Safely Integrating ERC20 Tokens to Your DeFi Application

Hello Security - January 7 2021

Who Am I?

- Josselin Feist, josselin@trailofbits.com, @montyly

- Trail of Bits: trailofbits.com
    - We help everyone build safer software
    - R&D focused: we use the latest program analysis techniques
        - Slither
        - Echidna
        - Manticore

Today Goals

- DeFi -> contracts composability
- Common risks when interacting with arbitrary tokens
- Recommendations and guidelines

General considerations

General considerations

- **You have contacted the developers.**
  - You may need to alert their team to an incident.
  - github.com/crytic/blockchain-security-contacts
- **They have a security mailing list for critical announcements.**
  - Their team should advise users (like you!) when critical issues are found or when upgrades occur.

General considerations

- **The token has a security review.**
  - Check
    - The length of the assessment (aka “level of effort”),
    - The reputation of the security firm, and
    - The number and severity of the findings.
  - Keep in mind security review != safe code

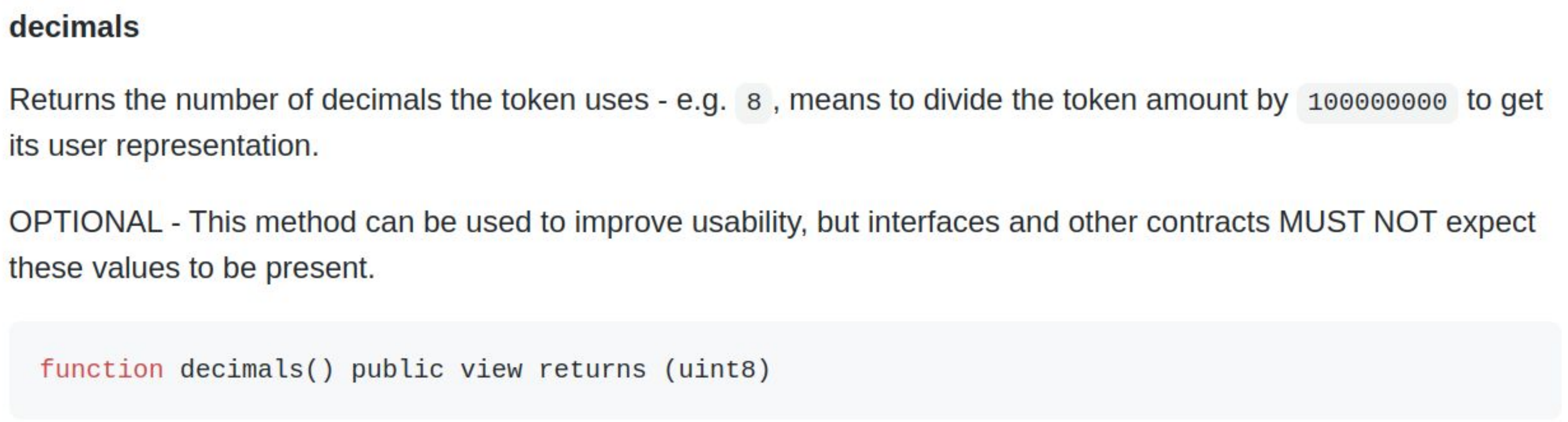
ERC conformity

ERC conformity

- Is this ok with tokens strictly following the specification?

```
function get(ERC20 token) internal returns(uint, uint8){
    uint8 decimals = token.decimals();
    uint balance = token.balanceOf(address(this));
    return balance, decimals;
}
```

ERC conformity - Optional



decimals

Returns the number of decimals the token uses - e.g. `8`, means to divide the token amount by `100000000` to get its user representation.

OPTIONAL - This method can be used to improve usability, but interfaces and other contracts MUST NOT expect these values to be present.

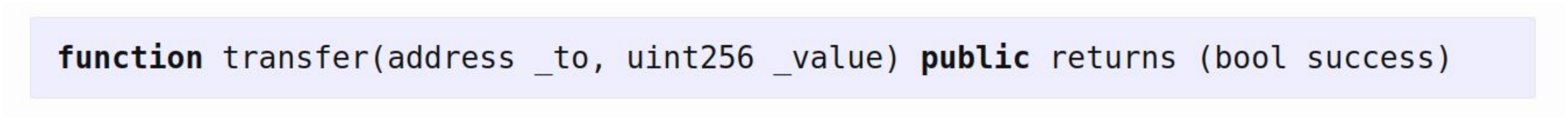
```
function decimals() public view returns (uint8)
```

ERC conformity - Optional

- Similar for `name` and `symbol`

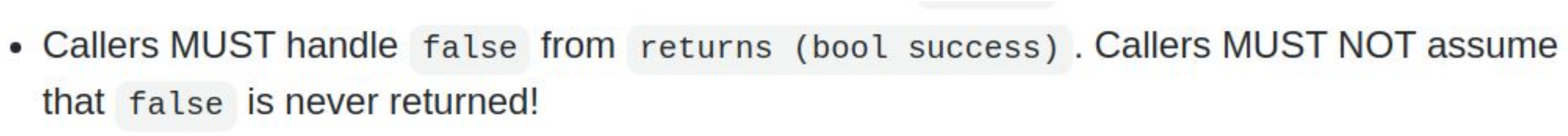
ERC conformity - Return value

- transfer/transferFrom returns a boolean



```
function transfer(address _to, uint256 _value) public returns (bool success)
```

- You must check for this value



- Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!

ERC conformity - Return value

- **Several tokens miss the return value on transfer/transferFrom**
  - Include high target (ex: USDT)
  - See [Missing return value bug — At least 130 tokens affected]()
- `require(token.transfer(..,..));` **will always revert**

ERC conformity - Return value

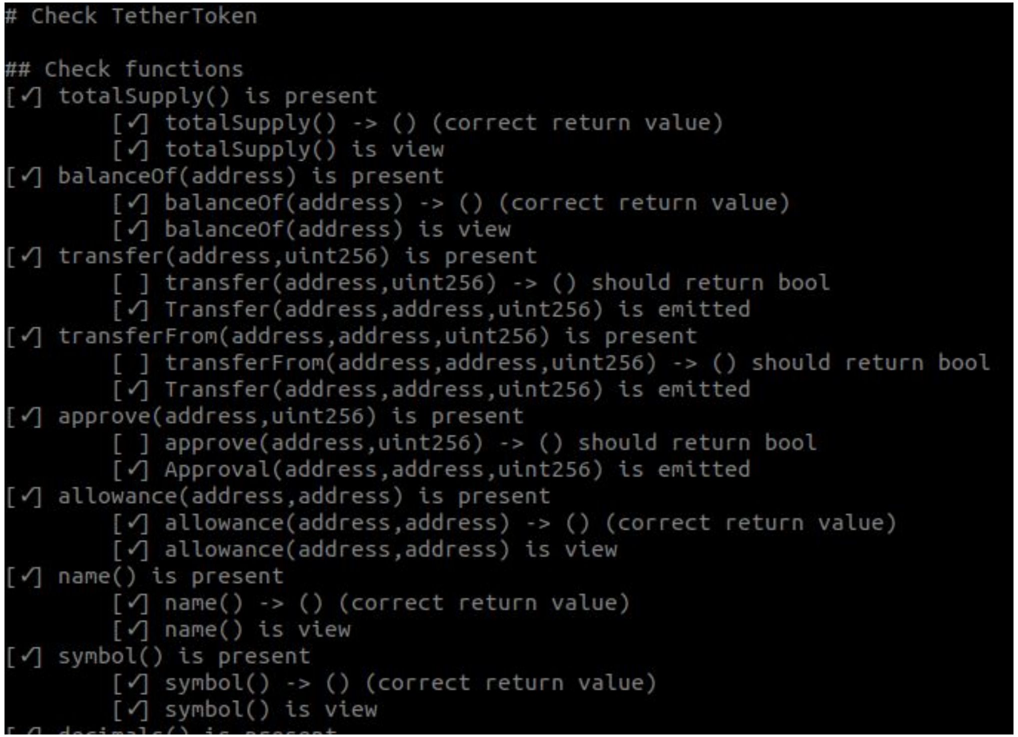
- **Two solutions**
  - Do not support ERC20 tokens that don’t follow the specification
  - or… use a “safe ERC20” approach
    - Low level call
    - Check for contract existence
    - Check if return value size is zero, or the value is true

slither-check-erc

- slither-check-erc
- **Tool based on Slither that will perform common checks**
  - ERC20, 223, 777, 721, 165, 1820
  - Check for
    - Missing / incorrect functions
    - Missing / incorrect events
    - Missing / incorrect return values
    - ...

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken
```

slither-check-erc



```
# Check TetherToken

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [ ] transfer(address,uint256) -> () should return bool
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [ ] transferFrom(address,address,uint256) -> () should return bool
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [ ] approve(address,uint256) -> () should return bool
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
```

Extensions Risks

ERC777-like

- **ERC777 (and equivalent) reentrancy**
  - Callback mechanism in `transfer/transferFrom`
  - Lead to reentrancy exploits in Uniswap and dForce

```
function withdraw(ERC20 token) internal{
    require(token.transfer(msg.sender, balance[msg.sender]));
    balance[msg.sender] = 0;
}
```

Unexpected Balance Update

- Does balance[msg.sender] always track correctly value?

```
function add(uint value) internal{
    require(token.transferFrom(msg.sender, address(this), value));
    balance[msg.sender] += value;
}
```

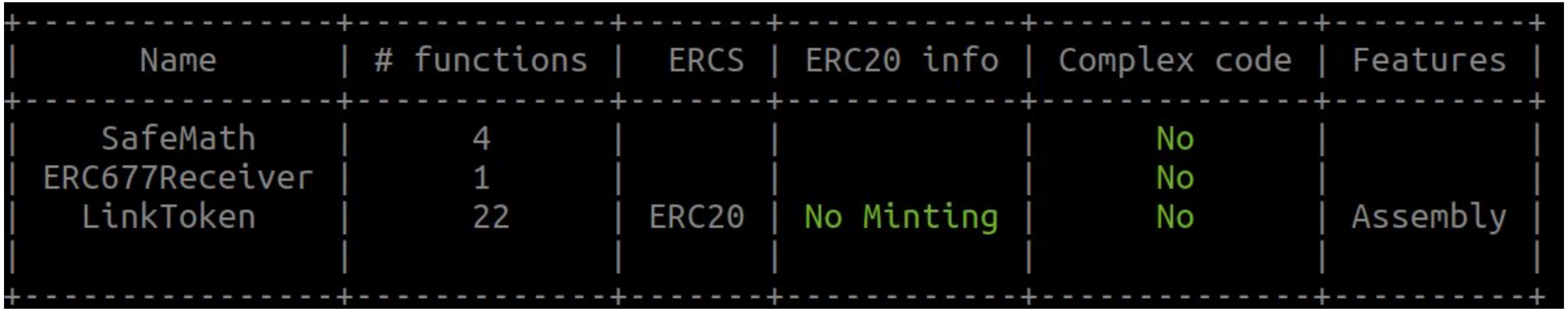
Unexpected Balance Update

- **Transfer fee.**
  - Deflationary tokens can lead to unexpected behavior.
  - Ex: USDT has a potential fee
- **Token can earn interest.**
  - Some tokens distribute interest to token holders. This interest might be trapped in the contract if not taken into account.
- **Both require manual inspection at the moment**

Contract Composition

Contract Composition

- **The token avoids unneeded complexity.**
  - The token should be a simple contract; a token with complex code requires a higher standard of review.
  - Use Slither's human-summary printer to identify complex code.

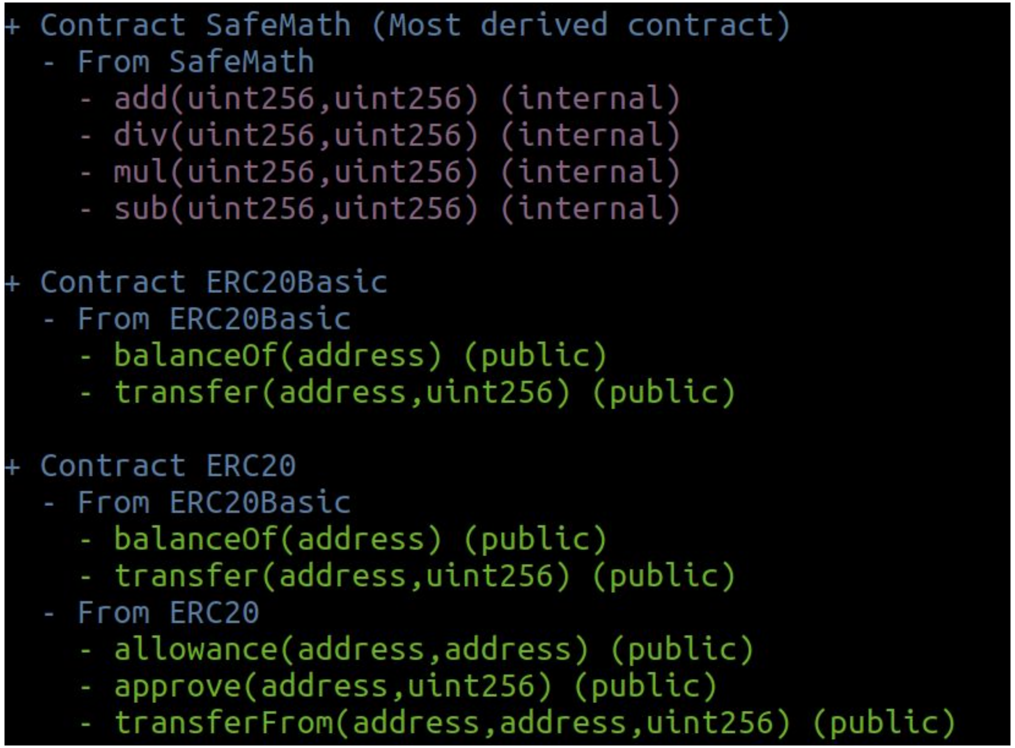


```
+-----------------+---------------+---------+---------------+---------------+-----------+
|      Name       | # functions   |  ERCS   | ERC20 info    | Complex code  | Features  |
+-----------------+---------------+---------+---------------+---------------+-----------+
|    SafeMath     |       4       |         |               |      No       |           |
| ERC677Receiver  |       1       |         |               |      No       |           |
|    LinkToken    |      22       |  ERC20  | No Minting    |      No       | Assembly  |
|                 |               |         |               |               |           |
+-----------------+---------------+---------+---------------+---------------+-----------+
```

Contract Composition

- **The token has only a few non–token-related functions.**
  - Non–token-related functions increase the likelihood of an issue in the contract.
  - Use Slither's contract-summary printer to broadly review the code used in the contract.

Contract Composition



```
+ Contract SafeMath (Most derived contract)
  - From SafeMath
    - add(uint256,uint256) (internal)
    - div(uint256,uint256) (internal)
    - mul(uint256,uint256) (internal)
    - sub(uint256,uint256) (internal)

+ Contract ERC20Basic
  - From ERC20Basic
    - balanceOf(address) (public)
    - transfer(address,uint256) (public)

+ Contract ERC20
  - From ERC20Basic
    - balanceOf(address) (public)
    - transfer(address,uint256) (public)
  - From ERC20
    - allowance(address,address) (public)
    - approve(address,uint256) (public)
    - transferFrom(address,address,uint256) (public)
```

Contract Composition

- **The token uses SafeMath.**
  - Contracts that do not use SafeMath require a higher standard of review.
- **The token only has one entry point.**
  - Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g. balances[token_address][msg.sender] might not reflect the actual balance).

Testing Basic Properties

slither-prop

- slither-prop
- **Generate automatically ERC20 properties**
  - Unit tests (Truffle)
  - Fuzzing (Echidna)
- **Contain 18 checks**
  - Self transfer is correctly implemented
  - Balance of the user is less or equal to the total supply
  - Cannot transfer more than the balance
  - ...

Owner privileges

Owner privileges

- **The token is not upgradeable.**
  - Upgradeable contracts might change their rules over time.
  - Use Slither's human-summary printer to determine if the contract is upgradeable.
- **The owner has limited minting capabilities.**
  - Malicious or compromised owners can abuse minting capabilities.
  - Use Slither's human-summary printer to review minting capabilities, and consider manually reviewing the code.

Owner privileges

- **The token is not pausable.**
  - Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pauseable code by hand.
- **The owner cannot blacklist the contract.**
  - Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand.

Owner privileges

- **The team behind the token is known and can be held responsible for abuse.**
    - Contracts with anonymous development teams, or that reside in legal shelters should require a higher standard of review.

Token scarcity

Token scarcity

- **No user owns most of the supply.**
  - If a few users own most of the tokens, they can influence operations based on the token's repartition.
- **The total supply is sufficient.**
  - Tokens with a low total supply can be easily manipulated.
- **The tokens are located in more than a few exchanges.**
  - If all the tokens are in one exchange, a compromise of the exchange can compromise the contract relying on the token.

Token scarcity

- **Users understand the associated risks of large funds or flash loans.**
  - Contracts relying on the token balance must carefully take in consideration attackers with large funds or attacks through flash loans.
- **The token does not allow flash minting.**
  - Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token.

Summary

Summary

- **Review every token that will interact with your codebase**
- github.com/crytic/building-secure-contracts
    - Token integration checklist
    - Guidelines and tools training