byterocket

INVICTUS

Invictus Capital Fund Contract

January 27, 2022

1. Preface

The developers of **Invictus Capital** contracted byterocket to conduct a smart contract audit of their fund contract. Users are able to stake certain tokens (potentially also on different chains) with a different contract, while this one registers the user's deposit and distributes fund tokens to the user, representing their stake in the fund pool. These tokens can be redeemed via one of the redemption currencies.

The team of byterocket reviewed and audited the above smart contracts in the course of this audit. We started on the 5th and finished on the 14th of January 2022.

- *Manual Multi-Pass Code Review*
- *Automated Code Review*
- *In-Depth Protocol Analysis*
- *Formal Report*

byterocket gained access to the code via a public block explorer link. We based the audit on the contract state, deployed on December 15th, 2021. The updated version was provided to us via another public block explorer link.

2. Manual Code Review

We conducted a manual multi-pass code review of the smart contracts mentioned in section (1). Three different people went through the smart contract independently and compared their results in multiple concluding discussions.

The contracts are mostly written according to the latest standard used within the Ethereum community and the Solidity community's best practices, with some exceptions listed below. The naming of variables is very logical and understandable, which results in the contract being useful to understand. The code is very well documented, except for some occurrences listed below. The developers did not provide us with any previously written tests or scripts.

On the code level, we **found 25 bugs or flaws, with 25 being fixed and acknowledged**. Our manual review and analysis were additionally supported by multiple automated reviewing tools, like Slither, GasGauge, Manticore, and different fuzzing tools.

2.1 Findings

We are categorizing our findings into four different levels of severity. There are findings that have **No Severity**, where there are issues like documentation issues or non-conflicting typos. The first level with an actual impact on the contract's safety is **Low Severity**, where the found bug or flaw does have little impact. **Medium Severity** findings do already pose a real danger to the contract's users, but would usually not result in any lost funds or items. Our highest level - **High Severity** - indicates that there is a bug or flaw, that can be abused to steal funds, break the contract or otherwise gain control of it.

2.1.A - ERC20InvictusFund.sol - Line 2316 [NO SEVERITY] FIXED

```
address public constant dealAddress = 0x...;
```

According to the Solidity style guide, all constant values should be written in upper case.

Recommendation:
Rename the variable dealAddress to DEAL_ADDRESS to follow the official style guide.

Update on the 25th of January 2022:

The variable has been renamed to DEAL_ADDRESS accordingly.

2.1.B - General [LOW SEVERITY] FIXED

```
pragma solidity >=0.4.22 <0.9.0;
```

The contract uses a floating pragma, which might pose a security risk, as a known vulnerable compiler version may accidentally be selected or security tools might fall back to an older compiler version ending up actually checking a different EVM compilation. The deployed version made use of Solidity version 0.8.2, which is also outdated.

Recommendation:
Avoid floating pragmas. We highly recommend pinning a concrete compiler version (*latest without security issues*) in at least the top-level "deployed" contracts to make it unambiguous which compiler version is being used.

Update on the 25th of January 2022:
The Solidity version has been fixed to 0.8.11.

2.1.C - ERC20InvictusFund.sol - Line 2357 [LOW SEVERITY] FIXED

```
address public feeAddress;
```

The variable feeAddress is not initialized, which leads to a race condition in which the function dealRedemption() should not be called before the feeAddress variable is initialized with the updateFeeAddress() function, otherwise fees taken by the contract would be sent to the zero-address.

Recommendation:
Initialize feeAddress in initializeInvictusFund().

Update on the 25th of January 2022:
The feeAddress is now being initialized in initializeInvictusFund().

2.1.D - ERC20InvictusFund.sol - Line 2404-2410 [NO SEVERITY] FIXED

The documentation of the initializeInvictusFund() function does not contain any info about the trustedForwarder parameter. Also, there are typos in the word "*invesmtents*" and "*redmeptions*".

Recommendation:
Update the documentation and fix the typos.

Update on the 25th of January 2022:
The trustedForwarder parameter has been removed from the contract. The typos have also been fixed.

2.1.E - ERC20InvictusFund.sol - Line 2495-2497 [NO SEVERITY] FIXED

```
* @param transactionHash The ticker you would like to recieved your redemption in.
* @param investor The ticker you would like to recieved your redemption in.
* @param chainId The ticker you would like to recieved your redemption in.
```

The documentation of the queueInvestment() function is not correct for the parameters transactionHash, investor, and chainId.

Recommendation:
Update the documentation accordingly.

Update on the 25th of January 2022:
The documentation has been updated accordingly.

2.1.F - InvictusWhitelist.sol - Line 2287-2289 [HIGH SEVERITY] FIXED

```
function setTrustedForwarder(address trustedForwarder) public {
 _trustedForwarder = trustedForwarder;
}
```

This contract inherits from ERC2771ContextUpgradeable, which implements the protocol defined in EIP2771 for native meta transactions. The EIP states "*The Recipient MUST check that it trusts the Forwarder to prevent it from extracting address data appended from an untrusted contract. This could result in a forged address.*" as well as "*A bad forwarder may allow forgery of the msg.sender returned from _msgSender() and allow transactions to appear to be coming from any address.*".

In the current implementations, anyone can set the trusted forwarder.

Recommendation:
Change the function implementation to only allow calls from an address holding the appropriate role and include a zero-address check.

Update on the 25th of January 2022:

The ERC2771 functionality for native meta transactions has been removed entirely.

2.1.G - ERC20InvictusFund.sol - Line 2524-2529 [LOW SEVERITY] FIXED

```
require(_rates.length > 0, "No rates");
require(_tickers.length == supportedTickers.length, "Invalid ticker list");
```

The function is missing a check on whether the two input values are of the same length, which if that is not the case, would lead the function updateTickerPrices() to fail.

Additionally, the input value _tickers is not necessary, as it essentially has to be exactly the same array as supportedTickers. Subsequently, the costly comparison of hashes for every element in line 275 is also unnecessary.

Recommendation:
Remove the input value _tickers and make direct use of the supportedTickers variable. Additionally, add a check that ensures, that _rates has the same length as supportedTickers.

Update on the 25th of January 2022:
The _tickers parameter has been removed accordingly. The additional check has been added in the updateTickerPrices() function.

2.1.H - ERC20InvictusFund.sol - Line 2552-2556 [NO SEVERITY] FIXED

```
* @dev Removes liquidity, allowing owner to transfer stable coin to the fund wallet.
```

The documentation is misleading, as the function can be used to send any kind of ERC20 tokens held in the contract to the function caller.

Recommendation:
Update the documentation to reflect the functionality.

Update on the 25th of January 2022:
The documentation has been updated to reflect the correct functionality.

2.1.I - ERC20InvictusFund.sol - Line 2579-2582 [MEDIUM SEVERITY] FIXED

```
function setExitFeePercentage(uint24 newExitFeePercentage) external onlyOwner {
 require(exitFeePercentage > 0, "Invalid param");
 exitFeePercentage = newExitFeePercentage;
}
```

A fee can logically not be higher than 100%, which is not enforced in the function. Hence, the exit fee could theoretically be set to very high values like 100,000%.

Recommendation:
To protect yourselves from input failures and to give users peace of mind on maximum fee values, enforce a reasonable maximum value through a require statement.

Update on the 25th of January 2022:
The maximum fee value has been enforced to a technical maximum of 30%.

2.1.J - ERC20InvictusFund.sol - Line 2879-2582 [NO SEVERITY] FIXED

```
function setExitFeePercentage(uint24 newExitFeePercentage) external onlyOwner {
 require(exitFeePercentage > 0, "Invalid param");
 exitFeePercentage = newExitFeePercentage;
}
```

Functions that are changing state variables should emit an event to allow anyone an easy tracking of past values and changes.

Recommendation:
Add an event to be emitted after a successful change of the exit fee.

Update on the 25th of January 2022:
The function now emits an event, UpdateExitFeePercentage.

2.1.K - ERC20InvictusFund.sol - Line 2695-2697 [NO SEVERITY] FIXED

```
function updateDealingNonce(uint24 _dealingNonce) public onlyPricingOracle {
```

The updateDealingNonce() function is only used in dealing() and should not be public.

Recommendation:
Set the updateDealingNonce() function to internal since it should only be updated after a successful call to dealing() anyways.

Update on the 25th of January 2022:

The function has been set to internal.

2.1.L - ERC20InvictusFund.sol - Line 2590-2599 [NO SEVERITY] FIXED

```
address contractAddress,
```

The documentation of the contractAddress parameter is missing for the addTicker() function.

Recommendation:
Update the documentation to include the contractAddress parameter.

Update on the 25th of January 2022:
The documentation has been updated accordingly.

2.1.M - ERC20InvictusFund.sol - Line 2605-2607 [NO SEVERITY] FIXED

```
function updateFeeAddress(address newFeeAddress) public onlyOwner {
 feeAddress = newFeeAddress;
}
```

The function changing the feeAddress parameter does not feature a require statement which ensures that no zero address can be set.

Recommendation:
Add a require statement that ensures a valid address to be set.

Update on the 25th of January 2022:
The function now properly ensures that no zero address can be set as the feeAddress.

2.1.N - ERC20InvictusFund.sol - Line 397-399 [NO SEVERITY] ACKNOWLEDGED

```
function burnFrom(address account, uint256 amount) public virtual override onlyOwner {
 _burn(account, amount);
}
```

The burnFrom() function is only used in decreaseTokenAmount() and should not be public.

Recommendation:
Set the burnFrom() function to internal to ensure that no abuse may happen.

2.1.O - ERC20InvictusFund.sol - Line 421-428 [NO SEVERITY] FIXED

```
function updateTickerPrices(string[] memory _tickers, uint256[] memory rates) public onlyPricingOracle {
```

The updateTickerPrices() function is only used in dealing() and should not be public.

Recommendation:
Set the updateTickerPrices() function to internal.

Update on the 25th of January 2022:
The function has been set to internal.

2.1.P - ERC20InvictusFund.sol - Line 2841 [NO SEVERITY] FIXED

```
* @dev _handleRedemptionsDealing Handle the investments during dealing.
```

The documentation for the _handleInvestmentsDealing() function is wrong.

Recommendation:
Update the documentation accordingly.

Update on the 25th of January 2022:
The documentation has been updated accordingly.

2.1.Q - ERC20InvictusFund.sol - Line 2590-2595 [NO SEVERITY] FIXED

The function addTicker() adds a new ticker to the tickers array. Since subsequent functions are ensuring the correctness of a ticker via hashing comparisons, it is important whether a ticker is written in lower or upper cases. This is not reflected in either the code or the documentation.

Recommendation:
Either ensure a certain way of writing the ticker in the code, or ensure correct usage via updated documentation.

Update on the 25th of January 2022:
The documentation has been updated accordingly.

2.1.R - ERC20InvictusFund.sol - Line 2687-2689 [NO SEVERITY] FIXED

```
function updateCutOfftime(uint32 time) public onlyPricingOracle {
 cutOffTime = time;
```

```
}
```

The cutOffTime variable is used to define the time after which an investment will not be included anymore in a dealing round. Hence, this variable should not be set to values in the past or 0. Otherwise, the contract would not work anymore, since no investment would be accepted in any dealing round.

Recommendation:
Ensure a correct minimum value through a require statement, like cutOffTime >= block.timestamp.

Update on the 25th of January 2022:
The function now ensures that the cutOffTime is not in the past.

2.1.S - ERC20InvictusFund.sol - Line 2795-2797 [MEDIUM SEVERITY] FIXED

```
function ceil(uint256 value, uint256 roundLimit) private pure returns (uint256 r) {
 return ((value + roundLimit - 1) / roundLimit) * roundLimit;
}
```

Since OpenZeppelin's SafeMath is not being used, the calculations in the ceil() function can overflow. Additionally, with roundLimit being set to 0, an invalid calculation would be attempted via a zero division.

Recommendation:
Use OpenZeppelin's SafeMath library here as well.

Update on the 25th of January 2022:
The mathematical operations have been updated to use SafeMath.

2.1.T - ERC20InvictusFund.sol - Line 2734-2741 [NO SEVERITY] FIXED

```
uint256 fundTokenAmount,
```

The documentation of the dealInvestment() function is missing for the fundTokenAmount parameter.

Recommendation:
Update the documentation to include the fundTokenAmount parameter.

Update on the 25th of January 2022:
The documentation has been updated accordingly.

2.1.U - ERC20InvictusFund.sol - General [NO SEVERITY] FIXED

The developers have followed our advice and changed the Solidity version from a floating pragma "solidity >=0.4.22 <0.9.0;" to a fixed Solidity version 0.8.11. Since now version 0.8 is being used exclusively, it has become unnecessary to use SafeMath since this version already includes an inherit under-/overflow protection. Using SafeMath, in this case, is not unsafe by any means, but requires slightly more gas with each calculation.

Recommendation:
Replace any SafeMath calls with the regular mathematical expression. This will slightly help with the gas cost.

Update on the 27th of January 2022:
SafeMath has been removed from the contract.

2.1.V - ERC20InvictusFund.sol - Line 2665-2671 [NO SEVERITY] FIXED

The documentation of the initializeInvictusFund() function does not contain any info about the feeAddressInput parameter.

Recommendation:
Update the documentation.

Update on the 27th of January 2022:
The corresponding documentation has been added.

2.1.W - InvictusWhitelist.sol - Line 2528-2530 [NO SEVERITY] FIXED

The initialize() function is using the _setupRole() function to set up roles. This function has been deprecated, as referenced here. The new function is called _grantRole(). While _setupRole() internally just calls _grantRole(), it just makes sense to directly call _grantRole().

Recommendation:
Replace _setupRole() calls with _grantRole() calls.

Update on the 27th of January 2022:
The _setupRole() calls have been replaced with _grantRole() calls.

2.1.X - ERC20InvictusFund.sol - Line 2601 [LOW SEVERITY] FIXED

```
struct Ticker {
Price price;
address contractAddress;
uint24 decimals;
}
```

The decimals are implemented with a uint24 variable, while the ERC20 standard defines this as a uint8. Using a uint24, in this case, requires a further check to ensure that the decimals are always less than 255, to circumvent an overflow.

Recommendation:
Use uint8 for decimals in an ERC20 environment.

Update on the 27th of January 2022:
The decimals variable has been changed to uint8.

2.1.Y - ERC20InvictusFund.sol - Line 2799-2805 [NO SEVERITY] FIXED

```
require(address(token) != address(0), "Invalid address");
uint256 balance = token.balanceOf(address(this));

token.safeTransfer(owner(), token.balanceOf(address(this)));
emit TokensClaimed(address(token), balance);
```

The token balance is retrieved in line 2801 of the claimTokens() function and stored into a variable. This variable is being used in line 2804 to emit an event, but not in line 2803, where the same balance is retrieved once again, which is unnecessary.

Recommendation:
Replace the second balanceOf() call with the balance variable.

Update on the 27th of January 2022:
The second balanceOf() call has been replaced accordingly.

3. Protocol/Logic Review

Part of our audits are also analyses of the protocol and its logic. A team of three auditors went through the implementation and documentation of the implemented protocol.

No outside tests or documentation has been provided to us, but the function and inline documentation are sufficient to fully understand the intended protocol that is being implemented.

According to our analysis, the protocol and logic are working as intended, given that the findings listed in section (2) with the severity of low or higher are fixed.

We were **not able to discover any additional problems** in the protocol implemented in the smart contract.

4. Summary

During our code review (*which was done manually and automated*), we found **25 bugs or flaws, with 25 being fixed and acknowledged**.

The protocol review and analysis did neither uncover any game-theoretical nature problems nor any other functions prone to abuse.

In general, there are some improvements that can be made to the overall quality of the code and its documentation.

information provided. The auditors hereby disclaim, and each client or user of this audit report hereby waives, releases and holds all auditors harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Download the Report

Stored on IPFS

We store our public audit reports on IPFS; a peer-to-peer network called the "**I**nter **P**lanetary **F**ile **S**ystem". This allows us to store our reports in a distributed network instead of just a single server, so even if our website is down, every report is still available.

Learn more about IPFS →

Signed On-Chain

The IPFS Hash, a unique identifier of the report, is signed on-chain by both the client and us to prove that both sides have approved this audit report. This signing mechanism allows users to verify that neither side has faked or tampered with the audit.

Check the Signatures →