Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Ethereum-Aleo Bridge

Veridise Inc.

April 25, 2024

► **Prepared For:**

Venture23
https://venture23.xyz/

► **Prepared By:**

Benjamin Mariano
Jacob Van Geffen

► **Contact Us:** contact@veridise.com

► **Version History:**

Apr. 25, 2024 V2
Mar. 29, 2024 V1

Contents

1 Executive Summary

From Feb. 19, 2024 to Mar. 19, 2024, Venture23 engaged Veridise to review the security of their Ethereum-Aleo Bridge. The review covered the implementation of a bridge between Ethereum and Aleo. Veridise conducted the assessment over 8 person-weeks, with 2 engineers reviewing code over 4 weeks from commits `c77637b-c424b4d`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Ethereum-Aleo Bridge developers provided the source code of the Ethereum-Aleo Bridge contracts for review. The source code appears to be mostly original code written by the Ethereum-Aleo Bridge developers. It contains documentation in the form of READMEs and documentation comments on functions and storage variables. To facilitate the Veridise auditors' understanding of the code, the Ethereum-Aleo Bridge developers also shared some high-level documentation explaining how the different larger systems fit together.

The source code contained a test suite, which the Veridise auditors noted tested each part of the codebase well in isolation.

Summary of issues detected. The audit uncovered 38 issues, 5 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, the critical issues involved insufficient access control which could allow sending of arbitrary packets (V-V23-VUL-001,V-V23-VUL-005) and denial of service (V-V23-VUL-003), as well as bad signature checking which could allow an attacker to take ownership of the bridge (V-V23-VUL-002) or remove a token service (V-V23-VUL-003). The Veridise auditors also identified 4 medium-severity issues, including incorrect logic for thresholds (V-V23-VUL-006) and bad pruning mechanisms for stored packets in the attestor (V-V23-VUL-008) as well as 4 warnings and 18 informational findings.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Ethereum-Aleo Bridge	`c77637b-c424b4d`	Solidity,Leo,Go	Ethereum,Aleo
Database Service	`8c53a0e`	Typescript	-

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Feb. 19 - Mar. 19, 2024	Manual & Tools	2	8 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	5	3	5
High-Severity Issues	0	0	0
Medium-Severity Issues	4	2	4
Low-Severity Issues	7	6	6
Warning-Severity Issues	4	3	4
Informational-Severity Issues	18	13	14
TOTAL	38	27	33

Table 2.4: Category Breakdown.

Name	Number
Maintainability	13
Logic Error	9
Data Validation	7
Access Control	3
Information Leakage	1
Denial of Service	1
Resource Leak	1
Dependency	1
Events	1
Library Usage	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the project's smart contracts and attestor logic. In our audit, we sought to answer questions such as:

- Can incorrect packets be emitted by smart contracts and signed by attestors?
- Can attestor signatures be forged or otherwise circumvented?
- Are attestors vulnerable to denial-of-service attacks?
- Can arbitrary users claim ownership of the bridge?
- Is the bridge vulnerable to replay attacks?
- Is state appropriately maintained/updated across both chains?
- Are credentials for attestors maintained securely?
- Can event reordering be used to disrupt the inteded behavior of the bridge?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of our custom Aleo static analyzer. This static analyzer is designed to find instances of common Aleo smart contract vulnerabilities, such as information leaks and uninitialized variables.

Scope. The scope of this audit is limited to the `solidity`, `aleo`, and `attestor` folders, as well as the full `dbservice` repository. The `solidity` and `aleo` folders contain smart contracts for their respective blockchains, while the `attestor` folder contains a Golang project that is run by each attestor for the bridge. The `dbservice` repository defines services for adding both signatures and unconfirmed packets to a database and is written in Typescript.

Methodology. Veridise auditors first met with the developers, who gave a high-level walkthrough of the code base. They then began a manual audit of the code, considering both the smart contracts' & attestor logic's isolated correctness, as well as the correctness of how the attestor logic worked with both sets of smart contracts. Throughout the audit, the Veridise auditors regularly met with the Ethereum-Aleo Bridge developers to ask questions about the code and to share any issues found.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-V23-VUL-001	Arbitrary messages can be sent	Critical	Fixed
V-V23-VUL-002	Anyone can take ownership of the bridge	Critical	Fixed
V-V23-VUL-003	Anyone can remove a token service	Critical	Fixed
V-V23-VUL-004	Attacker can create signature database entry	Critical	Acknowledged
V-V23-VUL-005	Arbitrary users can add unconfirmed packets	Critical	Acknowledged
V-V23-VUL-006	Incorrect logic for low thresholds	Medium	Acknowledged
V-V23-VUL-007	Credentials stored as plain text	Medium	Acknowledged
V-V23-VUL-008	Index never incremented for pruning	Medium	Fixed
V-V23-VUL-009	Add rate limiting to DB service	Medium	Fixed
V-V23-VUL-010	Quorum threshold initialized to zero	Low	Fixed
V-V23-VUL-011	Missing check for valid threshold	Low	Fixed
V-V23-VUL-012	Disable vote updating on executed proposals	Low	Fixed
V-V23-VUL-013	Missing response body close after HTTP request	Low	Fixed
V-V23-VUL-014	Arbitrary length Aleo addresses	Low	Fixed
V-V23-VUL-015	Npm audit issues	Low	Fixed
V-V23-VUL-016	Old unconfirmed packets fetched first	Low	Intended Behavior
V-V23-VUL-017	Updating unsupported token info	Warning	Fixed
V-V23-VUL-018	No verification that token is supported on release	Warning	Acknowledged
V-V23-VUL-019	Missing bounds check	Warning	Fixed
V-V23-VUL-020	Non-restrictive types	Warning	Fixed
V-V23-VUL-021	Unnecessary unchecked blocks	Info	Fixed
V-V23-VUL-022	Unclear events on quorum threshold update	Info	Fixed
V-V23-VUL-023	Unnecessary internal function	Info	Fixed
V-V23-VUL-024	Unnecessary function arguments	Info	Fixed
V-V23-VUL-025	Unnecessary Getter Functions	Info	Fixed
V-V23-VUL-026	Missing non-zero checks on transaction parameters	Info	Acknowledged
V-V23-VUL-027	Remove commented code	Info	Fixed
V-V23-VUL-028	Unused function _splitSignature	Info	Fixed
V-V23-VUL-029	Unnecessary cast to address	Info	Fixed
V-V23-VUL-030	Refactor get_valid_unique_address_count method	Info	Intended Behavior
V-V23-VUL-031	Unnecessary check for equal vote counts	Info	Intended Behavior
V-V23-VUL-032	Can make stronger voting check	Info	Intended Behavior
V-V23-VUL-033	Unnecessary function argument for chain ID	Info	Fixed
V-V23-VUL-034	Typos and incorrect comments	Info	Fixed
V-V23-VUL-035	Unused mapping proposal_vote_counts	Info	Intended Behavior
V-V23-VUL-036	Unnecessary code and typos in attestor	Info	Fixed
V-V23-VUL-037	Use express best practices	Info	Fixed
V-V23-VUL-038	Typos and unused code in database service	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-V23-VUL-001: Arbitrary messages can be sent

Severity	Critical	**Commit**	c77637b
Type	Access Control	**Status**	Fixed
File(s)	base/bridge/OutgoingPacketManagerImpl.sol		
Location(s)	_sendMessage		
Confirmed Fix At	N/A		

The function `_sendMessage` is used to first add metadata to a packet (including sequence number and version number) and then "send" the message by emitting a `PacketDispatched` event as shown below.

```
function _sendMessage(PacketLibrary.OutPacket memory packet) public virtual {
   packet.version = 1;
   packet.sequence = ++sequence;
   outgoingPackets[packet.sequence] = packet.hash();
   emit PacketDispatched(packet);
}
```

Snippet 4.1: `_sendMessage` implementation from `outgoingPacketManagerImpl`

This function is marked `public`, meaning any user can call this function to send an arbitrary message.

Impact Messages are intended to be sent via the `sendMessage` function in `Bridge.sol`, which includes a number of checks that the packet is valid, including checks that the destination chain is supported and the contract is not paused. In addition to these checks, the most important check is that the only caller of `sendMessage` should be the `TokenService` contract, which only calls `sendMessage` after transfers of either ETH or ERC20 tokens (see `transfer` functions in `TokenService`). However, none of the checks from `sendMessage` are performed in `_sendMessage`. This means an attacker can send any message they want, including ones that indicate a transfer which they never actually performed. This could allow an attack to steal funds.

Proof of Concept Below is a test case added to `001.Bridge.test.js` which shows that calling `_sendMessage` doesn't revert, even if passed an invalid chain ID.

```
it('VERIDISE: doesnt revert when calling _sendMessage with unknown destination
    chainId', async () => {
    const unknowndestChainId = 3;
    const outPacket = [
        1,
        1,
        [1, ethers.Wallet.createRandom().address],
        [unknowndestChainId, "
    aleo1fg8y0ax9g0yhahrknngzwxkpcf7ejy3mm6cent4mmtwew5ueps8s6jzl27"], [ethers.Wallet
    .createRandom().address, "
    aleo1fg8y0ax9g0yhahrknngzwxkpcf7ejy3mm6cent4mmtwew5ueps8s6jzl27", 10, "
    aleo1fg8y0ax9g0yhahrknngzwxkpcf7ejy3mm6cent4mmtwew5ueps8s6jzl27"],
```

```
        100
    ];
    await proxiedV1.connect(tokenService).sendMessage(outPacket);
});
```

Recommendation Change the visibility of the function to `internal`.

4.1.2 V-V23-VUL-002: Anyone can take ownership of the bridge

Severity	Critical	Commit	c77637b
Type	Logic Error	Status	Fixed
File(s)	aleo/programs/council_v0003.leo		
Location(s)	tb_transfer_ownership		
Confirmed Fix At	N/A		

The function `tb_transfer_ownership` is used to transfer ownership of the bridge. The `finalize` function is as follows.

```
finalize tb_transfer_ownership(proposal_hash: field, voters: [address; 5], vote_keys:
     [field; 5], votes: u8) {
    // Ensure that the votes are from valid members
    for i: u8 in 0u8..SUPPORTED_THRESHOLD {
        assert(Mapping::contains(members, voters[i]));
    }

    // Get the threshold
    let threshold: u8 = Mapping::get(settings, THRESHOLD_INDEX);

    assert(votes >= threshold);

    // Ensure that the proposal has not been executed
    assert(!Mapping::contains(proposal_executed, proposal_hash));

    // Mark the proposal as executed
    Mapping::set(proposal_executed, proposal_hash, true);
}
```

Snippet 4.2: Implementation of `finalize tb_transfer_ownership`

The function does not use the `vote_keys` argument, which is where the actual votes are stored. Therefore, the votes of the members are never actually checked.

Impact Anyone can call this function and as long as they pass in an array of valid voters and the length of this array is greater than the threshold, they can transfer ownership of the bridge.

Recommendation Add the check of `vote_keys[i]`.

4.1.3 V-V23-VUL-003: Anyone can remove a token service

Severity	Critical	Commit	c77637b
Type	Logic Error	Status	Fixed
File(s)	aleo/programs/council_v0003.leo		
Location(s)	tb_remove_service()		
Confirmed Fix At	N/A		

The function `tb_remove_service` is used to remove a token service. The `finalize` function is as follows.

```
finalize tb_remove_service(proposal_hash: field, voters: [address; 5], vote_keys: [
    field; 5], votes: u8) {
    // Ensure that the votes are from valid members
    for i: u8 in 0u8..SUPPORTED_THRESHOLD {
        assert(Mapping::contains(members, voters[i]));
    }

    // Get the threshold
    let threshold: u8 = Mapping::get(settings, THRESHOLD_INDEX);

    assert(votes >= threshold);

    // Ensure that the proposal has not been executed
    assert(!Mapping::contains(proposal_executed, proposal_hash));

    // Mark the proposal as executed
    Mapping::set(proposal_executed, proposal_hash, true);
}
```

Snippet 4.3: Implementation of `finalize tb_remove_service`

The function does not use the `vote_keys` argument, which is where the actual votes are stored. Therefore, the votes of the members are never actually checked.

Impact Anyone can call this function and as long as they pass in an array of valid voters and the length of this array is greater than the threshold, they can remove a token service.

Recommendation Add the check of `vote_keys[i]`.

4.1.4 V-V23-VUL-004: Attacker can create signature database entry

Severity	Critical	Commit	08f5a8e
Type	Access Control	Status	Acknowledged
File(s)		N/A	
Location(s)		N/A	
Confirmed Fix At		N/A	

Signatures are added to the database by making POST requests, per the following code snippet which adds the handler `this.chainSignatureController.create`:

```
  private initializeRoutes() {
    ...
    this.router.post(
      `${this.path}`,
      validationMiddleware(ChainSignatureDto, 'body'),
      signatureValidatorMiddleware,
      this.chainSignatureController.create,
    );
  }
```

Snippet 4.4: Snippet from `initializeRoutes()` in `chainSignature.route.ts`

The `validationMiddleware` ensures that the body of the request adheres to the format specified in `ChainSignatureDto` and the signature validation ensures that the signature, packet hash, and specified attestor all match with the following logic:

```
const signatureValidatorMiddleware = (req: Request, res: Response, next: NextFunction
    ) => {
  try {
    const data = req.body;
    const { attestorSigner, packetHash, signature } = data;
    let isSignatureValid = false;

        ...

    if (attestorSigner.startsWith('aleo')) {
      isSignatureValid = signVerify(signature, attestorSigner, packetHash);
    } else {
      const signer = ethers.recoverAddress(packetHash, signature);

      isSignatureValid = compareAddress(signer, attestorSigner);
    }

    if (!isSignatureValid) throw new HttpException(403, 'Packet validation failed');
    next();
  } catch (error) {
    next(error);
  }
};
```

Snippet 4.5: Snippet from `signatureValidatorMiddleware()`

At no point is it verified that the sender of such a request is a trusted sender — thus, any attacker

can create signature entries in the database.

Impact Because of the signature validator middleware, even an attacker who is attempting to post a signature must post a "valid" signature, in the sense that the signature must be validated to match the given attestor and packet hash. However, an attacker can still perform the following attacks:

1. Fill the database with bogus signatures
2. Block legitimate signatures from being added to the database
3. Delete valid unconfirmed packet requests before they are processed

These attacks all take advantage of the fact that packets are stored according to the following `unique` index in the database:

```
chainSignatureSchema.index(
  {
    sourceChainId: 1,
    destChainId: 1,
    sequence: 1,
    attestorSigner: 1,
  },
  { unique: true },
);
```

Snippet 4.6: Snippet from `chainSignature.model.ts`

When creating entries in the database, the following code is used:

```
public async create(chainData: ChainSignature): Promise<ChainSignature> {
  // Create
  const packetExists = await this.chainSignature.findOne({
    destChainId: chainData.destChainId,
    sourceChainId: chainData.sourceChainId,
    attestorSigner: chainData.attestorSigner,
    sequence: chainData.sequence,
  });

  if (packetExists) {
    // Throw error with success status
    throw new HttpException(201, 'Duplicate packet');
  }

  const createdData = await this.chainSignature.create(chainData);

  return createdData;
}
```

Snippet 4.7: Snippet from `create()` in `chainSignature.service.ts`

The `create` call creates a new entry by first checking if one exists, using the unique index including the destination chain ID, source chain ID, attestor, and sequence number. If an entry for this signature already exists, an exception is thrown. Otherwise, the signature is added to the database.

Bogus Database Entries The issue is that there is never any verification that the destination/-source chain IDs nor the sequence number actually match the signature. The caller can simply make up these values. This means an attacker can take a valid signature, attestor, and packet hash from the database, change the source/destination chain IDs and/or sequence number, and submit it. This will pass all validation steps and will be added as a new element of the database. Thus, an attacker can enter as many bogus entries as they want.

Blocking Valid Signatures Furthermore, due to the check in `create` which throws an exception if the packet exists, an attacker can use this to block legitimate packets from being entered. For instance, an attacker can use the technique described above to create a bogus entry for sequence number `X`, which will get entered and block the real entry for sequence number `X` whenever it is attempted to be added.

Deleting Valid Unconfirmed Packets On creation of a signature entry on the database, the corresponding unconfirmed packet entry in the database is deleted by the following code:

```
public create = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const isUnConfirmedPacket = req.query.unconfirmed == 'true';

    const reqData: ChainSignatureDto = req.body;
    await this.chainSignatureService.create(reqData);
    if (isUnConfirmedPacket) await this.unconfirmedPacketService.delete(reqData);

    res.status(201).json({ message: 'create' });
  } catch (error) {
    next(error);
  }
};
```

Snippet 4.8: Snippet from `create()` in `chainSignature.controller.ts`

This code deletes the corresponding unconfirmed packet entry from the `unconfirmedPacketService` . As mentioned previously, an attacker can make up the source/destination chain IDs and the sequence number. Similarly, there is no validation of the `unconfirmed` field of the request. Thus, an attacker can create a bogus signature whose source/chain IDs and sequence number correspond to a real `unconfirmed` packet. After the signature is added to the database, the unconfirmed packet will be deleted and blocked from future entry in the database.

Recommendation Add in validation that attestors are the only users able to add signatures to the database. It should be noted that even in the case that only attestors can write signatures, any individual attestor could perform the attacks described above. As a result, either attestors should be carefully chosen/trusted entities or further validation will be needed on all database entries.

Developer Response The developers are planning to implement a solution where the attestors and database service will communicate through MTLS. In that way, writes to this database will no longer be accessible publically and all read requests will be throttled through nginx.

4.1.5 V-V23-VUL-005: Arbitrary users can add unconfirmed packets

Severity	Critical	**Commit**	08f5a8e
Type	Access Control	**Status**	Acknowledged
File(s)		N/A	
Location(s)		N/A	
Confirmed Fix At		N/A	

There is no validation that posting new unconfirmed packets is done through a trusted user.

```
1 private initializeRoutes() {
2   this.router.get(`${this.path}`, /*validationMiddleware(UnconfirmedPacketDto, 'body
      '),*/ this.unconfirmedPacketController.get);
3   this.router.post(`${this.path}`, validationMiddleware(UnconfirmedPacketDto, 'body')
      , this.unconfirmedPacketController.create);
4 }
```

Snippet 4.9: Snippet from the `unconfirmedPacket` route initialization, which only validates that the unconfirmed packet is well-formed.

Since unconfirmed packets do not undergo any additional checks on the part of attestors or the per-blockchain smart contracts, this essentially allows arbitrary users to add any packets they wish.

Impact Since no validation is done on the part of the attestor, the power to add arbitrary packets allows users to also get those packets signed. This means that attackers could construct an unconfirmed packet with any transaction hash they would like, get the packet signed, and then propagate the transaction across the bridge.

By using this exploit, attackers could construct a bogus transaction that adds funds into an attacker's account. The attacker could then spend the funds arbitrarily before any altruistic party has the opportunity to delete the faulty packet.

Recommendation Only allow a small number of trusted origins to add unconfirmed packets. This would allow for a separate review process through which unconfirmed packets can be validated against the actual packet events emitted by the blockchain (or against the packet mapping in the case of the Aleo blockchain). Note that this review process may involve some degree of manual inspection.

Following this, we recommend requiring multiple attestor signatures to *confirm* unconfirmed packets. This separate process would ensure that a majority of attestors agree with the result of the separate review process.

Developer Response The developers are planning to implement a solution using MTLS. In particular, admin certificates will be required to post unconfirmed packets to the database service.

4.1.6 V-V23-VUL-006: Incorrect logic for low thresholds

Severity	Medium	**Commit**	c77637b
Type	Logic Error	**Status**	Acknowledged
File(s)	`token_bridge_v0003.leo`, `ConsumedPacketManagerImpl.sol`		
Location(s)	_checkSignatures(), get_majority_count()		
Confirmed Fix At	N/A		

On both the Aleo and Ethereum chains, when a packet is received, the consumer of the packet can choose which signatures to pass in. The logic on both will choose whichever vote-type ("yay" or "nay") got the most votes, and will allow the packet to be consumed accordingly if the threshold for that count is reached. This works in the assumed case of a threshold of `3` with `5` attestors. However, if the threshold is set to less than half the attestors, the logic no longer works as expected.

Impact If the threshold is set to less than half the number of attestors and a packet gets greater than or equal to `threshold` votes for both yay and nay, the caller can simply choose their desired outcome by passing in the votes with their desired outcome.

Recommendation Add a check that the threshold is always greater than half the number of attestors.

Developer Response The developers have acknowledged the issue but at this time have decided that it will be assumed to be an operational requirement of the system. In particular, the threshold will always be more than 51%. The council will act correctly and the minimum threshold will always be fulfilled. The current configuration allows more flexibility, so will be kept.

4.1.7 V-V23-VUL-007: Credentials stored as plain text

Severity	Medium	**Commit**	c424b4d
Type	Information Leakage	**Status**	Acknowledged
File(s)		N/A	
Location(s)		N/A	
Confirmed Fix At		N/A	

Throughout the `chainService` attestor codebase, the user's username and password are stored as plaintext.

```
...

signing_service:
  ...
  username: "username"
  password: "password"
```

Snippet 4.10: Username and password fields from `config.yaml`.

```
func SetupSigner(cfg *config.SigningServiceConfig) error {
    logger.GetLogger().Info("Setting up signer",
        zap.String("username", cfg.Username),
        zap.String("password", cfg.Password),
        ...
}
```

Snippet 4.11: Username and password are written directly to logs in `sign.go`.

Additionally, password strings are compared directly when authenticating users.

```
func registerHandlers() {
    http.HandleFunc("/sign", func(w http.ResponseWriter, r *http.Request) {
        r.Close = true

        username, password, _ := r.BasicAuth()
        cfgUser, cfgPass := config.GetUsernamePassword()

        if username != cfgUser || password != cfgPass {
            w.WriteHeader(http.StatusForbidden)
            return
        }

        ...
    }
}
```

Snippet 4.12: Direct password comparisons in `serve.go`.

Impact Storing user passwords as plain-text in config files and logs increases the risk of compromising those credentials. While users could reasonably keep the config and logs secret, keeping passwords in plain-text unnecessarily adds a surface of attack of which users must be

cognizant. Additionally, comparing plain-text password strings directly can be vulnerable to side-channel attacks.

Recommendation Avoid storing the plain-text credentials of users. Instead, use a service like Vault (`https://www.hashicorp.com/products/vault`) or AWS secrets manager (`https://aws.amazon.com/secrets-manager/`). When comparing passwords, hashes of the passwords should be compared instead of comparing the passwords directly.

Developer Response Plaintext logging of credentials has been removed. For password verification, we are still exploring better methods for performing authentication.

4.1.8 V-V23-VUL-008: Index never incremented for pruning

Severity	Medium	**Commit**	c424b4d
Type	Logic Error	**Status**	Fixed
File(s)	`attestor/chainService/chain/ethereum/client.go`		
Location(s)	pruneBaseSeqNum()		
Confirmed Fix At	N/A		

The `pruneBaseSeqNum` function is responsible for iterating through the different `baseSeqNamespaces` checking for potentially missed packets and re-sending them. To transition to a new namespace, it is supposed to increment a variable named `index` — this variable is never incremented.

Impact Not incrementing this variable means that, if there is more than one namespace, the pruning and re-sending procedure will never be activated for any namespace other than the first.

Recommendation Add an increment to `index`.

4.1.9 V-V23-VUL-009: Add rate limiting to DB service

Severity	Medium	**Commit**	08f5a8e
Type	Denial of Service	**Status**	Fixed
File(s)		N/A	
Location(s)		N/A	
Confirmed Fix At		N/A	

Rate limiting is a common technique for reducing repeated actions from the same user. For servers and databases, using rate limiting is critical for avoiding denial of service attacks which flood the application with bogus information, limiting the bandwidth to deal with actual requests.

Impact Without appropriate rate limiting, the application may be vulnerable to denial of service attacks.

Recommendation Add in rate limiting.

4.1.10 V-V23-VUL-010: Quorum threshold initialized to zero

Severity	Low	**Commit**	c77637b
Type	Logic Error	**Status**	Fixed
File(s)	`bridge/AttestorManager.sol`		
Location(s)	N/A		
Confirmed Fix At	N/A		

The quorum threshold is initialized to `0` and can be updated by calling `updateQuorum`. When the threshold is `0`, no signatures are required to vote down a given packet (see `_checkSignatures` implementation). Thus, on initialization of contracts, a malicious user can vote down valid packets until this threshold value is set.

Impact A malicious user can vote down packets without any attestor input until the threshold is updated to be greater than `0`.

Recommendation Set the threshold to something greater than `0` during initialization and do not allow the threshold to be set to `0` when updating it.

4.1.11 V-V23-VUL-011: Missing check for valid threshold

Severity	Low	Commit	c77637b
Type	Data Validation	Status	Fixed
File(s)	aleo/programs/token_bridge_v0003.leo		
Location(s)	remove_attestor_tb(), add_attestor_tb()		
Confirmed Fix At	N/A		

Unlike `update_threshold_tb`, there is no check in `remove_attestor_tb` that the `new_threshold` is no more than the total number of attestors after the update. The same check is missing from `add_attestor_tb`.

Impact It is possible to accidentally update the threshold to be greater than the total number of attestors, which makes passing any proposal impossible.

Recommendation Add a check that the new threshold is less than or equal to the total number of attestors.

4.1.12 V-V23-VUL-012: Disable vote updating on executed proposals

Severity	Low	**Commit**	c77637b
Type	Logic Error	**Status**	Fixed
File(s)	`aleo/programs/council_v0003.leo`		
Location(s)	update_vote()		
Confirmed Fix At	N/A		

The function `update_vote` allows a member to update their vote on a proposal which they have already voted on. However, there is no check that this proposal is still active (i.e., that the proposal has not yet been executed).

Impact This could lead to confusing accounting where the results of a vote are obscured by votes that are updated after the proposal is executed.

Recommendation Disable vote updating once a proposal has been executed.

4.1.13 V-V23-VUL-013: Missing response body close after HTTP request

Severity	Low	**Commit**	c424b4d
Type	Resource Leak	**Status**	Fixed
File(s)	`sign.go`		
Location(s)	dial		
Confirmed Fix At	N/A		

The documentation for Go's HTTP client (`https://go.dev/src/net/http/response.go`) states that whenever the error returned by executing an HTTP request is `nil`, the body of the response must be closed by the caller. However, within the `dial` function in `sign.go`, the response body is never closed.

```
1  func dial(u string) error {
2      ...
3
4      resp, err := http.DefaultClient.Do(req)
5      if err != nil {
6          return err
7      }
8
9      if resp.StatusCode < 400 || resp.StatusCode > 499 {
10         return fmt.Errorf("expected status code 4xx, got %d", resp.StatusCode)
11     }
12
13     return nil
14 }
```

Snippet 4.13: Snippet from `dial()`

Impact Failing to close the body of the response can lead to a resource leakage. However, since `dial` should only be called once in the lifespan of the service, the impact of this leak should be minimal.

Recommendation Call `resp.Body.Close()` whenever `err` is non-nil.

4.1.14 V-V23-VUL-014: Arbitrary length Aleo addresses

Severity	Low	**Commit**	c424b4d
Type	Data Validation	**Status**	Fixed
File(s)	`solidity/contracts/main/tokenservice/TokenService.sol`		
Location(s)	transfer()		
Confirmed Fix At	N/A		

In the function `transfer`, the argument `receiver` is a string that is meant to represent an Aleo address. There are currently no checks that this is a valid Aleo address, including specifically no checks on the length of this string.

Impact A malicious user could send packets with extremely long `receiver` strings which are clearly invalid (as Aleo addresses are known to be 63 characters long). These large packets could take a long time for the attestors to handle, slowing down the bridge unnecessarily.

Recommendation Because the length of Aleo addresses is known, add a check to the `transfer` functions which verifies that the length of the `receiver` Aleo address is exactly 63 characters long.

4.1.15 V-V23-VUL-015: Npm audit issues

Severity	Low	**Commit**	08f5a8e
Type	Dependency	**Status**	Fixed
File(s)		N/A	
Location(s)		N/A	
Confirmed Fix At		N/A	

On running `npm audit`, the following summary was reported:

```
# npm audit report

...

8 vulnerabilities (6 moderate, 2 critical)

To address all issues, run:
  npm audit fix
```

Snippet 4.14: Output from `npm audit`

Impact Some of the vulnerabilities are reported as potentially critical, including command injection and sandbox escaping.

Recommendation Run `npm audit fix` to address the issues.

4.1.16 V-V23-VUL-016: Old unconfirmed packets fetched first

Severity	Low	Commit	08f5a8e
Type	Logic Error	Status	Intended Behavior
File(s)	`services/unconfirmedPacket.service.ts`		
Location(s)	findByAttestor()		
Confirmed Fix At	N/A		

When fetching unconfirmed packets, the following function is used which fetches the first `limit` number of unconfirmed packets associated with a particular `attestor`:

```
public async findByAttestor(attestor: string, limit: number): Promise<
    UnconfirmedPacket[]> {
  const DEAFULT_LIMIT = 1000;

  if (isEmpty(attestor)) throw new HttpException(400, 'Attestor is empty');

  const docLimit = isNaN(Number(limit)) ? DEAFULT_LIMIT : limit || DEAFULT_LIMIT;

  const unconfirmedPacket: UnconfirmedPacket[] = await this.unconfirmedPacket
    .find({
      attestorSigner: attestor,
    })
    .sort({
      createdAt: 1,
    })
    .limit(docLimit);

  return unconfirmedPacket;
}
```

Snippet 4.15: Snippet from `findByAttestor()`

The function fetches all entries with `find`, sorts them by their creation date using `sort`, and finally cuts off the first `docLimit` entries with `limit`.

The issue is with `.sort({createAt: 1})`, which sorts the entries in *ascending* order. This means that packets will be retrieved starting with the oldest. As a result, if the number of entries associated with a particular attestor is greater than the limit, the newest entries will be left out.

Impact Users may call this and not realize that the newest entries are left out. This could lead to missed `unconfirmed` packets, especially if the reading from this database is automated by the attestors.

Recommendation Make the order *descending* by changing `.sort({createAt: 1})` to `.sort({createAt: -1})`.

Developer Response This is actually intended behavior. Because unconfirmed packets are periodically deleted, subsequent calls will reveal newer unconfirmed packets as older ones are processed.

4.1.17 V-V23-VUL-017: Updating unsupported token info

Severity	Warning	**Commit**	c77637b
Type	Data Validation	**Status**	Fixed
File(s)	`base/tokenservice/TokenSupport.sol`		
Location(s)	updateVault()		
Confirmed Fix At	N/A		

The function `updateVault` allows the owner of the contract to update the `vault` address for a supported token. However, there is no check that the token whose vault is being updated is actually a supported token.

Impact This could lead to mistakes where the vault for some not-yet-supported token is updated to an incorrect vault address for that vault. While this vault address would be updated when the token is added via `addToken`, the successful call to `updateVault` would still yield a potentially confusing event.

Recommendation Add a check that `updateVault` is only called on a supported token.

4.1.18 V-V23-VUL-018: No verification that token is supported on release

Severity	Warning	**Commit**	c77637b
Type	Data Validation	**Status**	Acknowledged
File(s)	`main/Holding.sol`		
Location(s)	release()		
Confirmed Fix At	N/A		

The `release` function in the `Holding` contract is used to allow users to reclaim their funds after they were locked and subsequently unlocked by the council.

```
function release(address user, address token) external virtual checkZeroAddress(token
    ){
    require(token != ETH_TOKEN, "Holding: eth token Address");
    uint256 amount = _release(user, token);
    require(IIERC20(token).transfer(user, amount), "Holding: erc20 release failed");
}
```

Snippet 4.16: `release` implementation.

`release` does not check that the token requested is in fact a token which is supported by the protocol (in fact, no such checking infrastructure exists in this contract).

Impact A malicious user may request a release on a token that is not supported. This is not disastrous, as the `amount` released is the amount for the token in the `unlocked` mapping (see `_release` implementation in the code), which would be `0` unless the council explicitly unlocked the requested amount on the token (after it was already locked). However, there are no checks that the `amount` is greater than `0`, meaning this function could be spammed to create bogus `Released` events (which are emitted in `_release`).

Recommendation Add checking for supported tokens in the `Holding` contract and add a `0` check on the `release` amount.

Developer Response The developers have acknowledged the issue but have decided at this time bogus `Released` events are not a major concern.The main reason not to make this change is to avoid unwanted dependencies. The council will manually check and release tokens, so the assumption is that the council will act correctly.

4.1.19 V-V23-VUL-019: Missing bounds check

Severity	Warning	Commit	c77637b
Type	Data Validation	Status	Fixed
File(s)	`token_service_v0003.leo`		
Location(s)	update_min_transfer_ts, update_max_transfer_ts		
Confirmed Fix At	N/A		

When adding a token with the token service, the min and max transfer values are validated to ensure that the min transfer is no greater than the max transfer.

```
transition add_token_ts(
    ...
    public min_transfer: u128,
    public max_transfer: u128,
    ...
) {
    assert(max_transfer >= min_transfer);
    ...
    return then finalize(...);
}
```

Snippet 4.17: Snippet from `add_token_ts()`

However, when updating the min or max transfer values for an existing token, no such validation is performed.

Impact If token contract owners are allowed to erroneously set their min and max transfer values to invalidate the conidtion `max_transfer >= min_transfer`, then no tokens can be sent. This is because no amount will satisfy the following assertions in `token_send`:

```
transition token_send(...) {
    ...
    let min_amount: u128 = Mapping::get(min_transfers, wrapped_addr);
    assert(amount >= min_amount);

    let max_amount: u128 = Mapping::get(max_transfers, wrapped_addr);
    assert(amount <= max_amount);
    ...
}
```

Snippet 4.18: Snippet from `token_send()`

Recommendation Add the check `assert(max_transfer >= min_transfer)` to any functions that can update the min and max transfer values for a token. Particularly, this includes `update_min_transfer_ts` and `update_max_transfer_ts`.

4.1.20 V-V23-VUL-020: Non-restrictive types

Severity	Warning	Commit	08f5a8e
Type	Data Validation	Status	Fixed
File(s)		N/A	
Location(s)		N/A	
Confirmed Fix At		N/A	

Several type specifications within the DB service lack sufficient strictness to validate parameters.

1. Within `utils.ts`, the `decodeNetworkChainId` can be restricted to only accept `bigint` arguments (instead of `number | bigint`).

```
const verifyNumber = (num: number) => {
  if (parseInt(num.toString()).toString() !== num.toString()) {
    throw Error('Error representing chainId as number. Pass as a BigInt');
  }
};
```

Snippet 4.19: Validation for `decodeNetworkChainId` arguments, which passes even when `num` is `Infinity`.

2. Within `chainSignature.dto.ts`, the `signature` parameter should be made non-optional.
3. Within `unconfirmedPacket.dto.ts`, `attestorSigner` should have type `0x${string} | aleo${string}`.

Impact The lack of data validation through sufficiently strict types could allow for illegal representations of the `ChainSignature` or `UnconfirmedPacket` objects to make their way onto the database.

4.1.21 V-V23-VUL-021: Unnecessary unchecked blocks

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	`main/Holding.sol`, `base/bridge/ConsumedPacketManagerImpl.sol`		
Location(s)	_release(), _checkSignatures()		
Confirmed Fix At	N/A		

There are two different places where an `unchecked` block is used unnecessarily. The first is in the function `_release` in `Holding.sol`.

```
unchecked {
    unlocked[user][token] = 0;
}
```

Snippet 4.20: Snippet from `_release` in `Holding.sol`

The second is in `_checkSignatures` in `ConsumedPacketManagerImpl.sol`.

```
unchecked {
    if(yeas > nays && yeas >= threshold) return PacketLibrary.Vote.YEA;
    else if(nays >= threshold) return PacketLibrary.Vote.NAY;
}
```

Snippet 4.21: Snippet from `_checkSignatures` in `ConsumedPacketManagerImpl.sol`

In both cases, there is no arithmetic performed in the `unchecked` block and therefore it does not do anything.

Impact This could confuse future programmers and makes the code harder to understand.

Recommendation Remove the unnecessary `unchecked` blocks.

4.1.22 V-V23-VUL-022: Unclear events on quorum threshold update

Severity	Info	**Commit**	c77637b
Type	Events	**Status**	Fixed
File(s)	`base/bridge/AttestorManager.sol`		
Location(s)	addAttestor(), removeAttestor()		
Confirmed Fix At	N/A		

In this contract, calls to `addAttestor`, `removeAttestor`, and `updateQuorum` all change the `quorumRequired`. Confusingly, there is an event `QuorumUpdated` which records the new and old quorum, but only if the quorum is updated through `updateQuorum` (and not if updated through `addAttestor` or `removeAttestor`).

Impact This may lead to confusion for outside applications which rely on reading events.

Recommendation Call `updateQuorum` from `addAttestor` and `removeAttestor` and remove the extra entry from the `AttestorAdded` and `AttestorRemoved` events.

4.1.23 V-V23-VUL-023: Unnecessary internal function

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	`base/tokenservice/TokenSupport.sol`		
Location(s)	_addToken()		
Confirmed Fix At	N/A		

In `TokenSupport.sol` there is an external function `addToken` which is just a wrapper around an internal function `_addToken` with an additional `onlyOwner` check. The internal function `_addToken` is only called from `addToken` and thus the logic from `_addToken` can just be added to `addToken`.

Impact The use of the unnecessary internal function may confuse future developers and makes understanding the code a bit more difficult.

Recommendation Merge `_addToken` into `addToken`.

4.1.24 V-V23-VUL-024: Unnecessary function arguments

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	`base/tokenservice/TokenSupport.sol`		
Location(s)	removeToken(), enable(), disable()		
Confirmed Fix At	N/A		

The function `removeToken` takes as argument `_destChainId` which is compared to the contract's `destChainId` and is then added as an argument to the emitted event.

```
function removeToken(
    address tokenAddress,
    uint256 _destChainId
) external virtual onlyOwner {
    require(isSupportedToken(tokenAddress),"TokenSupport: token not supported");
    require(_destChainId == destChainId, "TokenSupport: target chain mismatch");
    emit TokenRemoved(tokenAddress, _destChainId);
    delete supportedTokens[tokenAddress];
}
```

Snippet 4.22: `removeToken` implementation

The argument `_destChainId` could simply be removed and replaced in the event with `destChainId` .

This same issue is also present for the functions `enable` and `disable`.

Impact This obscures the desired behavior of the function and makes future maintainability and understanding of the code more challenging.

Recommendation Remove the unnecessary function argument.

4.1.25 V-V23-VUL-025: Unnecessary Getter Functions

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	`base/tokenservice/VaultService.sol`		
Location(s)	token(), name()		
Confirmed Fix At	N/A		

The `VaultService` contract defines two `private` variables `_token_` and `_name_` and then adds two getters named `token()` and `name()` for each respectively. This can be succinctly achieved by making the variables `public` and removing the underscores from their name, as Solidity adds getters for `public` variables of the same name.

Impact This unnecessary code makes the contract larger than it needs to be and can confuse developers as to the intended behavior.

Recommendation Make the `public` variables as suggested.

4.1.26 V-V23-VUL-026: Missing non-zero checks on transaction parameters

Severity	Info	**Commit**	c77637b
Type	Data Validation	**Status**	Acknowledged
File(s)	`solidity/.../Holding.sol`, `solidity/.../TokenSupport.sol`		
Location(s)	lock(user, token, amount), lock(user), addToken()		
Confirmed Fix At	N/A		

Some transactions within `Holding.sol` and `TokenSupport.sol` are missing validation checks on parameter values. While some checks are present, the following transaction parameters are missing checks for zero values:

```
function lock(address user, address token, uint256 amount) external virtual
    checkZeroAddress(user){
    require(token != ETH_TOKEN, "Holding: eth token address");
    _lock(user,token,amount);
}
```

Snippet 4.23: Missing check for non-zero `amount` on `lock(user, token, amount)` in `Holding.sol`

```
function lock(address user) external virtual payable {
    require(msg.value > 0, "Holding: requires eth transfer");
    _lock(user, ETH_TOKEN, msg.value);
}
```

Snippet 4.24: Missing check for non-zero `user` on `lock(user)` in `Holding.sol`

```
function addToken(
    address tokenAddress,
    uint256 _destChainId,
    address vault,
    string memory destTokenAddress,
    string memory destTokenService,
    uint256 min,
    uint256 max
) external virtual onlyOwner {
    ....
}
```

Snippet 4.25: Missing check for non-zero `_vault` on `addToken(...)` in `TokenSupport.sol`

Impact While zero values for these parameters do not pose any serious security risk, they allow users to issue spurious transactions on the Holding contract.

Recommendation Add checks that force the transactions to revert when passed invalid zero values for the parameters described above.

Developer Response The developers implemented the suggested fixes for `Holding.sol`. At this time, they have elected to allow the zero address for vaults in `TokenSupport.sol`. The main

reason is that the vault is used to send funds to a high yield contract and acts only as temporary storage. The council sends to the vault and the vaults sends to the high yield contract. A vault for tokens that are not supported by the high yield contract is not necessary (e.g. USDT). In those cases, the vault address can safely be the zero address.

4.1.27 V-V23-VUL-027: Remove commented code

Severity	Info	Commit	c77637b
Type	Maintainability	Status	Fixed
File(s)	ConsumedPacketManagerImpl.sol, TokenSupport.sol, Holding.sol		
Location(s)	_checkSignatures(), enable(), _release()		
Confirmed Fix At	N/A		

There are several instances of code that has been commented out within the repository.

```
function _checkSignatures(bytes32 packetHash, bytes memory signatures, uint256
    threshold) internal view returns (PacketLibrary.Vote) {
    ...
    for(uint256 i = 0; i < threshold; i++) {
        // require(signatures[i].length == 65, "ConsumedPacketManagerImpl: invalid
    signature length");
        (v,r,s) = _signatureSplit(signatures, i);
        ...
    }
    ...
}
```

Snippet 4.26: From `ConsumedPacketManagerImpl.sol`

```
function enable(
    address tokenAddress,
    uint256 _destChainId
) external virtual onlyOwner {
    // require(tokenAddress != ZERO_ADDRESS, "Zero Address");
    ...
}
```

Snippet 4.27: From `TokenSupport.sol`

```
function _release(address user, address token) internal whenNotPaused nonReentrant
    checkZeroAddress(user) returns (uint256 amount) {
    // require(unlocked[user][token] >= amount, "Insufficient amount");
    ...
}
```

Snippet 4.28: From `Holding.sol`

Impact Code in comments hurts the readability of the code base overall.

Recommendation For maintainability reasons, we recommend removing commented code.

4.1.28 V-V23-VUL-028: Unused function _splitSignature

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	`solidity/contracts/base/bridge/ConsumedPacketManagerImpl.sol`		
Location(s)	_splitSignature()		
Confirmed Fix At	N/A		

`ConsumedPacketManagerImpl.sol` contains an unused function shown below.

```
/// @dev Splits a signature into its components (v, r, s)
/// @param sig The signature to be split
/// @return v The recovery id as part of the signature
/// @return r The R component of the signature
/// @return s The S component of the signature
function _splitSignature(bytes memory sig) internal pure returns (uint8 v, bytes32 r,
     bytes32 s) {
    ...
}
```

Snippet 4.29: Unused function `_splitSignature`

Impact Since this function is very similar to the `_signatureSplit` function, correctly maintaining the code may be difficult for developers in the future.

Recommendation Remove the unused `_splitSignature` function.

4.1.29 V-V23-VUL-029: Unnecessary cast to address

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	`solidity/.../TokenSupport.sol`		
Location(s)	updateVault()		
Confirmed Fix At	N/A		

The `updateVault` function within `TokenSupport.sol` unnecessarily casts the token vault to an address.

```
/// @notice Updates the vault address associated with a token, callable only by the
    owner
/// @param token The address of the token
/// @param _vault The new address of the vault
function updateVault(address token, address _vault) external virtual onlyOwner {
    address vault = address(supportedTokens[token].vault);
    ...
}
```

Snippet 4.30: Unnecessary cast to `address`

Since `supportedTokens` maps addresses to `Token` instances, this cast is not necessary.

```
struct Token {
    address tokenAddress;
    address vault;
    string destTokenAddress;
    string destTokenService;
    uint256 minValue;
    uint256 maxValue;
    bool enabled;
}
```

Snippet 4.31: The `vault` parameter of `Token` is type `address`

Impact Casting unnecessarily slightly diminishes code readability and may cause minor confusion for developers in the future.

Recommendation Remove the unnecessary cast to `address`.

4.1.30 V-V23-VUL-030: Refactor get_valid_unique_address_count method

Severity	Info	Commit	c77637b
Type	Maintainability	Status	Intended Behavior
File(s)	`council_v0003.leo`, `token_bridge_v0003.leo`, `utils.leo`		
Location(s)	get_valid_unique_address_count		
Confirmed Fix At	N/A		

Currently, `get_valid_unique_address_count` is defined in multiple files, including `countil_v0003.leo` and `token_bridge_v0003.leo`.

Additionally, the `get_valid_unique_address_count` logic could be simplified by using two nested for loops instead of a sequence of `if` statements.

```
function get_valid_unique_address_count(addresses: [address; 5]) -> u8 {
    let unique_addresses: u8 = 0u8;
    if (addresses[0u8] != ZERO_ADDRESS) {
        assert_neq(addresses[0u8], addresses[1u8]);
        assert_neq(addresses[0u8], addresses[2u8]);
        assert_neq(addresses[0u8], addresses[3u8]);
        assert_neq(addresses[0u8], addresses[4u8]);
        unique_addresses += 1u8;
    }

        ...
}
```

Snippet 4.32: Original version of `get_valid_unique_address_count`

Impact Defining `get_valid_unique_address_count` multiple times hurts the maintainability of the code, as any change to the function must be replicated at every definition. Additionally, simplifying the code by using nested for loops will make the code easier to understand for future developers.

Recommendation There are two ways that we suggest `get_valid_unique_address_count` be refactored:

1. Use two nested `for` loops to simplify the logic of checking for unique addresses.
2. Define `get_valid_unique_address_count` only once, and call that version from all files that use the function. Note that the version defined in `utils.leo` is currently not used.

The code block below describes the suggested revision for `get_valid_unique_address_count`.

```
function get_valid_unique_address_count(addresses: [address; 5]) -> u8 {
    let unique_addresses: u8 = 0u8;
    for i:u32 in 0u32..5u32 {
      if (addresses[i] != ZERO_ADDRESS) {
        for j:u32 in 0u32..5u32 {
          if (i < j) {
            assert_neq(addresses[i], addresses[j]);
          }
```



```
        }
        unique_addresses += 1u8;
      }
    }
}
```

Developer Response The developers have chosen to keep this implementation as an optimization to avoid unnecessary looping.

4.1.31 V-V23-VUL-031: Unnecessary check for equal vote counts

Severity	Info	**Commit**	c77637b
Type	Logic Error	**Status**	Intended Behavior
File(s)	`token_bridge_v0003.leo`		
Location(s)	get_majority_count		
Confirmed Fix At	N/A		

In `get_majority_count`, the program halts whenever the number of yay and nay votes is equal. However, given that some signers may be the zero address, the count may indeed be equal. Additionally, since the threshold can be updated to any value between 1 and 5, it is possible to have a valid vote count where the yays and nays are equal. In these cases, transitions that call `get_majority_count` will revert unnecessarily.

```
transition get_majority_count(packet_hash: field, signers: [address; 5], signs: [
    signature; 5]) -> (bool, u8) {
    let unique_signers: u8 = get_valid_unique_address_count(signers);
    let yay_count: u8 = 0u8;
    let nay_count: u8 = 0u8;

    ...
    for i: u8 in 0u8..SUPPORTED_THRESHOLD {
        if (signers[i] != ZERO_ADDRESS) {
            let yay: bool = signature::verify(signs[i], signers[i],
    packet_hash_with_yay);
            let nay: bool = signature::verify(signs[i], signers[i],
    packet_hash_with_nay);
            assert(yay | nay);
            if (yay) { yay_count = yay_count + 1u8; }
            if (nay) { nay_count = nay_count + 1u8; }
        }
    }

    assert(yay_count != nay_count);
    ...
}
```

Snippet 4.33: Snippet from `get_majority_count`

Impact On a call to `token_service_v0003.leo/token_receive` by any token connector, if the yay and nay counts are equal, the transaction will revert. This may not be the desired behavior when the yay and nay vote counts reach the threshold.

Recommendation If neither count reaches the set threshold, revert the transaction. If the vote counts are equal and above the threshold, default to a victory for the nay votes.

Developer Response The developers indicated that this is the intended behavior. In particular, when the threshold is less than `50%` and there are an equal number of yes and no votes, the assert will not be satisfied and the transaction will revert.

4.1.32 V-V23-VUL-032: Can make stronger voting check

Severity	Info	**Commit**	c77637b
Type	Logic Error	**Status**	Intended Behavior
File(s)	`aleo/programs/token_bridge_v0003.leo`		
Location(s)	get_majority_count()		
Confirmed Fix At	N/A		

The function `get_majority_count` iterates over the signatures in the provided array and checks whether or not the signer voted "yay" or "nay" on the proposal. The following logic shows this verification loop.

```
for i: u8 in 0u8..SUPPORTED_THRESHOLD {
    if (signers[i] != ZERO_ADDRESS) {
        let yay: bool = signature::verify(signs[i], signers[i], packet_hash_with_yay)
    ;
        let nay: bool = signature::verify(signs[i], signers[i], packet_hash_with_nay)
    ;
        assert(yay | nay);
        if (yay) { yay_count = yay_count + 1u8; }
        if (nay) { nay_count = nay_count + 1u8; }
    }
}
```

Snippet 4.34: Snippet from `example()`

The check `assert(yay | nay)` is intended to check that the signer has in fact voted either affirmatively or negatively on the proposal. However, a strong check would use XOR to ensure they only voted one way. It should be noted that other issues would have to occur for both `yay` and `nay` to be `true` in this context, but it doesn't hurt to make the stronger assertion.

Impact Strengthening the assertion will make the intention more clear and could help avoid bugs in the future if the code changes.

Recommendation Update the assertion to use XOR.

Developer Response At this time, the developers do not want to add the stronger check. In particular, because both yes and no votes cannot be signed in the same packet, the OR operator is equivalent to XOR in this case..

4.1.33 V-V23-VUL-033: Unnecessary function argument for chain ID

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	`aleo/programs/token_bridge_v0003.leo`		
Location(s)	finalize publish()		
Confirmed Fix At	N/A		

In `finalize publish`, the `source_chain_id` is always just set to `ALEO_CHAIN_ID`, and `ALEO_CHAIN_ID` is used for getting the `bridge_sequence_no` in the function body.

Impact This can be confusing to developers who may not realize the source chain ID is always Aleo and furthermore may forget to update `ALEO_CHAIN_ID` in the body of `finalize publish` if this assumption were to ever change.

Recommendation Remove the unnecessary function argument and use `ALEO_CHAIN_ID` instead.

4.1.34 V-V23-VUL-034: Typos and incorrect comments

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description In the following locations, the auditors identified **minor typos and potentially misleading comments**:

- `council_v0003.leo`
  - `proposal_votes`: The comment indicates the mapping is `true` if the member has voted, but it is actually `true` only if they voted to accept.
- `token_service_v0003.leo`
  - `max_transfers`: The comment above the mapping says `minimum` instead of `maximum`.
- `wusdc_token_v0003.leo`
  - `mint_public`: The comment above the computation of `receiver_amount` references `transfer_public` instead of `min_public`.
  - `burn_public`: Same issue as `mint_public`.
  - `burn_public`: Parameter of `transition` is called `spender` but parameter of same value in `finalize` is called `receiver`.

Impact These minor errors may lead to future developer confusion.

4.1.35 V-V23-VUL-035: Unused mapping proposal_vote_counts

Severity	Info	**Commit**	c77637b
Type	Maintainability	**Status**	Intended Behavior
File(s)	`council_v0003.leo`		
Location(s)	N/A		
Confirmed Fix At	N/A		

Within `council_v0003.leo`, the mapping `proposal_vote_counts` is never used. Though values are set, they are never referenced in the program.

```
1 /// Tracks the number of votes received by the given proposal
2 mapping proposal_vote_counts: field => u8;
```

Snippet 4.35: Definition of `proposal_vote_counts`

Impact Unused mappings like `proposal_vote_counts` may lead to confusion for future developers, hurting code maintainability.

Recommendation Remove `proposal_vote_counts` from `council_v0003.leo`.

Developer Response At this time, the developers have decided not to remove the unused mapping. This is because the mapping is used by the frontend to see the council votes.

4.1.36 V-V23-VUL-036: Unnecessary code and typos in attestor

Severity	Info	**Commit**	c424b4d
Type	Maintainability	**Status**	Fixed
File(s)	N/A		
Location(s)	See description		
Confirmed Fix At	N/A		

Unnecessary Code Several files in the attestor repository contain unnecessary or unused code.

- `db.go` contains the unused `RetrieveAndDeleteFirstPacket` method. Additionally, since the method `retrieveAndDeleteFirstKey` is only referenced here, it should also be removed from `bolt.go`.
- The function `RemoveKey` in `db.go` is unused and should be removed.
- The functions `initPacketFeeder` and `consumeMissedPackets` are defined as methods on the `relay` struct but do not use the relay, so can just be defined as normal functions.
- The function body of `CloseDB` in `db.go` is equivalent to just `return closeDB()`
- The `unconfirmedEndPoint` constant in `collector.go` is unused.
- `GetTransactionById`, `GetMappingNames`, `GindTransactionByProgramId`, and `Send` in `rpc.go` for Aleo are never used.

Typos Additionally, we found the following typos that should be corrected:

- The function named `exitsInGivenBucket` from `bolt.go` should instead be called `existsInGivenBucket`, as it is checking if a given key "exists" in the given bucket.
- In the chain service `config.yaml`, the `node_url` for testnet is set to `https://api.explorer.aleo.org/v1|testnet3`. The final `|` in that URL should instead be `/`.
- In the configuration, the bridge contract is given as `token_bridge_v0002.aleo` instead of the newer `token_bridge_v0003.aleo`.

Impact Including unnecessary/unused code and typos hurts readability and maintainability for developers.

Recommendation Remove all unnecessary/unused code and fix typos.

Developer Response The developers provide the following responses for suggested fixes:

1. For the function `RemoveKey` in `db.go`, they want to keep this in case they want to delete entries from the Bolt database on the fly.
2. For the `node_url` in `config.yaml`, this syntax is intentional.
3. For the bridge contract in `config.yaml`, they intentionally specify `token_bridge_v0002.aleo` as this is the current contract in use. They will update this when they update the contract being used.

4.1.37 V-V23-VUL-037: Use express best practices

Severity	Info	**Commit**	08f5a8e
Type	Library Usage	**Status**	Fixed
File(s)		N/A	
Location(s)		N/A	
Confirmed Fix At		N/A	

The express library best practices for security contain a number of suggestions which are not currently followed by the database service, including:

- Reduce fingerprinting by disabling `X-Powered-By`
- Use either `express-session` or `cookie-session` to handle cookies and don't use the default session cookie name
- Use TLS

Impact Following best practices can help avoid unanticipated attacks.

Recommendation Adopt all best practices suggested by the express developers.

Developer Response The developers have implemented most of the best practices and are in the process of implementing TLS.

4.1.38 V-V23-VUL-038: Typos and unused code in database service

Severity	Info	**Commit**	08f5a8e
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description In the following locations, the auditors identified minor typos and unused code:

- `src/routes/chainSignature.route.ts`:
  - `initializeRoutes()`: commented code
- `src/routes/unconfirmedPacket.route.ts`:
  - `initializeRoutes()`: commented code
- `src/middlewares/signatureValidator.middleware.ts`:
  - `signatureValidatorMiddleware()`: commented code
- `src/services/chainSignature.service.ts`:
  - `update()`: entire function commented
  - `delete()`: unimplemented function
- `src/services/unconfirmedPacket.service.ts`:
  - `update()`: unimplemented function

Impact These minor errors may lead to future developer confusion.