# Full Audit Report

## Jeti Services[Authenticate] Security Assessment

Real Cybersecurity
Protecting digital assets

Made in Thailand

SECURI LAB
(THAILAND) contact@securi-lab.com

FULL AUDIT REPORT

# Table of Contents 1

# FULL AUDIT REPORT

## Report Information

| | |
|---|---|
| About Report | **Jeti Services[Authenticate] Security Assessment** |
| Version | **v1.0** |
| Client | **Jeti Service** |
| Language | **Solidity** |
| Confidentiality | **Public** |
| Contract File | **Authenticate.sol**<br>SHA-1: e270cc64b84b65eb1909debfc27111f6526a53ee<br>https://testnet.bscscan.com/address/0x0b38AE57acc4B60D80FC024B0F23aF4be6922EC4#code<br>**Re-assessment**<br>**Authenticate.sol**<br>SHA-1: b36c871d2acea5a9089eb6fc675356c3f3e54f62<br>https://testnet.bscscan.com/address/0x4B1af0db5FB82974ec082e0E882aC34A4DF83F7E#code |
| Audit Method | **Whitebox** |
| Security Assessment Author | **Auditor**<br>**Mark K. [Security Researcher \| Redteam]**<br>**Kevin N. [Security Researcher \| Web3 Dev]**<br>**Yusheng T. [Security Researcher \| Incident Response]**<br><br>**Approve Document**<br>**Ronny C. CTO & Head of Security Researcher**<br>**Chinnakit J. CEO & Founder** |

*Audit Method

**Whitebox:** SECURI LAB Team receives all source code from the client to provide the assessment.
**Blackbox:** SECURI LAB Team receives only bytecode from the client to provide the assessment.

**Digital Sign (Only Full Audit Report)**

Digitally signed by Chinnakit J.
DN: cn=Chinnakit J., o=SECURI LAB (Thailand), ou=CEO & Founder, email=contact@securi-lab.com, c=TH
Date: 2023.05.12 11:54:55 +07'00'

# Disclaimer

Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as "**Source code**".
And **SECURI Lab** hereinafter referred to as "**Service Provider**", the **Service Provider** will not be held liable for any legal liability arising from errors in the security assessment. The responsibility will be the responsibility of the **Client**, hereinafter referred to as "**Service User**" and the **Service User** agrees not to be held liable to the **service provider** in any case. By contract **Service Provider** to conduct security assessments with integrity with professional ethics, and transparency to deliver security assessments to users The **Service Provider** has the right to postpone the delivery of the security assessment. If the security assessment is delayed whether caused by any reason and is not responsible for any delayed security assessments.
If **the service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security assessment. Or happened later All responsibility shall be sole with the **service user**.

**Security Assessment Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.**

**SECURI LAB disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull**

The SECURI LAB team has conducted a comprehensive security assessment of the vulnerabilities.
This assessment is tested with an expert assessment. Using the following test requirements

1. Smart Contract Testing with Expert Analysis By testing the most common and uncommon vulnerabilities.
2. Automated program testing It includes a sample vulnerability test and a sample of the potential vulnerabilities being used for the most frequent attacks.
3. Manual Testing with AST/WAS/ASE/SMT and reviewed code line by line
4. Visibility, Mutability, Modifier function testing, such as whether a function can be seen in general, or whether a function can be changed and if so, who can change it.
5. Function association test It will be displayed through the association graph.
6. This safety assessment is cross-checked prior to the delivery of the assessment results.

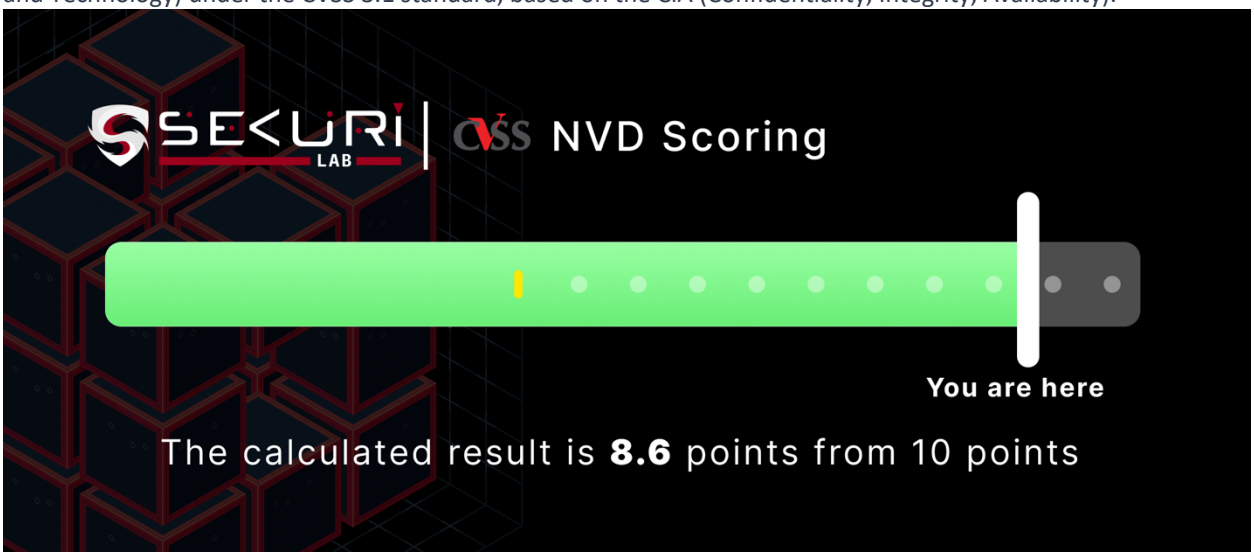
FULL AUDIT REPORT

# Executive Summary

For this security assessment, SECURI LAB received a request from Jeti Services on Friday, April 21, 2023.

## NVD CVSS Scoring

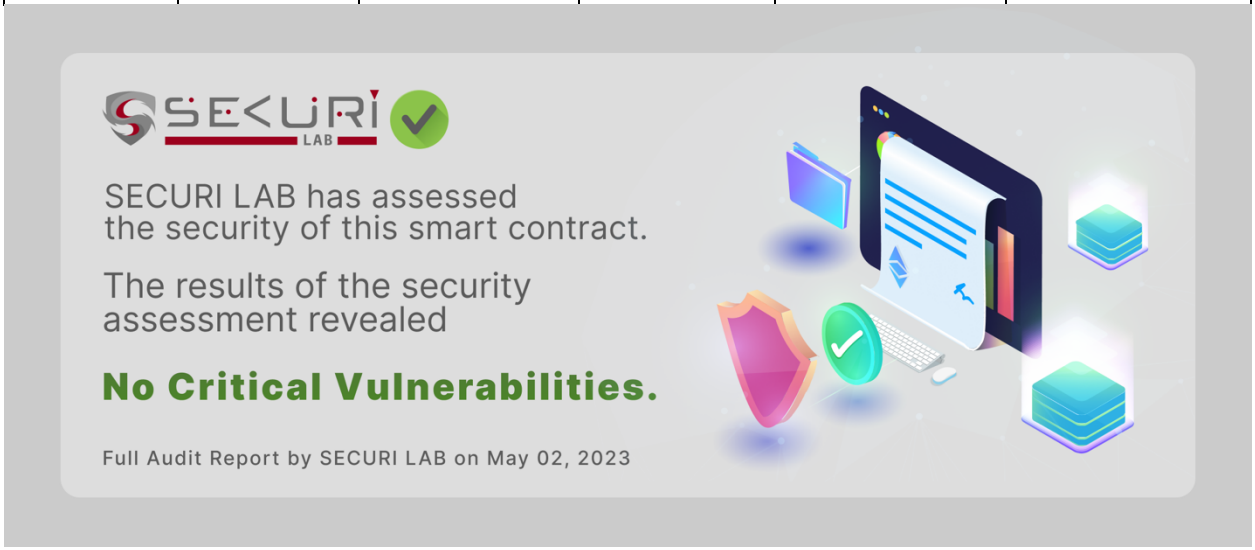
The score was calculated using the NVD (National Vulnerability Database) of NIST (National Institute of Standards and Technology) under the CVSS 3.1 standard, based on the CIA (Confidentiality, Integrity, Availability).




## Audit Result

SECURI LAB evaluated the smart contract security of the project and found: **[Total : 4]**

| Critical | High | Medium | Low | Very Low | Informational |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 2 | 1 | 0 |



# FULL AUDIT REPORT

## Project Introduction

**Scope Information:**

| | |
|---|---|
| Project Name | **Jeti Services** |
| Website | **https://jeti.one/** |
| Chain | **-** |
| Language | **Solidity** |

## Audit Information:

| | |
|---|---|
| Request Date | **Friday, April 21, 2023** |
| Audit Date | **Sunday, April 30, 2023** |
| Re-assessment Date | **Monday, May 8, 2023** |

## Audit Version History:

| Version | Date | Description |
|---|---|---|
| **1.0** | **Tuesday, May 2, 2023** | **Preliminary Report** |
| **1.1** | **Friday, May 12, 2023** | **Full Audit Report With Re-assessment** |

## FULL AUDIT REPORT

### Initial Audit Scope:

| | |
|---|---|
| Smart Contract File | **Authenticate.sol**<br>SHA-1: e270cc64b84b65eb1909debfc27111f6526a53ee<br>https://testnet.bscscan.com/address/0x0b38AE57acc4B60D80FC024B0F23aF4be6922EC4#code |
| Compiler Version | **v0.8.17** |

Source Units Analyzed: `1`
Source Units in Scope: `1` (**100%**)

| Type | File | Logic Contracts | Interfaces | Lines | nLines | nSLOC | Comment Lines | Complex. Score | Capabilities |
|---|---|---|---|---|---|---|---|---|---|
| 📝 🔍 | contracts/Authenticate.sol | 1 | 2 | 187 | 181 | 133 | 5 | 94 | 💰📪 🧮 |
| 📝 🔍 | **Totals** | **1** | **2** | **187** | **181** | **133** | **5** | **94** | 💰📪 🧮 |

Legend: [▔]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

## Re-assessment Audit Scope:

| | |
|---|---|
| Smart Contract File | **Authenticate.sol**<br>SHA-1: b36c871d2acea5a9089eb6fc675356c3f3e54f62<br>https://testnet.bscscan.com/address/0x4B1af0db5FB82974ec082e0E882aC34A4DF83F7E#code |
| Compiler Version | **v0.8.17** |

Source Units Analyzed: `1`
Source Units in Scope: `1` (**100%**)

| Type | File | Logic Contracts | Interfaces | Lines | nLines | nSLOC | Comment Lines | Complex. Score | Capabilities |
|---|---|---|---|---|---|---|---|---|---|
| 📝 🔍 | Authenticate.sol | 1 | 2 | 204 | 198 | 141 | 13 | 102 | 💰📩🧮 |
| 📝 🔍 | **Totals** | **1** | **2** | **204** | **198** | **141** | **13** | **102** | 💰📩🧮 |

Legend: [▔]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

## Dependencies / External Imports

| Dependency / Import Path | Count |
|---|---|
| @openzeppelin/contracts/access/Ownable.sol | 1 |
| @openzeppelin/contracts/utils/Address.sol | 1 |
| @openzeppelin/contracts/utils/cryptography/MerkleProof.sol | 1 |

Description Report Files Description Table

Initial Audit:

| File Name | SHA-1 Hash |
|---|---|
| contracts/Authenticate.sol | e270cc64b84b65eb1909debfc27111f6526a53ee |

Re-assessment Audit:

| File Name | SHA-1 Hash |
|---|---|
| contracts/Authenticate.sol | b36c871d2acea5a9089eb6fc675356c3f3e54f62 |

## Security Assessment Procedure

Securi has the following procedures and regulations for conducting security assessments:

**1.Request Audit** Client submits a form request through the Securi channel. After receiving the request, Securi will discuss a security assessment. And drafting a contract and agreeing to sign a contract together with the Client

**2.Auditing** Securi performs security assessments of smart contracts obtained through automated analysis and expert manual audits.

**3.Preliminary Report** At this stage, Securi will deliver an initial security assessment. To report on vulnerabilities and errors found under Audit Scope will not publish preliminary reports for safety.

**4.Reassessment** After Securi has delivered the Preliminary Report to the Client, Securi will track the status of the vulnerability or error, which will be published to the Final Report at a later date with the following statuses:

- **a.Acknowledge** The client has been informed about errors or vulnerabilities from the security assessment.
- **b.Resolved** The client has resolved the error or vulnerability. Resolved is probably just a commit, and Securi is unable to verify that the resolved has been implemented or not.
- **c.Decline** Client has rejected the results of the security assessment on the issue.

**5.Final Report** Securi providing full security assessment report and public

## Risk Rating

Risk rating using this commonly defined: $Risk\ rating\ =\ impact\ *\ confidence$

**Impact** The severity and potential impact of an attacker attack

**Confidence** Ensuring that attackers expose and use this vulnerability

Both have a total of 3 levels: **High**, **Medium**, **Low**. By *Informational* will not be classified as a level

| Confidence<br>Impact<br>[Likelihood] | Low | Medium | High |
|---|---|---|---|
| Low | **Very Low** | **Low** | **Medium** |
| Medium | **Low** | **Medium** | **High** |
| High | **Medium** | **High** | **Critical** |

FULL AUDIT REPORT

# Vulnerability Severity Summary

**Severity** is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,

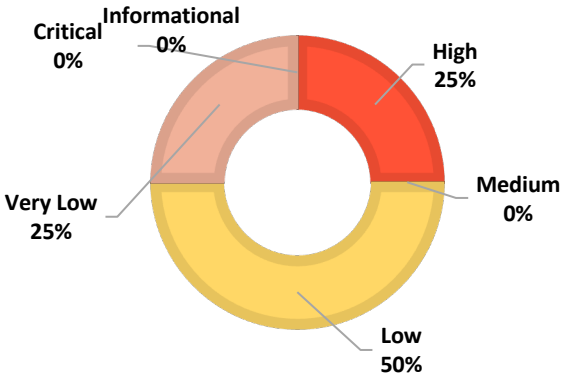
$Risk\ rating\ =\ impact\ *\ confidence$

It is categorized into

**5 categories based** on the **lowest severity**:

Very Low , Low , Medium , **High** , Critical .

For **Informational** & will **Non-class/Optimization/Best-practices will** not be counted as **severity**




| Vulnerability Severity Level | Total |
|---|---|
| Critical | 0 |
| High | 1 |
| Medium | 0 |
| Low | 2 |
| Very Low | 1 |
| Informational | 0 |
| Non-class/Optimization/Best-practices | 2 |

**Category information:**

Centralization
**Centralization Risk** is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission

Economics Risk
**Economics Risk** is Risks that may affect the economic mechanism system, such as the ability to increase Mint token

Logical Issue
**Logical Issue** is that can cause errors to core processing, such as any prior operations that cause background processes to crash.

Authorization
**Authorization** is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.

Mathematical
**Mathematical** Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.

Naming Conventions
**Naming Conventions** naming variables that may affect code understanding or naming inconsistencies

Security Risk
**Security Risk** of loss or damage if it's no mitigate

Coding Style
**Coding Style** is Tips coding for efficiency performance

Best Practices
**Best Practices** is suggestions for improvement

Optimization
**Optimization** is performance improvement

Gas Optimization
**Gas Optimization** is increase performance to avoid expensive gas

Dead Code
**Dead Code** having unused code This may result in wasted resources and gas fees.

FULL AUDIT REPORT

## Vulnerability Findings

| ID | Vulnerability Detail | Severity | Category | Status |
|---|---|---|---|---|
| SEC-01 | Centralization Risk | High | Centralization | **Mitigate** |
| SEC-02 | Avoid using block timestamp | Low | Best Practices | **Mitigate** |
| SEC-03 | `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()` | Low | Best Practices | **Resolved** |
| SEC-04 | avoid-encode-packed-rule | Very Low | Best Practices | **Resolved** |
| GAS-01 | Use Custom Errors | - | Gas Optimization | **Resolved** |
| GAS-02 | Use != 0 instead of > 0 for unsigned integer comparison | - | Gas Optimization | **Resolved** |

# SEC-01: Centralization Risk

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Centralization Risk | High | Check on finding | Centralization | **Mitigate** |

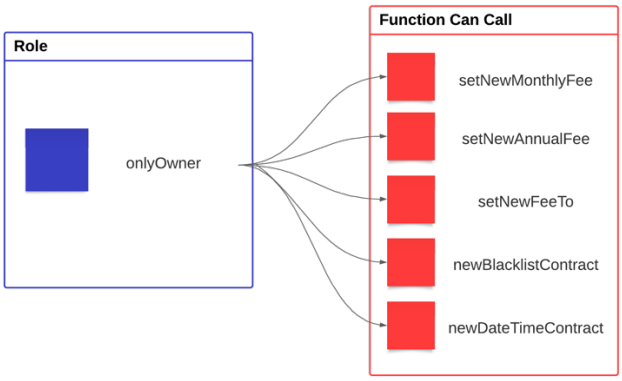
## Finding:

```
File: Authenticate.sol

16: contract Authenticate is Ownable {

158:  function setNewMonthlyFee(uint256[2] memory _newMonthlyFee) public onlyOwner {

164:  function setNewAnnualFee(uint256[2] memory _newAnnualFee) public onlyOwner {

170:  function setNewFeeTo(address _newFeeTo) public onlyOwner {

176:  function newBlacklistContract(address _newBlacklist) public onlyOwner {

182:  function newDateTimeContract(address _newDateTime) public onlyOwner {
```

## Scenario:

Centralized risk refers to the potential security risks that arise when a smart contract is controlled by a central entity or a single point of failure. If the contract is controlled by a central authority, then the contract may be vulnerable to attacks that target the centralized entity.
Centralized risk that can lead to rug pulls typically arises from the centralization of control or ownership of a project's assets, particularly in decentralize d finance (DeFi) projects built on blockchain platforms like Ethereum.

Contract Authenticate (File: Authenticate.sol)



In the Authenticate contract, Owner can call functions setNewMonthlyFee, setNewAnnualFee, setNewFeeTo, newBlacklistContract, newDateTimeContract. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.

**Recommendation:**

In terms of timeframes, there are three categories: short-term, long-term, and permanent.

For short-term solutions, a combination of timelock and multi-signature (2/3 or 3/5) can be used to mitigate risk by delaying sensitive operations and avoiding a single point of failure in key management. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; assigning privileged roles to multi-signature wallets to prevent private key compromise; and sharing the timelock contract and multi-signer addresses with the public via a medium/blog link.

For long-term solutions, a combination of timelock and DAO can be used to apply decentralization and transparency to the system. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; introducing a DAO/governance/voting module to increase transparency and user involvement; and sharing the timelock contract, multi-signer addresses, and DAO information with the public via a medium/blog link.

Finally, permanent solutions should be implemented to ensure the ongoing security and protection of the system.

**Alleviation:**

Regarding this, we discussed and found a solution to the matter with Jeti One Team, because the contract needed a function to be suspended—important settings to comply with the mechanism of the system. We deserve to see that such issues are addressed on Mitigate part and users are encouraged to follow and update platform announcements at all times.

# SEC-02: Avoid using block timestamp

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Avoid using block timestamp | Low | Check on finding | Best Practices | **Mitigate** |

## Finding:

```
File: Authenticate.sol

60:  require(project._startDate <= block.timestamp && project._expiry >=
block.timestamp, "Project is not active");

60:  require(project._startDate <= block.timestamp && project._expiry >=
block.timestamp, "Project is not active");

103:  projectInfo[counter]._startDate = block.timestamp;

104:  projectInfo[counter]._expiry = block.timestamp +
iDateTime(dateTimeContract).addMonths(block.timestamp, _length);

104:  projectInfo[counter]._expiry = block.timestamp +
iDateTime(dateTimeContract).addMonths(block.timestamp, _length);

115:  emit AddUserAndWebsite(msg.sender, _website, _projectMerkleRoot, _includeUsers,
block.timestamp, iDateTime(dateTimeContract).addMonths(block.timestamp, _length),
_fee, _refund);

115:  emit AddUserAndWebsite(msg.sender, _website, _projectMerkleRoot, _includeUsers,
block.timestamp, iDateTime(dateTimeContract).addMonths(block.timestamp, _length),
_fee, _refund);
```

## Recommendation:

Avoid relying on `block.timestamp`.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

## Exploit Scenario:

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

"Bob's contract relies on block.timestamp for its randomness. Eve is a miner and manipulates block.timestamp to exploit Bob's contract.

## Alleviation:

About avoiding block. timestamp We found that it was not a serious problem and block. timestamp is used correctly We see fit to keep it as a Mitigate.

## SEC-03: `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()` | Low | Check on finding | Best Practices | **Resolved** |

### Finding:

```
File: Authenticate.sol

147:  return keccak256(abi.encodePacked(_account));
```

### Recommendation:

Use `abi.encode()` instead which will pad items to 32 bytes, which will [prevent hash collisions](https://docs.soliditylang.org/en/v0.8.13/abi-spec.html#non-standard-packed-mode) (e.g. `abi.encodePacked(0x123,0x456)` => `0x123456` => `abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456)` => `0x0...1230...456`). "Unless there is a compelling reason, `abi.encode` should be preferred". If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` [instead](https://ethereum.stackexchange.com/questions/30912/how-to-compare-strings-in-solidity#answer-82739).
If all arguments are strings and or bytes, `bytes.concat()` should be used instead

### Exploit Scenario:

-

### Alleviation:

Jeti One Team has already resolved this issue.

## SEC-04: avoid-encode-packed-rule

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| avoid-encode-packed-rule | Very Low | Check on finding | Best Practices | **Resolved** |

### Finding:

```
function leaf(address _account) internal pure returns(bytes32) {
 return keccak256(abi.encodePacked(_account));
 }
```

### Recommendation:

abi.encodePacked function can be used to tightly pack the arguments for creating a hashed message. However, this method may lead to vulnerabilities if not used with caution. To avoid the risks associated with abi.encodePacked

### Exploit Scenario:

-

### Alleviation:

Jeti One Team has already resolved this issue.

## GAS-01: Use Custom Errors

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Use Custom Errors | - | Check on finding | Gas Optimization | **Resolved** |

### Finding:

```
File: Authenticate.sol

53:  require(projectOwner[msg.sender] == 1, "Not a valid project owner");

60:  require(project._startDate <= block.timestamp && project._expiry >=
block.timestamp, "Project is not active");

67:  require(project._includeUsers == 1, "Users are not allowed");

82:  require(_length == 1 || _length == 12, "You must select monthly or yearly");

83:  require(_includeUsers <= 1, "Invalid user selection");

84:  require(iBlacklist(blacklistContract).getBlacklist(msg.sender, address(this)) ==
0, "You are blacklisted!");

88:  require(msg.value >= _monthlyFee[_includeUsers], "You must pay the monthly fee");

91:  require(msg.value >= _annualFee[_includeUsers], "You must pay the annual fee");
```

### Recommendation:

[Source](https://blog.soliditylang.org/2021/04/21/custom-errors/)
Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

### Alleviation:

Jeti One Team has already resolved this issue.

## GAS-02: Use != 0 instead of > 0 for unsigned integer comparison

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Use != 0 instead of > 0 for unsigned integer comparison | - | Check on finding | Gas Optimization | **Resolved** |

### Finding:

```
File: Authenticate.sol

109:  if (_refund > 0) {
```

### Recommendation:

-

### Alleviation:

Jeti One Team has already resolved this issue.

## SWC Findings

| ID | Title | Scanning | Result |
|---|---|---|---|
| SWC-100 | Function Default Visibility | Complete | No risk |
| SWC-101 | Integer Overflow and Underflow | Complete | No risk |
| SWC-102 | Outdated Compiler Version | Complete | No risk |
| SWC-103 | Floating Pragma | Complete | No risk |
| SWC-104 | Unchecked Call Return Value | Complete | No risk |
| SWC-105 | Unprotected Ether Withdrawal | Complete | No risk |
| SWC-106 | Unprotected SELFDESTRUCT Instruction | Complete | No risk |
| SWC-107 | Reentrancy | Complete | No risk |
| SWC-108 | State Variable Default Visibility | Complete | No risk |
| SWC-109 | Uninitialized Storage Pointer | Complete | No risk |
| SWC-110 | Assert Violation | Complete | No risk |
| SWC-111 | Use of Deprecated Solidity Functions | Complete | No risk |
| SWC-112 | Delegatecall to Untrusted Callee | Complete | No risk |
| SWC-113 | DoS with Failed Call | Complete | No risk |
| SWC-114 | Transaction Order Dependence | Complete | No risk |
| SWC-115 | Authorization through tx.origin | Complete | No risk |

| | | | |
|---|---|---|---|
| SWC-116 | Block values as a proxy for time | Complete | No risk |
| SWC-117 | Signature Malleability | Complete | No risk |
| SWC-118 | Incorrect Constructor Name | Complete | No risk |
| SWC-119 | Shadowing State Variables | Complete | No risk |
| SWC-120 | Weak Sources of Randomness from Chain Attributes | Complete | No risk |
| SWC-121 | Missing Protection against Signature Replay Attacks | Complete | No risk |
| SWC-122 | Lack of Proper Signature Verification | Complete | No risk |
| SWC-123 | Requirement Violation | Complete | No risk |
| SWC-124 | Write to Arbitrary Storage Location | Complete | No risk |
| SWC-125 | Incorrect Inheritance Order | Complete | No risk |
| SWC-126 | Insufficient Gas Griefing | Complete | No risk |
| SWC-127 | Arbitrary Jump with Function Type Variable | Complete | No risk |
| SWC-128 | DoS With Block Gas Limit | Complete | No risk |
| SWC-129 | Typographical Error | Complete | No risk |
| SWC-130 | Right-To-Left-Override control character (U+202E) | Complete | No risk |
| SWC-131 | Presence of unused variables | Complete | No risk |
| SWC-132 | Unexpected Ether balance | Complete | No risk |

| | | | |
|---|---|---|---|
| SWC-133 | Hash Collisions With Multiple Variable Length Arguments | Complete | No risk |
| SWC-134 | Message call with hardcoded gas amount | Complete | No risk |
| SWC-135 | Code With No Effects | Complete | No risk |
| SWC-136 | Unencrypted Private Data On-Chain | Complete | No risk |

FULL AUDIT REPORT

# Visibility, Mutability, Modifier function testing

## Components

| 📝Contracts | 📚Libraries | 🔍Interfaces | 🎨Abstract |
|---|---|---|---|
| 1 | 0 | 2 | 0 |

## Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

| 🌐Public | 💰Payable |
|---|---|
| 12 | 1 |

| External | Internal | Private | Pure | View |
|---|---|---|---|---|
| 2 | 11 | 0 | 1 | 6 |

## StateVariables

| Total | 🌐Public |
|---|---|
| 11 | 5 |

## Capabilities

| Solidity Versions observed | 🧪 Experimental Features | 💰 Can Receive Funds | 🖥️ Uses Assembly | 💣 Has Destroyable Contracts |
|---|---|---|---|---|
| `=0.8.17` | | `yes` | | |

| 📩 Transfers ETH | ⚡ Low-Level Calls | 👥 DelegateCall | 🧮 Uses Hash Functions | 🧹 ECRecover | 🌀 New/Create/Create2 |
|---|---|---|---|---|---|
| `yes` | | | `yes` | | |

FULL AUDIT REPORT

| TryCatch | Σ Unchecked |
| --- | --- |
| | |

Contracts Description Table

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | | | | |
| └ | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **iBlacklist** | Interface | | | |
| └ | getBlacklist | External ❗ | | NO ❗ |
| | | | | |
| **iDateTime** | Interface | | | |
| └ | addMonths | External ❗ | | NO ❗ |
| | | | | |
| **Authenticate** | Implementation | Ownable | | |
| └ | | Public ❗ | 🛑 | NO ❗ |
| └ | addAdminAndWebsite | Public ❗ | 💵 | NO ❗ |
| └ | updateProjectMerkle | Public ❗ | 🛑 | isProjectOwner<br>isActive |
| └ | updateUserMerkle | Public ❗ | 🛑 | isProjectOwner<br>isActive<br>isUserAllowed |
| └ | authAdmin | Public ❗ | | isActive |
| └ | authUser | Public ❗ | | isActive<br>isUserAllowed |
| └ | leaf | Internal 🔒 | | |
| └ | _verifyAdmin | Internal 🔒 | | |
| └ | _verifyUser | Internal 🔒 | | |
| └ | setNewMonthlyFee | Public ❗ | 🛑 | onlyOwner |
| └ | setNewAnnualFee | Public ❗ | 🛑 | onlyOwner |

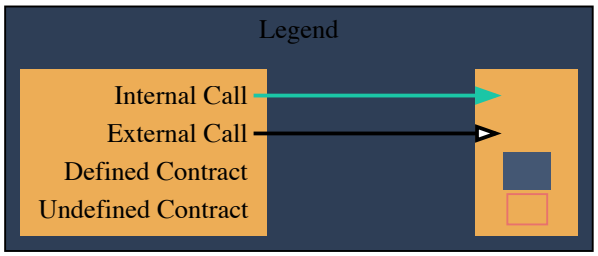
## FULL AUDIT REPORT

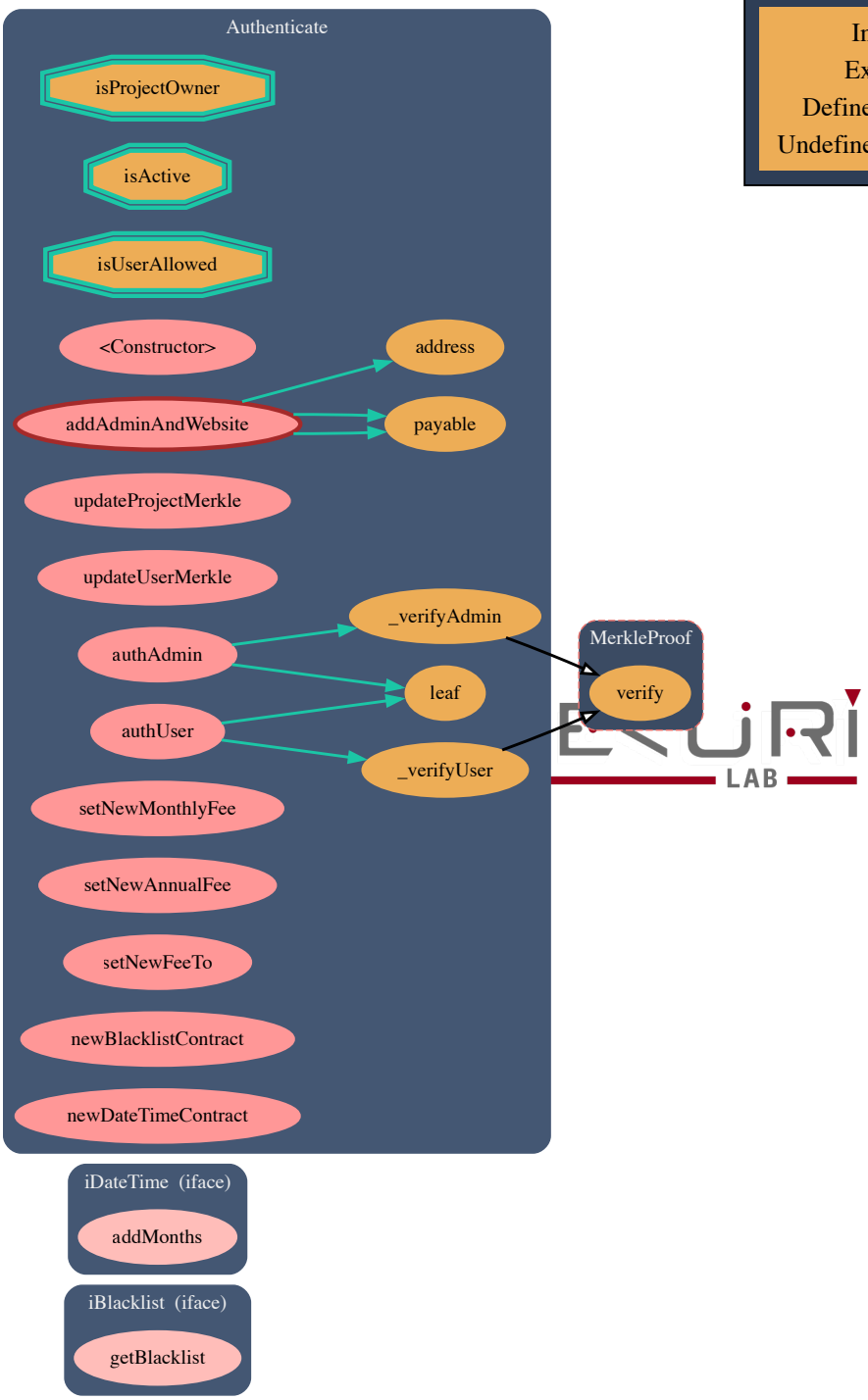
| Contract | Type | Bases | | |
|---|---|---|---|---|
| └ | setNewFeeTo | Public ! | 🛑 | onlyOwner |
| └ | newBlacklistContract | Public ! | 🛑 | onlyOwner |
| └ | newDateTimeContract | Public ! | 🛑 | onlyOwner |

Legend

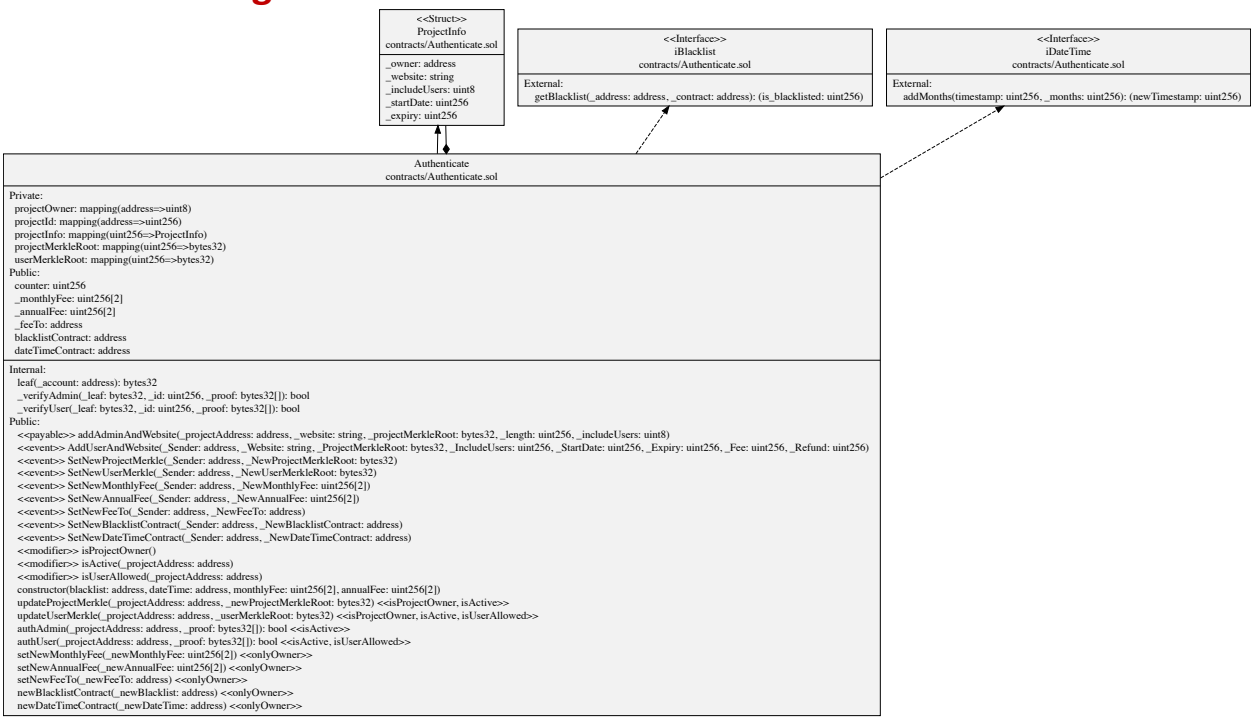
| Symbol | Meaning |
|---|---|
| 🛑 | Function can modify state |
| 💵 | Function is payable |

# Inheritate Function Relation Graph



# UML Class Diagram



# About SECURI LAB

SECURI LAB is a group of cyber security experts providing cyber security consulting, smart contract security audits, and KYC services.


## Follow Us On:

| | |
|---|---|
| Website | https://securi-lab.com/ |
| Twitter | https://twitter.com/SECURI_LAB |
| Telegram | https://t.me/securi_lab |
| Medium | https://medium.com/@securi |