Cyberscope

Audit Report
Sakai Vault

December 2023

Analysis

● Critical   ● Medium   ● Minor / Informative   ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical   ● Medium   ● Minor / Informative

Severity	Code	Description	Status
●	PVC	Price Volatility Concern	Unresolved
●	MU	Modifiers Usage	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	SAKAI
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0x43b35e89d15b91162dea1c511 33c4c93bdd1c4af
Address	0x43b35e89d15b91162dea1c51133c4c93bdd1c4af
Network	BSC
Symbol	SAKAI
Decimals	18
Total Supply	8,000,000

Audit Updates

Initial Audit	18 Dec 2023

Source Files

Filename	SHA256
SAKAI.sol	653805318186ed85f7c2dcb550e71f656e39f42cab2fdafd2256c2171f1d dde4

Findings Breakdown
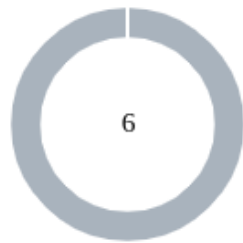
● Critical	0
● Medium	0
● Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	6	0	0	0

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	SAKAI.sol#L1657
Status	Unresolved

Description

The contract can continuously accumulate tokens from the taxes to swap them for ETH if the `isSwapBackEnabled` variable is set to false. If the contract owner invokes the `manualSwapBack` function then the contract will trigger the swap functionality. If the contract holds a large number of tokens, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
    function manualSwapBack() external {
        uint256 contractTokenBalance = balanceOf(address(this));

        require(contractTokenBalance > 0, "Cant Swap Back 0 Token!");

        swapBack();
    }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	SAKAI.sol#L1470,1484,1502
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
    require(
        _treasuryWallet != treasuryWallet,
        "Treasury wallet is already that address"
    );
    require(
        _treasuryWallet != address(0),
        "Treasury wallet cannot be the zero address"
    );
...
    require(
        _stakingContract != stakingContract,
        "Staking contract is already that address"
    );
    require(
        _stakingContract != address(0),
        "Staking contract cannot be the zero address"
    );
...
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these

checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	SAKAI.sol#L1410,1411,1419
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
swapTax
denominator
uniswapV2Pair
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SAKAI.sol#L361,1170,1391,1469,1483,1497
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function WETH() external pure returns (address);
mapping(address => bool) public _isAutomatedMarketMakerPairs
address _treasuryWallet
address _stakingContract
address _gasWallet
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	SAKAI.sol#L58,89,127,175,194,215,233,278,575,594,612,628,651,1028
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```solidity
function sendValue(address payable recipient, uint256 amount) internal
{
        require(
            address(this).balance >= amount,
            "Address: insufficient balance"
        );

        (bool success, ) = recipient.call{value: amount}("");
        require(
            success,
            "Address: unable to send value, recipient may have
reverted"
        );
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	SAKAI.sol#L298,1439,1449
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            let returndata_size := mload(returndata)
            revert(add(32, returndata), returndata_size)
        }

assembly {
        id := chainid()
      }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-

Context	Implementation				
	_msgSender	Internal			
	_msgData	Internal			
Ownable	Implementation	Context			
		Public	✓	-	
	owner	Public		-	
	_checkOwner	Internal			
	renounceOwnership	Public	✓	onlyOwner	
	transferOwnership	Public	✓	onlyOwner	
	_transferOwnership	Internal	✓		
IERC20	Interface				
	totalSupply	External		-	
	balanceOf	External		-	
	transfer	External	✓	-	
	allowance	External		-	
	approve	External	✓	-	
	transferFrom	External	✓	-	
SafeERC20	Library				
	safeTransfer	Internal	✓		

	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-

	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-

	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-

|				
	swapExactTokensForTokensSupporting FeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFee OnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFee OnTransferTokens	External	✓	-
SAKAI	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
		External	Payable	-
	getRouterAddress	Public		-
	getUSDTAddress	Public		-
	claimStuckTokens	External	✓	onlyOwner
	setTreasuryWallet	External	✓	onlyOwner
	setStakingContract	External	✓	onlyOwner
	setGasWallet	External	✓	-
	setSwapTokensAtAmount	External	✓	onlyOwner
	toggleSwapBack	External	✓	onlyOwner
	setExcludeFromFees	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	isAutomatedMarketMakerPair	Public		-
	_transfer	Internal	✓	
	swapBack	Internal	✓	inSwap

	manualSwapBack	External	✓	-

Inheritance Graph
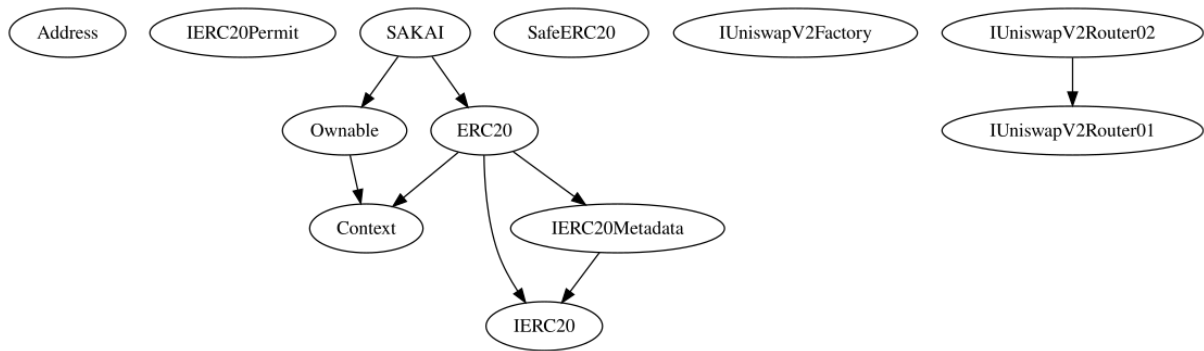
Flow Graph
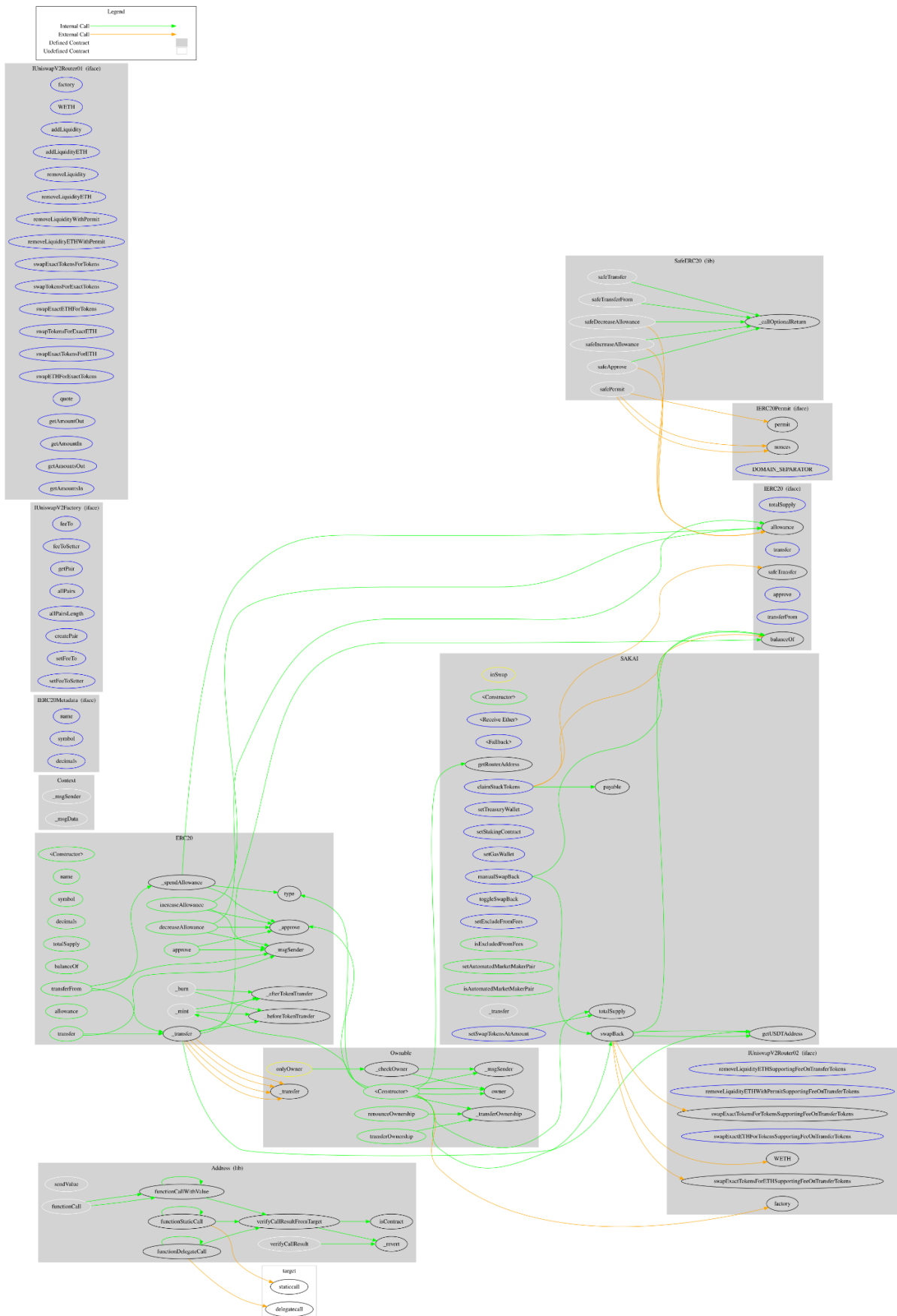
Summary

Sakai Vault contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Sakai Vault is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a 3% fee on buy and sell transactions.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. 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 Cyberscope's prior written consent. 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io