$ cat title.txt

How Type-Driven Development Can Turbocharge your Contract's Security

Brought to you by @philogy

$ whoami

Philippe Dumonet
aka Philogy

- ~5 years experience working with smart contracts
- Programming Language R&D
- Lead Security Researcher @ Spearbit
- Lead Smart Contract Engineer @ Sorella

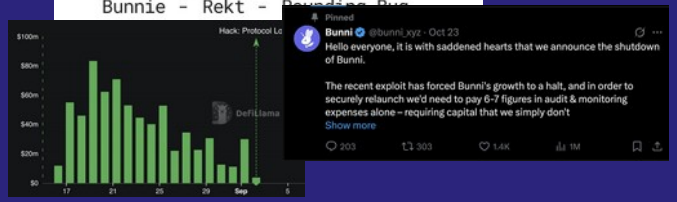
$ cat intro-1.txt

The Bar is High: Write Bug-Free Code or Die

BUNNI - REKT

Tuesday, September 9, 2025

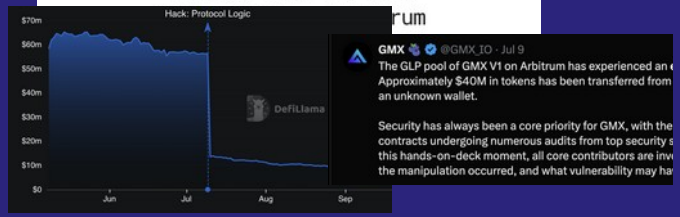
Bunnie - Rekt - Rounding Bug



GMX - REKT

Friday, July 11, 2025

...rum



... and dozens more

$ cat intro-2.txt

Security = Resources * Productivity

$ tdd --help

What is Type Driven Development (TDD)?

- Types as Documentation
- Types as Cheap Static Analysis
- Design through Types

$ man ecrecover

Case Study I: ecrecover

⚠ have to **remember** address(0) means failure

⚠ no clear difference between checked & unchecked state

Solution: Introduce a type to make difference clear

```
function ecrecover(bytes32 digest, uint8 v, bytes32 r, bytes32 s)
    returns (address)
{
    // ...
}
```

```
function usingEcrecover() public {
    address recovered = ecrecover(digest, v, r, s);
    // Can forget to check `recovered != 0`
    use(recovered);
}
```

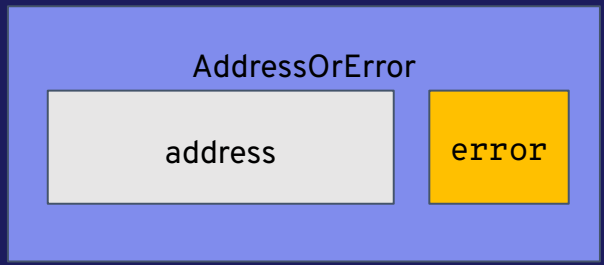
```
function ecrecover(bytes32 digest, uint8 v, bytes32 r, bytes32 s)
    returns (AddressOrError)
{}

function usingEcrecover() public {
    address recovered = ecrecover(digest, v, r, s); // ❌ Type mismatch error

    use(recovered);
}
```

$ man ecrecover

Case Study I: ecrecover

```
function ecrecover(bytes32 digest, uint8 v, bytes32 r, bytes32 s)
    returns (AddressOrError)
{}

function usingEcrecover() public {
    AddressOrError result = ecrecover(digest, v, r, s);
    address recovered = result.unwrap(); // ✅ Forced to explicitly handle

    use(recovered);


    // Alternatively
    if (result.isOk()) {
        address recovered = result.unwrap();
        use(recovered);
    } else {
        // Error handling
    }
}
```

- ✔ custom return type documents possible failure
- ✔ statically checks that error was handled
- ✔ removes cognitive burden of remembering edge case



$ man unit-math

Case Study II: math with units

⚠ API has footguns (can mix units incorrectly)

⚠ Documented via parameter names

⚠ Correct usage imposes high cognitive burden

```
function convertAtoB(uint priceAoverB, uint amountA)
    returns (uint)
{
    return amountA * 1e18 / priceAoverB;
}

function convertBtoA(uint priceAoverB, uint amountB)
    returns (uint)
{
    return amountB * priceAoverB / 1e18;
}

function doStuff(uint priceAoverB) {
    uint amountInA;
    uint out1 = convertAtoB(priceAoverB, amountInA);
    uint out2 = convertBtoA(priceAoverB, amountInA);
}
```

$ man unit-math

Case Study II: math with units

Solution: specialize uint256 using custom types

✓ compiler now statically checks correct usage

✓ correct function automatically chosen based on types

```
type PriceAOverB is uint;
type AmountA is uint;
type AmountB is uint;

function convert(PriceAOverB price, AmountA amount)
    returns (AmountB)
{
    return AmountB.wrap(
        AmountA.unwrap(amount) * 1e18 / PriceAOverB.unwrap(price)
    );
}

function convert(PriceAOverB price, AmountB amount)
    returns (AmountA)
{
    return AmountA.wrap(
        AmountB.unwrap(amount) * PriceAOverB.unwrap(price) / 1e18
    );
}

function doStuff(PriceAOverB price) {
    AmountA amountInA;
    AmountB out1 = convert(price, amountInA);
    AmountA out2 = convert(price, amountInA); // ✗ type error
}
```

$ man transfer-ownership

Case Study III: transferOwnership

```
function transferOwnership(address newOwner) public {
    require(newOwner != address(0));
    _setOwner(newOwner);
}

function _setOwner(address newOwner) internal {
    require(newOwner != address(0));
    // ... some logic
}
```

⚠ Input requirements not clear from definition

⚠ Duplication of verification logic “just in case”

```
type NonZeroAddress is address;

function transferOwnership(NonZeroAddress newOwner) public {
    _setOwner(newOwner);
}

function _setOwner(NonZeroAddress newOwner) internal {
    // ... some logic
}
```

Solution: Signal intent with types

When to apply Type Driven Development (TDD)

- documenting subtle requirements in comments

When to apply Type Driven Development (TDD)

- documenting subtle requirements in comments
- want to ensure caller doesn't forget to handle certain states/values

When to apply Type Driven Development (TDD)

- documenting subtle requirements in comments
- want to ensure caller doesn't forget to handle certain states/values
- using base types for similar but distinct concepts such as different units

“

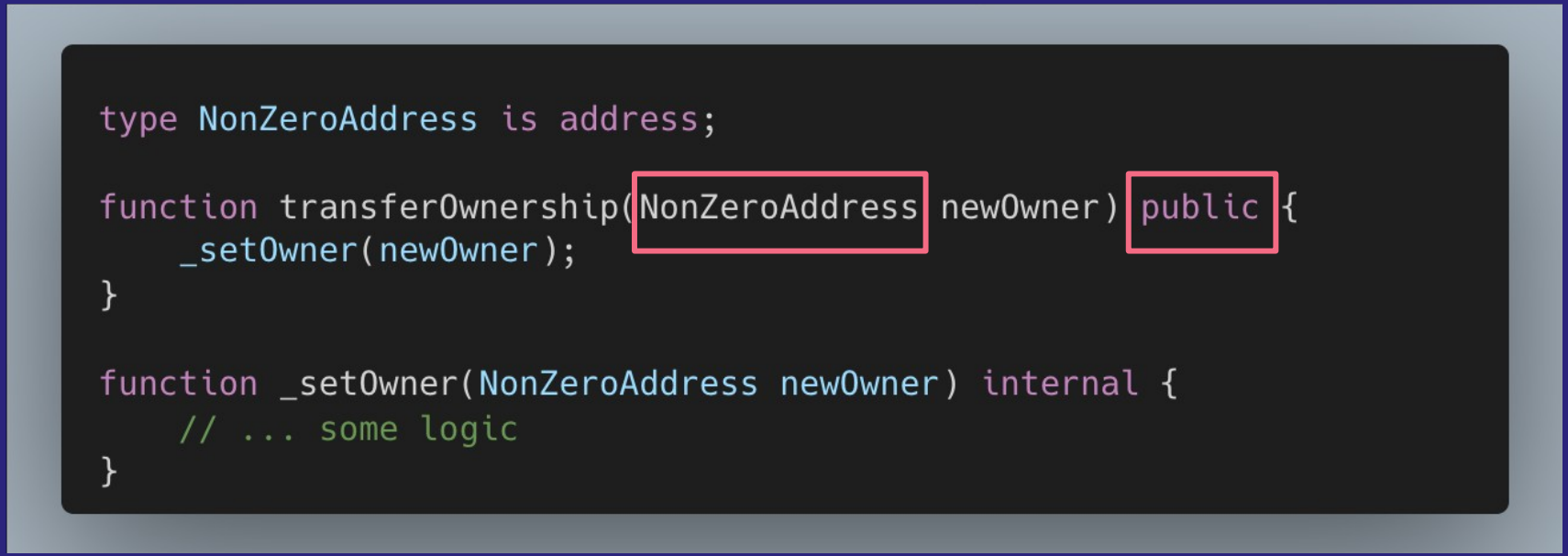
... Btw, did you spot the problem in one of the examples?

can't we take a closer look 🔍

```
type NonZeroAddress is address;

function transferOwnership(NonZeroAddress newOwner) public {
    _setOwner(newOwner);
}

function _setOwner(NonZeroAddress newOwner) internal {
    // ... some logic
}
```

❌ Solidity sees “NonZeroAddress” as an alias for “address” for purposes of ABI encoding/decoding

...let's take a closer look

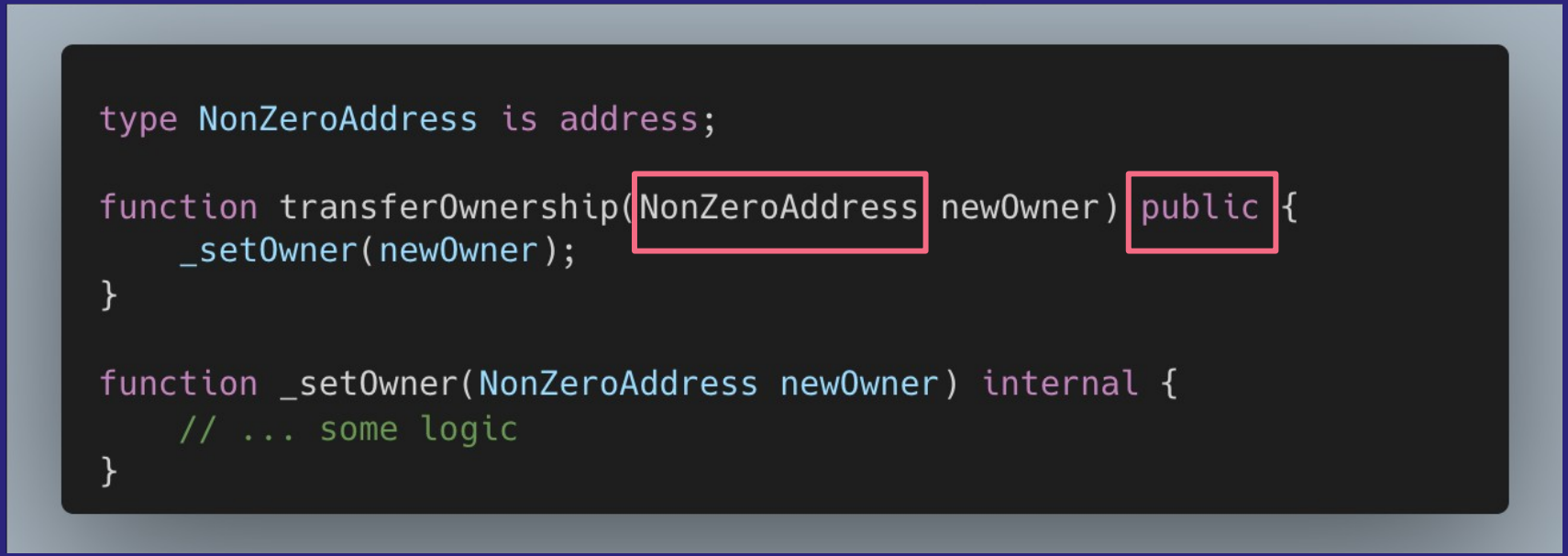


```
type NonZeroAddress is address;

function transferOwnership(NonZeroAddress newOwner) public {
    _setOwner(newOwner);
}

function _setOwner(NonZeroAddress newOwner) internal {
    // ... some logic
}
```

❌ Solidity sees "NonZeroAddress" as an alias for "address" for purposes of ABI encoding/decoding

❌ Will not validate our invariant that "x != 0"

...let's take a closer look



```
type NonZeroAddress is address;

function transferOwnership(NonZeroAddress newOwner) public {
    _setOwner(newOwner);
}

function _setOwner(NonZeroAddress newOwner) internal {
    // ... some logic
}
```

- ✖ Solidity sees "NonZeroAddress" as an alias for "address" for purposes of ABI encoding/decoding
- ✖ Will not validate our invariant that "x != 0"
- ✖ Will decode & instantiate "address(0)" as a "NonZeroAddress" without errors

$ solc tdd.sol 2>&1

Solidity TDD Pitfalls

- ✗ Users can override encapsulation
- ✗ Compiler will directly instantiate types
- ✗ Structs not zero-cost
- ✗ Repetitive boilerplate
- ✗ No standard library

“

Most of these issues are ***language level*** issues

To fix these we need to change the language, or just build our own

Introducing: Sensei

Helping good devs write better EVM contracts

$ sensei goals

Day 1 Goals

Fast Compile Times

Goal: 500ms worst case debug build

Standard Library

ABI, EIP712, TDD, ERC20, ERC721, etc.

Editor Support

LSP with autocomplete & go-to-definition

TDD-Focused Upgrades

Encapsulation

Private/public methods, functions & attributes

Generics

Make any code reusable with minimal boilerplate

Gas Optimization

Stronger inlining, structs members on stack

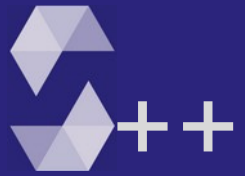
Why do we need a new language?

Solidity

- large backlog of existing issues
- team conservative about changes
- legacy codebase

Vyper

- simplicity maximalism
- building IR alone
- not prioritizing key dev needs (storage packing, dyn. memory)



Core Solidity

- complex type system
- split solc focus
- unclear how to contribute

Competition + I think it’d be Fun

$ sensei join

Closed Beta Begins: Q1 2026

(insert FOMO title to get you to scan)

18 NOV 2025

Q&A

philogy philogy philogy.21