Cyberscope

Audit Report

SafePad

October 2023

Network	ETH
Address	0xd28c8ff18f811e5fcd9b5b07889a343da8fd6502
Audited by	© cyberscope

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
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	SafePad
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	10000 runs
Explorer	https://etherscan.io/address/0xd28c8ff18f811e5fcd9b5b07889a343da8fd6502
Address	0xd28c8ff18f811e5fcd9b5b07889a343da8fd6502
Network	ETH
Symbol	SAFEPAD
Decimals	18
Total Supply	1,000,000,000,000

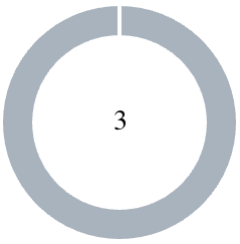
Audit Updates

Initial Audit	21 Oct 2023

Source Files

Filename	SHA256
SafePad.sol	4de0379dcb830d7da046193a4c31b735ad76b6ea8f9663ace955dd6d377a682b

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	3	0	0	0

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SafePad.sol#L192
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);
```

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	SafePad.sol#L10,23,44,80,92,107,119,150,234,246,261,277,299,319,349
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.

```
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	SafePad.sol#L168
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)
            }
```

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

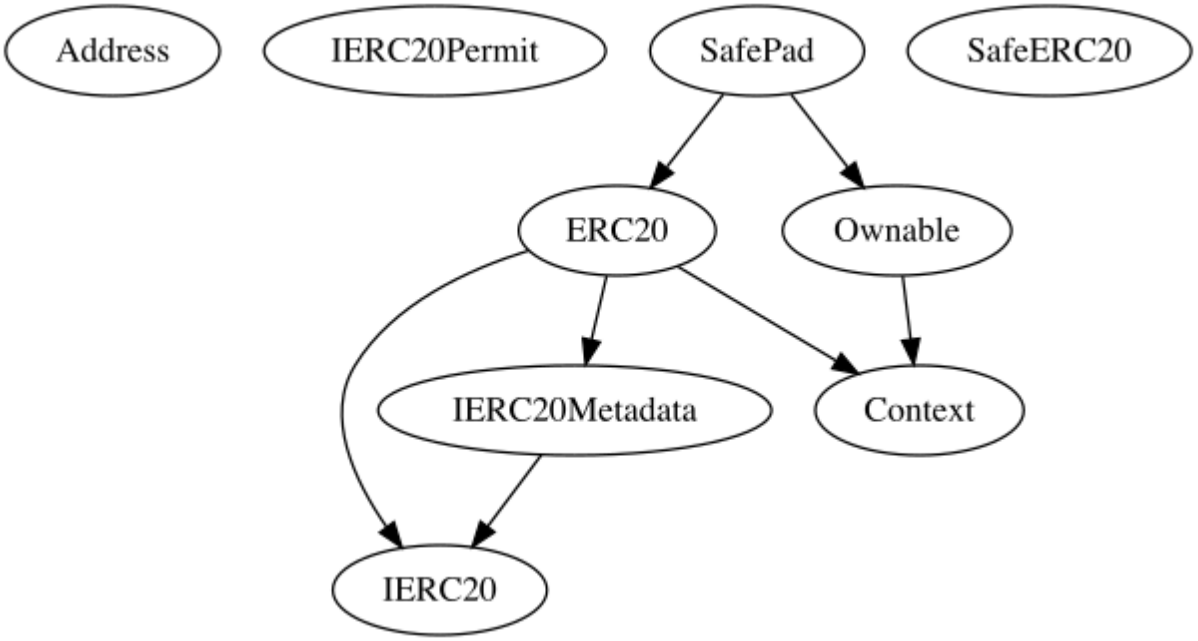
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall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

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	✓	
	forceApprove	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
	_callOptionalReturnBool	Private	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-

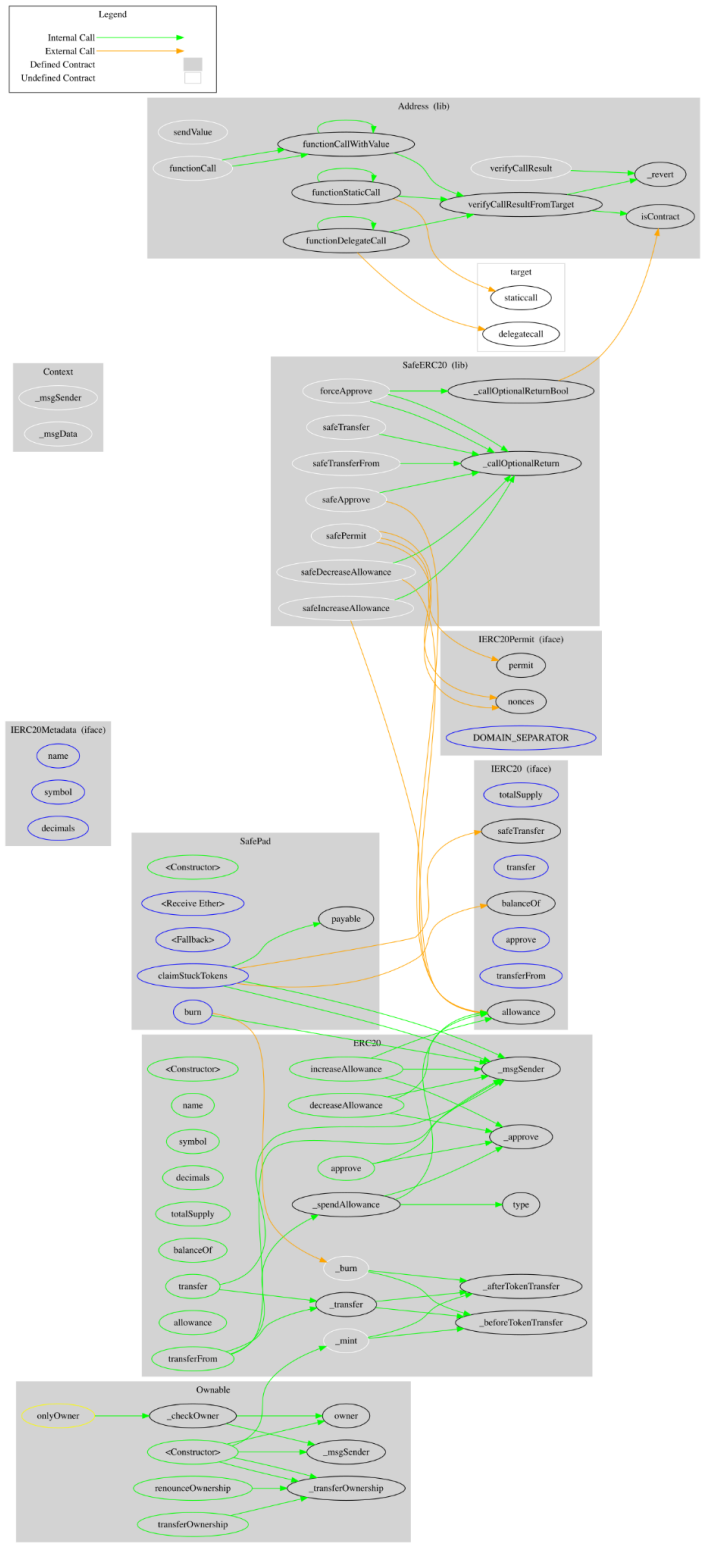
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
SafePad	Implementation	Ownable, ERC20		
		Public	✓	ERC20
		External	Payable	-
		External	Payable	-
	burn	External	✓	-
	claimStuckTokens	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
_revert
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
target
staticcall
delegatecall
Context
_msgSender
_msgData
SafeERC20 (lib)
forceApprove
_callOptionalReturnBool
safeTransfer
safeTransferFrom
_callOptionalReturn
safeApprove
safePermit
safeDecreaseAllowance
safeIncreaseAllowance
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
safeTransfer
transfer
balanceOf
approve
transferFrom
allowance
SafePad
<Constructor>
<Receive Ether>
<Fallback>
payable
claimStuckTokens
burn
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_spendAllowance
_burn
_transfer
_mint
_msgSender
_approve
type
_afterTokenTransfer
_beforeTokenTransfer
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership

Summary

SafePad contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. SafePad 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.

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