OMNISCIA

March 22, 2024

SMART CONTRACT AUDIT REPORT

Sygnum Bank
SyGuard


omniscia.io


info@omniscia.io


Online report: sygnum-bank-syguard

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.

omniscia.io


info@omniscia.io

SyGuard Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
8e77213a5f	February 23rd 2024	77dd10bdc4
c09041282f	March 20th 2024	f17307561d
e99879a2e9	March 22nd 2024	bf063162e2

Audit Overview

We were tasked with performing an audit of the Sygnum Bank codebase and in particular their SyGuard module.

Over the course of the audit, we did not identify any security vulnerabilities relating to the code and its implementation.

The SyGuard implementation has been created as a Safe module and specifically a Gnosis Guild module that can be attached to a multi-signature Safe wallet, permitting a Sygnum-affiliated member to replace one of the owners of the wallet in case of ownership loss.

The system is meant to be utilized with the `Delay` modifier of the Gnosis Guild to ensure that ownership adjustments are not instant, increasing the resilience of the overall system and minimizing the Single-Point-of-Failure effect of the SyGuard.

As part of our review, we evaluated and ensured that the security of the `Delay` modifier is sound. We did not identify any security vulnerabilities and solely made notices of optimizations as well as style changes that can be performed but can also be safely ignored by the Sygnum team.

From a design perspective, **there are a few notes we would like to make in relation to how the SyGuard module is utilized**. Specifically, the `Delay` modifier that is meant to be used in tandem is not purpose-specific and the security of the overall SyGuard system can be enhanced by using a custom `Modifier` implementation rather than the Gnosis Guild one.

In detail, we would advise the following changes to the `Delay` modifier presently in use from a design perspective:

All findings in the audit report can be safely acknowledged, however, we strongly advise the Sygnum team to evaluate our analysis above and introduce a custom-knit implementation of a `Modifier` that is better suited and more secure at processing owner adjustment requests by the Sygnum team.

Post-Audit Conclusion

The Sygnum Bank team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Sygnum Bank and have identified that all exhibits within the audit report that concern the `SwapOwnerModule` have been resolved.

After discussions with the Sygnum Bank team, we concluded that certain findings concerned an external dependency of the project maintained by the Safe (a.k.a. Gnosis) team.

As a result, the following findings have been marked as acknowledged given that they have been forwarded by the Sygnum Bank team to the contract's maintainers and bear no effect on the functionality of the `SwapOwnerModule`: **`DYA-01S`**, **`DYA-02C`**, **`DYA-01C`**

In relation to the higher-level recommendations we outlined in the previous chapter, the Sygnum Bank team wished to apply our first and second bullet points.

While logic to accommodate for these has been introduced to the **`SwapOwnerModule::startRecovery`** function, we do not consider it sufficient.

Specifically, the uppermost `if` conditional should validate whether the `lastTxNonce` is less-than-or-equal-to the `Delay` implementation's nonce, indicating a past transaction.

The above would ensure our first bullet point is adhered to, however, the second bullet point the Sygnum Bank team wished to implement has not been done so.

To note, the Sygnum Bank specified the `OwnerManager` implementation of a Safe already takes care of this check which is incorrect.

The scenario we outlined is a recovery being queued swapping owner A to owner B multiple times.

If this occurs and owner B is once again replaced by owner A, for example because the private key was recovered, any transaction that was submitted beyond the first can be re-activated to cause the owner to transition again incorrectly.

We advise the Sygnum Bank team to revisit the code and implement the aforementioned change as well as re-evaluate the second bullet point they wish to integrate in the code.

Post-Audit Conclusion (e99879a2e9)

The Sygnum Bank team evaluated our feedback outlined above and maintained that our assessment was incorrect, providing supplemental material to justify this statement.

In light of this, we revisited the `Delay` implementation code and concluded that we mistook the `txNonce` exposed by the `Delay` implementation as the `queueNonce` which represents the nonce of the latest queued transaction.

As such, **our original statements in the previous chapter are invalid** and the code adaptations made by the Sygnum Bank team have properly alleviated the first and second bullet points we had shared.

To note, an edge case within the revised codebase was identified by the Sygnum Bank team in which the first recovery operation by the `SwapOwnerModule` may have failed.

We evaluated the edge case and proposed a slight refactor of the contract's security mechanism to ensure it is optimally evaluated and does not result in the aforementioned edge case.

The Sygnum Bank team proceeded to apply the adjustments we requested which we evaluated as behaving to their specification.

Based on the aforementioned actions, we consider the first and second bullet points securely upheld by the latest iteration of the code.

During this audit round, we evaluated the systematic risk associated with relying on the `IDelay` contract's operation for the recovery functionality exposed by the `SwapOwnerModule`.

A scenario was identified in which **a recovery operation may become impossible due to external factors** that we advise the Sygnum Bank team to consider and potentially disallow on-chain.

In detail, an `IDelay` contract configured with a zero-value `txExpiration` **may cause recoveries to become impossible** if the `IDelay` queue contains an inexecutable transaction.

Normally, the **`Delay::setTxNonce`** function can be invoked by the multi-signature wallet to skip transactions that cannot be executed.

As transaction skipping is solely executable by the multi-signature wallet itself, a transaction that always fails will not be possible to skip and thus the recovery transaction that is queued after it would never become executable.

Given that we consider the `SwapOwnerModule` contract to be an emergency mechanism that should be accessible when the multi-signature wallet is unable to vote, we advise the Sygnum Bank team to ensure that the [illegible] of the [illegible] is non-zero.

that the `txExpiration` of the `target` is non-zero.

We would like to clarify that this sanitization would be a precaution as the `Delay` implementation could be interacted beyond the configuration of the `SwapOwnerModule` and thus have its `txExpiration` adjusted.

To this end, users of the `SwapOwnerModule` will have to be adequately instructed to never change the `txExpiration` of the `IDelay` contract to avoid breaking the recovery functionality of Sygnum Bank's module.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	5	2	0	3
Minor	0	0	0	0
Medium	0	0	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **3 findings during the manual review** of the codebase.

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/sygnumbank/syguard-contracts**
- Commit: 8e77213a5f98e45177536e4d1c0bb104c81a3eaf
- Language: Solidity
- Network: Ethereum
- Revisions: **8e77213a5f**, **c09041282f**, **e99879a2e9**

Contracts Assessed

File	Total Finding(s)
contracts/mock/Delay.sol (DYA)	3
contracts/SwapOwnerModule.sol (SOM)	2

Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in TypeScript.

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.20` based on the version specified within the `hardhat.config.ts` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.18`).

We advise them to be locked to `0.8.20` (`=0.8.20`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **13 potential issues** within the codebase of which **8 were ruled out to be false positives** or negligible findings.

The remaining **5 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
DYA-01S	Informational	Acknowledged	Suboptimal Event Declarations
SOM-01S	Informational	Yes	Multiple Top-Level Declarations

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Sygnum Bank.

As the project at hand implements a Gnosis Guild recovery module, intricate care was put into ensuring that the **access flow within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **did not pinpoint any non-informational issues within the codebase**, however, **we strongly advise the Sygnum team to evaluate our recommendation** in relation to a new `Modifier` implementation that is purpose-built for the SyGuard.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be.

A total of **3 findings** were identified over the course of the manual review of which **1 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
SOM-01M	Informational	Yes	Improper Initialization of Ownership

Code Style

During the manual portion of the audit, we identified **2 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
DYA-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
DYA-02C	Informational	Acknowledged	Redundant Parenthesis Statements

Delay Static Analysis Findings

DYA-01S: Suboptimal Event Declarations

Type	Severity	Location
Gas Optimization	Informational	Delay.sol:L13, L14, L15

Description:

The referenced `event` declarations do not have any `indexed` argument or have less than three `indexed` arguments that are a primitive type.

Example:

contracts/mock/Delay.sol

SOL

```
13  event TxCooldownSet(uint256 cooldown);
```

Recommendation:

Apart from aiding off-chain integrators in consuming and filtering such events, primitive types that are set as `indexed` **will result in a gas optimization** due to reduced memory costs. As such, we advise the `indexed` keyword to be introduced to up to three different primitive types in total optimizing the referenced `event` declarations.

Alleviation (c09041282f878e1f510fb00af59798b25cb5f824):

The Sygnum Bank team evaluated this exhibit and specified that this particular contract is an external dependency of the project managed by the Safe (a.k.a. Gnosis) team and the findings that pertain to it will be forwarded to their team.

As a result, we consider these exhibits acknowledged as they have no bearing to the code of the Sygnum Bank team.

SwapOwnerModule Static Analysis Findings

SOM-01S: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	SwapOwnerModule.sol:L6, L14

Description:

The referenced file contains multiple top-level declarations that decrease the legibility of the codebase.

Example:

contracts/SwapOwnerModule.sol

SOL

```sol
6   interface IOwnerManager {
7       function swapOwner(
8           address prevOwner,
9           address oldOwner,
10          address newOwner
11      ) external;
12  }
13
14  contract SwapOwnerModule is Module {
```

Recommendation:

We advise all highlighted top-level declarations to be split into their respective code files, avoiding unnecessary imports as well as increasing the legibility of the codebase.

Alleviation (c09041282f878e1f510fb00af59798b25cb5f824):

The `IOwnerManager` interface has been relocated to its dedicated `IOwnerManager` file, being imported in the `SwapOwnerModule` file correctly and thus optimizing the project's structure as advised.

SwapOwnerModule Manual Review Findings

SOM-01M: Improper Initialization of Ownership

Type	Severity	Location
Logical Fault	Informational	SwapOwnerModule.sol:L35

Description:

The `SwapOwnerModule::setUp` function will initialize the `OwnableUpgradeable` dependency to the deployer (`msg.sender`) instead of the `_owner` temporarily.

Impact:

A severity of `informational` has been assigned given that the role ultimately ends up to the correct address (`_owner`) at the end of the transaction and only the `event` trail as well as gas costs are affected.

Example:

contracts/SwapOwnerModule.sol

SOL

```sol
/// @notice Public setup function to allow deployment via factory / proxy pattern
/// @param initializeParams ABI encoded parameters (see constructor)
function setUp(bytes memory initializeParams) public override initializer {
    __Ownable_init(msg.sender);
    (address _target, address _avatar, address _owner) = abi.decode(
        initializeParams,
        (address, address, address)
    );
    require(_avatar != address(0), "Avatar can not be zero address");
    require(_target != address(0), "Target can not be zero address");
```

Example (Cont.):

```sol
        require(_owner != address(0), "Owner can not be zero address");
        avatar = _avatar;
        target = _target;
        _transferOwnership(_owner);

        emit SwapOwnerSetup(msg.sender, _target, _avatar, _owner);
        emit AvatarSet(address(0), _avatar);
        emit TargetSet(address(0), _target);
    }
```

Recommendation:

We advise the dependency to be initialized with the `_owner` argument directly, optimizing the set up cost of the module as well as ensuring that the `event` on-chain trail is proper.

Alleviation (c09041282f878e1f510fb00af59798b25cb5f824):

The `msg.sender` initialization has been removed as advised, initializing the `OwnableUpgradeable` dependency directly to the `_owner` instead.

Delay Code Style Findings

DYA-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	Delay.sol:L138, L167, L199, L210

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

contracts/mock/Delay.sol

SOL

```
167 queueNonce++;
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation (c09041282f878e1f510fb00af59798b25cb5f824):

The Sygnum Bank team evaluated this exhibit and specified that this particular contract is an external dependency of the project managed by the Safe (a.k.a. Gnosis) team and the findings that pertain to it will be forwarded to their team.

As a result, we consider these exhibits acknowledged as they have no bearing to the code of the Sygnum Bank team.

DYA-02C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	Delay.sol:L224, L228

Description:

The referenced statements are redundantly wrapped in parenthesis' (`()`).

Example:

contracts/mock/Delay.sol

SOL

```
224 return (txHash[_nonce]);
```

Recommendation:

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation (c09041282f878e1f510fb00af59798b25cb5f824):

The Sygnum Bank team evaluated this exhibit and specified that this particular contract is an external dependency of the project managed by the Safe (a.k.a. Gnosis) team and the findings that pertain to it will be forwarded to their team.

As a result, we consider these exhibits acknowledged as they have no bearing to the code of the Sygnum Bank team.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.