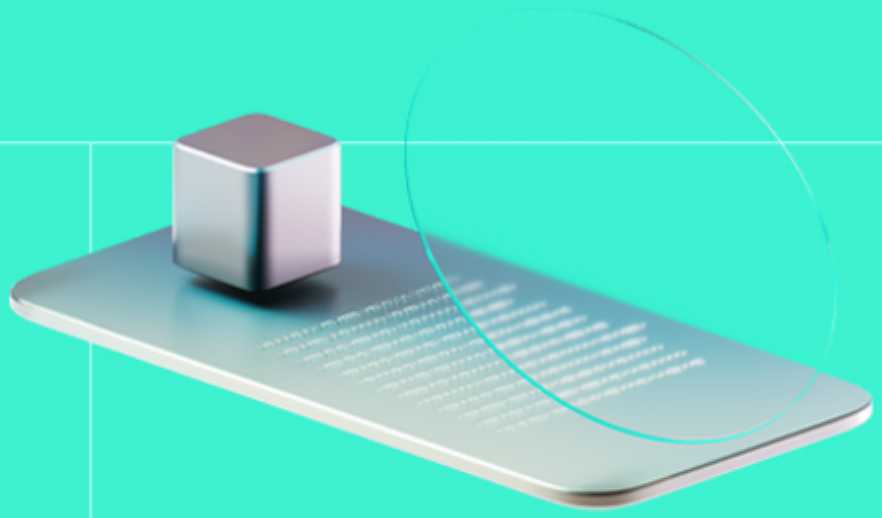
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Possum Labs

Date: 15/05/2024

We express our gratitude to the Possum Labs team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Platform: EVM

Language: Solidity

Tags: Stake; ERC20; ERC721

Timeline: 03/05/2024 - 08/05/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/PossumLabsCrypto/PortalsV2
Commit	17dd887dab420d4f39573cbb9c013e7106e02b0d

Audit Summary

10/10	9/10	96%	9/10
Security score	Code quality score	Test coverage	Documentation quality score

Total 9.6/10

The system users should acknowledge all the risks summed up in the risks section of the report

3	3	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	1
High	0
Medium	0
Low	2

Vulnerability	Status
F-2024-2252 - Missing registered portals validation	Fixed
F-2024-2256 - Impossible to mint token with ID equals 0	Fixed
F-2024-2453 - Arbitrage vulnerability in quoteBuyPortalEnergy and quoteSellPortalEnergy functions	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Possum Labs
Audited By	Carlo Parisi, Viktor Raboshchuk
Approved By	Przemyslaw Swiatowiec
Website	https://www.possumlabs.io/
Changelog	10/05/2024 - Preliminary Report
	14/05/2024 - Final Report

Table of Contents

System Overview

The primary purpose of Possum Portals (`Portals`) is to enable users to receive upfront, fixed-rate yield from DeFi staking opportunities instead of accruing yield over time at an unpredictable, variable rate. Portals enable duration-independent speculation on expected future yield rates of the composed yield sources (external protocols). Lastly, the funding mechanism of Portals allows PSM holders to deploy their tokens to a productive use case by lending them to the Portal for a potential profit. The protocol has the following contracts:

- src/MintBurnToken.sol - an ERC20 contract with a permit and burnable extension, mintable by owner
- src/PortalNFT.sol - This contract can save account information from a Portal user's stake and also return this information upon redemption. It has a single metadata URI that is used for all ID mints because the relevant account data is saved inside the NFT itself instead of outsourcing this to metadata. The metadata is merely a generic description, name, and picture.
- src/PortalV2MultiAsset.sol - contains the main business logic related to upfront yield. Most of the Portal functionality can only work when the vLP is activated. The contract accepts user deposits and withdrawals of a specific token, routing deposits to an external protocol for yield generation. Users accumulate portalEnergy points over time while staking their tokens, exchangeable for the PSM token via the internal Liquidity Pool or minted as ERC20. PortalEnergy Tokens can be burned to increase a recipient's internal balance, with users able to purchase more portalEnergy through the internal LP using PSM. bTokens received during funding initialize the internal LP and can be redeemed against the fundingRewardPool, consisting of PSM tokens. Users can mint NFTs representing their account balances, which are transferable and can be redeemed to add balances back internally.
- src/VirtualLP.sol - The shared, virtual liquidity pool that facilitates the payout of upfront yield and the recovery of yield over time. Hosts the integration of the external protocol that generates the yield on staked user assets. This contract acts as the shared, virtual LP for multiple Portals, each requiring registration by the owner for a predetermined duration. Each Portal must be registered by the owner, and once registered, Portals cannot be removed to ensure integrity. The full PSM amount within the LP is accessible to provide upfront yield for each Portal, with capital staked through connected Portals redirected to an external yield source. Yield is claimed and collected by this contract, which also accepts PSM tokens during the funding phase, issuing bTokens as a receipt. These bTokens initialize the internal LP and can be redeemed against the fundingRewardPool, filled over time with a 10% cut from the Converter, an arbitrage mechanism sweeping token balances. Upon triggering the Converter, the caller (arbitrager) must send a fixed amount of PSM tokens to the contract.
- src/interfaces/IPortalV2MultiAsset.sol - interface for the PortalV2MultiAsset contract
- src/interfaces/IVirtualLP.sol - interface for the VirtualLP contract.

Privileged roles

- The owner of the MintBurnToken contract can mint tokens
- The owner of the PortalNFT contract is the Portal that deploys it. Only the owner can call `mint()` and `redeem()`.
- The owner of the VirtualLP can create new Portals. The owner can be revoked by anyone. Only the registered Portal can send PSM to a user, and deposit/withdraw assets to/from external protocols.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation quality score is **9** out of **10**.

- Functional requirements are partially missed:
  - Intended outcome of mathematical operations.
- Technical description is provided.
- Natspec is sufficient.

Code quality

The total Code quality score is **9** out of **10**.

- The development environment is configured.
- Missing validations.
- Rounding errors are present in the code.

Test coverage

Code coverage of the project is **96%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Test in the remediations are not running because the stack is too deep.

Security score

Upon auditing, the code was found to contain **1** critical, **0** high, **0** medium, and **2** low severity issues. Out of these, **3** issues have been addressed and resolved, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.6** This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- There is no limit set for amount of ERC20 tokens can be created, as a result, the token owner can mint unlimited tokens, possibly disrupting the token supply and value.
- The lack of documentation regarding the mathematics operations poses a significant risk. This makes it harder for reviewers to understand what the code is meant to do, which is essential for accurately assess both its security and correctness.
- The ownership duration is configured for 9 days (7 days of the funding phase and 2 days of normal operation). Once this period elapses, the owner can be removed, effectively preventing the creation of new portals. Failure to register portals poses a risk to the protocol's functionality.
- There are 2 out of scope contracts, IWater and PSM token. IWater in particular is interacted with a lot in the VirtualLP.sol contract, but the contract functionality cannot be checked since it's out of scope, this will lower the quality of the audit.
- The convert() function in the VirtualLP.sol contract could potentially be exploited in scenarios where the liquidity pool is low on PSM tokens. An attacker could execute a sequence of transactions to extract value from the pool. The sequence involves selling a large amount of PSM tokens (thereby buying PE), executing the convert() function, and then buying back the initial amount of PSM tokens (by selling PE).

Findings

Vulnerability Details

F-2024-2453 - Arbitrage vulnerability in quoteBuyPortalEnergy and quoteSellPortalEnergy functions - Critical

Description: The `quoteBuyPortalEnergy()` and `quoteSellPortalEnergy()` functions in the `PortalV2MultiAsset.sol` contract are susceptible to an arbitrage vulnerability. This vulnerability arises when a user buys a large amount of Portal Energy (PE) tokens using the `quoteBuyPortalEnergy()` function and then immediately sells these tokens using the `quoteSellPortalEnergy()` function. The issue lies in the calculation of the reserves and the constant product, which can lead to a situation where the user receives more PSM tokens from the sale than they initially used for the purchase. This vulnerability could potentially be exploited by an attacker to drain the PSM token reserves, leading to significant financial loss for the liquidity pool

Assets:

- interfaces/IPortalV2MultiAsset.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 5/5

Exploitability: Independent

Complexity: Complex

Likelihood [1-5]: 5

Impact [1-5]: 5

Exploitability [0-2]: 0

Complexity [0-2]: 2

Final Score: 4.6 (Critical)

Severity: Critical

Recommendations

Remediation:

The arbitrage vulnerability in the `quoteBuyPortalEnergy()` and `quoteSellPortalEnergy()` functions appears to stem from a precision loss in the denominator of the `amountReceived` calculation in `quoteSellPortalEnergy()`. To mitigate this, it is recommended to add 1 WEI to the denominator of both quote calculations. This will result in a negligible loss for regular users but will effectively prevent attacks that severely imbalance the pool.

Additionally, it has been observed that the LP fee does not function as intended under extreme circumstances. Instead of reducing the `inputPE` by 1%, it would be more effective to reduce the output `PSM` by 1% in `quoteBuyPortalEnergy()`.

Lastly, it is recommended to increase the `LP_PROTECTION_HURDLE` from 1% to 2% or 3%. This would provide a stronger safeguard against potential attacks and further protect the liquidity pool.

Resolution:

The Finding was fixed in commit **a4e7509**.

`LP_PROTECTION_HURDLE` has been increased to 2%, the calculation for the protection is now happening at the end of the `quoteBuyPortalEnergy()` function and a +1 has been added to the denominator of both quote calculations.

Evidences

Arbitrage scenario

Reproduce:

Variables for the `quoteBuyPortalEnergy()` function:

- constant product: 2,5e25
- reserve0: 3e19
- reserve1: 833333
- PSM Input: 1e23
- PSM after lp protection hurdle calculation: 9,9e22
- PE received: 833080

Variables for the `quoteSellPortalEnergy()` function:

- constant product: 2,5e25
- reserve0: 1,0003e23
- reserve1: 249
- input PE: 833080

Output PSM: 1,000001108805766e23

Results:

The attacker received 0.110880576600000000 PSM more than he put in.

F-2024-2252 - Missing registered portals validation - Low

Description: The functions `collectProfitOfPortal` and `increaseAllowanceVault` should only be allowed to call for existing portals. However, these functions currently lack validation for the existence of portals.

Assets:

- VirtualLP.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status: Fixed

Classification

Impact: 2/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 2.5 (Low)

Severity: Low

Recommendations

Remediation: The functions should start with a check to verify if the called portal is registered.

Resolution: The Finding was fixed in commit **a4e7509**.

The functions are verifying that the portals are registered.

F-2024-2256 - Impossible to mint token with ID equals 0 - Low

Description: In the `mint()` function of the `PortalNFT.sol` contract, the `totalSupply` of tokens increases at the beginning of the function. Meaning that there won't be a token with an Id of 0. If the tokenId is incremented before minting, the very first tokenId will be unserviceable, and tokenId 0 won't be accessible. This poses a risk of compatibility issues with third-party services, that may rely on tokens with zero ID.

```
function mint(
....
) external onlyOwner returns (uint256 nftID) {
totalSupply++;
_safeMint(_recipient, totalSupply);
_setTokenURI(totalSupply, metadataURI);
```

Status: Fixed

Classification

Impact: 2/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 2.5 (Low)

Severity: Low

Recommendations

Remediation: It is considered best practice to increment the tokenId after minting. Alternatively, the documentation should be updated to explain why the `tokenId` is increased before minting.

Resolution: The Finding was fixed in commit **a4e7509**.

TokenID of 0 is now mintable.

Observation Details

F-2024-2251 - Missing events emitting for important functions - Info

Description:

Events for critical state changes should be emitted for tracking actions off-chain. It was observed that events in `VirtualLP.sol, PortalNFT.sol` are missing in the following functions:

- `withdrawFromYieldSource()`
- `depositToYieldSource()`
- `mint()`
- `redeem()`

Events are crucial for tracking changes on the blockchain, especially for actions that alter significant contract states or permissions. The absence of events in these functions means that external entities, such as user interfaces or off-chain monitoring systems, cannot effectively track these important changes.

Assets:

- PortalNFT.sol [https://github.com/PossumLabsCrypto/PortalsV2]
- PortalV2MultiAsset.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status: Mitigated

Recommendations

Remediation: To improve the transparency and traceability of these functions, consider emitting an event after each state modification.

Resolution: The Finding was mitigated in commit **a4e7509**.
Client Comment: “The mentioned 4 functions that don´t emit events can only be called by higher level functions that emit events. In interest of saving gas and byte-size, we won´t add events where they seem redundant.”

F-2024-2357 - Floating Point Precision by Rounding Error - Info

Description:

In both `withdrawFunding(), getBurnValuePSM(), quoteBuyPortalEnergy()` functions, there's a small rounding error. It arises from the calculation performed, leading to the inadvertent deletion of the last wei.

```
// line 549
uint256 withdrawAmount = (_amountBtoken * 100) / FUNDING_MAX_RETURN_PER
CENT;

// line 573
uint256 minValue = (_amount * 100) / FUNDING_MAX_RETURN_PERCENT;

// line 681
_amountInputPSM = (_amountInputPSM * (100 - LP_PROTECTION_HURDLE)) / 10
0;
```

This vulnerability emerges when Solidity's 256-bit precision is inadequate to accurately represent certain numbers with fractional components. As a result, arithmetic operations involving such numbers can lead to rounding errors, yielding inaccurate results. This vulnerability can compromise the reliability and accuracy of some of the contract's calculations.

Assets:

- VirtualLP.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status: Accepted

Recommendations

Remediation:

Consider adopting fixed-point arithmetic for decimal calculations. Utilizing libraries that support fixed-point arithmetic can enhance predictability and precision, surpassing the capabilities of floating-point arithmetic.

Resolution:

The Finding was mitigated in commit **a4e7509**.
Client Comment: “The precision loss of the last digit in the mentioned functions isn´t noticable in any economic reality. We prefer to keep the code simple with less libraries and mathematical operations over avoiding negligible precision loss.”

F-2024-2359 - Use private rather than public for constants - Info

Description:

In Solidity, constants represent immutable values that cannot be changed after they are set at compile-time. By default, constants have internal visibility, meaning they can be accessed within the contract they are declared in and in derived contracts. If a constant is explicitly declared as `public`, Solidity automatically generates a getter function for it. While this might seem harmless, it actually incurs a gas overhead, especially when the contract is deployed, as the EVM needs to generate bytecode for that getter. Conversely, declaring constants as `private` ensures that no additional getter is generated, optimizing gas usage.

VirtualLP.sol:

```
uint256 constant SECONDS_PER_YEAR = 31536000; // seconds in a 365 day y
ear
uint256 constant MAX_UINT = 115792089237316195423570985008687907853269984665640564039457584007913129639935;
uint256 public constant FUNDING_APR = 48; // annual redemption value in
crease (APR) of bTokens
uint256 public constant FUNDING_MAX_RETURN_PERCENT = 1000; // maximum r
edemption value percent of bTokens (must be >100)
uint256 public constant FUNDING_REWARD_SHARE = 10; // 10% of yield goes
to the funding pool until funders are paid back
address constant WETH_ADDRESS = 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1;address constant PSM_ADDRESS = 0x17A8541B82BF67e10B0874284b4Ae66858cb1fd5; // address of PSM token
address constant USDCE_WATER = 0x806e8538FC05774Ea83d9428F778E423F6492475;
address constant USDC_WATER = 0x9045ae36f963b7184861BDce205ea8B08913B48c;
address constant ARB_WATER = 0x175995159ca4F833794C88f7873B3e7fB12Bb1b6;
address constant WBTC_WATER = 0x4e9e41Bbf099fE0ef960017861d181a9aF6DDa07;
address constant WETH_WATER = 0x8A98929750e6709Af765F976c6bddb5BfFE6C06c;
address constant LINK_WATER = 0xFF614Dd6fC857e4daDa196d75DaC51D522a2ccf7;
```

PortalV2MultiAsset.sol:

```
address constant WETH_ADDRESS = 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1;
address constant PSM_ADDRESS = 0x17A8541B82BF67e10B0874284b4Ae66858cb1fd5; // address of PSM token
uint256 constant TERMINAL_MAX_LOCK_DURATION = 157680000; // terminal ma
ximum lock duration of a user stake in seconds (5y)
uint256 constant SECONDS_PER_YEAR = 31536000; // seconds in a 365 day y
ear
uint256 public constant LP_PROTECTION_HURDLE = 1; // percent reduction
of output amount when minting or buying PE
```

Assets:

- PortalV2MultiAsset.sol [https://github.com/PossumLabsCrypto/PortalsV2]
- VirtualLP.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status: Fixed

Recommendations

Remediation: To optimize gas usage in your Solidity contracts, declare constants with `private` visibility rather than `public` when possible. Using `private` prevents the automatic generation of a getter function, reducing gas overhead, especially during contract deployment.

F-2024-2363 - Missing validation before division - Info

Description:

The `quoteBuyPortalEnergy` function conducts a division involving the `reserve0` variable, but lacks a mechanism to ensure that reserve0 is not 0, which could lead to a division by zero error.

```
uint256 reserve0 = IERC20(PSM_ADDRESS).balanceOf(VIRTUAL_LP) - virtualL
P.fundingRewardPool();
// line 677
uint256 reserve1 = CONSTANT_PRODUCT / reserve0;
```

Within the context of Solidity, division by zero presents a critical concern due to its potential to trigger an exception, thereby abruptly terminating the execution of the smart contract. This abrupt halting of contract execution carries significant repercussions, including the potential loss of funds for users engaged with the contract and the prospect of an entire contract failure.

Assets:

- PortalV2MultiAsset.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status:

Accepted

Recommendations

Remediation:

Prior to performing any division operation, check whether the divisor is equal to zero. If the divisor is zero, handle the situation with error handling mechanisms to prevent exceptions. Use if-statements or require statements to verify that the divisor is not equal to zero before attempting the division operation.

Resolution:

The Finding was fixed in commit **a4e7509**.
Client Comment: “Cannot follow the recommendation because it kicks the contract above the byte-limit.
In practice, this will never cause a problem because the x*y=k formula will ensure that there is always some amount of PSM in the contract after swaps (reserve0).”

F-2024-2365 - Documentation mismatch - Info

Description: As per the documentation, the constant `_FUNDING_PHASE_DURATION` should be set to 604800 (which equals 7 days). Consequently, the VirtualLP constructor contains an if statement to ensure that the constant value falls within the range of 3 to 30 days.

```
if (
_FUNDING_PHASE_DURATION < 259200 || _FUNDING_PHASE_DURATION > 2592000
)
```

This discrepancy between the documentation and the contract's code leads to confusion and potential misunderstandings about the contract's behavior and capabilities.

Status: Mitigated

Recommendations

Remediation: Review the documentation and update the implementation to match the expected result.

Resolution: The Finding was fixed in commit **a4e7509**.
Client Comment: “To clarify, the specific implemetation in our release will have a 7 day funding phase duration which will be given as input in the constructor, but the contract allows for a more generic range of 3 and 30 days that will suit any reasonable use case. I don´t see a fundamental mismatch here since the chosen duration of the implemention is a subset of the generic range.”

F-2024-2454 - Unchecked return value from transfer functions - Info

Description:

The `stake()`, `buyPortalEnergy()`, `PSM_sendToPortalUser()`, `convert()`, `contributeFunding()`, `withdrawFunding()`, `burnBtokens()` functions currently lack a step in its implementation by not verifying the return value of the call to the `transferFrom()` and `transfer()` functions.

If the return value indicates an error condition, the absence of validation might lead to unintended consequences, including the completion of a transaction despite the presence of errors.

Assets:

- PortalV2MultiAsset.sol [https://github.com/PossumLabsCrypto/PortalsV2]
- VirtualLP.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status: Accepted

Recommendations

Remediation:

It's recommended to use the `safeTransferFrom()` and `safeTransfer()` method from the SafeERC20 and SafeERC721 libraries, which automatically checks the return value and reverts on failure.

Resolution:

The Finding was fixed in commit **a4e7509**.
Client Comment: “Plain "transfer" that doesn´t check return values is only used in functions that interact with tokens with a known, standard ERC20 behaviour (PE tokens, bTokens, PSM). These will cause a revert of the entire function call if the transfer fails, which to my knowledge makes the check of the return value irrelevant.”

F-2024-2455 - Missing checks for the zero address - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The "
Missing zero address control" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.
For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

Missing checks were observed in the following contracts:

- `./VitrualLP.sol: registerPortal()`

Assets:

- VirtualLP.sol [https://github.com/PossumLabsCrypto/PortalsV2]

Status: Accepted

Recommendations

Remediation:

It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Resolution:

The Finding was fixed in commit **a4e7509**.
Client Comment: “The zero checks seem redundant so they are left out.
Reasoning:

- Successfully registering address(0) as portal has no effect since this address cannot call functions on the LP contract
- _asset has a scenario where the address(0) is a valid input (when native ETH is the principal token)
- Address(0) is the default value of the mapping vaults[__portal][__asset]. Being able to set the mapping to the default value doesn´t cause a concern. The greater concern is to insert a wrong address which cannot be checked by an on-chain check, unless all addresses are hardcoded from the start, which is not possible because the vLP is deployed before the Portals.”

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/PossumLabsCrypto/PortalsV2
Commit	a4e7509a39c4d746024961e8ee230e7c42806aaf
Whitepaper	-
Requirements	https://github.com/PossumLabsCrypto/PortalsV2/tree/main/docs
Technical Requirements	https://github.com/PossumLabsCrypto/PortalsV2/tree/main/docs

Contracts in Scope

MintBurnToken.sol

PortalNFT.sol

PortalV2MultiAsset.sol

VirtualLP.sol

interfaces/IPortalV2MultiAsset.sol

interfaces/IVirtualLP.sol