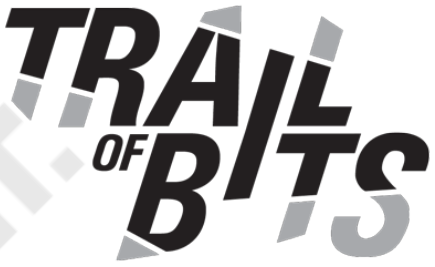

AElf Blockchain

Security Assessment

July 26, 2021

Prepared For:
Ada Niu | *AElf*
ada.niu@aelf.io

Prepared By:
Mike Martel | *Trail of Bits*
mike.martel@trailofbits.com

Rory Mackie | *Trail of Bits*
rory.mackie@trailofbits.com

Executive Summary

From July 12 to July 23, 2021, AElf engaged Trail of Bits to review the security of its blockchain platform. Trail of Bits conducted this assessment over four person-weeks, with two engineers working from commit hash 08878f6 from the AElf repository.

During the first week of the assessment, we gained an understanding of the codebase and its supporting documentation. We reviewed cryptographic components such as `CryptoHelper` and `TlsHelper`, network-related functionality including node peer-to-peer communication, and aspects of the Delegated Proof of Stake (DPoS) functionality and smart contract execution. In the final week of the assessment, we continued to focus on smart contract execution and sandboxing and reviewed a subset of the AElf smart contracts.

Our review resulted in nine findings ranging from high to informational severity. The most concerning high-severity finding involves the smart contract sandboxing functionality, which is insufficient to provide security guarantees and can provide a direct path to remote code execution (TOB-AELF-005). Other issues include the lack of a mechanism for removing a malicious miner ahead of scheduled elections (TOB-AELF-002) and the ambiguity surrounding consensus behavior (TOB-AELF-006). Several more result from the trust placed in client input and the system's misaligned economic incentives.

Given the significant size of the codebase and scope of the audit, we focused on the most security-critical components of the system. The two high-severity findings, particularly the issue that could result in remote code execution on nodes, highlight a need to increase the security maturity of the AElf platform. Additionally, while test coverage generally is high, the codebase lacks enough adversarial testing to appropriately cover malicious input.

Trail of Bits recommends addressing the findings in this report and implementing the short- and long-term recommendations. AElf should then perform another assessment to ensure that the fixes are adequate and do not introduce additional security risks. An assessment of the components not listed in the Coverage section (those omitted from the audit due to its large scope and time constraints) would also be beneficial.

Update: From August 9 to 11, 2021, Trail of Bits reviewed fixes implemented by AElf for the issues presented in this report. See a detailed review of the current status of each issue in Appendix C.

Project Dashboard

Application Summary

Name	AElf blockchain
Version	08878f6
Type	Blockchain
Platforms	.NET Core, C#

Engagement Summary

Dates	July 12–23, 2021
Method	Full knowledge
Consultants Engaged	2
Level of Effort	4 person-weeks

Vulnerability Summary

Total High-Severity Issues	2	■■
Total Medium-Severity Issues	3	■■■
Total Low-Severity Issues	3	■■■
Total Informational-Severity Issues	1	■
Total Undetermined-Severity Issues	0	
Total	9	

Category Breakdown

Access Controls	2	■■
Configuration	2	■■
Data Exposure	1	■
Data Validation	2	■■
Denial of Service	1	■
Undefined Behavior	1	■
Total	9	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Weak.** The smart contracts under review generally implemented appropriate access controls. However, the use of node-level API authentication (TOB-AELF-004) is a significant concern, as is the fact that numerous components rely on users to behave properly rather than enforcing security properties (TOB-AELF-005).
Centralization	**Weak.** The consensus mechanism and related economic incentives encourage the centralization of key infrastructure (TOB-AELF-003) and a dependence on parliamentary actors to assure security (TOB-AELF-005). Centralization and misaligned incentives can also lead to unexpected failure modes (TOB-AELF-007).
Function Composition	**Satisfactory.** The code was highly modularized and logically organized. However, certain areas would benefit from the use of built-in functions instead of custom workarounds with built-in names.
Front-Running	**Moderate.** Issues like those detailed in TOB-AELF-003 and TOB-AELF-004 indicate that the AElf blockchain may not be sufficiently protected against transaction front-running.
Specification	**Moderate.** The documentation on the codebase itself could be improved, as AElf uses tests, its documentation site, and the whitepaper to document the code. Additionally, certain key components remain undocumented, and the node setup instructions lack details on how to add a password to the Redis database instance.
Testing & Verification	**Satisfactory.** The codebase has extensive tests but would benefit from additional adversarial test coverage. A peer-to-peer client fuzzing tool would be very effective in detecting low-level vulnerabilities in the networking code.

Engagement Goals

This engagement was scoped to provide a security assessment of the AElf blockchain. Given the size of the codebase and the time constraints of the audit, Trail of Bits focused exclusively on the platform's core components, including networking-related code, the use of cryptography, code related to smart contract sandboxing and execution, and the blockchain consensus mechanism.

Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the AElf blockchain operate as specified in the documentation?
- Are smart contracts sufficiently sandboxed when they are executed?
- Are the network endpoints sufficiently scoped and protected?
- Could malicious actors abuse the blockchain consensus mechanisms?
- Is the cryptographic code implemented correctly, and does it adhere to best practices?
- Is the protocol designed in a way that incentivizes users to act fairly and as intended?

Coverage

Cryptographic code. We manually reviewed components of the `AElf.Cryptography` code, as well as other files, looking for any issues related to the initialization and use of cryptographic primitives. We also reviewed the external libraries used in the code.

`AElf.WebApp.*`. Our analysis of the web application components centered on the exposure of APIs, which could allow for node state manipulation; given this public exposure, we also focused on the authentication of requests. This review resulted in one finding, TOB-AELF-004.

`AElf.Kernel.*`. We performed static analysis and a detailed manual review of the kernel components, focusing on smart contract and node execution, the transaction pool, and the consensus code. Our review of the code implemented in the kernel led to several findings, including TOB-AELF-005, TOB-AELF-006, and TOB-AELF-008.

`AElf.OS.*`. We manually reviewed the AElf OS' core components, with a focus on network-related components such as the `NetworkService` and `PeerDiscoveryService`. This resulted in one finding, TOB-AELF-001.

`AElf.CSharp.*`. We performed a manual review of the `AElf.CSharp.*` code and assessed the smart contract execution services. We also analyzed the whitelisting and patching

mechanisms and looked for code injection opportunities to assess the security of the smart contract execution under both normal and adversarial circumstances.

`AElf.Contracts.*`. We analyzed the correctness of several contracts, though time constraints prevented us from achieving comprehensive coverage of them. We reviewed the `Genesis`, `Consensus`, `Parliament`, `Treasury`, and `Profit` contracts, focusing on the contracts' adherence to their specifications and on ensuring that adequate access controls were in place for sensitive methods. We uncovered several issues through our review of the contract code and AElf whitepaper (TOB-AELF-002, TOB-AELF-003, TOB-AELF-007, and TOB-AELF-009).

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

❑ **If constraints on API endpoints need to be enforced, do not rely on clients to respect them; instead, ensure that response methods include constraint verification.** TOB-AELF-001

❑ **Consider adding a mechanism through which validators can propose the removal of malicious nodes and can add them to the deny list outside of the standard election cycle.** TOB-AELF-002

❑ **Consider adding incentives for voters to maintain diversity among miner nodes within the network.** TOB-AELF-003

❑ **Consider adding a mechanism for selectively disabling HTTP API components such as those that manage peer connections so that public nodes can be used safely.** TOB-AELF-004

❑ **Implement sufficient automated server-side verification and sandboxing for submitted contract code and do not allow any smart contracts to be deployed until those systems are in place.** Additionally, implement a mechanism for scanning existing smart contracts, and add any that do not meet the current security requirements to the deny list. TOB-AELF-005

❑ **Define consensus rules and reconciliation behavior for situations in which two forks have the same block height, and implement the new consensus rules.** TOB-AELF-006

❑ **Consider scaling dividend payments based on the number of blocks a miner has missed to eliminate any economic incentives for opportunistic shutdowns.** TOB-AELF-007

❑ **Consider charging a minimum fee for all non-view transactions to create an economic disincentive for transaction spamming.** TOB-AELF-008

❑ **Prevent contracts from returning full stack traces to callers when throwing exceptions.** All callers need to know is that the contract execution has failed. TOB-AELF-009

Long Term

❑ **Document and test all system invariants to ensure that they are well understood and operate as intended.** TOB-AELF-001

❑ **Ensure that system components are coded defensively, and assume that malicious behavior will occur.** TOB-AELF-002, TOB-AELF-004, TOB-AELF-005

❑ **Consider simulating the economic behavior and the incentive system of the blockchain network to ensure that they are robust.** TOB-AELF-003, TOB-AELF-007, TOB-AELF-008

❑ **Ensure that protocol features are designed to account for even unlikely edge cases to increase the robustness of the protocol.** TOB-AELF-006

❑ **Ensure information is handled on a need-to-know basis.** Do not share potentially sensitive information to external parties unless they have a need for it. TOB-AELF-009

Findings Summary

#	Title	Type	Severity
1	Nodes do not enforce constraints on all API requests	Data Validation	Informational
2	Malicious miners can avoid removal until elections	Access Controls	High
3	Miner election mechanism encourages centralization	Configuration	Medium
4	Users can add and remove peer nodes without authenticating	Access Controls	Medium
5	Insufficient smart contract sandboxing	Data Validation	High
6	Certain consensus rules are not well defined	Undefined Behavior	Medium
7	Mining node downtime does not impact dividend payouts	Configuration	Low
8	Smart contracts that do not collect fees can be used to cause a denial of service	Denial of Service	Low
9	Exceptions in contract code leak stack traces to the caller	Data Exposure	Low

1. Nodes do not enforce constraints on all API requests

Severity: Informational
Type: Data Validation
Target: `Multiple files`

Difficulty: Low
Finding ID: TOB-AELF-001

Description
The AElf code includes several constraints that determine the default request-response behavior for node interactions. However, these constraints are not enforced by the responder and in some cases can be modified by the requester. For example, when a node requests a peer list from another node, the size of the list is limited to 10 randomly determined nodes by default. If a malicious node wanted to retrieve more information from its peers, it could modify that limit prior to making a request, and the responding node would return more than 10 nodes. The below code shows the lack of constraint enforcement when a node responds to such a request:

```
public Task<NodeList> GetRandomNodesAsync(int maxCount)
{
    var randomPeers = _nodes.OrderBy(x => RandomHelper.GetRandom()).Take(maxCount).Select(n
=> n.Value)
        .ToList();


    NodeList nodes = new NodeList();
    nodes.Nodes.AddRange(randomPeers);


    return Task.FromResult(nodes);
}
```

Figure 1.1: `NodeManager` code that does not enforce any request constraints

The lack of verification of request constraints could make it easier for a malicious node to enumerate all other nodes in a network. This may not be desirable, as it could allow a malicious node to gain a high level of centrality within the network.

Recommendations
Short term, if constraints on API endpoints need to be enforced, do not rely on clients to respect them; instead, ensure that response methods include constraint verification.

Long term, document and test all system invariants to ensure that they are well understood and operate as intended.

2. Malicious miners can avoid removal until elections

Severity: High
Type: Access Controls
Target: `DPoS protocol`

Difficulty: Low
Finding ID: TOB-AELF-002

Description
Malicious miners in the AElf system are detected by an automated mechanism but are evicted only if they have not mined a block in 72 hours. However, a miner who is elected and then exhibits detectable malicious behavior cannot be replaced ahead of the weekly miner election; there is no mechanism for holding an unscheduled vote. For instance, if a malicious miner executes transactions and includes bad results, validators will be able to detect the behavior but will lack a mechanism for adding the miner's public key to the deny list.

Exploit Scenario
Eve controls a malicious node that was elected as a miner. As she mines blocks, Eve forges transaction results, which is detected by other nodes validating the transactions. Eve continues to act maliciously, undermining trust in the deployed blockchain and effectively limiting blockchain transaction throughput.

Recommendations
Short term, consider adding a mechanism through which validators can propose the removal of malicious nodes and can add them to the deny list outside of the standard election cycle.

Long term, ensure that system components are coded defensively, and assume that malicious behavior will occur.

3. Miner election mechanism encourages centralization

Severity: Medium
Type: Configuration
Target: `DPoS protocol`

Difficulty: Low
Finding ID: TOB-AELF-003

Description
In weekly miner votes, the number of nodes elected is equal to `2n + 1`, where `n` starts at 8 nodes and can increase over time. The rewards distributed to miners are taken from the dividend pool: 20 percent of the dividend pool is earmarked for miners, and half of that amount is distributed evenly among miners. The other half is split into two portions. One is allocated to miners based on the proportion of votes received, and the other is allocated based on the number of consecutive re-elections. This incentivizes miners to repeatedly seek election; it also encourages voters to centralize their votes so that they will receive a greater proportion of the vote-weighted reward pool if elected.

Exploit Scenario
Eve operates a node with the intent of becoming a malicious miner. Centralized voting means that nodes at the tail end of the top `2n + 1` nodes will receive significantly fewer votes than those at the top; this enables Eve to increase her standing among the remaining nodes with fewer votes than she would otherwise need. In the event that a miner goes offline for 72 hours and is added to the deny list, Eve will be promoted to a miner and will be able to behave maliciously.

Recommendations
Short term, consider adding incentives for voters to maintain diversity among miner nodes within the network.

Long term, consider simulating the economic behavior and the incentive system of the blockchain network to ensure that they are robust.

References
- AElf Economic System Whitepaper

4. Users can add and remove peer nodes without authenticating

Severity: Medium
Type: Access Controls
Target: `src/AElf.WebApp.Application.Net/NetAppService.cs`

Difficulty: Low
Finding ID: TOB-AELF-004

Description
If a node has exposed its HTTP API to users, such as when it is acting as a receiver for incoming transactions, users will be able to add or remove peers without authenticating. Because there is no mechanism for selectively enabling or disabling portions of the API, filtering the requests of public nodes to prevent abuse would require a proxy in front of each node. The API endpoint that allows for this activity is `/api/net/peer`.

Exploit Scenario
Eve finds that Alice has an open HTTP API port and obtains Alice's peer list. She then instructs Alice to disconnect from all of her peers and to connect to 24 malicious nodes under Eve's control. As a result, Eve controls every peer in Alice's peer list, which by default is capped at 25 peers. Eve uses this control to drop incoming transactions from Alice's node.

Recommendations
Short term, consider adding a mechanism for selectively disabling HTTP API components such as those that manage peer connections so that public nodes can be used safely.

Long term, ensure that system components are coded defensively, and assume that malicious behavior will occur.

5. Insufficient smart contract sandboxing

Severity: High | Difficulty: Low
Type: Data Validation | Finding ID: TOB-AELF-005
Target: `Genesis contract, AElf.Kernel.CodeCheck, AElf.CSharp.CodeOps`

Description
AElf smart contracts are written in C# and compiled into bytecode that can be deployed directly through the `DeploySmartContract` or `ProposeNewContract` code path in the `Genesis` contract. If, at compile time, one is using AElf smart contract development tools such as those in the `aelf-boilerplate` repository, the code will be checked against an allow list of acceptable `.NET` assemblies, types, and methods; the resulting bytecode will be injected with an `ExecutionObserver` module that will count calls and branches during the contract's execution.

However, nodes do not attempt to verify any of the constraints imposed on smart contracts at compile time. A user could therefore bypass these checks and the insertion of the `ExecutionObserver` by using intermediate outputs of the build process. A malicious user could then execute arbitrary code on any node executing smart contract code, such as a miner or validator.

When users deploy correctly patched code without modifications, the smart contract sandboxing functionality generally works as intended. However, the functionality does allow for the use of code marked as `unsafe`, which can facilitate raw pointer manipulation; a user could abuse this deficiency to introduce non-deterministic contract behavior that impacts the transaction validation process and the overall security of the blockchain.

We tested several scenarios using both the `DeploySmartContract` and `ProposeNewContract` code paths and can confirm that it is possible to do the following:

- Manually patch over the `ExecutionObserver` instructions to allow an infinite loop to exhaust node resources or to crash a running node process using infinite recursion
- Use `System.Random` to introduce non-deterministic contract behavior, preventing transaction validation and finality
- Use `System.IO` to read and write files or `System.Net` to open arbitrary network connections to other systems

The AElf main chain uses a parliamentary system to decide whether to deploy a smart contract. The only way to prevent the above scenarios is for approvers to manually review submitted code. Given the breadth of the code that can be submitted and the extremely subtle bugs that can occur as a result, this sole mitigation is insufficient to provide any security guarantees.

Additionally, in a sidechain environment, especially one with more permissive rules for smart contract deployment, it would be even easier for a bad actor to attack nodes or the blockchain itself. In such an environment, a malicious actor could do much more than

leverage a smart contract bug to steal funds; instead, that actor could easily execute remote code on key network components.

Exploit Scenario
Eve deploys a malicious smart contract to either the AElf main chain or a sidechain. Eve includes code that downloads a malicious binary and executes it, gaining control over node host machines. She can then exfiltrate sensitive data such as nodes' private keys or influence the integrity of the blockchain.

Recommendations
Short term, implement sufficient automated server-side verification and sandboxing for submitted contract code and do not allow any smart contracts to be deployed until those systems are in place. Additionally, implement a mechanism for scanning existing smart contracts, and add any that do not meet the current security requirements to the deny list.

Long term, ensure that system components are coded defensively, and assume that malicious behavior will occur.

6. Certain consensus rules are not well defined

Severity: Medium
Type: Undefined Behavior
Target: `DPoS protocol`

Difficulty: High
Finding ID: TOB-AELF-006

Description
In the event of a fork in the AElf node network, the protocol relies on one primary rule for determining consensus: it uses the chain with the greatest block height when the nodes are reintegrated into a single consensus pool. However, if the sets of nodes reintegrated into the consensus pool are of the same size, and the node meant to serve as the tiebreaker is not added to either set, their block heights will remain equal. As such, the consensus code's assumption that there will always be a longest chain may not hold, as shown in figure 6.1:

```
if (isLinkedToLongestChain && chainBlockLink.Height > chain.LongestChainHeight
    || chainBlockLink.Height >= chain.LongestChainHeight + 8)
{
    chain.LongestChainHeight = chainBlockLink.Height;
    chain.LongestChainHash = chainBlockLink.BlockHash;
    status |= BlockAttachOperationStatus.LongestChainFound;
}
```

Figure 6.1: `IChainManager` code that fails to handle block height equality in a well-defined way

Exploit Scenario
Eve is able to induce a network partition such that two sets of competing nodes produce an equal number of blocks at the time of the fork. This results in a consensus deadlock or causes one fork to be effectively randomly selected over the other based on the other network state. This can impact blockchain participants' confidence in the consensus properties.

Recommendations
Short term, define consensus rules and reconciliation behavior for situations in which two forks have the same block height, and implement the new consensus rules.

Long term, ensure that protocol features are designed to account for even unlikely edge cases to increase the robustness of the protocol.

References
- AElf Consensus Documentation

7. Mining node downtime does not impact dividend payouts

Severity: Low
Type: Configuration
Target: `DPoS protocol`

Difficulty: High
Finding ID: TOB-AELF-007

Description
After weekly node elections, miners are added to the appropriate dividend pools, which provide economic incentives for filling the miner role. However, these payments are not associated with the number of blocks produced by a given miner. Since miners can be inactive for 72 hours a week without being removed from the miner pool, they may be incentivized to refrain from participating in the network, as they will continue earning dividends without incurring operating costs.

Exploit Scenario
Eve, a miner, wants to reduce her operating costs and increase her profit margin. She selectively shuts down her miner node but respects the limit on the number of missed blocks. This decreases the effective throughput of the blockchain. However, Eve can still collect dividends during these shutdowns despite her failure to contribute to the AElf blockchain.

Recommendations
Short term, consider scaling dividend payments based on the number of blocks a miner has missed to eliminate any economic incentives for opportunistic shutdowns.

Long term, consider simulating the economic behavior and the incentive system of the blockchain network to ensure that they are robust.

References
- AElf Economic System Whitepaper

8. Smart contracts that do not collect fees can be used to cause a denial of service

Severity: Low
Type: Denial of Service
Target: `AElf.Kernel.SmartContract`

Difficulty: Low
Finding ID: TOB-AELF-008

Description
AElf smart contracts must implement the AElf ACS1 standard in order to charge transaction fees; otherwise, the only transaction fee charged to users is that related to a transaction's size. If a smart contract does not implement the appropriate interfaces but engages in non-trivial computation, it will be possible for a malicious user to generate thousands of unique transactions on a node, wasting system resources at no cost to the user. Since node transaction pools are limited by default to 5,120 transactions, this behavior could prevent others from submitting their own valid transactions.

Exploit Scenario
Eve, a malicious user, wants to prevent other users from submitting transactions to a node. She finds a smart contract that performs a computationally intense set of operations and does not charge transaction fees. Eve then spams the node with requests, filling the transaction pool.

Recommendations
Short term, consider charging a minimum fee for all non-view transactions to create an economic disincentive for transaction spamming.

Long term, consider simulating the economic behavior and the incentive system of the blockchain network to ensure that they are robust.

9. Exceptions in contract code leak stack traces to the caller

Severity: Low
Type: Data Exposure
Target: `AElf.Contracts.*`

Difficulty: Low
Finding ID: TOB-AELF-009

Description
Smart contracts on the AElf network may throw exceptions when encountering unknown, unexpected, or malicious input. These exceptions may include sensitive information from the node executing the contract, such as the working directory, needlessly leaking information to any attackers.

Exploit Scenario
Eve sends a malicious RPC to Alice's node, asking it to execute a built-in smart contract with invalid parameters. Alice then executes the smart contract and encounters an exception. The resultant stack trace includes the username on Alice's node and is returned verbatim to Eve, who can infer Alice's username and operating system from the message.

Recommendations
Short term, prevent contracts from returning full stack traces to callers when throwing exceptions. All callers need to know is that the contract execution has failed.

Long term, ensure information is handled on a need-to-know basis. Do not share potentially sensitive information to external parties unless they have a need for it.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing a system failure
Documentation	Related to documentation errors, omissions, or inaccuracies
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.

Medium	Individual users' information is at risk; exploitation could pose reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

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

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Fix Log

From August 9 to 11, 2021, Trail of Bits reviewed fixes for issues identified in this report. The review was performed by one engineer over three person-days, using pull requests identified by the client relevant to the fixes. Of the issues reported in the original assessment, the AElf team fixed 7 and partially fixed 1; 1 issue was not addressed. We reviewed each of the fixes to help ensure that the proposed remediation would be effective.

#	Title	Type	Severity	Status
1	Nodes do not enforce constraints on all API requests	Data Validation	Informational	Fixed
2	Malicious miners can avoid removal until elections	Access Controls	High	Fixed
3	Miner election mechanism encourages centralization	Configuration	Medium	Fixed
4	Users can add and remove peer nodes without authenticating	Access Controls	Medium	Fixed
5	Insufficient smart contract sandboxing	Data Validation	High	Partially fixed
6	Certain consensus rules are not well defined	Undefined Behavior	Medium	Fixed
7	Mining node downtime does not impact dividend payouts	Configuration	Low	Fixed
8	Smart contracts that do not collect fees can be used to cause a denial of service	Denial of Service	Low	Not fixed
9	Exceptions in contract code leak stack traces to the caller	Data Exposure	Low	Fixed

Detailed Fix Log

This section briefly describes fixes made after the assessment and reviewed by Trail of Bits.

Finding 1:
Fixed. PR 3292 resolves this issue by ensuring that the number of nodes returned by an API call cannot exceed `GrpcConstants.DefaultDiscoveryMaxNodesToResponse`.

Finding 2:
Fixed. PR 3292 resolves this issue by adding an "Emergency Response Organization" that allows a 90% vote of the nodes in parliament to remove a known-malicious node that is not detected automatically through other means.

Finding 3:
Fixed. PR 3298 resolves this issue by adding a "Welcome Reward" and "Flexible Reward" that increases the rewards for miners elected to parliament for the first time, which provides incentives for voters to be less centralized.

Finding 4:
Fixed. PR 3293 resolves this issue by adding basic authentication over HTTP to sensitive API requests. After we provided additional feedback on the fix, additional measures were put in place to ensure that requests will fail if default, empty credentials are used.

Finding 5:
Partially fixed. PR 3294 partially resolved this issue by ensuring that the same code checking mechanism that is included in their build tools will also execute on nodes validating contract deployment. This mostly works as intended. However, a node that accepts an invalid smart contract will cease mining activity until it can be restarted, which creates a denial of service vector. A more robust solution that does not require downtime for miners or validators that performs the same code checking would be needed to completely fix this issue. Additionally, there are still concerns around allowable code, such as the use of `unsafe`, that were not addressed in this fix.

Finding 6:
Fixed. PR 3296 resolves this issue by rolling back state in the event of a deadlock on-chain. This is a valid method to reconcile equal-height forks.

Finding 7:
Fixed. PR 3298 resolves this issue by adding a mechanism to slash miner rewards should they miss too many blocks on average relative to other miners.

Finding 8:
Not fixed. Transaction fees are calculated after transaction execution, and would require significant reworking of the transaction execution engine to facilitate this type of change, which was not attempted during the scope of this fix review.

Finding 9:
Fixed. PR 3299 resolves this issue by removing the printing of full stack traces in favor of the top-level error only.