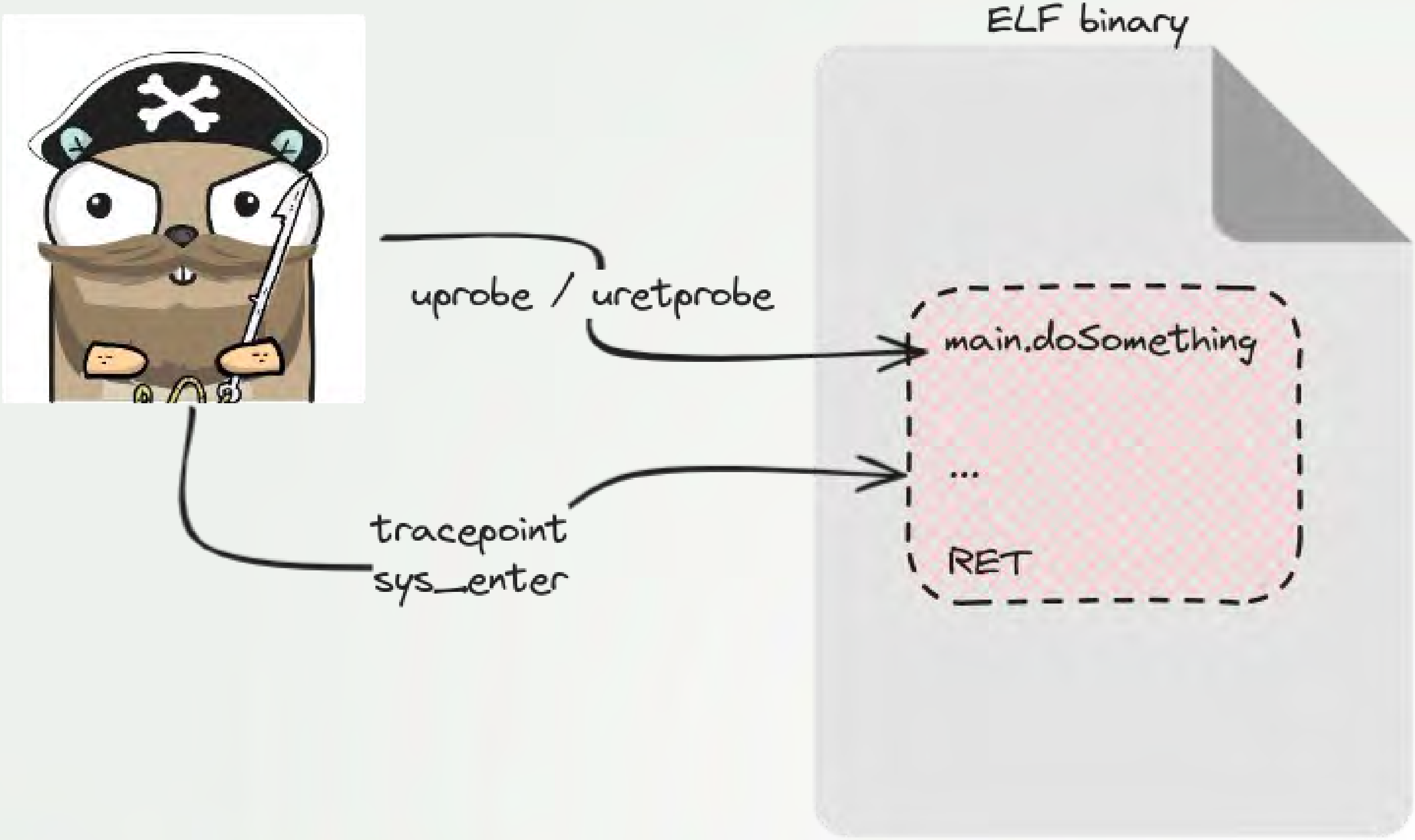
- Search for the function symbol name within the test binary:

```
objdump --syms ./binary.test | grep
myFunction
```



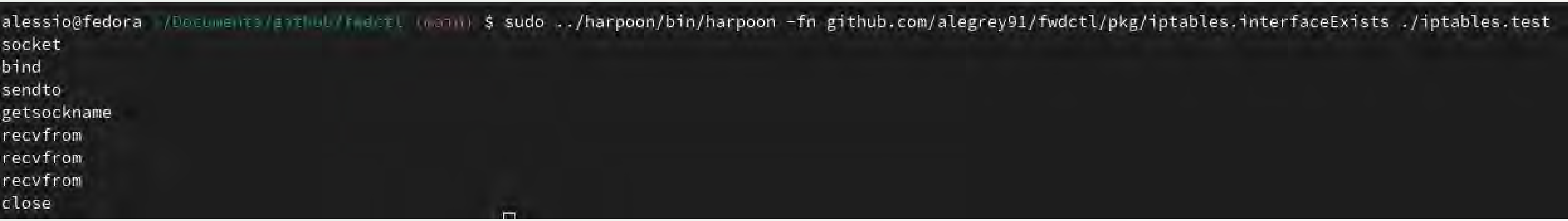
```
alessio@fedora ~/Documents/github/fwdctl (main) $ objdump --syms ./iptables.test | grep interfaceExists
0000000000503240 g     F .text  0000000000000064              github.com/alegrey91/fwdctl/pkg/iptables.interfaceExists
0000000000504220 g     F .text  0000000000000195              github.com/alegrey91/fwdctl/pkg/iptables.Test_interfaceExists
00000000005043c0 g     F .text  0000000000000132              github.com/alegrey91/fwdctl/pkg/iptables.Test_interfaceExists.func1
```

Harpoon



```
# harpoon -fn main.doSomething ./binary.test
```

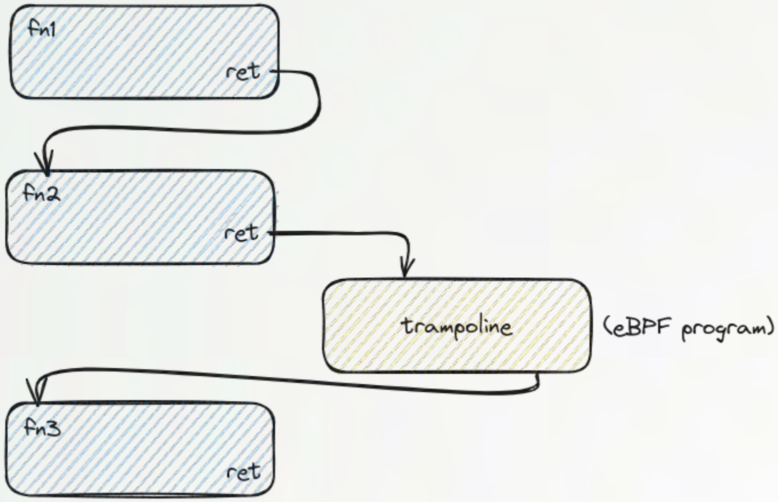
Harpoon



```
alessio@fedora /Documents/github/fwdctl (main) $ sudo ../harpoon/bin/harpoon -fn github.com/alegrey91/fwdctl/pkg/iptables.interfaceExists ./iptables.test
socket
bind
sendto
getsockname
recvfrom
recvfrom
recvfrom
close
```

The Uretprobe issue

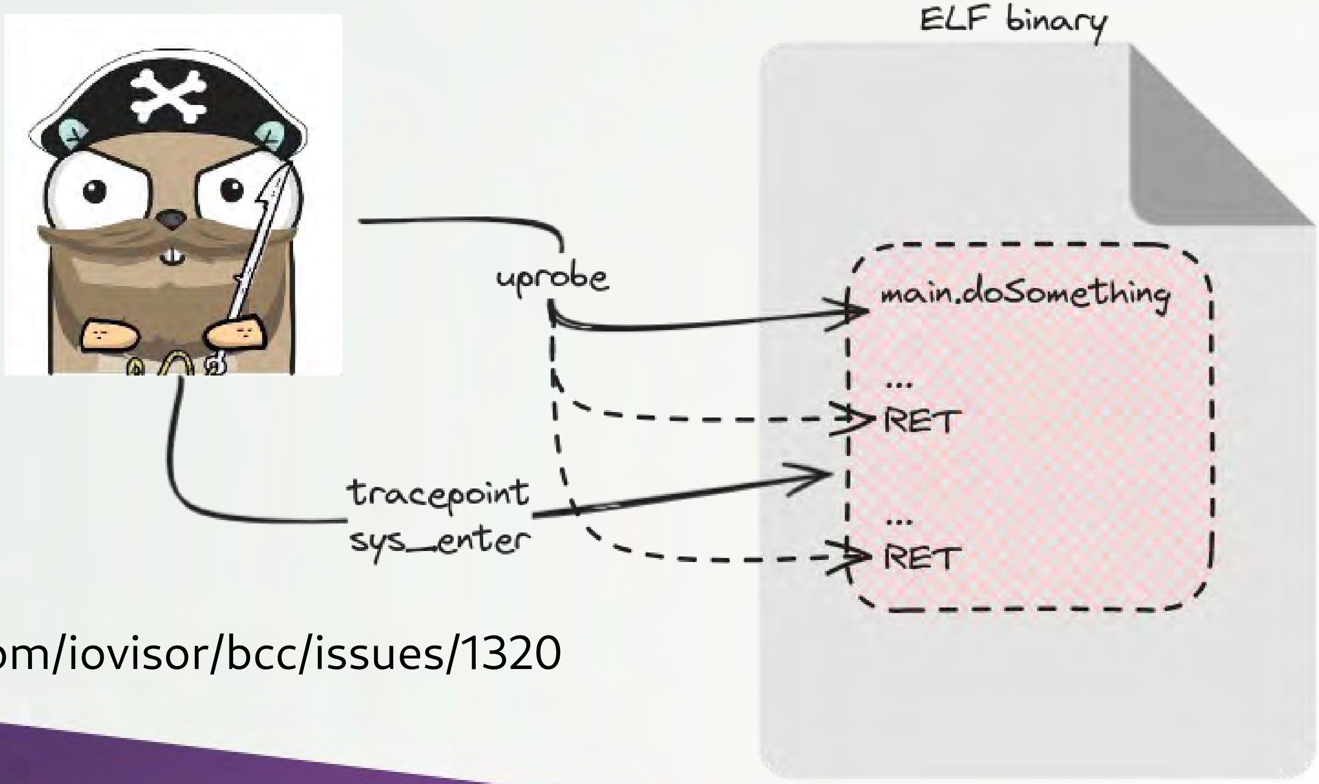
- A **uretprobe** overwrite the return address of the probed function with the address of a trampoline
- Once hit, the eBPF code is executed and after its end, the instruction pointer is restored to point to the next instruction



- Since the stack dinamically changes (due to the GC), it could cause the program corruption

Workaround

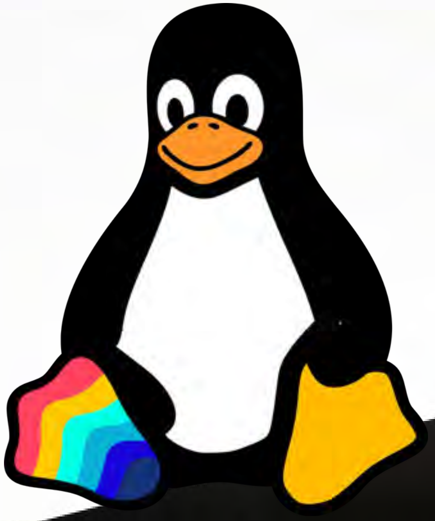
- **uprobes** can be attached to specific offsets
- Simulate a **uretprobe** by adding a **uprobe** on each RET instruction



Suggested here:
https://github.com/iovisor/bcc/issues/1320

Benefits of moving to `libbpfgo`

- More efficient:
  We can simulate a **uretprobe** by attaching **uprobes** at RET instructions
- Easily distributable:
  eBPF program is now **CO-RE** (no more GCC dependency)



References / Special Thanks

- https://github.com/iovisor/bcc/issues/1320#issuecomment-407927542
- https://github.com/golang/go/issues/22008#issuecomment-523237105
- https://github.com/golang/go/issues/22008#issuecomment-864559684
- https://github.com/golang/go/issues/27077#issuecomment-415141461
- https://medium.com/bumble-tech/bpf-and-go-modern-forms-of-introspection-in-linux-6b9802682223

- Gianluca Borello (gianlucaborello)
- Mattia Meleleo (matt11)
- Luca Di Maio (89luca89)

Thanks for your attention