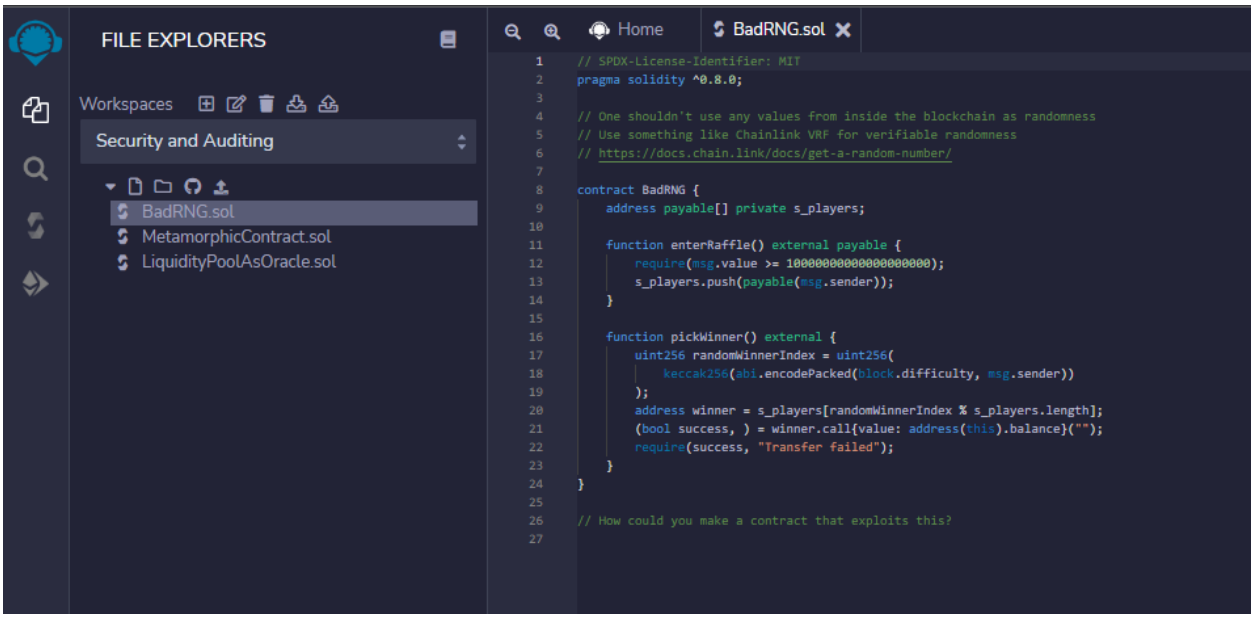
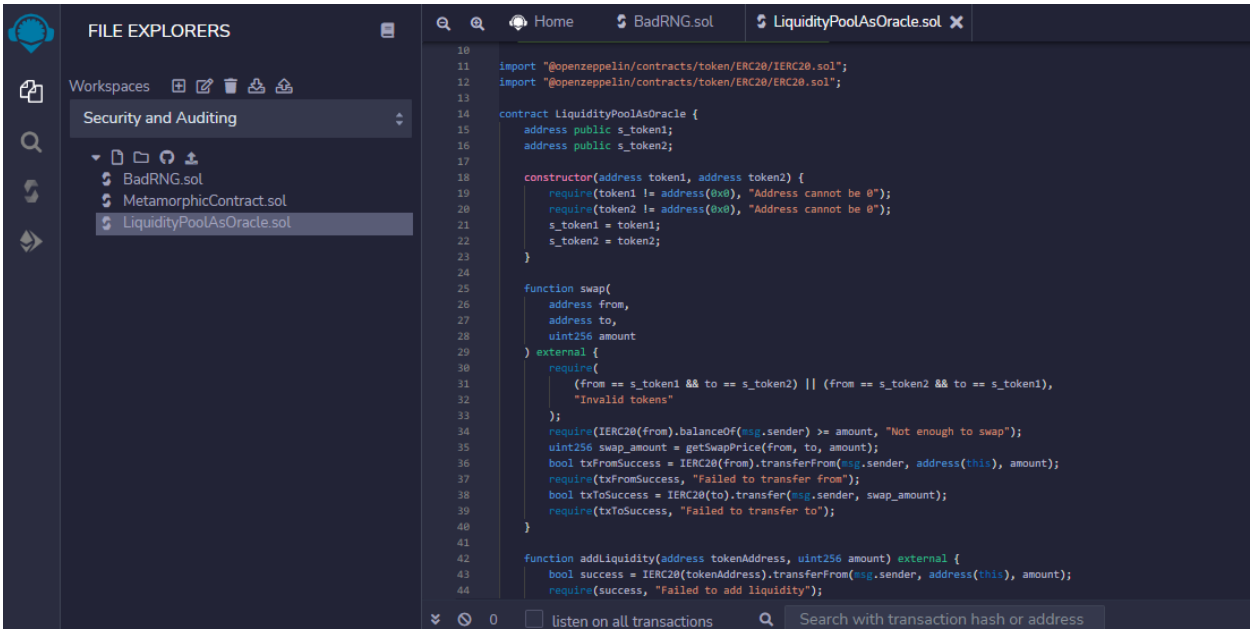
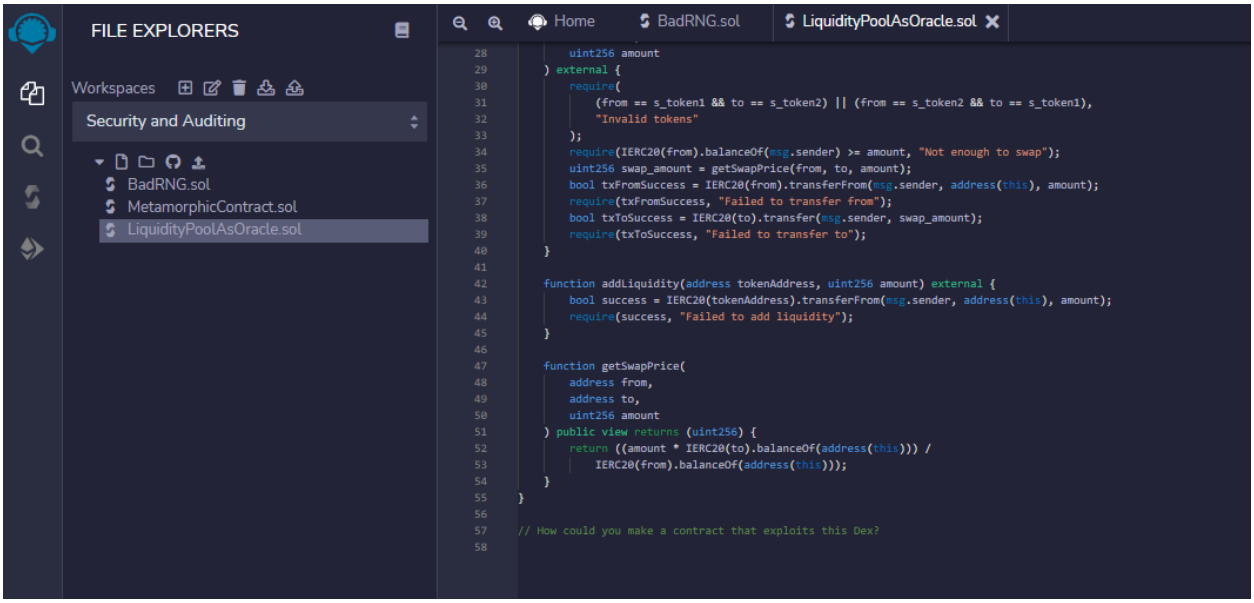
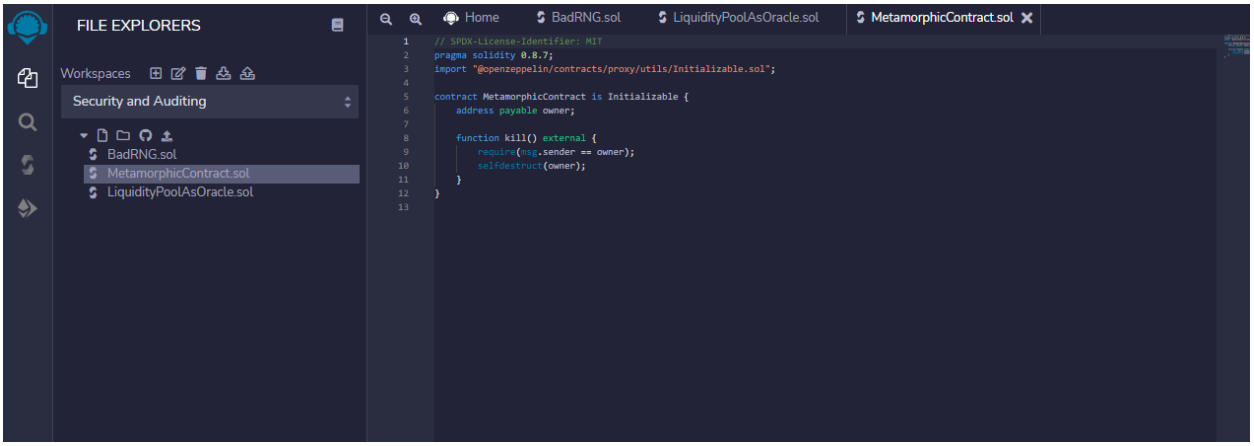
Security and Auditing






FILE EXPLORERS
Workspaces
Security and Auditing
BadRNG.sol
MetamorphicContract.sol
LiquidityPoolAsOracle.sol
Home
BadRNG.sol
LiquidityPoolAsOracle.sol
uint256 amount
) external {
require(
(from == s_token1 && to == s_token2) || (from == s_token2 && to == s_token1),
"Invalid tokens"
);
require(IERC20(from).balanceOf(msg.sender) >= amount, "Not enough to swap");
uint256 swap_amount = getSwapPrice(from, to, amount);
bool txFromSuccess = IERC20(from).transferFrom(msg.sender, address(this), amount);
require(txFromSuccess, "Failed to transfer from");
bool txToSuccess = IERC20(to).transfer(msg.sender, swap_amount);
require(txToSuccess, "Failed to transfer to");
}
function addLiquidity(address tokenAddress, uint256 amount) external {
bool success = IERC20(tokenAddress).transferFrom(msg.sender, address(this), amount);
require(success, "Failed to add liquidity");
}
function getSwapPrice(
address from,
address to,
uint256 amount
) public view returns (uint256) {
return ((amount * IERC20(to).balanceOf(address(this))) /
IERC20(from).balanceOf(address(this)));
}
}
// How could you make a contract that exploits this Dex?


FILE EXPLORERS
Workspaces
Security and Auditing
BadRNG.sol
MetamorphicContract.sol
LiquidityPoolAsOracle.sol
Home
BadRNG.sol
LiquidityPoolAsOracle.sol
MetamorphicContract.sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.7;
import "@openzeppelin/contracts/proxy/utils/Initializable.sol";
contract MetamorphicContract is Initializable {
address payable owner;
function kill() external {
require(msg.sender == owner);
selfdestruct(owner);
}
}