ALESSIO GREGGI

Software Engineer

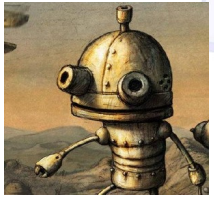
Test-driven Hardening: Crafting Seccomp Profiles within Test Pipeline

$ whoami

Alessio Greggi

- Software Engineer @ **SUSE**
- Cat food opener for my cat Gino

- Enthusiastic reader and hiker
- `$ cat {github,twitter}.com alegrey91`



CODE COVERAGE

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

Go Code Coverage

- 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 Code Coverage

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

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

code.google.com/p/go.blog/content/cover/size.go not tracked not covered covered

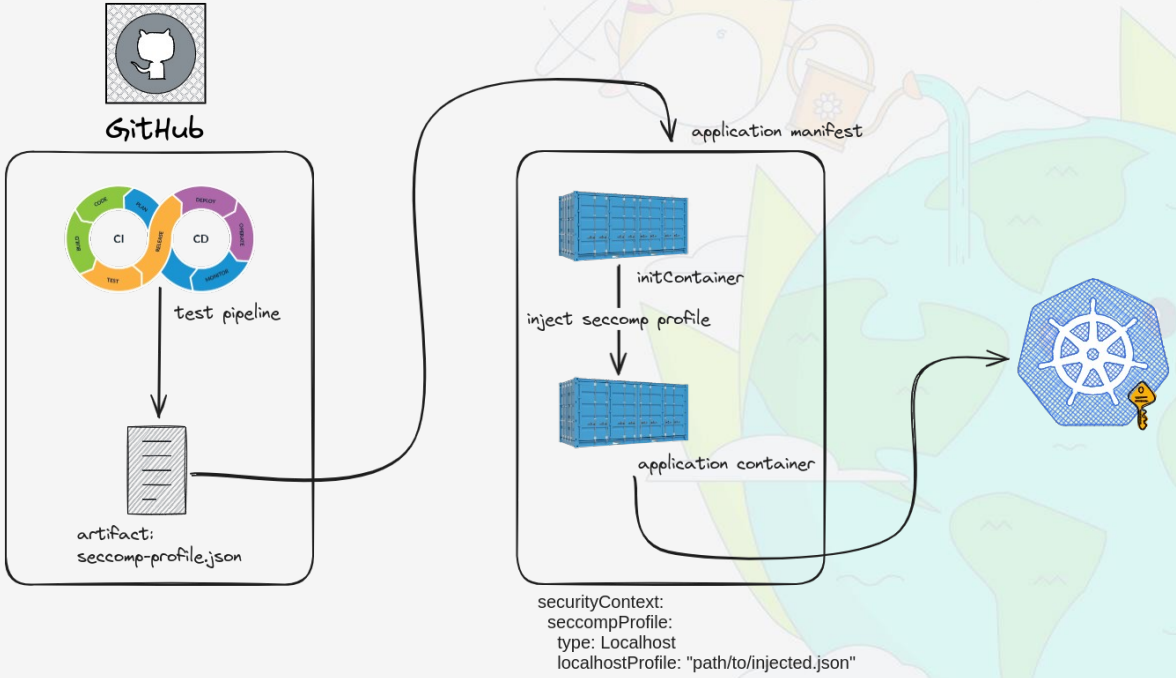
```
package size

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"
}
```

SECCOMP

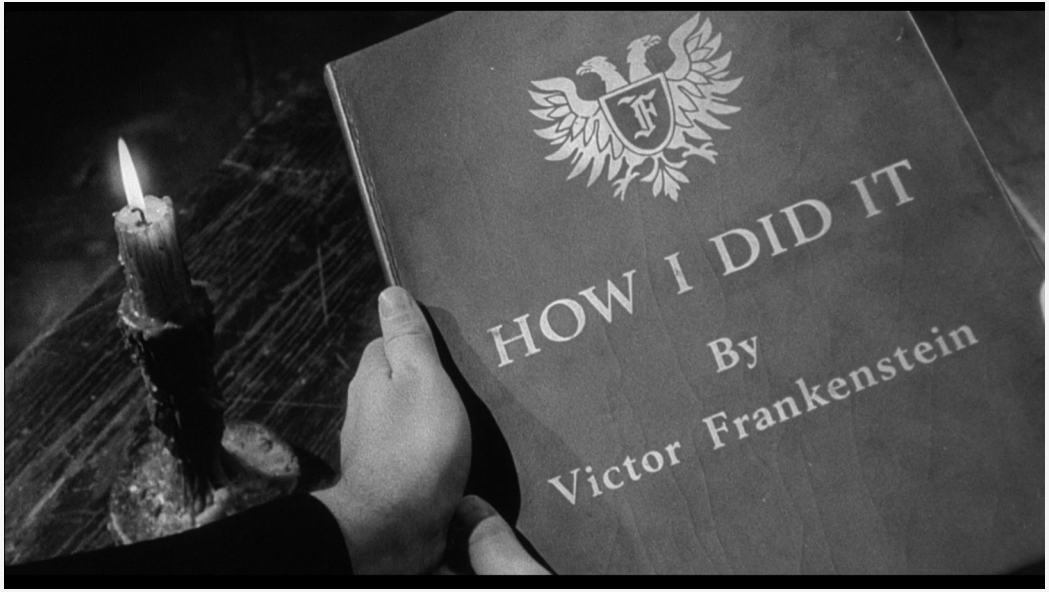
```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
        "SCMP_ARCH_X86_64",
        "SCMP_ARCH_X86",
        "SCMP_ARCH_X32"
    ],
    "syscalls": [
        {
            "name": "accept",
            "action": "SCMP_ACT_ALLOW",
            "args": []
        },
        {
      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

- Security feature in the Linux kernel
- Rules are defined in a file and referred as a seccomp profile
- Extensively used in the **Kubernetes** ecosystem (default profile)

Seccomp Profile as Artifact



TRACING SYSCALLS: HOW I DID IT



Tracing 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, but not ALL the syscalls

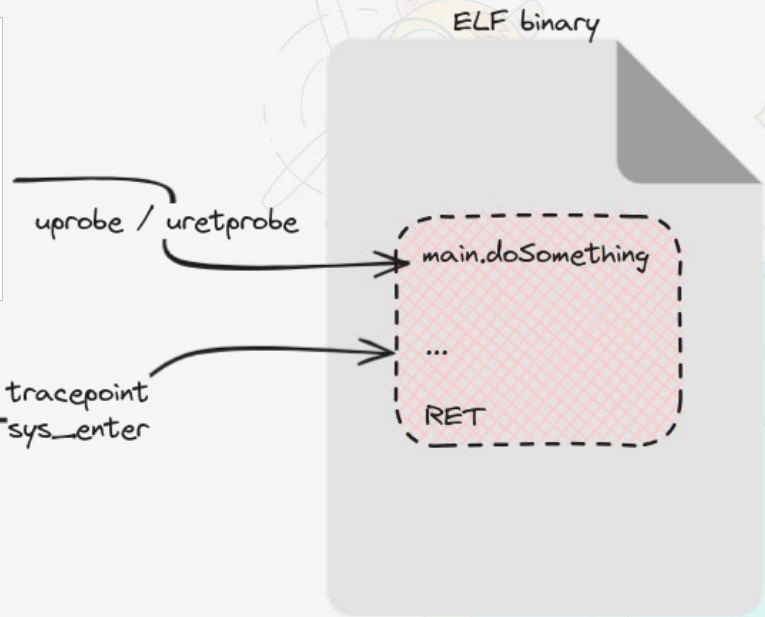
Tracing Syscalls (unit tests)

- A bit more complicated
- `go test` command **compile** and **run** the test binary all at once (no `strace go test .`)
- Compile test binary separately and then tracing it could include noise not related to our syscalls (no `strace ./test-binary`)
- In order to avoid of catching noise, we should **trace** only user-defined functions within the test binary

HARPOON

- Use eBPF to define a **tracepoint** that starts when a **uprobe** attached to the function is triggered and stop once the **uretprobe** return
- `github.com/alegrey91/harpoon`
- Uses `aquasecurity/libbpfgo`

Harpoon




```
# harpoon capture -f main.doSomething -- ./command
```

```
# harpoon capture -f main.main -- ./command
```

- Harpoon

DEMO TIME

RECIPE

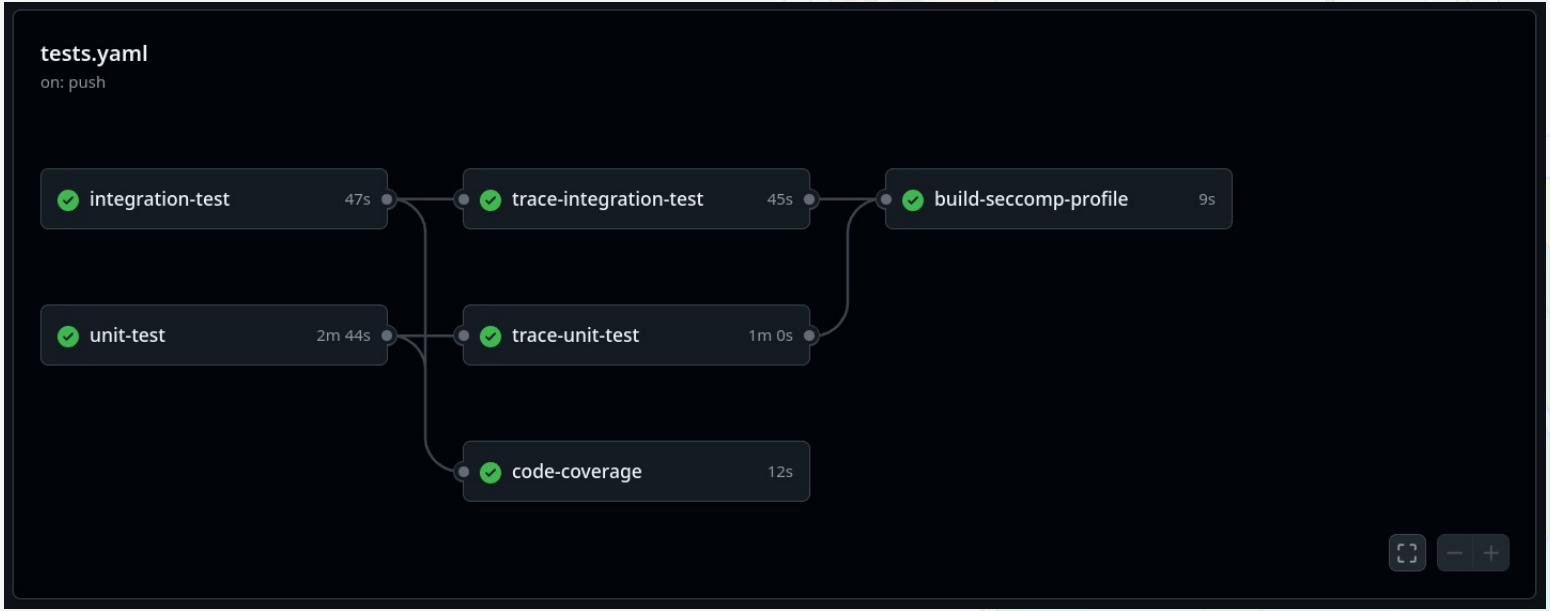
- https://github.com/alegrey91/harpoon
- https://github.com/alegrey91/fwdctl
- `testscript`

Recipe

script-based testing based on txtar files

```
exec fwdctl apply --help
stdout 'Usage:'

exec fwdctl apply --file rules.yaml

fwd_exists lo tcp 3000 127.0.0.1 80
fwd_exists lo tcp 3001 127.0.0.1 80
fwd_exists lo udp 3002 127.0.0.1 80

exec fwdctl delete -n 1
```

Recipe


tests.yaml
on: push
integration-test
47s
trace-integration-test
45s
build-seccomp-profile
9s
unit-test
2m 44s
trace-unit-test
1m 0s
code-coverage
12s

- Recipe (`trace-integration-test`)
  - Installs `harpoon`
  - Run `testscript` suite
    - Wrapped `exec` command to execute `harpoon` under the hood when running integration tests (`exec_cmd`) →

Recipe (`trace-integration-test`)

```go
func execCmd(ts *testscript.TestScript, neg bool, args []string) {
	var backgroundSpecifier = regexp.MustCompile(`^&([a-zA-Z_0-9]+&)?$`)
	uuid := getRandomString()
	workDir, err := os.Getwd()
	if err != nil {
		ts.Fatalf("unable to find work dir: %v", err)
	}
	customCommand := []string{
		"/usr/local/bin/harpoon",
		"capture",
		"-f",
		"main.main",
		"--save",
		"--directory",
		fmt.Sprintf("%s/integration-test-syscalls", workDir),
		"--include-cmd-stdout",
		"--include-cmd-stderr",
		"--name",
		fmt.Sprintf("main_main_%s", uuid),
		"--",
	}
```

```
exec_cmd fwdctl create -d 3000 -s 127.0.0.1 -p 80 -i lo
fwd_exists lo tcp 3000 127.0.0.1 80

# test alternative name 'add'
exec fwdctl add -d 3001 -s 127.0.0.1 -p 80 -i lo
fwd_exists lo tcp 3001 127.0.0.1 80

# test creation of udp rule
exec_cmd fwdctl create -d 3002 -s 127.0.0.1 -p 80 -i lo -P udp
fwd_exists lo udp 3002 127.0.0.1 80

exec fwdctl list -o json
cmp stdout fwdctl_list.json

# clean up environment
exec fwdctl delete -n 1
exec fwdctl delete -n 1
exec fwdctl delete -n 1
```

Recipe (`trace-unit-test`)

- Installs `harpoon`
- `harpoon analyze`
- `harpoon build`

- Recipe (`build-seccomp-profile`)
  - Final job downloads metadata from previous jobs
  - Installs `harpoon`
  - Generate profile using `harpoon build`
  - Upload Seccomp profile as artifact

- Recipe

```
1  {
2      "defaultAction": "SCMP_ACT_ERRNO",
3      "architectures": [
4          "SCMP_ARCH_X86_64",
5          "SCMP_ARCH_X86",
6          "SCMP_ARCH_X32"
7      ],
8      "syscalls": [
9          {
10             "names": [
11                 "bind",
12                 "close",
13                 "epoll_create1",
14                 "epoll_ctl",
15                 "fcntl",
16                 "futex",
17                 "getsockname",
18                 "nanosleep",
19                 "openat",
20                 "pipe2",
21                 "recvfrom",
22                 "sendto",
23                 "socket",
24                 "write"
25             ],
26             "action": "SCMP_ACT_ALLOW"
27         }
28     ]
29 }
```


Not sure if thank you
for your attention
or thank you for not
sleeping during the presentation
quickmeme.com