



Gibble

Interim - Smart Contract Audit Report

DATE: 1 March 2024 | **PREPARED BY:** Entersoft Pte Ltd

Contents

Revision History & Version Control

1.0 Disclaimer	4
2.0 Overview	5
2.1 Project Overview	6
2.2 Scope	6
2.3 Project Summary	6
2.4 Audit Summary	6
2.5 Security Level References	7
2.6 Vulnerability Summary	7
4.1 Use of Unsafe ERC20 operations in settleFilled function	9-15
4.2 Potential DoS Attack due to unrestricted Array Size	
4.3 Centralization Risk for Trusted Owners	
4.4 Lack of Error Handling in claimAllYield Function	
4.5 Lack of Token ID Validation Checks in createToken and tokenToggleActivation Function	
4.6 Lack of Validation Checks in calculate Function	
4.7 Unsafe Use of block.timestamp in timeToLiquidate Function	
4.8 Potential PUSH0 Compatibility Issue	
4.9 Use of Assembly in _getOwnStorage Function	
4.10 Potential Loss of Precision in settleCancelled Function	
4.11 Solidity Pragma Specificity Issue	
4.12 Functions Usage and Visibility Issue	
4.13 Usage of Constants Instead of Literals	
4.14 Events Missing indexed Fields	
5.1 Code Review / Manual Analysis	16
5.2 Tools Used for Audit	17

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	1 March 2024	R. Dixit G. Singh D. Bhavar M. Gowda	Interim report

Entersoft was commissioned by Gibble to perform Smart contract code review on their product. The review was conducted from 29th February 2024 to 1st March 2024, to ensure overall code quality, security, and correctness, and ensure that the code will work as intended.

The report is structured into two main sections:

- Executive Summary which provides a high-level overview of the audit findings.
- Technical Analysis which offers a detailed analysis of the smart contract code.

Please note that the analysis is static and entirely limited to the smart contract code. The information provided in this report should be used to understand the security and quality of the code, as well as its expected behavior.

Scope included:

- Market.sol

Standards followed include:

- OWASP (partially, for instance, role validations, input validations, etc.)
- Solidity best security practices

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to (i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, you must read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose, and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

Entersoft has meticulously audited the smart contract project from 29th February 2024 to 1st March 2024, with a primary focus on Solidity code files integral to blockchain functionality, emphasizing vulnerabilities in associated gas claiming. The working of basic functionalities was also tested during the review.

2.2 Scope

The audit scope covers the Gibble smart contracts available in the GitHub private repository. The audit focused on the checklist provided for the smart contract code audit.

1. Market.sol

2.3 Project Summary

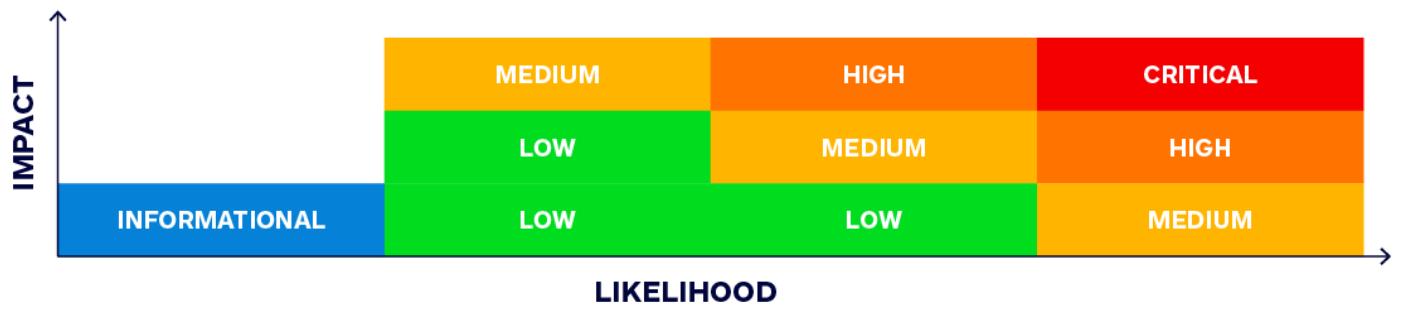
Name	Verified	Audited	Vulnerabilities
Gibble	Yes	Yes	Please review Section 4

2.4 Audit Summary

Delivery Date	Method of Audit	Consultants Engaged
1st March 2024	Static Analysis: The static part of a smart contract audit refers to the process of reviewing the code of a smart contract to identify security vulnerabilities, coding errors, and adherence to best practices without executing the code and without interacting with it in a live environment. The smart contract is reviewed as is. In this we will cover logic vulnerabilities, dependency security flaws and more.	4

2.5 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:



2.6 Vulnerability Summary

● Critical	● High	● Medium	● Low	● Informational
0	2	1	5	6

3.0 Executive Summary

Entersoft has conducted a comprehensive technical audit of the Gibble project through a comprehensive smart contract audit. The primary objective was to identify potential vulnerabilities and risks within the codebase, ensuring adherence to industry-leading standards while prioritizing security, reliability, and performance. Our focus was on prompt and efficient identification and resolution of vulnerabilities to enhance the overall robustness of the solidity smart contract.

Testing Methodology:

Our testing methodology in Solidity adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures.

We have performed a detailed manual analysis, adherence to industry standards, and the use of a comprehensive toolset. Our approach ensured a thorough evaluation within the designated Solidity code files.

Findings and Security Posture:













Our primary focus was on Access Control Policies, Transaction Signature Validations, Reentrancy, Time Manipulation, Default Visibility, Outdated Compiler Version, Input Validation, Deprecated Solidity Functions, Shadowing State Variables, Presence of Unused Variables, Overflow and Underflow Conditions, Assets Integrity, Errors and Exception.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from February 29, 2023, to March 1, 2024, our team diligently assessed and validated the security posture of the solidity smart contract, ultimately classifying it as "Secure," reflecting the absence of identified vulnerabilities and the robustness of the codebase against potential threats.

Result

The following table provides an overall summary of the findings and security posture of the smart contract code in scope.

-  **No Security vulnerabilities were identified**
-  **Security vulnerabilities were identified**

#	Gibble Audit Attack Vectors	Result
1	Access Control Policies	
2	Transaction Signature Validations	
3	Default Visibility	
4	Address Validation	
5	Reentrancy attacks	
6	Presence of unused variables	
7	Overflow and Underflow Conditions	
8	Assets Integrity	
10	Program Validations	
11	DOS	
12	Time Manipulation Vulnerability	
13	ERC20 token issues	
Overall Security Posture		NOT SECURE

4.0 Technical Analysis

The following results are the efforts of static analysis.

Note: The following values for “Result” mean:

- **PASS** indicates that there is no security risk.
- **FAIL** indicates that there is a security risk that needs to be remediated.
- **Informational** findings should be followed as a best practice, and they are not visible from the solidity smart contract.
- **Not Applicable** means the attack vector is Not applicable or Not available

4.1 Use of Unsafe ERC20 operations in settleFilled function

Result	FAIL
Severity	High
Description	The settleFilled function within the smart contract does not implement proper error handling by failing to check the return value of the 'transferFrom' function when transferring tokens using the iToken contract. This oversight introduces a significant risk, as some tokens may not revert on a failed transfer, potentially leading to an inconsistent state and the loss of funds.
Location / Source File	Market.sol
Observation	The settleFilled function lacks validation of the return value of the 'transferFrom' function, leaving the contract vulnerable to potential inconsistencies if the transfer fails without reverting.
Remediation	To address this issue, it is recommended to replace the 'transferFrom' function with 'transferFrom' in the settleFilled function. This change will enable proper error handling by checking the return value of the token transfer and reverting the transaction with a reason if the transfer fails. By implementing this solution, transaction integrity will be ensured, and the risk of potential loss of funds will be mitigated.
Reference	NA

4.2 Potential DoS Attack due to unrestricted Array Size

Result	FAIL
Severity	High
Description	The functions forceCancelOffers, cancelOffers, settleFilleds and settleCancelleds functions within the smart contract do not limit the length of the offerIds array, potentially exposing the contract to denial-of-service (DoS) attacks. Without a length limitation, malicious actors could potentially submit an excessively large array of offerIds, causing the function to consume excessive gas and leading to a DoS attack by exhausting the contract's resources.
Location / Source File	Market.sol functions (forceCancelOffers, cancelOffers, settleFilleds and settleCancelleds)
Observation	These functions do not enforce a maximum length for the offerIds array, leaving the contract vulnerable to DoS attacks through excessive gas consumption.
Remediation	To mitigate the risk of DoS attacks, it is recommended to implement a length limitation for the offerIds array in the mentioned functions. By enforcing a maximum array length, the contract can prevent malicious actors from overwhelming the functions with excessively large arrays, thereby safeguarding the contract's resources and ensuring uninterrupted operation.

Reference	NA
-----------	----

4.3 Centralization Risk for Trusted Owners

Result	FAIL
Severity	Medium
Description	The contracts 'Market.sol' and 'SampleToken.sol' exhibit a centralization risk due to the privileged rights granted to specific roles or the contract owner. These privileges include critical administrative tasks such as claiming yield, pausing the contract, minting tokens, and others.
Location / Source File	Market.sol & SampleToken.sol
Observation	<p>Certain functions in 'Market.sol' are accessible only to the 'Operator_Role', concentrating significant power in the hands of a few designated operators.</p> <p>The mint function in 'SampleToken.sol' is restricted to the contract owner, centralizing the control over token creation.</p>
Remediation	<p>Implement decentralized governance mechanisms to distribute administrative tasks and reduce reliance on single entities.</p> <p>Consider using multi-signature schemes to require multiple approvals for critical operations.</p> <p>Audit and revise access control mechanisms to ensure proper segregation of duties and reduce centralization risks.</p>
Reference	NA

4.4 Lack of Error Handling in claimAllYield Function

Result	FAIL
Severity	Low
Description	The claimAllYield function within the smart contract lacks error handling to check the return value of the BLAST.claimAllYield function call. This omission can lead to potential issues where the function call fails but no indication is provided to the caller, leading to ambiguity and potential misuse of the contract.
Location / Source File	Market.sol
Observation	The claimAllYield function does not check the return value of the BLAST.claimAllYield function call, which may result in unhandled failures and ambiguity for the caller.
Remediation	Implement error handling in the claimAllYield function to check the return value of the BLAST.claimAllYield function call and revert with a reason if the call fails. This will provide clarity and prevent potential misuse of the contract.
Reference	NA

4.5 Lack of Token ID Validation Checks in createToken and tokenToggleActivation Function

Result	FAIL
Severity	Low
Description	These functions within the smart contract does not include validation checks for the tokenId parameter, potentially allowing for the creation of tokens with invalid or unauthorized identifiers.
Location / Source File	Market.sol
Observation	The mentioned functions does not validate the tokenId parameter.
Remediation	Implement token ID validation checks in the createToken function to ensure that only valid and authorized token identifiers are accepted. This will help prevent unauthorized access and mitigate potential security risks associated with invalid token identifiers.
Reference	NA

4.6 Lack of Validation Checks in calculate Function

Result	FAIL
Severity	Low
Description	The calculate function within the smart contract lacks validation checks for the <code>climbRate</code> and <code>daysPassed</code> parameters, which are used in the calculation. This omission introduces a potential risk as it may allow for the execution of the function with invalid or unauthorized inputs, leading to inaccurate results or unexpected behavior.
Location / Source File	Market.sol
Observation	The createToken function does not validate the tokenId parameter.
Remediation	To mitigate this risk, it is recommended to add validation checks for the <code>climbRate</code> and <code>daysPassed</code> parameters in the calculate function. This can be achieved by implementing preconditions to ensure that the inputs meet the required criteria before proceeding with the calculation. By adding these validation checks, the contract can enforce the use of valid and authorized inputs, reducing the risk of inaccurate results or unexpected behavior.
Reference	NA

4.7 Unsafe Use of block.timestamp in timeToLiquidate Function

Result	FAIL
Severity	Low
Description	The timeToLiquidate function within the smart contract utilizes block.timestamp to determine whether it is time to liquidate a token. This practice introduces a potential security vulnerability as block.timestamp can be manipulated by miners, leading to inaccurate or unreliable results. Adversaries could exploit this vulnerability by manipulating block timestamps to bypass intended liquidation conditions or disrupt the contract's functionality.
Location / Source File	Market.sol
Observation	The timeToLiquidate function relies on block.timestamp to determine the liquidation status of a token, which exposes the contract to potential manipulation by miners.
Remediation	To mitigate the risk associated with reliance on block.timestamp, it is recommended to use a more secure and tamper-resistant time source, such as block.number or an external timestamp oracle, in the timeToLiquidate function. By utilizing a more robust time source, the contract can reduce its susceptibility to manipulation and ensure the accuracy of liquidation conditions.
Reference	NA

4.8 Potential PUSH0 Compatibility Issue

Result	PASS
Severity	Low
Description	The pragma directive in the contract 'Market.sol' at line 2 specifies Solidity version 0.8.20, which defaults to the Shanghai EVM version. This EVM version introduces PUSH0 opcodes, which may not be supported by all chains, especially Layer 2 (L2) chains. Deploying contracts containing PUSH0 opcodes on such chains can lead to deployment failures.
Location / Source File	Market.sol
Observation	<p>The use of Solidity version 0.8.20 by default targets the Shanghai EVM version, which includes PUSH0 opcodes.</p> <p>Some chains, particularly Layer 2 chains, may not support PUSH0 opcodes, potentially causing deployment failures for contracts targeting the Shanghai EVM version.</p>
Remediation	<p>Consider selecting a specific EVM version that is compatible with the target chain when defining the pragma directive. For example, specify <code>pragma solidity 0.8.20;</code> to target the Shanghai version explicitly.</p> <p>Ensure compatibility with the deployment environment by testing contracts on the intended chain or using compiler options to specify the desired EVM version.</p>

	Audit and revise access control mechanisms to ensure proper segregation of duties and reduce centralization risks.
Reference	NA

4.9 Use of Assembly in _getOwnStorage Function

Result	Informational
Description	The function Market._getOwnStorage in the Market.sol contract uses assembly.
Location / Source File	BaseBlastBotSwapAndFeesHandler.sol
Observation	Using assembly can make the code harder to understand and maintain, and it introduces potential risks if not implemented correctly. Assembly code is also less portable and may not be compatible with certain compiler optimizations or future upgrades.
Remediation	Consider refactoring the assembly code into higher-level Solidity code where possible, as it may improve readability, maintainability, and compatibility with future updates.
Reference	https://github.com/crytic/slither/wiki/Detector-Documentation#division-followed-by-multiplication

4.10 Potential Loss of Precision in settleCancelled Function

Result	Informational
Description	The function Market.settleCancelled in the file Market.sol performs a multiplication on the result of a division.
Location / Source File	Market.sol
Observation	The expression <code>settleFee = (collateralToDeduct * \$.config.feeSettle) / WEI6</code> and <code>(success2) = \$.config.feeWallet.call{value: settleFee * 2}()</code> indicate potential issues where the result of a division is multiplied without considering possible loss of precision.
Remediation	Ensure proper handling of division and multiplication operations to avoid loss of precision. Consider using appropriate data types or scaling factors to maintain accuracy.
Reference	NA

4.11 Solidity Pragma Specificity Issue

Result	Informational
Description	The pragma directive in the contract 'Market.sol' at line 2 specifies a wide range of Solidity versions using the caret (^) symbol. This can potentially introduce compatibility issues with future compiler versions, as it allows any version of Solidity greater than or equal to 0.8.20, including major updates that may introduce breaking changes.
Location / Source File	Market.sol

Observation	The use of <code>^</code> in the pragma directive allows for flexibility but can lead to unexpected behavior if there are breaking changes in future compiler versions.
Remediation	Replace the pragma directive with a specific Solidity version, such as <code>pragma solidity 0.8.20;</code>
Reference	NA

4.12 Functions Usage and Visibility Issue

Result	Informational
Description	The functions <code>pause</code> , <code>unpause</code> , <code>settleFilleeds</code> , and <code>settleCancelleds</code> in the <code>Market.sol</code> contract are defined as <code>public</code> , but they are not used externally. If these functions are intended for internal use only, marking them as <code>external</code> or <code>internal</code> can enhance code clarity and potentially reduce gas costs.
Location / Source File	Market.sol (Line: 161 Line: 165 Line: 430 Line: 436)
Observation	<p>The functions are defined as <code>public</code> but are not called externally or by other contracts.</p> <p>Marking them as <code>external</code> or <code>internal</code> explicitly communicates their intended usage and reduces the contract's surface area for potential attacks.</p>
Remediation	<p>Determine if the functions <code>pause</code>, <code>unpause</code>, <code>settleFilleeds</code>, and <code>settleCancelleds</code> are meant for internal use only.</p> <p>If so, mark them as <code>internal</code> to restrict their visibility within the contract. If they are intended for use by derived contracts, mark them as <code>external</code>.</p> <p>If the functions are indeed meant to be called externally, ensure they are utilized in the contract's external interface or consider removing them if they are unnecessary.</p>
Reference	NA

4.13 Usage of Constants Instead of Literals

Result	Informational
Description	The contract <code>Market.sol</code> utilizes literals instead of constants in various places. Using constants enhances code readability, reduces the risk of errors, and makes it easier to adjust values in the future.
Location / Source File	<p>Market.sol Line: 132</p> <ul style="list-style-type: none"> Line: 133 Line: 134 Line: 171 Line: 390 Line: 411 Line: 412 Line: 455

	<ul style="list-style-type: none"> Line: 456
Observation	<p>Literals such as WEI6 / 200, WEI6 / 40000, 10000, 1 days, 100, and WEI6 / 100 are used throughout the contract.</p> <p>These literals represent fixed values, fees, or conditions.</p>
Remediation	<p>Define meaningful constants for these values at the beginning of the contract or in a separate configuration file.</p> <p>Replace the literals with the defined constants to improve code readability and maintainability.</p> <p>Ensure that the names of the constants are descriptive and accurately represent their purpose in the contract.</p>
Reference	NA

4.14 Events Missing indexed Fields

Result	Informational
Description	<p>The events in the <code>Market.sol</code> contract do not specify <code>indexed</code> fields for parameters. Adding <code>indexed</code> fields to events can improve the accessibility of event data for off-chain tools that parse events. However, it's essential to consider the gas cost associated with indexing fields, especially if there are many indexed fields or if gas usage is a concern.</p>
Location / Source File	<p>Market.sol</p> <ul style="list-style-type: none"> Line: 83 Line: 84 Line: 93 Line: 94 Line: 96 Line: 97 Line: 107 Line: 120 Line: 121 Line: 122

Observation	<p>All events listed lack indexed fields for their parameters.</p> <p>Indexed fields can make event data more easily accessible and filterable off-chain.</p>
Remediation	<p>Determine which parameters of each event are most relevant for filtering and accessibility.</p> <p>Add the indexed keyword to those parameters in the event declarations.</p> <p>Carefully consider the gas cost associated with indexing fields, especially if the events are emitted frequently or if gas usage is a concern.</p>
Reference	NA

5.0 Auditing Approach and Methodologies Applied

The solidity smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- **Code quality and structure:** We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analyzing the overall architecture of the solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities:** Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.
- **Documentation and comments:** Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behavior and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices:** We checked that the code follows best practices and coding standards that are recommended by the solidity community and industry experts. This ensures that the solidity smart contract is secure, reliable, and efficient.

Our audit team followed OWASP and Ethereum(Solidity) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract security and performance.

Throughout the audit of the smart contract, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behavior. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimized for performance.

5.1 Code Review / Manual Analysis

Our team conducted a manual analysis of the Solidity smart contracts to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analyzed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the smart contract was secure and reliable.

5.2 Tools Used for Audit

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Adreyn, Slither. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. Entersoft takes pride in utilizing these tools, which significantly contribute to the quality, security, and maintainability of our codebase.

6.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of Gibble Project and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Gibble solidity smart contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of Gibble and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Gibble governs the disclosure of this report to all other parties including product vendors and suppliers.