CHAIN TROOPERS

Energy Web Foundation

Bridge of T1 and Substrate sidechain T2

Security Assessment Report

April 30th, 2024

Version 1.1

CONFIDENTIAL

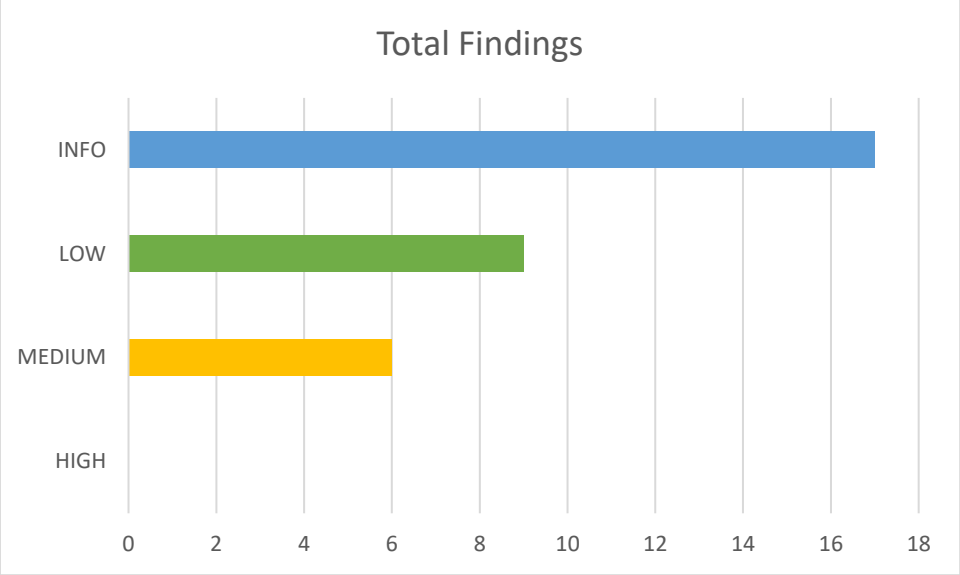
Table of Contents

1 Executive Summary

1.1 Introduction

The report contains the results of Energy Web security assessment that took place from March 6th, 2024, to March 28th, 2024. The in-scope component was the bridge of Energy Web (T1) with a Substrate-based sidechain (T2). The security engineers performed an in-depth manual analysis of the provided functionalities, and uncovered issues that may be used by adversaries to affect the confidentiality, the integrity, and the availability of the in-scope components.

In total, the team identified fifteen (15) vulnerabilities. There were also seventeen (17) informational issues of no-risk.



All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity. A retesting phase was carried out on April 30th, 2024, and the results are presented in Section 6.

1.2 Assessment Results

The assessment results revealed that the in-scope application components were mainly vulnerable to six (6) Business Logic, Third-Party components and Data Validation issues of MEDIUM risk. Regarding the Business Logic issues, the team identified that the implementation of the “add_corroboration” functionality does not enforce that the transaction sender cannot submit a valid (properly signed) corroboration *("5.1.3 - Potential voting fraud by transaction sender in "add_corroboration" at "eth-bridge""),* allowing the sender to influence the result of the voting for their own benefit. Moreover, it was found that the weight of multiple functionalities in the “summary”, “eth-bridge” and the “ethereum-events” pallets is not correctly calculated *("5.1.1 - Weight of "create_and_report_summary_offence" is not calculated in multiple functionalities at "summary"", "5.1.2 - Weight of MAX_NUMBER_OF_ROOT_DATA_PER_RANGE is not calculated in "record_summary_calculation" at "summary"", "5.1.6 - Weight of ConfirmationsLimit is not calculated in "add_confirmation" at "eth-bridge""),* permitting adversaries to overload the network with messages that call these functionalities, exhaust the available blockchain computation time and eventually stop the network from producing new blocks. As a result, these issues can be exploited to perform a denial-of-service (DoS) attack.

In reference to the MEDIUM-risk Data Validation issues, the team identified that the “avn-service” APIs do not check the size of the HTTP body, before processing it (*"5.1.4 - The size of the HTTP body in "send_main", "view_main" and "tx_query_main" functionalities is not checked at "avn-service""),* allowing adversaries who are able to trick the off chain workers to submit large packets, to consume all the available processing resources, and conduct a denial of service (DoS) attack. Regarding the Third-Party components issues of MEDIUM-risk, the team identified that multiple dependencies in *“avn-parachain”* and the pallets that were outdated and vulnerable (*"5.1.5 - Outdated and Vulnerable dependencies in-use at "Cargo.lock""),* allowing adversaries to target and exploit the vulnerable components

The in-scope components were also affected by nine (9) Data Validation, Administration, Information Disclosure and Transmission Security vulnerabilities of LOW risk. Regarding the LOW-risk Data Validation issue, it was found that the "*_releaseFunds*()" functionality is affected by a gas griefing vulnerability *("5.2.1 - Potential gas griefing attack in "_releaseFunds()" at "EnergyBridge.sol""),* allowing the recipient of "*claimLower*" and "*legacyLower*" functionalities to return a large

memory array that will consume a lot of gas, overcharging the caller or even making the calls to fail. Furthermore, it was found that the "*/roothash/:from_block/:to_block*" API call does not validate that the provided input parameters are within an acceptable range *("5.2.5 - No max range limitation in "/roothash/:from_block/:to_block" API call at "avn-service""),* allowing an adversary who is able to gain access to the API through another attack vector, to provide a very large range in order to force the service to exhaust the available resources, leading to a Denial of Service (DoS) attack. Additionally, the team identified that application does not validate the return value of the *"ecrecover()"* functionality (*"5.2.7 - Return value of "ecrecover()" is not checked at "EnergyBridge.sol""*), ignoring the error state. An adversary who is able to trick the admin user into registering a collator with the zero address, would be able to provide any invalid signature and potentially circumvent the confirmation control.

In reference to the LOW-risk Transmission Security issues, it was found that the "avn-service" pallet makes asynchronous calls to external blockchain services using the "web3_utils", without defining a max-time limitation *("5.2.8 - No max-time limitation in asynchronous calls to external services in "web3_utils" at "avn-service"")*. If these services are slow to respond or are unavailable, they could tie up server resources, leading to a denial-of-service condition.

Regarding the Administration issues of LOW risk, it was found that the "*claimLower*()" function emits events that only include the "*lowerId*" *("5.2.2 - Insufficient Information for event emission in "claimLower()" at "EnergyBridge.sol"")*. While it is still possible for a monitoring solution to correlate this identifier with a previously submitted proof, it is impossible to use it to retrieve the transaction details, such as the "token", the "recipient", the "t2PubKey", and the "amount", as no public view functionality is available. Furthermore, it was identified that the "*settle_transfer*()" function does not emit an event for a security-critical operation *("5.2.9 - Missing event emit in "settle_transfer" at "token-manager""),* reducing the transparency, and affecting the credibility of the system. Moreover, it was found that the owner of the Solidity contract is able to renounce its ownership *("5.2.6 - Owner can renounce Ownership at "EnergyBridge.sol""),* as the smart contract inherits the business logic from "*OwnableUpgradeable*" contract, which includes a "*renounceOwnership*()" function that can be potentially dangerous in certain contexts.

Finally, In reference to LOW-risk Information Disclosure issues, it was found that the "*avn-service*" node may leak sensitive information in server error messages (*"5.2.3 - Insecure error handling exposes sensitive information in "raw_public_keys" at "avn-service"", "5.2.4 - Insecure error handling exposes sensitive information in "key_phrase_by_type" at "avn-service""),* allowing an adversary to infer information regarding the application inner workings regarding the private key management process.

There were also seventeen (17) findings of no-risk (INFORMATIONAL). Chaintroopers recommend the immediate mitigation of all MEDIUM-risk issues. It is also advisable to address all LOW and INFORMATIONAL findings to enhance the overall security posture of the components.

1.2.1 Retesting Results

Results from retesting carried out on April 2024, determined that four (4 out of 6) reported MEDIUM risk issues were sufficiently addressed (see sections *5.1.1, 5.1.3, 5.1.4* and *5.1.6*).

Furthermore, two (2 out of 9) reported LOW risk issues and two (2 out of 17) reported issues bearing no risk (INFORMATIONAL) were also found to be sufficiently mitigated (see sections *5.2.2, 5.2.6*, , *5.3.3* and *5.3.6*). One (1 out of 17) INFORMATIONAL issues was also marked as risk-accepted (see section *5.3.2*).

The rest twenty-three (23 out of 32 total findings) were found to remain OPEN (see sections *5.1.2, 5.1.5, 5.2.1, 5.2.3, 5.2.4, 5.2.5, 5.2.7, 5.2.8, 5.2.9, 5.3.1, 5.3.4, 5.3.5, 5.3.7, 5.3.8, 5.3.9, 5.3.10, 5.3.11, 5.3.12, 5.3.13, 5.3.14, 5.3.15, 5.3.16* and *5.3.17*). More information can be found in Section 6.

1.3 Summary of Findings

The following findings were identified in the examined source code:

Vulnerability Name	Status	Retest Status	Page
Weight of "create_and_report" is not calculated in multiple functionalities at "summary", "eth-bridge" and "ethereum-events"	MEDIUM	CLOSED	19
Weight of MAX_NUMBER_OF_ROOT_DATA_PER_RANGE is not calculated in "record_summary_calculation" at "summary"	MEDIUM	MEDIUM	25
Potential voting fraud by transaction sender in "add_corroboration" at "eth-bridge"	MEDIUM	CLOSED	28
The size of the HTTP body in "send_main", "view_main" and "tx_query_main" functionalities is not checked at "avn-service"	MEDIUM	CLOSED	30
Outdated and Vulnerable dependencies in-use at "Cargo.lock"	MEDIUM	MEDIUM	33
Weight of ConfirmationsLimit is not calculated in "add_confirmation" at "eth-bridge"	MEDIUM	CLOSED	37
Potential gas griefing attack in "_releaseFunds()" at "EnergyBridge.sol"	LOW	LOW	41
Insufficient Information for event emission in "claimLower()" at "EnergyBridge.sol"	LOW	CLOSED	43
Insecure error handling exposes sensitive information in "raw_public_keys" at "avn-service"	LOW	LOW	45

Insecure error handling exposes sensitive information in "key_phrase_by_type" at "avn-service"	LOW	LOW	47
No max range limitation in "/roothash/:from_block/:to_block" API call at "avn-service"	LOW	LOW	49
Owner can renounce Ownership at "EnergyBridge.sol"	LOW	CLOSED	52
Return value of "ecrecover()" is not checked at "EnergyBridge.sol"	LOW	LOW	54
No max-time limitation in asynchronous calls to external services in "web3_utils" at "avn-service"	LOW	LOW	57
Missing event emit in "settle_transfer" at "token-manager"	LOW	LOW	60
Use of "ecrecover()" at EnergyBridge.sol	INFO	INFO	62
Removal of Merkle tree root hash is not possible at "EnergyBridge.sol"	INFO	Risk Accepted	64
Costly length comparison at "ethereum-events" and "summary"	INFO	CLOSED	66
Use of "encodePacked" at "EnergyBridge.sol"	INFO	INFO	68
Remove “legacyLower()” from EnergyBridge.sol	INFO	INFO	70
Insufficient argument validation in "_initialiseAuthors()" at "EnergyBridge.sol"	INFO	CLOSED	71
Off-by-one access in "_verifyConfirmations()" at "EnergyBridge.sol"	INFO	INFO	73

Empty array access in "_verifyConfirmations()" at "EnergyBridge.sol"	INFO	INFO	75
Empty array access in "confirmTransaction()" at "EnergyBridge.sol"	INFO	INFO	77
Unbounded recursion in "request::process_next_request" at "eth-bridge"	INFO	INFO	79
File-based instead of a hardware-backed keystore in-use at "avn-service"	INFO	INFO	81
Strict Transport Security (HSTS) is not enforced in Tide framework at "avn-service"	INFO	INFO	84
MIME sniffing is not disabled in Tide framework at "avn-service"	INFO	INFO	86
CSP is not enabled in Tide framework at "avn-service"	INFO	INFO	87
HTTP Caching is allowed in Tide framework at "avn-service"	INFO	INFO	89
TLS/SSL is not enabled in Tide framework at "avn-service"	INFO	INFO	91
Authentication not enforced in "/eth/sign" and "/eth/send" functionalities at "avn-service"	INFO	INFO	93

2 Assessment Description

2.1 Target Description

This Energy Web smart contract is a bridge between two blockchains: Energy Web tier 1 (T1) and AVN tier 2 (T2). A bridge connects two blockchains, enabling the bilateral transfer of data and currency between the chains. Bridged chains are linked by nodes that simultaneously operate on both of the blockchain networks.

In the specific case, the bridge allows authors to publish the transactional state of T2 to the contract periodically, enables the addition and removal of collators from consensus, and facilitates the "lifting" and "lowering" of EWT (Energy Web Token) or ERC20 tokens between the two blockchains. Collators are special participants from the Energy Web blockchain (T1) who play a role in making sure transactions between the two blockchains happen in a secure way.

The key functionalities are the following:

- **Author Management:** The addition and removal of EWX "authors" - nodes which perform block creation on T2 and can interact with T1 on behalf of T2 via proof of consensus.
- **Root publishing:** The periodic checkpointing on T1 of all transaction calls having occurred on T2, recorded in the form of Merkle Roots by T2.
- **Bridging funds:** The secure movement of fungible tokens (any ERC20 compliant token or EWT) between T1 (Energy Web Chain) and T2 (Energy Web X) via Lifting and Lowering.
- **Lifting:** Locking tokens sent to the T1 contract and authorizing the generation of an identical amount in the designated recipient's account on T2.
- **Lowering:** Unlocking tokens from the contract and transferring them to the specified T1 recipient, based on having received proof of their destruction on T2.

The AVN Tier 2, is an Aventus network parachain. A parachain is an application-specific data structure that is globally coherent and can be validated by the validators. The Aventus Network comprises both Layer 1 (L1) and Layer 2 (L2) technologies. These two layers communicate important transaction information

state changes etc. between blockchains. The main in-scope pallets are the following:

- **AvN:** This pallet provides functionality that is common for other AvN pallets such as handling off-chain workers, validations, and managing a list of validator accounts.
- **Ethereum-events**: This pallet provides functionality to get ethereum events.
- **eth-bridge:** This pallet handles Ethereum-related transactions. After T2 validates and executes a scheduled lower transaction, it uses eth-bridge to generate a lower proof. The eth-bridge will collect the ECDSA signature from all collators.
- **Summary:** This pallet handles the checkpointing to the Ethereum blockchain. Periodically, the merkle root is calculated.
- **avn-service:** The node service. Provides an API that is used by the offchain workers.
- **token-manager:** Schedules the execution of lower transactions from T2 to a recipient in T1 and emits event. Performs signed transfers. Saves the lower proof of eth-bridge after the scheduled lower transaction is completed and emits an event indicating that the tokens can be claimed from EVM chain.

2.2 In-Scope Components

The following five (5) pallets were in-scope for this assessment:

- *avn*
- *ethereum-events*
- *eth-bridge*
- *summary*
- *token-manager*

The following node service:

- *avn-service*

The following solidity smart contract:

- Energy-Bridge

The components are located at the following URLs:

- *https://github.com/energywebfoundation/energy-bridge/blob/a4494f215225e8c9cad39bc364c145024b96419d/contracts/EnergyBridge.sol*
- *https://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1/pallets/ethereum-events*
- *https://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1/pallets/avn*
- *https://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1/pallets/summary*
- *https://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1/pallets/eth-bridge*
- *https://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1/pallets/token-manager*
- *https://ghttps://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1/node/avn-service*

Component	Commit Identifier
https://github.com/energywebfoundation/energy-bridge	*a4494f215225e8c9cad39bc364c145024b96419d*
https://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1	*6b9595d93571eb3a227441cae8ad1c9c61f2968c (Release tag v5.1.1)*

A retesting phase was carried out on April 30th, 2024.

Component	Commit Identifier
https://github.com/energywebfoundation/energy-bridge	*308d3aed6912f38926fd4af61e5ef781a9fa48b5*
https://github.com/AventusProtocolFoundation/avn-parachain/tree/v5.1.1	*5260523a22b42ce011fb3b6b0fd1b51b9c1fd48d* *(And pull request* *https://github.com/AventusProtocolFoundation/avn-parachain/pull/392)*

3 Methodology

3.1 Assessment Methodology

Chaintroopers' methodology attempts to bridge the penetration testing and source code reviewing approaches in order to maximize the effectiveness of a security assessment.

Traditional pentesting or source code review can be done individually and can yield great results, but their effectiveness cannot be compared when both techniques are used in conjunction.

In our approach, the application is stress tested in all viable scenarios though utilizing penetration testing techniques with the intention to uncover as many vulnerabilities as possible. This is further enhanced by reviewing the source code in parallel to optimize this process.

When feasible our testing methodology embraces the Test-Driven Development process where our team develops security tests for faster identification and reproducibility of security vulnerabilities. In addition, this allows for easier understanding and mitigation by development teams.

Chaintroopers' security assessments are aligned with OWASP TOP10 and NIST guidance.

This approach, by bridging penetration testing and code review while bringing the security assessment in a format closer to engineering teams has proven to be highly effective not only in the identification of security vulnerabilities but also in their mitigation and this is what makes Chaintroopers' methodology so unique.

The specific phase of the assessment was only limited to source code review.

3.2 Smart Contracts

The testing methodology used is based on the empirical study "Defining Smart Contract Defects on Ethereum" by J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo and T. Chen, in IEEE Transactions on Software Engineering, and the security best practices as described in "Security Considerations" section of the solidity wiki.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope contract:

- Unchecked External Calls
- Strict Balance Equality
- Transaction State Dependency
- Hard Code Address
- Nested Call
- Unspecified Compiler Version
- Unused Statement
- Missing Return Statement
- Missing Reminder
- High Gas Consumption Function Type
- DoS Under External Influence
- Unmatched Type Assignment
- Re-entrancy
- Block Info Dependency
- Deprecated APIs
- Misleading Data Location
- Unmatched ERC-20 standard
- Missing Interrupter
- Greedy Contract
- High Gas Consumption Data Type

In Substrate Pallets, the list of vulnerabilities that are identified also includes:

- Static or Erroneously Calculated Weights
- Arithmetic Overflows
- Unvalidated Inputs
- Runtime Panic Conditions
- Missing Storage Deposit Charges
- Non-Transactional Dispatch Functions
- Unhandled Errors &Unclear Return Types
- Missing Origin Authorization Checks

4 Scoring System

4.1 CVSS

All issues identified as a result of Chaintroopers' security assessments are evaluated based on Common Vulnerability Scoring System version 3.1 (https://www.first.org/cvss/).

With the use of CVSS, taking into account a variety of factors a final score is produced ranging from 0 up to 10. The higher the number goes the more critical an issue is.

The following table helps provide a qualitative severity rating:

Rating	CVSS Score
None/Informational	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Issues reported in this document contain a CVSS Score section, this code is provided as an aid to help verify the logic of the team behind the evaluation of a said issue. A CVSS calculator can be found in the following URL:

https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

5 Identified Findings

5.1 Medium Severity Findings

5.1.1 Weight of "create_and_report" is not calculated in multiple functionalities at "summary", "eth-bridge" and "ethereum-events"

Description	MEDIUM

The team identified that the weight of the "*create_and_report_summary_offence*" functionality is not taken into account when the weight of other functionalities is calculated. In Rust pallets, when transactions are executed or data is stored on-chain, the activity changes the state of the chain and consumes blockchain resources. Because the resources available to a blockchain are limited, it's important to manage how operations on-chain consume them. The weights are introduced to estimate the time it takes to validate a block (the time it takes to execute the calls in the body of a block). By controlling the execution time that a block can consume, weights set limits on storage input and output and computation. If the weights are not correctly calculated, adversaries might attempt to overload the network with messages triggering these functionalities to stop the network from producing new blocks.

In the specific case it was found that the weight of the "*create_and_report_summary_offence*" functionality is not always taken into account. The specific functionality calls the "*create_offenders_identification*" function, which iterates over the *offenders_accounts,* moving them into the new scope.

```
File: /pallets/summary/src/offence.rs
067: pub fn create_offenders_identification<T: crate::Config>(
068:     offenders_accounts: &Vec<T::AccountId>,
069: ) -> Vec<IdentificationTuple<T>> {
070:     let offenders = offenders_accounts
071:         .into_iter()
072:         .filter_map(|id| <T as
SessionConfig>::ValidatorIdOf::convert(id.clone()))
073:         .filter_map(|id|
T::FullIdentificationOf::convert(id.clone()).map(|full_id| (id,
full_id)))
```

```
074:         .collect::<Vec<IdentificationTuple<T>>>();
075:     return offenders
076: }
077:
078: pub fn create_and_report_summary_offence<T: crate::Config>(
079:     reporter: &T::AccountId,
080:     offenders_accounts: &Vec<T::AccountId>,
081:     offence_type: SummaryOffenceType,
082: ) {
083:     let offenders =
create_offenders_identification::<T>(offenders_accounts);
084:
085:     if offenders.len() > 0 {
...
112:     }
113: }
```

However, it was found that calls such as the *"add_challenge()"*, does not take into account he *MAX_OFFENDERS* when the weight is calculated:

```
File: /pallets/summary/src/lib.rs
569:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::add_challenge())]
570:         #[pallet::call_index(6)]
571:         pub fn add_challenge(
572:             origin: OriginFor<T>,
573:             challenge: SummaryChallenge<T::AccountId>,
574:             validator: Validator<T::AuthorityId, T::AccountId>,
575:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
576:         ) -> DispatchResult {
...
590:             let offender = challenge.challengee.clone();
...
601:             create_and_report_summary_offence::<T>(
602:                 &validator.account_id,
603:                 &vec![offender],
604:                 challenge_type.expect("Already checked"),
605:             );
```

```
606:
...
617:     }
618:
```

A similar issue occurs in "*advance_slot":*

```
File: /pallets/summary/src/lib.rs
552:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::advance_slot_with_offence().max(
553:             <T as
Config>::WeightInfo::advance_slot_without_offence()
554:         ))]
555:         #[pallet::call_index(5)]
556:         pub fn advance_slot(
557:             origin: OriginFor<T>,
558:             validator: Validator<<T as avn::Config>::AuthorityId,
T::AccountId>,
559:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
560:         ) -> DispatchResult {
561:             ensure_none(origin)?;
562:
563:             Self::update_slot_number(validator)?;
564: …
```

```
File: /pallets/summary/src/lib.rs
803:         pub fn update_slot_number(
804:             validator: Validator<<T as avn::Config>::AuthorityId,
T::AccountId>,
805:         ) -> DispatchResult {
...
810:
Self::register_offence_if_no_summary_created_in_slot(&validator);
...
```

```
File: /pallets/summary/src/lib.rs
0994:         fn register_offence_if_no_summary_created_in_slot(
0995:             reporter: &Validator<T::AuthorityId, T::AccountId>,
0996:         ) {
...
1008:                 create_and_report_summary_offence::<T>(
1009:                     &reporter.account_id,
1010:                     &vec![current_slot_validator.clone()],
1011:                     SummaryOffenceType::NoSummaryCreated,
1012:                 );
1013:
```

It should be noted that in other cases, *the MAX_OFFENDERS* is taken into account for calculating the weight, such as the following:

```
File: /pallets/summary/src/lib.rs
467:
468:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::approve_root_with_end_voting(MAX_VALIDATOR_A
CCOUNT_IDS, MAX_OFFENDERS).max(
469:             <T as
Config>::WeightInfo::approve_root_without_end_voting(MAX_VALIDATOR_ACCOUN
T_IDS)
470:         ))]
471:         #[pallet::call_index(2)]
472:         pub fn approve_root(
473:             origin: OriginFor<T>,
474:             root_id: RootId<T::BlockNumber>,
475:             validator: Validator<<T as avn::Config>::AuthorityId,
T::AccountId>,
476:             approval_signature: ecdsa::Signature,
477:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
478:         ) -> DispatchResult {
...
```

```
File: /pallets/summary/src/lib.rs
532:
533:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::end_voting_period_with_rejected_valid_votes(
MAX_OFFENDERS).max(
534:             <T as
Config>::WeightInfo::end_voting_period_with_approved_invalid_votes(MAX_OF
FENDERS)
535:         ))]
536:         #[pallet::call_index(4)]
537:         pub fn end_voting_period(
538:             origin: OriginFor<T>,
539:             root_id: RootId<T::BlockNumber>,
540:             validator: Validator<<T as avn::Config>::AuthorityId,
T::AccountId>,
541:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
542:         ) -> DispatchResult {
...
```

The exact same issue affects the "*eth-bridge*" and "*ethereum-events*" pallets in the corresponding functionalities.

Impact

Adversaries might attempt to overload the network with messages triggering these functionalities, in order to exhaust the available resources and stop the network from producing new blocks, leading to a Denial-of-Service attack.

Recommendation

It is advisable to include the MAX_OFFENDERS parameter in the calculation of the weight for all the extrinsics that utilize the "*create_and_report_summary_offence*" functionality. The same mitigation should be applied in the corresponding functionalities at "*eth-bridge*" and *"ethereum-events*" pallets.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.2 *Weight of MAX_NUMBER_OF_ROOT_DATA_PER_RANGE is not calculated in "record_summary_calculation" at "summary"*

Description	MEDIUM

The team identified that the weight of root range is not taken into account when the weight of "*record_summary_calculation*" functionality is calculated. In Rust pallets, when transactions are executed or data is stored on-chain, the activity changes the state of the chain and consumes blockchain resources. Because the resources available to a blockchain are limited, it's important to manage how operations on-chain consume them. The weights are introduced to estimate the time it takes to validate a block (the time it takes to execute the calls in the body of a block). By controlling the execution time that a block can consume, weights set limits on storage input and output and computation. If the weights are not correctly calculated, adversaries might attempt to overload the network with messages triggering these functionalities to stop the network from producing new blocks.

In the specific case it was found that while the *MAX_NUMBER_OF_ROOT_DATA_PER_RANGE* was provided as a parameter (r) in "*record_summary_calculation*", it was not used:

```
File: /pallets/summary/src/default_weights.rs
87:   /// The range of component `v` is `[3, 10]`.
88:   /// The range of component `r` is `[1, 2]`.
89:   fn record_summary_calculation(v: u32, _r: u32, ) -> Weight {
90:         Weight::from_ref_time(62_440_000)
91:               // Standard Error: 38_000
92:
      .saturating_add(Weight::from_ref_time(242_000).saturating_mul(v as
u64))
93:               .saturating_add(T::DbWeight::get().reads(10))
94:               .saturating_add(T::DbWeight::get().writes(6))
95:   }
```

```
File: /avn-parachain-3.0.0/pallets/summary/src/lib.rs
393:         #[pallet::weight( <T as
```

```
pallet::Config>::WeightInfo::record_summary_calculation(
394:             MAX_VALIDATOR_ACCOUNT_IDS,
395:             MAX_NUMBER_OF_ROOT_DATA_PER_RANGE
396:         ))]
397:         #[pallet::call_index(1)]
398:         pub fn record_summary_calculation(
399:             origin: OriginFor<T>,
400:             new_block_number: T::BlockNumber,
401:             root_hash: H256,
402:             ingress_counter: IngressCounter,
403:             validator: Validator<<T as avn::Config>::AuthorityId,
T::AccountId>,
404:             _signature: <<T as avn::Config>::AuthorityId as
RuntimeAppPublic>::Signature,
405:         ) -> DispatchResult {
...
413:             let root_range =
RootRange::new(Self::get_next_block_to_process(), new_block_number);
414:             let root_id = RootId::new(root_range, ingress_counter);
...
417:
418:             ensure!(
419:
Self::summary_is_neither_pending_nor_approved(&root_id.range),
420:                 Error::<T>::SummaryPendingOrApproved
421:             );
...
```

File: /avn-parachain-3.0.0/pallets/summary/src/lib.rs

```
1350:         fn summary_is_neither_pending_nor_approved(root_range:
&RootRange<T::BlockNumber>) -> bool {
1351:             let has_been_approved =
1352:
<Roots<T>>::iter_prefix_values(root_range).any(|root| root.is_validated);
1353:             let is_pending =
<PendingApproval<T>>::contains_key(root_range);
1354:
```

```
1355:             return !is_pending && !has_been_approved
1356:         }
```

Impact

Adversaries might attempt to overload the network with messages triggering these functionalities, in order to exhaust the available resources and stop the network from producing new blocks, leading to a Denial of Service attack.

Recommendation

It is advisable to include the *MAX_NUMBER_OF_ROOT_DATA_PER_RANGE* parameter in the calculation of the weight for all the extrinsics that utilize the "*record_summary_calculation*" functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.3 Potential voting fraud by transaction sender in "add_corroboration" at "eth-bridge"

Description MEDIUM

The "*add_corroboration*" extrinsic collects from the authors the corroborations regarding the status of a transaction on the Ethereum-based contract. All authors except the transaction sender, whose action is to be validated, are expected to submit a corroboration.

It was identified that the implementation does not prevent the transaction sender from submitting valid corroboration, thus affecting the outcome of the voting. As it can be seen in the code snippets below, the implementation correctly calculates the corroborator quorum based on the number of authors minus the transaction sender, however it does not check if a corroboration is submitted by the transaction sender.

```
File: pallets/eth-bridge/src/lib.rs
400:         pub fn add_corroboration(
...
440:                     if
util::has_enough_corroborations::<T>(matching_corroborations.len()) {
File: pallets/eth-bridge/src/util.rs
10: pub fn has_enough_corroborations<T: Config>(corroborations: usize) ->
bool {
11:     // the sender cannot corroborate their own transaction
12:     let num_authors_excluding_sender = AVN::<T>::validators().len()
as u32 - 1;
```

Impact

A malicious transaction sender (author) may submit a valid (properly signed) corroboration to influence the result of the voting for their own benefit, as the consensus of corroborations is required to determine whether the transaction sender have properly submitted the transaction to the Ethereum-based contract.

Recommendation

It is recommended that the "*add_corroboration*" extrinsic checks that the submitted corroboration is not signed by the transaction sender (e.g. *tx.tx_data.sender != author.account_id*) before recording it.

CVSS Score

AV:N/AC:L/PR:H/UI:N/S:U/C:N/I:H/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.4 The size of the HTTP body in "send_main", "view_main" and "tx_query_main" functionalities is not checked at "avn-service"

Description	MEDIUM

The team identified that the *"send_main", "view_main"* and "*tx_query_main*" functionalities do not check the size of the HTTP body, before processing it. In general, the specific service is an interface with Ethereum blockchain technology built upon an asynchronous web server framework (Tide). As a result, the service needs to query and parse the transaction blocks.

In the specific case, the code currently decodes Ethereum transaction data directly from the request body without apparent validation, such as checking the size limits of the incoming request to avoid denial-of-service (DoS) attacks caused by processing large inputs. The issue exists at the following locations:

```
File: /avn-parachain-5.1.1/node/avn-service/src/lib.rs
264: #[tokio::main]
265: async fn send_main<Block: BlockT, ClientT>(
266:     mut req: tide::Request<Arc<Config<Block, ClientT>>>,
267: ) -> Result<String, TideError>
268: where
269:     ClientT: BlockBackend<Block> + UsageProvider<Block> + Send +
Sync + 'static,
270: {
271:     log::info!("⛓  avn-service: send Request");
272:     let post_body = req.body_bytes().await?;
273:     let send_request = &EthTransaction::decode(&mut &post_body[..])
274:         .map_err(|e| server_error(format!("Error decoding eth
transaction data: {:?}", e)))?;
```

```
File: /avn-parachain-5.1.1/node/avn-service/src/lib.rs
321: #[tokio::main]
322: async fn view_main<Block: BlockT, ClientT>(
323:     mut req: tide::Request<Arc<Config<Block, ClientT>>>,
324: ) -> Result<String, TideError>
325: where
```

```
326:     ClientT: BlockBackend<Block> + UsageProvider<Block> + Send +
Sync + 'static,
327: {
328:     log::info!("⛓  avn-service: view Request");
329:     let post_body = req.body_bytes().await?;
330:     let view_request = &EthTransaction::decode(&mut &post_body[..])
331:         .map_err(|e| server_error(format!("Error decoding eth
transaction data: {:?}", e)))?;
```

```
File: /avn-parachain-5.1.1/node/avn-service/src/lib.rs
354: #[tokio::main]
355: async fn tx_query_main<Block: BlockT, ClientT>(
356:     mut req: tide::Request<Arc<Config<Block, ClientT>>>,
357: ) -> Result<String, TideError>
358: where
359:     ClientT: BlockBackend<Block> + UsageProvider<Block> + Send +
Sync + 'static,
360: {
361:     log::info!("⛓  avn-service: query Request.");
362:     let post_body = req.body_bytes().await?;
363:
364:     let request = &EthTransaction::decode(&mut &post_body[..])
365:         .map_err(|e| server_error(format!("Error decoding eth
transaction data: {:?}", e)))?;
```

Impact

An adversary who is able to trick the node service to parse an HTTP response body of significant size, will be able to force the service to consume all the available processing resources, leading to a denial of service (DoS) attack.

Recommendation

Limiting the size of the incoming request body before processing it is crucial to protect your application from various attacks, including denial-of-service (DoS) attacks. In Rust, especially in asynchronous web servers, it is possible to limit

the size of the body by checking its size before attempting to decode it into an *EthTransaction*:

```
const MAX_BODY_SIZE: usize = 10_000; // for example, 10 KB
let post_body = req.body_bytes().await?;
    // Check the size of the post_body before proceeding
if post_body.len() > MAX_BODY_SIZE {
     return Err(server_error("Request body too large".to_string()));
}
let view_request = &EthTransaction::decode(&mut &post_body[..])
        .map_err(|e| server_error(format!("Error decoding eth transaction
data: {:?}", e)))?;
```

CVSS Score

AV:N/AC:L/PR:H/UI:N/S:U/C:N/I:N/A:H/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.5 Outdated and Vulnerable dependencies in-use at "Cargo.lock"

Description	MEDIUM

The team identified multiple dependencies in "*avn-parachain*" and the pallets, that were outdated and vulnerable. Outdated and vulnerable dependencies are open-source or proprietary components, in the form of libraries or frameworks, that contain known software vulnerabilities or are no longer maintained. Once a vulnerable component is discovered by adversaries, applications using this component can be targeted and exploited.

Outdated and vulnerable dependencies found at "*Cargo.lock*":

```
Crate: ed25519-dalek
Version: 1.0.1
Title: Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
Date: 2022-06-11
ID: RUSTSEC-2022-0093
URL: https://rustsec.org/advisories/RUSTSEC-2022-0093
Solution: Upgrade to >=2
```

```
Crate: h2
Version: 0.3.15
Title: Resource exhaustion vulnerability in h2 may lead to Denial of Service (DoS)
Date: 2023-04-14
ID: RUSTSEC-2023-0034
URL: https://rustsec.org/advisories/RUSTSEC-2023-0034
Solution: Upgrade to >=0.3.17
```

```
Crate: openssl
Version: 0.10.45
Title: `openssl` `X509VerifyParamRef::set_host` buffer over-read
Date: 2023-06-20
```

```
ID: RUSTSEC-2023-0044
URL: https://rustsec.org/advisories/RUSTSEC-2023-0044
Title:     `openssl` `SubjectAlternativeName` and
`ExtendedKeyUsage::other` allow arbitrary file read
Date:      2023-03-24
ID:        RUSTSEC-2023-0023
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0023
Title:     `openssl` `X509NameBuilder::build` returned object is not
thread safe
Date:      2023-03-24
ID:        RUSTSEC-2023-0022
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0022
Title:     `openssl` `X509Extension::new` and `X509Extension::new_nid`
null pointer dereference
Date:      2023-03-24
ID:        RUSTSEC-2023-0024
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0024
Solution:  Upgrade to >=0.10.48
```

```
Crate:     owning_ref
Version:   0.4.1
Title:     Multiple soundness issues in `owning_ref`
Date:      2022-01-26
ID:        RUSTSEC-2022-0040
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0040
Solution:  No fixed upgrade is available!
```

```
Crate:     remove_dir_all
Version:   0.5.3
Title:     Race Condition Enabling Link Following and Time-of-check Time-
of-use (TOCTOU)
Date:      2023-02-24
ID:        RUSTSEC-2023-0018
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0018
Solution:  Upgrade to >=0.8.0
```

```
Crate:     time
Version:   0.1.45
Title:     Potential segfault in the time crate
Date:      2020-11-18
ID:        RUSTSEC-2020-0071
URL:       https://rustsec.org/advisories/RUSTSEC-2020-0071
Severity:  6.2 (medium)
Solution:  Upgrade to >=0.2.23
```

```
Crate:     tokio
Version:   1.23.0
Title:     reject_remote_clients Configuration corruption
Date:      2023-01-04
ID:        RUSTSEC-2023-0001
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0001
Solution:  Upgrade to >=1.18.4, <1.19.0 OR >=1.20.3, <1.21.0 OR >=1.23.1
```

```
Crate:     webpki
Version:   0.22.0
Title:     webpki: CPU denial of service in certificate path building
Date:      2023-08-22
ID:        RUSTSEC-2023-0052
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0052
Severity:  7.5 (high)
Solution:  Upgrade to >=0.22.2
```

And other dependencies that were unmaintained, consider using cargo-audit to check for outdated and vulnerable dependencies.

Impact

When a vulnerable 3rd-party component is discovered by adversaries, all applications that utilize this component can be identified and targeted. Even if

the vulnerability might initially look like a small weakness in the application codebase, in some cases it can lead to a full system compromise. Such a breach can have a seriously affect the users' data and potentially lead to lost revenue and reputational damage the organization.

Recommendation

It is recommended to update the related dependencies to the latest version.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.6 Weight of ConfirmationsLimit is not calculated in "add_confirmation" at "eth-bridge"

Description	MEDIUM

The team identified that the weight of confirmations is not taken into account when the weight of "*add_confirmation*" functionality is calculated. In Rust pallets, when transactions are executed or data is stored on-chain, the activity changes the state of the chain and consumes blockchain resources. Because the resources available to a blockchain are limited, it's important to manage how operations on-chain consume them. The weights are introduced to estimate the time it takes to validate a block (the time it takes to execute the calls in the body of a block). By controlling the execution time that a block can consume, weights set limits on storage input and output and computation. If the weights are not correctly calculated, adversaries might attempt to overload the network with messages triggering these functionalities to stop the network from producing new blocks.

The issue exists at "*encode_confirmations*" functionality, which iterates over the provided confirmations:

```
File: /avn-parachain-5.1.1/pallets/eth-bridge/src/eth.rs
105: pub fn encode_confirmations(
106:     confirmations: &BoundedVec<ecdsa::Signature,
ConfirmationsLimit>,
107: ) -> Vec<u8> {
108:     let mut concatenated_confirmations = Vec::new();
109:     for conf in confirmations {
110:         concatenated_confirmations.extend_from_slice(conf.as_ref());
111:     }
112:     concatenated_confirmations
113: }
114:
```

Then, this functionality is used by other functions including the "*generate_encoded_lower_proof*":

```
File: /avn-parachain-5.1.1/pallets/eth-bridge/src/eth.rs
137: pub fn generate_encoded_lower_proof<T: Config>(
138:     lower_req: &LowerProofRequestData,
139:     confirmations: BoundedVec<ecdsa::Signature, ConfirmationsLimit>,
140: ) -> Vec<u8> {
141:     let concatenated_confirmations =
encode_confirmations(&confirmations);
142:     let mut compact_lower_data = Vec::new();
143:
compact_lower_data.extend_from_slice(&lower_req.params.to_vec());
144:
compact_lower_data.extend_from_slice(&concatenated_confirmations);
145:
146:     return compact_lower_data
147: }
```

And then the "*complete_lower_proof_request*" calls the "*generate_encoded_lower_proof*":

```
File: /avn-parachain-5.1.1/pallets/eth-bridge/src/request.rs
101: pub fn complete_lower_proof_request<T: Config>(
102:     lower_req: &LowerProofRequestData,
103:     confirmations: BoundedVec<ecdsa::Signature, ConfirmationsLimit>,
104: ) -> Result<(), Error<T>> {
105:     let lower_proof =
eth::generate_encoded_lower_proof::<T>(lower_req, confirmations);
106:     let result =
T::BridgeInterfaceNotification::process_lower_proof_result(
107:         lower_req.lower_id,
108:         lower_req.caller_id.clone().into(),
109:         Ok(lower_proof),
110:     );
111:
....
119:     Ok(())
120: }
```

Finally, the "*add_confirmation*" calls the "*complete_lower_proof_request*", but does not take into account the max confirmations during the weight calculation:

```
File: /avn-parachain-5.1.1/pallets/eth-bridge/src/lib.rs
313:         #[pallet::call_index(2)]
314:         #[pallet::weight(<T as
Config>::WeightInfo::add_confirmation())]
315:         pub fn add_confirmation(
316:             origin: OriginFor<T>,
317:             request_id: u32,
318:             confirmation: ecdsa::Signature,
319:             author: Author<T>,
320:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
321:         ) -> DispatchResultWithPostInfo {
322:             ensure_none(origin)?;
323:
...
345:                 req.confirmation
346:                     .confirmations
347:                     .try_push(confirmation)
348:                     .map_err(|_|
Error::<T>::ExceedsConfirmationLimit)?;
349:
350:                 match req.request {
351:                     Request::LowerProof(lower_req) if
request::has_enough_confirmations(&req) =>
352:                         request::complete_lower_proof_request::<T>(
353:                             &lower_req,
354:                             req.confirmation.confirmations,
355:                         )?,
356:                     _ => {
357:                         save_active_request_to_storage(req);
358:                     },
359:                 }
360:             }
361:
362:             Ok(().into())
363:         }
```

Impact

Adversaries might attempt to overload the network with messages triggering these functionalities, in order to exhaust the available resources and stop the network from producing new blocks, leading to a Denial-of-Service attack.

Recommendation

It is advisable to include the *ConfirmationsLimit* parameter in the calculation of the weight for all the extrinsics that utilize the "*add_confirmation*" functionality.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2 Low Severity Findings

5.2.1 Potential gas griefing attack in "_releaseFunds()" at "EnergyBridge.sol"

Description	LOW

The team identified that the "*_releaseFunds()*" functionality, is affected by a gas griefing vulnerability. Gas griefing refers to a type of attack or malicious behavior in the Ethereum network (or other blockchain platforms that use gas for transaction fees) where an attacker causes a victim to expend more gas than necessary. This can happen in several contexts, especially in smart contract interactions where the attacker can manipulate the circumstances of a transaction to increase the gas cost for the victim.

In the specific case, it was found that the "_releaseFunds()" functionality is using the low level call() function even if the returned data is not required, and unnecessarily exposes the contract to gas griefing attacks from huge returned data payload. In general, the following call:

```
(bool success, ) = payable(recipient).call{value: amount}("");
if (!success) revert PaymentFailed();
```

is the same as the following call which assigns the return data to the data parameter:

```
(bool success, bytes memory data) = payable(recipient).call{value: amount}("");
if (!success) revert PaymentFailed();
```

Memory arrays use up quadratic amount of gas after 724 bytes, so a carefully chosen return data size can grief the caller. Even if the variable data is not used, it is still copied to memory.

File: EnergyBridge.sol

```
507:   function _releaseFunds(address token, uint256 amount, address
recipient)
508:     private
509:   {
510:     if (token == PSEUDO_EWT_ADDRESS) {
511:       (bool success, ) = payable(recipient).call{value: amount}("");
512:       if (!success) revert PaymentFailed();
```

Impact

The recipient of "*claimLower*" and "*legacyLower*" can return a large memory array that will consume a lot of gas, overcharging the caller or even making the calls to fail. Regarding the second case, recipients might observe a pending transaction that interacts with a contract in a beneficial way and submit their transaction with a higher gas fee to have it processed first, forcing the original transaction to fail or to be processed under less favorable conditions (front-running) Since in most cases the "*claimLower()*" will be called by the recipient, the issue is marked as LOW.

Recommendation

It is advisable to limit the gas that can be used as part of the external call, or to replace the call with the *ExcessivelySafeCall*.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.2 *Insufficient Information for event emission in "claimLower()" at "EnergyBridge.sol"*

Description **LOW**

It was found that the "*claimLower*()" function in the EnergyBridge.sol smart contract emits events that only include the "*lowerId*". While it is still possible for a monitoring solution to correlate this identifier with a previously submitted proof, it is impossible to use it without the rest of the proof to retrieve the transaction details, such as the "token", the "recipient", the "t2PubKey", and the "amount". The only available "view" functionality in the contract is the "*checkLower*" which requires the whole proof to be supplied as input parameter.

```
File: EnergyBridge.sol
345:   function claimLower(bytes calldata proof)
346:     onlyWhenLoweringEnabled
347:     lock
348:     external
349:   {
..
371:     emit LogLowerClaimed(lowerId);
372:   }
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle in order to provide credibility and confidence in the system. In the specific case, when the "*claimLower*()" function is called, insufficient information is emitted.

Recommendation

It is advisable to adjust the event emission in the "*claimLower*()" function to include "*token*", "*recipient*", "*t2PubKey*", and "amount", or to create a similar view functionality that requires only the "*tokenId*" as an input parameter.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:R/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.3 Insecure error handling exposes sensitive information in "raw_public_keys" at "avn-service"

Description	LOW

It was identified that the "*avn-service*" node may leak sensitive information in server error messages. Specifically, the "*raw_public_keys*" function returns a server error response with a message that includes the error returned by a call to "*fs::read_dir*", as shown at line 53 in the code snippet below:

```
File: /avn-parachain-5.1.1/node/avn-service/src/keystore_utils.rs
45: pub fn raw_public_keys(
46:     key_type: KeyTypeId,
47:     keystore_path: &PathBuf,
48: ) -> Result<Vec<Vec<u8>>, TideError> {
49:     let mut public_keys: Vec<Vec<u8>> = vec![];
50:
51:     for entry in fs::read_dir(keystore_path)? {
52:         let entry = entry
53:             .map_err(|e| server_error(format!("Error getting files
from directory: {:?}", e)))?;
54:         let path = entry.path();
55:
```

The error of a potential failure of *fs::read_dir* may include sensitive system information, including local filesystem paths.

Impact

An adversary may use the information obtained from error messages to carry out further attacks. In this particular case, the leaked information might prove useful to an attacker in the exploitation of a potential future path traversal or local file inclusion vulnerability.

Recommendation

It is recommended to ensure that error messages do not leak sensitive information. In this particular case, the server error message should not include information from the returned error value, but rather be a generic server error

message informing about the failure. The returned error value may be logged in application's logs for further inspection.

CVSS Score
AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.4 Insecure error handling exposes sensitive information in "key_phrase_by_type" at "avn-service"

Description	LOW

It was identified that the "*avn-service*" node may leak sensitive information in server error messages. Specifically, the "*key_phrase_by_type*" function returns a server error response with a message that includes the error returned by a call to File::open, as shown at line 85 in the code snippet below:

```
File: /avn-parachain-5.1.1/node/avn-service/src/keystore_utils.rs
76: fn key_phrase_by_type(
77:     eth_address: &[u8],
78:     key_type: KeyTypeId,
79:     keystore_path: &PathBuf,
80: ) -> Result<String, TideError> {
81:     let mut path = keystore_path.clone();
82:     path.push(hex::encode(key_type.0) +
hex::encode(eth_address).as_str());
83:
84:     if path.exists() {
85:         let file = File::open(path)
86:             .map_err(|e| server_error(format!("Error opening EthKey
file: {:?}", e)))?;
87:         serde_json::from_reader(&file).map_err(Into::into)
88:     } else {
89:         Err(server_error(format!(
90:             "Keystore file for EthKey: {:?} not found",
91:             ETHEREUM_SIGNING_KEY
92:         )))?
93:     }
94: }
95:
```

The error of a potential failure of File::open may include sensitive system information, including local filesystem paths.

Impact

An adversary may use the information obtained from error messages to carry out further attacks. In this particular case, the leaked information might prove useful to an attacker in the exploitation of a potential future path traversal or local file inclusion vulnerability.

Recommendation

It is recommended to ensure that error messages do not leak sensitive information. In this particular case, the server error message should not include information from the returned error value, but rather be a generic server error message informing about the failure. The returned error value may be logged in application's logs for further inspection.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.5 No max range limitation in "/roothash/:from_block/:to_block" API call at "avn-service"

Description	LOW

The team identified that the "*/roothash/:from_block/:to_block*" API call does not validate that the provided input parameters are within an acceptable range. If the range between *"from_block"* and *"to_block"* can be very large, fetching extrinsics for a wide range and calculating the root hash could be resource intensive.

The issue exists at the following location:

```
File: /avn-parachain-5.1.1/node/avn-service/src/lib.rs
458:     app.at("/roothash/:from_block/:to_block").get(
459:         |req: tide::Request<Arc<Config<Block, ClientT>>>| async move
{
460:             log::info!("⛓  avn-service: roothash");
461:             // We cannot use a number bigger than a u32, but with
block times of 12 sec it would
462:             // take of few hundred years before we reach it.
463:             let from_block_number: u32 =
req.param("from_block")?.parse()?;
464:             let to_block_number: u32 =
req.param("to_block")?.parse()?;
465:
466:             let extrinsics_start_time = Instant::now();
467:
468:             let extrinsics =
469:                 get_extrinsics::<Block, ClientT>(&req,
from_block_number, to_block_number)?;
470:             let extrinsics_duration =
extrinsics_start_time.elapsed();
471:             log::info!(
472:                 "🕐  get_extrinsics on block range [{:?}, {:?}]
time: {:?}",
473:                 from_block_number,
474:                 to_block_number,
475:                 extrinsics_duration
```

```
476:             );
477:
478:             if extrinsics.len() > 0 {
479:                 let root_hash_start_time = Instant::now();
480:                 let root_hash = generate_tree_root(extrinsics)?;
481:                 let root_hash_duration =
root_hash_start_time.elapsed();
482:                 log::info!(
483:                     "🕛  generate_tree_root on block range [{:?},
{:?}] time: {:?}",
484:                     from_block_number,
485:                     to_block_number,
486:                     root_hash_duration
487:                 );
488:
489:                 return Ok(hex::encode(root_hash))
490:             }
491:
492:             // the tree is empty
493:             Ok(hex::encode([0; 32]))
494:         },
495:     );
```

Impact

An adversary, could provide a very large range in order to force the service to exhaust the available resources, leading to a Denial of Service (DoS) attack.

Recommendation

It is recommended to introduce a validation control to ensure that these numbers are within a sensible range (e.g., *from_block* is less than *to_block*, and both are within the current blockchain height). For example:

```
let max_allowed_range = 10_000; // Example limit
if to_block_number - from_block_number > max_allowed_range {
    return Err(tide::Error::from_str(
        tide::StatusCode::BadRequest,
        format!("Block range cannot exceed {}", max_allowed_range),
```

```
    ));
}
```

CVSS Score

AV:N/AC:L/PR:H/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.6 Owner can renounce Ownership at "EnergyBridge.sol"

Description	LOW

The team identified that the owner of the contract can renounce its ownership. The contract's owner is typically the account that deploys it, allowing them to execute certain exclusive actions. In the case of the EnergyBridge.sol smart contract, it inherits the business logic from "*OwnableUpgradeable*" contract, which includes a "*renounceOwnership()*" function that can be potentially dangerous in certain contexts.

The "*renounceOwnership()*" function in smart contracts, particularly those following the *"OpenZeppelin Ownable"* contract pattern in Solidity, is designed to transfer the ownership of the contract to a zero address, effectively making it ownerless. This action is irreversible: once ownership is renounced, there is no way to regain control over certain aspects of the contract that are restricted to the owner. This can have both intended and unintended consequences.

Impact

Renouncing ownership means that any functionality or administrative privileges restricted to the owner can no longer be executed. If the contract requires future updates, bug fixes, or administrative actions (like enabling or disabling lift and lower operations), these will become impossible, potentially leading to a loss of functionality or, in the worst case, making the contract vulnerable to exploits that cannot be addressed. Moreover, the action of renouncing ownership is irreversible. Unlike transferring ownership to another account, which can be seen as a reversible action if the new owner decides to transfer it back, renouncing ownership does not allow for any form of recovery or undoing. In the specific case, if an owner accidentally or deliberately renounced the ownership, all functions marked with '*onlyOwner*' modifier will no longer work.

Recommendation

It is advised to override the "*renounceOwnership()*" function to revert in order to prevent this issue. For instance:

```
function renounceOwnership() public override onlyOwner {
   revert("Not allowed to renounce OwnerShip");
}
```

CVSS Score

AV:N/AC:L/PR:H/UI:R/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.7 Return value of "ecrecover()" is not checked at "EnergyBridge.sol"

Description	LOW

The team identified that the return value of "*ecrecover()*" functionality is not checked (CWE-252: Unchecked Return Value). Many functions will return some value about the success or failure of their actions. This will alert the program whether or not to handle any errors caused by that function.

Ecrecover is an Ethereum-specific function that allows a user to recover the address associated with a signed message. This function is primarily used for authentication, such as when logging into an account or approving a transaction. In general, "*ecrecover()*" takes four parameters: a message hash (bytes32 hash), the recovery ID (uint8 v), and the two 32-byte components of the signature (bytes32 r and bytes32 s), and use them to calculate the public key associated with the signature. Then, it returns the Ethereum address derived from this public key. If the signature is invalid, it returns zero, indicating an error. However, it was found that the contract does not check for this error flow and use the return value to retrieve the corresponding identifier from the "*t1AddressToId*" mapping.

```
File: /energy-bridge-master/contracts/EnergyBridge.sol
474:   function _verifyConfirmations(bytes32 msgHash, bytes memory
confirmations)
475:     private
476:   {
...
484:     uint256 validConfirmations;
...
489:     bool[] memory confirmed = new bool[](nextCollatorId);
...
491:     for (uint256 i; i < numConfirmations;) {
...
503:       if (...) {
...
506:       } else {
507:         id = t1AddressToId[ecrecover(ethSignedPrefixMsgHash, v, r,
s)];
508:
```

```
509:          if (!isActiveCollator[id]) {
...
513:              unchecked {
514:               numActiveCollators++;
515:               validConfirmations++;
516:              }
...
521:              confirmed[id] = true;
...
523:          } else if (!confirmed[id]) {
524:            unchecked { validConfirmations++; }
...
526:            confirmed[id] = true;
527:          }
528:        }
...
531:      }
...
534:    }
```

Impact

An adversary, who is able to trick the admin user into registering a Collator with the zero address, would be able to provide any invalid signature and protential circumvent the confirmation control.

Recommendation

It is advisable to validate that the return value of the specific functionality is not zero. Furthermore, it is recommended to use the *OpenZeppelin's ECDSA* Helper Library. *OpenZeppelin* offers an ECDSA helper library that wraps around *ecrecover*, validates the result and mitigates any potential signature malleability issue. Using this library can simplify the process of signature verification while enhancing security (*https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol)*

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.8 No max-time limitation in asynchronous calls to external services in "web3_utils" at "avn-service"

Description	LOW

It was identified that the "*avn-service*" pallet makes asynchronous calls to external blockchain services using the "*web3_utils*" (e.g., *"get_current_block_number"*, *"get_tx_call_data"*, *"get_tx_receipt"*, *"send_raw_transaction"*). If these services are slow to respond or are unavailable, it could tie up server resources, leading to a denial-of-service condition.

The issue exists at the following location:

```
File: /avn-parachain-5.1.1/node/avn-service/src/web3_utils.rs
159: pub async fn get_current_block_number(web3: &Web3<Http>) ->
anyhow::Result<u64> {
160:     Ok(web3.eth().block_number().await?.as_u64())
161: }
162:
163: pub async fn get_tx_receipt(
164:     web3: &Web3<Http>,
165:     tx_hash: ethereum_types::H256,
166: ) -> anyhow::Result<Option<TransactionReceipt>> {
167:
Ok(web3.eth().transaction_receipt(web3::types::H256(tx_hash.0)).await?)
168: }
169:
170: pub async fn get_tx_call_data(
171:     web3: &Web3<Http>,
172:     tx_hash: ethereum_types::H256,
173: ) -> anyhow::Result<Option<Transaction>> {
174:     Ok(web3
175:         .eth()
176:
.transaction(web3::types::TransactionId::Hash(web3::types::H256(tx_hash.0
)))
177:         .await?)
178: }
179:
180: pub async fn send_raw_transaction(
```

```
181:     web3: &Web3<Http>,
182:     tx: Bytes,
183: ) -> anyhow::Result<web3::types::H256> {
184:     Ok(web3
185:         .eth()
186:         .send_raw_transaction(tx)
187:         .await
188:         .with_context(|| format!("Error while sending raw
transaction to Ethereum"))?)
189: }
```

These functionalities are then called by other parts of the service:

```
File: /avn-parachain-5.1.1/node/avn-service/src/lib.rs
354: #[tokio::main]
355: async fn tx_query_main<Block: BlockT, ClientT>(
356:     mut req: tide::Request<Arc<Config<Block, ClientT>>>,
357: ) -> Result<String, TideError>
358: where
359:     ClientT: BlockBackend<Block> + UsageProvider<Block> + Send +
Sync + 'static,
360: {
...
378:         let current_block_number =
web3_utils::get_current_block_number(&web3)
379:             .await
380:             .map_err(|e| server_error(format!("Error getting block
number: {:?}", e)))?;
```

Impact

An adversary, who is able to make these services slow to respond or unavailable, could tie up server resources, leading to a denial-of-service condition.

Recommendation

This issue can be mitigated by introducing code-level strategies aimed at reducing the impact of delays or failures in the external service.

For example:

```
let current_block_number_result = timeout(
    Duration::from_secs(5),
    web3_utils::get_current_block_number(&web3)
).await;
let current_block_number = match current_block_number_result {
    Ok(Ok(block_number)) => block_number,
    Ok(Err(e)) => return Err(server_error(format!("Error getting block number: {:?}", e))),
    Err(_) => return Err(server_error("Timeout getting block number".to_string())),
};
```

Finally, the mutex should be handled properly in case of an error, to avoid deadlocks.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.9 Missing event emit in "settle_transfer" at "token-manager"

Description	LOW

The team identified that the "*settle_transfer*()" function does not emit an event for a security-critical operation (CWE-778: Insufficient Logging). A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain.

The issue exists in the following location:

```
File: /avn-parachain-5.1.1/pallets/token-manager/src/lib.rs
567: impl<T: Config> Pallet<T> {
568:     fn settle_transfer(
569:         token_id: &T::TokenId,
570:         from: &T::AccountId,
571:         to: &T::AccountId,
572:         amount: &T::TokenBalance,
573:     ) -> DispatchResult {
574:         if *token_id == Self::avt_token_contract().into() {
575:             // First convert TokenBalance to u128
576:             let amount_u128 = TryInto::<u128>::try_into(*amount)
577:                 .map_err(|_|
Error::<T>::ErrorConvertingTokenBalance)?;
578:             // Then convert to Balance
579:             let transfer_amount = <BalanceOf<T> as
TryFrom<u128>>::try_from(amount_u128)
580:                 .or_else(|_error|
Err(Error::<T>::ErrorConvertingToBalance))?;
581:
582:             <T as pallet::Config>::Currency::transfer(
583:                 from,
584:                 to,
585:                 transfer_amount,
586:                 ExistenceRequirement::KeepAlive,
587:             )?;
588:         } else {
```

```
...
610:          }
611:
612:          <Nonces<T>>::mutate(from, |n| *n += 1);
613:
614:          Ok(())
615:      }
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle in order to provide credibility and confidence in the system.

In the specific case, when the "*settle_transfer()*" function is called for an *"avt_token_contract",* no event will be emitted.

Recommendation

It is advisable to emit an event. For example:

```
        Self::deposit_event(Event::<T>::TokenTransferred {
                token_id: token_id.clone(),
                sender: from.clone(),
                recipient: to.clone(),
                token_balance: amount.clone(),
            });
```

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3 Informational Findings

5.3.1 Use of "ecrecover()" at EnergyBridge.sol

Description	INFO

The team identified that EnergyBridge.sol directly calls the EVM precompiled "*ecrecover*()" functionality. The "*ecrecover*()" functionality is utilized for verifying transactions. However, the native EVM precompile "*ecrecover*" is vulnerable to signature malleability due to non-unique s and v values, potentially resulting in replay attacks.

While this is not exploitable in the current implementation because it verifies that s and v are within valid bounds to avoid signature malleability at line 503, this may become an issue if used elsewhere.

```
File: EnergyBridge.sol
503:        if (v != 27 && v != 28 || uint256(s) >
0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
504:          unchecked { i++; }
505:          continue;
506:        } else {
507:          id = t1AddressToId[ecrecover(ethSignedPrefixMsgHash, v, r,
s)];
508:
```

Impact

Since the issue is not exploitable in the current implementation because the contract verifies that both s and v are within valid bounds, it is marked as INFORMATIONAL.

Recommendation

It is advisable to use "*OpenZeppelin's ECDSA*" library which prevents signature malleability instead of directly calling the built-in "*ecrecover()*" function.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N/E:U/RL:X/RC:R/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.2 Removal of Merkle tree root hash is not possible at "EnergyBridge.sol"

Description	INFO

The team identified that the contract does not provide any functionality to remove an approved and published *merkle* root hash that has not been fulfilled yet, in case of a security incident (CWE-671: Lack of Administrator Control over Security).

While it is not expected that an approved action would need to be reverted as part of the normal contract's business logic, this operation might be required as part of an incident response case. For example, in case that an adversary has managed to publish an incorrect *merkle* tree root hash to the contract, the owner or the collators would be able to active the Emergency Stop pattern (using the "*toggleLowering*" functionality) and halt the "*lower*" contract functionality. However, it will not be possible to remove the malicious *merkle* tree root hash and successfully recover from the attack, without having to upgrade the contract.

```
File: /energy-bridge-master/contracts/EnergyBridge.sol
298:   function publishRoot(bytes32 rootHash, uint256 t2TransactionId,
bytes calldata confirmations)
299:     onlyWhenCollatorFunctionsAreEnabled
300:     external
301:   {
302:     _verifyConfirmations(_toConfirmationHash(rootHash,
t2TransactionId), confirmations);
303:     _storeT2TransactionId(t2TransactionId);
304:     if (isPublishedRootHash[rootHash]) revert
RootHashAlreadyPublished();
305:     isPublishedRootHash[rootHash] = true;
306:     emit LogRootPublished(rootHash, t2TransactionId);
307:   }
```

Impact

In case of an attack, in which the adversary has managed to publish a malicious *merkle* tree root hash in the contract, that has not been fulfilled yet (e.g. the

owner activated the "*toggleLowering*" functionality), the owner or the collators will not be able to remove the injected root hash and recover from the attack without upgrading the contract.

This is an added layer of security. As a result, the issue is marked as INFORMATIONAL.

Recommendation

It is advisable to add an extra functionality that would allow the owner to remove a *merkle* tree root hash that has not been fulfilled yet.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.3 Costly length comparison at "ethereum-events" and "summary"

Description	INFO

The team identified that the application is using *".len()"* and compares the return value with zero (0) to check if an array is empty. However, this is a more costly operation than using the faster *".is_empty()"* function call.

The issue exists at the following locations:

```
File: avn-parachain-5.1.1/pallets/ethereum-events/src/lib.rs
1342:     fn compute_result(
1343:         block_number: T::BlockNumber,
1344:         response_body: Result<Vec<u8>, DispatchError>,
1345:         event_id: &EthEventId,
1346:         validator_account_id: &T::AccountId,
1347:     ) -> EthEventCheckResult<T::BlockNumber, T::AccountId> {
...
1377:         if response_data_object.len() == 0 {
1378:             log::error!("❌ Response data json is empty");
1379:             return invalid_result
1380:         };
...
```

```
File: avn-parachain-5.1.1/pallets/ethereum-events/src/event_parser.rs
123: fn get_data(event: &JsonValue)
...
130:     let bytes = hex_to_bytes(data)?;
131:
132:     if bytes.len() > 0 {
133:         return Ok(Some(bytes))
134:     }
...
```

```
File: /avn-parachain-5.1.1/pallets/ethereum-events/src/offence.rs
73: pub fn create_and_report_invalid_log_offence<T: crate::Config>(
```

```
74:     reporter: &T::AccountId,
75:     offenders_accounts: &Vec<T::AccountId>,
76:     offence_type: EthereumLogOffenceType,
77: ) {
78:     let offenders =
create_offenders_identification::<T>(offenders_accounts);
79:
80:     if offenders.len() > 0 {
```

Impact

The application is currently using a more costly implementation to perform a validation control. An adversary may be able to abuse this issue and perform an excessive number of calls to the affected functionalities, in order to force the application to consume more resources.

Since the difference in cost between "*.len*()" and "*is_empty()*" is not significant, the issue is marked as INFORMATIONAL.

Recommendation

It is recommended to replace all the "*.len*()" comparisons with the zero (0) value with the "*.is_empty*()" function call.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.4 Use of "encodePacked" at "EnergyBridge.sol"

Description	INFO

The team identified multiple usage of *"abi.encodePacked"* in the EnergyBridge.sol smart contract. The *"abi.encodePacked"* is a function used for tightly packing variables before hashing or sending them. It's primarily used because it can save gas and storage space compared to *"abi.encode"*. However, its behavior can introduce risks, particularly related to the way it handles variable-length data.

The way *"abi.encodePacked"* handles variable-length inputs can lead to hash collisions. This is because it concatenates inputs without clearly delineating their boundaries. For instance, concatenating a string with a *uint* could result in the same output as concatenating a different string with a different *uint* if the combined bytes end up being the same. Furthermore, Because *"abi.encodePacked"* does not pad its inputs, different sets of inputs can produce the same output. This can be particularly problematic in scenarios where the output is used as a key or identifier (e.g., in generating unique hash values for signatures or transactions).

From the solidity documentation:

If you use keccak256(abi.encodePacked(a, b)) and both a and b are dynamic types, it is easy to craft collisions in the hash value by moving parts of a into b and vice-versa. More specifically, abi.encodePacked("a", "bc") == abi.encodePacked("ab", "c")

Additionally, discussions are also underway about removing *"abi.encodePacked"* in upcoming Solidity releases (*ethereum/solidity#11593*), so using *"abi.encode"* instead at this time guarantees future compatibility.

The issue exists at the following location:

```
File: EnergyBridge.sol
545:     bytes32 ethSignedPrefixMsgHash =
keccak256(abi.encodePacked(ESM_PREFIX, msgHash));
..
File: EnergyBridge.sol
401:     bytes32 ethSignedPrefixMsgHash =
keccak256(abi.encodePacked(ESM_PREFIX, lowerHash));
```

```
..
398:     bytes32 lowerHash = keccak256(abi.encodePacked(token, amount,
recipient, lowerId));
..
364:     bytes32 lowerHash = keccak256(abi.encodePacked(token, amount,
recipient, lowerId));
```

Impact

The usage of "*abi.encodePacked*" allows potential hash collisions. Additionally, upcoming Solidity updates may deprecate "*abi.encodePacked*". Since the contract is using only fixed-length parameters (address, *uint*) when calling the specific functionality, the issue is marked as INFORMATIONAL. When used with fixed-length data types or when it is possible to guarantee the data structure won't introduce ambiguity, "*abi.encodePacked*" can be safe. It's the variable-length data types (like strings and bytes) that introduces the most significant risk.

Recommendation

It is recommended to use "*abi.encode*" instead. The "*abi.encode*" provides clear delimitation between inputs by padding them, which helps avoid the ambiguities and potential collisions associated with "*abi.encodePacked*".

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:R/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.5 Remove "legacyLower()" from EnergyBridge.sol

Description	INFO

Within the EnergyBridge.sol smart contract, the function "*legacyLower()*" is marked as deprecated, with a recommendation for users to utilize "*claimLower()*" as its replacement. The presence of deprecated functions like "*legacyLower()*" within the codebase could lead to confusion and additional deployment costs.

```
File: EnergyBridge.sol
283:   /**
284:    * @dev Method deprecated - please use claimLower() instead.
285:    */
286:   function legacyLower(bytes calldata leaf, bytes32[] calldata
merklePath) //@audit should be removed
287:     onlyWhenLoweringEnabled
288:     lock
289:     external
290:   {
```

Impact

Retaining the deprecated "*legacyLower*()" function in EnergyBridge.sol can lead to confusion among users and developers, alongside incurring additional deployment costs due to the unnecessary code.

Recommendation

It is advised to remove the "*legacyLower*()" function from the EnergyBridge.sol smart contract.

CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:X/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.6 Insufficient argument validation in "_initialiseAuthors()" at "EnergyBridge.sol"

Description	INFO

It was identified that the contract does not ensure that the initial author data are unique during initialization. Specifically, function "*_initialiseAuthors*", which is called by the initializer function, takes as arguments arrays of T1 addresses / public keys and T2 public keys of the authors and registers them to the contract. However, it does not verify that a T1 address or a T2 public key does not appear multiple time in the respective array and, as a result, has already been registered. Consequently, a duplicate T1 address or T2 public key will be re-registered with a new author ID and potentially with a different T2 public key or T1 address, respectively. This is shown in lines 493-497 below:

```
File: contracts/EnergyBridge.sol
488:     do {
489:       _t1Address = t1Address[i];
490:       _t2PubKey = t2PubKey[i];
491:       t1PubKey = abi.encode(t1PubKeyLHS[i], t1PubKeyRHS[i]);
492:       if (address(uint160(uint256(keccak256(t1PubKey)))) !=
_t1Address) revert AddressMismatch();
493:       idToT1Address[nextAuthorId] = _t1Address;
494:       idToT2PubKey[nextAuthorId] = _t2PubKey;
495:       t1AddressToId[_t1Address] = nextAuthorId;
496:       t2PubKeyToId[_t2PubKey] = nextAuthorId;
497:       isAuthor[nextAuthorId] = true;
498:       authorIsActive[nextAuthorId] = true;
```

In a previous version of the contract, the following checks were in place to mitigate these issues, however they have now been removed:

```
File: contracts/EnergyBridge.sol
128:       if (t1AddressToId[_t1Address] != 0) revert
AddressAlreadyInUse(_t1Address);
129:       if (t2PublicKeyToId[_t2PublicKey] != 0) revert
T2PublicKeyAlreadyInUse(_t2PublicKey);
```

Impact

A duplicate T1 address or T2 public key will break the consistency of the data structures of the contract. However, this function is only called by the owner during initialization of the contract and as a result is marked as INFORMATIONAL.

Recommendation

It is recommended to verify that all supplied T1 addresses and T2 public keys have not already been registered before registering them with a new author ID.

CVSS Score

AV:N/AC:L/PR:H/UI:N/S:U/C:N/I:N/A:N/E:X/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.7 Off-by-one access in "_verifyConfirmations()" at "EnergyBridge.sol"

Description	INFO

It was identified that the contract does not properly iterate over the array of confirmation proofs during their verification. Specifically, function "*_verifyConfirmations*" iterates over the array using a do-while loop, the condition of which compares the current confirmation index to the total number of confirmations. Although the confirmation index is zero-based (i.e. runs in range zero to total number of confirmations minus 1), the do-while loop condition is "less or equal to". This allows the current confirmation index to have a value equal to the total number of confirmations, which is outside the range of valid confirmation indices. This is shown in line 586 below:

```
File: contracts/EnergyBridge.sol
564:     do {
...
582:       // Setup the next iteration of the loop:
583:       authorId = _recoverAuthorId(ethSignedPrefixMsgHash,
confirmationsOffset, confirmationsIndex);
584:       unchecked { ++confirmationsIndex; }
585:
586:     } while (confirmationsIndex <= numConfirmations);
```

As a result, before the end of the last iteration, function "*_recoverAuthorId*" will be called with an off-by-one confirmation index. Furthermore, if this verification is a non-lower verification, the value of the total number of confirmations is incremented by one, thus adding an extra iteration with an off-by-two confirmation index. This is shown in line 561 below:

```
File: contracts/EnergyBridge.sol
556:     if (isLower) {
...
559:     } else { // For non-lowers there is a high likelihood the sender
is an author, so their confirmation is taken to be implicit
560:       authorId = t1AddressToId[msg.sender];
```

```
561:         unchecked { ++numConfirmations; }
562:     }
```

Finally, for this condition to be achieved, it is assumed that the number of required confirmations is not reached and thus the function does not early return.

Impact

In the do-while loop, function "*_recoverAuthorId*" may be called with an invalid confirmation index. However, in the current implementation this will only lead to out-of-bound access to *calldata*, which are already public. Furthermore, these data cannot be leaked to an adversary. As a result, this issue is marked as INFROMATIONAL.

Recommendation

It is recommended to ensure that an invalid (e.g. off-by-one) confirmation index is not used to access the confirmation proofs array. This may be implemented by changing the loop condition to "lower than" and making the appropriate changes in the rest of the function implementation.

CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.8 Empty array access in "_verifyConfirmations()" at "EnergyBridge.sol"

Description	INFO

It was identified that the contract does not ensure that the array of confirmation proofs contains at least one during their verification. Specifically, function "*_verifyConfirmations*" access the first confirmation (confirmation index zero) without explicitly checking that it exists and that the confirmation proofs array is not empty. This is shown in line 557 below:

```
File: contracts/EnergyBridge.sol
547:     uint256 numConfirmations = confirmations.length /
SIGNATURE_LENGTH;
548:     uint256 confirmationsOffset;
549:     uint256 confirmationsIndex;
...
553:     assembly { confirmationsOffset := confirmations.offset }
...
556:     if (isLower) {
557:       authorId = _recoverAuthorId(ethSignedPrefixMsgHash,
confirmationsOffset, confirmationsIndex);
```

This function is called by external function "*addAuthor*", "*removeAuthor*", "*publishRoot*" and "*claimLower*". With the exception of the last one, the first three function do not verify that the confirmation proofs array contains at least one confirmation proof.

Impact

If an adversary calls one of the aforementioned external functions with an empty confirmation proofs array, function "*_recoverAuthorId*" will be called with an invalid confirmation index. However, in the current implementation this will only lead to out-of-bound access to *calldata*, which are already public. Furthermore, these data cannot be leaked to an adversary. As a result, this issue is marked as INFORMATIONAL.

Recommendation

It is recommended to verify that at least one confirmation proof exists before accessing the first element of the confirmation proofs array.

CVSS Score
AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.9 Empty array access in "confirmTransaction()" at "EnergyBridge.sol"

Description	INFO

It was identified that the contract does not ensure that the array of *Merkle* path nodes contains at least one during their confirmation. Specifically, public function "*confirmTransaction*" access the first node (index zero) without explicitly checking that it exists and the Merkle path nodes array is not empty. This is shown in line 454 below:

```
File: contracts/EnergyBridge.sol
445:   function confirmTransaction(bytes32 leafHash, bytes32[] calldata
merklePath)
...
451:     uint256 i;
452:
453:     do {
454:       node = merklePath[i];
...
457:     } while (i < merklePath.length);
```

Furthermore, function *legacyLower* (which calls this function) does not check the array's size, neither.

Impact

An adversary cannot currently exploit this issue, as the array access is a checked array access that will revert if an invalid index is used. As a result, this issue is marked as INFORMATIONAL.

Recommendation

It is recommended to verify that at least one Merkle path node exists before accessing the first element of the Merkle path nodes array.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.10 Unbounded recursion in "request::process_next_request" at "eth-bridge"

Description	INFO

The team identified that there is an unbounded recursion in the queue processing functionality of the "eth-bridge" pallet. More precisely, the following extrinsics of the "eth-bridge" pallet utilize (directly or indirectly via other functions) the *"request::process_next_request"* function to trigger the processing of the next request in the active request queue:

- *add_confirmation*
- *add_corroboration*
- *remove_active_request*

It was identified this function recursively calls itself as long as more requests are present in the active request queue, without a limit on the number of active requests that may be processed, potentially consuming the weight of the respective extrinsic. This is shown at lines 73 and 85 in the code snippet below:

File: pallets/eth-bridge/src/request.rs

```
59: pub fn process_next_request<T: Config>() {
60:     ActiveRequest::<T>::kill();
61:
62:     if let Some(req) = request::dequeue_request::<T>() {
63:         match req {
64:             Request::Send(send_req) => {
65:                 if let Err(e) =
tx::set_up_active_tx::<T>(send_req.clone()) {
66:                     // If we failed to setup the next request, notify
caller
...
73:                     process_next_request::<T>();
74:                 }
75:             },
76:             Request::LowerProof(lower_req) => {
77:                 if let Err(e) =
set_up_active_lower_proof::<T>(lower_req.clone()) {
...
85:                     process_next_request::<T>();
86:                 }
```

```
87:                 },
88:             };
89:         };
90: }
```

Impact

Assuming that a malicious author can put a large number of requests in the queue, a future call to one of the aforementioned extrinsics may consume the weight of the extrinsic and fail as the *"request::process_next_request"* function would try to process all the requests in the queue. However, in the current implementation such a case is not possible, as a malicious author cannot put an arbitrary number of requests in the queue. For this reason, this issue is marked as INFORMATIONAL.

Recommendation

It is recommended to limit the maximum number of requests that can be recursively processed by a call to "*request::process_next_request*". Furthermore, the weight calculation of all extrinsics that call (directly or indirectly) this function should take the maximum number of requests that may be processed into account.

CVSS Score

AV:N/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.11 File-based instead of a hardware-backed keystore in-use at "avn-service"

Description	INFO

The team identified that the "*avn-service*" is currently using a file-based keystore to maintain the private key which is used to perform the signing operations, instead of a keystore located in a trusted hardware.

The trusted hardware (also known as "Trusted Computing') refers to computing components or modules designed with hardware-based security features to provide a foundation of trust for the systems they operate within. These components are engineered to resist tampering and to securely handle sensitive information, including cryptographic keys, user data, and software integrity verification. The trusted hardware is designed to be secured both for its owner and against its owner. It usually includes the following attributes: an endorsement key (a key generated in the chip that never leaves the chip, also known as "hardware bound" key), secure input and output, memory curtaining / protected execution, sealed storage, and remote attestation. Several types of trusted hardware are commonly used in various computing environments, such as Trusted Platform Modules (TPM), Trusted Execution Environments (TEE), Hardware Security Modules (HSM), Cloud-based Hardware Security Modules, Secure Enclaves, Smart Cards (SC), Secure Elements (SE), etc.

With the file-based keystore implementation, the key can be leaked if an adversary is able to access the file or the server's memory components. A hardware-backed keystore is able to withstand such attacks, at it runs all sensitive operations in an isolated environment, minimizing the risk of exposure to other components, while at the same time it offers robust mechanisms for managing cryptographic keys, including key generation, storage and usage.

The issue exists at the following location:

```
File: /avn-parachain-5.1.1/node/avn-service/src/keystore_utils.rs
74: /// Get the key phrase for a given public key and key type.
75: // See:
https://github.com/paritytech/substrate/blob/7db3c4fc5221d1f3fde36f1a5ef3
042725a0f616/client/keystore/src/local.rs#L469
76: fn key_phrase_by_type(
```

```
77:     eth_address: &[u8],
78:     key_type: KeyTypeId,
79:     keystore_path: &PathBuf,
80: ) -> Result<String, TideError> {
81:     let mut path = keystore_path.clone();
82:     path.push(hex::encode(key_type.0) +
hex::encode(eth_address).as_str());
83:
84:     if path.exists() {
85:         let file = File::open(path)
86:             .map_err(|e| server_error(format!("Error opening EthKey
file: {:?}", e)))?;
87:         serde_json::from_reader(&file).map_err(Into::into)
88:     } else {
89:         Err(server_error(format!(
90:             "Keystore file for EthKey: {:?} not found",
91:             ETHEREUM_SIGNING_KEY
92:         )))?
93:     }
94: }
```

Impact

An adversary who is able to gain access at the keystore file or at the server's memory through another attack vector (e.g. OpenSSL's *heartbleed*), will be able to extract the protected cryptographic keys, and in the specific case the seed key phrase.

Recommendation

It is advisable to use a hardware-backed keystore that is implemented using Trusted Hardware such as a Physical HSM, a cloud-based HSM (also known as *HSM as a Service*). The application needs to use libraries or SDKs compatible with the selected solution for cryptographic operations. Rust crates such as pkcs11 or rust-crypto might be useful, depending on the trusted hardware interface.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.12 Strict Transport Security (HSTS) is not enforced in Tide framework at "avn-service"

Description	INFO

It was identified that the "*avn-service*" does not enforce the use of Strict Transport Security (HSTS), allowing users to establish unencrypted connections to it. If an attacker can alter the network traffic of a legitimate user, they could circumvent the application's SSL/TLS encryption, by downgrading the connection to an insecure one. The method involves changing HTTPS links to HTTP. This way, if a user clicks on a link to the site from an HTTP page, their browser will not attempt to establish an encrypted connection. For example, the *sslstrip* tool can automate this attack process.

For an attacker to exploit this weakness, they need to be in a position where they can intercept and alter the victim's network traffic. This situation is common on insecure connections like public Wi-Fi or corporate or home networks that share space with an infected computer. Traditional defenses, like using switched networks, are inadequate against such attacks. An attacker located within the user's ISP or within the infrastructure hosting the application could also carry out this attack. It's important to recognize that a sophisticated adversary might be able to target any internet-based connection.

Impact

An adversary who is in a MiTM position, can downgrade a secure connection (over TLS/SSL) of a legitimate user to an insecure one, and eavesdrop, intercept or tamper with the HTTP packets.

Since this issue is an added layer of security, it is marked as INFORMATIONAL.

Recommendation

In Rust's Tide framework, middleware can be used to set or modify HTTP headers. For example:

```
use tide::{Next, Request, Response, Result};
async fn security_headers_middleware(req: Request<()>, next: Next<'_,
()>) -> Result {
```

```
    let mut res = next.run(req).await;
    let headers = res.headers_mut();
    headers.insert("Strict-Transport-Security", "max-age=63072000;
includeSubDomains; preload".parse().unwrap());
    Ok(res)
}
```

CVSS Score

AV:A/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.13 MIME sniffing is not disabled in Tide framework at "avn-service"

Description	INFO

It was found that the "*avn-service*" does not disable MIME sniffing in the Tide HTTP server responses. If a response does not specify a content type, then the browser will usually analyze the response and attempt to determine the MIME type of its content (MIME sniffing). This can have unexpected results, and if the content contains any user-controllable data may lead to cross-site scripting or other client-side vulnerabilities.

Impact

An adversary may be able to exploit this issue in order to use the Tide API to perform a Cross Site Scripting (XSS) attack. It should be noted that the absence of a content type statement does not necessarily constitute a security flaw, particularly if the response contains static content. As a result, this issue is marked as INFORMATIONAL.

Recommendation

In Rust's Tide framework, middleware can be used to set or modify HTTP headers. For example:

```
use tide::{Next, Request, Response, Result};
async fn security_headers_middleware(req: Request<()>, next: Next<'_,
()>) -> Result {
    let mut res = next.run(req).await;
    let headers = res.headers_mut();
    headers.insert("X-Content-Type-Options", "nosniff".parse().unwrap());
    Ok(res)
}
```

CVSS Score

AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:N/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.14 CSP is not enabled in Tide framework at "avn-service"

Description	INFO

The team identified that the Tide HTTP server is not using the (Content Security Policy) CSP headers. CSP is a browser security mechanism that aims to mitigate XSS and some other attacks. It works by restricting the resources (such as scripts and images) that a page can load and restricting whether a page can be framed by other pages.

To enable CSP, a response needs to include an HTTP response header called Content-Security-Policy with a value containing the policy. The policy itself consists of one or more directives, separated by semicolons.

Impact

An adversary may be able to exploit this issue in order to use the Tide API to perform a Cross Site Scripting (XSS) attack. It should be noted that the absence of this header does not necessarily constitute a security flaw, particularly if there is no unvalidated user-controlled input that reaches the HTTP response body.

Since this is an added layer of security, the issue is marked as INFORMATIONAL.

Recommendation

In Rust's Tide framework, middleware can be used to set or modify HTTP headers. For example:

```
use tide::{Next, Request, Response, Result};
async fn security_headers_middleware(req: Request<()>, next: Next<'_,
()>) -> Result {
    let mut res = next.run(req).await;
    let headers = res.headers_mut();
    headers.insert("Content-Security-Policy", "default-src
'self';".parse().unwrap());
    Ok(res)
}
```

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.15 HTTP Caching is allowed in Tide framework at "avn-service"

Description	INFO

It was identified that the "*avn-service*" does not set the appropriate HTTP directives in the HTTP headers in order to disable the HTTP Caching by the user browsers and intermediate proxies or middlewares.

Browsers are capable of storing local copies of content obtained from web servers in their cache, unless specifically instructed not to. This behavior is also true for content delivered over HTTPS, with browsers like Internet Explorer known to cache such secure content. Consequently, if the responses from applications contain sensitive information and are cached locally, there's a risk that this data could later be accessed by others using the same computer.

Disabling HTTP caching in Tide, a Rust web framework, ensures that responses from your web application are not cached by clients or proxies. This can be important for dynamic content that changes frequently or for sensitive information that should not be stored on the client side.

Impact

An adversary who is able to gain access at the cached HTTP responses of the avn-service, may be able to extract sensitive content.

Since this is an added layer of security issue, it is marked as INFORMATIONAL.

Recommendation

It is recommended to disable HTTP caching by introducing a middleware that injects the appropriate HTTP directives. For example:

```
use tide::{Next, Request, Result};
async fn no_cache_middleware(req: Request<()>, next: Next<'_, ()>) ->
Result {
    let mut res = next.run(req).await;
    let headers = res.headers_mut();
    headers.insert("Cache-Control", "no-store, max-
age=0".parse().unwrap());
    headers.insert("Pragma", "no-cache".parse().unwrap());
```

```
    headers.insert("Expires", "0".parse().unwrap());
    Ok(res)
}
#[async_std::main]
async fn main() -> tide::Result<()> {
    let mut app = tide::new();
    app.with(no_cache_middleware);

}
```

CVSS Score

AV:L/AC:L/PR:H/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.16 TLS/SSL is not enabled in Tide framework at "avn-service"

Description	INFO

The team identified that the "*avn-service*" does not support TLS/SSL, and as a result, it permits users to establish unencrypted connections. If an attacker is strategically placed to observe a legitimate user's network activity, they could capture and scrutinize the user's engagement with the application, gaining access to any data provided by the user.

Moreover, if an attacker has the capability to alter the network traffic, they could leverage the application to launch attacks against both its users and external websites. For example, the adversary could alter the data that is sent to be signed, or to inject malicious transactions.

Unencrypted connections have been exploited by Internet Service Providers (ISPs) and government entities for user tracking, as well as for injecting advertisements and malicious JavaScript into web pages. Given these risks, web browser developers are moving towards marking unencrypted connections as unsafe visually.

Impact

An adversary who is in a Man-in-the-Middle (MiTM) position, will be able to eavesdrop, intercept and tamper with the established communication channel, and as a result affect the responses of the API or change the provided values to the API.

As this is currently an internal service, the issue is marked as INFORMATIONAL.

Recommendation

It is advisable to enable TLS/SSL support in the "*avn-service*". As the Rust Tide framework does not natively support TLS/SSL, either a third-party library should be integrated, or a reverse proxy solution should be used in the production environment.

- If a reverse proxy like Nginx or Traefik is placed in front of the "*avn-service*", it will handle the HTTPS connections and forward the traffic to the application over HTTP locally.

- Otherwise, an external tool like "*async-rustls*" can be used to integrate TLS into the avn-service. For example:

```
use async_std::task;
use tide::security::{CorsMiddleware, Origin};
use tide_rustls::TlsListener;
#[async_std::main]
async fn main() -> tide::Result<()> {
    let mut app = tide::new();
    app.with(CorsMiddleware::new().allow_origin(Origin::from("*")));
    // Specify the path to your Let's Encrypt certificates
    let tls_config = TlsListener::build()
        .addrs("127.0.0.1:443")
        .cert("path/to/fullchain.pem")
        .key("path/to/privkey.pem");
    app.listen(tls_config).await?;
    Ok(())
}
```

CVSS Score

AV:A/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.17 Authentication not enforced in "/eth/sign" and "/eth/send" functionalities at "avn-service"

Description	INFO

It was identified that two APIs of the "*avn-service*" that are responsible to perform cryptographic operations using the AVN private key which is stored in the local keystore, are not protected by an authentication control.

The issue exists in the following locations:

```
File: /avn-parachain-5.1.1/node/avn-service/src/lib.rs
411:     app.at("/eth/sign/:data_to_sign").get(
412:         |req: tide::Request<Arc<Config<Block, ClientT>>>| async move
{
413:             log::info!("⛓  avn-service: sign Request");
414:             let secp = Secp256k1::new();
415:             let keystore_path = &req.state().keystore_path;
416:
417:             let data_to_sign: Vec<u8> =
418:
hex::decode(req.param("data_to_sign")?.trim_start_matches("0x")).map_err(
|e| {
419:                     server_error(format!("Error converting
data_to_sign into hex string {:?}", e))
420:                 })?;
421:
422:             let hashed_message =
hash_with_ethereum_prefix(&data_to_sign);
423:
424:             log::info!(
425:                 "⛓  avn-service: data to sign: {:?},\n hashed data
to sign: {:?}",
426:                 hex::encode(data_to_sign),
427:                 hex::encode(hashed_message)
428:             );
429:             let my_eth_address =
get_eth_address_bytes_from_keystore(keystore_path)?;
430:             let my_priv_key = get_priv_key(keystore_path,
&my_eth_address)?;
```

```
431:
432:             let secret = SecretKey::from_slice(&my_priv_key)?;
433:             let message =
secp256k1::Message::from_slice(&hashed_message)?;
434:             let signature: Signature =
secp.sign_ecdsa_recoverable(&message, &secret).into();
435:
436:             Ok(hex::encode(signature.encode()))
437:         },
438:     );
439:
440:     app.at("/eth/send")
441:         .post(|req: tide::Request<Arc<Config<Block, ClientT>>>|
async move {
442:             // Methods that require web3 must be run within the
tokio runtime (#[tokio::main])
443:             return send_main(req)
444:         });
445:
```

And in:

File: /avn-parachain-5.1.1/node/avn-service/src/lib.rs

```
264: #[tokio::main]
265: async fn send_main<Block: BlockT, ClientT>(
266:     mut req: tide::Request<Arc<Config<Block, ClientT>>>,
267: ) -> Result<String, TideError>
268: where
269:     ClientT: BlockBackend<Block> + UsageProvider<Block> + Send +
Sync + 'static,
270: {
271:     log::info!("⛓️  avn-service: send Request");
272:     let post_body = req.body_bytes().await?;
273:     let send_request = &EthTransaction::decode(&mut &post_body[..])
274:         .map_err(|e| server_error(format!("Error decoding eth
transaction data: {:?}", e)))?;
275:
276:     if let Some(mut mutex_web3_data) =
req.state().web3_data_mutex.try_lock() {
277:         if mutex_web3_data.web3.is_none() {
```

```
278:             return Err(server_error("Web3 connection not
setup".to_string()))
279:         }
280:         let keystore_path = &req.state().keystore_path;
281:
282:         let my_eth_address =
get_eth_address_bytes_from_keystore(&keystore_path)?;
283:         let my_priv_key = get_priv_key(&keystore_path,
&my_eth_address)?;
284:
285:         let mut tx_hash =
286:             send_tx(&mut *mutex_web3_data, send_request,
&my_eth_address, my_priv_key).await;
```

Impact

An adversary, who is able to access the exposed API service, will be able to perform unwanted signing operations.

It should be noted that the adversary does not need to have direct access to the local service, as it is possible to trick a user with access to the service to perform the attack using a Cross Site Request Forgery (CSRF) vector.

Recommendation

Implementing authentication control in a Tide application involves creating middleware or leveraging existing libraries to manage user sessions and validate credentials. For example, a simple solution is the following:

```
use tide::{Request, Response, Next, Result, StatusCode};
struct User {
    username: String,
    password: String, // In a real app, passwords should be hashed
}
async fn authenticate(req: Request<()>) -> Result {
    let maybe_user = req.header("Authorization")
        .and_then(|headers| headers.get(0))
        .map(|value| {
            // Basic parsing for demonstration; use a more robust method
```

```
in production
            let encoded = value.as_str().trim_start_matches("Basic ");
            let decoded = base64::decode(encoded).unwrap_or_default();
            let credentials =
String::from_utf8(decoded).unwrap_or_default();
            let mut parts = credentials.splitn(2, ':');
            User {
                username: parts.next().unwrap_or("").to_string(),
                password: parts.next().unwrap_or("").to_string(),
            }
        });
    // Here you should verify the user credentials, e.g., against a
database
    if let Some(user) = maybe_user {
        if user.username == "admin" && user.password == "secret" {
            Ok(Response::new(StatusCode::Ok))
        } else {
            Ok(Response::new(StatusCode::Unauthorized))
        }
    } else {
        Ok(Response::new(StatusCode::Unauthorized))
    }}
async fn protected_endpoint(_req: Request<()>) -> tide::Result<String> {
    Ok("You are authenticated".to_string())
}
#[async_std::main]
async fn main() -> tide::Result<()> {
    let mut app = tide::new();
    app.at("/protected").with(authenticate).get(protected_endpoint);
    app.listen("127.0.0.1:8080").await?;
    Ok(())
}
```

However, it is recommended to use a more robust solution with an authentication control library that is able to withstand brute force attack, or choose sophisticated authentication flows, such as OAuth2, OpenID Connect, or token-based authentication using JWT, and password-less authentication schemes.

CVSS Score

AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

6 Retesting Results

6.1 Retest of Medium Severity Findings

Results from retesting carried out on April 2024, determined that four (4 out of 6) reported MEDIUM risk issues were sufficiently addressed (see sections *5.1.1, 5.1.3, 5.1.4* and *5.1.6*). The rest MEDIUM risk findings remain OPEN:

- *5.1.2* - Weight of MAX_NUMBER_OF_ROOT_DATA_PER_RANGE is not calculated in "record_summary_calculation" at "summary"
- *5.1.5* - Outdated and Vulnerable dependencies in-use at "Cargo.lock"

6.1.1 Weight of "create_and_report" is not calculated in multiple functionalities at "summary", "eth-bridge" and "ethereum-events"

The issue has been fixed by taking into account the MAX_VALIDATOR_ACCOUNTS during the weight calculation as it can be seen in the following commits:

- *https://github.com/AventusProtocolFoundation/avn-parachain/blob/6b9595d93571eb3a227441cae8ad1c9c61f2968c/pallets/summary/src/benchmarking.rs#L495*
- *https://github.com/AventusProtocolFoundation/avn-parachain/blob/6b9595d93571eb3a227441cae8ad1c9c61f2968c/pallets/summary/src/benchmarking.rs#L565*
- *https://github.com/AventusProtocolFoundation/avn-parachain/pull/383*

```
File: /avn-parachain-avn-v5.1.1/pallets/summary/src/lib.rs
546:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::add_challenge(MAX_VALIDATOR_ACCOUNTS))]
547:         #[pallet::call_index(6)]
548:         pub fn add_challenge(
549:             origin: OriginFor<T>,
550:             challenge: SummaryChallenge<T::AccountId>,
551:             validator: Validator<T::AuthorityId, T::AccountId>,
552:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
553:         ) -> DispatchResult {
```

```
File: avn-parachain-avn-v5.1.1/pallets/summary/src/lib.rs
```

```
529:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::advance_slot_with_offence(MAX_VALIDATOR_ACCOUN
TS).max(
530:             <T as
Config>::WeightInfo::advance_slot_without_offence(MAX_VALIDATOR_ACCOUNTS)
531:         ))]
532:         #[pallet::call_index(5)]
533:         pub fn advance_slot(
534:             origin: OriginFor<T>,
535:             validator: Validator<<T as avn::Config>::AuthorityId,
T::AccountId>,
536:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
537:         ) -> DispatchResult {
```

And similarly, in “Ethereum-events” and “eth-bridge”:

```
File: /avn-parachain-avn-v5.1.1/pallets/ethereum-events/src/lib.rs
503:
504:         #[pallet::call_index(3)]
505:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::process_event_with_successful_challenge(
506:             MAX_VALIDATOR_ACCOUNTS,
507:             MAX_NUMBER_OF_EVENTS_PENDING_CHALLENGES
508:         )
509:             .max(<T as
Config>::WeightInfo::process_event_without_successful_challenge(
510:                 MAX_VALIDATOR_ACCOUNTS,
511:                 MAX_NUMBER_OF_EVENTS_PENDING_CHALLENGES
512:             ))]
513:         pub fn process_event(
514:             origin: OriginFor<T>,
515:             event_id: EthEventId,
516:             _ingress_counter: IngressCounter, /* this is not used in
this function, but is added
517:                                               * here so that
`_signature` can use this value to
518:                                               * become different from
previous calls. */
519:             validator: Validator<T::AuthorityId, T::AccountId>,
```

```
520:             // Signature and structural validation is already done in
validate unsigned so no need
521:             // to do it here
522:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
523:         ) -> DispatchResultWithPostInfo {
```

```
File: /avn-parachain-avn-v5.1.1/pallets/eth-bridge/src/lib.rs
401:         #[pallet::call_index(4)]
402:         #[pallet::weight( <T as
pallet::Config>::WeightInfo::add_corroboration().max(
403:             <T as
Config>::WeightInfo::add_corroboration_with_challenge(MAX_VALIDATOR_ACCOUNT
S)
404:         ))]
405:         pub fn add_corroboration(
406:             origin: OriginFor<T>,
407:             tx_id: EthereumId,
408:             tx_succeeded: bool,
409:             tx_hash_is_valid: bool,
410:             author: Author<T>,
411:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
412:         ) -> DispatchResultWithPostInfo {
```

6.1.2 Potential voting fraud by transaction sender in "add_corroboration" at "eth-bridge"

A check has been introduced to mitigate this issue in the following commits:

- https://github.com/AventusProtocolFoundation/avn-parachain/commit/0a3fe9a0edad7407d28aea6203439c395077decf
- https://github.com/AventusProtocolFoundation/avn-parachain/pull/388/files

```
File: /avn-parachain-avn-v5.1.1/pallets/eth-bridge/src/lib.rs
418:                 if tx.tx_data.is_some() {
419:                     let data = tx.tx_data.as_mut().expect("has data");
```

```
420:
421:                         let author_is_sender = author.account_id ==
data.sender;
422:                         ensure!(!author_is_sender,
Error::<T>::CannotCorroborateOwnTransaction);
```

6.1.3 The size of the HTTP body in "send_main", "view_main" and "tx_query_main" functionalities is not checked at "avn-service"

A check has been introduced to mitigate this issue in the following commit:

- https://github.com/AventusProtocolFoundation/avn-parachain/commit/44ca910dec4e749bc5e71e26b9bc6ef71b63ce42

```
File: /avn-parachain-avn-v5.1.1/node/avn-service/src/lib.rs:
  273       log::info!("⛓  avn-service: send Request");
  274       let post_body = req.body_bytes().await?;
  275:      if post_body.len() > MAX_BODY_SIZE {
  276           return Err(server_error(format!("Request body too large.
Size: {:?}", post_body.len())))
  277       }
  ...
  333       log::info!("⛓  avn-service: view Request");
  334       let post_body = req.body_bytes().await?;
  335:      if post_body.len() > MAX_BODY_SIZE {
  336           return Err(server_error(format!("Request body too large.
Size: {:?}", post_body.len())))
  337       }
  ...
  370       log::info!("⛓  avn-service: query Request.");
  371       let post_body = req.body_bytes().await?;
  372:      if post_body.len() > MAX_BODY_SIZE {
  373           return Err(server_error(format!("Request body too large.
Size: {:?}", post_body.len())))
  374       }
```

6.1.4 Weight of ConfirmationsLimit is not calculated in "add_confirmation" at "eth-bridge"

The weight is now correctly calculated in the following commits:

- *https://github.com/AventusProtocolFoundation/avn-parachain/pull/377/files*
- *https://github.com/AventusProtocolFoundation/avn-parachain/pull/386/files*

```
File: /avn-parachain-avn-v5.1.1/pallets/eth-bridge/src/lib.rs
316:         #[pallet::call_index(2)]
317:         #[pallet::weight(<T as
Config>::WeightInfo::add_confirmation(MAX_CONFIRMATIONS))]
318:         pub fn add_confirmation(
319:             origin: OriginFor<T>,
320:             request_id: u32,
321:             confirmation: ecdsa::Signature,
322:             author: Author<T>,
323:             _signature: <T::AuthorityId as
RuntimeAppPublic>::Signature,
324:         ) -> DispatchResultWithPostInfo {
325:             ensure_none(origin)?;
```

6.2 Retest of Low Severity Findings

Two (2 out of 9) reported LOW risk issues were found to be sufficiently mitigated (*5.2.2, 5.2.6)*. The rest LOW risk findings remain OPEN:

- *5.2.1* - Potential gas griefing attack in "_releaseFunds()" at "EnergyBridge.sol"
- *5.2.3* - Insecure error handling exposes sensitive information in "raw_public_keys" at "avn-service"
- *5.2.4* - Insecure error handling exposes sensitive information in "key_phrase_by_type" at "avn-service"
- *5.2.5* - No max range limitation in "/roothash/:from_block/:to_block" API call at "avn-service"
- *5.2.7* - Return value of "ecrecover()" is not checked at "EnergyBridge.sol"
- *5.2.8* - No max-time limitation in asynchronous calls to external services in "web3_utils" at "avn-service"
- *5.2.9* - Missing event emit in "settle_transfer" at "token-manager"

6.2.1 Insufficient Information for event emission in "claimLower()" at "EnergyBridge.sol"

The issue has been fixed, as an event is now emitted with all the necessary information:

```
File: /energy-bridge-
308d3aed6912f38926fd4af61e5ef781a9fa48b5/contracts/EnergyBridge.sol
396:     emit LogLowered(lowerId, token, recipient, amount);
```

6.2.2 Owner can renounce Ownership at "EnergyBridge.sol"

The issue has been fixed in the following commit id:

- *https://github.com/energywebfoundation/energy-bridge/pull/40*

```
File: /energy-bridge-
308d3aed6912f38926fd4af61e5ef781a9fa48b5/contracts/EnergyBridge.sol
490:
491:   /**
492:    * @dev Disables the renounceOwnership function to prevent
relinquishing ownership.
493:    */
494:   function renounceOwnership() public view override onlyOwner {
495:     revert RenounceOwnershipDisabled();
496:   }
```

6.3 Retest of Informational Findings

Two (2 out of 17) reported issues bearing no risk (INFORMATIONAL) were found to be sufficiently mitigated (*5.3.3, 5.3.6)*. One (1 out of 17) INFORMATIONAL issues was also marked as risk-accepted (see section *5.3.2*). The rest INFORMATIOANL findings remain OPEN:

- *5.3.1* - Use of "ecrecover()" at EnergyBridge.sol
- *5.3.4* - Use of "encodePacked" at "EnergyBridge.sol"
- *5.3.5* - Remove “legacyLower()” from EnergyBridge.sol
- *5.3.7* -Off-by-one access in "_verifyConfirmations()" at "EnergyBridge.sol"
- *5.3.8*—Empty array access in "_verifyConfirmations()" at "EnergyBridge.sol"
- *5.3.9* – Empty array access in "confirmTransaction()" at "EnergyBridge.sol"

- *5.3.10* – Unbounded recursion in "request::process_next_request" at "eth-bridge"
- *5.3.11* – File-based instead of a hardware-backed keystore in-use at "avn-service"
- *5.3.12* – Strict Transport Security (HSTS) is not enforced in Tide framework at "avn-service"
- *5.3.13* – MIME sniffing is not disabled in Tide framework at "avn-service"
- *5.3.14* – CSP is not enabled in Tide framework at "avn-service"
- *5.3.15* – HTTP Caching is allowed in Tide framework at "avn-service"
- *5.3.16* – TLS/SSL is not enabled in Tide framework at "avn-service"
- *5.3.17* - Authentication not enforced in "/eth/sign" and "/eth/send" functionalities at "avn-service"

6.3.1 Costly length comparison at "ethereum-events" and "summary"

The comparison has been replaced in the following pull request (it had not been merged yet):

- *https://github.com/AventusProtocolFoundation/avn-parachain/pull/392*

```
File: /avn-parachain-avn-v5.1.1/pallets/ethereum-events/src/lib.rs
1343:     fn compute_result(
1344:         block_number: T::BlockNumber,
1345:         response_body: Result<Vec<u8>, DispatchError>,
1346:         event_id: &EthEventId,
1347:         validator_account_id: &T::AccountId,
1348:     ) -> EthEventCheckResult<T::BlockNumber, T::AccountId> {
...
1378:         if response_data_object.is_empty() {
1379:             log::error!("❌ Response data json is empty");
1380:             return invalid_result
1381:         };
```

```
https://github.com/AventusProtocolFoundation/avn-parachain/blob/748789fd5a2d7440e46d3690d0d2672a03c81d22/pallets/ethereum-events/src/event_parser.rs
fn get_data(event: &JsonValue) -> Result<Option<Vec<u8>>, SimpleError> {
    let event = event.get_object()?;
```

```
    let data = get_value_of(String::from("data"),
event)?.get_string().map_err(|e| {
        log::error!("❌ Unable to extract data from event {:?} - {:?}",
event, e);
        e
    })?;

    let bytes = hex_to_bytes(data)?;

    if !bytes.is_empty() {
        return Ok(Some(bytes))
    }

    return Ok(None)
}
```

https://github.com/AventusProtocolFoundation/avn-parachain/blob/748789fd5a2d7440e46d3690d0d2672a03c81d22/pallets/ethereum-events/src/event_parser.rs

```
pub fn create_and_report_invalid_log_offence<T: crate::Config>(
    reporter: &T::AccountId,
    offenders_accounts: &Vec<T::AccountId>,
    offence_type: EthereumLogOffenceType,
) {
    let offenders =
create_offenders_identification::<T>(offenders_accounts);

    if !offenders.is_empty() {
        let invalid_event_offence
```

https://github.com/AventusProtocolFoundation/avn-parachain/blob/748789fd5a2d7440e46d3690d0d2672a03c81d22/pallets/ethereum-events/src/event_parser.rs

```
pub fn create_and_report_summary_offence<T: crate::Config>(
    reporter: &T::AccountId,
    offenders_accounts: &Vec<T::AccountId>,
```

```
    offence_type: SummaryOffenceType,
) {
    let offenders =
create_offenders_identification::<T>(offenders_accounts);

if !offenders.is_empty() {
```

6.3.2 Insufficient argument validation in "_initialiseAuthors()" at "EnergyBridge.sol"

Two (2) checks have been introduced and the issue is now mitigated:

```
File: /Users/fisherman/Desktop/ENG-002/energy-bridge-
308d3aed6912f38926fd4af61e5ef781a9fa48b5/contracts/EnergyBridge.sol
500:   function _initialiseAuthors(
501:     address[] calldata t1Addresses,
502:     bytes32[] calldata t1PubKeysLHS,
503:     bytes32[] calldata t1PubKeysRHS,
504:     bytes32[] calldata t2PubKeys
505:   ) private {
506:     uint256 numAuth = t1Addresses.length;
507:     if (numAuth < MINIMUM_NUMBER_OF_AUTHORS) revert
NotEnoughAuthors();
508:     if (t1PubKeysLHS.length != numAuth || t1PubKeysRHS.length !=
numAuth || t2PubKeys.length != numAuth) revert MissingKeys();
509:
510:     bytes memory t1PubKey;
511:     address t1Address;
512:     uint256 i;
513:
514:     do {
515:       t1Address = t1Addresses[i];
516:       t1PubKey = abi.encode(t1PubKeysLHS[i], t1PubKeysRHS[i]);
517:       if (address(uint160(uint256(keccak256(t1PubKey)))) != t1Address)
revert AddressMismatch();
518:       if (t1AddressToId[t1Address] != 0) revert
T1AddressInUse(t1Address);
519:       _activateAuthor(_addNewAuthor(t1Address, t2PubKeys[i]));
520:       unchecked {
521:         ++i;
```

```
522:         }
523:       } while (i < numAuth);
524:     }
525:
526:     function _addNewAuthor(address t1Address, bytes32 t2PubKey) private
returns (uint256 id) {
527:       unchecked {
528:         id = nextAuthorId++;
529:       }
530:       if (t2PubKeyToId[t2PubKey] != 0) revert T2KeyInUse(t2PubKey);
531:       idToT1Address[id] = t1Address;
532:       idToT2PubKey[id] = t2PubKey;
533:       t1AddressToId[t1Address] = id;
534:       t2PubKeyToId[t2PubKey] = id;
535:       isAuthor[id] = true;
536:     }
```

6.3.3 Removal of Merkle tree root hash is not possible at "EnergyBridge.sol"

The team replied that this is by design since root hashes are intended to reflect the finalized state of T2. If an unexpected issue should occur then, as the report notes, lowering can be turned off until it's fixed.

References & Applicable Documents

Ref.	Title	Version
N/A	*N/A*	*N/A*

Document History

Revision	Description	Changes Made By	Date
0.5	*Initial Draft*	*Chaintroopers*	*March 29th, 2024*
1.0	*First Version*	*Chaintroopers*	*March 29th, 2024*
1.1	*Retet Report*	*Chaintroopers*	*April 30th, 2024*