CFG NINJA

SECURITY ASSESSMENT

Jack Eth bundle Script

November 3, 2024

Audit Status: Fail

BLADE POOL

CFG NINJA

CFG Ninja Verified on November 3, 2024

Jack Eth bundle

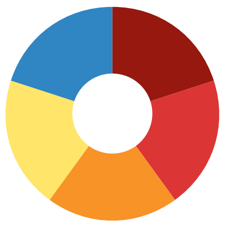
Executive Summary

TYPES	ECOSYSTEM	LANGUAGE
DeFi	ETHEREUM	Ganache

Timeline

Audit Request	Onboarding Process	Audit Preview	Audit Release
2024-10-21	2024-10-21	2024-10-28	2024-10-28

Vulnerability Summary



5	0	5	5
Total Findings	Resolved	Pending	Unresolved

● 1 Critical	0 Resolved, 1 Pending	Critical risks are the most severe and can have a significant impact on the smart contracts functionality, security, or the entire system. These vulnerabilities can lead to the loss of user funds, unauthorized access, or complete system compromise.
● 1 High	0 Resolved, 1 Pending	High-risk vulnerabilities have the potential to cause significant harm to the smart contract or the system. While not as severe as critical risks, they can still result in financial losses, data breaches, or denial of service attacks.
● 1 Medium	0 Resolved, 1 Pending	Medium-risk vulnerabilities pose a moderate level of risk to the smart contracts security and functionality. They may not have an immediate and severe impact but can still lead to potential issues if exploited. These risks should be addressed to ensure the contracts overall security.
● 1 Low	0 Resolved, 1 Pending	Low-risk vulnerabilities have a minimal impact on the smart contracts security and functionality. They may not pose a significant threat, but it is still advisable to address them to maintain a robust security posture.

● 1 Informational	0 Resolved, 1 Pending	Informational risks are not actual vulnerabilities but provide useful information about potential improvements or best practices. These findings may include suggestions for code optimizations, documentation enhancements, or other non-critical areas for improvement.

PROJECT OVERVIEW | Jack Eth bundle.

Token Summary

Parameter	Result
Address	0x3b25C77A806bA3F014358180A99eD91cf5dbe694
Name	Jack Eth bundle
Token Tracker	Jack Eth bundle (Jack)
Decimals	18
Supply	0
Platform	ETHEREUM
Compiler	v0.8.24+commit.e11b9ed9
Contract Name	SPL
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Ganache
Codebase	https://etherscan.io/ token/0xED9E67ef7A90757A1C163d86aDc6b9cd7A930Cef#code

TECHNICAL FINDINGS | Jack Eth bundle.

Smart contract security audits classify risks into several categories: Critical, High, Medium, Low, and Informational. These classifications help assess the severity and potential impact of vulnerabilities found in smart contracts.

Classification of Risk

Severity	Description
● Critical	Critical risks are the most severe and can have a significant impact on the smart contracts functionality, security, or the entire system. These vulnerabilities can lead to the loss of user funds, unauthorized access, or complete system compromise.
● High	High-risk vulnerabilities have the potential to cause significant harm to the smart contract or the system. While not as severe as critical risks, they can still result in financial losses, data breaches, or denial of service attacks.
● Medium	Medium-risk vulnerabilities pose a moderate level of risk to the smart contracts security and functionality. They may not have an immediate and severe impact but can still lead to potential issues if exploited. These risks should be addressed to ensure the contracts overall security.
● Low	Low-risk vulnerabilities have a minimal impact on the smart contracts security and functionality. They may not pose a significant threat, but it is still advisable to address them to maintain a robust security posture.
ⓘ Informational	Informational risks are not actual vulnerabilities but provide useful information about potential improvements or best practices. These findings may include suggestions for code optimizations, documentation enhancements, or other non-critical areas for improvement.

By categorizing risks into these classifications, smart contract security audits can prioritize the resolution of critical and high-risk vulnerabilities to ensure the contract's overall security and protect user funds and data.

Jack-20 | Sensible Gas Options.

Category	Severity	Location	Status
Coding Style	🟢 Low	jack.zip: config.js	Detected

Description

The gas options provided in the GAS_OPTIONS object should be carefully considered..

Recommendation

The values for limit and price should be chosen based on the specific requirements of the application and the current gas prices on the network..

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

Jack-21 | Security of Private Keys.

Category	Severity	Location	Status
Coding Style	● Medium	jack.zip: utils.js	Detected

Description

The code generates a new wallet with a private key. .

Recommendation

It is important to ensure the secure storage and handling of private keys to prevent unauthorized access..

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

Jack-22 | Use of dotenv within the project..

Category	Severity	Location	Status
Coding Style	● High	jack.zip: bot.js	Detected

Description

The code uses the `dotenv` package to load environment variables from a `.env` file. This is a good practice for keeping sensitive information like API keys and credentials separate from the code..

Recommendation

Ensure that .env is excluded from github or any other security exposure as private keys can be stored within the .env..

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

Jack-23 | Private Key Handling.

Category	Severity	Location	Status
Coding Style	🔴 Critical	jack.zip: command.js	Detected

Description

The code prompts for private keys for the deployer and main wallets. Storing private keys in plain text is not secure..

Recommendation

It is recommended to use a secure key management solution or a hardware wallet for handling private keys..

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

Jack-24 | Input Validation.

Category	Severity	Location	Status
Coding Style	Informational	jack.zip: bot.js,initialcheck,js, utils.js,command.js,config.js	Detected

Description

The code does not include input validation for function parameters..

Recommendation

It would be beneficial to validate input parameters to ensure they meet the expected format and prevent potential vulnerabilities, such as SQL injection or code injection..

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

FINDINGS

In this document, we present the findings and results of the smart contract security audit. The identified vulnerabilities, weaknesses, and potential risks are outlined, along with recommendations for mitigating these issues. It is crucial for the team to address these findings promptly to enhance the security and trustworthiness of the smart contract code.

Severity	Found	Pending	Resolved
● Critical	1	1	0
● High	1	1	0
● Medium	1	1	0
● Low	1	1	0
ⓘ Informational	1	1	0
Total	5	5	0

In a smart contract, a technical finding summary refers to a compilation of identified issues or vulnerabilities discovered during a security audit. These findings can range from coding errors and logical flaws to potential security risks. It is crucial for the project owner to thoroughly review each identified item and take necessary actions to resolve them. By carefully examining the technical finding summary, the project owner can gain insights into the weaknesses or potential threats present in the smart contract. They should prioritize addressing these issues promptly to mitigate any risks associated with the contract's security. Neglecting to address any identified item in the security audit can expose the smart contract to significant risks. Unresolved vulnerabilities can be exploited by malicious actors, potentially leading to financial losses, data breaches, or other detrimental consequences. To ensure the integrity and security of the smart contract, the project owner should engage in a comprehensive review process. This involves understanding the nature and severity of each identified item, consulting with experts if needed, and implementing appropriate fixes or enhancements. Regularly updating and maintaining the smart contract's codebase is also essential to address any emerging security concerns. By diligently reviewing and resolving all identified items in the technical finding summary, the project owner can significantly reduce the risks associated with the smart contract and enhance its overall security posture.

SOCIAL MEDIA CHECKS | Jack Eth bundle.

Social Media	URL	Result
Website		Fail
Telegram		
Twitter	3	Fail
Facebook		N/A
Reddit	N/A	N/A
Instagram	N/A	N/A
CoinGecko		Fail
Github		N/A
CMC		Fail
Email		Contact
Other		N/A

From a security assessment standpoint, inspecting a project's social media presence is essential. It enables the evaluation of the project's reputation, credibility, and trustworthiness within the community. By analyzing the content shared, engagement levels, and the response to any security-related incidents, one can assess the project's commitment to security practices and its ability to handle potential threats.

Social Media Information Notes:

Auditor Notes: Website needs a bit of improvement.

Project Owner Notes:

Assessment Results | Final Audit Score Jack.

Review	Score
Security Score	70
Auditor Score	70

Our security assessment or audit score system for the smart contract and project follows a comprehensive evaluation process to ensure the highest level of security. The system assigns a score based on various security parameters and benchmarks, with a passing score set at 80 out of a total attainable score of 100.The assessment process includes a thorough review of the smart contracts codebase, architecture, and design principles. It examines potential vulnerabilities, such as code bugs, logical flaws, and potential attack vectors. The evaluation also considers the adherence to best practices and industry standards for secure coding. Additionally, the system assesses the projects overall security measures, including infrastructure security, data protection, and access controls. It evaluates the implementation of encryption, authentication mechanisms, and secure communication protocols. To achieve a passing score, the smart contract and project must attain a minimum of 80 points out of the total attainable score of 100. This ensures that the system has undergone a rigorous security assessment and meets the required standards for secure operation.

Audit Fail

AUDIT FAILED

Important Notes for Jack

- Security Assessment Summary for bot.js, command.js, config.js, and utils.js
- Overall Observations: The code generally adheres to security and coding best practices. Here are detailed observations and recommendations:
- 1. Importing Libraries:
- The code imports necessary libraries such as ethers, fs, and crypto. This ensures that only required functionality is included, which is a good practice for security and efficiency.
- 2. Function Naming:
- Functions are descriptively named and follow the camel case naming convention, enhancing readability and maintainability.
- 3. Error Handling:
- The code includes try-catch blocks to handle potential errors during file operations and wallet generation. This improves robustness and prevents crashes.
- 4. File Operations:

- Functions to read and write JSON data to files are included. However, additional error handling, such as checking if a file exists before reading or writing, would enhance reliability.
- 5. Randomness:
- The use of crypto.randomBytes() for generating private keys ensures secure random number generation, which is crucial for cryptographic operations.
- 6. Logging:
- Console.log statements are used for error handling and debugging. For more structured and configurable logging, consider using a logging library like Winston or Bunyan.
- 7. Code Comments:
- The code lacks comments explaining the purpose and functionality of certain functions. Adding comments would improve maintainability and understanding for other developers.
- 8. Input Validation:
- Input validation for function parameters is missing. Implementing validation can prevent potential vulnerabilities such as SQL injection or code injection.

- 9. Gas Calculation:█
- The code calculates gas limits and prices for Ethereum transactions. Using ethers.utils.parseUnits() to convert gas values from human-readable format to the required format would be more precise.█
- 10. Security of Private Keys:█
- The code generates new wallets with private keys. Ensuring secure storage and handling of these keys is critical to prevent unauthorized access.█
- 11. Code Formatting:█
- Consistent indentation and formatting are followed, which improves code readability and maintainability.█
- Additional Observations:█
- 1. Environment Variables:█
- The use of dotenv to load environment variables from a .env file is a good practice, keeping sensitive information like API keys and private keys separate from the code.█
- 2. Object-Oriented Programming:█
- Classes such as Project and Wallet are defined to organize and structure data, improving code readability and maintainability.█

- 3. Secure URLs and Contract Addresses:█
- URLs for Flashbots relay and Ethereum RPC endpoints should be validated and obtained from trusted sources. Contract addresses for Uniswap, WETH, GoLive, and Bribe contracts should be verified for accuracy.█
- 4. Sensible Gas Options:█
- Gas options in the GAS_OPTIONS object should be carefully considered based on application requirements and current network gas prices.█
- 5. Code Documentation:█
- Adding comments or documentation to explain the purpose and usage of each constant can improve code readability and make it easier for other developers to understand and maintain the code.█
- 6. Code Review:█
- Regular code reviews by experienced developers can help identify potential security vulnerabilities and improve coding best practices.█
- Conclusion:█
- The following code was inspected during the week of October 21st and concluded at October 28th 2025.█

- The folder shared was dated 9/26/2025.
- ------ 9/25/2024 7:30 AM 5772 initialCheck.js
- ------ 9/25/2024 9:57 AM 2023 config.js
- ------ 9/26/2024 7:26 AM 33077 bot.js
-

Auditor Score =70
Audit Fail

AUDIT
FAILED

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that actagainst the nature of decentralization, such as explicit ownership or specialized access roles incombination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code,such as a constructor assignment imposing different require statements on the input variables than a setterfunction.

Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias and is readable by all the developers.

Disclaimer

The purpose of this disclaimer is to outline the responsibilities and limitations of the security assessment and smart contract audit conducted by Bladepool/CFG NINJA. By engaging our services, the project owner acknowledges and agrees to the following terms:

1. Limitation of Liability: Bladepool/CFG NINJA shall not be held liable for any damages, losses, or expenses incurred as a result of any contract malfunctions, vulnerabilities, or exploits discovered during the security assessment and smart contract audit. The project owner assumes full responsibility for any consequences arising from the use or implementation of the audited smart contract. 2. No Guarantee of Absolute Security: While Bladepool/CFG NINJA employs industry-standard practices and methodologies to identify potential security risks, it is important to note that no security assessment or smart contract audit can provide an absolute guarantee of security. The project owner acknowledges that there may still be unknown vulnerabilities or risks that are beyond the scope of our assessment. 3. Transfer of Responsibility: By engaging our services, the project owner agrees to assume full responsibility for addressing and mitigating any identified vulnerabilities or risks discovered during the security assessment and smart contract audit. It is the project owner s sole responsibility to ensure the proper implementation of necessary security measures and to address any identified issues promptly. 4. Compliance with Applicable Laws and Regulations: The project owner acknowledges and agrees to comply with all applicable laws, regulations, and industry standards related to the use and implementation of smart contracts. Bladepool/CFG NINJA shall not be held responsible for any non-compliance by the project owner. 5. Third-Party Services: The security assessment and smart contract audit conducted by Bladepool/CFG NINJA may involve the use of third-party tools, services, or technologies. While we exercise due diligence in selecting and utilizing these resources, we cannot be held liable for any issues or damages arising from the use of such third-party services. 6. Confidentiality: Bladepool/CFG NINJA maintains strict confidentiality regarding all information and data obtained during the security assessment and smart contract audit. However, we cannot guarantee the security of data transmitted over the internet or through any other means. 7. Not a Financial Advice: Bladepool/CFG NINJA please note that the information provided in the security assessment or audit should not be considered as financial advice. It is always recommended to consult with a financial professional or do thorough research before making any investment decisions.

By engaging our services, the project owner acknowledges and accepts these terms and releases Bladepool/CFG NINJA from any liability, claims, or damages arising from the security assessment and smart contract audit. It is recommended that the project owner consult legal counsel before entering into any agreement or contract.

CFG NINJA
AUDITS