Фаззинг программ на языке Go

Форк оригинального репозитория
https://meowgit.nekoea.red/vovuas2003/linux-auth

Ветка для фаззинга
https://meowgit.nekoea.red/vovuas2003/linux-auth/src/branch/fuzz

План:

1. Удавшийся нативный фаззинг функции аутентификации.

2. Удавшийся blackbox фаззинг через AFL++.

3. Неудачные попытки запустить другие инструменты.

4. Итоги.

Go нативно поддерживает фаззинг с версии 1.18 (проект написан на 1.23, после сборки на Debian 13 версия автоматически поднялась до 1.24). Нужно написать отдельную функцию для фаззинга, по мотивам кода оригинальной программы была написана функция проверки аутентификации (не считая формы ввода данных, полный пайплайн с походом в БД).

```go
package myfuzz

import (
    "testing"

    "linux-auth/internal/auth"
    "linux-auth/internal/db"
)

func FuzzAuth(f *testing.F) {
    err := db.Init("test_data.db")
    if err != nil {
        f.Fatalf("DB init error: %v", err)
    }
    f.Cleanup(func() {
        db.Close()
    })

    f.Add("admin", "admin123") // right admin
    f.Add("user1", "password1") // right user
    f.Add("admin", "admin") // wrong admin
    f.Add("user", "password") // wrong user
    f.Add("", "") // blank test

    f.Fuzz(func(t *testing.T, username string, password string) {
        ok, err := auth.Authenticate(username, password)
        if err != nil {
            t.Errorf("For username %q and password %q error: %v", username, password, err)
        }
        if ok {
            if (username == "admin" && password == "admin123") || (username == "user1" && password == "password1") {
                return
            }
            t.Errorf("Unexpected login for username %q and password %q", username, password)
        }
    })
}
```

Запуск фаззинга с последующим сбором покрытия. Время для примера задано 5 минут, ограничение на единственный поток из-за SQLite (иначе могла быть ошибка database is locked). Скорость – несколько (2-3) тысяч запусков в секунду.

```bash
#!/bin/bash

cp ../data/users.db test_data.db

go test -fuzz=FuzzAuth -fuzztime=5m --parallel=1
go test -run=FuzzAuth -coverprofile=cover.out -coverpkg=../... | tee cover.txt
```

Удобный просмотр покрытия в браузере.

```bash
#!/bin/bash

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

Т.к. приложение консольное, то удалось запустить blackbox фаззинг готового бинарника через QEMU режим AFL++ (для этого пришлось собрать AFL++ из исходников), данные подаются на stdin. В таком варианте приложение проверяется полностью как оно есть (весь пайплайн обработки данных от начала до конца), а не только отдельные функции.

Запуск фаззинга в QEMU режиме, для примера на 5 минут. Скорость около 6 запусков в секунду.

```
#!/bin/bash

mkdir -p data
cp ../data/users.db ./data/users.db
mkdir -p configs
cp ../configs/config.toml ./configs/config.toml

mkdir -p fuzz_out

afl-fuzz -V $(( 60 * 5 )) -Q -i fuzz_in -o fuzz_out -- ../build/release/authapp
```

```
vovuas@vbox:~/linux-auth/aflfuzz$ ls -l fuzz_in/
total 20
-rw-rw-r-- 3 vovuas vovuas  2 May  6 11:20 blank_test.txt
-rw-rw-r-- 3 vovuas vovuas 15 May  6 11:18 right_admin.txt
-rw-rw-r-- 3 vovuas vovuas 16 May  6 11:19 right_user.txt
-rw-rw-r-- 3 vovuas vovuas 12 May  6 11:19 wrong_admin.txt
-rw-rw-r-- 3 vovuas vovuas 14 May  6 11:19 wrong_user.txt
vovuas@vbox:~/linux-auth/aflfuzz$ cat fuzz_in/*


admin
admin123
user1
password1
admin
admin
user
password
vovuas@vbox:~/linux-auth/aflfuzz$
```

Начальные данные для фаззинга.

Также были предприняты попытки (неудачные):

– Заставить AFL++ работать не в blackbox режиме, а в связке с gccgo. Возможно это вообще даже в теории не должно было заработать, либо я не нашёл правильного способа подменить компилятор.

– Запустить утилиту go-fuzz, которая существовала до добавления в Go нативного фаззинга. В отличие от нативного фаззинга, в функцию нельзя передать, например, 2 строки, а можно только массив байт (придётся самому разделить на 2 строки). Но установка по официальной инструкции из readme не работает по причине golang.org/x/tools@v0.44.0 requires go >= 1.25.0 (running go 1.24.4).

– Использовать golang-fuzz. Как оказалось, это обёртка для перевода 1 формата фаззинг тестов в 4 других и запуск готовых фаззеров, которые подразумевается заранее установить. Формат входных данных для функции фаззинга такой же, как в go-fuzz. Кроме нативного фаззера (который я и так сделал сразу в нормальном виде) поддерживается go-fuzz (тесты сгенерировались, а запустить не получилось по причине, описанной в предыдущем пункте), libfuzzer и afl (тут даже тесты не сгенерировались).

Итоги 6-часового фаззинга (+ в процессе отладки скриптов тоже были запуски)

```
fuzz: elapsed: 1h49m48s, execs: 18401340 (2312/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h49m51s, execs: 18408474 (2377/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h49m54s, execs: 18415478 (2335/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h49m57s, execs: 18422669 (2397/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m0s, execs: 18429851 (2394/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m3s, execs: 18436279 (2143/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m6s, execs: 18442134 (1952/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m9s, execs: 18449045 (2303/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m12s, execs: 18456418 (2458/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m15s, execs: 18463253 (2278/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m18s, execs: 18469711 (2146/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 1h50m21s, execs: 18473499 (1266/sec), new interesting: 3 (total: 29)


fuzz: elapsed: 5h59m39s, execs: 53678374 (2041/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 5h59m42s, execs: 53685005 (2210/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 5h59m45s, execs: 53692410 (2469/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 5h59m48s, execs: 53699822 (2470/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 5h59m51s, execs: 53707345 (2507/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 5h59m54s, execs: 53714629 (2428/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 5h59m57s, execs: 53722523 (2631/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 6h0m0s, execs: 53730604 (2693/sec), new interesting: 3 (total: 29)
fuzz: elapsed: 6h0m0s, execs: 53730604 (0/sec), new interesting: 3 (total: 29)
PASS
ok      linux-auth/myfuzz       21600.083s
vovuas@vbox:~/linux-auth/myfuzz$
```

```
PASS
coverage: 75.0% of statements in ../...
ok      linux-auth/myfuzz       0.030s
```

```
american fuzzy lop ++4.41a {default} (../build/release/authapp) [explore]
┌─ process timing ──────────────────────┐┌─ overall results ──────┐
│        run time : 0 days, 6 hrs, 0 min, 0 sec ││  cycles done : 2       │
│   last new find : 0 days, 0 hrs, 0 min, 26 sec ││ corpus count : 1818    │
│last saved crash : none seen yet       ││saved crashes : 0       │
│ last saved hang : none seen yet       ││  saved hangs : 0       │
├─ cycle progress ──────────────────────┤├─ map coverage ─────────┤
│  now processing : 1718.1 (94.5%)      ││   map density : 20.03% / 35.13%
│  runs timed out : 0 (0.00%)           ││count coverage : 5.27 bits/tuple
├─ stage progress ──────────────────────┤├─ findings in depth ────┤
│  now trying : havoc                   ││favored items : 1187 (65.29%)
│ stage execs : 1831/5120 (35.76%)      ││ new edges on : 1215 (66.83%)
│ total execs : 137k                    ││total crashes : 0 (0 saved)
│  exec speed : 6.61/sec (zzzz...)      ││ total tmouts : 1 (0 saved)
├─ fuzzing strategy yields ─────────────┴┤ item geometry ──────────┤
│   bit flips : 19/267k, 17/267k, 13/267k ││   levels : 12          │
│  byte flips : 2/33.4k, 1/33.4k, 0/33.4k ││  pending : 361         │
│ arithmetics : 79/112, 24/50, 0/0      ││ pend fav : 27          │
│  known ints : 4/8, 8/28, 0/0          ││own finds : 1813        │
│  dictionary : 0/0, 0/0, 0/0, 0/0      ││ imported : 0           │
│havoc/splice : 1456/80.9k, 0/0         ││stability : 39.48%      │
│py/custom/rq : unused, unused, unused, unused
│    trim/eff : n/a, 99.90%             │└────────────────────────┘
└─ strategy: explore ──────── state: in progress ─┘
                                                    [cpu000:250%]
```

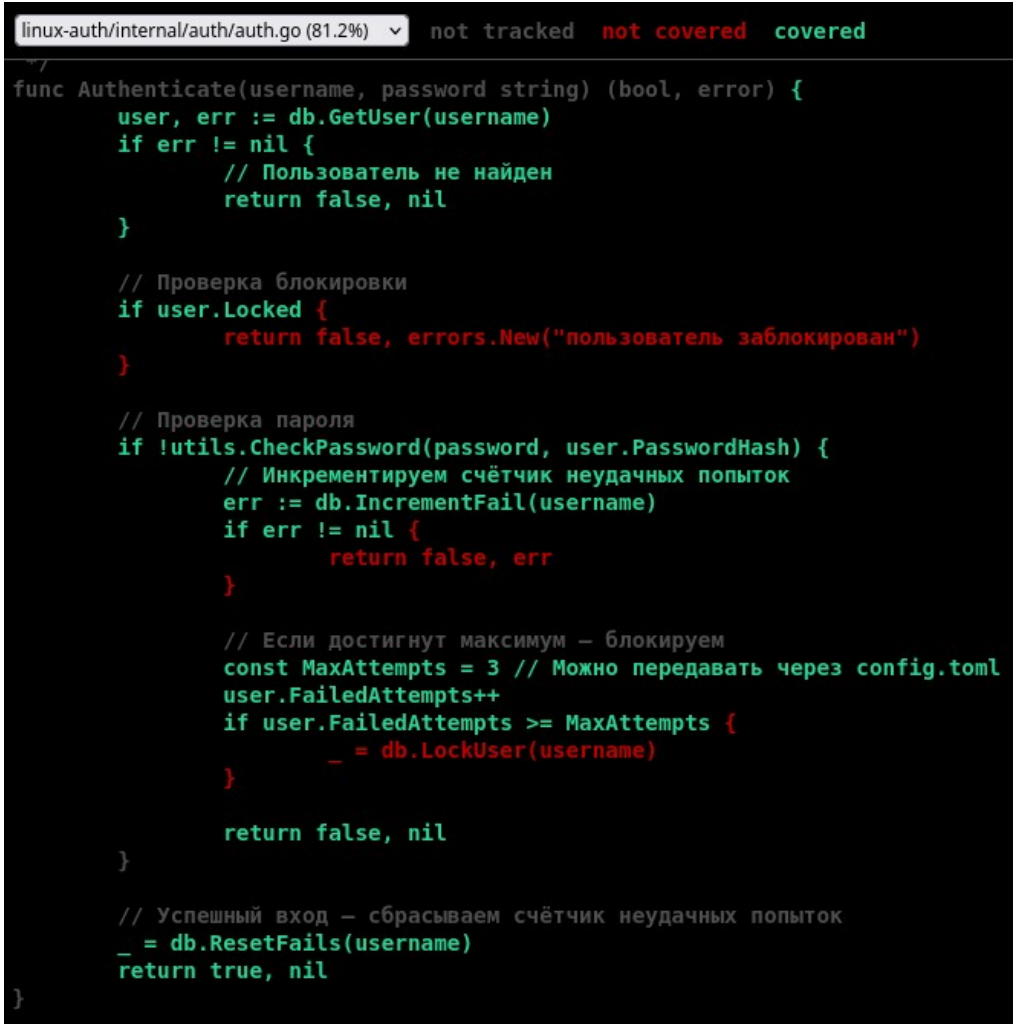
```
+++ Testing aborted programmatically +++
[!] Time limit was reached
[*] Writing fuzz_out/default/fastresume.bin ...
[+] fastresume.bin successfully written with 10563810 bytes.
[+] We're done here. Have a nice day!

vovuas@vbox:~/linux-auth/aflfuzz$
```

Нативный фаззинг

AFL++

Покрытие нативным фаззингом (1 / 3)



Покрытие нативным фаззингом (2 / 3)

linux-auth/internal/db/sqlite.go (70.3%) not tracked not covered

```
 */
func Init(path string) error {
	var err error

	database, err = sql.Open("sqlite3", path)
	if err != nil {
		return err
	}

	// Проверка соединения
	if err = database.Ping(); err != nil {
		return err
	}

	return createTables()
}

/**
 * @brief Закрытие соединения с БД
 */
func Close() {
	if database != nil {
		_ = database.Close()
	}
}

/**
 * @brief Создание таблиц при первом запуске
 */
func createTables() error {
	query := `
	CREATE TABLE IF NOT EXISTS users (
		id INTEGER PRIMARY KEY AUTOINCREMENT,
		username TEXT UNIQUE NOT NULL,
		password_hash TEXT NOT NULL,
		failed_attempts INTEGER DEFAULT 0,
		locked INTEGER DEFAULT 0
	);
	`

	_, err := database.Exec(query)
	return err
}
```

linux-auth/internal/db/sqlite.go (70.3%) not tracked not covered covered

```
 */
func GetUser(username string) (*User, error) {
	query := `
	SELECT id, username, password_hash, failed_attempts, locked
	FROM users
	WHERE username = ?;
	`

	row := database.QueryRow(query, username)

	var user User
	var locked int

	err := row.Scan(
		&user.ID,
		&user.Username,
		&user.PasswordHash,
		&user.FailedAttempts,
		&locked,
	)

	if err == sql.ErrNoRows {
		return nil, errors.New("пользователь не найден")
	}
	if err != nil {
		return nil, err
	}

	user.Locked = locked != 0
	return &user, nil
}

/**
 * @brief Увеличить счётчик неудачных попыток входа
 * @param username Логин пользователя
 */
func IncrementFail(username string) error {
	query := `
	UPDATE users
	SET failed_attempts = failed_attempts + 1
	WHERE username = ?;
	`

	_, err := database.Exec(query, username)
	return err
}
```

linux-auth/internal/db/sqlite.go (70.3%) not tracked not covered covered

```
 */
func ResetFails(username string) error {
	query := `
	UPDATE users
	SET failed_attempts = 0
	WHERE username = ?;
	`

	_, err := database.Exec(query, username)
	return err
}

/**
 * @brief Заблокировать пользователя
 * @param username Логин пользователя
 */
func LockUser(username string) error {
	query := `
	UPDATE users
	SET locked = 1
	WHERE username = ?;
	`

	_, err := database.Exec(query)
	return err
}

/**
 * @brief Добавить нового пользователя (для инициализации)
 * @param username Логин
 * @param passwordHash Хеш пароля
 */
func CreateUser(username, passwordHash string) error {
	query := `
	INSERT INTO users (username, password_hash)
	VALUES (?, ?);
	`

	_, err := database.Exec(query, username, passwordHash)
	if err != nil {
		return fmt.Errorf("не удалось создать пользователя: %w", err)
	}

	return nil
}
```

Покрытие нативным фаззингом (3 / 3)