Vegasino

Smart Contract Audit Report

SYSFIXED

11 Mar 2022

TABLE OF CONTENTS

Audited Details

- Audited Project
- Blockchain
- Addresses
- Project Website
- Codebase

Summary

- Contract Summary
- Audit Findings Summary
- Vulnerabilities Summary

Conclusion

Audit Results

Smart Contract Analysis

- Detected Vulnerabilities

Disclaimer

About Us

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
Vegasino	VEGAS	Binance Smart Chain

Addresses

Contract address	0xe6884e29ffe5c6f68f4958cf201b0e308f982ac9
Contract deployer address	0xf1e95C214D01419736f1FdF44262DfDE4B296CFD

Project Website

https://vegasino.io/

Codebase

https://bscscan.com/address/0xe6884e29ffe5c6f68f4958cf201b0e308f982ac9#code

SUMMARY

With over a hundred games available on the Vegasino platform, you can bet on finding something tailored to your tastes and preferences. Our games are carefully curated and tested for fairness, so you get all the fun and excitement of being at a world-class casino. Our $VGS token powers the Vegasino platform developed to exploit the latest defi innovations. The token contract has been fully audited by CertiK, a leading security-focused ranking platform to analyze and monitor blockchain protocols and DeFi projects. Our previous Nevada contract has been ranked in the top 1% of all audits conducted by CertiK. The InterFI Network has also performed an additional audit for extra peace of mind. The Vegasino liquidity aligns with the industry safety standards and best practices. The core team consists of experts in their fields with proven field experience. The team has also passed an external KYC audit to add a layer of safety to the entire project. You can invest in Vegasino, knowing the platform is backed by a capable team and industry-leading security measures. When using our platform, you can focus on having fun and playing the game as it should be. We take pride in our goal and capability to create a secure and fair casino platform for everyone on the Binance Smart Chain. Driven by a constant need to innovate, you will not find a better home to play games in.

Contract Summary

Documentation Quality

Vegasino provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

Code Quality

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by Vegasino with the discovery of several low issues.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-100 SWC-108 | Explicitly define visibility for all state variables on lines 594 and 602.
- SWC-101 | It is recommended to use vetted safe math libraries for arithmetic operations consistently on lines 376, 399, 432, 434, 455, 456, 481, 483, 532, 616, 616, 645, 648, 649, 649, 676, 676 and 679.
- SWC-120 | It is recommended to use external sources of randomness via oracles on lines 659 and 660.

CONCLUSION

We have audited the Vegasino project released on March 2022 to discover issues and identify potential security vulnerabilities in Vegasino Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides satisfactory results with low-risk issues.

The Vegasino smart contract code issues do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues found are arithmetic operation issues, a floating pragma is set, state variable visibility is not set, and the potential use of "block.number" as a source of randomness. State variable visibility is not set. It is best practice to set the visibility of state variables explicitly. The default visibility for "taxRecipient" is internal. Other possible visibility settings are public and private. Potential use of "block.number" as a source of randomness, the environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number, and timestamp are predictable and can be manipulated by a malicious miner. Also, keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness; be aware that using these variables introduces a certain level of trust in miners.

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	ISSUE FOUND
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	ISSUE FOUND
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	PASS
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	PASS
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
Unprotected Ether Withdrawal	SWC-105	Due to missing or insufficient access controls, malicious parties can withdraw from the contract.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	PASS
Uninitialized Storage Pointer	SWC-109	Uninitialized local storage variables can point to unexpected storage locations in the contract.	PASS
Assert Violation	SWC-110 SWC-123	Properly functioning code should never reach a failing assert statement.	PASS
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	PASS
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	PASS
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	PASS
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	PASS
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Incorrect Constructor Name	SWC-118	Constructors are special functions that are called only once during the contract creation.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	ISSUE FOUND
Write to Arbitrary Storage Location	SWC-124	The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations.	PASS
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS
Insufficient Gas Griefing	SWC-126	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract.	PASS
Arbitrary Jump Function	SWC-127	As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value.	PASS

Typographical Error	SWC-129	A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable.	PASS
Override control character	SWC-130	Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.	PASS
Unused variables	SWC-131 SWC-135	Unused variables are allowed in Solidity and they do not pose a direct security issue.	PASS
Unexpected Ether balance	SWC-132	Contracts can behave erroneously when they strictly assume a specific Ether balance.	PASS
Hash Collisions Variable	SWC-133	Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision.	PASS
Hardcoded gas amount	SWC-134	The transfer() and send() functions forward a fixed amount of 2300 gas.	PASS
Unencrypted Private Data	SWC-136	It is a common misconception that private type variables cannot be read.	PASS

SMART CONTRACT ANALYSIS

Started	Thursday Mar 10 2022 14:53:21 GMT+0000 (Coordinated Universal Time)
Finished	Friday Mar 11 2022 12:38:09 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	Vegasino.sol

Detected Issues

ID	Title	Severity	Status
SWC-101	ARITHMETIC OPERATION "+" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "+=" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "+=" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "+=" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-=" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "*" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "**" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "+" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged

SWC-101	ARITHMETIC OPERATION "*" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "*" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged

SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED

LINE 376

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
375   address owner = _msgSender();
376   _approve(owner, spender, _allowances[owner][spender] + addedValue);
377   return true;
378   }
379
380
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 399

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
398   unchecked {
399   _approve(owner, spender, currentAllowance - subtractedValue);
400   }
401
402   return true;
403
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 432

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
431   unchecked {
432   _balances[from] = fromBalance - amount;
433   }
434   _balances[to] += amount;
435
436
```

SWC-101 | ARITHMETIC OPERATION "+=" DISCOVERED

LINE 434

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
433   }
434   _balances[to] += amount;
435
436   emit Transfer(from, to, amount);
437
438
```

SWC-101 | ARITHMETIC OPERATION "+=" DISCOVERED

LINE 455

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
454
455   _totalSupply += amount;
456   _balances[account] += amount;
457   emit Transfer(address(0), account, amount);
458
459
```

SWC-101 | ARITHMETIC OPERATION "+=" DISCOVERED

LINE 456

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
455   _totalSupply += amount;
456   _balances[account] += amount;
457   emit Transfer(address(0), account, amount);
458
459   _afterTokenTransfer(address(0), account, amount);
460
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 481

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
480   unchecked {
481   _balances[account] = accountBalance - amount;
482   }
483   _totalSupply -= amount;
484
485
```

SWC-101 | ARITHMETIC OPERATION "-=" DISCOVERED

LINE 483

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
482   }
483   _totalSupply -= amount;
484
485   emit Transfer(account, address(0), amount);
486
487
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 532

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
531   unchecked {
532   _approve(owner, spender, currentAllowance - amount);
533   }
534   }
535   }
536
```

SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED

LINE 616

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
615   isWhitelisted[recipient] = true;
616   _mint(recipient, 5_000_000_000 * 10**18);
617   }
618
619   // Override
620
```

SWC-101 | ARITHMETIC OPERATION "**" DISCOVERED

LINE 616

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
615   isWhitelisted[recipient] = true;
616   _mint(recipient, 5_000_000_000 * 10**18);
617   }
618
619   // Override
620
```

SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED

LINE 645

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
644   function getDynamicSellTax() public view returns (uint256) {
645   uint256 endingTime = launchTime + 10 days;
646
647   if (endingTime > block.timestamp) {
648   uint256 remainingTime = endingTime - block.timestamp;
649
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 648

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
647   if (endingTime > block.timestamp) {
648   uint256 remainingTime = endingTime - block.timestamp;
649   return 3000 * remainingTime / 10 days;
650   } else {
651   return 0;
652
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 649

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
648   uint256 remainingTime = endingTime - block.timestamp;
649   return 3000 * remainingTime / 10 days;
650   } else {
651   return 0;
652   }
653
```

SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED

LINE 649

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
648   uint256 remainingTime = endingTime - block.timestamp;
649   return 3000 * remainingTime / 10 days;
650   } else {
651   return 0;
652   }
653
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 676

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
675
676   uint256 taxAmount = amount * _getTotalTax(recipient) / TAX_DENOMINATOR;
677   if (taxAmount > 0) { super._transfer(sender, taxRecipient, taxAmount); }
678
679   return amount - taxAmount;
680
```

SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED

LINE 676

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
675
676   uint256 taxAmount = amount * _getTotalTax(recipient) / TAX_DENOMINATOR;
677   if (taxAmount > 0) { super._transfer(sender, taxRecipient, taxAmount); }
678
679   return amount - taxAmount;
680
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 679

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- Vegasino.sol

Locations

```
678
679   return amount - taxAmount;
680   }
681
682   function _getTotalTax(address recipient) private view returns (uint256) {
683
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 594

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "taxRecipient" is internal. Other possible visibility settings are public and private.

Source File

- Vegasino.sol

Locations

```
593   uint256 constant TAX_DENOMINATOR = 10000;
594   address immutable taxRecipient;
595   uint256 public launchTime;
596   bool public tradingEnabled;
597   bool public frontRunProtectionEnabled = true;
598
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 602

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "previousBuyBlock" is internal. Other possible visibility settings are public and private.

Source File

- Vegasino.sol

Locations

```
601   mapping (address => bool) public isBot;
602   mapping (address => uint256) previousBuyBlock;
603
604   event EnableTrading();
605   event DisableFrontRunProtection();
606
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 659

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- Vegasino.sol

Locations

```
658   if (isMarketMaker[sender]) {
659   previousBuyBlock[recipient] = block.number;
660   } else if (isMarketMaker[recipient] && previousBuyBlock[sender] == block.number) {
661   isBot[sender] = true;
662   }
663
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 660

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- Vegasino.sol

Locations

```
659   previousBuyBlock[recipient] = block.number;
660   } else if (isMarketMaker[recipient] && previousBuyBlock[sender] == block.number) {
661   isBot[sender] = true;
662   }
663
664
```

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to, or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Sysfixed to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.