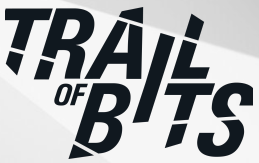

ChainPort

Security Assessment

July 5, 2022

Prepared for:
eiTan LaVi, Idan Portal
DcentraLab

Prepared by: **Michael Colburn and Vasco Franco**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

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

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

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

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

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

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

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

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

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

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

Table of Contents

Executive Summary

Engagement Overview

DcentraLab engaged Trail of Bits to review the security of its ChainPort bridge. From May 31 to June 24, 2022, a team of two consultants conducted a security review of the client-provided source code, with eight person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

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

Summary of Findings

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

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	3
Low	10
Informational	6
Undetermined	3

CATEGORY BREAKDOWN

Category	*Count*
Auditing and Logging	2
Configuration	6
Cryptography	2
Data Exposure	2
Data Validation	3
Denial of Service	1
Patching	1
Testing	1

Timing	1
Undefined Behavior	3

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

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

The following engineers were associated with this project:

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Vasco Franco, Consultant
vasco.franco@trailofbits.com

Project Timeline

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

Date	Event
April 28, 2022	Pre-project onboarding architecture call
May 25, 2022	Pre-project kickoff call
June 7, 2022	Status update meeting #1
June 13, 2022	Status update meeting #2
June 21, 2022	Status update meeting #3
June 23, 2022	Delivery of initial report draft
June 23, 2022	Report readout meeting
June 24, 2022	Delivery of second report draft
July 5, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the DcentraLab ChainPort bridge. Specifically, we sought to answer the following non-exhaustive list of questions:

- Is the signature scheme robust? Which algorithms does it use?
- Are the signatures malleable? Does the signature scheme sign every parameter?
- Are signature replay attacks between chains possible? Are replay attacks between the main and side bridges on the same chain possible?
- Does the codebase contain any cryptographic code that was implemented in-house unnecessarily? Or are well-tested libraries used instead?
- Does the cryptographic code perform sanity checks on its inputs?
- Are fees properly accounted for?
- Are there any inconsistencies in fee payments across the protocol's various operations, or between the use of the base token and staked tokens for fee payments?
- Are there adequate access controls in place to prevent the unauthorized minting of tokens?
- Could tokens become stuck in the contracts?
- Does the back end properly capture on-chain events? Does the system properly confirm blockchain events?
- Is the system robust against chain reorganization?
- Could an attacker execute a denial-of-service attack against the bridges?

Project Targets

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

ChainPort Smart Contracts

Repository	https://github.com/chainport/smart-contracts
Versions	`f10179da0ed2ecaa4e10346023c245f69fd3478e`
	`48ff2a78b3d9aa24b203db308141231c72767152` (Cardano)
Type	Solidity
Platform	EVM

ChainPort Back End

Repository	https://gitlab.com/chainport/chainport-backend
Versions	`a5b218fcd19fb2ecb4e279764c26ac4cf3c2941e`
	`00556e84bfffd965fdb6b91c53a43c8427d0f6d9` (fees)
	`ead0a71784c3a3c488483c55b1e6c5afcf51392a` (Cardano)
Type	Python
Platform	AWS Lambda

ChainPort Front End

Repository	https://gitlab.com/chainport/chainport-app
Version	`32c6ef297ff8168a2aff5ec22c337f15a0951884`
Types	JavaScript, React
Platform	Web

Project Coverage

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

- **On-chain governance.** The system's on-chain components are governed by a congress that can modify certain system parameters and toggle safety features. The congress also maintains the registry of maintainers, semi-trusted actors that relay signed bridge messages from the back end and can trigger certain safety features. We reviewed the on-chain governance components to ensure that the voting process is sound, that the governance functionalities have the proper access controls, and that maintainers do not have any unintended privileges.

- **Tokens.** The system integrates various tokens, including its own protocol token, PORTX, and the tokens that are minted by the side bridges. We reviewed these token contracts to ensure that they conform to the ERC20 standard and that the extra functionality does not introduce any vulnerabilities.

- **Bridges.** The core smart contracts of the ChainPort system are the main bridge and side bridges. The main bridge acts as a deposit contract, allowing users to deposit tokens on their base chains. The side bridges mint the bridged versions of those tokens on other chains; the side bridges also burn the bridged versions of tokens, allowing users to transfer them to other side bridges or back to the main bridge to redeem them for the original token. We reviewed these contracts to ensure that proper events are emitted for the back end to capture and that appropriate access controls are in place to prevent unauthorized minting or redeeming of tokens. We also reviewed the effectiveness of the various safety mechanisms, such as the ability to pause or freeze the bridges and protections for minting operations.

- **Signatures.** Bridging operations are relayed on-chain by maintainers as signed messages from the back end. We reviewed the off-chain signature generation and on-chain signature validation logic to ensure that the signature scheme covers all of the necessary data, that the signatures are not malleable, and that the signatures cannot be replayed, including across chains or across the main bridge and side bridges on the same chain.

- **Fees.** Users are charged a fee for bridging operations: a 0.3% fee if users pay in the token they are transferring or a reduced 0.2% fee if users pay in staked PORTX tokens. We reviewed the fee logic to ensure that fees are calculated correctly and consistently across the various bridging operations, that the PORTX staking logic is sound, and that users are not able to evade fee payment.

- **Secret management.** Secret management is integral to keeping ChainPort safe; if an attacker (or a compromised developer) has access to a maintainer's private key or to ChainPort's AWS account, which stores these private keys, the attacker can generate signatures and release tokens that do not belong to him. We reviewed the repository for hard-coded secrets and provided recommendations on how to better store these secrets.

- **The front end.** We reviewed the front end for common client-side vulnerabilities; however, the front end was a lower-priority component for this audit, so our coverage of it was limited.

- **The back end.** The back end is the off-chain component that listens to blockchain events and acts on them. We used both a manual review and existing and custom-developed static analysis tools to review its code for common flaws. We also manually reviewed its monitors for bridge-specific bugs, such as the reliance on blocks that have not been finalized and other logic bugs.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We received a commit including initial support for bridging with Cardano in the final week of the assessment, so we were unable to review it with the same depth as the other components under audit. Furthermore, the Cardano code contains several "TODO" comments and basic coding mistakes such as function calls that do not contain the correct number of arguments (e.g., the `cardano_to_evm_transaction` function calls both `session_get_token_by_network_web3_address` and `cardano_to_mint_transaction` with an incorrect number of arguments). These factors indicate that the Cardano code was not ready for an in-depth review.

- Due to the tight coupling of on-chain and off-chain components, we were not able to test the system using dynamic testing techniques, such as fuzzing.

- The front end was not reviewed in depth, as it was marked as a low-priority component by the ChainPort team. We used only a set of static analysis tools and a quick dynamic assessment to evaluate the front end for low-complexity issues.

- We did not report issues related to files inside the `deprecated_files` folder.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	Math is used in the smart contracts mostly for straightforward balance updates. The contracts use the SafeMath library, as they use an older version of the Solidity compiler.	**Satisfactory**
Auditing	The smart contracts emit proper events for the bridging functionality; however, we identified a few critical operations that are missing events, such as parameter updates (TOB-CHPT-7). The back end uses an error-prone pattern to log functions (TOB-CHPT-3).	**Moderate**
Authentication / Access Controls	Appropriate access controls are in place in the smart contracts. Any movement of tokens out of the bridges requires a signed message from the bridge. Semi-trusted maintainers can trigger certain safety mechanisms, which can be unset only by a vote from the ChainPort congress, which manages the set of maintainers.	**Satisfactory**
Complexity Management	The smart contracts and their functions are organized and scoped appropriately, and most of them have inline documentation to explain their workings. However, there are some instances of duplicate code in the smart contracts. The back-end folder structure is well organized; however, the back-end code lacks inline documentation, contains a substantial amount of commented-out code, and repeatedly uses hard-coded variables instead of	**Moderate**

	constants (TOB-CHPT-4).	
Cryptography and Key Management	The system relies heavily on digital signatures to authorize bridging operations. Though we did not identify any exploitable issues, documentation and specification of the signature scheme (TOB-CHPT-16) and data validation (TOB-CHPT-17) could be improved. The Git repositories store secrets that should instead be stored in a purpose-built secret management solution such as Vault.	**Moderate**
Data Handling	The back end generally handles data safely; however, it frequently uses error-prone patterns (TOB-CHPT-3, TOB-CHPT-4, TOB-CHPT-9, and TOB-CHPT-12) and contains mistakes that a static analysis tool would catch (TOB-CHPT-5 and TOB-CHPT-6).	**Moderate**
Decentralization	As a bridging protocol, ChainPort acts as a trusted centralized entity.	**Not Applicable**
Documentation	The back end system generally lacks inline documentation and other documentation describing the components' purposes. The signature scheme also lacks a proper specification (TOB-CHPT-16).	**Weak**
Front-Running Resistance	We identified one minor issue that could allow a user to front-run fee payments (TOB-CHPT-15) but did not identify any other front-running vectors.	**Satisfactory**
Low-Level Manipulation	The smart contracts use low-level calls only for upgradeability and use minimal inline assembly for upgradeability and signature validation.	**Satisfactory**
Maintenance	The back-end CI does not check for vulnerable Python libraries with a tool such as `pip-audit`.	**Weak**

Testing and Verification	The smart contract test suite that runs as part of the CI process has adequate coverage of a variety of scenarios. However, the system is not tested with dynamic testing techniques such as fuzzing. The back end lacks tests. For example, there are no tests to verify the correctness of the critical signature-related code.	**Weak**

Summary of Findings

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

ID	Title	Type	Severity
1	Several secrets checked into source control	Data Exposure	Medium
2	Same credentials used for staging, test, and production environment databases	Configuration	Low
3	Use of error-prone pattern for logging functions	Auditing and Logging	Low
4	Use of hard-coded strings instead of constants	Data Validation	Informational
5	Use of incorrect operator in SQLAlchemy filter	Undefined Behavior	Undetermined
6	Several functions receive the wrong number of arguments	Undefined Behavior	Undetermined
7	Lack of events for critical operations	Auditing and Logging	Informational
8	Lack of zero address checks in setter functions	Data Validation	Informational
9	Python type annotations are missing from most functions	Undefined Behavior	Low
10	Use of libraries with known vulnerabilities	Patching	Low
11	Use of JavaScript instead of TypeScript	Configuration	Low
12	Use of .format to create SQL queries	Data Validation	Informational

13	Many rules are disabled in the ESLint configuration	Testing	**Informational**
14	Congress can lose quorum after manually setting the quorum value	Configuration	**Medium**
15	Potential race condition could allow users to bypass PORTX fee payments	Timing	**Low**
16	Signature-related code lacks a proper specification and documentation	Cryptography	**Medium**
17	Cryptographic primitives lack sanity checks and clear function names	Cryptography	**Informational**
18	Use of requests without the timeout argument	Denial of Service	**Low**
19	Lack of noopener attribute on external links	Configuration	**Low**
20	Use of urllib could allow users to leak local files	Data Exposure	**Undetermined**
21	The front end is vulnerable to iFraming	Configuration	**Low**
22	Lack of CSP header in the ChainPort front end	Configuration	**Low**

Detailed Findings

1. Several secrets checked into source control	
Severity: **Medium**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-CHPT-1
Target: The `chainport-backend` repository	

Description

The `chainport-backend` repository contains several secrets that are checked into source control. Secrets that are stored in source control are accessible to anyone who has had access to the repository (e.g., former employees or attackers who have managed to gain access to the repository).

We used TruffleHog to identify these secrets (by running the command `trufflehog git file://.` in the root directory of the repository). TruffleHog found several types of credentials, including the following, which were verified through TruffleHog's credential verification checks:

- GitHub personal access tokens
- Slack access tokens

TruffleHog also found unverified GitLab authentication tokens and Polygon API credentials.

Furthermore, we found hard-coded credentials, such as database credentials, in the source code, as shown in figure 1.1.

```
[REDACTED]
```

Figure 1.1: `chainport-backend/env.prod.json#L3-L4`

Exploit Scenario

An attacker obtains a copy of the source code from a former DcentraLab employee. The attacker extracts the secrets from it and uses them to exploit DcentraLab's database and insert events in the database that did not occur. Consequently, ChainPort's AWS lambdas process the fake events and allow the attacker to steal funds.

Recommendations

Short term, remove credentials from source control and rotate them. Run TruffleHog by invoking the `trufflehog git file://.` command; if it identifies any unverified credentials, check whether they need to be addressed.

Long term, consider using a secret management solution such as Vault to store secrets.

2. Same credentials used for staging, test, and production environment databases	
Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CHPT-2
Target: Database authentication	

Description
The staging, test, and production environments' databases have the same username and password credentials.

Exploit Scenario
An attacker is able to obtain the password for the test environment's database, which is less tightly secured than that of the production environment. He tries the same credentials on the production database and gains access to the database's contents as well as the ability to write to it. He inserts fake events into the database. ChainPort's AWS lambdas process the fake events and allow the attacker to steal funds.

Recommendations
Short term, rotate the current credentials and safely generate different credentials for each environment. This will prevent attackers who have compromised credentials for one environment from accessing other environments.

Long term, use a secret management solution such as Vault to store the database credentials instead of relying on configuration files stored in the source code.

3. Use of error-prone pattern for logging functions

Severity: **Low**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: TOB-CHPT-3
Target: The `chainport-backend` repository	

Description

The pattern shown in figure 3.1 is used repeatedly throughout the codebase to log function names.

```
[REDACTED]
```

Figure 3.1: An example of the pattern used by ChainPort to log function names

This pattern is prone to copy-and-paste errors. Developers may copy the code from one function to another but forget to change the function name, as exemplified in figure 3.2.

```
[REDACTED]
```

Figure 3.2: An example of an incorrect use of the pattern used by ChainPort to log function names

We wrote a Semgrep rule to detect these problems (appendix D). This rule detected 46 errors associated with this pattern in the back-end application. Figure 3.3 shows an example of one of these findings.

```
[REDACTED]
```

Figure 3.3: An example of one of the 46 errors resulting from the function-name logging pattern (`chainport-backend/modules/web_3/helpers.py#L313-L315`)

Exploit Scenario

A ChainPort developer is auditing the back-end application logs to determine the root cause of a bug. Because an incorrect function name was logged, the developer cannot correctly trace the application's flow and determine the root cause in a timely manner.

Recommendations

Short term, use the Python decorator in figure 3.4 to log function names. This will eliminate the risk of copy-and-paste errors.

```
[REDACTED]
```

Figure 3.4: A Python decorator that logs function names, eliminating the risk of copy-and-paste errors

Long term, review the codebase for other error-prone patterns. If such patterns are found, rewrite the code in a way that eliminates or reduces the risk of errors, and write a Semgrep rule to find the errors before the code hits production.

4. Use of hard-coded strings instead of constants	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-CHPT-4
Target: The `chainport-backend` repository	

Description

The back-end code uses several hard-coded strings that could be defined as constants to prevent any typos from introducing vulnerabilities.

For example, the checks that determine the system's environment compare the result of the `get_env` function with the strings "`develop`", "`staging`", "`prod`", or "`local`". Figure 4.1 shows an example of one of these checks.

```
[REDACTED]
```

Figure 4.1:
chainport-backend/project/lambdas/mainchain/rebalance_monitor.py#L42-L43

We did not find any typos in these literal strings, so we set the severity of this finding to informational. However, the use of hard-coded strings in place of constants is not best practice; we suggest fixing this issue and following other best practices for writing safe code to prevent the introduction of bugs in the future.

Exploit Scenario

A ChainPort developer creates code that should run only in the development build and safeguards it with the check in figure 4.2.

```
[REDACTED]
```

Figure 4.2: An example of a check against a hard-coded string that could lead to a vulnerability

This test always fails—the correct value to test should have been "`develop`". Now, the poorly tested, experimental code that was meant to run only in development mode is deployed in production.

Recommendations

Short term, create a constant for each of the four possible environments. Then, to check the system's environment, import the corresponding constant and use it in the comparison instead of the hard-coded string. Alternatively, use an enum instead of a string to perform these comparisons.

Long term, review the code for other instances of hard-coded strings where constants could be used instead. Create Semgrep rules to ensure that developers never use hard-coded strings where constants are available.

5. Use of incorrect operator in SQLAlchemy filter

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-CHPT-5
Target: `chainport-backend/project/data/db/port.py#L173`	

Description
The back-end code uses the `is not` operator in an `SQLAlchemy` query's `filter`. `SQLAlchemy` relies on the `__eq__` family of methods to apply the filter; however, the `is` and `is not` operators do not trigger these methods. Therefore, only the comparison operators (== or `!=`) should be used.

```
[REDACTED]
```

Figure 5.1: `chainport-backend/project/data/db/port.py#L173`

We did not review whether this flaw could be used to bypass the system's business logic, so we set the severity of this issue to undetermined.

Exploit Scenario
An attacker exploits this flawed check to bypass the system's business logic and steal user funds.

Recommendations
Short term, replace the `is not` operator with `!=` in the `filter` indicated above.

Long term, to continuously monitor the codebase for reintroductions of this issue, run the `python.sqlalchemy.correctness.bad-operator-in-filter.bad-operator-in-filter` Semgrep rule as part of the CI/CD flow.

References
- SQLAlchemy: Common Filter Operators
- Stack Overflow: Select NULL Values in SQLAlchemy

6. Several functions receive the wrong number of arguments

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-CHPT-6
Target: The `chainport-backend` repository	

Description

Several functions in the `chainport-backend` repository are called with an incorrect number of arguments:

- Several functions in the `/project/deprecated_files` folder
- A call to `release_tokens_by_maintainer` from the `rebalance_bridge` function (figures 6.1 and 6.2)
- A call to `generate_redeem_signature` from the `regenerate_signature` function (figures 6.3 and 6.4)
- A call to `get_next_nonce_for_public_address` from the `prepare_erc20_transfer_transaction` function (figures 6.5 and 6.6)
- A call to `get_cg_token_address_list` from the main function of the file (likely old debugging code)

```
[REDACTED]
```

Figure 6.1: The `release_tokens_by_maintainer` function is called with four arguments, but at least five are required. (`chainport-backend/project/lambdas/mainchain/rebalance_monitor.py#L109-L114`)

```
[REDACTED]
```

Figure 6.2: The definition of the `release_tokens_by_maintainer` function (`chainport-backend/project/lambdas/release_tokens_by_maintainer.py#L27-L34`)

```
[REDACTED]
```

Figure 6.3: A call to `generate_redeem_signature` that is missing the `network_id` argument (chainport-backend/project/scripts/keys_maintainers_signature/regenerate_signature.py#L38-L43)

```
[REDACTED]
```

Figure 6.4: The definition of the `generate_redeem_signature` function (chainport-backend/project/lambdas/sidechain/events_handlers/handle_burn_event.py#L46-L48)

```
[REDACTED]
```

Figure 6.5: A call to `get_next_nonce_for_public_address` that is missing the `outer_session` argument (chainport-backend/project/web3_cp/erc20/prepare_erc20_transfer_transaction.py#L32-L34)

```
[REDACTED]
```

Figure 6.6: The definition of the `get_next_nonce_for_public_address` function (chainport-backend/project/web3_cp/nonce.py#L19-L21)

```
[REDACTED]
```

Figure 6.7: A call to `get_cg_token_address_list` that is missing all three arguments (chainport-backend/project/lambdas/token_endpoints/cg_list_get.py#L90-91)

```
[REDACTED]
```

Figure 6.8: The definition of the `get_cg_token_address_list` function (chainport-backend/project/lambdas/token_endpoints/cg_list_get.py#L37)

We did not review whether this flaw could be used to bypass the system's business logic, so we set the severity of this issue to undetermined.

Exploit Scenario

The `release_tokens_by_maintainer` function is called from the `rebalance_bridge` function with the incorrect number of arguments. As a result, the `rebalance_bridge` function fails if the token balance is over the threshold limit, and the tokens are not moved to a safe address. An attacker finds another flaw and is able to steal more tokens than he would have been able to if the tokens were safely stored in another account.

Recommendations

Short term, fix the errors presented in the description of this finding by adding the missing arguments to the function calls.

Long term, run `pylint` or a similar static analysis tool to detect these problems (and others) before the code is committed and deployed in production. This will ensure that if the list of a function's arguments ever changes (which was likely the root cause of this problem), a call that does not match the new arguments will be flagged before the code is deployed.

7. Lack of events for critical operations

Severity: **Informational**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: TOB-CHPT-7
Target: `ChainportMainBridge.sol`, `ChainportSideBridge.sol`, `Validator.sol`,	

Description
Several critical operations do not trigger events. As a result, it will be difficult to review the correct behavior of the contracts once they have been deployed.

For example, the `setSignatoryAddress` function, which is called in the `Validator` contract to set the signatory address, does not emit an event providing confirmation of that operation to the contract's caller (figure 7.1).

```
[REDACTED]
```

Figure 7.1: The `setSignatoryAddress` function in `Validator:43-52`

Without events, users and blockchain-monitoring systems cannot easily detect suspicious behavior.

Exploit Scenario
Eve, an attacker, is able to compromise a quorum of the ChainPort congress voters contract. She then sets a new signatory address. Alice, a ChainPort team member, is unaware of the change and does not raise a security incident.

Recommendations
Short term, add events for all critical operations that result in state changes. Events aid in contract monitoring and the detection of suspicious behavior.

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

8. Lack of zero address checks in setter functions

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-CHPT-8
Target: `ChainportMainBridge.sol`, `ChainportMiddleware.sol`, `ChainportSideBridge.sol`	

Description
Certain setter functions fail to validate incoming arguments, so callers can accidentally set important state variables to the zero address.

For example, in the `initialize` function of the `ChainportMainBridge` contract, developers can define the maintainer registry, the congress address for governance, and the signature validator and set their addresses to the zero address.

```
[REDACTED]
```

Figure 8.1: The `initialize` function of `ChainportMainBridge.sol`

Failure to immediately reset an address that has been set to the zero address could result in unexpected behavior.

Exploit Scenario
Alice accidentally sets the ChainPort congress address to the zero address when initializing a new version of the `ChainportMainBridge` contract. The misconfiguration causes the system to behave unexpectedly, and the system must be redeployed once the misconfiguration is detected.

Recommendations
Short term, add zero-value checks to all constructor functions and for all setter arguments to ensure that users cannot accidentally set incorrect values, misconfiguring the system. Document any arguments that are intended to be set to the zero address, highlighting the expected values of those arguments on each chain.

Long term, use the Slither static analyzer to catch common issues such as this one. Consider integrating a Slither scan into the project's CI pipeline, pre-commit hooks, or build scripts.

9. Python type annotations are missing from most functions	
Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-CHPT-9
Target: The `chainport-backend` repository	

Description
The back-end code uses Python type annotations; however, their use is sporadic, and most functions are missing them.

Exploit Scenario
The `cg_rest_call` function receives the `exception` argument without specifying its type with a Python type annotation. The `get_token_details_by_cg_id` function calls `cg_rest_call` with an object of the incorrect type, an `Exception` instance instead of an `Exception` class, causing the program to crash (figure 9.1).

```
[REDACTED]
```

Figure 9.1: chainport-backend/modules/coingecko/api.py#L41-L42

Recommendations
Short term, add type annotations to the arguments of every function. This will not prevent the code from crashing or causing undefined behavior during runtime; however, it will allow developers to clearly see each argument's expected type and static analyzers to better detect type mismatches.

Long term, implement checks in the CI/CD pipeline to ensure that code without type annotations is not accepted.

10. Use of libraries with known vulnerabilities

Severity: **Low**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-CHPT-10
Target: The `chainport-backend` repository	

Description

The back-end repository uses outdated libraries with known vulnerabilities. We used `pip-audit`, a tool developed by Trail of Bits with support from Google to audit Python environments and dependency trees for known vulnerabilities, and identified two known vulnerabilities in the project's dependencies (as shown in figure 10.1).

```
[REDACTED]
```

Figure 10.1: A list of outdated libraries in the back-end repository

Recommendations

Short term, update the project's dependencies to their latest versions wherever possible. Use `pip-audit` to confirm that no vulnerable dependencies remain.

Long term, add `pip-audit` to the project's CI/CD pipeline. Do not allow builds to succeed with dependencies that have known vulnerabilities.

11. Use of JavaScript instead of TypeScript	
Severity: **Low**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-CHPT-11
Target: The `chainport-app` repository	

Description
The ChainPort front end is developed with JavaScript instead of TypeScript. TypeScript is a strongly typed language that compiles to JavaScript. It allows developers to specify the types of variables and function arguments, and TypeScript code will fail to compile if there are type mismatches. Contrarily, JavaScript code will crash (or worse) during runtime if there are type mismatches.

In summary, TypeScript is preferred over JavaScript for the following reasons:

- It improves code readability; developers can easily identify variable types and the types that functions receive.
- It improves security by providing static type checking that catches errors during compilation.
- It improves support for integrated development environments (IDEs) and other tools by allowing them to reason about the types of variables.

Exploit Scenario
A bug in the front-end application is missed, and the code is deployed in production. The bug causes the application to crash, preventing users from using it. This bug would have been caught if the front-end application were written in TypeScript.

Recommendations
Short term, rewrite newer parts of the application in TypeScript. TypeScript can be used side-by-side with JavaScript in the same application, allowing it to be introduced gradually.

Long term, gradually rewrite the whole application in TypeScript.

12. Use of .format to create SQL queries

Severity: **Informational**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-CHPT-12
Target: [REDACTED]	

Description
The back end builds SQL queries with the `.format` function. An attacker that controls one of the variables that the function is formatting will be able to inject SQL code to steal information or damage the database.

```
[REDACTED]
```

Figure 12.1: chainport-backend/project/data/db/postgres.py#L4-L24

```
[REDACTED]
```

Figure 12.2: chainport-backend/project/lambdas/database_monitor/clear_lock.py#L29-L31

None of the fields described above are attacker-controlled, so we set the severity of this finding to informational. However, the use of `.format` to create SQL queries is an anti-pattern; parameterized queries should be used instead.

Exploit Scenario
A developer copies the vulnerable code to create a new SQL query. This query receives an attacker-controlled string. The attacker conducts a time-based SQL injection attack, leaking the whole database.

Recommendations
Short term, use parameterized queries instead of building strings with variables by hand.

Long term, create or use a static analysis check that forbids this pattern. This will ensure that this pattern is never reintroduced by a less security-aware developer.

13. Many rules are disabled in the ESLint configuration

Severity: **Informational**	Difficulty: **High**
Type: Testing	Finding ID: TOB-CHPT-13
Target: `chainport-app/.eslintrc.js`	

Description

There are 34 rules disabled in the front-end `eslint` configuration. Disabling some of these rules does not cause problems, but disabling others reduces the code's security and reliability (e.g., `react/no-unescaped-entities`, `consistent-return`, `no-shadow`) and the code's readability (e.g., `react/jsx-boolean-value`, `react/jsx-one-expression-per-line`).

Furthermore, the code contains 46 inline `eslint-disable` comments to disable specific rules. While disabling some of these rules in this way may be valid, we recommend adding a comment to each instance explaining why the specific rule was disabled.

Recommendations

Short term, create a list of rules that can be safely disabled without reducing the code's security or readability, document the justification, and enable every other rule. Fix any findings that these rules may report. For rules that are disabled with inline `eslint-disable` comments, include explanatory comments justifying why they are disabled.

14. Congress can lose quorum after manually setting the quorum value

Severity: **Medium**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CHPT-14
Target: `contracts/governance/ChainportCongressMembersRegistry.sol`	

Description

Proposals to the ChainPort congress must be approved by a minimum quorum of members before they can be executed. By default, when a new member is added to the congress, the quorum is updated to be N − 1, where N is the number of congress members.

```
[REDACTED]
```

Figure 14.1: smart-contracts/contracts/governance/ChainportCongressMembersRegistry.sol#L98-L119

However, the congress has the ability to overwrite the quorum number to any nonzero number, including values larger than the current membership.

```
[REDACTED]
```

Figure 14.2: smart-contracts//contracts/governance/ChainportCongressMembersRegistry.sol#L69-L77

If the congress manually lowers the quorum number and later adds a member, the quorum number will be reset to one less than the total membership. If for some reason certain members are temporarily or permanently unavailable (e.g., they are on vacation or their private keys were destroyed), the minimum quorum would not be reached.

Exploit Scenario

The ChainPort congress is composed of 10 members. Alice submits a proposal to reduce the minimum quorum to six members to ensure continuity while several members take vacations over a period of several months. During this period, a proposal to add Bob as a new member of the ChainPort congress is passed while Carol and Dave, two other congress members, are on vacation. This unexpectedly resets the minimum quorum to 10 members of the 11-person congress, preventing new proposals from being passed.

Recommendations

Short term, rewrite the code so that, when a new member is added to the congress, the minimum quorum number increases by one rather than being updated to the current number of congress members subtracted by one. Add a cap to the minimum quorum number to prevent it from being manually set to values larger than the current membership of the congress.

Long term, uncouple operations for increasing and decreasing quorum values from operations for making congress membership changes. Instead, require that such operations be included as additional actions in proposals for membership changes.

15. Potential race condition could allow users to bypass PORTX fee payments

Severity: **Low**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-CHPT-15
Target: `contracts/ChainportFeeManager.sol`	

Description

ChainPort fees are paid either as a 0.3% fee deducted from the amount transferred or as a 0.2% fee in PORTX tokens that the user has deposited into the `ChainportFeeManager` contract. To determine whether a fee should be paid in the base token or in PORTX, the back end checks whether the user has a sufficient PORTX balance in the `ChainportFeeManager` contract.

```
[REDACTED]
```

Figure 15.1: chainport-backend//project/lambdas/fees/fees.py#L219-249

However, the `ChainportFeeManager` contract does not enforce an unbonding period, a period of time before users can unstake their PORTX tokens.

```
[REDACTED]
```

Figure 15.2: smart-contracts/contracts/ChainportFeeManager.sol#L113-L125

Since pending fee payments are generated as part of deposit, transfer, and burn events but the actual processing is handled by a separate monitor, it could be possible for a user to withdraw her PORTX tokens on-chain after the deposit event has been processed and before the fee payment transaction is confirmed, allowing her to avoid paying a fee for the transfer.

Exploit Scenario

Alice uses ChainPort to bridge one million USDC from the Ethereum mainnet to Polygon. She has enough PORTX deposited in the `ChainportFeeManager` contract to cover the $2,000 fee. She watches for the pending fee payment transaction and front-runs it to remove her PORTX from the `ChainportFeeManager` contract. Her transfer succeeds, but she is not required to pay the fee.

Recommendations

Short term, add an unbonding period preventing users from unstaking PORTX before the period has passed.

Long term, ensure that deposit, transfer, and redemption operations are executed atomically with their corresponding fee payments.

16. Signature-related code lacks a proper specification and documentation	
Severity: **Medium**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-CHPT-16
Target: Signature-related code	

Description

ChainPort uses signatures to ensure that messages to mint and release tokens were generated by the back end. These signatures are not well documented, and the properties they attempt to provide are often unclear. For example, answers to the following questions are not obvious; we provide example answers that could be provided in the documentation of ChainPort's use of signatures:

- **Why does the signed message contain a `networkId` field, and why does it have to be unique?** If not, an operation to mint tokens on one chain could be replayed on another chain.
- **Why does the signed message contain an `action` field?** The `action` field prevents replay attacks in networks that have both a main and side bridge. Without this field, a signature for minting tokens could be used on a sidechain contract of the same network to release tokens.
- **Why are both the signature and nonce checked for uniqueness in the contracts?** The signatures could be represented in more than one format, which means that storing them is not enough to ensure uniqueness.

Recommendations

Short term, create a specification describing what the signatures protect against, what properties they attempt to provide (e.g., integrity, non-repudiation), and how these properties are provided.

17. Cryptographic primitives lack sanity checks and clear function names

Severity: **Informational**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-CHPT-17
Target: `chainport-backend/modules/cryptography_2key/signatures.py`	

Description

Several cryptographic primitives are missing sanity checks on their inputs. Without such checks, problems could occur if the primitives are used incorrectly.

The `remove_0x` function (figure 17.1) does not check that the input starts with `0x`. A similar function in the `eth-utils` library has a more robust implementation, as it includes a check on its input (figure 17.2).

```
[REDACTED]
```

Figure 17.1: chainport-backend/modules/cryptography_2key/signatures.py#L10-L16

```
[REDACTED]
```

Figure 17.2: ethereum/eth-utils/eth_utils/hexadecimal.py#L43-L46

The `add_leading_0` function's name does not indicate that the value is padded to a length of 64 (figure 17.3).

```
[REDACTED]
```

Figure 17.3: chainport-backend/modules/cryptography_2key/signatures.py#L19-L25

The `_build_withdraw_message` function does not ensure that the `beneficiary_address` and `token_address` inputs have the expected length of 66 bytes and that they start with `0x` (figure 17.4).

```
[REDACTED]
```

Figure 17.4: chainport-backend/modules/cryptography_2key/signatures.py#L28-62

We did not identify problems in the way these primitives are currently used in the code, so we set the severity of this finding to informational. However, if the primitives are used

improperly in the future, cryptographic bugs that can have severe consequences could be introduced, which is why we highly recommend fixing the issues described in this finding.

Exploit Scenario

A developer fails to understand the purpose of a function or receives an input from outside the system that has an unexpected format. Because the functions lack sanity checks, the code fails to do what the developer expected. This leads to a cryptographic vulnerability and the loss of funds.

Recommendations

Short term, add the missing checks and fix the naming issues described above. Where possible, use well-reviewed libraries rather than implementing cryptographic primitives in-house.

Long term, review all the cryptographic primitives used in the codebase to ensure that the functions' purposes are clear and that functions perform sanity checks, preventing them from being used improperly. Where necessary, add comments to describe the functions' purposes.

18. Use of requests without the timeout argument	
Severity: **Low**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-CHPT-18
Target: The `chainport-backend` repository	

Description

The Python `requests` library is used in the ChainPort back end without the `timeout` argument. By default, the `requests` library will wait until the connection is closed before fulfilling a request. Without the `timeout` argument, the program will hang indefinitely.

The following locations in the back-end code are missing the timeout argument:

- `chainport-backend/modules/coingecko/api.py#L29`
- `chainport-backend/modules/requests_2key/requests.py#L14`
- `chainport-backend/project/stats/cg_prices.py#L74`
- `chainport-backend/project/stats/cg_prices.py#L95`

The code in these locations makes requests to the following websites:

- https://api.coingecko.com
- https://ethgasstation.info
- https://gasstation-mainnet.matic.network

If any of these websites hang indefinitely, so will the back-end code.

Exploit Scenario

One of the requested websites hangs indefinitely. This causes the back end to hang, and token ports from other users cannot be processed.

Recommendations

Short term, add the `timeout` argument to each of the code locations indicated above. This will ensure that the code will not hang if the website being requested hangs.

Long term, integrate Semgrep into the CI pipeline to ensure that uses of the `requests` library always have the `timeout` argument.

19. Lack of noopener attribute on external links

Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CHPT-19
Target: `chainport-app/src/modules/exchange/components/PortOutModal.jsx`	

Description

In the ChainPort front-end application, there are links to external websites that have the `target` attribute set to `_blank` but lack the `noopener` attribute. Without this attribute, an attacker could perform a reverse tabnabbing attack.

```
[REDACTED]
```

Figure 19.1: chainport-app/src/modules/exchange/components/PortOutModal.jsx#L126

Exploit Scenario

An attacker takes control of one of the external domains linked by the front end. The attacker prepares a malicious script on the domain that uses the `window.opener` variable to control the parent window's location. A user clicks on the link in the ChainPort front end. The malicious website is opened in a new window, and the original ChainPort front end is seamlessly replaced by a phishing website. The victim then returns to a page that appears to be the original ChainPort front end but is actually a web page controlled by the attacker. The attacker tricks the user into transferring his funds to the attacker.

Recommendations

Short term, add the missing `rel="noopener noreferrer"` attribute to the anchor tags.

References

- OWASP: Reverse tabnabbing attacks

20. Use of urllib could allow users to leak local files

Severity: **Undetermined**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-CHPT-20
Target: `chainport-backend/modules/infrastructure/aws/s3.py`	

Description

To upload images of new tokens to S3, the `upload_media_from_url_to_s3` function uses the `urllib` library (figure 20.1), which supports the `file://` scheme; therefore, if a malicious actor controls a dynamic value uploaded to S3, she could read arbitrary local files.

```
[REDACTED]
```

Figure 20.1: `chainport-backend/modules/infrastructure/aws/s3.py#L25-29`

The code in figure 20.2 replicates this issue.

```
[REDACTED]
```

Figure 20.2: Code to test `urlopen`'s support of the `file://` scheme

We set the severity of this finding to undetermined because it is unclear whether an attacker (e.g., a token owner) would have control over token images uploaded to S3 and whether the server holds files that an attacker would want to extract.

Exploit Scenario

A token owner makes the image of his token point to a local file (e.g., `file:///etc/passwd`). This local file is uploaded to the S3 bucket and is shown to an attacker attempting to port his own token into the ChainPort front end. The local file is leaked to the attacker.

Recommendations

Short term, use the `requests` library instead of `urllib`. The `requests` library does not support the `file://` scheme.

21. The front end is vulnerable to iFraming	
Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CHPT-21
Target: The `chainport-app` repository	

Description
The ChainPort front end does not prevent other websites from iFraming it.

Figure 21.1 shows an example of how another website could iFrame the ChainPort front end.

```
[REDACTED]
```

Figure 21.1: An example of how another website could iFrame the ChainPort front end

Exploit Scenario
An attacker creates a website that iFrames ChainPort's front end. The attacker performs a clickjacking attack to trick users into submitting malicious transactions.

Recommendations
Short term, add the `X-Frame-Options: DENY` header on every server response. This will prevent other websites from iFraming the ChainPort front end.

22. Lack of CSP header in the ChainPort front end	
Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-CHPT-22
Target: The `chainport-app` repository	

Description
The ChainPort front end lacks a Content Security Policy (CSP) header, leaving it vulnerable to cross-site scripting (XSS) attacks.

A CSP header adds extra protection against XSS and data injection attacks by enabling developers to select the sources that the browser can execute or render code from. This safeguard requires the use of the CSP HTTP header and appropriate directives in every server response.

Exploit Scenario
An attacker finds an XSS vulnerability in the ChainPort front end and crafts a custom XSS payload. Because of the lack of a CSP header, the browser executes the attack, enabling the attacker to trick users into transferring their funds to him.

Recommendations
Short term, use a CSP header in the ChainPort front end and validate it with the CSP Evaluator. This will help mitigate the effects of XSS attacks.

Long term, track the development of the CSP header and similar web browser features that help mitigate security risks. Ensure that new protections are adopted as quickly as possible.

References
- HTTP Content Security Policy (CSP)
- Google CSP Evaluator
- Google Web Security Fundamentals: Eval
- Google Web Security Fundamentals: Inline code is considered harmful

Summary of Recommendations

The DcentraLab ChainPort bridge is under continuous development. Trail of Bits recommends that DcentraLab address the findings detailed in this report and take the following additional steps prior to deployment:

- Review the way secrets are stored. Instead of hard-coding secrets in source code or configuration files, use a purpose-built secret management solution such as Vault.
- Continue to develop documentation and specifications for the protocol, especially for the most security-critical components, such as the signature scheme.
- Implement dynamic testing techniques on the smart contracts, such as fuzzing with Echidna.
- Add static analysis tooling to the project's CI process. Several issues identified in this report could have been detected by static analysis tools. For the smart contracts, consider integrating Slither via `slither-action`.
- Integrate `pip-audit` into the project's CI process to detect outdated vulnerable dependencies.
- Conduct an additional security review focused on the Cardano components once they are finalized.

A. Vulnerability Categories

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

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

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

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

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Maintenance	The timely maintenance of system components to mitigate risk
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

This appendix contains findings that do not have immediate or obvious security implications. However, they may facilitate exploit chains targeting other vulnerabilities or may become easily exploitable in future releases.

ChainPort Back End

- **Useless `if` statements.** All blocks of the following `if` statements have the same code.

```
[REDACTED]
```

Figure C.1: chainport-backend/project/web3_cp/events_cp/attr_mapping.py#L33-L36

```
[REDACTED]
```

Figure C.2: hainport-backend/project/env/aws.py#L61-L66

- **Comparisons with None using ==.** When comparing values with None, it is best practice to use the `is` operator, as it is faster than the == operator and could prevent undefined behavior with certain `__eq__` implementations. Instances of this problem exist in the following locations:

```
[REDACTED]
```

Figure C.3: chainport-backend/project/lambdas/fees/fees.py#L528-L529

```
[REDACTED]
```

Figure C.4: chainport-backend/project/lambdas/stats/stats_get.py#L53-L54

```
[REDACTED]
```

Figure C.5: chainport-backend/project/lambdas/total_value_locked_monitor.py#L94-L96

- **Use of possibly uninitialized variables.** In several locations in the code, variables are defined within `if` statements that do not have corresponding `else` statements. Therefore, if none of the `if` cases in a given statement are true, the variable may be uninitialized. Instances of this problem exist in the following locations:

```
[REDACTED]
```

Figure C.6: chainport-backend/modules/web_3/events.py#L26-L41

Figure C.7: chainport-backend/project/lambdas/fees/fees.py#L383-L390

Figure C.8: chainport-backend/project/web3_cp/events_cp/fetch_events_cp.py#L42-L48

- **Mixed returns.** Several functions can return types other than those specified by their type annotations. As shown in figure C.9, the `handle_token_transfer_event_tx` function's type annotation indicates that it should always return `None` (i.e., it should not return), but it can return the result of a call to the `create_new_bridge_token` function. Additionally, as shown in figures C.10 and C.11, the `bridge_stats_to_s3` and `mint_new_token` functions can return `None`, but their type annotations indicate that they should return a `dict` and `Transaction`, respectively.

Figure C.9: chainport-backend/project/lambdas/sidechain/handle_token_transfer_event.py#L45-L213

Figure C.10: chainport-backend/project/lambdas/stats/helper.py#L80-87

Figure C.11: chainport-backend/project/web3_cp/contracts/side_bridge/side_bridge.py#L146-L151

- **Unnecessary pass statements.** The following functions use a `pass` statement that can be safely removed:
  - `FeeManager.__init__`
  - `MainBridge.__init__`
- **Shadowing of built-in functions.** Figures C.12 and C.13 show instances in which the built-in Python functions `hash` and `dict` are shadowed by a local variable.

Figure C.12: chainport-backend/modules/utils/random.py#L9

```
[REDACTED]
```

Figure C.13: chainport-backend/modules/utils/tests/test_json_2key.py#L28-36

- **Unused local variables.** Figures C.14 and C.15 show instances of local variables that are defined but never used. If they are not necessary, they should be removed from the code to improve its readability.

```
[REDACTED]
```

Figure C.14: chainport-backend/project/lambdas/lambdas_helpers.py#L18

```
[REDACTED]
```

Figure C.15: chainport-backend/project/lambdas/sidechain/events_handlers/handle_burn_event.py#L59

- **Shadowing of global variables.** Figures C.16 and C.17 show instances in which the code shadows global variables. Shadowing global variables diminishes the code's readability and can lead to bugs if the actual global variables were expected to be used.

```
[REDACTED]
```

Figure C.16: chainport-backend/modules/data_2key/db/session.py#L21

```
[REDACTED]
```

Figure C.17: chainport-backend/project/stats/cg_prices.py#L48

- **Calling of a non-callable object.** Figure C.18 shows the `cg_rest_call` function, which receives the `exception` parameter and calls it if an error occurs; however, `get_token_details_by_cg_id` passes in a Python object that is not callable.

```
[REDACTED]
```

Figure C.18: chainport-backend/modules/coingecko/api.py#L41-L42

- **Redefinition of variable before its use.** Figure C.19 shows a function in which the `optimal_gas_price` variable is redefined before it is used.

```
[REDACTED]
```

Figure C.19: chainport-backend/project/web3_cp/contracts/helpers.py#L108-L115

- **Duplicate code.** The exception handling in figure C.20 uses the same code for both

`TransactionNotFound` and `TimeExhausted`. Instead of duplicating the code, handle both exceptions by using `except (TransactionNotFound, TimeExhausted) as e:`.

```
[REDACTED]
```

Figure C.20: chainport-backend/project/lambdas/handlers/lambda_aws.py#L38-L48

- **Root user as the last Dockerfile user.** Leaving the last user of a Dockerfile as the root user is a security hazard. If an attacker gains control of the container, he will have root access and, therefore, a larger attack surface to attempt to escape the container. We add this issue as a code quality recommendation because the container is intended to be used only during local development and is unlikely to be reachable by attackers. Nonetheless, to prevent attacks and to follow best practices, we recommend leaving the last user of the Dockerfile as one without root privileges.

ChainPort Front End

- **Use of `debugger` statement.** The `debugger` statement should not be used in production code. This statement is used in `chainport-app/src/redux/actions/transactionActions.js#L573`.
- **Unused parameter.** The `unchecked` parameter in `chainport-app/src/components/CheckBox/CheckBox.jsx#L5` is not used. If it is not needed, remove it.
- **Useless assignment to local variables.** In the following code locations, the code assigns a value to a local variable that is never used:
  - `chainport-app/src/modules/exchange/containers/ContainerExchangePortInView.jsx#L268`
  - `chainport-app/src/redux/actions/transactionActions.js#L306`
  - `chainport-app/src/redux/actions/transactionActions.js#L374`
  - `chainport-app/src/redux/actions/transactionActions.js#L428`
  - `chainport-app/src/redux/actions/transactionActions.js#L478`
  - `chainport-app/src/redux/actions/transactionActions.js#L553`
- **Superfluous trailing argument.** Several function calls use too many arguments. In particular, figure C.21 shows a call to `parseFloat` with two arguments (`parseFloat` should receive only one argument), where the second argument was

meant for the `bridge.fromWei` call instead. This list includes other instances of this problem:

- chainport-app/src/redux/actions/myActivityActions.js#L40
- chainport-app/src/redux/actions/myActivityActions.js#L92
- chainport-app/src/redux/actions/myActivityActions.js#L94
- chainport-app/src/redux/actions/myActivityActions.js#L98
- chainport-app/src/redux/actions/myActivityActions.js#L336
- chainport-app/src/redux/actions/myActivityActions.js#L568
- chainport-app/src/redux/actions/myActivityActions.js#L601
- chainport-app/src/services/utilsService.js#L406

```
[REDACTED]
```

Figure C.21: chainport-app/src/redux/actions/myActivityActions.js#L40

- **Unused dependencies.** The package.json file contains several unused dependencies.

Smart Contracts

- **Incorrect comments.** The inline comment in the `recoverSignatureWithdraw` function indicates that the function is used by the `verifyMint` function (figure C.22), which is incorrect; `recoverSignatureWithdraw` is used by the `verifyWithdraw` and `verifyWithdrawDeprecated` functions.

```
[REDACTED]
```

Figure C.22: smart-contracts/contracts/Validator.sol#L121-L155

- **Comparison of boolean values against boolean literals.** Several contracts compare boolean values against `true` or `false`. The boolean literals can be omitted to simplify these statements and make them easier to read. For example, the comparison in figure C.23 can be simplified to `!receipt.hasVoted`.

```
[REDACTED]
```

Figure C.23: smart-contracts/contracts/governance/ChainportCongress.sol#L216

```
[REDACTED]
```

Figure C.24: `smart-contracts//contracts/governance/ChainportCongress.sol#L92`

D. Semgrep Rule to Detect the Logging of Incorrect Function Names

The Semgrep rule in figure D.1 detects the logging of incorrect function names, a problem described in TOB-CHPT-3. This rule detects 46 incorrect uses of the logging pattern.

```
rules:
  - id: logging-wrong-function-name
    message: Wrong function name is logged
    languages: [python]
    severity: WARNING
    patterns:
        - pattern: >
            def $X(...):
                ...
                $Z = $Y.__name__
        - pattern-not: >
            def $X(...):
                ...
                $Z = $X.__name__
```

Figure D.1: Semgrep rule to detect the logging of incorrect function names

Run the following command on the base folder of the back-end repository to run this rule:

```
semgrep --metrics=off --config semgrep_find_log_of_wrong_func_name.yml
```

E. Token Integration Checklist

The following checklist provides recommendations for interactions with arbitrary tokens. Every unchecked item should be justified, and its associated risks, understood. For an up-to-date version of the checklist, see `crytic/building-secure-contracts`.

For convenience, all Slither utilities can be run directly on a token address, such as the following:

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken --erc erc20
slither-check-erc 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d KittyCore --erc erc721
```

To follow this checklist, use the below output from Slither for the token:

```
slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER]
slither [target] --print human-summary
slither [target] --print contract-summary
slither-prop . --contract ContractName # requires configuration, and use of Echidna
and Manticore
```

General Considerations

- ❏ **The contract has a security review.** Avoid interacting with contracts that lack a security review. Check the length of the assessment (i.e., the level of effort), the reputation of the security firm, and the number and severity of the findings.
- ❏ **You have contacted the developers.** You may need to alert their team to an incident. Look for appropriate contacts on `blockchain-security-contacts`.
- ❏ **They have a security mailing list for critical announcements.** Their team should advise users (like you!) when critical issues are found or when upgrades occur.

Contract Composition

- ❏ **The contract avoids unnecessary complexity.** The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither's `human-summary` printer to identify complex code.
- ❏ **The contract uses `SafeMath`.** Contracts that do not use `SafeMath` require a higher standard of review. Inspect the contract by hand for `SafeMath` usage.
- ❏ **The contract has only a few non-token-related functions.** Non-token-related functions increase the likelihood of an issue in the contract. Use Slither's `contract-summary` printer to broadly review the code used in the contract.

- ❏ **The token has only one address.** Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g., `balances[token_address][msg.sender]` may not reflect the actual balance).

Owner Privileges

- ❏ **The token is not upgradeable.** Upgradeable contracts may change their rules over time. Use Slither's `human-summary` printer to determine whether the contract is upgradeable.
- ❏ **The owner has limited minting capabilities.** Malicious or compromised owners can abuse minting capabilities. Use Slither's `human-summary` printer to review minting capabilities, and consider manually reviewing the code.
- ❏ **The token is not pausable.** Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pausable code by hand.
- ❏ **The owner cannot blacklist the contract.** Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand.
- ❏ **The team behind the token is known and can be held responsible for abuse.** Contracts with anonymous development teams or teams that reside in legal shelters require a higher standard of review.

ERC20 Tokens

ERC20 Conformity Checks

Slither includes a utility, `slither-check-erc`, that reviews the conformance of a token to many related ERC standards. Use `slither-check-erc` to review the following:

- ❏ **`Transfer` and `transferFrom` return a boolean.** Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail.
- ❏ **The `name`, `decimals`, and `symbol` functions are present if used.** These functions are optional in the ERC20 standard and may not be present.
- ❏ **`Decimals` returns a `uint8`.** Several tokens incorrectly return a `uint256`. In such cases, ensure that the value returned is below 255.
- ❏ **The token mitigates the known ERC20 race condition.** The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens.

Slither includes a utility, `slither-prop`, that generates unit tests and security properties that can discover many common ERC flaws. Use `slither-prop` to review the following:

- ❑ **The contract passes all unit tests and security properties from `slither-prop`.** Run the generated unit tests and then check the properties with Echidna and Manticore.

Risks of ERC20 Extensions

The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ❑ **The token is not an ERC777 token and has no external function call in `transfer` or `transferFrom`.** External calls in the transfer functions can lead to reentrancies.
- ❑ **`Transfer` and `transferFrom` should not take a fee.** Deflationary tokens can lead to unexpected behavior.
- ❑ **Potential interest earned from the token is taken into account.** Some tokens distribute interest to token holders. This interest may be trapped in the contract if not taken into account.

Token Scarcity

Reviews of token scarcity issues must be executed manually. Check for the following conditions:

- ❑ **The supply is owned by more than a few users.** If a few users own most of the tokens, they can influence operations based on the tokens' repartition.
- ❑ **The total supply is sufficient.** Tokens with a low total supply can be easily manipulated.
- ❑ **The tokens are located in more than a few exchanges.** If all the tokens are in one exchange, a compromise of the exchange could compromise the contract relying on the token.
- ❑ **Users understand the risks associated with a large amount of funds or flash loans.** Contracts relying on the token balance must account for attackers with a large amount of funds or attacks executed through flash loans.
- ❑ **The token does not allow flash minting.** Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token.