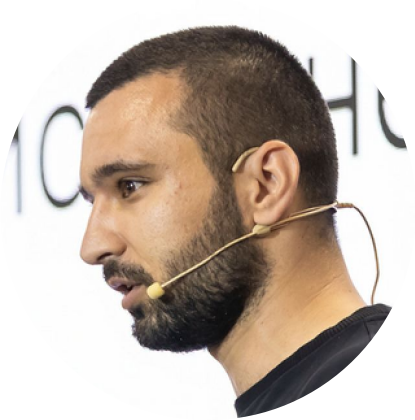
Программируемый фаззинг как инструмент глубокого тестирования сложных систем

ТБ Форум 2026

COREINFRA

AI × Verification → Future



Григорий Петросян

– Сооснователь и CTO, CoreInfra

– github.com/flyingmutant

sec ⟷ dev

sec →← dev

$$\text{sec}_{\text{AI}} \rightarrow\leftarrow \text{dev}_{\text{AI}}$$

~~AUTOMATIC~~

~~SCANNER~~

Обычный фаззер:

```
const uint8_t *Data
      size_t  Size
```

Обычно фаззят:
крэш с *Sanitizer

Проблемы в 2026+

– программы не работают с сырыми байтами
– программы не падают

Top 10:2025 List

1. A01:2025 - Broken Access Control
2. A02:2025 - Security Misconfiguration
3. A03:2025 - Software Supply Chain Failures
4. A04:2025 - Cryptographic Failures
5. A05:2025 - Injection
6. A06:2025 - Insecure Design
7. A07:2025 - Authentication Failures
8. A08:2025 - Software or Data Integrity Failures
9. A09:2025 - Security Logging and Alerting Failures
10. A10:2025 - Mishandling of Exceptional Conditions

Программируемый фаззер:

```
src.any_of(…)
src.repeat(…)
src.select(…)
```

Программируемый фаззер:

структурная генерация

+

структурный поток управления

Программируемо фаззим:
нарушение инвариантов

Top 10:2025 List

1. A01:2025 - Broken Access Control
2. A02:2025 - Security Misconfiguration
3. A03:2025 - Software Supply Chain Failures
4. A04:2025 - Cryptographic Failures
5. A05:2025 - Injection
6. A06:2025 - Insecure Design
7. A07:2025 - Authentication Failures
8. A08:2025 - Software or Data Integrity Failures
9. A09:2025 - Security Logging and Alerting Failures
10. A10:2025 - Mishandling of Exceptional Conditions

Top 10:2025 List

1. A01:2025 - Broken Access Control
2. A02:2025 - Security Misconfiguration
3. A03:2025 - Software Supply Chain Failures
4. A04:2025 - Cryptographic Failures
5. A05:2025 - Injection
6. A06:2025 - Insecure Design
7. A07:2025 - Authentication Failures
8. A08:2025 - Software or Data Integrity Failures
9. A09:2025 - Security Logging and Alerting Failures
10. A10:2025 - Mishandling of Exceptional Conditions

Инвариант:
данные должны быть обезврежены
в контексте использования

```
HTML: <
SQL:  ‘
path: /
```

poison token

```
   inert + POISON + inert
d8384d81 +   ‘     + b4fa517c
```

DEMO

CoreInfra AT1: программируемый полносистемный фаззер

// первый в мире

```
@at1.driver
def blob_storage_driver(src):
    src = at1.Session(src)
    model = SimpleStorage()
    client = Client("http://127.0.0.1:8080")

    os.system("docker run -d blob_storage_service:1.0")

    with src.snapshot() as src:
        def bucket_create(src):…
        def bucket_protect(src):…
        def bucket_delete(src):…
        def object_put(src):…
        def object_get(src):…
        def object_get_deleted(src):…
        def object_protect(src):…
        def object_delete(src):…

        def step(src):
            src.select("action", (
                bucket_create,
                bucket_protect,
                bucket_delete,
                object_put,
                object_get,
                object_get_deleted,
                object_protect,
                object_delete,
            ))

        src.repeat("step", step)
```

OSS DEMO

chaos_theory

1. poison token injection
2. poison token broken access control

3. *авторизация совпадает с формальной политикой*

```
// Policy 0: Any User can create a list
// and see what lists they own.
permit(
    principal,
    action in [Action::"CreateList",
               Action::"GetOwnedLists"],
    resource == Application::"TinyTodo");

// Policy 2: Admins can perform any action.
permit(
    principal in Team::"admin",
    action,
    resource in Application::"TinyTodo");

// Policy 4: Interns can't create task lists.
forbid(
    principal in Team::"interns",
    action == Action::"CreateList",
    resource == Application::"TinyTodo");
```

```
// Policy 1: Any User can perform any action
// on a List they own.
permit(principal, action, resource)
when {
    resource is List &&
    resource.owner == principal
};

// Policy 3: A User can see a List
// if they are either a reader or editor.
permit(
    principal,
    action == Action::"GetList",
    resource)
when {
    principal in resource.readers ||
    principal in resource.editors
};
```

Fig. 2. Cedar policies for TinyTodo

sec → dev:
пишем код и проверяем инварианты

Q & A

COREINFRA

@coreinfra
coreinfra.ai

AI × Verification → Future

```
src.repeat_n(label: "step", n: 5.., |src: &mut Source<'_>| {
    src.select(
        label: "api_method",
        variants: BUGAZOO_API_METHODS,
        |src: &mut Source<'_>, variant: &str, _| match variant {
            "signup" => client.signup(src),
            "login" => client.login(src),
            "logout" => client.logout(src),
            "create_ticket" => client.create_ticket(src),
            "add_ticket_tag" => client.add_ticket_tag(src),
            "assign_ticket" => client.assign_ticket(src),
            "create_filter" => client.create_filter(src),
            "list_tickets" => client.list_tickets(src),
            _ => unreachable!(),
        },
    )
});
```

```rust
fn logout(&mut self, src: &mut Source<'_>) -> Effect {
    let (user_ix: usize, token: String) = match self.choose_logged_in_user(src, label: "logout_user") {
        Some(state: (usize, String)) => state,
        None => return Effect::Noop,
    };

    let effect: Effect = match self.http.post_empty(path: "/logout", token: Some(&token)) {
        CallResult::Ok(()) => {
            self.users[user_ix].token = None;
            Effect::Success
        }
        CallResult::ClientError(_) => Effect::Change,
    };
    self.check_sql_log(context: "logout");
    effect
}
```