PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For DegenDojo

01 February 2022


paladinsec.co


info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that 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 completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin 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 and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Degen Dojo on the Binance Smart Chain. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	DegenDojo
URL	http://degendojo.io
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
DegenDojo (Token)	0xbeCE7EdDDd7aB1B7cfD6C93703c3E34C7182b33E	MATCH
DojoHouse	0x258bb545c2a58882a3b1d1c83ef667f3cbc2dd1c	MATCH
DojoBar	0x790ce67ec6b53dc9f18284f133d27a1c51c1cbbf	MATCH
DojoRouter	0x7c82706ef543dc969dd53e8e263ceb6fc0efe191	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	2	2	-	-
Low	5	4	1	-
Informational	24	22	2	-
Total	**33**	**30**	**3**	**-**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 DegenDojo

ID	Severity	Summary	Status
01	HIGH	An exploiter can frontrun Chainlink fulfillments to always let small trades win	RESOLVED
02	HIGH	The governance claim function does not set jackpotPaid	RESOLVED
03	MEDIUM	Usage of `tx.origin` for key business logic is prone to phishing	RESOLVED
04	LOW	Lack of parameter validation	RESOLVED
05	LOW	`getMinimumTradeSize` lacks safeguards for several edge cases	RESOLVED
06	LOW	Several variables are private	RESOLVED
07	INFO	Division before multiplication is done which could cause excessive rounding within `getMinimumTradeSize`	RESOLVED
08	INFO	Typographical errors	RESOLVED
09	INFO	Trade size check does not allow trades of the minimum or maximum trade size	RESOLVED
10	INFO	`ethUsdPriceFeed` and `linkUsdPriceFeed` can be made immutable	RESOLVED
11	INFO	Several functions can be made external	RESOLVED
12	INFO	Lack of events for several functions	RESOLVED
13	INFO	Jackpot mechanism does not adhere to VRF requirements	RESOLVED

1.3.2 DojoHouse

ID	Severity	Summary	Status
14	MEDIUM	Balance will almost never be zero because it includes `msg.value`	RESOLVED
15	INFO	Inconsistent usage of uint256	RESOLVED
16	INFO	`enter`, `leave` and `getValue` can be made `external`	RESOLVED
17	INFO	Contracts cannot execute the `enter` function but they can still execute `transfer` and `transferFrom`	RESOLVED
18	INFO	`treasury` can be made immutable	RESOLVED
19	INFO	`treasury` and `bar` are private	RESOLVED
20	INFO	`getValue` is imprecise and can revert because of division by zero	RESOLVED
21	INFO	Typographical errors	PARTIAL

1.3.3 DojoBar

ID	Severity	Summary	Status
22	LOW	`collectFees` is vulnerable to sandwich attacks	RESOLVED
23	INFO	Constructor variables can be made `IERC20` and `IUniswapV2Router02`	RESOLVED
24	INFO	`router` can be made immutable	RESOLVED
25	INFO	Gas optimization: Swap deadline does not need to be higher than `block.timestamp`	RESOLVED
26	INFO	Several functions can be made external	RESOLVED
27	INFO	Contracts cannot execute the enter function but they can still execute `transfer` and `transferFrom`	RESOLVED
28	INFO	Unnecessary bracket use	RESOLVED
29	INFO	Unused variable: `amountOut`	RESOLVED
30	INFO	`getRatio` is imprecise	RESOLVED
31	INFO	Typographic errors: `xDoji`	RESOLVED

1.3.4 DojoRouter

ID	Severity	Summary	Status
32	LOW	`dojo`, `AddressToETHTrade` and `WETH` are private	PARTIAL
33	INFO	Typographical errors	PARTIAL

2 Findings

2.1 DegenDojo

The DegenDojo token is an ERC-20 token which contains advanced gambling functionality.

The contract has two functions: `initiateTrade` and `initiateSmallTrade` which require the user to wager BNB after which the contract requests a random number from Chainlink. Once Chainlink fulfills this request with a random number, the user can withdraw the BNB they won based on the gambling result which is calculated in the DojoHouse contract. The user can also specify a belt type to indicate what sort of odds and payoffs they want to play with. Throughout the contract, gambles are called "trades".

The different belts that can be used are:

For a spin trade
- 1 = white belt
- 2 = blue belt
- 3 = black belt

For an all or nothing trade
- 4 = white belt
- 5 = blue belt
- 6 = black belt

To summarize, DegenDojo is an ERC-20 token which also acts as the user interface into the gambling functionality of the Dojo contracts. Specifically, it is the entrypoint for all gambling logic which gets executed in the DojoHouse.

A configurable initial supply is minted to the contract creator.

The minimum size of a gamble must be 100 times the Chainlink LINK cost to generate the random number. As the fee is paid in LINK but the gamble is done in BNB, Chainlink oracles are used to calculate this minimum size using the most recent on-chain prices of the tokens in question. There is a maximum trade size which is presently set at 1/20th of the gas tokens in the House contract. The comments indicate that this will be lowered to 1/100th in production. It should be noted that by itself this limit does not guarantee that the House will always have sufficient tokens.

Users cannot start a gamble if they already have one pending. Users are defined as the transaction origin wallet, which means that even if a user starts a gamble through some contract like the router, they would still be considered as the party that initiates the gamble. This has some phishing consequences as untrusted contracts might unexpectedly claim or initiate gambles for you. The initiation would however not take tokens from your balance fortunately.

If users do not have enough BNB to initiate a trade, they can initiate a small trade/gamble. Small gambles will use the randomness fulfillment of normal gambles to calculate their outcome. This way, they do not need to pay a Chainlink fee.

Once the user has initiated a large trade/gamble, their address is assigned a request ID (which is the Chainlink randomness ID) and at this point the user cannot create subsequent requests/gambles. Small trades will temporarily be assigned the `bytes32("smallTrade") requestId`, which is a temporary dummy that is later replaced when a large gamble comes in. Up to 50 trades/gambles are possible at a time.

As soon as Chainlink fulfills the randomness request, the contract links the request to the randomness value. At this point, the user can claim the trade.

Whenever a wager is made, the user is also subscribed to the spin jackpot where they have a chance to win a large jackpot of Dojo tokens. This is the main source of Dojo emissions apart from the initial supply. There are 8 jackpots in total and the user

participates in all eight at once. Each jackpot is 2 times less likely to be hit then the previous one and all of them accumulate Dojo rewards over time. If a jackpot is not won for a long time, it will become significantly large. Governance can adjust the rate at which the jackpots accumulate and any win takes half of the total accumulated jackpot amount, while the other half is combined with subsequent accumulations for the next jackpot.

The contract specifies a 9th jackpot which can only be claimed by governance and which takes 10% of the emission rate. Governance therefore receives 10% of the emissions. Jackpot winnings are given to `tx.origin` at the time of audit.

2.1.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `initiateSetHouse`
- `setHouse`
- `setKeyhash`
- `setRewardsRate`
- `claim`
- `transferOwnership`
- `renounceOwnership`
- `addRouter`

2.1.2 Issues & Recommendations

Issue #01	**An exploiter can frontrun Chainlink fulfillments to always let small trades win**
Severity	HIGH SEVERITY
Description	Presently after a large trade/gamble has requested a random number from Chainlink, the fulfillment (the response transaction of this request) of the random number will pick up any pending small trades and assign these the random number. This is an extremely severe issue that should not be underestimated as anyone with mempool access or even simply access to the Chainlink network can know this random number beforehand and add themselves to the list of small trades if they would win. A malicious actor can and eventually will figure this out and abuse this mechanism to consistently win their trades and eventually drain the house of all gas tokens supplied to it.
Recommendation	When using Chainlink, it is of utmost importance that the outcome of a gamble is completely determined at the time of requesting the randomness, given the random value that will be returned. This means that the small trades should be assigned at the time of the randomness request and not at the time of fulfillment. The client should also consider stepping away from using `msg.value` to rehash the values, as this is unnecessarily allows the user to potentially affect the randomness. It additionally causes outcomes to be correlated if multiple small trades have the same `msg.value`. It is better to simply use an incremental nonce to rehash the randomness.
Resolution	RESOLVED The small trades are now assigned a request as soon as the request is created and not fulfilled.

Issue #02	The governance `claim` function does not set `jackpotPaid`
Severity	HIGH SEVERITY

Location	Lines 360-362
	`function claim() external onlyOwner {`
	`    _mint(owner(), viewJackpots(8));`
	`}`

Description	The governance `claim` function is used to mint 10% of the emissions to the governance wallet. However, governance can call this multiple times to each time receive 10% of the emissions since contract genesis because the `jackpotPaid` variable is not increased on this call. This allows a malicious or hacked governance wallet to mint and dump a very large amount of tokens.
Recommendation	Consider increasing the `jackpotPaid` variable after the mint.
Resolution	RESOLVED `jackpotPaid` is now incremented.

Issue #03	Usage of `tx.origin` for key business logic is prone to phishing
Severity	MEDIUM SEVERITY
Description	Most contracts nowadays use `msg.sender` to indicate the user. This is the contract or wallet that called the DegenDojo contract. If you transfer some random token and this token contract calls DegenDojo, the `msg.sender` is that token and not your wallet. However, when using `tx.origin`, the wallet is always the transaction creator, e.g. your user wallet. This is risky because a malicious party could airdrop tokens to all users that have pending trades with these tokens calling DegenDojo when they are sold, and could allow the exploiter to claim your pending payout if you transferred or sold their exploit token. Examples of how this has occurred in practice and has led to the loss of millions can be read in the following interesting blog post: https://www.adrianhetman.com/unboxing-tx-origin/
Recommendation	Consider not using `tx.origin` as the user domain. Instead, consider providing the user as a parameter but only allowing whitelisted contracts to make interactions for users other than `msg.sender`.
Resolution	RESOLVED The client now also sends the earnings to `tx.origin`. It should be noted that we have discussed with the client that they should avoid using `tx.origin` altogether in their subsequent projects as this is considered extremely bad practice. We have marked this issue as resolved as the major exploit vector is mitigated.

Issue #04	Lack of parameter validation
Severity	LOW SEVERITY
Description	Currently the contract insufficiently validates the parameters of certain functions. Specifically the `_belt` parameter from `initiateTrade` should be validated to lie within [1, 6]. It can also be changed to an `uint256` as `uint8` does not save gas, but on the contrary causes extra gas usage as it requires conversion logic to be compiled in. The lack of a sensible minimum `msg.value` within `initiateSmallTrade` could also cause a malicious party to constantly fill the queue of small trades and deny service to others. The `smallTrades` length check on line 178 is an inclusive check, which causes the real maximum number of small trades to be 51 as one more small trade is permitted at this point, the one of the call itself. We believe the developer might have intended 50 small trades at most and therefore the developer should consider using a smaller than check in this requirement. The `newRequest` value returned by Chainlink on line 155 is not validated. If this were ever `bytes32("smallTrade")` or all zeros, this would cause the contract to severely malfunction. The `msg.value` is not validated to be non-zero which could cause `_spinJackpot` to revert due to a division by zero, causing the user to be permanently locked out of the contract.
Recommendation	Consider adding the above validation and changing the belt to an `uint256` in all locations. In general, parameters should always be `uint256`, unless they save memory in a struct due to tight packing. Consider validating that `newRequest` does not equal `bytes32(0)` and `bytes32("smallTrade")` as an extra precaution. This is highly improbable and will also be resolved on the note that the client has verified that this can never occur.

Resolution	RESOLVED The client has indicated they do not want to restrict `_belt` so they can allow for House upgrades in the future. A sensible minimum has been added to `initiateSmallTrade` to prevent malicious actors. The inclusive check has been fixed to be made exclusive allowing for exactly 50 small trades.

Issue #05	`getMinimumTradeSize` lacks safeguards for several edge cases
Severity	LOW SEVERITY
Description	The `getMinimumTradeSize` function currently does not handle the following cases: 1. What if the Chainlink oracles have different decimals? 2. What if the LINK token does not have 18 decimals? 3. What if the Chainlink oracle returns a zero or smaller price? 4. What if the Chainlink oracle response is stale (outdated)?
Recommendation	Consider explicitly handling each of these different situations: 1. Consider explicitly validating that the two oracles have identical decimals. As this is nearly always the case, this is easier than adding logic to handle oracles with different decimals. 2. Consider validating that this is the case in the constructor, we believe this is the case for all LINK tokens we are aware of. 3. Consider reverting in this scenario, this should not occur as long as Chainlink is functioning correctly. If this is not explicitly caught the code will underflow and give erroneous results. 4. We believe this should not be a big deal and stale data could be considered as the best estimate of the price. Perhaps the minimum could be doubled to be safe if either of the quotes is older than 24 hours. Or increased with 10% every hour of staleness. It is up to the client to either be satisfied with doing nothing or explicitly handling this scenario.
Resolution	RESOLVED The client has implemented the necessary safeguards except for the staleness check, as they would like to use the most recent price still in this scenario.

Issue #06	Several variables are private
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts: - `ethUsdPriceFeed` - `linkUsdPriceFeed` - `fee` - `keyhash` - `startBlock` - `requestToRandom` - `AddressToPendingTrade` - `smallTrades`
Recommendation	Consider marking the above variables as `public`. `smallTrades` should also expose a `smallTradesLength` function.
Resolution	RESOLVED

Issue #07	Division before multiplication is done which could cause excessive rounding within `getMinimumTradeSize`
Severity	INFORMATIONAL
Description	Within the `getMinimumTradeSize` function, the arithmetic first divides and then multiplies which causes the math to round excessively.
Recommendation	Consider always executing multiplication before division.
Resolution	RESOLVED

Issue #08	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographical mistakes which we've enumerated below in a single issue in an effort to keep the report size reasonable.
	
	Line 57
	`//address _factory,`
	
	This line can be removed
	
	Line 67
	`ethUsdPriceFeed = AggregatorV3Interface(_ETHpriceFeedAddress);`
	
	Parameters like `_ETHpriceFeedAddress` can be directly provided as `AggregatorV3Interface` to avoid casting them explicitly.
	
	Line 130
	`* Initiates a trade with weth to the DojoHouse`
	
	Throughout the contracts, usage of WETH is specified. The contract in fact uses real ETH (the actual gas coin), not the wrapped variant.
	
	Line 233
	`require(requestToRandom[requestID] != 0);`
	
	The requirement lacks an error message which might be confusing to third parties and users.
	
	Line 323
	`uint8 number /// @title A title that should describe the contract/ interface`
	
	The comment makes very little sense and is presumably an accidental copy.
	
	Line 193
	`if (AddressToPendingTrade[_address]._requestID != 0) {`
	
	This whole codeblock can be simplified to return `AddressToPendingTrade[_address]._requestID != 0`. A similar simplification can be done for.

	Line 292 `uint256 resetTime = block.number + 0;` Currently, the timelock is set to zero seconds. The developer cannot forget to adjust this.
Recommendation	Consider fixing the above typographical errors.
Resolution	RESOLVED

Issue #09	**Trade size check does not allow trades of the minimum or maximum trade size**
Severity	INFORMATIONAL

Location	Line 139-143
	`require(`
	`  msg.value > getMinimumTradeSize() &&`
	`  msg.value < getMaximumTradeSize(),`
	`  "Invalid trade amount!"`
	`);`

Description	Presently, due to the smaller and greater than check within the `initiateTrade` function, the minimum and maximum are not allowed. This goes contrary to the rest of the documentation where it is said that trades should be possible at these sizes.
Recommendation	Consider using greater than or equal and smaller than or equal checks.
Resolution	RESOLVED

Issue #10	`ethUsdPriceFeed` and `linkUsdPriceFeed` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	RESOLVED

Issue #11	Several functions can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword: - `initiateSetHouse` - `setHouse` - `setKeyhash` - `setFee` - `setRewardsRate` Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as external.
Resolution	RESOLVED

Issue #12	Lack of events for several functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications: - `initiateTrade` - `initiateSmallTrade` - `initiateSetHouse` - `setHouse` - `setKeyhash` - `setFee` - `setRewardsRate` In addition, `fulfillRandomness` emits the wrong event.
Recommendation	Add events for the above functions and rename the `fulfillRandomness` event.
Resolution	RESOLVED

Issue #13	Jackpot mechanism does not adhere to VRF requirements
Severity	INFORMATIONAL
Description	The jackpot mechanism theoretically does not adhere to the Chainlink VRF requirements as the outcome is not solely determined by the 'random' value but also by the order of execution. Theoretically speaking, this allows an extremely privileged exploiter to let the jackpot accumulate until they see that a winner is about to win, then drop their transactions by for example increasing the transaction fee by spamming and finally adding a large entry to claim the jackpot themselves. As this requires extreme privileges and as the profitability of this exploit will likely be disappointing, this issue will be marked as resolved on the note that the impact is too minimal. However, it is included as a reminder to the team that ideally a VRF outcome must always solely be based on parameters known at the time of the randomness request and the random value only.
Recommendation	As the impact of this issue is minimal, no modifications are required. We hope that the client takes inspiration from this issue in their subsequent contracts.
Resolution	RESOLVED The client understands this weakness and will consider it in subsequent contracts.

2.2 DojoHouse

DojoHouse is a simple staking contract which allows the locking of BNB in the contract and mints DLP tokens to the staker.

Fees from throughout the system can be sent as BNB tokens to the DojoHouse which will be compounded into users' stakes. Users can therefore simply stake BNB in DojoHouse to earn more BNB from the system fees.

Additionally, the contract has two functions which can be called by the token contract only: `spinTrade` and `allOrNothingTrade`. These functions are gambling features for the users which take a 2% tax from each call and distribute it to the treasure and bar contract.

On a technical note, this contract presently uses `.transfer` to transfer out the gas token which is considered sub-ideal due to the limited gas limit on this operation. However, as the code location is not written in checks-effects-interactions this was not raised as an issue as it would increase the theoretical reentrancy risk (even though `claimTrade` has a non-standard reentrancy guard due to the `AddressToPendingTrade` lock).

2.2.1 Issues & Recommendations

Issue #14	Balance will almost never be zero because it includes `msg.value`
Severity	MEDIUM SEVERITY
Location	Line 37 `if (totalSupply() == 0 || address(this).balance == 0) {`
Description	The balance check within enter is wrongly defined as even if the contract is empty, the balance will be equal to the provided value of the function call.
Recommendation	Consider checking that the balance is equal to `msg.value`.
Resolution	RESOLVED

Issue #15	Inconsistent usage of uint256
Severity	INFORMATIONAL
Location	Line 77 `uint8 _belt,`
Description	Within the contract, both the variables `uint256` and `uint8` are used. However, it is recommended to remain consistent and only use `uint256`. Being consistent shows to third-party validators that the code has been carefully thought through.
Recommendation	Consider using `uint256` throughout the contract.
Resolution	RESOLVED

Issue #16	`enter`, `leave` and `getValue` can be made `external`
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as external.
Resolution	RESOLVED

Issue #17	Contracts cannot execute the `enter` function but they can still execute `transfer` and `transferFrom`
Severity	INFORMATIONAL
Description	The contract prevents other smart contracts from interacting with it by disallowing them to call enter, however, these contracts can still be transferred DLP. This might not be desirable.
Recommendation	Consider whether this is intentional. If not, consider adding safety measures to `transfer` and `transferFrom`. It should be noted that the client should be careful to still allow desired contracts to transfer DLP.
Resolution	RESOLVED The client has indicated this behavior is desirable as they simply want to prevent contracts from entering and leaving in the same transaction.

Issue #18	`treasury` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	RESOLVED

Issue #19	`treasury` and `bar` are private
Severity	INFORMATIONAL
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the above variables as public.
Resolution	RESOLVED

Issue #20	`getValue` is imprecise and can revert because of division by zero
Severity	INFORMATIONAL
Location	Line 205 `return address(this).balance / totalSupply();`
Description	The `getValue` zero lacks a safeguard in case `totalSupply` is zero and is extremely imprecise due to `address(this).balance` and `totalSupply` being similar magnitude.
Recommendation	Consider adding an `if` statement for `totalSupply` being zero and consider adding a `1e18` precision multiplier.
Resolution	RESOLVED

Issue #21	Typographical errors
Severity	INFORMATIONAL

Description	Line 30
	`* - accure gains/losses from hosue edge over time`
	This should be house.
	
	Line 58
	`//calcuate how much WETH is to be withdrawn`
	WETH is not used.
	
	Line 155
	`* If they win, depending on type, user will get back either 2x,3x,5.05x`
	This last multiplier should be 5x.
	
	Line 156
	`* House edge is always 4-4.05% with a portion going as follows:`
	The house edge is between 4 and 5%.
	
	Line 202
	`* Get the DLP value of Dojo:xDojo`
	This contract does not use Dojo or xDojo.

Recommendation	Consider fixing the above errors.
Resolution	PARTIALLY RESOLVED Some of the errors above have been fixed.

2.3 DojoBar

DojoBar is a simple staking contract which allows the locking of DOJO tokens in the contract and mints xDOJO tokens to the staker.

Fees from the protocol can be sent as Dojo tokens to the DojoBar which will be compounded into users' stakes. Users can therefore simply stake Dojo in the DojoBar to earn more Dojo from the protocol fees.

It should be noted that during the first deposits and withdrawals, the share ratio could become very imprecise. We recommend that the client carefully does the first deposit and confirms that the share ratio (DOJO/xDOJO) is within a reasonable range.

2.3.1 Issues & Recommendations

Issue #22	`collectFees` is vulnerable to sandwich attacks
Severity	LOW SEVERITY
Description	`collectFees` is vulnerable to sandwich attacks where an exploiter might buy Dojo, call `collectFees` which further increases the price, and then sell their Dojo again for a profit.
Recommendation	Consider whether the sandwich attack is relevant to the usage of DojoBar. If so, consider making the function privileged.
Resolution	RESOLVED The client has indicated that they will keep fee collections sufficiently small to avoid significant arbitrage.

Issue #23	Constructor variables can be made IERC20 and IUniswapV2Router02
Severity	INFORMATIONAL
Location	Lines 17-20

```
constructor(address _dojo, address _router) public {
   dojo = IERC20(_dojo);
   router = IUniswapV2Router02(_router);
}
```

Description	Currently, the variables `dojo` and `router` do not have a specific type other than `address`. This requires them to be cast to a contract explicitly. Since we already know their type (`IERC20` and `IUniswapv2Router02`) we can simply define them like that to make the code more readable.
Recommendation	Consider defining Dojo as `IERC20` and router as `IUniswapV2Router02`.
Resolution	RESOLVED

Issue #24	`router` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	RESOLVED

Issue #25	Gas optimization: Swap deadline does not need to be higher than `block.timestamp`
Severity	INFORMATIONAL

Location	Line 74~
	`function collectFees() public {`
	`  //use uniswapv2`
	`  uint256 amountOut = 0;`
	`  uint256 deadline = block.timestamp + 30;`
	`  address[] memory path = new address[](2);`
	`  path[0] = router.WETH();`
	`  path[1] = address(dojo);`
	`  router.swapExactETHForTokens{value: address(this).balance}(`
	`  0,`
	`  path,`
	`  address(this),`
	`  deadline`
	`  );`
	`}`

Description	Since the contract directly executes the function, there is no change in `block.timestamp` because the block does not change.
Recommendation	Consider removing the unnecessary `+30`.
Resolution	RESOLVED The +30 has been removed as recommended.

Issue #26	Several functions can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword: - `enter` - `leave` - `getRatio` - `collectFees` Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above functions as external.
Resolution	RESOLVED

Issue #27	Contracts cannot execute the `enter` function but they can still execute `transfer` and `transferFrom`
Severity	INFORMATIONAL
Description	The contract prevents other smart contracts from interacting with it by disallowing them to call enter, however, these contracts can still be transferred xDojo. This might not be desirable.
Recommendation	Consider whether this is intentional. If not, consider adding safety measures to `transfer` and `transferFrom`. It should be noted that the client should be careful to still allow desired contracts to transfer `xDojo`.
Resolution	RESOLVED The client has indicated that this is intentional as they simply want to avoid contracts from entering and leaving in the same transaction.

Issue #28	Unnecessary bracket use
Severity	INFORMATIONAL

Location	Line 53~
	`function leave(uint256 share) public returns (uint256 what) {`
	`  // Gets the amount of xDojo in existence`
	`  uint256 totalShares = totalSupply();`
	`  // Calculates the amount of Dojo the xDojo is worth`
	`  uint256 what = (share * (dojo.balanceOf(address(this)))) / totalShares;`
	`  _burn(msg.sender, share);`
	`  dojo.transfer(msg.sender, what);`
	`}`

Description	Especially for third-party auditors the use of excessive brackets can be confusing and also unnecessarily increases the contract length.
Recommendation	Consider removing the unnecessary brackets.
Resolution	RESOLVED

Issue #29	Unused variable: `amountOut`
Severity	INFORMATIONAL
Location	Line 73
Description	Variables defined in a contract but not used within said contract could confuse third-party auditors. They furthermore increase the contract length and bytecode size for no reason.
Recommendation	Consider removing the variable to keep the contract short and simple.
Resolution	RESOLVED

Issue #30	**`getRatio` is imprecise**
Severity	INFORMATIONAL
Description	The `getRatio` function simply divides the Dojo balances by the total supply of xDojo. This could be very imprecise and could leave the variable unchanged for as much as 99% increments.
Recommendation	Consider adding a precision multiplier (eg. `1e18`) to the `getRatio` returned variable.
Resolution	RESOLVED The precision has been increased to `1e18` as recommended.

Issue #31	**Typographic errors: `xDoji`**
Severity	INFORMATIONAL
Location	Line 11
Description	On line 11, `xDojo` is mis-spelled as `xDoji`.
Recommendation	Consider changing `xDoji` to `xDojo`.
Resolution	RESOLVED

2.4 DojoRouter

DojoRouter is a modification of the standard Uniswap router contract. It is heavily simplified but adds in logic to gamble with the results of your swap. It allows you to gamble with the result of your swap by gambling it in the DojoHouse. This not only marks you as eligible for the DegenDojo jackpot, but allows you to potentially receive your resulting tokens multiple times if you win. The flip side is that you also could lose everything if you lose.

It presently does not support fee-on-transfer tokens.

2.4.1 Issues & Recommendations

Issue #32	`dojo`, `AddressToETHTrade` and `WETH` are `private`
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the above variables as public.
Resolution	PARTIALLY RESOLVED `AddressToETHTrade` has been removed, `WETH` has been made public but `dojo` is still marked as private.

Issue #33	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains the following typographical errors:
	
	Line 16
	`event ClaimToeknTrade(uint256 tokenPayout, uint256 winnings);`
	
	Consider writing this as `Token`.
	
	Line 53
	`* NOTE: minOut input if for 1 BNB. Trade size is unknown, so we do slippage check on a 1 BNB trade`
	
	This comment is incorrect. The trade size is known at the time of this function call as Chainlink must have already uploaded the randomness on-chain. It is therefore possible for the frontend to specify the correct `minOut`. This is not raised as an explicit issue as we find this solution good enough as well since it is a bit simpler to reason about.

Recommendation	Consider fixing the typographical errors.
Resolution	PARTIALLY RESOLVED The client has fixed the event typographical error but did not adjust the logic or wording on line 53.


PALADIN
BLOCKCHAIN SECURITY