CVL: Parametric Rules

Michael George

Stanford, August 2022

Are my funds safe?

So far:

- `transfer` spends the sender's funds
- `transferFrom` reverts if caller's allowance is 0

So if I don't call `transfer` and don't give anyone an allowance, my funds are safe ...right?

- Do I control my own allowance?
- Do I control my own balance?

Only token holder can approve (stakeholder rule)

We want to show that the token holder controls their allowances

- Allowances are controlled by `approve`:

```
//// contracts/IERC20.sol

/// Sets `amount` as the allowance of `spender` over the caller's tokens.
function approve(address spender, uint256 amount) external returns (bool);
```

- Maybe check that only the holder can call `approve`?

```
//// certora/specs/ERC20.spec

/// Approve reverts unless called by the owner
rule onlyHolderCanCallApprove {

    address holder; address spender;

    env e; uint256 amount;
    approve@withrevert(e, spender, amount);

    // note: P => Q means "if P then Q" or "P implies Q"
    assert e.msg.sender != holder => lastReverted,
        "approve can only successfully be called by the holder";
}
```

- Fails (results link)! Who is the holder?
  - ...the address whose (outgoing) allowance changes

Only holder can approve, take 2 (variable change rule)

We want to show that the token holder controls their allowances

- if approve changes holder's allowance, then holder called it. (passes)

```
rule onlyHolderCanChangeAllowance {

    address holder; address spender;
    mathint allowance_before = allowance(holder, spender);

    env e; uint256 amount;
    approve(e, spender, amount);

    mathint allowance_after = allowance(holder, spender);

    assert allowance_after > allowance_before => e.msg.sender == holder,
        "addresses other than holder must not affect holder's allowance";


}
```

Only holder can approve, take 2 (variable change rule)

We want to show that the token holder controls their allowances

- if any method changes holder's allowance, then the holder called it. (link)
- if any method increases holder's allowance, then the holder called it (passes)
  - ...and they meant to change the balance (passes)

```
rule onlyHolderCanChangeAllowance {

    address holder; address spender;
    mathint allowance_before = allowance(holder, spender);

    method f; env e; calldataarg args; // was: env e; uint256 amount;
    f(e, args);                        // was: approve(e, spender, amount);

    mathint allowance_after = allowance(holder, spender);

    assert allowance_after > allowance_before => e.msg.sender == holder,
        "addresses other than holder must not affect holder's allowance";
    assert allowance_after > allowance_before =>
        (f.selector == approve(address,uint).selector || f.selector == increaseAllowance(address,uint).selector),
        "only approve and increaseAllowance can increase allowances";
}
```

Summary

- You can use a `method` variable to stand in for an arbitrary method
  - Need to pass an `env` and a `calldataarg` parameter
  - Prover will verify separately on every (external) method in the contract
  - Note: rules using `method` variables are called “parametric rules.”
- You can identify `method` object `f` using `f.selector`
- The expression `P => Q` means “if `P` then `Q`” or “`P` implies `Q`”
- Some general rule patterns:
  - Generalizing rules can get good coverage quickly
  - “Unit test rules”: describe behavior of specific methods
    - e.g. `transferSpec`
  - “Stakeholder rules”: put yourself in user’s shoes
    - e.g. `onlyHolderCanChangeAllowance`
  - “Variable change rules”: describe conditions of variable changes
    - e.g. `onlyHolderCanChangeAllowance`
  - More on rule patterns tomorrow!

Exercise (~15 minutes)

We just wrote rules for allowance changes

- In certora/specs/ERC20.spec
- If allowance increases, then the sender was the holder, and the method was appropriate

Now, write rules for balance changes

- In certora/specs/ERC20.spec
- If my balance goes down, what should I know?