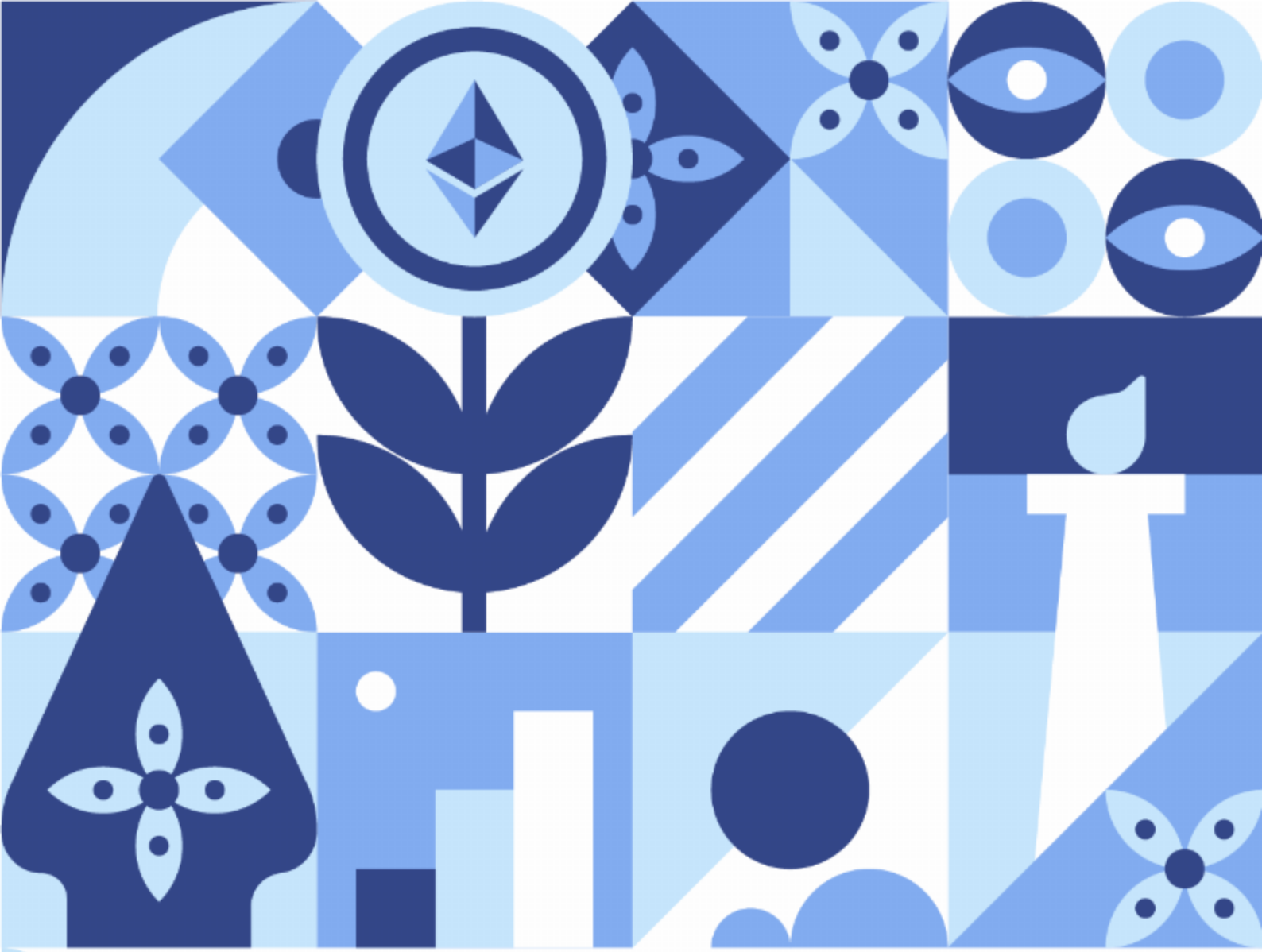

Energy Web Token Bridged

Smart Contract Audit Report

SYSFIXED

07 Apr 2020

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
Energy Web Token Bridged	EWTB	Ethereum

Addresses

Contract address	0x178c820f862B14f316509ec36b13123DA19A6054
Contract deployer address	0x4Fa5fE98fe0556C9AF37542149D59B0bDe52B17B

Project Website

https://energyweb.org/

Codebase

https://etherscan.io/address/0x178c820f862B14f316509ec36b13123DA19A6054#code

SUMMARY

Energy Web Token (EWT) is the operational token behind the Energy Web Chain, a blockchain-based virtual machine designed to support and further application development for the energy sector.

Contract Summary

Documentation Quality

Energy Web Token Bridged provides a very poor documentation with standard of solidity base code.

- The technical description is provided unclear and disorganized.

Code Quality

The Overall quality of the basecode is poor.

- Solidity basecode and rules are unclear and disorganized by Energy Web Token Bridged.

Test Coverage

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

Audit Findings Summary

- SWC-102 | It is recommended to use a recent version of the Solidity compiler on lines 500, 514, 525, 535 and 593.
- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 7, 24, 79, 130, 164, 189, 315, 382, 444 and 468.
- SWC-104 | It is recommended to use handle at low-level call methods on lines 659 and 558.
- SWC-107 | It is recommended to use a reentrancy lock, reentrancy weaknesses detected on lines 653.
- SWC-110 SWC-123 | It is recommended to use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM on lines 72, 659 and 574.
- SWC-113 SWC-128 | It is recommended to implement the contract logic to handle failed calls and block gas limit on lines 559.

CONCLUSION

We have audited the Energy Web Token Bridged project released in April 2020 to find issues and identify potential security vulnerabilities in the Energy Web Token Bridged project. This process is used to find technical issues and security loopholes that may be found in smart contracts.

The security audit report gave unsatisfactory results with the discovery of high-risk issues and several other low-risk issues.

Writing a contract that does not follow the Solidity style guide can pose a significant risk. The medium-risk problem we found is an unchecked return value from external calls, and multiple calls are executed in the same transaction. External calls return a boolean value. If the callee halts with an exception, 'false' is returned and execution continues in the caller. We recommend to the caller should check whether an exception happened and react accordingly to avoid unexpected behavior. For example, it is often desirable to wrap external calls in require() so the transaction is reverted if the call fails also this call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase). Whereas Low-risk Issues we found are some outdated compiler version is used, floating pragma is set, an assertion violation was triggered, requirement violation, and read of persistent state following the external call.

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.	PASS
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	PASS
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	ISSUE FOUND
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	ISSUE FOUND
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	ISSUE FOUND
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.	ISSUE FOUND
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.	ISSUE FOUND
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.	ISSUE FOUND
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.	PASS
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	Monday Apr 06 2020 08:04:51 GMT+0000 (Coordinated Universal Time)
Finished	Tuesday Apr 07 2020 12:00:21 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	ERC677BridgeToken.sol

Detected Issues

ID	Title	Severity	Status
SWC-104	UNCHECKED RETURN VALUE FROM EXTERNAL CALL.	medium	acknowledged
SWC-104	UNCHECKED RETURN VALUE FROM EXTERNAL CALL.	medium	acknowledged
SWC-113	MULTIPLE CALLS ARE EXECUTED IN THE SAME TRANSACTION.	medium	acknowledged
SWC-102	AN OUTDATED COMPILER VERSION IS USED.	low	acknowledged
SWC-102	AN OUTDATED COMPILER VERSION IS USED.	low	acknowledged
SWC-102	AN OUTDATED COMPILER VERSION IS USED.	low	acknowledged
SWC-102	AN OUTDATED COMPILER VERSION IS USED.	low	acknowledged
SWC-102	AN OUTDATED COMPILER VERSION IS USED.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged

SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-107	READ OF PERSISTENT STATE FOLLOWING EXTERNAL CALL.	low	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged

SWC-104 | UNCHECKED RETURN VALUE FROM EXTERNAL CALL.

LINE 659

medium SEVERITY

External calls return a boolean value. If the callee halts with an exception, 'false' is returned and execution continues in the caller. The caller should check whether an exception happened and react accordingly to avoid unexpected behavior. For example it is often desirable to wrap external calls in require() so the transaction is reverted if the call fails.

Source File

- ERC677BridgeToken.sol

Locations

```
658   function contractFallback(address _from, address _to, uint256 _value, bytes _data)
private returns (bool) {
659   return _to.call(abi.encodeWithSignature("onTokenTransfer(address,uint256,bytes)",
_from, _value, _data));
660   }
661
662   function finishMinting() public returns (bool) {
663
```

SWC-104 | UNCHECKED RETURN VALUE FROM EXTERNAL CALL.

LINE 558

medium SEVERITY

External calls return a boolean value. If the callee halts with an exception, 'false' is returned and execution continues in the caller. The caller should check whether an exception happened and react accordingly to avoid unexpected behavior. For example it is often desirable to wrap external calls in require() so the transaction is reverted if the call fails.

Source File

- ERC677BridgeToken.sol

Locations

```
557   uint256 value = address(this).balance;
558   if (!_to.send(value)) {
559   (new Sacrifice).value(value)(_to);
560   }
561   }
562
```

SWC-113 | MULTIPLE CALLS ARE EXECUTED IN THE SAME TRANSACTION.

LINE 559

medium SEVERITY

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source File

- ERC677BridgeToken.sol

Locations

```
558   if (!_to.send(value)) {
559   (new Sacrifice).value(value)(_to);
560   }
561   }
562
563
```

SWC-102 | AN OUTDATED COMPILER VERSION IS USED.

LINE 500

low SEVERITY

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source File

- ERC677BridgeToken.sol

Locations

```
499
500   pragma solidity 0.4.24;
501
502
503   contract ERC677 is ERC20 {
504
```

SWC-102 | AN OUTDATED COMPILER VERSION IS USED.

LINE 514

low SEVERITY

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source File

- ERC677BridgeToken.sol

Locations

```
513
514   pragma solidity 0.4.24;
515
516
517   contract IBurnableMintableERC677Token is ERC677 {
518
```

SWC-102 | AN OUTDATED COMPILER VERSION IS USED.

LINE 525

low SEVERITY

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source File

- ERC677BridgeToken.sol

Locations

```
524
525   pragma solidity 0.4.24;
526
527   contract Sacrifice {
528   constructor(address _recipient) public payable {
529
```

SWC-102 | AN OUTDATED COMPILER VERSION IS USED.

LINE 535

low SEVERITY

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source File

- ERC677BridgeToken.sol

Locations

```
534
535   pragma solidity 0.4.24;
536
537
538
539
```

SWC-102 | AN OUTDATED COMPILER VERSION IS USED.

LINE 593

low SEVERITY

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source File

- ERC677BridgeToken.sol

Locations

```
592
593   pragma solidity 0.4.24;
594
595
596
597
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 7

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
6
7   pragma solidity ^0.4.24;
8
9
10   /**
11
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 24

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
23
24   pragma solidity ^0.4.24;
25
26
27   /**
28
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 79

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
78
79   pragma solidity ^0.4.24;
80
81
82
83
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 130

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
129
130   pragma solidity ^0.4.24;
131
132
133
134
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 164

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
163
164   pragma solidity ^0.4.24;
165
166
167
168
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 189

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
188
189   pragma solidity ^0.4.24;
190
191
192
193
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 315

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
314
315   pragma solidity ^0.4.24;
316
317
318   /**
319
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 382

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
381
382   pragma solidity ^0.4.24;
383
384
385
386
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 444

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
443
444   pragma solidity ^0.4.24;
445
446
447
448
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 468

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- ERC677BridgeToken.sol

Locations

```
467
468   pragma solidity ^0.4.24;
469
470
471   /**
472
```

SWC-107 | READ OF PERSISTENT STATE FOLLOWING EXTERNAL CALL.

LINE 653

low SEVERITY

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source File

- ERC677BridgeToken.sol

Locations

```
652   if (AddressUtils.isContract(_to) && !contractFallback(_from, _to, _value, new
bytes(0))) {
653   require(_to != bridgeContract);
654   emit ContractFallbackCallFailed(_from, _to, _value);
655   }
656   }
657
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 72

low SEVERITY

It is possible to cause an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- ERC677BridgeToken.sol

Locations

```
71   c = _a + _b;
72   assert(c >= _a);
73   return c;
74   }
75   }
76
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 659

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- ERC677BridgeToken.sol

Locations

```
658   function contractFallback(address _from, address _to, uint256 _value, bytes _data)
private returns (bool) {
659   return _to.call(abi.encodeWithSignature("onTokenTransfer(address,uint256,bytes)",
_from, _value, _data));
660   }
661
662   function finishMinting() public returns (bool) {
663
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 574

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- ERC677BridgeToken.sol

Locations

```
573   assembly {
574   let result := call(gas, _token, 0x0, add(callData, 0x20), mload(callData), 0, 32)
575   returnData := mload(0)
576   returnDataResult := mload(0)
577
578
```

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.