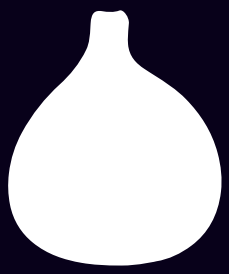


HINKAL

HINKAL-PROTOCOL SECURITY REVIEW

zokyo

February 20th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this protocol passed a security audit.

ZOKYO AUDIT SCORING HINKAL

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 acknowledged = -4 points deducted
- 8 Low issues: 8 resolved = 0 points deducted
- 1 Informational issue: 1 resolved = 0 points deducted

Thus, 100 - 4 = 96

TECHNICAL SUMMARY

Hinkal enlisted the services of Zokyo to conduct a security assessment on their Hinkal protocol repository from December 12th to December 27th, 2023. Hinkal, a TypeScript codebase, facilitates staking and yield farming on platforms like Curve, Convex, and Beefy. Users begin by undergoing a Know Your Customer (KYC) procedure to acquire an "Access token," allowing for the creation of shielded addresses for asset deposits.

Hinkal stands out for its ability to provide a private platform for executing trading strategies, enabling users to discreetly manage and grow their assets. This functionality extends to private transfers, withdrawals to new or existing wallets, and the capacity to obscure the connection between public wallets, enhancing user privacy.

Through Hinkal, users can accumulate assets on private addresses, earn yields on idle assets without on-chain visibility, and engage in private trading, staking, and yield farming activities. Assets return to private addresses, with any on-chain appearance masked by random relayer addresses.

The security assessment focused on the GitHub repository of Hinkal, evaluating security risks and implications related to changes introduced by the development team before its production release, which followed shortly after the assessment deadline. Due to time constraints, only essential aspects of the application were tested and verified.

Despite these limitations, the outcome of the security audit is positive, highlighting the adherence to best practices for secure Hinkal development.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Hinkal repository and Server/ relayer/dapps endpoints:
Repo: https://github.com/Hinkal-Protocol/Hinkal-Protocol/tree

Last commit - 258716071b5a2f939c46af9e95c2e8904378c24a

The team at Zokyo was provided nearly two week for the engagement and assigned a full time security engineer to audit the security of the Hinkal code. The security engineer is a blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.
The purpose of this audit is to achieve the following:

- Correctness of the implementation: reviewed Hinkal's code to ensure that it is implemented correctly, adheres to best practices, and follows industry-standard coding practices.
- Authentication mechanisms: evaluated Hinkal's authentication mechanisms to ensure that they are secure and not susceptible to brute force attacks.
- Security of private keys: reviewed Hinkal's handling of private keys to ensure that they are encrypted, stored securely, and not exposed to unauthorized parties.
- Mismanagement of funds via transactions: We will review Hinkal's handling of transactions to ensure that only authorized transactions are executed and that rate-limiters are in place to prevent misuse.

- Vulnerabilities in the code: We will perform a rigorous analysis of the Hinkal's code to identify and address any vulnerabilities that could be exploited by attackers.
- Static Analysis using automated tools (Dependency-Check, eslint, LGTM etc)
- Dynamic Analysis Hinkal functions and data types
- Secure interaction between components: We will evaluate Hinkal's interaction with other components, such as Web3 components, to ensure that it is secure and free from vulnerabilities.
- Data privacy and information integrity: We will review Hinkal's handling of user data to ensure that it is encrypted, stored securely, and not exposed to unauthorized parties.

In summary, Zokyo identified a few security risks and recommends performing further testing to validated extended safety and correctness in context to the whole structure and no critical vulnerability were found.

Executive Summary

No critical issues were identified during the review. The review identified one medium-severity finding, several low-severity findings and one informational issue. Almost all issues have been resolved, detailed explanations of these findings can be found in the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Hinkal team and the Hinkal team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

- **Critical**

  The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

- **High**

  The issue affects the ability of the contract to compile or operate in a significant way.

- **Medium**

  The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

- **Low**

  The issue has minimal impact on the contract’s ability to operate.

- **Informational**

  The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Absence of Rate Limiting on Critical Endpoints(All servers URLs)	Medium	Acknowledged
2	Insufficient Signature Verification	Low	Resolved
3	Potential Exposure to Enumeration Attacks	Low	Resolved
4	Lack of Input Validation in react/src/hooks/useDeposit	Low	Resolved
5	Insufficient Validation Checks in react/src/hooks/useSwap	Low	Resolved
6	Path Disclosure in Error Response(All Relayers/Servers URLs)	Low	Resolved
7	Server Version Disclosure	Low	Resolved
8	Inconsistency in SSL/TLS Configuration Between Domain and ELB Endpoint(All Servers and Relayers)	Low	Resolved
9	Components With Known Vulnerability	Low	Resolved
10	Remove/Resolve TODO comments	Informational	Resolved

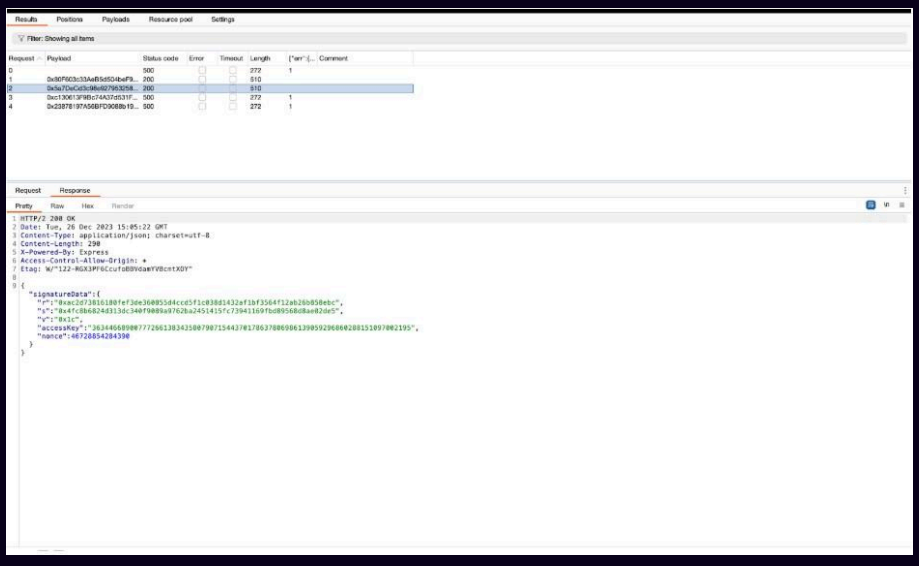
Absence of Rate Limiting on Critical Endpoints(All servers URLs)

Description:

Several endpoints in the Hinkal Protocol, particularly those involved in sensitive operations such as the https://polygon.server.hinkal.pro/users/0xC6Cd9d90F1323308B49891175572B78cdd83eCFF/signature/363446689007772661383435807907154437017863780698613905929686028815109700219 5/kyc/ function, do not implement rate limiting. This absence of rate limiting poses a significant risk as it allows attackers to send a high volume of requests in a short period. Such unrestricted access can lead to various issues, including resource exhaustion, degraded performance, and potentially, Denial-of-Service (DoS) attacks. Overloading the server with requests could disrupt the service for legitimate users and might also expose the server to further vulnerabilities as it struggles to handle the load.

Proof of Concept/Scenario:

An attacker can exploit this vulnerability by repeatedly calling the endpoint with different Ethereum addresses or even the same address. This could be automated using a script that rapidly sends requests, aiming to overwhelm the server. Such an attack could degrade the performance of the server, leading to slow response times or complete denial of service for **other users.**



Recommendation:

Implement Rate Limiting: Introduce rate limiting on all sensitive endpoints, including /users. The rate limits should be configured based on typical user behavior and the server's capacity.

Comment from Client : #1 We are aware of this issue, however some users have issues using the app if we enable rate-limiting.
Thus, we have chosen not to enable it and instead, we closely monitor the traffic on the server to identify if any kind of DOS attack is taking place.

Insufficient Signature Verification

Description: In hexagate-callback.ts the endpoint conditionally checks the `IS_HEXAGATE_SIGNATURE_DISABLED` flag to determine whether to verify the `x-hexagate-signature` header. If this flag is disabled, the endpoint processes requests without validating their authenticity, leading to a security risk.

Code Snippet:

```
if (!IS_HEXAGATE_SIGNATURE_DISABLED) {
    const hexagateSignature = req.headers['x-hexagate-signature'];
    // ... signature verification logic ...
}
```

POC/Scenario: An attacker could exploit this by sending a spoofed request when the signature verification is disabled, possibly leading to unauthorized actions

Recommendation:
Always enforce signature verification for incoming requests to ensure data integrity and source authenticity. Remove the conditional check and consistently validate the signature.

Client comment : The check in question was only being used for local testing and not production environment.However we understand the pontential issue that it can cause.Thus we have resolved the issue by enforcing that the signature verification takes place at all times.

Potential Exposure to Enumeration Attacks

Description:
In user-get-signature.ts throwing a `UserDoesNotExistError` when a user is not found can lead to user enumeration attacks, where an attacker can determine if a user exists in the system.

Code Snippet:

```
if (!user) throw new UserDoesNotExistError();
```

POC/Scenario:
An attacker could systematically try different Ethereum addresses to infer which ones are registered in the system.

Recommendation:
Return a generic message that does not indicate whether the user exists or not, to avoid giving away information about the user's address.

Client comment : We have resolved this issue.
Now, instead of throwing an error, we just return `False` as a response indicating that KYC/KYB verification is not complete for the user in question.

Lack of Input Validation in react/src/hooks/useDeposit

Description:
The function only checks the first element of the `amountChanges` array for validation. It does not validate the entire array or the `erc20Addresses` array. This might lead to issues if the arrays contain invalid or inconsistent data.

Scenario/POC:
An attacker could potentially pass an array with the first element as a valid amount and subsequent elements as invalid or malicious data, which might not be properly handled by the system.

Recommendation:
Perform thorough validation on all elements of the `erc20Addresses` and `amountChanges` arrays. Ensure that all addresses are valid and that all amounts are appropriate for the transaction.

Client comment :
We only check the first element of the array as the array should only have one element incase of deposits.
However, we have introduced additional check to ensure that the array only has 1 element.

Insufficient Validation Checks in react/src/hooks/useSwap

Code Snippet:

```
const swap = useCallback(
  async (erc20Addresses: string[], amountChanges: bigint[], externalActionId:
ExternalActionId, data: string) => {
    ...
    if (amountChanges[0] === 0n) {
      throw new Error(transactionErrorCodes.AMOUNT_EMPTY);
    }
    // TODO: Add other validators
    ...
  },
  [hinkal, shieldedAddress, setTransactionStatus, requireAccessToken, runTransaction,
onSuccess, onError],
);
```

Scenario / Proof of Concept (POC):
The current validation in the `swap` function checks if the first element of `amountChanges` is zero but does not validate other elements in the array. There's a potential risk that invalid or malicious data could be processed, especially when the array contains more than one element. The absence of comprehensive validation could lead to unintended behavior, incorrect transaction processing, or vulnerabilities to attack vectors like denial of service or unexpected contract behavior.

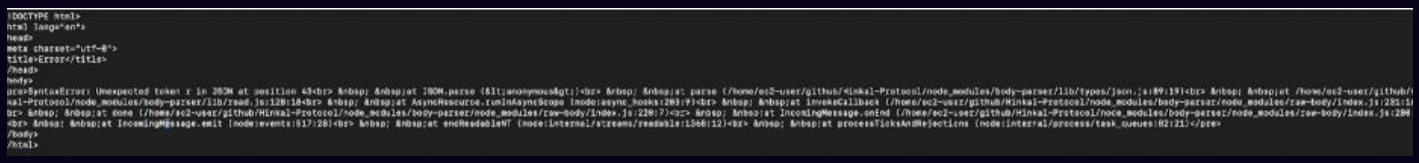
Recommendation:
Implement comprehensive validation for all elements in the `amountChanges` array. This includes checking if all elements are greater than zero and whether they meet other business logic criteria (e.g., maximum allowed amount, alignment with token decimals, etc.).

Client Comment:
We only check the first element of the array as the array should only have one element incase of deposits.
However, we have introduced additional check to ensure that the array only has 1 element.

Path Disclosure in Error Response(All Relayers/Servers URLs)

Description:

The Hinkal Protocol's Relayer endpoint is revealing sensitive file path information in its error response when it encounters malformed JSON input. In the provided scenario, a malformed POST request to `All relayers URLS /general-transact` leads to an error response that includes detailed server path information such as `/home/ec2-user/github/Hinkal-Protocol/node_modules/....` This information can be exploited by an attacker to gain insights about the server's directory structure, software stack, and potentially other system details, which could facilitate further attacks, such as directory traversal or targeted exploits against known vulnerabilities in the server's tech stack.



Recommendation:

Modify the error handling mechanism to avoid sending detailed internal error information to the client. Instead, log these details on the server for debugging purposes and return generic error messages to the client.

Implement custom error pages that provide minimal information to the end-user, ensuring that no sensitive information is leaked

Client Comment :
We have resolved this issue.
We have updated code to just pass a generic error message to the client.
So that no other information is leaked.

Server Version Disclosure

Description

If you are running a web server, it often shows the world what type of server it is, its version number, and the operating system. This information is available in header fields and can be acquired using a web browser to make a simple HTTP request to any web application. It is often called the webserver banner and is ignored by most people with the exception of malicious ones.

Attackers can perform banner grabbing using even simple TCP tools like telnet or netcat. Then they launch targeted attacks against your web server and version. In addition, if a particular web server version is known to be vulnerable to a specific exploit, the attacker would just need to use that exploit as part of their assault on the target web server.

Proof of Concept

The following is needed in order to reproduce this issue:
The following call shows the application disclosing the server version in the response.

The application discloses the **NGINX** server version which allows an attacker to gain more information on the system.

Recommendation:

- Obscuring web server information in headers.
- Using a hardened reverse proxy server to create an additional layer of security between the web server and the internet.
- Ensuring that web servers are kept up-to-date with the latest software and security patches.

Client comment :
We have resolved this issue by making changes to server configs to hide the version of nginx.

Inconsistency in SSL/TLS Configuration Between Domain and ELB Endpoint(All Servers and Relayers)

Description

An inconsistency has been identified in the SSL/TLS configuration between the domain and its underlying AWS Elastic Load Balancer (ELB) endpoint. While the domain is configured to establish a secure HTTPS connection, direct access to the ELB endpoint results in a non-secure connection. This discrepancy creates a vulnerability where sensitive data could be exposed during transmission to and from the ELB endpoint, potentially leading to a man-in-the-middle (MITM) attack.

Recommendation:

1. SSL/TLS Configuration for ELB:
    - Ensure SSL/TLS certificates are installed and correctly configured on the ELB. This ensures encryption of all data in transit.
2. Enforce HTTPS:
    - Implement a policy to redirect all HTTP requests to HTTPS, ensuring that no unencrypted communication occurs.
3. HTTP Strict Transport Security (HSTS):
    - Employ HSTS to force clients to use HTTPS connections only, preventing any attempts to connect via HTTP.

Client Comment :

We have removed all Http port from ELB instances. Therefore now only secured https connection can take place. Additionally only calls via domain are accessible and direct calls to ELB will not be possible anymore.

Components With Known Vulnerability

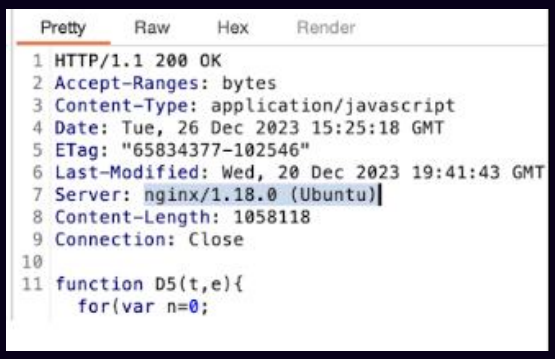
Description

The service depends upon several third-party software packages to operate. Any of these components may, over time, have security vulnerabilities that are publicly disclosed and fixed in later versions. It is a common flaw for software development to focus on hardening the code written in-house but neglect to properly manage the dependent third-party software packages used. Hence there should be a process for patch management and all the software should be reviewed and updated to patch for security issues.

The application uses an outdated NGINX 1.18.0 server version. which contains known vulnerabilities. Depending on the vulnerability exploited there could be a potential loss of data.

Proof of Concept

The following is needed in order to reproduce this issue:



Recommendation:

- As a best practice, keep all software up to date, especially if there exists a known vulnerability or weakness associated with an older version.
- None of these attacks was found possible during the test, but this might change in the future if the application is found using the old component with insecure user-supplied input.

Client comment :

We have resolved this issue and updated the server for nginx 1.20.

Remove/Resolve TODO comments

Description:
The comment in react/src/hooks/useSwap.tsx
`// TODO: Add other validators` indicates a known gap in the current implementation. It's crucial to address these TODOs, especially where security and accuracy are paramount. Across the Hinkal repository, in many typescripts files , there are several TODO comments which show unresolved work and are advised to remove or resolve before deployment.

Recommendation:
Resolve or remove the TODO statements.

Client comment :
The TODO: statements were old and irrelevant. I have removed all the TODO statements from codebase so that it is not confusing.

We are grateful for the opportunity to work with the Hinkal team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Hinkal team put in place a bug bounty program to encourage further analysis of the protocol by third parties.

zokyo