ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

SECURITY ASSESSMENT REPORT

NAME:	FLOORISLAVA
STATUS:	PASS
DATE:	18/01/2025

Risk Analysis

Vulnerability summary

Classification	Description
🔴 High	Vulnerabilities that can lead to **loss or theft of funds**, **permanent locking of assets**, **unauthorized minting/burning**, **protocol insolvency**, or **full control of the contract or protocol**. These issues are typically **exploitable on-chain** and require immediate remediation.
🟠 Medium	Vulnerabilities that **weaken protocol security or trust assumptions** but do not directly enable immediate loss of funds under normal conditions. Exploitation may require **specific conditions, privileged roles, or external failures**.
🟡 Low	Issues that have **limited direct impact** on funds or protocol integrity but may cause **unexpected behavior**, **reduced resilience**, or **future exploitability** if combined with other weaknesses.
🟢 Informational	Findings that **do not represent security vulnerabilities**, but highlight **code quality, clarity, maintainability, or best-practice improvements** that may enhance long-term safety and auditability.

SCOPE

Target Code And Revision

Project	Assure
Codebase	https://sepolia.basescan.org/address/0x5cfa9eee182a5294d4c1564c6d8b162db9d726c7#code Fixed version: https://sepolia.basescan.org/address/0xc4b51ccbc96ef18367139cf39be4aca61a603e26#code Deployed version: https://basescan.org/address/0x135d7797452405358ea9d38d156603F73275B99b#code
Audit Methodology	Static, Manual

Detailed Technical Report

HIGH

---

1. isExcludedFromFees skips core buy/sell rule enforcement (floor + timer + game state) [Fixed ✅]

Location:

_transfer() buy/sell branches

Issue:

The contract uses isExcludedFromFees[...] as a gate not only for taxes, but also for all the game invariants:

Buy path executes only if from == pair && !isExcludedFromFees[to]

This block contains: updateFloor(), minimum-buy enforcement, near-expiry MEV check, and (most importantly) updating lastBuyer / lastBuyTime / lastBuyAmount.

If to is excluded from fees, none of that happens.

Sell path executes only if to == pair && !isExcludedFromFees[from]

This block contains: require(gameActive), require(timer not expired), updateFloor(), require(currentPrice > floor), and the “post-sell floor breach simulation”.

If from is excluded from fees, the sell is treated like a normal transfer to the pair (with _basicTransfer for from==address(this) cases, or standard transfer otherwise), meaning floor and rug timer protections are bypassed.

This is not theoretical: it is a direct consequence of the if (to == pair && !isExcludedFromFees[from]) { ... } structure.

Remediation:

Separate concerns: introduce distinct flags, for example:

isFeeExempt[address]

isGameRuleExempt[address] (default false; ideally only the contract itself uses this)

Apply game invariants unconditionally for transfers involving the pair, regardless of fee exemption:

Always enforce gameActive, rug timer, floor checks on sells to pair (unless you explicitly want exemptions).

Always update lastBuyer/lastBuyTime and enforce min buy for buys from pair (again, unless explicitly exempt for the router/contract).

If you truly need certain system addresses to bypass some rules, whitelist them narrowly (router / contract only), and document it.

Fix: Buy/sell rule enforcement is no longer gated by !isExcludedFromFees[...].

Buy: if (from == pair) always runs game logic (timer gate + floor update + qualifying-buy handling).

Sell: if (to == pair) always runs gameActive/timer/floor/simulatePriceAfterSell checks.

2. Floor growth logic is wrong across the 90-period boundary [Fixed ✅]

Location:

updateFloor() and getCurrentFloor(), via getFloorRate()

Issue:

getFloorRate() returns a single rate based on current periodsSinceLaunch:

```
uint256 periodsSinceLaunch = (block.timestamp - gameStartTime) / TIME_UNIT;
if (periodsSinceLaunch <= PERIODS_IN_FIRST_PHASE) return 500;
else return 100;
```

But updateFloor() applies that same rate for every elapsed period since lastFloorUpdate:

```
for (i=0; i<periodsSinceUpdate; i++) {
    uint256 rate = getFloorRate(); // <-- constant within the loop
    currentFloor = (currentFloor * (10000 + rate)) / 10000;
}
```

If periodsSinceUpdate spans across the moment when the rate changes (after 90 periods), the function does not apply 5% for the earlier periods and 1% for the later periods. It applies only the rate valid “now”.

Remediation:

Compute floor growth using piecewise compounding:

Apply 5% for periods remaining in the first phase, then 1% for the rest.

Avoid loops entirely (see Finding #3) by using exponentiation in fixed-point or a precomputed multiplier table (per-day).

Fix: updateFloor() and getCurrentFloor() now compute the rate per period via _getRateForPeriod(period) inside the loop, so a span that crosses period 90 applies 5% then 1% correctly.

3. Unbounded loop in updateFloor() can permanently DoS the protocol over time [Partially Fixed ✅]

Location:

updateFloor() and getCurrentFloor()

Issue:

Both functions loop periodsSinceUpdate times. periodsSinceUpdate grows linearly with time if nobody calls updateFloor().

In production (TIME_UNIT = 1 days), multi-year inactivity means thousands of iterations.

In this test version (TIME_UNIT = 1 hours), a year is ~8,760 iterations, which is plausibly enough to exceed the block gas limit depending on the chain.

Once the loop becomes too large, any function that calls updateFloor() will revert/out-of-gas:

- _transfer() on buys and sells (when not excluded)
- sellForEth()
- checkAndExecuteRug()

That can lock the game permanently, even though anyone can call updateFloor() because they can't, due to gas.

Remediation:

Replace iterative compounding with a closed-form calculation:

floor = floor0 * (1+r)^n computed in fixed-point, or

precompute multipliers per period and use fast exponentiation (pow) with rounding control.

If you keep iteration:

enforce a hard cap on periods processed per call and allow multi-call catch-up (but beware of state drift and boundary rate issues).

Fix: Now, the contract capped iterations with MAX_FLOOR_UPDATE_PERIODS = 500, preventing “infinite growth” gas DoS. Additionally, we need to consider that both updateFloor() and getCurrentFloor() also cap at 500, meaning after long inactivity, floor growth beyond 500 periods is ignored until multiple transactions catch up.

That means the “true” intended floor may be much higher than the enforced floor, and sells could be allowed that should be blocked (security/economic invariant break) if the game is left untouched for >500 periods.

Safer fix: closed-form / fast exponentiation, or multi-call catchup but require catchup before allowing sells/buys that depend on floor.

4. Skim-buy can set lastBuyer without a real buy [Fixed ✅]

Location:

_transfer() BUY branch (if (from == pair ...)) + reliance on from == pair as “buy” signal

Issue:

An attacker can transfer LAVA to the pair (no swap), then call pair.skim(attacker). The pair sends the “excess” LAVA back to the attacker, and your token treats that from == pair transfer as a BUY: it updates lastBuyer/lastBuyTime and can satisfy minBuy based purely on amount, even though no ETH was paid and price/reserves didn’t change. This enables cheap timer resets and near-free “winner” positioning for the rug payout.

Remediation:

Don’t use from == pair as a reliable “buy happened” signal. Move game-state updates to explicit functions (for example only buyWithEth), or introduce a stricter mechanism that cannot be triggered by skim()/plain transfers (hard in ERC20). At minimum, redesign winner selection/timer updates so they can’t be advanced by non-swap pair transfers.

Fix: locked the required setup step for the classic skim-buy attack by preventing direct token transfers to the pair:

if (to == pair) require(msg.sender == router, "Use router for sells");

So an attacker can’t easily “donate” LAVA to the pair to create excess and then skim() it back.

As a potential improvement, contract can also gate buy-branch game-state updates with something stricter than from==pair (or at least add require(msg.sender == router) for from==pair when it’s meant to represent a swap).

5. Winner can brick the game by rejecting ETH [Fixed ✅]

Location:

checkAndExecuteRug() to (bool success,) = payable(winner).call{value: ethRemoved}("");
require(success, ...)l

Issue:

If lastBuyer is a contract that reverts on receiving ETH, rug execution always reverts. After the timer expires, buys and sells are blocked until the rug executes so the market can become permanently frozen (deadlock) with no recovery path.

Remediation:

Switch to a pull-payment pattern: record pendingPrize[winner] += ethRemoved and let the winner claim later via claimPrize(). Alternatively, if payout fails, do not revert escrow the prize and continue the round transition.

Fix: Rug payout is now escrowed (pendingPrize[winner] += ethRemoved) and claimed via claimPrize(). No rug execution revert due to winner refusing ETH.

6. Reentrancy during rug payout bypasses floor/timer/taxes because inSwap disables checks [Fixed ✅]

Location:

checkAndExecuteRug() uses lockTheSwap (sets inSwap=true)

_transfer() early bypass: if (inSwap || from == address(this) || to == address(this)) { _basicTransfer(...); return; }

External call: payable(winner).call{value: ethRemoved}("")

Issue:

During checkAndExecuteRug(), inSwap=true and you call the winner with ETH. The winner contract can re-enter (not into checkAndExecuteRug nonReentrant blocks that) but can call the router to swap its LAVA for ETH. When the router transfers LAVA into the pair, _transfer() sees inSwap==true and uses _basicTransfer(), skipping all sell restrictions (gameActive/timer/floor simulation) and skipping sell tax. This can allow a large dump into the pool at the exact moment your protections are intentionally disabled.

Remediation:

Do not let inSwap bypass game invariants. Remove inSwap from the bypass condition and only bypass checks for strictly internal flows (for example when from == address(this) or to == address(this)), and/or eliminate external ETH calls while inSwap is true (use escrow/pull payment for winner).

Fix: removed inSwap from the early bypass (_transfer no longer skips protections based on a swap flag).

checkAndExecuteRug() makes no external ETH call to winner anymore (pull payment), closing the reentrancy entry point that mattered.

7. Floor can be bypassed by donating tokens/WETH to the pair and swapping later [Fixed ✅]

Location:

_transfer() (SELL checks only trigger when to == pair)

simulatePriceAfterSell() / getCurrentPrice() (assume reserves represent tradable state)

Uniswap V2 Pair swap() (external, called directly by attacker; not intercepted by token)

Issue:

The contract enforces the floor only when it observes a token transfer to the pair (to == pair).

However, Uniswap V2 allows a trader to separate “transfer-in” and “swap-out” across transactions:

Attacker repeatedly calls transfer(pair, smallAmount) (each transfer passes your floor simulation since it’s small).

These transfers increase the pair’s actual token balance but do not update reserves (reserves change only on swap/mint/burn/sync).

Later, attacker calls the pair directly: pair.swap(...). The pair computes amountIn from balance delta vs reserves, so it treats the entire accumulated “donation” as input in one shot.

This executes an effective large sell that can push price below the floor, bypassing the token’s sell restrictions.

This breaks the core invariant “no sell can breach floor” and can extract materially more ETH than intended.

Remediation:

Prevent direct EOA sells/donations to the pair by enforcing router-only access for pair-bound transfers, e.g. in _transfer:

If to == pair: require(msg.sender == address(router), "Direct pair interaction disabled");

Alternatively, if you must allow direct sells, you cannot reliably enforce the floor at the token layer (because pair.swap can consume pre-loaded balances without transfers at swap time). In that case, redesign the mechanic (for example enforce via a dedicated swap contract).

Fix: Requiring msg.sender == router for to == pair transfers prevents EOAs/contracts from preloading the pair with LAVA in a standalone transfer, which is what enables the “accumulate donation then pair.swap” bypass.

MEDIUM

1. totalSupply() is constant and ignores burns [Fixed ✅]

Location:

totalSupply() and burn logic in checkAndExecuteRug()

Issue:

When burning:

```
_balances[address(this)] -= toBurn;
totalBurned += toBurn;
emit Transfer(address(this), address(0), toBurn);
```

But totalSupply() always returns TOTAL_SUPPLY and never subtracts burned tokens.

me permanently, even though anyone can call updateFloor() because they can't, due to gas.

Remediation:

Track _totalSupply mutable and decrement it on burn.

Or keep constant supply but do not emit burns as Transfer(..., address(0), ...) (that event strongly implies supply reduction). Best practice is: burning should reduce supply.

Fix: _totalSupply is introduced and decremented on burn. totalSupply() returns _totalSupply.

2. Standard Uniswap router sells can revert due to transfer tax on input token [Acknowledge]

Location:

_transfer() SELL branch

Issue:

When a user sells through the standard Uniswap V2 router swapExactTokensForETH, the router transfers amountIn from the user to the pair. Your token taxes that transfer (2%), so the pair receives only amountIn - tax.

Standard router methods assume the exact input amount reaches the pair. Because less arrives, the pair cannot satisfy the computed output without violating k, so the swap can revert. This often appears as a “can’t sell” / honeypot-like behavior unless front-ends use either:

router “supporting fee-on-transfer tokens” functions, or

your custom sellForEth() path (which avoids LAVA tax and takes ETH tax after).

Remediation:

Pick one coherent integration path:

- Option A (strongest): block direct to == pair transfers unless msg.sender == router, and require sellForEth() for sells.
- Option B: remove LAVA-side sell tax for direct router sells and only tax on ETH-out (like sellForEth).
- Option C: explicitly support router fee-on-transfer sell methods in your integration docs/interfaces and ensure UIs use them.

LOW

No low issues were found.

INFORMATIONAL

No informational issues were found.

Testing coverage

During the testing phase, custom use cases were written to cover all the logic of contracts. *Check "Annexes" to see the testing code.*

```
tests/test_lava.py::test_transfer RUNNING
Transaction sent: 0x630175e2b8028df55fadb3f2e43c4b8fd480cfcbd10f45155fb85cb188b4a285
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  UniswapV2Factory.constructor confirmed   Block: 1   Gas used: 2412742 (20.11%)
  UniswapV2Factory deployed at: 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87

Transaction sent: 0xf441fd91b57539cf07605ae1d89ac28b9d879054c215c3ae6ffceb5827765d51
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  WETH9.constructor confirmed   Block: 2   Gas used: 476546 (3.97%)
  WETH9 deployed at: 0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

Transaction sent: 0xf99c34447c2dec1afc53e0e268dee0ab9253f622233d4b39a4c948d17ad8c81a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  UniswapV2Router02.constructor confirmed   Block: 3   Gas used: 3895430 (32.46%)
  UniswapV2Router02 deployed at: 0xE7eD6747FaC5360f88a2EFC03E00d25789F69291

Transaction sent: 0xc2cdc53f1c6b7abf65add87cec03b450c081752b82e081e75f1bb1811b049926
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  LAVA.constructor confirmed (Invalid dev wallet)   Block: 4   Gas used: 394603 (3.29%)

Transaction sent: 0x362943007ef2c187967702bf1944a5fac2544c833799b85b98b05e711fdd11bd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  LAVA.constructor confirmed (Invalid presale wallet)   Block: 5   Gas used: 394643 (3.29%)

Transaction sent: 0x0d5445ee75c80fe7111b2bd4728a317cc12f7a8ce987ed52560bd43b481c4dfb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  LAVA.constructor confirmed   Block: 6   Gas used: 6343224 (52.86%)
  LAVA deployed at: 0x6b4BDe1086912A6Cb24ce3dB43b3466e6c72AFd3

Transaction sent: 0x25cc51243c9c5280bdbc1859b00f8177e2a647c729de1a835bd32bc1fb10791d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  LAVA.transfer confirmed (Transfer from zero)   Block: 7   Gas used: 23077 (0.19%)

Transaction sent: 0xd23f467add9e07c10de8896710269b21d12e73016bc9c0e21b714aefb7d3bdfd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  LAVA.transfer confirmed (Transfer to zero)   Block: 8   Gas used: 22872 (0.19%)

Transaction sent: 0x49fe4128fe233602861fd908a228310cbb3451d6d26f7c15b44d9058e63970cf
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  LAVA.transfer confirmed (Zero amount)   Block: 9   Gas used: 23063 (0.19%)

Transaction sent: 0x3157010c12d497bfd56106fd19e2b4c04837e3ed3bfa86dc6e0033d63430ddc0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  LAVA.transfer confirmed (Insufficient balance)   Block: 10   Gas used: 24048 (0.20%)

Transaction sent: 0xd32ed87192d23e0c2e026a5d8906f219a2bea31f62b70f9d47b7ef34640999d8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  LAVA.transfer confirmed (Trading not enabled)   Block: 11   Gas used: 26658 (0.22%)

Transaction sent: 0x9306199ac4bbb9684cb4b932367891f052af4cdb9ef276c6bd51f931e51195fe
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  LAVA.transfer confirmed   Block: 12   Gas used: 58578 (0.49%)

Transaction sent: 0x93b9851b74fec68f1b3ef10af8747ab44e4c81a8d1fa1f8cea409731c3d10238
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  LAVA.transfer confirmed (Game not active)   Block: 13   Gas used: 30487 (0.25%)

tests/test_lava.py::test_transfer PASSED
```

```
tests/test_lava.py::test_trading RUNNING
Transaction sent: 0x8ec6efa68a381d3c9dc45702ba2210f27a3144bb6ca3a3f6e60ccabfbc5d8d10
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  UniswapV2Factory.constructor confirmed   Block: 14   Gas used: 2412742 (20.11%)
  UniswapV2Factory deployed at: 0xcCB53c9429d32594F404d01fbe9E65ED1DCda8D9

Transaction sent: 0x70db79be55f1a1871d85b6a9daf9ccc221641be478fb946b46e6e0897fc38a08
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  WETH9.constructor confirmed   Block: 15   Gas used: 476546 (3.97%)
  WETH9 deployed at: 0x420b1099B9eF5baba6D92029594eF45E19A04A4A

Transaction sent: 0x676a6bb38e6867e8280d3296359b2f728c7313cd7d1d1d6f3e421eedde86650c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  UniswapV2Router02.constructor confirmed   Block: 16   Gas used: 3895430 (32.46%)
  UniswapV2Router02 deployed at: 0xa3B53dDCd2E3fC28e8E130288F2aBD8d5EE37472

Transaction sent: 0x2495da126c366b6503238f32413b428637ad2bde992eec289e6451966e17ee26
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  LAVA.constructor confirmed   Block: 17   Gas used: 6343236 (52.86%)
  LAVA deployed at: 0xb6286fAFd0451320ad6A8143089b216C2152c025

Transaction sent: 0xf494a90a0bc4e66f114df5dd382b87440c4bac4267f979852a486f67c84ade29
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  LAVA.startTrading confirmed (Not owner)   Block: 18   Gas used: 22250 (0.19%)

Transaction sent: 0x93ff5b4a3d4ba89b63e5f7a0c2a560daf7704207f6004a16e86a0ec1ce37f479
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  LAVA.startTrading confirmed (Add liquidity first)   Block: 19   Gas used: 25171 (0.21%)

Transaction sent: 0x274f41dcce09a39a820c2ebef7d519d43bdc23e88d30d01bdfd6add9cb94b53b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  LAVA.addLiquidity confirmed (Not owner)   Block: 20   Gas used: 22569 (0.19%)

Transaction sent: 0xeec282a8c77a41fc47ae86fe087af937c35065e156285e0e7a18b1f54b16fd5d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  LAVA.addLiquidity confirmed (Need ETH)   Block: 21   Gas used: 28412 (0.24%)

Transaction sent: 0x2687747b4da0d7016bfd10e1a0185777378a3185726ca4b1b9f9ad6604f44b02
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  LAVA.addLiquidity confirmed   Block: 22   Gas used: 252299 (2.10%)

Transaction sent: 0x106da8abf5eda2f33b7fafd6256a28332f0e1943e02c028fb04470b68b4a4462
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  LAVA.addLiquidity confirmed (Exceeds initial liquidity allocation)   Block: 23   Gas used: 29323 (0.24%)

Transaction sent: 0x11a0d74a11541b9c5d4394a350dfe999a673a1c98c5ed4ddc6248f7a9177779f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  LAVA.addLiquidity confirmed (Insufficient non-treasury balance)   Block: 24   Gas used: 31083 (0.26%)

Transaction sent: 0xb4a099da63c0ad91b5ca1962e20fb5ea5a01ef4cb8eb3fb9495719f81784278a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  LAVA.startTrading confirmed   Block: 25   Gas used: 137148 (1.14%)

Transaction sent: 0x9a2131af91eb2c29cd96444311cb9145b826c01f7d90cbc88023523dee459bac
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  LAVA.transfer confirmed   Block: 26   Gas used: 58044 (0.48%)

Transaction sent: 0xb8779db986eb8d5b31c8cb1447a0762898c910561aef49e800f687f456a02d30
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  LAVA.transfer confirmed   Block: 27   Gas used: 59123 (0.49%)

Transaction sent: 0x19e61f893745c92c348a893f25689c0fd7dd8fab616a64ba8d41d2d0a1300675
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  LAVA.transfer confirmed (Exceeds max wallet)   Block: 28   Gas used: 27989 (0.23%)

Transaction sent: 0x02a64c2a170cc5fa7f7923acc0afb733258e2a9ed09ffffc03cc8df6d5946566
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  LAVA.buyWithEth confirmed   Block: 29   Gas used: 208464 (1.74%)

Transaction sent: 0x9ad04077487bf122ab4cf3185691da2fa27b397bc703bc7811be20c22fa28cd5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  LAVA.sellForEth confirmed (No amount)   Block: 30   Gas used: 29500 (0.25%)

Transaction sent: 0x0b79b161946fcd91750f1d7cd45ef6aa1f19fdc21ddeb16cdc1c9f77b16ca13d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  LAVA.sellForEth confirmed (Insufficient balance)   Block: 31   Gas used: 30514 (0.25%)

Transaction sent: 0x31d35194bc2fddfdd21b4ef8bdff8f1e77b56678e29bd9d0fb3ddcfb0a4697af
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  LAVA.sellForEth confirmed   Block: 32   Gas used: 226076 (1.88%)

tests/test_lava.py::test_trading PASSED
```

```
tests/test_lava.py::test_floor_price_blocks RUNNING
Transaction sent: 0x6dc2a8c1097c7934aa95ee9059c837660babadd07c65ef1802b73dd7e668561c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  UniswapV2Factory.constructor confirmed   Block: 33   Gas used: 2412742 (20.11%)
  UniswapV2Factory deployed at: 0xFbD588c72B438faD4Cf7cD879c8F730Faa213Da0

Transaction sent: 0xc1c7dd122437478f6a54e0bcc33ac88e1115f3f75529745dadde1610f32b4bdb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  WETH9.constructor confirmed   Block: 34   Gas used: 476546 (3.97%)
  WETH9 deployed at: 0xed00238F9A0F7b4d93842033cdF56cCB32C781c2

Transaction sent: 0x4a3718682bbe911705bc2fdb94a8616d532821793ae6b17bf61dc9295b79faba
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  UniswapV2Router02.constructor confirmed   Block: 35   Gas used: 3895418 (32.46%)
  UniswapV2Router02 deployed at: 0xDae02e4fE488952cFB8c95177154D188647a0146

Transaction sent: 0xf912b5d594941626645f5b32b52f843e337a8169fe304e0d12ae7123356e2293d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 20
  LAVA.constructor confirmed   Block: 36   Gas used: 6343226 (52.86%)
  LAVA deployed at: 0xdCF93F11ef216cEC9C07fd31dD801c9b2b39Afb4

Transaction sent: 0x6aa652e090fcdfddfaf94ea451e1471fbc669e30e0169ae43c8ae4a06dfc8231
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  LAVA.buyWithEth confirmed (Trading not enabled)   Block: 37   Gas used: 29345 (0.24%)

Transaction sent: 0xde5d84178d9c0fe0b30698d78fc15a0f748f1fc2da6a532f063d41f314c84838
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 21
  LAVA.addLiquidity confirmed   Block: 38   Gas used: 251157 (2.09%)

Transaction sent: 0x8b6f8e09ba65ba934a6d3c3bad499be5de2e72609e2c63b3b43c80b472bf66bd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 22
  LAVA.startTrading confirmed   Block: 39   Gas used: 137147 (1.14%)

Transaction sent: 0xde9f13857fe915b7425056b2005268188d4adb53b04311366f62b08a49896a34
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  LAVA.buyWithEth confirmed (No ETH sent)   Block: 40   Gas used: 29346 (0.24%)

Transaction sent: 0x4220b54150bb635dfb3a8abd176cc974501bb407d155fe2fbd7ba28bf9d003f8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  LAVA.buyWithEth confirmed (Below minimum buy)   Block: 41   Gas used: 33395 (0.28%)

Transaction sent: 0xe7e345c541c20b45b79d5e86df4d0ec012cce663beb298f4ac85365aebe739ba
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  LAVA.buyWithEth confirmed   Block: 42   Gas used: 223554 (1.86%)

Transaction sent: 0xd861e1cd2adb1fe55f2bccc2b096bd0487b961370d1a62ff92f17b5085ccea21
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  LAVA.sellForEth confirmed (Rug timer expired - execute rug first)   Block: 44   Gas used: 32240 (0.27%)

Transaction sent: 0xdeb8ae2a27c8c8fe3fab99619fd631fdfd01a01e3cc0daa1b72567afead4c09d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 23
  LAVA.checkAndExecuteRug confirmed   Block: 45   Gas used: 1112837 (9.27%)

Transaction sent: 0xf9806922bc913f75bb0e81be51bb712b305bcfd09b8a6ccae0eb261a3d4d78f8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  LAVA.sellForEth confirmed (Price at or below floor)   Block: 46   Gas used: 42421 (0.35%)

tests/test_lava.py::test_floor_price_blocks PASSED
```

```
tests/test_lava.py::test_set_fee_wallet RUNNING
Transaction sent: 0x46e8e3ab8e2b57d0ebf51fde062fbf53d43b152fdfad4eec2330a6c7682a6cc4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 24
  UniswapV2Factory.constructor confirmed   Block: 47   Gas used: 2412742 (20.11%)
  UniswapV2Factory deployed at: 0xf9C8Cf55f2E520B08d869df7bc76aa3d3ddDF913

Transaction sent: 0x662a967656dee69fbf5ebac96ec1bd0e4df1564ca7f2f08409e90e796776ba57
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 25
  WETH9.constructor confirmed   Block: 48   Gas used: 476546 (3.97%)
  WETH9 deployed at: 0x654f70d8442EA18904FA1AD79114f7250F7E9336

Transaction sent: 0x5e8d9478560f9b450bbf3c5006c9dd1d0f374045ab700a78f2fdc862329d73d7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 26
  UniswapV2Router02.constructor confirmed   Block: 49   Gas used: 3895430 (32.46%)
  UniswapV2Router02 deployed at: 0xADeD61D42dE86f9058386D1D0d739d20C7eAfC43

Transaction sent: 0x5bc5a7e86ae9644568d7ba69430f2699b30eb1014ae6b4bf67757aff468c9fbf
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 27
  LAVA.constructor confirmed   Block: 50   Gas used: 6343236 (52.86%)
  LAVA deployed at: 0x832698Daec363C9A7aB036C224Af5B21280b3AC6

Transaction sent: 0x495e75cd170ade442ba8abcc6863fe21c0ad9d93a5fa4bb03f19381b9390e98c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  LAVA.setFeeWallet confirmed (Not owner)   Block: 51   Gas used: 22824 (0.19%)

Transaction sent: 0x01741de443ed8fcef962ff06a7771bb9060d2e1538e71ff22bac2b4bbb65c7e9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 28
  LAVA.setFeeWallet confirmed (Invalid fee wallet)   Block: 52   Gas used: 22610 (0.19%)

Transaction sent: 0x0dd9afa0c641dbc75ca537de4e3b4b6c3dfba2db8ac56636c1a9c8255343c418
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 29
  LAVA.setFeeWallet confirmed   Block: 53   Gas used: 64584 (0.54%)

tests/test_lava.py::test_set_fee_wallet PASSED
```

Annexes

Testing code:

```
from brownie import (
    reverts,
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    DAY_TIMESTAMP,
    get_account,
    get_timestamp,
    increase_timestamp
)

from scripts.deploy import (
    deploy_factory,
    deploy_router,
    deploy_weth,
    deploy_erc,
    deploy_lava
)

'''
TOTAL_SUPPLY = 1_000_000_000 * 10**18;
TREASURY_SUPPLY = 600_000_000 * 10**18;
DEV_SUPPLY = 200_000_000 * 10**18;
PRESALE_SUPPLY = 100_000_000 * 10**18;
INITIAL_LIQUIDITY = 100_000_000 * 10**18;

MAX_WALLET_PERCENT = 5;                    MAX_TX_PERCENT = 2;
MEV_BLOCKS = 1;                            BURN_PERCENT = 20;
TREASURY_RETURN_PERCENT = 80;              REFILL_TARGET = 100_000_000 * 10**18;
BUY_TAX_BPS = 200; // 2% tax on buys       SELL_TAX_BPS = 200; // 2% tax on sells
MAX_MIN_BUY = 0.02 ether; // Production: 0.5 ether      FLOOR_MULTIPLIER = 4;
```

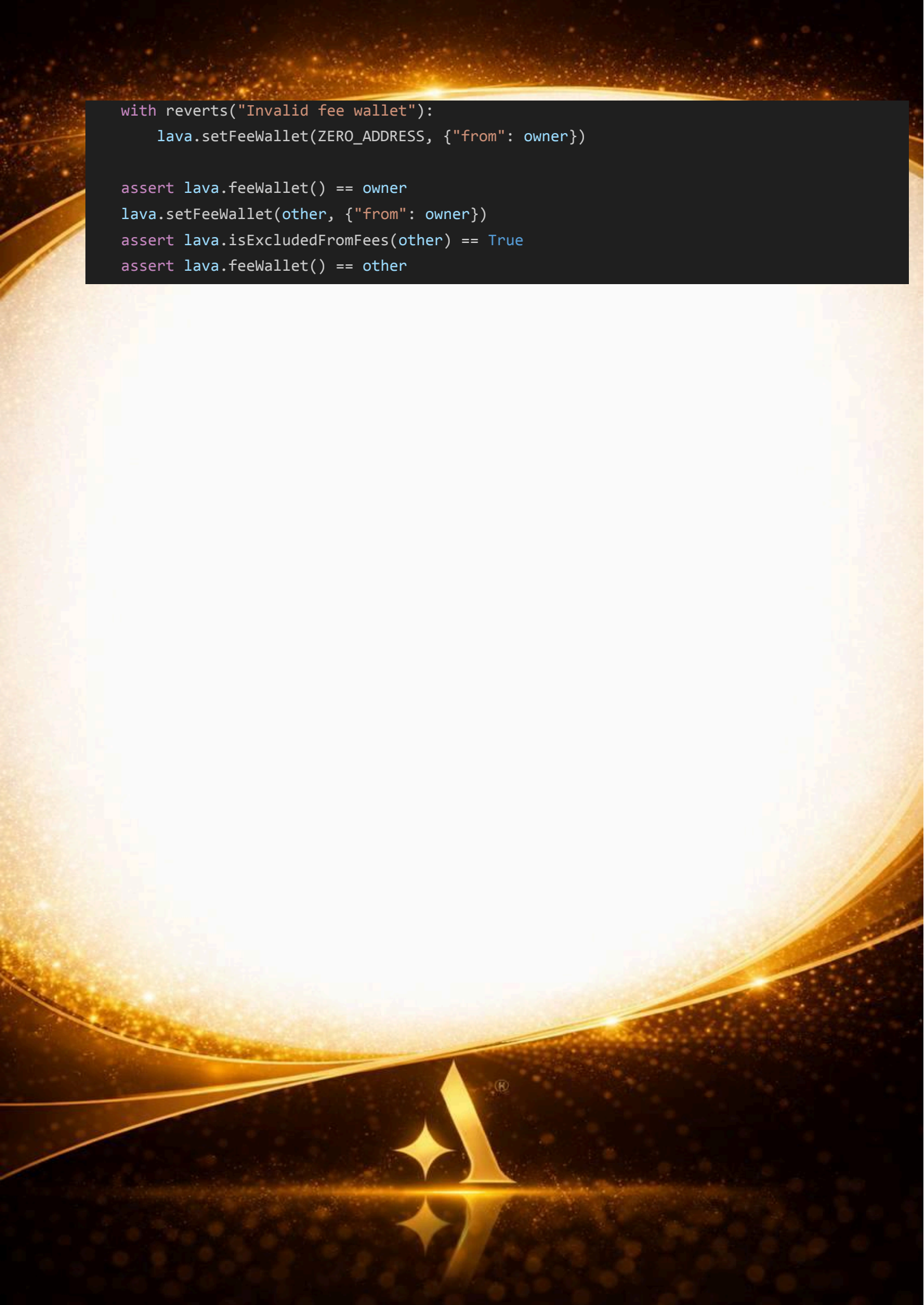
```
'''

def test_transfer(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    dev_wallet = get_account(3)
    presale_wallet = get_account(4)

    factory = deploy_factory(owner, owner)
    weth = deploy_weth(owner)
    router = deploy_router(owner, factory.address, weth.address)

    with reverts("Invalid dev wallet"):
        deploy_lava(owner, ZERO_ADDRESS, presale_wallet, router.address)
    with reverts("Invalid presale wallet"):
        deploy_lava(owner, dev_wallet, ZERO_ADDRESS, router.address)

    lava = deploy_lava(owner, dev_wallet, presale_wallet, router.address)
    pair = lava.pair()

    with reverts("Transfer from zero"):
        lava.transfer(other, 1e18, {"from": ZERO_ADDRESS})
    with reverts("Transfer to zero"):
        lava.transfer(ZERO_ADDRESS, 1e18, {"from": extra})
    with reverts("Zero amount"):
        lava.transfer(other, 0, {"from": extra})
    with reverts("Insufficient balance"):
        lava.transfer(other, 1e18, {"from": extra})
    with reverts("Trading not enabled"):
        lava.transfer(other, 1e18, {"from": dev_wallet})

    tx = lava.transfer(owner, 50000000e18, {"from": dev_wallet})
    assert tx.events['Transfer'][0]['from'] == dev_wallet
    assert tx.events['Transfer'][0]['to'] == owner
    assert tx.events['Transfer'][0]['value'] == 50000000e18

    with reverts("Game not active"):
        lava.transfer(pair, 1e18, {"from": owner})

    # max_wallet 50 000 000
    # max_tx 20 000 000

def test_trading(only_local):
```

```
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    dev_wallet = get_account(3)
    presale_wallet = get_account(4)

    factory = deploy_factory(owner, owner)
    weth = deploy_weth(owner)
    router = deploy_router(owner, factory.address, weth.address)

    lava = deploy_lava(owner, dev_wallet, presale_wallet, router.address)
    pair = lava.pair()

    with reverts("Not owner"):
        lava.startTrading({"from": other})
    with reverts("Add liquidity first"):
        lava.startTrading({"from": owner})

    with reverts("Not owner"):
        lava.addLiquidity(1e18, {"from": other})
    with reverts("Need ETH"):
        lava.addLiquidity(1000e17, {"from": owner})

    lava.addLiquidity(1000000e18, {"from": owner, "value": 100e18})

    with reverts("Exceeds initial liquidity allocation"):
        lava.addLiquidity(100000001e18, {"from": owner, "value": 1e18})
    with reverts("Insufficient non-treasury balance"):
        lava.addLiquidity(100000000e18, {"from": owner, "value": 1e18})

    lava.startTrading({"from": owner})

    tx = lava.transfer(other, 10e18, {"from": dev_wallet})
    assert tx.events['Transfer'][0]['from'] == dev_wallet
    assert tx.events['Transfer'][0]['to'] == other
    assert tx.events['Transfer'][0]['value'] == 10e18

    tx = lava.transfer(extra, 2e18, {"from": other})
    assert tx.events['Transfer'][0]['from'] == other
    assert tx.events['Transfer'][0]['to'] == extra
    assert tx.events['Transfer'][0]['value'] == 2e18

    with reverts("Exceeds max wallet"):
        lava.transfer(other, 50000000e18, {"from": dev_wallet})
```

```
    # buy
    tx = lava.buyWithEth(0, get_timestamp() + 60, {"from": other, "value": 1e18})
    assert tx.events['EthTaxCollected'][0]['user'] == other
    assert tx.events['EthTaxCollected'][0]['ethAmount'] == (1e18 * 0.02)
    assert tx.events['EthTaxCollected'][0]['isBuy'] == True

    # sell
    with reverts("No amount"):
        lava.sellForEth(
            0,
            0,
            get_timestamp() + 60,
            {"from": other}
        )
    with reverts("Insufficient balance"):
        lava.sellForEth(
            lava.balanceOf(other) + 1e18,
            0,
            get_timestamp() + 60,
            {"from": other}
        )

    tx = lava.sellForEth(
        lava.balanceOf(other) // 2,
        0,
        get_timestamp() + 60,
        {"from": other}
    )
    assert tx.events['EthTaxCollected'][0]['user'] == other
    assert tx.events['EthTaxCollected'][0]['isBuy'] == False


def test_floor_price_blocks(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    dev_wallet = get_account(3)
    presale_wallet = get_account(4)

    factory = deploy_factory(owner, owner)
    weth = deploy_weth(owner)
    router = deploy_router(owner, factory.address, weth.address)
```

```
    lava = deploy_lava(owner, dev_wallet, presale_wallet, router.address)
    pair = lava.pair()

    with reverts("Trading not enabled"):
        lava.buyWithEth(0, get_timestamp() + 60, {"from": other, "value": 1e18})

    lava.addLiquidity(5e18, {"from": owner, "value": 10e18})
    lava.startTrading({"from": owner})

    with reverts("No ETH sent"):
        lava.buyWithEth(0, get_timestamp() + 60, {"from": other, "value": 0})

    with reverts("Below minimum buy"):
        lava.buyWithEth(0, get_timestamp() + 60, {"from": other, "value": 1})

    lava.buyWithEth(0, get_timestamp() + 60, {"from": other, "value": 1e17})
    balance = lava.balanceOf(other)

    increase_timestamp(DAY_TIMESTAMP * 7)

    with reverts("Rug timer expired - execute rug first"):
        lava.sellForEth(balance, 0, get_timestamp() + 60, {"from": other})

    lava.checkAndExecuteRug({"from": owner})


    with reverts("Price at or below floor"):
        lava.sellForEth(balance, 0, get_timestamp() + 60, {"from": other})

def test_set_fee_wallet(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    dev_wallet = get_account(3)
    presale_wallet = get_account(4)

    factory = deploy_factory(owner, owner)
    weth = deploy_weth(owner)
    router = deploy_router(owner, factory.address, weth.address)

    lava = deploy_lava(owner, dev_wallet, presale_wallet, router.address)

    with reverts("Not owner"):
        lava.setFeeWallet(extra, {"from": other})
```

```
    with reverts("Invalid fee wallet"):
        lava.setFeeWallet(ZERO_ADDRESS, {"from": owner})

    assert lava.feeWallet() == owner
    lava.setFeeWallet(other, {"from": owner})
    assert lava.isExcludedFromFees(other) == True
    assert lava.feeWallet() == other
```



Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	7				1	6
Medium	2			1		1
Low						
Informational						

Assessment Results

Score Results

Review	Score
Global Score	**85/100**
Assure KYC	Not completed
Audit Score	85/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit PASS

Following our comprehensive security audit of the token contract for the **FLOORISLAVA** project, the project did not fulfill the necessary criteria required to pass the security audit. **After the development team's review, all critical vulnerabilities have been addressed/reviewed, and the audit results are satisfactory.**

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial Token in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies. All information provided in this report does not constitute financial or investment in Token, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report. The assessment of Tokens provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any Tokens, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The Token may access, and depend upon, multiple layers of third parties.