



SOLIDProof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Dephaser

AUDIT

SECURITY ASSESSMENT

12. September, 2025

FOR



SolidProof_io



@solidproof_io

Introduction	4
Disclaimer	4
Project Overview	5
Summary	5
Social Medias	5
Audit Summary	6
File Overview	7
Imported packages	8
Audit Information	9
Vulnerability & Risk Level	9
Auditing Strategy and Techniques Applied	10
Methodology	10
Overall Security	11
Upgradeability	11
Ownership	13
Ownership Privileges	14
Minting tokens	14
Burning Tokens without Allowance	15
Blacklist addresses	16
Fees and Tax	17
Lock User Funds	18
Components	19
Exposed Functions	19
StateVariables	19
Capabilities	20
Inheritance Graph	21
Centralization Privileges	22
Audit Results	23
Critical issues	23
High issues	23



Medium issues	23
Low issues	23
Informational issues	23





Introduction

[SolidProof.io](https://solidproof.io) is a brand of the officially registered company Future Visions Deutschland. We're mainly focused on Blockchain Security, such as Smart Contract Audits and KYC verification for project teams.

Solidproof.io assesses potential security issues in the smart contracts implementations, reviews for potential inconsistencies between the code base and the whitepaper/documentation, and provides suggestions for improvement.

Disclaimer

[SolidProof.io](https://solidproof.io) reports are not, nor should they be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should they be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io does not cover testing or auditing the integration with external contracts or services (such as Unicrypt, Uniswap, PancakeSwap, etc.).

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analysed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyse.

Project Overview

Summary

Project Name	Dephaser
Website	https://dephaser.com/
About the project	DePhaser is a leading DeFi platform allowing users to exchange USDT for JPYT- a stablecoin pegged 1:1 to the Japanese Yen (¥) using a reliable external oracle. JPYT is fully backed by USDT deposits, ensuring transparency and stability, supported by USDT's strong reputation.
Chain	Optimism, BASE
Language	Solidity
Codebase Link	DephaserJPY Token: https://optimistic.etherscan.io/address/0xc47da4cb96ce65a96844a01bfae509f9d5454534#code DephaserJPY Token: https://basescan.org/address/0xc47da4cb96ce65a96844a01bfae509f9d5454534#code USDT Deposit Manager Proxy: https://optimistic.etherscan.io/address/0x77c1e40f5e7bb19c34d81f986fb13fe655f23cd9#code USDTDepositManager: https://optimistic.etherscan.io/address/0xb46aaeeb63aa3c41728f348e06d8c4db7c5ee9f5#code USDCDepositManager Proxy: https://basescan.org/address/0x0eab9d66adb8527cd7c0e7e7c1782580431ec276#code USDCDepositManager: https://basescan.org/address/0xf3b4eaaa9660143712fa3b3ebf5f460e3e8ea2cb#code
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/DePhaser
Twitter	https://x.com/DePhaser_JPYT
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	https://medium.com/@DePhaser_JPYT
Discord	https://discord.com/invite/vfbced9PjN
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	09. October 2024	<ul style="list-style-type: none"> • Layout Project • Automated- /Manual-Security Testing • Summary
v1.2	12. September 2025	<ul style="list-style-type: none"> • Reaudit

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes malicious outside manipulation of the contract's functions. This analysis did not include functional testing (or unit testing) of the contracts logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it. This includes internal calculations in the formulae used in the contract.



File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
UsdtDepositManager.sol	9gd67bb406cf04a22bghfty625fcb0732ghf6789
DephaserJPY.sol	78ba2f017de71b9d5b7082c91e30986df28bb394
UsdcDepositManager.sol	bc0e69470093c0ec248e5327c54617a0f4e3ec49

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) indicate a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	2
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol	2
@openzeppelin/contracts/access/AccessControl.sol	1
@openzeppelin/contracts/access/extensions/AccessControlEnumerable.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	2
@openzeppelin/contracts/utils/math/Math.sol	2
@openzeppelin/contracts/utils/math/SignedMath.sol	1
@src/constants/NumericConstants.sol	2
@src/constants/RoleConstants.sol	2
@src/interfaces/external/IAggregatorV3.sol	2
@src/interfaces/external/IERC20MintableBurnable.sol	2
@src/interfaces/external/IERC20PermitMinimal.sol	2
@src/interfaces/external/IPool.sol	2
@src/interfaces/external/IPyth.sol	1
@src/interfaces/internal/IDepositManager.sol	2
@src/libraries/PythUtils.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way

Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - a. Review the specifications, sources, and instructions provided to SolidProof to ensure we understand the smart contract's size, scope, and functionality.
 - b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
 - c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.
2. Testing and automated analysis that includes the following:
 - a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
 - b. Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
3. Review best practices, i.e., smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.
4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Upgradeability

The contract is upgradeable

✗ The deployer can update the contract with new functionalities

Description

The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.

Example

We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.

Comment

The deployer of the contract can upgrade and deploy a new version of the contract.

File/Line(s): L89-118

Codebase: UsdtDepositManager.sol

```
function initialize(
    address defaultAdmin↑,
    address pauser↑,
    address operator↑,
    address aavePoolAddress↑,
    address depositTokenAddress↑,
    address depositTokenUsdPriceFeedAddress↑,
    address jpyTokenAddress↑,
    address jpyUsdPriceFeedAddress↑,
    uint256 initialCooldownBlocks↑
)
{
    public
    initializer
    {
        __AccessControl_init();
        __UUPSUpgradeable_init();
        __ReentrancyGuard_init();
        __Pausable_init();

        _grantRole(DEFAULT_ADMIN_ROLE, defaultAdmin↑);
        _grantRole(PAUSER_ROLE, pauser↑);
        _grantRole(OPERATOR_ROLE, operator↑);
        _grantRole(UNPAUSER_ROLE, defaultAdmin↑);

        _setCooldownBlocks(initialCooldownBlocks↑);
        _setPriceFeed(jpyTokenAddress↑, jpyUsdPriceFeedAddress↑);
        _setPriceFeed(depositTokenAddress↑, depositTokenUsdPriceFeedAddress↑);

        // these variables must be immutable
        aavePool = IPool(aavePoolAddress↑);
        depositToken = IERC20(depositTokenAddress↑);
        jpyToken = IERC20MintableBurnable(jpyTokenAddress↑);
        aaveWrappedDepositToken = IERC20(IPool(aavePoolAddress↑).getReserveData(depositTokenAddress↑).aTokenAddress);
    }
}
```

File/Line(s): L95-131

Codebase: UsdcDepositManager.sol

```

function initialize(
    address defaultAdmin↑,
    address operator↑,
    address aavePoolAddress↑,
    address pythAddress↑,
    bytes32 usdJpyPriceId↑,
    uint256 validTimePeriod↑,
    address depositTokenAddress↑,
    address depositTokenUsdPriceFeedAddress↑,
    address jpyTokenAddress↑,
    uint256 initialCooldownBlocks↑
) public initializer {
    __AccessControl_init();
    __UUPSUpgradeable_init();
    __ReentrancyGuard_init();

    _grantRole(DEFAULT_ADMIN_ROLE, defaultAdmin↑);
    _grantRole(OPERATOR_ROLE, operator↑);

    _setCooldownBlocks(initialCooldownBlocks↑);
    _setPriceFeed(depositTokenAddress↑, depositTokenUsdPriceFeedAddress↑);
    _setPythValidTimePeriod(validTimePeriod↑);

    // these variables must be immutable
    aavePool = IPool(aavePoolAddress↑);

    pyth = IPyth(pythAddress↑);
    pythUsdJpyPriceId = usdJpyPriceId↑;

    depositToken = IERC20(depositTokenAddress↑);
    jpyToken = IERC20MintableBurnable(jpyTokenAddress↑);
    aaveWrappedDepositToken = IERC20(
        IPool(aavePoolAddress↑)
        .getReserveData(depositTokenAddress↑)
        .aTokenAddress
    );
}

```

Ownership

The ownership is not renounced

✗ The owner has not renounced ownership

Description	<p>The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including:</p> <ul style="list-style-type: none"> • Centralizations • The owner has significant control over contract's operations
Example	<p>We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.</p>
Comment	N/A

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refers to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who can add new tokens to the network's total supply.

The contract owner can mint new tokens	✗ The owner can mint new tokens
Description	Owners who have the ability to mint new tokens can reward themselves or other stakeholders, who can then sell the newly minted tokens on a cryptocurrency exchange to raise funds. However, there is a risk that the owner may abuse this power, leading to a decrease in trust and credibility in the project or platform. If stakeholders perceive that the owner is using their power to mint new tokens unfairly or without transparency, it can result in decreased demand for the token and a reduction in its value.
Example	If investors drive up the token price, the owner may choose to mint new tokens and sell them on a cryptocurrency exchange to raise funds. If the owner is not transparent and honest about their actions, they may be attempting a rugpull, where they suddenly abandon the project after raising funds, leaving investors with worthless tokens. This can lead to a decrease in the value of existing tokens, potentially rendering them worthless, and causing investors to suffer losses. It is essential for investors to carefully research the project and its developers and exercise caution before investing in any cryptocurrency or DeFi project.
Comment	The minter of the contract. Can mint an unlimited amount of tokens after the initial deployment which is not recommended as there must be a fixed supply present in the contract so that the supply of the tokens cannot be manipulated in the contract.

File, Line/s: L46-48

Codebase: DephaserJPY.sol

```
function mint(address to↑, uint256 amount↑) external onlyRole(MINTER_ROLE) {
    _mint(to↑, amount↑);
}
```



Burning Tokens without Allowance

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

The contract owner cannot burn tokens

 **The owner cannot burn tokens**

Description	The owner is not able burn tokens without any allowances.
Comment	N/A





Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

The contract owner cannot blacklist addresses



The owner cannot blacklist addresses

Description	The owner is not able to blacklist addresses to lock funds.
Comment	N/A





Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the contract's cost, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

The contract owner cannot levy high taxes.

 **The owner cannot levy unfair taxes**

Description

The owner of the contract cannot levy high taxes in the contract.

Comment

The operator of the contract can update the fees up to 1%.



Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

The contract owner cannot lock the contract



The owner cannot lock the contract

Description

The owner is not able to lock the contract by any functions or by updating any variables.

Comment

N/A

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

 Contracts	 Libraries	 Interfaces	 Abstract
3	0	0	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
38	0











External	Internal	Private	Pure	View
24	63	0	1	15

StateVariables

Total	 Public
27	27



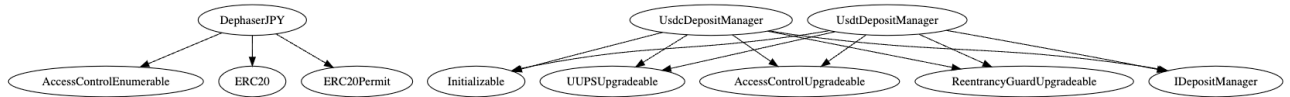
Capabilities

 Solidity Versions observed	Transfers ETH	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
0.8.27					
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECREcover	 New/Create/Create2
			yes		



Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if a single entity controls the contract or if certain participants have special permissions or abilities that others do not.

In the project, some authorities have access to the following functions:

File	Privileges
JPYTTToken.sol	<ul style="list-style-type: none"> • The minter can mint an unlimited amount of tokens after the initial deployment of token. • The burner role can burn his tokens. • The burner role can burn tokens from other wallet with allowance.
UsdtDepositManager.sol	<ul style="list-style-type: none"> • The operator can update the price feed address. • The operator can update the cooldown blocks between 1 to 100. • The operator can update the protocol fees upto 1%. • The operator can withdraw the fee amount from the contract. • The operator can withdraw the profit from the contract.
UsdcDepositManager.sol	<ul style="list-style-type: none"> • The operator can update the price feed address. • The operator can update the cooldown blocks between 1 to 100. • The operator can update the protocol fees upto 1%. • The operator can withdraw the fee amount from the contract. • The operator can withdraw the profit from the contract.

Recommendations

To avoid potential hacking risks, the client should manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security, e.g. Gnosis Safe
- Use of a timelock at least with a latency of, e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner can no longer modify any state variables of the contract. Make sure to set up everything before renouncing.

Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

#1 | The operator can lock claim.

File	Severity	Location	Status
UsdtDepositManager.sol	Medium	L357-368	Fixed

Description - The pause role can pause the withdrawal functionality from the contract for an indefinite period of time which is not recommended as this can lock the user funds and user will not be able to withdraw their tokens from the contract.

Alleviation - The functionality is removed from the contract. Hence, marking the issues to fixed.

Low issues

#1 | Missing Zero Address Validation

File	Severity	Location	Status
UsdtDepositManager.sol	Low	L307-309	Fixed

Description - Make sure to validate that the address passed in the function parameters is "non-zero".

Informational issues

No informational issues



Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.





**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY