



Elixir Protocol

Security Assessment

October 10, 2024

Prepared for:

Chris Gilbert

Elixir Technologies Ltd.

Prepared by: **Artur Cygan, Damilola Edwards, Bo Henderson, and Emilio López**

About Trail of Bits

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

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

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

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

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

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

Trail of Bits, Inc.

497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215

<https://www.trailofbits.com>

info@trailofbits.com

Notices and Remarks

Copyright and Distribution

© 2024 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to Elixir Technologies Ltd under the terms of the project statement of work and has been made public at Elixir Technologies Ltd's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

The sole canonical source for Trail of Bits publications is the [Trail of Bits Publications page](#). Reports accessed through any source other than that page may have been modified and should not be considered authentic.

Test Coverage Disclaimer

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

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

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

Table of Contents

About Trail of Bits	1
Notices and Remarks	2
Table of Contents	3
Project Summary	5
Executive Summary	6
Project Goals	9
Project Targets	10
Project Coverage	11
Automated Testing	13
Codebase Maturity Evaluation	15
Summary of Findings	18
Detailed Findings	20
1. Use of outdated dependencies	20
2. Use of HTTP requests without timeout	22
3. Private keys stored in environment variables	24
4. Majority proposal group will always satisfy minimum size requirement	25
5. Authentication at risk of replay attacks if misconfigured	27
6. Strategy executor does not validate payload to sign	29
7. Missing IP address validation allows bypassing Redpanda ACL	30
8. Use of assert statement in production code	31
9. Redpanda accounts and associated permissions are never revoked	32
10. Delegators can redelegate stakes to jailed delegatee	34
11. Attackers can cause slashing to become economically infeasible	35
12. Delegators can immediately undelegate before their delegatee is jailed	37
13. Minority validators may participate in consensus process	38
14. An influx of new strategy executors may halt consensus	40
15. Lack of two-step process for ownership transfers	42
16. Response payload to authentication challenge is not signed	43
17. API does not validate display_name and app_version	44
18. All on-chain events replayed upon startup	45
19. Redpanda exposed to the Internet	47
20. API has admin access to Redpanda	48
21. Use of unpinned third-party Docker images and actions on workflows	50

22. Absence of access controls on pool creation function	52
A. Vulnerability Categories	54
B. Code Maturity Categories	56
C. Code Quality Recommendations	58
D. Automated Analysis Tool Configuration	61
E. Mutation Testing	63
F. Incident Response Recommendations	65
G. Security Best Practices for Using Multisignature Wallets	67
H. Fix Review Results	69
Detailed Fix Review Results	71
I. Fix Review Status Categories	74

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Keith Hoodlet, Engineering Director, Application Security
keith.hoodlet@trailofbits.com

Josselin Feist, Engineering Director, Blockchain
josselin@trailofbits.com

The following consultants were associated with this project:

Artur Cygan, Consultant
artur.cygan@trailofbits.com

Damilola Edwards, Consultant
damilola.edwards@trailofbits.com

Bo Henderson, Consultant
bo.henderson@trailofbits.com

Emilio López, Consultant
emilio.lopez@trailofbits.com

Project Timeline

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

Date	Event
July 24, 2024	Pre-project kickoff call
August 2, 2024	Status update meeting #1
August 9, 2024	Delivery of report draft
August 9, 2024	Report readout meeting
October 10, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

Elixir Technologies Ltd engaged Trail of Bits to review the security of Elixir Protocol. Elixir is a consensus-based peer-to-peer networking protocol that performs algorithmic trades on centralized and decentralized exchanges.

A team of four consultants conducted the review from July 29 to August 7, 2024, for a total of five engineer-weeks of effort. Our testing efforts focused on the staking and delegation methods of the smart contracts, user authentication, and soundness of the consensus mechanism. With full access to source code and documentation, we performed static and dynamic testing of the `elixir-protocol` and `elixir-protocol-api` repositories, using automated and manual processes.

Observations and Impact

Our review of Elixir Protocol uncovered several high-severity issues, although most are also highly difficult to exploit. However, one high-severity issue requires only low difficulty to exploit; this issue allows delegators to bypass the redelegate cooldown period (TOB-ELIXIR-12) by immediately signaling after the initial delegation. Other noteworthy findings include a lack of data validation that enables the strategy executor to sign arbitrary payloads provided by the API (TOB-ELIXIR-6).

We identified two issues relating to the consensus mechanism. One relates to inadequate deauthorization of users who were previously in the set of core validators, allowing minority validators to participate in consensus (TOB-ELIXIR-13). A second issue in the quorum requirement could cause the consensus mechanism to halt if the queue of active, non-core validators grows too large (TOB-ELIXIR-14).

The system exposes Redpanda to the internet; this design decision led to several findings (TOB-ELIXIR-9, TOB-ELIXIR-19, and TOB-ELIXIR-20).

All in all, the codebase is well-organized and features thorough code comments, which aided our security review. However, the system needs to be tested more thoroughly prior to deployment. Some issues, such as an incorrect calculation of the minimum proposal requirement in the majority group (TOB-ELIXIR-4), could have been caught earlier by more thorough unit tests.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Elixir Technologies Ltd take the following steps before deploying the protocol to production:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Expand the test suite.** We identified gaps in the smart contract test coverage using mutation tests. Write additional tests that assert the expected behavior of every branch of the business logic, and then use code coverage assessments to confirm that gaps in test coverage have been closed.
- **Write documentation.** Although the code is well commented, high-level documentation is necessary to ensure that end users can safely interact with Elixir protocol.
 - Write goal-oriented usage manuals, one for each type of end user (e.g., staker, validator, trader). These should provide clear how-to instructions that guide users through every action they might need to take and should clearly outline the risks associated with any given action with accompanying guidance on mitigating or eliminating these risks.
 - Create an API reference that documents all smart contract methods and API endpoints. The parameters, return values, and side effects should be listed in one place that is easy to search. In addition, interactions between components should be described to complement the existing system diagrams. This will help current developers add new features safely, aid future developers in ramping up quickly, and help other security reviewers to assess the codebase productively.
 - Create an incident response plan. See [appendix F](#) for guidance.
- **Assign ownership to a multisignature wallet.** Contract upgrades are extremely sensitive operations and do not require frequent or automated execution, so multisignature wallets would be a good fit for ownership roles that can authorize upgrades. See [appendix G](#) for guidance regarding the secure usage of multisignature wallets.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

<i>Severity</i>	<i>Count</i>
High	8
Medium	4
Low	2
Informational	5
Undetermined	3

CATEGORY BREAKDOWN

<i>Category</i>	<i>Count</i>
Access Controls	2
Configuration	4
Cryptography	1
Data Exposure	1
Data Validation	7
Denial of Service	4
Patching	1
Session Management	1
Timing	1

Project Goals

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

- Are there any error-prone or incorrect steps in the deployment and initialization of the system's smart contracts?
- Is there any way for a user to bypass the redelegate and unstake cooldown periods?
- Does off-chain code manage keys, use cryptographic primitives, and interact with smart contracts safely?
- Could an attacker steal from users or deny them access to their staked funds?
- Is all user-provided input properly validated?
- Can an attacker capture or halt the system's consensus mechanism?
- Are any access controls missing, and do any roles have unsuitable permissions?
- Can any system actions be front-run, or are there other race conditions that an attacker could take advantage of?

Project Targets

The engagement involved a review and testing of the targets listed below.

protocol

Repository	https://github.com/ElixirProtocol/elixir-protocol
Version	99e669d6a1cd88011a395fc74cf374372f8b5c52
Type	Python, Solidity
Platform	EVM, Linux

protocol-api

Repository	https://github.com/ElixirProtocol/elixir-protocol-api
Version	f508fc1a120886688da020f55dd22dbb0f303ccd
Type	Python
Platform	Linux

Project Coverage

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

- **Smart contracts.** We ran slither and slither-mutate to automatically identify points of interest and gaps in the test coverage to guide our manual review.
 - **Core:** A permissioned actor that controls administrative functions such as setting important addresses and slashing users. We reviewed the initialization process, access controls, and this contract's interactions with other components.
 - **DisputeManager:** This contract allows anyone with a valid `ConsensusAuditDetails` struct to trigger the jailing of misbehaving validators. We reviewed the signature validation logic and address checks to assess whether invalid parameters can be used to prevent jailing a dishonest validator or to cause an honest actor to be jailed.
 - **Pool** and **PoolManager:** These contracts act as an auditable record of all pools and their configuration parameters. Since these components contain little logic, we spent less time reviewing them beyond determining if initialization and ownership methods behave as expected.
 - **StakeManager:** This contract contains critical methods that tie directly into the consensus mechanism and was the primary focus of our smart contract review. We assessed the movement of funds into and out of the contract, looking for opportunities to steal or lock user funds. We listed the different contract states and transitions between them to identify ways that a user could bypass the signal delays, delegate more than they staked, or otherwise disrupt the system.
- **Protocol services.** These Python modules implement the core off-chain business logic of the Elixir protocol. The two services we looked at most closely are the following:
 - **Consensus processor:** We looked for ways that an attacker could abuse delegations or perform sybil attacks to disrupt the consensus building process. We reviewed the arithmetic supporting validator and proposal counts along with the overall business logic searching for ways that an attacker could capture or halt consensus.

- **Strategy executor:** This component ingests data frames and produces order proposals. We reviewed the strategy executor paying particular attention to its authorization with the protocol api.
- **Protocol API:** The `protocol-api` manages user authentication and acts as a gateway for interactions with ancillary Redis and database services. We searched for ways that authentication could be bypassed and carefully studied this component's interactions with the consensus service.

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 performed only automated analysis and light manual review of the protocol services that were not explicitly listed above.
- The order proposal generation and GLT strategy code was not a primary focus of our review.
- We did not thoroughly review files managing the production environment such as Dockerfiles and other cloud config files.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Slither	A static analysis framework that can statically verify algebraic relationships between Solidity variables	Appendix D
slither-mutate	A deterministic mutation generator that detects gaps in test coverage	Appendix E
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	Appendix D
Bandit	A static analysis tool designed to find common security issues in Python code	Appendix D
CodeQL	A code analysis engine developed by GitHub to automate security checks	Appendix D

Areas of Focus

Our automated testing and verification work focused on the following system properties:

- Identify common issues and anti-patterns in Solidity and Python
- Identify gaps in test coverage to guide our review efforts

Test Results

The results of this focused testing are detailed below.

slither-mutate: The following table displays the portion of each type of mutant for which all unit tests passed. The presence of valid mutants indicates that there are gaps in test coverage because the test suite did not catch the introduced change.

- Uncaught revert mutants replace a given expression with a `revert` statement and indicate that the line is not executed during testing.
- Uncaught comment mutants comment out a given expression and indicate that the effects of this line are not checked by any assertions.
- Uncaught tweak mutants indicate that the expression being executed features edge cases that are not covered by the test suite.

The `protocol/contracts/src` subdirectory is the root for all target paths listed below. Contracts supporting tests or that produced zero analyzed mutants (e.g., interfaces) were omitted from mutation testing analysis.

Target	Uncaught Reverts	Uncaught Comments	Uncaught Tweaks
ELX.sol	100%	100%	100%
Pool.sol	0%	42%	14%
PoolManager.sol	11%	50%	N/A
StakeManager.sol	0%	22%	11%
Core.sol	24%	45%	32%
DisputeManager.sol	0%	29%	0%

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	<p>The protocol uses a modern version of Solidity without any unchecked arithmetic blocks, preventing overflows. Smart contract arithmetic uses only simple addition and subtraction, eliminating the possibility of issues due to rounding errors.</p> <p>We observed that the protocol services use Python's <code>Decimal</code> to help prevent unexpected rounding issues in off-chain monetary calculations. However, an incorrect variable was used in one consensus-impacting calculation (TOB-ELIXIR-4), indicating that the off-chain arithmetic would benefit from more thorough tests.</p>	Moderate
Auditing	<p>The Elixir smart contracts consistently emit events that include enough data for effective off-chain monitoring. Event and parameter names provide context, although the system would benefit from additional documentation regarding the implications of each event as well as an incident response plan.</p> <p>The off-chain components print sufficient logs for monitoring the internal state.</p>	Satisfactory
Authentication / Access Controls	<p>The Core contract manages granular access roles that follow the principle of least privilege. We did not identify any missing or misconfigured access controls, although a two-step process for ownership transfers would help prevent irrevocable mistakes (TOB-ELIXIR-15).</p> <p>Multiple issues were identified in off-chain authentication code. For example, the <code>protocol-api</code> auth service does not revoke access when, for example, a validator's delegations drop out of the top 100 or their IP address</p>	Weak

	changes (TOB-ELIXIR-9).	
Complexity Management	<p>Smart contracts and their methods have well-defined responsibilities that rely on clear and consistent naming conventions. We identified few instances of duplicate code or functions with high cyclomatic complexity.</p> <p>Each Python service adheres to a predictable and well-organized structure, aiding review of the codebase.</p>	Satisfactory
Configuration	Example environment variable values clarify developer intent and aided our review. However, we identified some instances of error-prone configurations, such as the Redis activation; (TOB-ELIXIR-5).	Moderate
Cryptography and Key Management	Well-established libraries are used to perform signing operations and to interact with other cryptographic primitives. However, private keys pass through environment variables during startup, increasing the risk of key exposure (TOB-ELIXIR-3). We identified a place where parts of the payload are not authenticated (TOB-ELIXIR-16).	Moderate
Data Handling	Some signing operations happen without sufficient payload validation (TOB-ELIXIR-6). There is no clearly defined data validation layer for untrusted user input in the API, which led to TOB-ELIXIR-7 and TOB-ELIXIR-17.	Weak
Decentralization	The smart contracts depend on off-chain services to function, one component of which is a centralized message delivery hub. All system logic, including the management of user ELX tokens, is upgradable by a single account without giving users the chance to opt out.	Weak
Documentation	Smart contract methods feature NatSpec comments, and Python functions are described with docstrings. The system architecture diagram provided context that aided our review. However, very little high-level documentation was provided.	Moderate
Low-Level Manipulation	The smart contracts do not rely on any EVM assembly code nor on any low-level calls, apart from those included in well established libraries.	Satisfactory

Maintenance	The code is organized into logical modules, making it relatively straightforward to maintain. However, outdated Python dependencies (TOB-ELIXIR-1) indicate the lack of a process for updating dependencies in a timely manner; the system could benefit from the use of tools such as Dependabot in CI to automatically keep dependencies up to date in the repository.	Moderate
Memory Safety and Error Handling	The protocol services are written in Python, a memory-safe language, eliminating the possibility of low-level memory problems. Error handling is generally performed appropriately.	Satisfactory
Testing and Verification	Both Solidity and Python portions of the codebase feature unit tests. However, mutation testing uncovered some important gaps in Solidity test coverage, and some of the issues we identified could have been found earlier with more thorough tests (TOB-ELIXIR-4).	Moderate
Transaction Ordering	We identified issues in the smart contracts that either arise from, or are made more severe due to, transaction-ordering risks. The system would benefit from further investigation and more rigorous specification of timing risks.	Moderate

Summary of Findings

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

ID	Title	Type	Severity
1	Use of outdated dependencies	Patching	Informational
2	Use of HTTP requests without timeout	Denial of Service	Informational
3	Private keys stored in environment variables	Data Exposure	High
4	Majority proposal group will always satisfy minimum size requirement	Data Validation	High
5	Authentication at risk of replay attacks if misconfigured	Configuration	High
6	Strategy executor does not validate payload to sign	Data Validation	High
7	Missing IP address validation allows bypassing Redpanda ACL	Data Validation	Medium
8	Use of assert statement in production code	Data Validation	Undetermined
9	Redpanda accounts and associated permissions are never revoked	Access Controls	Undetermined
10	Delegators can redelegate stakes to jailed delegatee	Data Validation	Medium
11	Attackers can cause slashing to become economically infeasible	Denial of Service	Low
12	Delegators can immediately undelegate before their delegatee is jailed	Timing	High

13	Minority validators may participate in consensus process	Session Management	High
14	An influx of new strategy executors may halt consensus	Denial of Service	Medium
15	Lack of two-step process for ownership transfers	Data Validation	Informational
16	Response payload to authentication challenge is not signed	Cryptography	Informational
17	API does not validate display_name and app_version	Data Validation	Undetermined
18	All on-chain events replayed upon startup	Denial of Service	Medium
19	Redpanda exposed to the Internet	Configuration	High
20	API has admin access to Redpanda	Configuration	High
21	Use of unpinned third-party Docker images and actions on workflows	Configuration	Low
22	Absence of access controls on pool creation function	Access Controls	Informational

Detailed Findings

1. Use of outdated dependencies	
Severity: Informational	Difficulty: Undetermined
Type: Patching	Finding ID: TOB-ELIXIR-1
Target: Multiple <code>pyproject.toml</code> , <code>poetry.lock</code> , Dockerfiles	

Description

We used `pip audit` to detect the use of outdated dependencies in the Python codebases, which identified a number of vulnerable packages referenced by the `pyproject.toml` and `poetry.lock` files.

The following is a list of the vulnerable dependencies used in the codebase, and known vulnerabilities that affect the versions in use:

- `certifi` (GHSA-248v-346w-9cwc)
- `pysha3` (GHSA-6w4m-2xhg-2658)
- `requests` (GHSA-9wx4-h78v-vm56)
- `urllib3` (GHSA-34jh-p97f-mpxf)

In many cases, the use of a vulnerable dependency does not necessarily mean the application is vulnerable. Vulnerable methods from such packages need to be called within a particular (exploitable) context. To determine whether the off-chain applications are vulnerable to these issues, each issue will have to be manually triaged. The severity is marked informational as upon preliminary inspection, these issues do not appear to impact the codebase directly.

Moreover, the services **use an outdated version of Python (3.10.12)**, which was released on June 6, 2023. A newer version (3.10.14), released on March 19, 2024, is available; additionally, version 3.12.4, released on June 6, 2024, can be used if upgrading to the minor version is feasible.

Recommendations

Short term, update system dependencies to their latest versions wherever possible. Use tools such as `pip audit` to confirm that no vulnerable dependencies remain.

Long term, implement these checks as part of the CI/CD pipeline of application development. Integrate an automated solution such as Dependabot into your development

process to assist in promptly detecting and updating dependencies with known security problems.

2. Use of HTTP requests without timeout

Severity: Informational

Difficulty: High

Type: Denial of Service

Finding ID: TOB-ELIXIR-2

Target: elixir-protocol and elixir-protocol-api Python applications

Description

The Elixir off-chain code uses the Python requests library to send HTTP requests. By default, **requests do not time out** unless the timeout parameter is explicitly passed to the method for a request. This behavior may cause the program to hang indefinitely when the requested server is slow or unresponsive. The programs send requests without a timeout in the following locations:

- To create a new user in Redpanda:
`elixir-protocol-api/app/api/services/auth.py#L103`
- To reset an existing user's password:
`elixir-protocol-api/app/api/services/auth.py#L111`
- To cancel all orders in Vertex:
`elixir-protocol/services/failsafe_exchange_processor/failsafe_exchange_processor/vertex/vertex_failsafe.py#L98`
- To fetch the pool configuration from IPFS:
`elixir-protocol/services/onchain_pool_config_feed/onchain_pool_config_feed/listeners/contract.py#L59`
- To create new orders in Vertex:
`elixir-protocol/services/order_reconciler/order_reconciler/exchanges/vertex/vertex_reconciler.py#L315`
- To cancel orders in Vertex:
`elixir-protocol/services/order_reconciler/order_reconciler/exchanges/vertex/vertex_reconciler.py#L368`
- To request an authentication challenge as a strategy executor:
`elixir-protocol/services/strategy_executor/strategy_executor/elixir_api.py#L22`
- To submit a response to an authentication challenge as a strategy executor:
`elixir-protocol/services/strategy_executor/strategy_executor/elixir_api.py#L55`

Recommendations

Short term, pass an explicit `timeout` parameter to all `requests` methods.

Long term, integrate static analysis tools such as **Bandit** into your software development life cycle and CI/CD processes. They can detect issues such as this one early on in the development process.

3. Private keys stored in environment variables

Severity: High

Difficulty: High

Type: Data Exposure

Finding ID: TOB-ELIXIR-3

Target: Off-chain services signing keys

Description

Elixir's off-chain services are configured via environment variables that include sensitive data, such as private keys, for signing. Environment variables are not suitable for storing sensitive data, as they may leak via multiple channels such as crash reports. Examples of this configuration include the `CONSENSUS_PROCESSOR_PRIVATE_KEY`, `STRATEGY_EXECUTOR_PRIVATE_KEY`, and `FEED_AGGREGATOR_PRIVATE_KEY` environment variables, which are used to sign critical data.

Exploit Scenario

The feed aggregator service crashes, and the crash report containing `FEED_AGGREGATOR_PRIVATE_KEY` key material is exported to a third-party analytics service.

Recommendations

Short term, resign from using environment variables for configuring key material. Instead, fetch the key material on service startup from a dedicated secret storage service. This will mitigate the issues with leaky environment variables.

Long term, use a dedicated service for signing instead of storing key material in individual services. This will protect the key material from leaking via other channels such as code execution inside the services. Examples of managed signing services include [Google Cloud Key Management Service](#) or [AWS CloudHSM](#).

4. Majority proposal group will always satisfy minimum size requirement

Severity: High

Difficulty: High

Type: Data Validation

Finding ID: TOB-ELIXIR-4

Target:

`elixir-protocol/services/consensus_processor/consensus_processor/consensus_builder.py`

Description

The consensus builder contains a check to ensure that the majority proposal group is large enough to form consensus. However, the check performed is incorrect and will never fail.

```
# determine if enough the majority proposal group is big enough to form consensus
consensus_proposal_group = majority_proposal_groups[0]
proposal_count = len(consensus_proposal_group.proofs)
minimum_requirement = math.ceil(proposal_count * self.min_consensus_threshold)
if proposal_count < minimum_requirement:
    self.logger.info(
        "Need to wait for more proposals, majority proposal group only has
        {proposal_count} proposals, {minimum_requirement} proposals are required",
        data_frame_id=data_frame_id,
    )
    return None
```

Figure 4.1: The check to ensure that the majority proposal is large enough to form consensus (`elixir-protocol/services/consensus_processor/consensus_processor/consensus_builder.py#L403-L412`)

This check compares the proposal count with the minimum requirement, which is computed as a percentage of the proposal count. The proposal count will therefore never be less than the minimum requirement.

Exploit Scenario

An attacker, who controls a small number of validators and is capable of confusing other validators to produce invalid proposals, obtains a small majority in the system. The check to ensure that the majority is large enough does not trigger adequately, and the attacker's proposal is accepted.

Recommendations

Short term, correct the check so that it correctly computes the minimum requirement, possibly as a percentage of the total proposals.

Long term, implement unit tests for important functionality to ensure that it behaves as intended.

5. Authentication at risk of replay attacks if misconfigured

Severity: High

Difficulty: High

Type: Configuration

Finding ID: TOB-ELIXIR-5

Target: elixir-protocol-api authentication

Description

The elixir-protocol-api authentication is vulnerable to replay attacks if the service is ever run with `config.redis_enabled` set to `False`. The implementation of the `connect` and `verify` endpoints enables running in this configuration without reporting any error (figures 5.1 and 5.2). In such a case, the nonce is set to `None`, and `connect` returns the message "Elixir Protocol Verification Message. None" to sign. The same happens in `verify`, which will validate messages with `None` as a nonce.

```
def connect(self, address: str):
    """Create a message for the validator to sign."""
    nonce = None
    key = f"validator-nonce-{address}"

    # check to see if we have a nonce for this validator.
    if self.config.redis_enabled:
        redis = self.redis_service.get_redis()
        nonce = redis.get(key)

        if nonce is None:
            nonce = uuid.uuid4()
            redis.set(key, nonce, ex=None)

    return {"message": f"{SIGNED_MESSAGE} {nonce}", "nonce": nonce, "address":
address}
```

Figure 5.1: The vulnerable `connect` endpoint function
([elixir-protocol-api/app/api/services/auth.py#L45-L59](#))

```
nonce = None
redis = self.redis_service.get_redis()

# search for saved nonce value
if self.config.redis_enabled:
    nonce = redis.get(f"validator-nonce-{address}")

    if not nonce:
        raise HTTPException(status_code=401, detail="INVALID_NONCE")
```

```
# recover the address
message = encode_defunct(text=f"{SIGNED_MESSAGE} {nonce}")
recovered_address = self.web3.w3.eth.account.recover_message(message,
signature=signature.replace(' ', ''))
```

*Figure 5.2: The vulnerable verify endpoint function
([elixir-protocol-api/app/api/services/auth.py#L167-L179](#))*

Exploit Scenario

An attacker captures a signed “Elixir Protocol Verification Message. None” message and replays it while the API service is misconfigured with `config.redis_enabled` set to `False`. The attacker then gains access to the Redpanda event bus.

Recommendations

Short term, reject `connect` and `verify` requests with 500 or 503 status code if Redis is disabled in a production environment. Disabling Redis should not make the system insecure.

Long term, explicitly mark insecure configuration options with unsafe wording and make it impossible to turn them on outside of the development environment.

6. Strategy executor does not validate payload to sign

Severity: High

Difficulty: High

Type: Data Validation

Finding ID: TOB-ELIXIR-6

Target:

elixir-protocol/services/strategy_executor/strategy_executor/elixir_api.py

Description

The strategy executor service signs the authentication message from the API without validating its structure (figure 6.1). The server can abuse this functionality to sign arbitrary EIP-191 personal_sign messages and use them as if they were originally signed by the strategy executor.

```
authentication_message = authentication["message"]
validator_address = get_address_from_key(private_key)
self.logger.info("Sending authorization request", address=validator_address)

# send signed message back to api for verification
message = encode_defunct(text=str(authentication_message))
signature = Web3().eth.account.sign_message(message, private_key=private_key)
```

Figure 6.1: Strategy executor signing data without validating its structure

(elixir-protocol/services/strategy_executor/strategy_executor/elixir_api.py#L47-L53)

Exploit Scenario

An attacker who controls the API replies with an authentication message that is a malicious OrderProposalDetail payload. The payload is signed by the strategy executor and returned to the attacker. The attacker can submit a fraudulent order proposal on behalf of the executor because, during the regular operation, the strategy executor **signs this structure** with the same key.

Recommendations

Short term, validate the structure of the authentication message returned by the API server. Make sure to validate the length and structure of the nonce to prevent injection attacks.

Long term, ensure that the payload structure is always validated before each signing operation.

7. Missing IP address validation allows bypassing Redpanda ACL

Severity: Medium

Difficulty: Low

Type: Data Validation

Finding ID: TOB-ELIXIR-7

Target: `elixir-protocol-api/app/api/services/auth.py`

Description

The IP address used to create Redpanda ACLs is not validated, which allows bypassing the host restriction from ACLs (figure 7.1). Because the IP address can be an arbitrary string, it is possible to send an all hosts wildcard (*) instead of a concrete IP address.

```
def create_acls(self, username: str, ip_address: str):
    """Create ACLs in redpanda for user."""
    self.logger.info("Creating ACLs")
    admin_client = KafkaAdminClient(bootstrap_servers=self.config.event_bus_brokers)

    read_acl = ACL(
        principal=f"User:{username}",
        host=ip_address,
        operation=ACLOperation.READ,
        permission_type=ACLPermissionType.ALLOW,
        resource_pattern=ResourcePattern(ResourceType.TOPIC, DATA_FRAME_TOPICS),
    )

    write_acl = ACL(
        principal=f"User:{username}",
        host=ip_address,
        operation=ACLOperation.WRITE,
        permission_type=ACLPermissionType.ALLOW,
        resource_pattern=ResourcePattern(ResourceType.TOPIC, ORDER_PROPOSAL_TOPICS),
    )
```

*Figure 7.1: Unvalidated IP address ends up straight in ACLs
(`elixir-protocol-api/app/api/services/auth.py#L125-L144`)*

Exploit Scenario

A malicious strategy executor sends a * instead of a concrete IP address to verify an endpoint. They use the Redpanda credentials to execute a distributed denial-of-service attack.

Recommendations

Short term, require the IP address sent by strategy executors to be a concrete value, or take the IP address from the connection instead of the user having to provide it.

8. Use of assert statement in production code

Severity: Undetermined

Difficulty: High

Type: Data Validation

Finding ID: TOB-ELIXIR-8

Target:

`elixir-protocol/services/strategy_executor/strategy_executor/glt_strategy.py,`
`elixir-protocol/shared/shared/event_scanner/event_scanner.py`

Description

When optimizations are requested with the `-O` flag, **assert statements** are removed from the program. The following functions use assert statements that will be skipped if optimizations are enabled.

- `GltStrategy.get_balance`
- `GltStrategy.get_price`
- `GltStrategy.get_metrics`
- `GltStrategy.apply_order_level`
- `EventScanner.scan`

Exploit Scenario

The strategy executor is run with optimizations enabled, and the `apply_order_level` function operates on an invalid proposal with a mismatched number of buys and sells, leading to undefined behavior.

Recommendations

Short term, change the assert statements to statements that explicitly raise exceptions. This will remove the risk that the validation layer will be optimized out.

Long term, integrate static analysis tools such as **Bandit** into your software development life cycle and CI/CD processes. They can detect issues such as this one early on in the development process.

9. Redpanda accounts and associated permissions are never revoked

Severity: **Undetermined**

Difficulty: **High**

Type: Access Controls

Finding ID: TOB-ELIXIR-9

Target: `elixir-protocol-api/app/api/services/auth.py`

Description

As part of the authentication process, when a strategy executor client calls the `/verify` API endpoint with a valid request that meets all the requirements to participate in the system, the back-end service will create or update an account in the Redpanda message broker for the strategy executor client to use, along with a pair of ACL rules that allow connections as this username from a certain IP address. However, there is no mechanism to expire or disable these accounts, so clients that stop participating or are jailed continue to have access to the event bus. Additionally, a new pair of ACL rules are created each time a user connects, and previous ACL rules are not cleared from Redpanda. As a result, an account may be accessible from multiple IP addresses even in absence of an issue like [TOB-ELIXIR-7](#).

Exploit Scenario #1

A malicious strategy executor operator repeatedly logs in with multiple concrete IP addresses to the `/verify` endpoint. As ACLs do not get cleared, the Redpanda user is now accessible from all the IP addresses provided by the malicious operator. They then distribute the credentials obtained on the last login operation and execute a distributed denial-of-service attack.

Exploit Scenario #2

A strategy executor operating with address `0xA` and a token delegation from address `0xB` is compromised and the attacker extracts the Redpanda credentials in use. The operator contains the compromise by rotating the strategy executor keys, which will now use wallet `0xC`, and by moving `0xB`'s delegation to the new address. While the previous address is now unsuitable to log in through the protocol API back end, as it has no tokens delegated to it, the attacker can still use the Redpanda credentials to observe messages or perform a denial-of-service attack.

Recommendations

Short term, implement credential revocation. Old ACL rules for a user should be revoked when a new authenticated session is established. This will give users the ability to remove old entries in the case of compromise. In addition, access should be revoked if users are jailed or their delegations drop to zero.

Long term, review authorized permissions and the contexts in which they are valid. It is equally important to ensure that authorizations are revoked when valid contexts become invalid, as it is when valid contexts are originally established.

10. Delegators can redelegate stakes to jailed delegatee

Severity: **Medium**

Difficulty: **Low**

Type: Data Validation

Finding ID: TOB-ELIXIR-10

Target: `elixir-protocol/contracts/src/StakeManager.sol`

Description

The internal `_updateDelegation` helper method does not prevent users from delegating to a validator that is jailed, allowing honest users to accidentally lose access to their stake.

The `delegate` function is responsible for handling both delegation and redelegation actions. During redelegation, several validation checks are performed: ensuring that the redelegation has been signaled in advance, confirming that the current delegatee of the delegator is not jailed, and verifying the delegator has not signaled an unstake. However, the status of the new delegatee, to whom the delegator intends to move their stakes, is not checked if they are jailed. This allows for redelegation to a jailed validator, leading to potential loss of funds for the delegator.

Exploit Scenario

Bob notices that Alice runs a validator with a good performance history, but he is unaware that Alice recently misbehaved. Bob sends a transaction delegating his stake to Alice right after another user sends a transaction jailing Alice, and both transactions succeed. As a result, Bob inadvertently loses access to his funds.

Recommendations

Short term, add a check to the `_updateDelegation` method that reverts a transaction that attempts to delegate to a validator who has been jailed.

Long term, strive to make it as hard as possible for an honest user to make a costly mistake. Pay particular attention to race conditions that occur during contract state transitions.

11. Attackers can cause slashing to become economically infeasible

Severity: Low

Difficulty: High

Type: Denial of Service

Finding ID: TOB-ELIXIR-11

Target: `elixir-protocol/contracts/src/StakeManager.sol`

Description

The slash function takes a single address as a parameter, alongside the amount to be slashed. When a significant number of users needs to be slashed, the slash function must be called each address individually. This approach becomes infeasible if there is a large number of small delegations.

```
/// @notice Slashes a staker
/// @param staker Staker to be slashed
/// @param amount Amount to be slashed
function slash(address staker, uint256 amount) external onlyCore {
    // Reduces staked balance.
    stakedBalance[staker] -= amount;

    // Reduced delegated balance.
    _moveDelegates(delegates[staker], address(0), amount);

    // Transfers the tokens.
    elixirToken.safeTransfer(msg.sender, amount);

    emit Slashed(staker, amount, stakedBalance[staker]);
}
```

Figure 11.1: The slash method (`StakeManager.sol#L271-L285`)

Based on the current design of the protocol, when a validator is jailed, all delegators who staked with that jailed validator will have their stakes locked and potentially slashed. However, in the event of slashing, each affected delegator needs to be slashed individually. The cost of performing slashing on the entire set of affected delegators increases proportionally with the number of delegators involved. In cases where the number of affected delegators is significantly large, the high cost may discourage the execution of slashing, allowing delegators to evade the intended penalty.

Exploit Scenario

Bob acquires ELX tokens with the intent of misbehaving. To grief the protocol by making it less cost effective for him to be slashed, he distributes small amounts of his tokens among a large number of accounts and delegates them all to himself. After Bob is jailed for acting maliciously, the cost of recovering ELX from his delegators is prohibitively expensive.

Recommendations

Short term, consider implementing an additional batch slash method in the StakeManager contract that accepts an array of addresses and slashes each down to a zero balance. Some arithmetic and conditional logic can be skipped, making this a cheaper and easier way to recover funds from a large number of misbehaving accounts.

Long term, keep in mind that a delegatee with a small number of large delegators may have an equal weight as a delegatee with a large number of small delegators. Consider both situations and ways an attacker may take advantage of differences such as gas economics.

12. Delegators can immediately undelegate before their delegatee is jailed

Severity: High

Difficulty: Low

Type: Timing

Finding ID: TOB-ELIXIR-12

Target: `elixir-protocol/contracts/src/StakeManager.sol`

Description

There is no penalty for preemptively signaling redelegation immediately after initial delegation. Doing so (and waiting through the cooldown) would give delegates the ability to front-run future jail transactions and avoid having their funds locked, effectively bypassing the delay that signaling is supposed to impose.

As part of the front-running protection mechanism designed to prevent delegators from unstaking or redelegating just before their delegatee is jailed, delegators are required to signal an undelegate and wait for the expiration of a waiting period before proceeding with the undelegation. However, the `signalUndelegate` function can be invoked at the start of the delegation period. Once the waiting period has passed, the undelegation/redelegation requirement is met, allowing the delegator to maintain this state and immediately undelegate or redelegate at any time. This is particularly problematic if the delegatee is about to be jailed, as the delegatee jail transaction can be front-run with an undelegate or redelegate transaction to avoid the penalty.

Exploit Scenario

Alice delegates her stake with Bob the delegatee, immediately after the delegation, she signals undelegate while maintaining her delegation with Bob even when the undelegation waiting period has passed. Bob is about to be jailed, she notices a jail transaction for Bob in the mempool and front-runs it with an undelegate transaction which gets executed first allowing her to effectively evade the jail

Recommendations

Short term, delegators that have signaled redelegate should be excluded from a validator's total delegations.

Long term, keep in mind that timestamps on the blockchain are error prone and a frequent source of errors. Review all timestamps used by the system and, for each, consider what would happen immediately before and after important thresholds as well as at the extremes. This will facilitate a review of the codebase and help prevent similar problems.

13. Minority validators may participate in consensus process

Severity: High

Difficulty: Medium

Type: Session Management

Finding ID: TOB-ELIXIR-13

Target:

`elixir-protocol/services/consensus_processor/consensus_processor/consensus_builder.py`

Description

The consensus builder receives and considers any proposals from active validators towards its goal of achieving consensus. Active validators are those with a non-zero delegation and which are not jailed. Considering that access to validators is not revoked if their delegated token amount becomes small (see [TOB-ELIXIR-9](#)), this could allow one or many minority strategy executors (i.e., those with a small delegation, and that should not be part of the “core” set) to participate in the consensus process. A malicious actor may take advantage of this fact to gain a majority of the consensus vote.

Exploit Scenario

Assuming that the top 99 validators have over 1,000 tokens delegated each, an attacker obtains 1,001 tokens. The attacker develops a malicious strategy executor software that will for the most part behave identically to the official software; however, when it receives a special signal, it will submit a malicious proposal.

The attacker then delegates their tokens to the address `0xA`, and runs a malicious strategy executor with this address. Once the executor is online, the attacker redelegates all but one cent of the token to a new address, `0xB`, and repeats the process. Each time, when the freshly-delegated-to validator attempts to log in to the system, this validator will be considered part of the “core” list and be allowed to connect to Redpanda, as their delegation will be one of the top 100 (`CORE_VALIDATOR_SET_SIZE`) largest by amount delegated. This process is repeated 200 times, until the attacker is operating a majority of the active validators that can participate in consensus.

The attacker can then choose to operate their strategy executors benevolently, and collect an outsized share of the system rewards. However, the attacker can also opt to signal to their validators when the market conditions are beneficial, and all of their executors will then submit a coordinated, malicious proposal, which becomes the majority proposal and gains consensus. The system then executes the malicious proposal, and potentially jails good validators, further strengthening the attacker’s position.

Recommendations

Short term, implement a mechanism to disconnect and revoke Redpanda access from strategy executors that should not be part of the core set. Enforce some sort of minimum delegation value to be able to participate in consensus.

Long term, run a threat modeling exercise covering the interaction of users with permissioned and permissionless components within the system. This exercise should identify a system's specific risks and the actors that could take advantage of them, both outside (e.g., validators, third-party providers) and from within (e.g., contract operators).

14. An influx of new strategy executors may halt consensus

Severity: Medium

Difficulty: Low

Type: Denial of Service

Finding ID: TOB-ELIXIR-14

Target: elixir-protocol-api/app/api/services/auth.py,
elixir-protocol/services/consensus_processor/consensus_processor/consensus_builder.py

Description

The consensus builder requires a minimum number of proposals to achieve consensus, as one of its many checks. This minimum requirement is computed as a percentage of active (i.e. with a non-zero delegation and not jailed) validators, as shown in figure 14.1.

```
def get_active_validators(self, data_frame: DataFrame):  
    return data_frame.data_frame_detail.active_validators.validator_addresses  
  
def get_min_required_proposals(self, data_frame: DataFrame):  
    active_validators = self.get_active_validators(data_frame)  
    return math.ceil(len(active_validators) * self.min_validators_threshold)  
  
def has_minimum_proposals(self, data_frame: DataFrame):  
    total_proposals = self.get_proposal_count(data_frame)  
    min_required_proposals = self.get_min_required_proposals(data_frame)  
    return total_proposals >= min_required_proposals
```

Figure 14.1: The check to ensure that there are enough proposals
([elixir-protocol/services/consensus_processor/consensus_processor/consensus_builder.py#317-327](#))

However, the authentication endpoint verification (/verify API route) will allow only a fixed number (CORE_VALIDATOR_SET_SIZE) of validators to log in and obtain Redpanda access; this value is currently set to 100. This means that, with a large enough number of validators, consensus may become unachievable, as 100 validators may not be enough to comply with the minimum number of proposals required for consensus.

```
# determine if the validator is in the core validator set  
validators = validator_service.get_validators(  
    offset=0, limit=CORE_VALIDATOR_SET_SIZE, order_by="total_delegated",  
    order_dir="desc"  
)  
matches = list(filter(lambda v: v.address == existing.address, validators))  
  
if not matches:
```

```
# create authentication event
self.create_authentication_event(existing.address, False)
raise HTTPException(status_code=401, detail="EXCLUDED_ADDRESS")
```

Figure 14.2: The /verify API endpoint will not let all validators in (elixir-protocol-api/app/api/services/auth.py#80-89)

Exploit Scenario

An attacker splits their tokens among 202 addresses and delegates them to 202 different validators, so that the minimum proposals required to achieve consensus increases by 101 (50% of 202). As a result, the system is halted and no further consensus can be achieved because only 100 validators may connect to Redpanda and the consensus builder cannot receive enough proposals in each round.

Recommendations

Short term, adjust the logic so that the minimum proposals required are a function of the smallest among the amount of participating validators or the core validator set size. Note that dynamically adjusting the size of the core validator set in relation to the total number of validators may not be enough, as malicious validators may purposely decide not to participate in consensus.

Long term, run a threat modeling exercise covering the interaction of users with permissioned and permissionless components within the system. This exercise should identify a system's specific risks and the actors that could take advantage of them, both outside (e.g., validators, third-party providers) and from within (e.g. contract operators).

15. Lack of two-step process for ownership transfers

Severity: Informational

Difficulty: High

Type: Data Validation

Finding ID: TOB-ELIXIR-15

Target: `elixir-protocol/contracts/src/*`

Description

When called, the `transferOwnership` function immediately sets the contract owner to the provided address. The use of a single step to make such a critical change is error-prone; if the function is called with erroneous input, the results could be irrevocable or difficult to recover from.

The issue is present in the following contracts:

- `Core`
- `DisputeManager`
- `ELX`
- `Pool`
- `PoolManager`
- `StakeManager`

The requirement for a separate acceptance step before role transfer is completed will guarantee that the new owner can properly execute transactions on the contracts it will be administering.

Exploit Scenario

Alice invokes `transferOwnership` to change the contract owner but accidentally enters the wrong address. She permanently loses the ability to set important system parameters and upgrade the contract.

Recommendations

Short term, implement a two-step process for all irrecoverable critical operations. Consider using the `Ownable2StepUpgradeable` OpenZeppelin dependency instead of `OwnableUpgradeable` for ownership management.

Long term, identify and document all possible actions that can be taken by privileged accounts, along with their associated risks. This will facilitate reviews of the codebase and prevent future mistakes.

16. Response payload to authentication challenge is not signed

Severity: Informational

Difficulty: High

Type: Cryptography

Finding ID: TOB-ELIXIR-16

Target:

`elixir-protocol/protocol/services/strategy_executor/strategy_executor/elixir_api.py`, `elixir-protocol-api/app/api/services/auth.py`

Description

During the authentication process, the strategy executor signs a challenge message with a nonce and returns it to the API to obtain credentials, together with other information such as its IP address, display name, beneficiary address, and application version. However, these additional details are not covered by the signature, so their authenticity could be compromised. This is, however, unlikely, as the exchange occurs over a TLS connection.

Exploit Scenario

An attacker on the same network as the strategy executor performs a machine-in-the-middle (MitM) attack to intercept and modify the challenge-response authentication messages. They do so by taking advantage of a compromised or malicious root CA installed on the strategy executor machine. The strategy executor requests a challenge from the server, signs it, and returns it together with their IP address, display name, beneficiary address, and application version. The attacker changes the beneficiary address to a different one under their control before passing the message over to the `/verify` endpoint. The server accepts the challenge, uses the modified beneficiary address, and pays out the validator rewards to the attacker.

Recommendations

Short term, include these extra details or a hash of them as part of the signed message, so that the protocol API can verify that they have not been tampered with. Consider pinning the certificates used by the protocol API in the strategy executor to ensure that the communications cannot be sent to an unexpected party.

17. API does not validate display_name and app_version

Severity: **Undetermined**

Difficulty: **High**

Type: Data Validation

Finding ID: TOB-ELIXIR-17

Target: elixir-protocol-api/app/api/services/auth.py

Description

During the authentication process, the protocol API server receives a “display name” string and the application version string provided by the strategy executor. These strings are arbitrary and they are not validated either in length or contents. An attacker may send large amounts of data to cause a denial of service, or abuse the unsanitized storage for malicious payloads.

Exploit Scenario

A malicious strategy executor submits a cross-site scripting (XSS) payload as their display name. That string is then rendered unsafely in a future administrative panel developed by Elixir, and malicious code gets executed in that context.

Recommendations

Short term, ensure that both strings conform to an expected format and maximum length, and sanitize their contents before storing them.

Long term, validate and sanitize any untrusted, user-provided input that is received by the system.

18. All on-chain events replayed upon startup

Severity: **Medium**

Difficulty: **High**

Type: Denial of Service

Finding ID: TOB-ELIXIR-18

Target: feed_aggregator, message_recorder, metrics_recorder, strategy_metrics services

Description

The feed_aggregator, message_recorder, metrics_recorder, and strategy_metrics services replay on startup all events from the beginning of the history (figure 18.1). This makes the services vulnerable to denial of service if the history takes a long time to process. In other words, the progress in processing events is not persisted throughout restarts of a service.

```
onchain_feed_consumer = SharedQueueConsumer(  
    brokers=config.event_bus_brokers,  
    topics=config.event_bus_onchain_feed_topics,  
    shared_data_queue=shared_data_queue,  
    offset=OFFSET_MODE_EARLIEST,  
)
```

Figure 18.1: The consumer configuration responsible for replaying all events
([elixir-protocol/services/feed_aggregator/feed_aggregator/__main__.py#L35-L40](#))

Exploit Scenario

An attacker finds a way to persist a large amount of events that are processed on startup in one of the vulnerable services. The attacker either finds a way to crash the service or waits for the service restart. The service takes a long time to start and it is not serving its purpose during that time.

Recommendations

Short term, update the services to persist the state built from the event stream. This can happen every event, after a batch of events, or periodically, depending on the performance requirements. It is important to persist the event stream offset atomically with the state built on the events up to the offset. If the cached state data schema changes, it can be either migrated or purged (since the event stream is the authoritative source). If it is purged, the state has to be rebuilt; however, since this is a one-time event, it should not significantly increase the risk of a denial-of-service attack.

Long term, add tests that check data integrity between the event history and the cached state and ensure that services are starting in a timely manner even in the face of massive amounts of data.

19. Redpanda exposed to the Internet

Severity: High

Difficulty: High

Type: Configuration

Finding ID: TOB-ELIXIR-19

Target: Redpanda

Description

The consequence of Elixir's off-chain system architecture is that the Redpanda broker is exposed to the internet, as the strategy executors connect directly to the broker. This makes Redpanda easily accessible to attackers, as they can directly try to exploit the service over the network. The consequences of this design decision also manifest in findings [TOB-ELIXIR-9](#) and [TOB-ELIXIR-20](#).

Exploit Scenario

A remotely exploitable vulnerability, such as [CVE-2023-50976](#), is discovered in Redpanda. Because Redpanda is accessible over the network to anyone, attackers easily exploit this vulnerability, potentially gaining write access or performing a denial-of-service attack.

Recommendations

Short term, hide Redpanda instances so they are accessible only on the internal network to the relevant services. Use ACLs to limit access to the minimum required by each service. Add functionality to the API service to stream data frames to and receive order proposals from strategy executors.

Long term, always limit public network access as much as possible and expose only the necessary APIs.

20. API has admin access to Redpanda

Severity: High

Difficulty: High

Type: Configuration

Finding ID: TOB-ELIXIR-20

Target: API service, Redpanda

Description

The API service has admin access to Redpanda to configure users (figure 20.1) and ACLs (figure 20.2). Having such powerful capability in a service exposed to the internet puts Redpanda at a high risk of exploitation.

```
def create_user(self, address: str):
    """Create a new user."""
    username = f"user_{address}"
    password = secrets.token_urlsafe(32)

    # create user
    self.logger.info("Creating user")
    create_response = requests.post(
        f"{self.config.admin_event_bus_brokers}/v1/security/users",
        json={"username": username, "password": password, "algorithm":
"SCRAM-SHA-256"},
    )

    # check response code
    if create_response.status_code == 500 and "User already exists" in
create_response.text:
        # update user information
        update_response = requests.put(
            f"{self.config.admin_event_bus_brokers}/v1/security/users/{username}",
            json={"password": password, "algorithm": "SCRAM-SHA-256"},
        )
```

*Figure 20.1: API uses admin endpoints to configure Redpanda users
(elixir-protocol-api/app/api/services/auth.py#L96-L123)*

```
def create_acls(self, username: str, ip_address: str):
    """Create ACLs in redpanda for user."""
    self.logger.info("Creating ACLs")
    admin_client = KafkaAdminClient(bootstrap_servers=self.config.event_bus_brokers)

    read_acl = ACL(
        principal=f"User:{username}",
        host=ip_address,
        operation=ACLOperation.READ,
```

```

        permission_type=ACLPermissionType.ALLOW,
        resource_pattern=ResourcePattern(ResourceType.TOPIC, DATA_FRAME_TOPICS),
    )

    write_acl = ACL(
        principal=f"User:{username}",
        host=ip_address,
        operation=ACLOperation.WRITE,
        permission_type=ACLPermissionType.ALLOW,
        resource_pattern=ResourcePattern(ResourceType.TOPIC, ORDER_PROPOSAL_TOPICS),
    )

    try:
        admin_client.create_acls([read_acl, write_acl])
    except Exception as e:
        self.logger.error("Error creating ACLs", exception=e)
        raise HTTPException(status_code=500, detail="BROKER_ERROR")

```

*Figure 20.1: API uses admin endpoints to configure ACLs
([elixir-protocol-api/app/api/services/auth.py#L125-L150](#))*

Exploit Scenario

An attacker finds a way to execute code in the context of an API service and uses it to reconfigure Redpanda to be accessible for any writes.

Recommendations

Short term, hide Redpanda inside the internal network, as detailed in the short-term recommendation for [TOB-ELIXIR-20](#). This will remove the need for dynamic configuration of Redpanda users and ACLs from the API, thus removing the need for having admin access.

Long term, avoid giving the power of administrative tasks to services exposed to the internet.

21. Use of unpinned third-party Docker images and actions on workflows

Severity: Low

Difficulty: High

Type: Configuration

Finding ID: TOB-ELIXIR-21

Target:

`elixir-protocol/.github/workflows/{abis.yaml, contracts.yaml, strategy_executor.yaml}`, multiple Dockerfiles

Description

Several workflows in the repository directly use third-party actions such as `stefanzweifel/git-auto-commit-action@v5`, `foundry-rs/foundry-toolchain@v1`, `docker/setup-qemu-action@v2`, `docker/setup-buildx-action@v2`, `docker/login-action@v2`, `docker/build-push-action@v3`, and `juliangruber/read-file-action@v1`. Some of these workflows have privileged access to secrets such as DockerHub tokens, or write access to the repository.

```
- uses: stefanzweifel/git-auto-commit-action@v5
  with:
    commit_message: 🔧 auto updating ABIs
```

Figure 21.1: An example workflow using a third-party action referenced by its tag (`elixir-protocol/.github/workflows/abis.yaml#57-59`)

Git tags are malleable. This means that while, for example, `juliangruber/read-file-action` is pinned to `v1`, the upstream may silently change the reference pointed to by `v1`. This can include malicious re-tags, which may cause Elixir's workflow to suddenly start executing malicious code.

GitHub's security hardening guidelines for third-party actions encourage developers to pin third-party actions to a full-length commit hash. Generally excluded from this are "official" actions under the actions organization; however, these other actions are not developed by GitHub.

Several Dockerfiles in the repository exhibit an analogous issue: their `FROM` lines indicate an image tag name that may change on the server side (figure 21.2). To prevent these images from silently changing, these should be pinned with a SHA256 hash.

```
FROM python:3.10.12-slim-buster
```

Figure 21.2: An example Dockerfile using a third-party image referenced by its tag ([elixir-protocol/services/feed_aggregator/Dockerfile#L1](#))

Exploit Scenario

An attacker (or compromised maintainer) silently replaces the v5 tag on the `stefanzweifel/git-auto-commit-action` repository with a malicious action that injects malicious code into the repository. When the “Generate ABIs” workflow is run, malicious code is committed and uploaded to the repository.

Recommendations

Short term, replace the current version tag references used in the workflow for third-party actions with full-length commit hashes. Similarly, pin Docker base images by their full hashes.

Long term, use Dependabot’s support for GitHub Actions to keep third-party action commit hashes up to date, complemented with maintainer reviews to ensure their safety. Incorporate static analysis tools such as Semgrep into the CI pipeline to detect issues earlier on.

22. Absence of access controls on pool creation function

Severity: Informational

Difficulty: Low

Type: Access Controls

Finding ID: TOB-ELIXIR-22

Target: protocol/contracts/src/PoolManager.sol

Description

Pools act as a form of on-chain data storage, with the `poolManager` contract responsible for maintaining the record of configuration changes over time. The `createPool` function is used to create and add new pools (records) to the contract's storage. However, this function currently lacks proper access controls, allowing any user to create and add new pools.

```
85     function createPool(  
86         address _baseToken,  
87         address _quoteToken,  
88         IPool.Exchange _exchange,  
89         bytes32 _strategy,  
90         string calldata _params  
91     ) external returns (address pool) {  
92         // Get ERC20 instance of baseToken to get decimals.  
93         ERC20 baseToken = ERC20(_baseToken);  
94         ERC20 quoteToken = ERC20(_quoteToken);  
95  
96         // Define pool parameters  
97         PoolConstants.PoolParams memory PoolParameters =  
98         PoolConstants.PoolParams({  
99             owner: msg.sender, // Owner of pool  
100             baseToken: baseToken, // Token to be paid  
101             quoteToken: quoteToken, // Token to be received  
102             exchange: _exchange, // Exchange  
103             strategy: _strategy, // Strategy  
104             params: _params // Parameters  
105         });  
106  
107         // Create pool and save its address  
108         pool = address(new Pool(abi.encode(PoolParameters)));  
109         pools.push(pool);  
110  
111         emit PoolCreated(pool, PoolParameters);  
112     }
```

Figure 22.1: *contracts/src/PoolManager.sol#L85-L111*

This could expose the protocol to potential issues:

- It allows users to submit garbage pool data to the contract.
- It creates an opportunity for recreating existing pools with potentially incorrect data or parameters.
- Legitimate pool creation transactions can be front-run with fake ones with similar data to trick the off-chain monitoring component into consuming the maliciously altered pool data.

Recommendations

Short term, consider adding the appropriate access controls to the `createPool` function.

Long term, carefully review all critical functions within the contracts to ensure correct access controls are in place, and add confirmation tests to verify the safeguards in place.

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
Configuration	The configuration of system components in accordance with best practices
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
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

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

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

- **Unused dependencies.** The following contracts are inherited but never used. Review these and re-assess whether they are required.
 - `PoolManager.ReentrancyGuard`
 - `Pool.ReentrancyGuard`
 - `ELX.ERC20VotesUpgradeable`
- **Error-prone constants.** The `FeeConstants` library defines a `DEFAULT_FEE_TIMESLICE_SECONDS` constant equal to the number of seconds per week. Solidity provides a [collection of suffixes](#) that allows one to define such constants in a way that is less error prone and easier to read. Consider replacing the value `604800` with `1 weeks`.
- **Unused storage variable and setter function.** The `minimumStakeAmount` state variable and its setter function are defined in the `Core` contract but are not used. Carefully review these and remove any dead or unused code.

```
126    /// @notice Sets the minumum stake amount to become a validator.
127    /// @param newMinimumStakeAmount Stake amount.
128    function setMinimumStakeAmount(uint256 newMinimumStakeAmount)
129        external
130        onlyRole(RoleConstants.STAKE_MANAGER_ADMIN)
131    {
132        minimumStakeAmount = newMinimumStakeAmount;
133        emit MinimumStakeAmountUpdated(newMinimumStakeAmount);
134    }
```

Figure C.1: `contracts/src/Core.sol#L126-L134`

- **Extract repeated logic into a helper function.** The `penalizeValidator` method of the `DisputeManager` contract has high cyclomatic complexity. It can be simplified by calling a helper function instead of repeating the following logic.

```
// Signer must be a validator.
bool isValidator = false;

// Check that the proof signer is a validator and not jailed.
for (uint256 j = 0; j < consensus.validatorsEnrolled.validators.length; j++) {
    if (consensus.validatorsEnrolled.validators[j] == proofSigner) {
        isValidator = true;
        break;
    }
}
```

```
}
}
```

Figure C.2: Logic from the `penalizeValidator` method that could be extracted into an internal helper method

- **Review and correct Python typing hints.** We observed several instances in the codebase where a function has an optional argument with a `None` default value; however, its type indicates it cannot be `None`. Figure C.2 shows one such example. This can confuse type checkers and IDEs and, for instance, result in incorrect dead code highlighting. Use `typing.Optional` (or the newer `X | None` syntax) to convey types correctly.

```
async def produce_message(self, message: Any, topic_name: str = None, key: str =
None, headers: Any = None):
    if topic_name is None:
        topic_name = generate_topic_name(category=self.producer_topic_name)

    if key is None:
        key = generate_message_key()
```

Figure C.3: Both `topic_name` and `key` are typed as strings; however, their default value is not a string ([elixir-protocol/shared/shared/event_bus/async_producer.py#32–37](#))

- **Prefer using `sys.exit(...)` over `exit(...)` when exiting the program.** The `exit(...)` function is a helper for the interactive shell and may not be available on all Python implementations.
- **Quote variable uses in shell scripts where splitting is undesirable.** Variable expansions must be double-quoted so as to prevent being split into multiple pieces according to whitespace or whichever separator is specified by the `IFS` variable.
- **Use `apt-get` instead of `apt`.** Several Dockerfiles invoke `apt` to perform package management. `apt` is an end-user tool and **its command line interface stability is not guaranteed**. Using `apt-get` is preferred when scripting, as it retains backwards compatibility.
- **Always perform `apt-get update` together with the `apt-get install` command.** Running the commands separately may cause issues or lead to out-of-date package installs due to layer caching.
- **Correct the use of `getattr(...)` default values when sorting validators and rewards.** If the specified property (second argument) is not found on the object (first argument), the default value (third argument) is returned instead. However, this argument is not a replacement property, but a default value. Consider replacing the instances in figures C.3 and C.4 with `validator.apy` and `reward.amount`, respectively.

```
sorted_validators = sorted(
    validators,
    key=lambda validator: getattr(validator, order_by, "apy"),
    reverse=order_dir == "desc",
)
```

Figure C.4: If `order_by` does not describe a valid property of `validator`, the string `"apy"` will be used for the sort comparisons.

([elixir-protocol-api/app/api/services/validator.py#46-50](#))

```
sorted_rewards = sorted(
    rewards,
    key=lambda reward: getattr(reward, order_by, "amount"),
    reverse=order_dir == "desc",
)
```

Figure C.5: If `order_by` does not describe a valid property of `reward`, the string `"amount"` will be used for the sort comparisons.

([elixir-protocol-api/app/api/services/rewards.py#57-61](#))

D. Automated Analysis Tool Configuration

This appendix describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, they detect certain categories of issues, such as memory leaks, misspecified format strings, and the use of unsafe APIs, with essentially perfect precision. We recommend periodically running these static analysis tools and reviewing their findings.

Slither

To install Slither, we used pip by running `python3 -m pip install slither-analyzer`.

We used Slither to detect common issues and anti-patterns in the codebase. Although Slither did not discover any severe issues during this review, some informational and code-quality issues were detected. Integrating Slither into the project's testing environment can help find other issues that may be introduced during further development and will help improve the overall quality of the smart contracts' code.

```
slither . --no-fail-pedantic --exclude='naming-convention,solc-version'
--filter-paths lib,script,test
```

Figure D.1: An example Slither configuration

Integrating `slither-action` into the project's CI pipeline can automate this process.

Bandit

To install Bandit, we used pip by running `python3 -m pip install bandit`.

After installing Bandit, we used the following command to analyze both repositories:

```
bandit -r repository-folder/
```

As this may result in many findings, it is possible to ignore some of the code by using the `-x` flag in Bandit. The following example ignores the `scripts.py` and test files, as well as the default Bandit ignore set:

```
bandit targets/ -r -x
".svn,CVS,.bzr,.hg,.git,__pycache__,.tox,.eggs,*.egg,tests,script
s.py"
```

Semgrep

To install Semgrep, we used pip by running `python3 -m pip install semgrep`.

To run Semgrep on the codebases, we ran the following command in the root directory containing the projects (running multiple predefined rules simultaneously by providing multiple `--config` arguments):

```
semgrep --config "p/trailofbits" --config "p/ci" --config "p/python"
--config "p/security-audit" --metrics=off
```

We also used `semgrep-rules-manager` to fetch and run other third-party rules.

We recommend integrating Semgrep into the project's CI/CD pipeline. To thoroughly understand the Semgrep tool, refer to the [Trail of Bits Testing Handbook](#), where we aim to streamline the use of Semgrep and improve security testing effectiveness. Also, consider doing the following:

- Limit results to error severity only by using the `--severity ERROR` flag.
- Focus first on rules with high confidence and medium- or high-impact metadata.
- Use the SARIF format (by using the `--sarif` Semgrep argument) with the [SARIF Viewer for Visual Studio Code](#) extension. This will make it easier to review the analysis results and drill down into specific issues to understand their impact and severity.

CodeQL

We installed CodeQL by following [CodeQL's installation guide](#).

After installing CodeQL, we ran the following command to create the project database for the Elixir repositories:

```
codeql database create elixir.db --language=python
```

We then ran the following command to query the database:

```
codeql database analyze elixir.db --format=sarif-latest
--output=codeql_res.sarif -- python-lgtm-full
python-security-and-quality python-security-experimental
```

For more information about CodeQL, refer to the [CodeQL chapter of the Trail of Bits Testing Handbook](#).

E. Mutation Testing

Mutation tests make changes to each line of a target contract's source code and re-run the test suite against each change. Changes that result in test failures indicate adequate test coverage, while changes that do not cause tests to fail indicate gaps in test coverage. Mutation testing allows auditors to focus their review on areas of the codebase that are most likely to contain latent bugs, and it helps developers identify and add missing tests.

Slither version 0.10.2 introduced a built-in Solidity mutation testing tool with first-class support for Foundry projects. We ran a mutation testing campaign against the target smart contracts using the following command:

```
slither-mutate ./contracts/src \  
  --test-cmd='forge test' \  
  --ignore-dirs='interfaces,constants'
```

Figure E.1: A command that runs a mutation testing campaign against all smart contracts in the protocol repository

Note that the overall runtime of this campaign is approximately 40 minutes on a consumer-grade laptop. An abbreviated, illustrative example of a mutation test output file is shown in figure E.2.

```
INFO:Slither-Mutate:Mutating contract PoolManager  
INFO:Slither-Mutate:[RR] Line 125: 'return pools' ==> 'revert()' --> UNCAUGHT  
INFO:Slither-Mutate:[CR] Line 62: '_disableInitializers()' ==>  
'//_disableInitializers()' --> UNCAUGHT  
INFO:Slither-Mutate:[CR] Line 71: '.__UUPSUpgradeable_init()' ==>  
'//__UUPSUpgradeable_init()' --> UNCAUGHT  
INFO:Slither-Mutate:[CR] Line 110: 'emit PoolCreated(pool, PoolParameters)' ==>  
'//emit PoolCreated(pool, PoolParameters)' --> UNCAUGHT  
INFO:Slither-Mutate:[CR] Line 120: 'emit PoolUpdated(address(pool), params)' ==>  
'//emit PoolUpdated(address(pool), params)' --> UNCAUGHT  
ERROR:Slither-Mutate:list index out of range  
INFO:Slither-Mutate:Done mutating PoolManager.  
INFO:Slither-Mutate:Revert mutants: 1 uncaught of 9 (11.11111111111111%)  
INFO:Slither-Mutate:Comment mutants: 4 uncaught of 8 (50.0%)  
INFO:Slither-Mutate:Zero Tweak mutants analyzed
```

Figure E.2: Abbreviated output from the mutation testing campaign on PoolManager.sol

In summary, the following features of the PoolManager contract lack sufficient tests:

- The pools getter method is not executed by any tests, as indicated by the uncaught revert mutant on line 125.

- No tests validate the initialization logic of implementation and proxy contracts. When important deployment steps are commented out on lines 62 and 71, the tests still pass.
- Event emissions, such as those on lines 110 and 120, are not validated by tests.

We recommend that the Elixir team review the existing tests and add additional verification that would catch the aforementioned types of mutations. Then, use a script similar to that provided in figure E.1 to re-run a mutation testing campaign to ensure that the added tests provide adequate coverage.

F. Incident Response Recommendations

This section provides recommendations on formulating an incident response plan.

- **Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.).**
- **Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug.**
 - Consider documenting a plan of action for handling failed remediations.
- **Clearly describe the intended contract deployment process.**
- **Outline the circumstances under which Elixir Technologies Ltd will compensate users affected by an issue (if any).**
 - Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw.
- **Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on.**
 - Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific system components.
- **Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers) and how it will onboard them.**
 - Effective remediation of certain issues may require collaboration with external parties.
- **Define contract behavior that would be considered abnormal by off-chain monitoring solutions.**

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop “muscle memory.” Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run.

G. Security Best Practices for Using Multisignature Wallets

Consensus requirements for sensitive actions, such as upgrading contracts, are meant to mitigate the risks of the following:

- Any one person overruling the judgment of others
- Failures caused by any one person's mistake
- Failures caused by the compromise of any one person's credentials

For example, in a 2-of-3 multisignature wallet, the authority to upgrade the StakeManager contract would require a consensus of two individuals in possession of two of the wallet's three private keys. For this model to be useful, the following conditions are required:

1. The private keys must be stored or held separately, and access to each one must be limited to a unique individual.
2. If the keys are physically held by third-party custodians (e.g., a bank), multiple keys should not be stored with the same custodian. (Doing so would violate requirement #1.)
3. The person asked to provide the second and final signature on a transaction (i.e., the cosigner) should refer to a pre-established policy specifying the conditions for approving the transaction by signing it with his or her key.
4. The cosigner should also verify that the half-signed transaction was generated willingly by the intended holder of the first signature's key.

Requirement #3 prevents the cosigner from becoming merely a "deputy" acting on behalf of the first signer (forfeiting the decision-making responsibility to the first signer and defeating the security model). If the cosigner can refuse to approve the transaction for any reason, the due-diligence conditions for approval may be unclear. That is why a policy for validating transactions is needed. A verification policy could include the following:

- A protocol for handling a request to cosign a transaction (e.g., a half-signed transaction will be accepted only via an approved channel)
- An allowlist of specific addresses allowed to be the payee of a transaction
- A limit on the amount of funds spent in a single transaction or in a single day

Requirement #4 mitigates the risks associated with a single stolen key. For example, say that an attacker somehow acquired the unlocked Ledger Nano S of one of the signatories. A voice call from the cosigner to the initiating signatory to confirm the transaction would reveal that the key had been stolen and that the transaction should not be cosigned. If the signatory were under an active threat of violence, he or she could use a **duress code** (a code word, a phrase, or another signal agreed upon in advance) to covertly alert the others that the transaction had not been initiated willingly, without alerting the attacker.

H. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On September 6, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Elixir Technologies Ltd. team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, all of the 22 issues described in this report were resolved by Elixir Technologies Ltd. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Use of outdated dependencies	Resolved
2	Use of HTTP requests without timeout	Resolved
3	Private keys stored in environment variables	Resolved
4	Majority proposal group will always satisfy minimum size requirement	Resolved
5	Authentication at risk of replay attacks if misconfigured	Resolved
6	Strategy executor does not validate payload to sign	Resolved
7	Missing IP address validation allows bypassing Redpanda ACL	Resolved
8	Use of assert statement in production code	Resolved
9	Redpanda accounts and associated permissions are never revoked	Resolved
10	Delegators can redelegate stakes to jailed delegatee	Resolved
11	Attackers can cause slashing to become economically infeasible	Resolved

12	Delegators can immediately undelegate before their delegatee is jailed	Resolved
13	Minority validators may participate in consensus process	Resolved
14	An influx of new strategy executors may halt consensus	Resolved
15	Lack of two-step process for ownership transfers	Resolved
16	Response payload to authentication challenge is not signed	Resolved
17	API does not validate display_name and app_version	Resolved
18	All on-chain events replayed upon startup	Resolved
19	Redpanda exposed to the Internet	Resolved
20	API has admin access to Redpanda	Resolved
21	Use of unpinned third-party Docker images and actions on workflows	Resolved
22	Absence of access controls on pool creation function	Resolved

Detailed Fix Review Results

TOB-ELIXIR-1: Use of outdated dependencies

Resolved in [PR 465](#). The Python dependencies were updated.

TOB-ELIXIR-2: Use of HTTP requests without timeout

Resolved in commit [6c0031d](#). A 45-second timeout was added to the requests.

TOB-ELIXIR-3: Private keys stored in environment variables

Resolved in [PR 572](#), [PR 478](#), and [PR 549](#). The services now issue sign requests to Google Cloud Key Management Service where the private key material is stored. The validators are provided with multiple options for key material storage, including the managed cloud solutions. The less-safe methods are still available, although they are discouraged because the Elixir Protocol team has no control over the validators runtime environment. We recommend documenting all the key management options so the users are aware of the risks.

TOB-ELIXIR-4: Majority proposal group will always satisfy minimum size requirement

Resolved in commit [a949bc1](#). The calculation of `minimum_requirement` was fixed and now is a fraction of the `total_proposals`.

TOB-ELIXIR-5: Authentication at risk of replay attacks if misconfigured

Resolved in commit [481a55a](#). The code was refactored, and the Redis service is injected as a dependency. The nonce is no longer empty when service is misconfigured.

TOB-ELIXIR-6: Strategy executor does not validate payload to sign

Resolved in [PR 474](#). The message structure validation was added to the strategy executor code.

TOB-ELIXIR-7: Missing IP address validation allows bypassing Redpanda ACL

Resolved in commits [0dbd7d6](#) and [38db652](#). The architecture has changed, and IP addresses are no longer collected from validators, removing the need for validation.

TOB-ELIXIR-8: Use of assert statement in production code

Resolved in [PR 487](#). The assert statements were replaced with explicit raise statements under violating conditions.

TOB-ELIXIR-9: Redpanda accounts and associated permissions are never revoked

Resolved in [PR 42](#). The architecture has changed, and API no longer configures the Redpanda ACLs. The access revocation mechanism is no longer needed.

TOB-ELIXIR-10: Delegators can redelegate stakes to jailed delegatee

Resolved in commit [a0ae84e](#). An additional check has been added to the `_updateDelegation` method that prevents a user from delegating to an address that has been jailed.

TOB-ELIXIR-11: Attackers can cause slashing to become economically infeasible

Resolved in commit [fd18709](#). The slash method of the Core contract now accepts a list of accounts and list of amounts, respectively, to slash, with appropriate verification of list lengths. This decreases the gas overhead from slashing a large number of accounts.

We recommend further analysis of gas usage. Review the short-term recommendation for this issue and consider whether it would be acceptable to slash the balance of accounts down to zero if and only if accounts are flagged as misbehaving in the specific way described by this issue. Doing so would significantly lower the gas usage of slashing many accounts in such scenarios.

TOB-ELIXIR-12: Delegators can immediately undelegate before their delegatee is jailed

Resolved in commit [af5634d](#). Event listeners in the `onchain_stake_manager_feed` service now respond to `SignalRedelegate` and `CancelSignalRedelegate` events. When a `SignalRedelegate` event is received, the current delegated balance is subtracted from the existing delegated balance, and this delegator is removed from the list of staker addresses (only if this delegator has not already been removed). This causes delegators that have signaled an intent to redelegate to be excluded from a validator's total delegations. Additionally, when a `CancelSignalRedelegate` event is received, this action is undone by adding the delegated balance back to the old delegatee and adding their address back to the list of staker addresses.

Critically, the StakeManager contract was updated in [PR 689](#) to allow signal cancellations only when a signal has been registered but the signaled action has not yet been completed. This will prevent a malicious user from repeatedly canceling signals to repeatedly increase their delegated balance.

TOB-ELIXIR-13: Minority validators may participate in consensus process

Resolved in [PR 497](#) and [PR 42](#). The validator system has been rearchitected, and it is no longer possible to submit a proposal without being in the core validator set.

TOB-ELIXIR-14: An influx of new strategy executors may halt consensus

Resolved in [PR 497](#) and [PR 42](#). The constraint is now a fraction of the core validator set size, which solves the issue with consensus halting by an influx of malicious validators.

TOB-ELIXIR-15: Lack of two-step process for ownership transfers

Resolved in commit [543b3e0](#). The `OwnableUpgradeable` dependencies have been replaced with `Ownable2StepUpgradeable` contracts that properly implement a two-step process for ownership transfers.

TOB-ELIXIR-16: Response payload to authentication challenge is not signed

Resolved in [PR 56](#). The request structure was refactored into `ValidatorAuthorizationRequest`, which contains a proof and detail. The detail structure is encoded with Ethereum ABI for signature verification.

TOB-ELIXIR-17: API does not validate `display_name` and `app_version`

Resolved in [PR 42](#). The values are now validated in the application code as well as in the database.

TOB-ELIXIR-18: All on-chain events replayed upon startup

Resolved in [PR 498](#). The services now resume event consumption from the last processed event instead of replaying the history from the beginning on startup.

TOB-ELIXIR-19: Redpanda exposed to the Internet

Resolved in [PR 42](#). The back end was rearchitected in such a way that Redpanda is split into two clusters, one on the internal network and one that is internet-facing. The internet-facing cluster is read-only, and validators do not write any data to it. The order proposals are now sent to the API. Although this fix was not part of our original recommendation and Redpanda is still exposed, the new architecture minimizes the possible damage to the point we consider it resolved.

TOB-ELIXIR-20: API has admin access to Redpanda

Resolved in [PR 42](#) and commit [8de6fdd](#). The architecture has changed, and the API no longer configures the Redpanda ACLs. The admin access and related code was removed from the service.

TOB-ELIXIR-21: Use of unpinned third-party Docker images and actions on workflows

Resolved in commits [c4360eb](#) and [f34b570](#). Third-party Docker images and actions were pinned.

TOB-ELIXIR-22: Absence of access controls on pool creation function

Resolved in commit [0d15c2f](#). An `onlyOwner` function modifier has been added to the `createPool` method of the `PoolManager` contract.

I. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	Description
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.