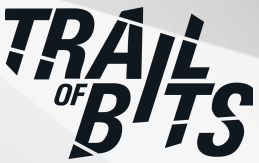

StarkNet Core Contracts

Security Assessment (Summary Report)

November 2, 2022

Prepared for:
Starkware
Starkware

Prepared by: **Alexander Remie and Jaime Iglesias**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

About Trail of Bits 1

Notices and Remarks 2

Table of Contents 3

Executive Summary 4

Project Summary 5

Project Goals 6

Project Targets 7

Project Coverage 8

Summary of Findings 9

Detailed Findings 10

1. Insufficient event generation 10
2. Configuration values could be updated to their existing values 11
3. Verifier address cannot be updated 12
4. Missing validation of the verifier address during contract initialization 14
5. Code comment describes behavior that is not implemented 15
6. Lack of overflow protection in encodeFactWithOnchainData 17
7. Governors can remove each other from the system 20
8. Confusing inheritance architecture can lead to errors 22
9. Reentrancy vulnerability in updateState 24

A. Vulnerability Categories 28

B. Code Quality Recommendations 30

Executive Summary

Engagement Overview

Starkware engaged Trail of Bits to review the security of its StarkNet core smart contracts. From September 19 to September 23, 2022, a team of two consultants conducted a security review of the client-provided source code, with two person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the target system, including access to the source code and documentation. We performed static testing of the target system and its codebase, using both automated and manual processes.

Summary of Findings

The audit did not uncover any significant flaws or defects that could impact system confidentiality, integrity, or availability. A summary of the findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	1
Low	1
Informational	6
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Auditing and Logging	1
Data Validation	2
Undefined Behavior	5

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Jaime Iglesias, Consultant
jaime.iglesias@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 7, 2022	Pre-project kickoff call
September 26, 2022	Delivery of report draft
September 28, 2022	Report readout meeting
November 2, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the StarkNet core smart contracts. Specifically, we sought to answer the following non-exhaustive list of questions:

- Could the configuration be updated by an account that is not a governor?
- Is it possible to cause a denial of service that prevents updates to the state from being processed?
- Does the use of the `delegatecall` proxy pattern cause any problems?
- Are all function inputs validated?
- Are events emitted in all areas of the codebase that should emit events?
- Could the system successfully process invalid state updates?
- Is the system vulnerable to reentrancy attacks?
- Do all of the codebase's functions have correct access controls?
- Could an attacker cancel another account's messages?
- Could the overall architecture of the smart contracts lead to problems?
- Could an attacker steal fees?

Project Targets

The engagement involved a review and testing of the following target.

cairo-lang

Repository	https://github.com/starkware-libs/cairo-lang
Version	`d61255f32a7011e9014e1204471103c719cfd5cb`
Type	Solidity
Platform	EVM

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Slither.** We ran Slither, our static analysis tool, over the Solidity contracts in the repository and triaged the findings. No major issues were identified.
- **Manual review.** We manually reviewed the Solidity contracts linked under Project Targets, with a focus on the governance, messaging, and state-updating parts of the system.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Insufficient event generation	Auditing and Logging	Informational
2	Configuration values could be updated to their existing values	Data Validation	Informational
3	Verifier address cannot be updated	Undefined Behavior	Informational
4	Missing validation of the verifier address during contract initialization	Data Validation	Low
5	Code comment describes behavior that is not implemented	Undefined Behavior	Informational
6	Lack of overflow protection in encodeFactWithOnchainData	Undefined Behavior	Informational
7	Governors can remove each other from the system	Access Controls	Medium
8	Confusing inheritance architecture can lead to errors	Undefined Behavior	Informational
9	Reentrancy vulnerability in updateState	Undefined Behavior	Undetermined

Detailed Findings

1. Insufficient event generation	
Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-STARK-1
Target: `src/starkware/starknet/solidity/Starknet.sol`	

Description
Several critical operations in the `Starknet` contract do not emit events. As a result, it will be difficult to review the correct behavior of the contract once it is deployed.

The following operations should trigger events:

- `Starknet.setProgramHash`
- `Starknet.setConfigHash`
- `Starknet.setMessageCancellationDelay`

Exploit Scenario
An attacker discovers a vulnerability in the `Starknet` contract and is able to modify its execution. Because the attacker's actions do not trigger any events, the behavior goes unnoticed until it has caused damage such as financial loss.

Recommendations
Short term, add events for all operations to strengthen the monitoring and alerting systems of the protocol. Events aid in contract monitoring and the detection of suspicious behavior.

Long term, consider using a blockchain-monitoring system to track any suspicious behavior in the contracts. The StarkNet system relies on several contracts to behave as expected. A monitoring mechanism for critical events would quickly detect any compromised system components.

2. Configuration values could be updated to their existing values	
Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STARK-2
Target: `src/starkware/starknet/solidity/Starknet.sol`	

Description

Several critical operations that update configuration parameters in the `Starknet` contract do not check that the given parameter's new value differs from its existing value. Updates that do not make any actual changes could confuse entities monitoring the `Starknet` contract.

The following operations should check that the given parameter's new value differs from its existing value:

- `Starknet.setProgramHash`
- `Starknet.setConfigHash`
- `Starknet.setMessageCancellationDelay`

Recommendations

Short term, add checks to the operations listed above that prevent configuration parameters from being updated to their existing values.

Long term, add validation to all function inputs.

3. Verifier address cannot be updated

Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-STARK-3
Target: `src/starkware/starknet/solidity/Starknet.sol`	

Description

Unlike other configuration values, the verifier address cannot be updated once it is set unless the `Starknet` implementation is reinitialized.

Certain configuration parameters of the system, such as the configuration hash or the program hash, can be changed directly through the governance system.

```
 43    function setProgramHash(uint256 newProgramHash) external notFinalized
onlyGovernance {
 44        programHash(newProgramHash);
 45    }
 46
 47    function setConfigHash(uint256 newConfigHash) external notFinalized
onlyGovernance {
 48        configHash(newConfigHash);
 49    }
```

Figure 3.1: Examples of configuration parameters that can be changed directly (`src/starkware/starknet/solidity/Starknet.sol`)

However, the verifier address cannot be changed directly unless the `Starknet` contract is reinitialized through the proxy.

```
 84    function setVerifierAddress(address value) internal {
 85        NamedStorage.setAddressValueOnce(VERIFIER_ADDRESS_TAG, value);
 86    }
```

Figure 3.2: The `setVerifierAddress` function (`src/starkware/starknet/solidity/Starknet.sol`)

```
112    function initializeContractState(bytes calldata data) internal override {
113        (
114            uint256 programHash_,
115            address verifier_,
116            uint256 configHash_,
117            StarknetState.State memory initialState
118        ) = abi.decode(data, (uint256, address, uint256, StarknetState.State));
```

```
119
120         programHash(programHash_);
121         setVerifierAddress(verifier_);
122         state().copy(initialState);
123         configHash(configHash_);
124         messageCancellationDelay(5 days);
```

Figure 3.3: The `Starknet` contract's initialization function (`src/starkware/starknet/solidity/Starknet.sol`)

To change the verifier address, developers would have to upgrade the proxy contract and reinitialize the `Starknet` contract.

Exploit Scenario

A bug is discovered in the verifier contract, requiring the contract to be redeployed. Because the `Starknet` contract cannot change the verifier address, developers have to upgrade the proxy contract and reinitialize the `Starknet` contract to make the change.

Recommendations

Short term, consider updating the code so that the verifier address can be changed through the governance system. Note that in the current implementation of the governance system, governors have the ability to unilaterally execute governance actions, meaning that they could change the verifier address to any arbitrary address. This may be undesirable behavior.

Long term, thoroughly document the configuration parameters that the governance system should be able to modify and any security implications that would affect the system if the governance system were compromised.

4. Missing validation of the verifier address during contract initialization

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-STARK-4
Target: `src/starkware/starknet/solidity/Starknet.sol`	

Description

The verifier address is not validated during the `Starknet` contract's initialization. As a result, it is possible to set the verifier address to the zero address. The verifier address can be set only once, so if it is mistakenly set to the zero address, the contract would have to be redeployed or upgraded to fix the error.

```
112    function initializeContractState(bytes calldata data) internal override {
113        (
114            uint256 programHash_,
115            address verifier_,
116            uint256 configHash_,
117            StarknetState.State memory initialState
118        ) = abi.decode(data, (uint256, address, uint256, StarknetState.State));
119
120        programHash(programHash_);
121        setVerifierAddress(verifier_);
122        state().copy(initialState);
123        configHash(configHash_);
124        messageCancellationDelay(5 days);
125    }
```

Figure 4.1: The `Starknet` contract's initialization function (`src/starkware/starknet/solidity/Starknet.sol`)

Exploit Scenario

Alice, an employee of Starkware, deploys the `Starknet` contract but mistakenly sets the verifier address to the zero address. The contract cannot be used since it cannot call the verifier.

Recommendations

Short term, add validation to the `Starknet` contract's initialization function that prevents the verifier address from being set to the zero address.

Long term, thoroughly document the expected values for the system's configuration and consider adding validation to all function inputs.

5. Code comment describes behavior that is not implemented

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-STARK-5
Target: `src/starkware/solidity/components/Governance.sol`	

Description

The `initGovernance` function contains a code comment describing behavior that is not actually implemented.

The `initGovernance` function's comment states that a variable is set to ensure that the next function to be executed, `acceptNewGovernor`, does not fail. However, the `acceptNewGovernor` function does not require this variable to be set.

```
39    function initGovernance() internal {
40       GovernanceInfoStruct storage gub = getGovernanceInfo();
41       require(!gub.initialized, "ALREADY_INITIALIZED");
42       gub.initialized = true; // to ensure acceptNewGovernor() won't fail.
43       // Add the initial governer.
44       acceptNewGovernor(msg.sender);
45    }
```

Figure 5.1: The `initGovernance` function's erroneous code comment, highlighted in yellow (`src/starkware/solidity/components/Governance.sol`)

```
79    function acceptNewGovernor(address newGovernor) private {
80       require(!_isGovernor(newGovernor), "ALREADY_GOVERNOR");
81       GovernanceInfoStruct storage gub = getGovernanceInfo();
82       gub.effectiveGovernors[newGovernor] = true;
83
84       // Emit governance information.
85       emit LogNewGovernorAccepted(newGovernor);
86    }
```

Figure 5.2: The `acceptNewGovernor` function (`src/starkware/solidity/components/Governance.sol`)

In fact, the `initialized` value for the new governor value is never used outside of the `initGovernance` function.

Recommendations

Short term, consider whether the behavior described in the `initGovernance` function's comment should be implemented. If so, implement the behavior; if not, remove the comment.

Long term, thoroughly document the expected behavior of the system and review the implementation to ensure that it behaves accordingly.

6. Lack of overflow protection in encodeFactWithOnchainData

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-STARK-6
Target: `src/starkware/solidity/components/OnchainDataFactTreeEncoder.sol.sol`	

Description

The `encodeFactWithOnchainData` function performs arithmetic operations without overflow protection.

The `Starknet` core contract uses the `OnchainDataFactTreeEncoder` library to build the fact to be proven when updating the `Starknet` state. To do so, the `encodeFactWithOnchainData` function encodes the fact using the inputs sent by the sequencer and hashes the result; finally, the verifier contract is queried to determine whether the fact is valid.

```
 19    function encodeFactWithOnchainData(
 20       uint256[] calldata programOutput,
 21       DataAvailabilityFact memory factData
 22    ) internal pure returns (bytes32) {
 23       // The state transition fact is computed as a Merkle tree, as defined in
 24       // GpsOutputParser.
 25       //
 26       // In our case the fact tree looks as follows:
 27       //   The root has two children.
 28       //   The left child is a leaf that includes the main part - the
information regarding
 29       //   the state transition required by this contract.
 30       //   The right child contains the onchain-data which shouldn't be accessed
by this
 31       //   contract, so we are only given its hash and length
 32       //   (it may be a leaf or an inner node, this has no effect on this
contract).
 33
 34       // Compute the hash without the two additional fields.
 35       uint256 mainPublicInputLen = programOutput.length;
 36       bytes32 mainPublicInputHash = keccak256(abi.encodePacked(programOutput));
 37
 38       // Compute the hash of the fact Merkle tree.
 39       bytes32 hashResult = keccak256(
 40           abi.encodePacked(
 41               mainPublicInputHash,
```

```
 42                mainPublicInputLen,
 43                factData.onchainDataHash,
 44                mainPublicInputLen + factData.onchainDataSize
 45            )
 46        );
 47        // Add one to the hash to indicate it represents an inner node, rather
than a leaf.
 48        return bytes32(uint256(hashResult) + 1);
 49    }
```

Figure 6.1: The `encodeFactWithOnchainData` function (`src/starkware/solidity/components/OnchainDataFactTreeEncoder.sol`)

```
157    function updateState(
158        uint256[] calldata programOutput,
159        uint256 onchainDataHash,
160        uint256 onchainDataSize
161    ) external onlyOperator {
162        // Validate program output.
163        StarknetOutput.validate(programOutput);
164
165        // Validate config hash.
166        require(
167            configHash() == programOutput[StarknetOutput.CONFIG_HASH_OFFSET],
168            "INVALID_CONFIG_HASH"
169        );
170
171        bytes32 stateTransitionFact =
OnchainDataFactTreeEncoder.encodeFactWithOnchainData(
172            programOutput,
173            OnchainDataFactTreeEncoder.DataAvailabilityFact(onchainDataHash,
onchainDataSize)
174        );
175        bytes32 sharpFact = keccak256(abi.encode(programHash(),
stateTransitionFact));
176        require(IFactRegistry(verifier()).isValid(sharpFact),
"NO_STATE_TRANSITION_PROOF");
          [...]
}
```

Figure 6.2: The `updateState` function, which queries the verifier to determine whether the fact is valid (`src/starkware/solidity/Starknet.sol`)

However, as shown in figure 6.1, the `encodeFactWithOnchainData` function performs a series of arithmetic operations without overflow protection; this is because the code is compiled using Solidity version 0.6. If any of these operations overflow, two different facts could collide during their encoding.

We classified this issue as informational, as the underlying hashing structure would likely prevent attackers from being able to create such collisions; however, it might still be possible to trigger this behavior by chance.

Exploit Scenario

The sequencer tries to update the `Starknet` state by calling the `updateState` function in the `Starknet` core contract. While the fact is being encoded, an overflow is triggered, resulting in the calculation of a different hash than the expected one.

The computed hash is not valid, so the state cannot be updated.

Recommendations

Short term, use `SafeMath` operations in the `encodeFactWithOnchainData` function to prevent overflows from affecting state updates. Note that using `SafeMath` will not prevent overflows; it will simply detect them and revert the execution, preventing the state update. Also note that if the state cannot be updated (e.g., because of a revert of the `encodeFactWithOnchainData` function), the `Starknet` contract could be left in an unusable state.

Long term, thoroughly document the risk of overflows in the `encodeFactWithOnchainData` function, the likelihood that they could occur, and the security implications for the StarkNet network if they do occur. Additionally, follow best practices, such as using `SafeMath`, when implementing arithmetic operations without native overflow protection.

7. Governors can remove each other from the system

Severity: **Medium**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-STARK-7
Target: `src/starkware/solidity/components/Governance.sol`	

Description

Compromised or malicious governors have the ability to unilaterally remove other governors from the system.

The `Starknet` core contract uses a governance system to control certain system parameters, such as the configuration hash and the list of operators (i.e., addresses that can call the `updateState` function).

The entities (addresses) that can exercise governance powers are called governors; there can be multiple governors active at a given time.

```
79    function acceptNewGovernor(address newGovernor) private {
80        require(!_isGovernor(newGovernor), "ALREADY_GOVERNOR");
81        GovernanceInfoStruct storage gub = getGovernanceInfo();
82        gub.effectiveGovernors[newGovernor] = true;
83
84        // Emit governance information.
85        emit LogNewGovernorAccepted(newGovernor);
86    }
```

Figure 7.1: The `acceptNewGovernor` function, showing the `effectiveGovernors` list (`src/starkware/solidity/components/Governance.sol`)

However, contrary to other governance systems such as DAOs, in which governance participants have to reach a majority to execute certain actions, each StarkNet governor has full governance powers. For example, any governor can unilaterally remove other governors from the system.

```
10    modifier onlyGovernance() {
11        require(_isGovernor(msg.sender), "ONLY_GOVERNANCE");
12        _;
13    }
```

Figure 7.2: The `onlyGovernance` modifier (`src/starkware/solidity/interfaces/MGovernance.sol`)

```
101  function _removeGovernor(address governorForRemoval) internal onlyGovernance {
102     require(msg.sender != governorForRemoval, "GOVERNOR_SELF_REMOVE");
103     GovernanceInfoStruct storage gub = getGovernanceInfo();
104     require(_isGovernor(governorForRemoval), "NOT_GOVERNOR");
105     gub.effectiveGovernors[governorForRemoval] = false;
106     emit LogRemovedGovernor(governorForRemoval);
107  }
```

Figure 7.3: The `removeGovernor` function, showing that any governor can perform this operation (`src/starkware/solidity/components/Governance.sol`)

Exploit Scenario

Eve, a malicious user, is able to gain access to a governor's private key. Using the compromised key, Eve calls the `starknetNominateNewGovernor` function to set herself as governor. In the same transaction, she calls the `starknetAcceptGovernance` function and the `starknetRemoveGovernor` function to remove the compromised key's associated address from the governance system, effectively gaining full control of it.

Recommendations

Short term, add a timelock to governance actions so that honest governors can act if a malicious or compromised governor tries to seize the system.

Long term, thoroughly document the expected behavior of the governance system, the types of accounts that the governors are (e.g., EOAs, multisignature wallets, DAOs, etc.), the impact that a compromised governor could have on the system, and the changes that could be made to minimize said impact (e.g., introducing emergency governance mechanisms, requiring a majority of governors to execute certain actions, etc.).

8. Confusing inheritance architecture can lead to errors

Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-STARK-8
Target: All Solidity contracts	

Description

Inheritance is extensively used to separate the StarkNet system into smaller logical components. However, in some instances, the separation of components seems unnecessary; in other instances, components that are logically separated in theory inherit from each other, making the architecture confusing.

Although inheritance is a great tool for creating modular architectures and for writing reusable code, it can easily be misused and result in confusing architectures and codebases that are difficult to navigate and maintain.

We identified some instances in which the use of inheritance and the resulting logical separation seem unnecessary. In these cases, two or more contracts could be merged into a single contract implementation, which would make code reviews, code maintenance, and developer onboarding easier to conduct.

For example, the governance system is split into multiple components:

- `GovernaceStorage.sol`
- `Governance.sol`
- `GenericGovernance.sol`
- `MGovernance.sol`
- `GovernedFinalizable.sol`
- `StarknetGovernance.sol`

There are a number of issues with this architecture.

First, `GovernanceStorage.sol` is never used.

Next, it is unclear why `MGovernance.sol` should be separated from `Governance.sol`. Additionally, certain components that are unrelated to governance, such as

`MOperator.sol`, seem to inherit from `MGovernance.sol` to use the `onlyGovernance` modifier, creating a very confusing design; these components could instead override the methods that need to use the modifier in the most derived contract (i.e., `Starknet.sol`).

Furthermore, `StarknetGovernance.sol` and `GenericGovernance.sol` are actually the same implementation, only that the names (i.e., the signatures) of the external functions are different. Consequently, any code change to either contract has to be made to both implementations; therefore, the StarkNet team should exercise caution when updating these components.

Finally, the `GovernedFinalizable.sol` contract seems like it should be an extension of the `Governance.sol` contract; however, it is implemented as a standalone contract that inherits from `MGovernance.sol` to use the `onlyGovernance` modifier instead of extending `Governance.sol` and overriding any methods that should be "finalizable."

Recommendations

Short term, thoroughly review the architecture of the system. Using both text and diagrams, document in detail the architecture, the intended use of each component (e.g., whether they are meant to be reusable, etc.), and the relationships between components. Investigate ways to merge deeply related components (such as `MGovernance.sol` and `Governance.sol`).

Long term, whenever the StarkNet team plans to add a new contract, follow a design-first approach: before development, first document the intended behavior of the contract, the users who will interact with it, the reasons for using it, and the way to use it, and produce an exhaustive list of properties that the contract should enforce.

9. Reentrancy vulnerability in updateState

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-STARK-9
Target: `src/starkware/starknet/solidity/Starknet.sol`	

Description

The introduction of fee payments sent to operators that update the `Starknet` state creates the possibility of reentrancy.

Operators are responsible for calling the `updateState` function to move the `Starknet` state forward on L1. To do so, the operator submits some data to the `Starknet` core contract. This data is encoded into a fact, which the verifier contract checks to ensure it has been proven; if the verifier's checks pass, the state will be moved forward. We can think of this updated state as "a snapshot of the L2 state that lives on L1."

```
157       function updateState(
158           uint256[] calldata programOutput,
159           uint256 onchainDataHash,
160           uint256 onchainDataSize
161       ) external onlyOperator {
162           // Validate program output.
163           StarknetOutput.validate(programOutput);
164
165           // Validate config hash.
166           require(
167               configHash() == programOutput[StarknetOutput.CONFIG_HASH_OFFSET],
168               "INVALID_CONFIG_HASH"
169           );
170
171           bytes32 stateTransitionFact =
OnchainDataFactTreeEncoder.encodeFactWithOnchainData(
172               programOutput,
173               OnchainDataFactTreeEncoder.DataAvailabilityFact(onchainDataHash,
onchainDataSize)
174           );
175           bytes32 sharpFact = keccak256(abi.encode(programHash(),
stateTransitionFact));
176           require(IFactRegistry(verifier()).isValid(sharpFact),
"NO_STATE_TRANSITION_PROOF");
177           emit LogStateTransitionFact(stateTransitionFact);
178
179           // Process L2 -> L1 messages.
180           uint256 outputOffset = StarknetOutput.HEADER_SIZE;
```

```
181             outputOffset += StarknetOutput.processMessages(
182                 // isL2ToL1=
183                 true,
184                 programOutput[outputOffset:],
185                 l2ToL1Messages()
186             );
187
188             // Process L1 -> L2 messages.
189             outputOffset += StarknetOutput.processMessages(
190                 // isL2ToL1=
191                 false,
192                 programOutput[outputOffset:],
193                 l1ToL2Messages()
194             );
195
196             require(outputOffset == programOutput.length,
"STARKNET_OUTPUT_TOO_LONG");
197
198             // Perform state update.
199             state().update(programOutput);
200             StarknetState.State storage state_ = state();
201             emit LogStateUpdate(state_.globalRoot, state_.blockNumber);
202         }
203     }
```

Figure 9.1: The `updateState` function (src/starkware/starknet/solidity/Starknet.sol)

As shown in figure 9.1, after the encoded fact is validated, the messages (L2 to L1 and L1 to L2) are processed and the state snapshot is updated. However, in the message processing function, before `updateState()` effectively changes the contract state, ETH is sent to the operator (figure 9.2).

```
 83       function processMessages(
 84           bool isL2ToL1,
 85           uint256[] calldata programOutputSlice,
 86           mapping(bytes32 => uint256) storage messages
 87       ) internal returns (uint256) {
108               [...]
109               if (isL2ToL1) {
110                   bytes32 messageHash = keccak256(
111                       abi.encodePacked(programOutputSlice[offset:endOffset])
112                   );
113
114                   emit LogMessageToL1(
115                       // from=
116                       programOutputSlice[offset +
MESSAGE_TO_L1_FROM_ADDRESS_OFFSET],
117                       // to=
118                       address(programOutputSlice[offset +
MESSAGE_TO_L1_TO_ADDRESS_OFFSET]),
```

```
119                         // payload=
120                         (uint256[])(programOutputSlice[offset +
MESSAGE_TO_L1_PREFIX_SIZE:endOffset])
121                     );
122                     messages[messageHash] += 1;
123                 } else {
124                     {
125                         bytes32 messageHash = keccak256(
126
abi.encodePacked(programOutputSlice[offset:endOffset])
127                         );
128
129                         uint256 msgFeePlusOne = messages[messageHash];
130                         require(msgFeePlusOne > 0, "INVALID_MESSAGE_TO_CONSUME");
131                         totalMsgFees += msgFeePlusOne - 1;
132                         messages[messageHash] = 0;
133                     }
                [...]
157             if (totalMsgFees > 0) {
158                 // NOLINTNEXTLINE: low-level-calls.
159                 (bool success, ) = msg.sender.call{value: totalMsgFees}("");
160                 require(success, "ETH_TRANSFER_FAILED");
161             }
162
163             return offset;
164         }
165     }
```

Figure 9.2: The `processMessages` function, showing that ETH is sent to the operator before the contract state is changed (`src/starkware/starknet/solidity/Output.sol`)

When the fees are sent to the operator, the control flow of the program is also passed to it. Therefore, if the operator is a contract, the operator could execute arbitrary code. Because the fees are sent before the state is updated, the operator could call any function in the contract, including `updateState`, or any other external contracts; if any of these functions or contracts depend on the `Starknet` core contract's state, such calls could result in state inconsistencies or the loss of funds.

We do not think a reentrancy into `updateState` directly is possible because the L1-to-L2 messages are consumed before the funds are sent to the operator, preventing the operator from consuming them multiple times; however, if future implementations of or general modifications to the contract were to change this behavior, this exploit scenario could become available. Finally, it is important to note that in the current state of the protocol, the operator is centralized; therefore, there is no immediate risk, but this could also change in the future.

Because this finding was still under investigation at the end of the engagement, we have classified its severity as undetermined.

Recommendations

Short term, modify the associated code to use the checks-effects-interactions pattern. This will prevent fees from being sent to the operator before the state is updated, thereby preventing reentrancy attacks that capitalize on an inconsistent state. Additionally, document the exploits that attackers could carry out by using the issue described in this finding, and evaluate their feasibility.

Long term, thoroughly document reentrancy opportunities in the system, and develop regression tests to identify code changes that would make reentrancy possible.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- The naming convention used for functions is inconsistent: sometimes underscores are used for internal functions, and sometimes they are not. Consider renaming certain functions so that function names are consistent.

```
function _isGovernor(address user) internal view override returns (bool)

function initGovernance() internal
```

- There are non-interface contracts in the `/interfaces` folder. Consider moving the abstract contracts into a different folder (e.g., `/components`) or moving them into a new folder (e.g., `/abstract`).

- Certain function names do not clearly convey their intended purposes. For example, changing `programHash(uint256 value)` to `_setProgramHash(uint256 value)` would make the following three function names less confusing:

```
function programHash(uint256 value) internal

function programHash() public view returns (uint256)

function setProgramHash(uint256 newProgramHash) external
```