ackee blockchain

Axelar

Utils & Squid Router

by Ackee Blockchain

19.10.2022

Contents

1. Document Revisions

0.1	Draft report	12.10.2022
1.0	Final report	19.10.2022

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Woke is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member's Name	Position
Michal Převrátil	Lead Auditor
Jan Kalivoda	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

The first objective of the audit is a part of the Axelar Utils repository, with a custom implementation of a const address contract deployer and helper libraries to convert between string and address and between string and bytes32. The second objective is Squid Router which allows users to send tokens through the Axelar Gateway and optionally execute arbitrary operations (external calls) on both the source and destination chains.

Revision 1

Axelar engaged Ackee Blockchain to perform a security review of the Axelar Utils and Squid Router implementation with a total time donation of 5 engineering days in a period between October 3 and October 7, 2022 with Michal Převrátil as the lead auditor.

The audit was performed on two repositories with the following commits and files.

- Axelar Utils - `726020f`
  - contracts/ConstAddressDeployer.sol
  - contracts/StringAddressUtils.sol
  - contracts/StringBytesUtils.sol
- A private repository - `cdd406a`
  - packages/squidswap-contracts/contracts/RoledPausable.sol
  - packages/squidswap-contracts/contracts/SquidMulticall.sol
  - packages/squidswap-contracts/contracts/SquidRouterProxy.sol
  - packages/squidswap-contracts/contracts/SquidRouter.sol

We began our review by using static analysis tools, namely Woke and Slither.

Outputs from Woke are enclosed in Appendix C. Then we implemented fuzz tests using Woke and Brownie to discover potential vulnerabilities. Sources of fuzz tests are available in Appendix D.

We took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- contracts are not susceptible to re-entrancy attacks,
- users of the contracts cannot lose their funds,
- helper and library functions work for all possible inputs,
- input data are properly validated.

Our review resulted in 20 findings, ranging from Info to High severity.

Ackee Blockchain recommends Axelar and Squid:

- reconsider the current architecture being too generic allowing loss of user funds with improperly crafted input data,
- not to rely only on the off-chain implementation and add data validation to the contracts,
- add NatSpec comments to the code,
- address all other reported issues.

See the full report in chapter Revision 1.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Solution*.

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Impact	Likelihood
H1: `fundAndRunMulticall` is not pausable	High	High	Medium
M1: Missing `Call.callType` validation	Medium	High	Low
M2: Missing `isContract` check in `SquidMulticall`	Medium	High	Low
M3: Memory address overflow in `setCallDataParameter`	Medium	High	Low
M4: Multicall implementation being too generic	Medium	High	Low
M5: Re-entrancy in `SquidRouter`	Medium	Medium	Low

	Severity	Impact	Likelihood
M6: Missing `refundRecipient` validation	Medium	High	Low
M7: Missing `destinationChain` validation	Medium	High	Low
W1: Missing validation of the `0x` prefix in string addresses	Warning	Warning	N/A
W2: Use of `solc` optimizer	Warning	Warning	N/A
W3: Address helper functions not respecting EIP-55	Warning	Warning	N/A
W4: `SquidRouter` pausable can be bypassed	Warning	Warning	N/A
I1: Unnecessary `abi.encodePacked`	Info	Info	N/A
I2: Multiple calls to `pendingPauser`	Info	Info	N/A
I3: Bytes length accessed in a for loop condition	Info	Info	N/A
I4: Inconsistent for loop incrementation	Info	Info	N/A
I5: Address code length can be checked before a call	Info	Info	N/A
I6: For loop variable can be incremented in an unchecked block	Info	Info	N/A
I7: Missing NatSpec documentation	Info	Info	N/A

	Severity	Impact	Likelihood
I8: Inconsistent behavior: Revert vs return default	Info	Info	N/A

Table 2. Table of Findings

5. Report revision 1.0

The first revision of the audit report.

System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

Contracts

Contracts we find important for better understanding are described in the following section.

SquidMulticall

`SquidMulticall` is a contract that allows executing multiple calls to other contracts in a single transaction. In a typical scenario, the external calls perform ERC-20 token purchases, swaps and transfers. There are multiple external call types to support these operations.

SquidRouter

`SquidRouter` optionally performs a user-defined multicall on a source chain (through the SquidMulticall contract), sends token through the Axelar Gateway, and then optionally performs another user-defined multicall on a destination chain. It is also supported to perform a multicall without sending tokens through the Axelar Gateway.

SquidRouterProxy

`SquidRouterProxy` suits as a proxy contract for the SquidRouter contract. This allows the SquidRouter contract to be upgraded with a new implementation while keeping the same address.

Actors

Pauser

A pauser of the SquidRouter contract can pause all public functions of the contract. This is useful for emergency situations, such as a bug in the contract or a security issue. A pauser can be changed using a two-step transfer process.

Owner

An owner of the SquidRouterProxy contract can deploy a new implementation of the SquidRouter contract. This effectively also changes the pauser of the SquidRouter contract to the owner.

User

A user can call the SquidRouter contract to perform an Axelar Gateway bridge operation with optional multicalls on both source and destination chains. A user can also perform a multicall without sending tokens through the Axelar Gateway.

Trust model

Users have to trust the Owner of the SquidRouterProxy contract to not deploy a malicious implementation of the SquidRouter contract. Users also have to trust the tokens they own and the tokens they want to be purchased/swapped in the SquidMulticall contract. Users have to trust any off-chain service that they use to interact with the SquidRouter contract. More specifically, users have to trust the off-chain implementation that generates commands (calls) for the SquidMulticall contract.

H1: `fundAndRunMulticall` is not pausable

High severity issue

Impact:	High	Likelihood:	Medium
Target:	/packages/squidswap-contracts/contracts/SquidRouter.sol	Type:	Logic error

Listing 1. Excerpt from /packages/squidswap-contracts/contracts/SquidRouter.sol#L100-L104[SquidRouter.fundAndRunMulticall]

```
    function fundAndRunMulticall(
        address token,
        uint256 amount,
        ISquidMulticall.Call[] memory calls
    ) public payable {
```

Description

From the documentation provided, all external and public functions should be pausable to be able to stop the contract in case of an emergency. However, the `fundAndRunMulticall` function is not pausable.

Exploit scenario

A serious vulnerability in the `SquidMulticall` is discovered. Before the vulnerability is fixed, anyone can call the `fundAndRunMulticall` function because it is not pausable.

Recommendation

Either add the `whenNotPaused` modifier to the `fundAndRunMulticall` function or clarify in the documentation that this function is not pausable.

Go back to Findings Summary

M1: Missing `Call.callType` validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	/packages/squidswap-contracts/contracts/SquidMulticall.sol	Type:	Data validation

Listing 2. Excerpt from /packages/squidswap-contracts/contracts/SquidMulticall.sol#L21-L31 [SquidMulticall.run]

```
            if (call.callType == CallType.FullTokenBalance) {
                (address token, uint256 amountParameterPosition) = abi.decode(call.payload, (address, uint256));
                uint256 amount = IERC20(token).balanceOf(address(this));
                _setCallDataParameter(call.callData, amountParameterPosition, amount);
            } else if (call.callType == CallType.FullNativeBalance) {
                call.value = address(this).balance;
            } else if (call.callType == CallType.CollectTokenBalance) {
                address token = abi.decode(call.payload, (address));
                _safeTransferFrom(token, msg.sender, IERC20(token).balanceOf(msg.sender));
                continue;
            }
```

Description

The `run` function does not handle the case where `call.callType` is not one of the expected values.

Exploit scenario

A new `callType` is implemented with the documentation being updated before the `SquidRouter` contract is upgraded. As a result, users send transactions

with the new `callType` value, leading to unexpected behavior.

Recommendation

Revert the transaction if `call.callType` is not one of the expected values.

Go back to Findings Summary

M2: Missing `isContract` check in `SquidMulticall`

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	/packages/squidswap-contracts/contracts/SquidMulticall.sol	Type:	Data validation

Listing 3. Excerpt from /packages/squidswap-contracts/contracts/SquidMulticall.sol#L33-L34[SquidMulticall.run]

```
33            (bool success, bytes memory data) = call.target.call{value: call.value}(call.callData);
34            if (!success) revert CallFailed(i, data);
```

Description

The `success` variable will be set to `true` even if `call.target` is not a contract.

Exploit scenario

A user mistypes an address of a decentralized exchange where he wanted to purchase ERC-20 tokens. Ether that would have been used to purchase the tokens is sent to a different account than expected. The transaction does not revert, resulting in a loss of Ether. Before the user manages to call a transaction requesting the Ether back, another user calls the `SquidMulticall` contract and receives the leftover Ether.

Recommendation

Add a boolean flag to the `Call` struct indicating whether the target is a contract. If the flag is set to `true`, check that `call.target` is a contract before performing the external call.

Go back to Findings Summary

M3: Memory address overflow in `_setCallDataParameter`

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	/packages/squidswap-contracts/contracts/SquidMulticall.sol	Type:	Data validation

Listing 4. Excerpt from /packages/squidswap-contracts/contracts/SquidMulticall.sol#L56-L65[SquidMulticall._setCallDataParameter]

```
    function _setCallDataParameter(
        bytes memory callData,
        uint256 parameterPosition,
        uint256 value
    ) private pure {
        assembly {
            // 36 bytes shift because 32 for prefix + 4 for selector
            mstore(add(callData, add(36, mul(parameterPosition, 32))), value)
        }
    }
```

Description

The `_setCallDataParameter` function allows overwriting the `callData` variable at a given position with a 256-bit value. The function does not check if the given position is within the bounds of the `callData` variable. Given the fact that the operation is performed in an inline assembly block, this can lead to a memory address overflow and overwrite arbitrary memory locations.

Among vulnerable objects in memory are:

- selector and prefix parts of the `callData` variable,
- address of the next external call to be performed in the `run` function,
- free memory pointer at memory location `0x40`.

Overwriting the free memory pointer can lead to memory corruption and malformation of any data to be stored in memory.

Exploit scenario

A user encodes `-1` as a value of the `parameterPosition` parameter. Because the data are encoded using the ABI encoding and interpreted as `uint256`, the `mstore` instruction is evaluated as:

```
mstore(add(callData, add(36, mul(2 ** 256 - 1, 32))), value)
```

which is equal to:

```
mstore(add(callData, 4), value)
```

This effectively overwrites both the prefix part (except the first four bytes) and the selector part of the `callData` variable, leading to unexpected behavior.

Recommendation

Add a check that the given position is within the bounds of the `callData` variable and does not overwrite the prefix and selector parts.

Go back to Findings Summary

M4: Multicall implementation being too generic

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	/packages/squidswap-contracts/contracts/SquidMulticall.sol	Type:	Data validation

Listing 5. Excerpt from /packages/squidswap-contracts/contracts/SquidMulticall.sol#L12-L42[SquidMulticall.run]

```
12     function run(Call[] calldata calls) external payable {
13         // Prevents reentrancy
14         if (isRunning) revert AlreadyRunning();
15         isRunning = true;
16
17         uint256 length = calls.length;
18         for (uint256 i = 0; i < length; ) {
19             Call memory call = calls[i];
20
21             if (call.callType == CallType.FullTokenBalance) {
22                 (address token, uint256 amountParameterPosition) =
   abi.decode(call.payload, (address, uint256));
23                 uint256 amount = IERC20(token).balanceOf(address(this));
24                 _setCallDataParameter(call.callData,
   amountParameterPosition, amount);
25             } else if (call.callType == CallType.FullNativeBalance) {
26                 call.value = address(this).balance;
27             } else if (call.callType == CallType.CollectTokenBalance) {
28                 address token = abi.decode(call.payload, (address));
29                 _safeTransferFrom(token, msg.sender,
   IERC20(token).balanceOf(msg.sender));
30                 continue;
31             }
32
33             (bool success, bytes memory data) = call.target.call{value:
   call.value}(call.callData);
34             if (!success) revert CallFailed(i, data);
35
36             unchecked {
37                 ++i;
38             }
39         }
40
41         isRunning = false;
42     }
```

Description

Given described a typical scenario in the documentation, the `SquidMulticall` contract implementation is too generic and does not perform any checks to ensure that the user cannot lose funds. Especially, it is not verified that:

- Ether (or the native currency) remaining after all calls are executed is returned to the caller (i.e. the `SquidMulticall` contract does not hold any Ether at the end of the transaction),
- all tokens are sent to the user or the `SquidRouter` contract (i.e. the `SquidMulticall` contract does not hold any tokens at the end of the transaction),
- up to one type of ERC-20 token is sent to the `SquidRouter` contract and this type of token is the same as the token type to be sent through the Axelar Gateway (i.e. the `SquidRouter` contract does not hold any tokens at the end of the transaction).

Exploit scenario

Due to faulty off-chain implementation, empty `calls` variable is passed to the `SquidMulticall.run` function. As a result, any tokens or Ether sent to the `SquidMulticall` contract (via the `SquidRouter.fundAndRunMulticall` function) are left in the contract. Before the user manages to call a transaction requesting the Ether or tokens back, another user calls the `SquidMulticall` contract and extracts the leftover Ether or tokens.

Recommendation

It is strongly advised to reconsider the current implementation of the `SquidMulticall` contract. If an architectural change is not an option, add safety checks for the invariants described in the previous paragraph and fix the following issues:

- M1: Missing `Call.callType` validation,
- M2: Missing `isContract` check in `SquidMulticall`,
- M3: Memory address overflow in `_setCallDataParameter`.

Go back to Findings Summary

M5: Re-entrancy in SquidRouter

Medium severity issue

Impact:	Medium	Likelihood:	Low
Target:	/packages/squidswap-contracts/contracts/SquidRouter.sol	Type:	Re-entrancy

Listing 6. Excerpt from /packages/squidswap-contracts/contracts//SquidRouter.sol#L72-L82[SquidRouter.callBridgeCall]

```
    function callBridgeCall(
        address token,
        uint256 amount,
        string calldata destinationChain,
        string calldata bridgedTokenSymbol,
        ISquidMulticall.Call[] calldata sourceCalls,
        ISquidMulticall.Call[] calldata destinationCalls,
        address refundRecipient,
        bool enableForecall
    ) external payable whenNotPaused {
        fundAndRunMulticall(token, amount, sourceCalls);
```

Listing 7. Excerpt from /packages/squidswap-contracts/contracts//SquidRouter.sol#L41-L51[SquidRouter.bridgeCall]

```
    function bridgeCall(
        string calldata destinationChain,
        string calldata bridgedTokenSymbol,
        uint256 amount,
        ISquidMulticall.Call[] calldata calls,
        address refundRecipient,
        bool enableForecall
    ) external payable whenNotPaused {
        address bridgedTokenAddress = gateway.tokenAddresses(bridgedTokenSymbol);

        _safeTransferFrom(bridgedTokenAddress, msg.sender, amount);
```

Listing 8. Excerpt from /packages/squidswap-contracts/contracts//SquidRouter.sol#L100-L114[SquidRouter.fundAndRunMulticall]

```
    function fundAndRunMulticall(
        address token,
        uint256 amount,
        ISquidMulticall.Call[] memory calls
    ) public payable {
        uint256 valueToSend;

        if (token == address(0)) {
            valueToSend = amount;
        } else {
            _transferTokenToMulticall(token, amount);
        }

        squidMulticall.run{value: valueToSend}(calls);
    }
```

Description

Assuming that tokens a user owns and/or swaps in the `SquidMulticall`

contract cannot be considered trusted, re-entrancy in the `SquidRouter` contract opens up the possibility to extract Ether (or the native currency) that would be otherwise used as an Axelar gateway fee.

Exploit scenario

The re-entrancy is possible in two different scenarios:

- a user calls `callBridgeCall` with a non-zero malicious `token` address and Ether to be paid to the Axelar Gas Service,
- `_transferTokenToMulticall` called from `fundAndRunMulticall` performs an external call to the malicious `token` address,
- `token` calls `fundAndRunMulticall` with zero `token` address, `amount` set to `address(msg.sender).balance` and `calls` saying to transfer all Ether to the malicious token,
- as a side effect, it is not paid to the Axelar Gas Service, resulting in tokens sent to the Axelar Gateway being stuck until the user pays the fee on the destination chain.

The second scenario is as follows:

- a user calls `bridgeCall` with Ether to be paid to the Axelar Gas Service,
- `_safeTransferFrom` called from `bridgeCall` performs an external call to the malicious `bridgedTokenAddress` address,
- the rest of the scenario is the same as above.

Recommendation

Add re-entrancy guards to the `SquidRouter` contract.

Go back to Findings Summary

M6: Missing `refundRecipient` validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	/packages/squidswap-contracts/contracts/SquidRouter.sol	Type:	Data validation

Description

Functions `bridgeCall` and `callBridgeCall` accept `refundRecipient` as a parameter. However, the value of the parameter is not validated. Given that the `refundRecipient` address is used to transfer funds in case of a revert in the `SquidMulticall` contract on the destination chain, lack of validation may lead to loss of funds.

Exploit scenario

Due to faulty off-chain implementation, the default value (which is the zero address) is passed to the `bridgeCall` function. The user-defined multicall on the destination chain fails and the bridged tokens are transferred to the zero address.

Recommendation

Add a check that the `refundRecipient` parameter is not the zero address.

Go back to Findings Summary

M7: Missing `destinationChain` validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	/packages/squidswap-contracts/contracts/SquidRouter.sol	Type:	Data validation

Description

Functions `bridgeCall`, `callBridge` and `callBridgeCall` accept `destinationChain` as a parameter. However, the value of the parameter is not validated. Additionally, Axelar `sendToken` and `callContractWithToken` functions do not perform the validation neither. This may lead to loss of funds if the `destinationChain` parameter is set to an invalid value.

Exploit scenario

Due to faulty off-chain implementation, the `destinationChain` parameter is set to an invalid value. Because Axelar does not perform any validation of the `destinationChain` parameter neither, the tokens sent to the Axelar gateway are burned on the source chain. The tokens are lost as Axelar does not support refunds of the tokens sent to an invalid destination chain.

Recommendation

Add validation for `destinationChain` parameters to all functions concerned to avoid potential loss of user funds. Axelar documents all supported chain names.

Go back to Findings Summary

W1: Missing validation of the 0x prefix in string addresses

Impact:	Warning	Likelihood:	N/A
Target:	Axelar Utils/contracts/StringAddressUtils.sol	Type:	Data validation

Listing 9. Excerpt from Axelar Utils/contracts/StringAddressUtils.sol#L6-L11[StringToAddress.toAddress]

```
6     function toAddress(string memory _a) internal pure returns (address)
  {
7         bytes memory tmp = bytes(_a);
8         if (tmp.length != 42) return address(0);
9         uint160 iaddr = 0;
10         uint8 b;
11         for (uint256 i = 2; i < 42; i++) {
```

Description

It is not checked whether the string passed to the `toAddress` function starts with the `0x` prefix.

Recommendation

Add a check to ensure that the string starts with the `0x` prefix.

Go back to Findings Summary

W2: Use of `solc` optimizer

Impact:	Warning	Likelihood:	N/A
Target:	**/*.sol	Type:	Compiler configuration

Description

Both audited projects use the `solc` optimizer. Enabling the optimizer may lead to unexpected bugs and should be used with caution. More significantly, both projects can be compiled with the latest version of the `solc` compiler that may be a subject to new undiscovered bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe to use in production.

Recommendation

Until the `solc` optimizer becomes more stable and undergoes more stringent security analysis, opt-out using it. This will ensure that the contracts are resilient to any existing bugs in the optimizer.

Go back to Findings Summary

W3: Address helper functions not respecting EIP-55

Impact:	Warning	Likelihood:	N/A
Target:	Axelar Utils/contracts/StringAddressUtils.sol	Type:	Standards violation

Description

EIP-55 defines a checksummed address format using mixed case letters to prevent mistyping of addresses. The functions `toAddress` and `toString` in `StringAddressUtils` do not respect this standard.

Recommendation

It should be either clearly stated in the documentation and NatSpec documentation strings that the functions do not respect EIP-55 or the functions should implement both EIP-55 checksum verification and generation.

Go back to Findings Summary

W4: SquidRouter pausable can be bypassed

Impact:	Warning	Likelihood:	N/A
Target:	/packages/squidswap-contracts/contracts/SquidRouter.sol	Type:	Logic error

Description

Given that the SquidRouter contract uses upgradeable proxies, functions paused through the proxy contract can still be executed by calling the function directly on the implementation contract. Furthermore, the pauser address (the address that can pause the contract) can be different when calling the function through the proxy contract and when calling the function directly on the implementation contract.

Recommendation

Either ensure that the fact that the SquidRouter pause ability can be bypassed is an expected behavior, or when pausing the SquidRouter contract, make sure to call the pause function both on the proxy contract and on the implementation contract.

Go back to Findings Summary

I1: Unnecessary abi.encodePacked

Impact:	Info	Likelihood:	N/A
Target:	Axelar Utils/contracts/StringBytesUtils.sol	Type:	Gas optimization

Listing 10. Excerpt from Axelar Utils/contracts/StringBytesUtils.sol#L15-L19[StringToBytes32.toBytes32]

```
15        uint256 stringNumber = uint256(bytes32(stringBytes));
16
17        // Storing string length as the last byte of the data
18        stringNumber |= 0xff & stringBytes.length;
19        return bytes32(abi.encodePacked(stringNumber));
```

Description

uint256 can be directly converted to bytes32 without using abi.encodePacked.

Recommendation

Remove the abi.encodePacked call.

Go back to Findings Summary

I2: Multiple calls to `pendingPauser`

Impact:	Info	Likelihood:	N/A
Target:	/packages/squidswap-contracts/contracts/RoledPausable.sol	Type:	Gas optimization

Listing 11. Excerpt from /packages/squidswap-contracts/contracts/RoledPausable.sol#L33-L37[RoledPausable.acceptPauser]

```
33    function acceptPauser() external {
34        if (msg.sender != pendingPauser()) revert NotPendingPauser();
35        _setPauser(pendingPauser());
36        PENDING_PAUSER_SLOT.setAddress(address(0));
37    }
```

Description

The `pendingPauser` function is called twice in the `acceptPauser` function, but there is no possibility of the pending pauser being set to a different address between these two calls.

Recommendation

The second call to `pendingPauser` can be replaced by `msg.sender`.

Go back to Findings Summary

I3: Bytes length accessed in a for loop condition

Impact:	Info	Likelihood:	N/A
Target:	Axelar Utils/contracts/StringAddressUtils.sol	Type:	Gas optimization

Listing 12. Excerpt from Axelar Utils/contracts/StringAddressUtils.sol#L25-L35[AddressToString.toString]

```
        bytes memory data = abi.encodePacked(a);
        bytes memory characters = '0123456789abcdef';
        bytes memory byteString = new bytes(2 + data.length * 2);

        byteString[0] = '0';
        byteString[1] = 'x';

        for (uint256 i; i < data.length; ++i) {
            byteString[2 + i * 2] = characters[uint256(uint8(data[i] >> 4))];
            byteString[3 + i * 2] = characters[uint256(uint8(data[i] & 0x0f))];
        }
```

Description

`data.length` is accessed in every iteration of the for loop. This is not necessary, as the length of the `data` variable is not modified in the loop.

Recommendation

`data.length` should be stored in a local variable before the loop, and the local variable should be used in the loop condition.

Go back to Findings Summary

I4: Inconsistent for loop incrementation

Impact:	Info	Likelihood:	N/A
Target:	Axelar Utils/contracts/StringAddressUtils.sol	Type:	Code style

Listing 13. Excerpt from Axelar Utils/contracts/StringAddressUtils.sol#L11-L11[StringToAddress.toAddress]

```
        for (uint256 i = 2; i < 42; i++) {
```

Listing 14. Excerpt from Axelar Utils/contracts/StringAddressUtils.sol#L32-L32[AddressToString.toString]

```
        for (uint256 i; i < data.length; ++i) {
```

Description

The loop `i` variable is incremented in the `AddressToString.toString` function using the post-fix syntax `i++`, while it is incremented using the pre-fix syntax `++i` in the `StringToAddress.toAddress` function.

Recommendation

Libraries in the same project should be consistent in their coding style.

Go back to Findings Summary

I5: Address code length can be checked before a call

Impact:	Info	Likelihood:	N/A
Target:	/packages/squidswap-contracts/contracts/{SquidMulticall.sol, SquidRouter.sol}	Type:	Gas optimization

Listing 15. Excerpt from /packages/squidswap-contracts/contracts/SquidMulticall.sol#L49-L53[SquidMulticall._safeTransferFrom]

```
        (bool success, bytes memory returnData) = token.call(
            abi.encodeWithSelector(IERC20.transferFrom.selector, from, address(this), amount)
        );
        bool transferred = success && (returnData.length == uint256(0) || abi.decode(returnData, (bool)));
        if (!transferred || token.code.length == 0) revert TransferFailed();
```

Listing 16. Excerpt from /packages/squidswap-contracts/contracts/SquidRouter.sol#L188-L192[SquidRouter._transferTokenToMulticall]

```
        (bool success, bytes memory returnData) = token.call(
            abi.encodeWithSelector(IERC20.transferFrom.selector, msg.sender, address(squidMulticall), amount)
        );
        bool transferred = success && (returnData.length == uint256(0) || abi.decode(returnData, (bool)));
        if (!transferred || token.code.length == 0) revert TransferFailed();
```

Description

The `token.code.length == 0` check can be performed before the actual call reducing the gas cost of the call in case `token` is not a contract.

Recommendation

Check the address code length before the call.

Go back to Findings Summary

I6: For loop variable can be incremented in an unchecked block

Impact:	Info	Likelihood:	N/A
Target:	Axelar Utils/contracts/StringAddressUtils.sol	Type:	Gas optimization

Listing 17. Excerpt from Axelar Utils/contracts/StringAddressUtils.sol#L11-L11[StringToAddress.toAddress]

```
        for (uint256 i = 2; i < 42; i++) {
```

Listing 18. Excerpt from Axelar Utils/contracts/StringAddressUtils.sol#L32-L32[AddressToString.toString]

```
        for (uint256 i; i < data.length; ++i) {
```

Description

Given the fact that library functions can be called many times in a single transaction, it is important to minimize the gas cost of each call. In this case, the for loop `i` variable can be incremented in an unchecked block to save gas.

Recommendation

Consider incrementing the for loop `i` variable in an unchecked block to save gas.

Go back to Findings Summary

I7: Missing NatSpec documentation

Impact:	Info	Likelihood:	N/A
Target:	**/*.sol	Type:	Documentation

Description

Both audited projects lack NatSpec documentation comments that are helpful for developers to understand the code.

Recommendation

Add NatSpec documentation to the source code, especially to public/external functions and state variables and libraries that are usually intended to be used by other contracts.

Go back to Findings Summary

I8: Inconsistent behavior: Revert vs return default

Impact:	Info	Likelihood:	N/A
Target:	Axelar Utils/contracts/{StringAddressUtils.sol, StringBytesUtils.sol}	Type:	Code style

Listing 19. Excerpt from Axelar Utils/contracts/StringAddressUtils.sol#L8-L8[StringToAddress.toAddress]

```
8        if (tmp.length != 42) return address(0);
```

Listing 20. Excerpt from Axelar Utils/contracts/StringBytesUtils.sol#L13-L13[StringToBytes32.toBytes32]

```
13        if (stringBytes.length == 0 || stringBytes.length > 31) revert
   InvalidStringLength();
```

Description

The function `toAddress` returns the zero address if the string is not a valid address while the function `toBytes32` reverts if the string cannot be stored in a `bytes32` variable. This behavior should be consistent across the libraries.

Recommendation

Revert the transaction if the input string of the `toAddress` function cannot be converted to a valid address.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Axelar: Utils & Squid Router, 19.10.2022.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

Appendix C: Woke outputs

This appendix shows the outputs from the Woke tool.

C.1. Detectors

Woke did not detect any issues in the scoped contracts in the Axelar Utils repository.

The following image lists detections for the contracts in the second audited (private) repository. Some reported detections resulted in the M5: Re-entrancy in `SquidRouter` issue. The rest of the detections were evaluated as false positives.

```
Woke
Using the following detectors:

 • function-call-options-not-called
   Function with call options actually is not called, e.g. this.externalFunction{value: targetValue}.
 • old-gas-value-not-called
   Function with gas or value set actually is not called, e.g.
   this.externalFunction.value(targetValue).
 • reentrancy
   Detects re-entrancy vulnerabilities.
 • unchecked-function-return-value
   Return value of a function call is ignored.
 • unsafe-address-balance-use
   Address.balance is either written to a state variable or used in a strict comparison (== or !=).
 • unsafe-delegatecall
   Delegatecall to an untrusted contract.
 • unsafe-selfdestruct
   Selfdestruct call is not protected.


┌─ Possible re-entrancy in `AxelarForecallable._safeTransferFrom` ──────────────────────────────────┐
│   204         address from,                                                                        │
│   205         uint256 amount                                                                      │
│   206     ) internal {                                                                            │
│ ❯ 207         (bool success, bytes memory returnData) = tokenAddress.call(                        │
│   208             abi.encodeWithSelector(IERC20.transferFrom.selector, from, address(this), amount) │
│   209         );                                                                                  │
│   210                                                                                             │
└─ @axelar-network/axelar-gmp-sdk-solidity/contracts/executables/AxelarForecallable.sol ─────────────┘
└──  ┌─ Exploitable from `SquidRouter.bridgeCall`, address is safe: None, state modified: SENDS_ETHER ─┐
     │   48     ) external payable whenNotPaused {                                                     │
     │   49         address bridgedTokenAddress = gateway.tokenAddresses(bridgedTokenSymbol);          │
     │   50                                                                                            │
     │ ❯ 51         _safeTransferFrom(bridgedTokenAddress, msg.sender, amount);                        │
     │   52         _bridgeCall(destinationChain, bridgedTokenSymbol, bridgedTokenAddress, calls, refu │
     │   53     }                                                                                      │
     │   54                                                                                            │
     └─ contracts/SquidRouter.sol ─────────────────────────────────────────────────────────────────────┘
```

```
┌─ Possible re-entrancy in `SquidRouter._bridgeCall` ──────────────────────────────────────────────
│  154
│  155         if (address(this).balance > 0) {
│  156             IAxelarGasService executionService = enableForecall ? forecallGasService : gasSer
│ ❯ 157             executionService.payNativeGasForContractCallWithToken{value: address(this).balanc
│  158                 address(this),
│  159                 destinationChain,
│  160
└─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────
└──┌─ Exploitable from `SquidRouter.bridgeCall`, address is safe: None, state modified: CALLS_UNIMPL─┐
   │  49         address bridgedTokenAddress = gateway.tokenAddresses(bridgedTokenSymbol);
   │  50
   │  51         _safeTransferFrom(bridgedTokenAddress, msg.sender, amount);
   │ ❯ 52         _bridgeCall(destinationChain, bridgedTokenSymbol, bridgedTokenAddress, calls, refu
   │  53     }
   │  54
   │  55
   └─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────


┌─ Possible re-entrancy in `SquidRouter._approve` ─────────────────────────────────────────────────
│  182     ) private {
│  183         if (IERC20(tokenAddress).allowance(address(this), spender) < amount) {
│  184             // Not a security issue since the contract doesn't store tokens
│ ❯ 185             IERC20(tokenAddress).approve(spender, type(uint256).max);
│  186         }
│  187     }
│  188
└─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────
└──┌─ Exploitable from `SquidRouter.bridgeCall`, address is safe: None, state modified: CALLS_UNIMPL─┐
   │  49         address bridgedTokenAddress = gateway.tokenAddresses(bridgedTokenSymbol);
   │  50
   │  51         _safeTransferFrom(bridgedTokenAddress, msg.sender, amount);
   │ ❯ 52         _bridgeCall(destinationChain, bridgedTokenSymbol, bridgedTokenAddress, calls, refu
   │  53     }
   │  54
   │  55
   └─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────


┌─ Possible re-entrancy in `SquidRouter._transferTokenToMulticall` ────────────────────────────────
│  187     }
│  188
│  189     function _transferTokenToMulticall(address token, uint256 amount) private {
│ ❯ 190         (bool success, bytes memory returnData) = token.call(
│  191             abi.encodeWithSelector(IERC20.transferFrom.selector, msg.sender, address(squidMul
│  192         );
│  193
└─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────
├──┌─ Exploitable from `SquidRouter.callBridge`, address is safe: False, state modified: SENDS_ETHER─┐
│  │  60         string calldata bridgedTokenSymbol,
│  │  61         ISquidMulticall.Call[] calldata calls
│  │  62     ) external payable whenNotPaused {
│  │ ❯ 63         fundAndRunMulticall(token, amount, calls);
│  │  64
│  │  65         address bridgedTokenAddress = gateway.tokenAddresses(bridgedTokenSymbol);
│  │  66
│  └─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────
├──┌─ Exploitable from `SquidRouter.callBridgeCall`, address is safe: False, state modified: SENDS_E─┐
│  │  81         address refundRecipient,
│  │  82         bool enableForecall
│  │  83     ) external payable whenNotPaused {
│  │ ❯ 84         fundAndRunMulticall(token, amount, sourceCalls);
│  │  85
│  │  86         address bridgedTokenAddress = gateway.tokenAddresses(bridgedTokenSymbol);
│  │  87
│  └─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────
└──┌─ Exploitable from `SquidRouter.fundAndRunMulticall`, address is safe: False, state modified: SE─┐
   │  109         if (token == address(0)) {
   │  110             valueToSend = amount;
   │  111         } else {
   │ ❯ 112             _transferTokenToMulticall(token, amount);
   │  113         }
   │  114
   │  115
   └─ contracts/SquidRouter.sol ──────────────────────────────────────────────────────────────────────
```

```
┌─ Unchecked return value ──────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│   182     ) private {                                                                                                                   │
│   183         if (IERC20(tokenAddress).allowance(address(this), spender) < amount) {                                                   │
│   184             // Not a security issue since the contract doesn't store tokens                                                       │
│ ❯ 185             IERC20(tokenAddress).approve(spender, type(uint256).max);                                                           │
│   186         }                                                                                                                           │
│   187     }                                                                                                                               │
│   188                                                                                                                                     │
└─ contracts/SquidRouter.sol ───────────────────────────────────────────────────────────────────────────────────────────────────────────┘


┌─ Possibly unsafe delegatecall in `Proxy.init` ────────────────────────────────────────────────────────────────────────────────────────┐
│   43             sstore(_OWNER_SLOT, newOwner)                                                                                            │
│   44         }                                                                                                                            │
│   45         // solhint-disable-next-line avoid-low-level-calls                                                                          │
│ ❯ 46         (bool success, ) = implementationAddress.delegatecall(                                                                       │
│   47             //0x9ded06df is the setup selector.                                                                                      │
│   48             abi.encodeWithSelector(0x9ded06df, params)                                                                               │
│   49                                                                                                                                      │
└─ @axelar-network/axelar-gmp-sdk-solidity/contracts/upgradables/Proxy.sol ──────────────────────────────────────────────────────────────┘
└──┬─ Exploitable from `Proxy.init`, address is safe: False ──────────────────────────────────────────────────────────────────────────────┐
   │   43             sstore(_OWNER_SLOT, newOwner)                                                                                        │
   │   44         }                                                                                                                        │
   │   45         // solhint-disable-next-line avoid-low-level-calls                                                                      │
   │ ❯ 46         (bool success, ) = implementationAddress.delegatecall(                                                                   │
   │   47             //0x9ded06df is the setup selector.                                                                                  │
   │   48             abi.encodeWithSelector(0x9ded06df, params)                                                                           │
   │   49                                                                                                                                  │
   └─ @axelar-network/axelar-gmp-sdk-solidity/contracts/upgradables/Proxy.sol ──────────────────────────────────────────────────────────┘


┌─ Possibly unsafe delegatecall in `Upgradable.upgrade` ────────────────────────────────────────────────────────────────────────────────┐
│   50                                                                                                                                      │
│   51         if (params.length > 0) {                                                                                                     │
│   52             // solhint-disable-next-line avoid-low-level-calls                                                                      │
│ ❯ 53             (bool success, ) = newImplementation.delegatecall(abi.encodeWithSelector(this.setu                                   │
│   54                                                                                                                                      │
│   55             if (!success) revert SetupFailed();                                                                                      │
│   56                                                                                                                                      │
└─ @axelar-network/axelar-gmp-sdk-solidity/contracts/upgradables/Upgradable.sol ──────────────────────────────────────────────────────────┘
└──┬─ Exploitable from `Upgradable.upgrade`, address is safe: False ──────────────────────────────────────────────────────────────────────┐
   │   50                                                                                                                                  │
   │   51         if (params.length > 0) {                                                                                                 │
   │   52             // solhint-disable-next-line avoid-low-level-calls                                                                  │
   │ ❯ 53             (bool success, ) = newImplementation.delegatecall(abi.encodeWithSelector(this.                                   │
   │   54                                                                                                                                  │
   │   55             if (!success) revert SetupFailed();                                                                                  │
   │   56                                                                                                                                  │
   └─ @axelar-network/axelar-gmp-sdk-solidity/contracts/upgradables/Upgradable.sol ──────────────────────────────────────────────────────┘
```

C.2. Graphs

The following graphs were used during the analysis.

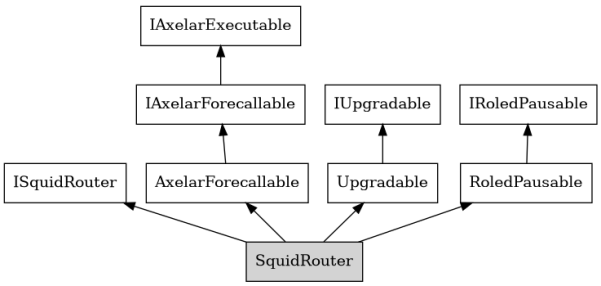

Figure 1. Inheritance graph of the SquidRouter contract

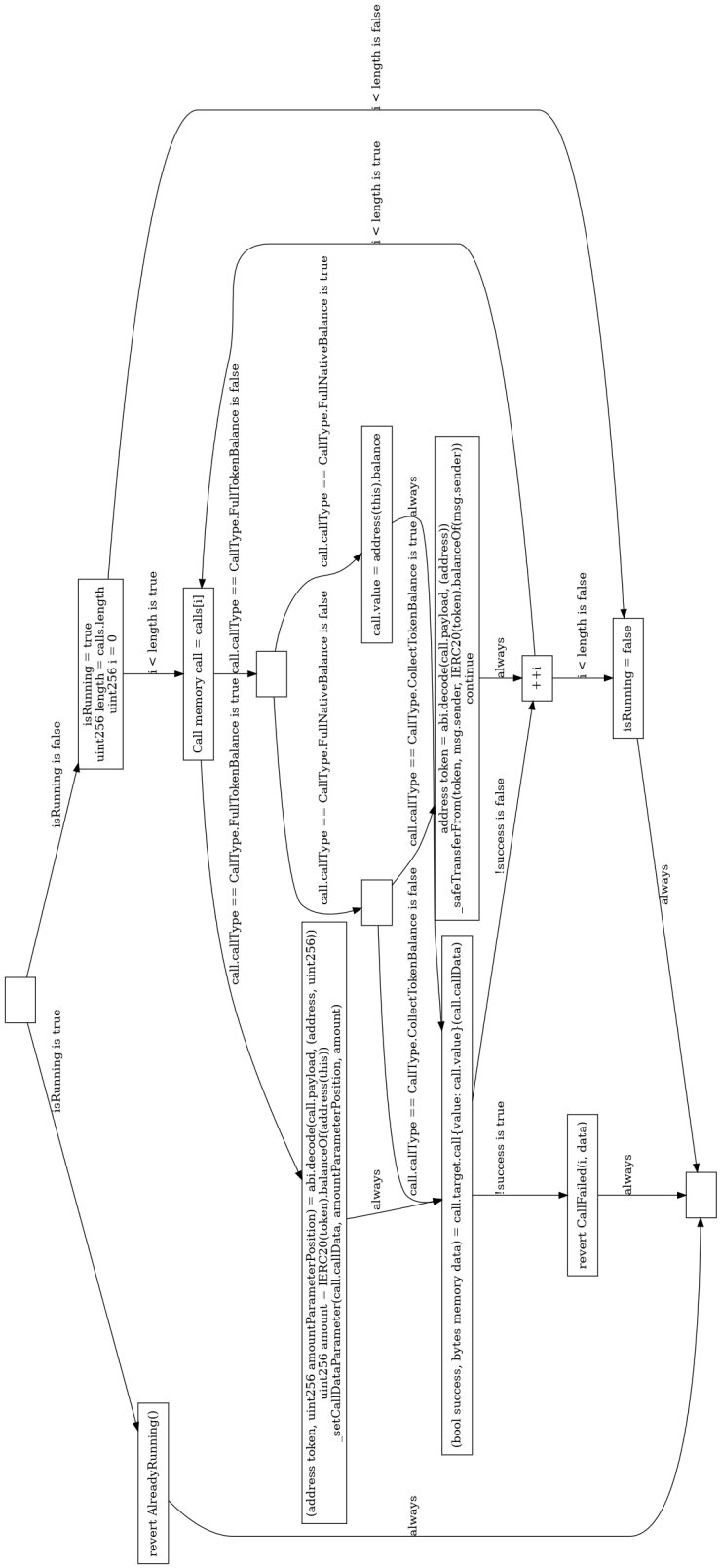

Figure 2. Control flow graph of the SquidMulticall.run function

Appendix D: Fuzz test sources

This appendix shows the fuzz tests used to verify the implementation of the provided contracts. All tests passed successfully.

D.1. ConstAddressDeployer fuzz test

```
import brownie
from brownie.test import given, strategy
from hypothesis import settings


@given(
    owner=strategy('address'),
    sender=strategy('address'),
    salt=strategy('bytes32'),
)
@settings(max_examples=5000)
def test_deploy(ConstAddressDeployer, owner, sender, salt):
    contract = ConstAddressDeployer.deploy({'from': owner})

    # omitted to fit in the document
    bytecode = "<<OMITTED>>"

    expected = contract.deployedAddress(bytecode, sender, salt)
    actual = contract.deploy(bytecode, salt, {'from': sender})

    with brownie.reverts():
        _ = contract.deploy(bytecode, salt, {'from': sender})

    expected2 = contract.deployedAddress(bytecode, sender, salt)
    assert expected == expected2
    assert expected == actual.return_value


@given(
    owner=strategy('address'),
    sender=strategy('address'),
    salt=strategy('bytes32'),
```

```
    init=strategy('bytes', min_size=0, max_size=1024)
)
@settings(max_examples=5000)
def test_deploy_and_init(ConstAddressDeployer, owner, sender, salt, init):
    contract = ConstAddressDeployer.deploy({'from': owner})

    # omitted to fit in the document
    bytecode = "<<OMITTED>>"

    expected = contract.deployedAddress(bytecode, sender, salt)
    actual = contract.deployAndInit(bytecode, salt, init, {'from': sender})

    with brownie.reverts():
        _ = contract.deployAndInit(bytecode, salt, init, {'from': sender})

    expected2 = contract.deployedAddress(bytecode, sender, salt)
    assert expected == expected2
    assert expected == actual.return_value
```

D.2. AddressToString fuzz test

```
from brownie.convert import to_address
from woke.fuzzer import Campaign
from woke.fuzzer.decorators import flow
from woke.fuzzer.random import random_account, random_bytes


class TestingSequence:
    def __init__(self, contract):
        self.contract = contract.deploy({'from': random_account()})

    @flow
    def flow_to_string(self):
        a = to_address("0x" + random_bytes(20, 20).hex())
        returned = self.contract.toString(a)
        assert str(a).lower()[2:] == returned[2:]


def test_address_to_string(AddressToString):
    campaign = Campaign(lambda: TestingSequence(AddressToString))
    campaign.run(1000, 400)
```

D.3. StringToAddress fuzz test

```
from brownie.convert import EthAddress
from woke.fuzzer import Campaign
from woke.fuzzer.decorators import flow
from woke.fuzzer.random import random_account, random_string


class TestingSequence:
    def __init__(self, contract):
        self.contract = contract.deploy({'from': random_account()})

    @flow
    def flow_to_address(self):
        passed = random_string(40, 40, "0123456789abcdef")
        returned = self.contract.toAddress("00" + passed)
        assert EthAddress("0x" + passed) == returned


def test_string_to_address(StringToAddress):
    campaign = Campaign(lambda: TestingSequence(StringToAddress))
    campaign.run(1000, 400)
```

D.4. StringToBytes32 fuzz test

```
import brownie
import string
from woke.fuzzer import Campaign
from woke.fuzzer.decorators import flow
from woke.fuzzer.random import random_account, random_string


class TestingSequence:
    def __init__(self, contract):
        self.contract = contract.deploy({'from': random_account()})

    @flow
    def flow_to_address(self):
        # also test unicode
        s = random_string(0, 35, string.ascii_letters + "ěščřžýáíé")
        b = s.encode('utf-8')
        if len(b) == 0 or len(b) >= 32:
            with brownie.reverts():
                self.contract.toBytes32(s)
        else:
            returned = self.contract.toBytes32(s)
            assert bytes(returned) == b + bytes(0 for _ in range(31 -
len(b))) + bytes([len(b)])


def test_string_to_bytes32(StringToBytes32):
    campaign = Campaign(lambda: TestingSequence(StringToBytes32))
    campaign.run(1000, 400)
```

D.5. StringToBytes32 and Bytes32ToString fuzz test

```
import brownie
import string
from woke.fuzzer import Campaign
from woke.fuzzer.decorators import flow
from woke.fuzzer.random import random_account, random_string


class TestingSequence:
    def __init__(self, StringToBytes32, Bytes32ToString):
        self.string_to_bytes = StringToBytes32.deploy({'from':
random_account()})
        self.bytes_to_string = Bytes32ToString.deploy({'from':
random_account()})

    @flow
    def flow_to_address(self):
        # also test unicode
        s = random_string(0, 35, string.ascii_letters + "ěščřžýáíé")
        b = s.encode('utf-8')
        if len(b) == 0 or len(b) >= 32:
            with brownie.reverts():
                self.string_to_bytes.toBytes32(s)
        else:
            print(s)
            assert s == self.bytes_to_string.toTrimmedString(
self.string_to_bytes.toBytes32(s))


def test_string_bytes_utils(StringToBytes32, Bytes32ToString):
    campaign = Campaign(lambda: TestingSequence(StringToBytes32,
Bytes32ToString))
    campaign.run(1000, 400)
```

ackee

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq