Bitconnect

Interim Smart Contract Audit Report

DATE: 28 February 2024 | **PREPARED BY:** Entersoft Pte Ltd

Contents

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	28 February 2024	R. Dixit G. Singh D. Bhavar M. Gowda	Interim report

Entersoft was commissioned by Bitconnect (hereinafter referred to as "Bitconnect") to perform a Security Audit on their smart contracts. The review was conducted from 20 February 2024 to 28 February 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:

Bitconnect program-related solidity files.

- BitDexFactory.sol
- BitDexPair.sol
- BitDexRouter.sol
- BitConnect.sol
- BitLock.sol
- BitSend.sol
- BitVault.sol
- BitconnectVesting.sol
- FeeManager.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 Bitconnect smart contract project from 20th February 2024 to 28th February 2024, with a primary focus on Solidity code files integral to blockchain functionality, emphasizing vulnerabilities in associated gas claiming and Burn functionalities. The working of basic functionalities was also tested during the review.

2.2 Scope

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

Commits under scope: cc5dc2fabc2e660b2180a2f5d1a752aa13db73cf,
e8ba26f9f0ec31a9e37e3798b7a5c1c0fc581e87,

Files in Examination:

1. Contract07:
    a. BitDexFactory.sol
    b. BitDexPair.sol
    c. BitDexRouter.sol

2. Contract08
    a. Bitconnect.sol
    b. BitLock.sol
    c. BitSend.sol
    d. BitVault.sol
    e. BitconnectVesting.sol
    f. FeeManager.sol

OUT-OF-SCOPE: External Solidity smart contract, other imported smart contracts.

2.3 Project Summary

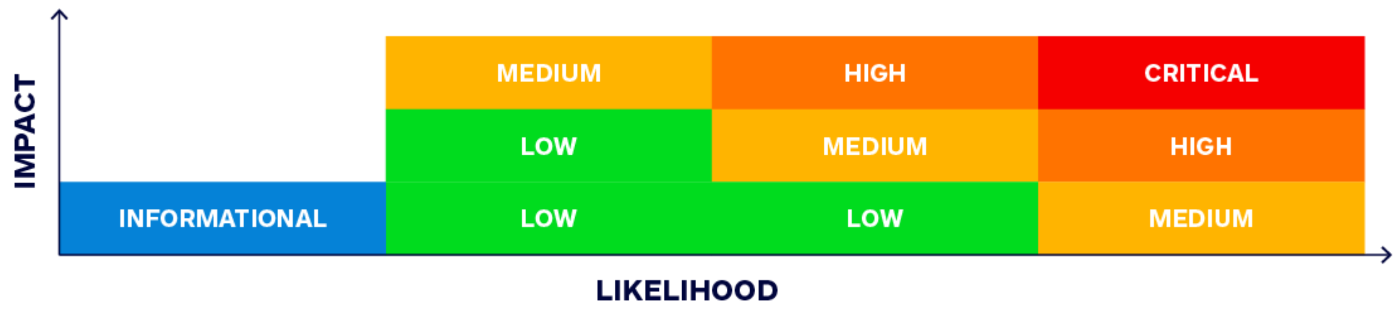
Name	Verified	Audited	Vulnerabilities
Bitconnect	Yes	Yes	Please review Section 4

2.4 Audit Summary

Delivery Date	Method of Audit	Team Members Engaged
28 February 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. **Dynamic Analysis:** The Dynamic audit of a smart contract involves analysing and testing the contract's code by executing it in a controlled environment to identify vulnerabilities, anomalies, and other security or quality issues that might not be evident through static analysis alone. Dynamic analysis involves interacting with the contract to observe its behavior and response to various inputs and conditions (we write both negative and positive test cases to test edge scenarios). This approach is crucial for uncovering security flaws that only occur during execution. This can include runtime operations, contract logic under specific conditions, and interactions with other contracts or external calls, etc.	3

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
1	1	4	2	4

3.0 Executive Summary

Entersoft has conducted a comprehensive technical audit of the Bitconnect smart contract through a comprehensive smart contract audit approach. The primary objective was to identify potential vulnerabilities and security 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. Solidity's renowned security practices were complemented by tools such as Solhint for linting, and the Solidity compiler for code optimization. Sol-profiler, Sol-coverage, and Sol-sec were employed to ensure code readability and eliminate unnecessary dependencies.

Findings and Security Posture:

Our primary focus was on Access Control Policies, Reentrancy, Default Visibility, Outdated Compiler Version, Input Validation,DoS possibilities, Shadowing State Variables, Presence of Unused Variables, Overflow and Underflow Conditions, Assets Integrity, Errors and Exceptions, Re-initialization with Cross-instance Confusion, and Casting Truncation.

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 20, 2023, to February 28, 2024, our team diligently assessed and validated the security posture of the solidity smart contract, ultimately finding a number of vulnerabilities as per vulnerability summary table.

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**
- X **Security vulnerabilities were identified**

#	Bitconnect Smart Contract Audit Attack Vectors	Result
1	Access Control Policies	✔
2	Transaction Signature Validations	✔
3	Default Visibility	✔
4	Address Validation	x
5	Reentrancy attacks	x
6	Presence of unused variables	✔
7	Overflow and Underflow Conditions	✔
8	Assets Integrity	✔
9	Gas claiming related issues	x
10	Program Validation	x
11	DoS	x
12	Time Manipulation Vulnerability	x
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 Gas Claiming Vulnerability in BitLock.sol

Result	**FAIL**
Severity	**CRITICAL**
Description	The auto gas-claiming mechanism in BitLock.sol lacks validation to verify the success of the low-level call to the Blast contract. This oversight was discovered during testing, where it was observed that gas claiming was not occurring as expected due to the absence of validation. Subsequent validation efforts revealed that the call was failing with the reason: "Transaction ran out of gas."
Location / Source File	BitLock.sol
Observation	The lack of validation for the success of the low-level call to the Blast contract undermines the reliability and effectiveness of the gas claiming mechanism in BitLock.sol.
Remediation	Implement a thorough validation mechanism to confirm the success of the low-level call to the Blast contract. This can include checking the return value or using appropriate error handling to ensure reliable gas claiming. Additionally, conduct comprehensive testing to verify the effectiveness of the mitigation and ensure the stability of the gas claiming functionality.
Remark	fixed in commit : a218c4f5a3b55a8b9c7f44fc35c1b2e32d3c0465

4.2 Potential Reentrancy Vulnerability in `disperseEther` Function

Result	**FAIL**
Severity	**HIGH**
Description	The disperseEther function within the smart contract is susceptible to a potential reentrancy vulnerability. This vulnerability arises when the function transfers Ether to a recipient address without adequate safeguards against reentrancy attacks. Specifically, if the recipient address corresponds to a malicious contract with a receive function, the contract can exploit the reentrancy vulnerability by calling back into the `disperseEther` function during the external call, thereby potentially executing additional transfers and manipulating account balances beyond the expected amount.
Location / Source File	Bitsend.sol
Observation	Unauthorized withdrawal of Ether from the contract, leading to potential financial losses.
Remediation	Implement Reentrancy Guard or validate recipient address: Introduce a modifier or state variable that prevents reentrant calls to the disperseEther function. This can help block recursive calls and mitigate the risk of unauthorized Ether transfers.
Reference	https://owasp.org/www-project-smart-contract-top-10/2023/en/src/SC01-reentrancy-attacks.html
Remark	Fixed in commit : e3f4a3b2e0cb2f2c5cfa393be8b3e9df0a1f4cf1

4.3 Lack of Validation for External Transfer/transferFrom Calls

Result	**FAIL**
Severity	**Medium**
Description	The current implementation lacks validation for the return value of external transfer/transferFrom calls. Specifically, the transferFrom function call does not verify whether the transfer was successful or not. Consequently, failed transfers may go unnoticed, leading to potential incorrect assumptions about transaction success.
Location / Source File	BitDexRouter.removeLiquidity(address,address,uint256,uint256,uint256,address,uint256) (contracts07/blast/bitdex/periphery/BitDexRouter.sol#134-150) (contracts07/blast/bitdex/periphery/BitDexRouter.sol#144)
Observation	The codebase does not include proper validation for the return value of transferFrom calls, potentially leaving the system vulnerable to undetected transfer failures.
Remediation	Implement proper validation of the return value for external transfers or transferFrom calls. This involves checking whether the transfer was successful and handling failures appropriately, such as reverting the transaction or notifying the user about the failure. Additionally, conduct comprehensive testing to ensure the effectiveness of the mitigation strategy.
Remark	

4.4 Potential Denial of Service (DoS) Vulnerability in disperseEther Function

Result	**FAIL**
Severity	**Medium**
Description	The current function iterates through the recipients array to transfer ether to each recipient without checking or limiting the array size. In case of a very large recipients array, potentially stemming from a malicious transaction, the function may consume an excessive amount of gas, surpassing the block gas limit. If the gas consumption exceeds the block gas limit, the transaction fails to be mined, resulting in denial of service to the sender and possible waste of gas fees.
Location / Source File	Bitsend.sol
Observation	The token transfer function lacks array size checks, risking denial-of-service attacks by exceeding gas limits; implementing array size limitations or batch processing is crucial for security.
Remediation	To mitigate this vulnerability, it's advisable to implement measures to limit the size of arrays processed by the function. Alternatively, batch processing transactions can be employed to distribute the workload across multiple transactions, reducing the risk of exceeding the block gas limit. Implementing either of these recommendations can help prevent DoS attacks and ensure the reliability and efficiency of the token transfer function.
Remark	Fixed in commit : 4ebd5954a423fdb56e557f679a6bb84586eb974c

4.5 Denial of Service Vulnerability in disperseToken Function

Result	**FAIL**
Severity	**Medium**
Description	There exists a potential Denial of Service (DoS) vulnerability in the disperseToken function if a user submits an excessively large array of recipients. The function does not currently enforce any limits on the size of the recipient array. As a result, if a malicious user sends a large array of recipients, it could lead to excessive gas consumption, potentially causing the transaction to fail due to exceeding the block gas limit.
Location / Source File	Bitsend.sol
Observation	The "disperseToken" function lacks array size limits, exposing it to potential DoS attacks through excessive gas consumption. Implementing array size constraints or batch processing can mitigate this vulnerability, enhancing system reliability and efficiency.
Remediation	To mitigate this vulnerability, it is recommended to implement measures to limit the size of arrays accepted by the disperseToken function. Alternatively, batch processing of transactions can be implemented to distribute the workload across multiple transactions, reducing the risk of exceeding the block gas limit. By implementing these recommendations, the system can better defend against DoS attacks and ensure the reliability and efficiency of the disperseToken function.
Remark	Fixed in commit : 4ebd5954a423fdb56e557f679a6bb84586eb974c

4.6 Lack of Validation for TransferFrom Return Value

Result	**FAIL**
Severity	**Medium**
Description	In the deposit function and Withdraw function of Bitvault.sol, there is a lack of validation for the return value of the transferFrom operation. The functions does not verify whether the transferFrom call is successful or not. Consequently, if the transfer fails, the functions does not revert, allowing an attacker to call deposit without paying for the transaction. Additionally, some tokens may not revert in case of transfer failure and instead return false.
Location / Source File	Bitvault.sol
Observation	Bitvault.sol's deposit and withdraw functions lack validation for transferFrom's return value, enabling potential exploit without transaction payment. Implement proper validation to enhance security.
Remediation	To address this vulnerability, it is recommended to implement proper validation for the return value of the transferFrom operation within the functions. This can be achieved by either using safeTransfer and safeWithdraw or checking the return value to ensure the success of the transfer/withdraw. Implementing these measures will help prevent attackers from exploiting the functions without paying for the transaction and enhance the security of the Bitvault.sol contract. Additionally, thorough testing should be conducted to verify the effectiveness of the mitigation strategy.
Remark	Fixed in commit : 98680961ed9e754311c334d94fbe70c1a750464f

4.7 Unsafe Use of block.timestamp

Result	**FAIL**
Severity	**Low**
Description	In Bitvault.sol, the function getUserWeightedTokenSeconds utilizes block.timestamp to calculate user weighted token seconds. However, this approach poses a risk as block.timestamp can be manipulated by miners to some extent. Relying solely on block.timestamp for such calculations may not be optimal, potentially leading to inaccuracies in the user weighted token seconds.
Location / Source File	Bitvault.sol
Observation	Bitvault.sol's getUserWeightedTokenSeconds function utilizes block.timestamp for calculations, which is susceptible to miner manipulation. To enhance accuracy, consider using alternative time sources like external timestamp oracles. Thorough testing is advised to validate this mitigation.
Remediation	To mitigate this vulnerability, it is advisable to avoid relying solely on block.timestamp for critical calculations like user weighted token seconds. Instead, alternative sources of time, such as external timestamp oracles, can be used to obtain more reliable and tamper-resistant timestamps. Implementing this mitigation will help enhance the accuracy and integrity of user rewards in Bitvault.sol. Additionally, thorough testing should be conducted to validate the effectiveness of the alternative time source.
Reference	N/A

4.8 Lack of Zero-Checks in BitDexFactory Contracts

Result	**Informational**
Severity	**Low**
Description	Uninitialized or zero addresses could be assigned to critical variables such as _feeTo, _feeToSetter, _feeManager, _gasFeeTo, _factory, _WETH, and router addresses, potentially leading to unexpected behavior, unauthorized access, or compromise of contract control and ownership.
Location / Source File	BitDexFactory.sol
Observation	The contracts lack zero-checks in constructors and relevant setter functions, allowing the possibility of zero addresses being assigned to critical variables.
Remediation	Implement zero-checks in constructors and relevant setter functions to prevent assignment of zero addresses to these variables, thereby enhancing contract robustness and security against potential exploits and vulnerabilities.
Remark	Fixed in commit :3bad33cc2f02d06ded0ebcdc5b65bffacd74406f

4.9 Missing Event Emission in BitDexRouter Functions

Result	**Informational**
Description	These functions in the BitDexRouter contract fail to emit events for specific state changes as recommended by Slither static analysis.
Location / Source File	BitDexRouter.sol
Observation	The setGasFeeTo(address) and setFeeManager(address) functions do not emit events when gas fee recipient or fee manager addresses are updated, hindering transparency and auditability.
Remediation	Modify the contract to emit events for changes, enhancing transparency and auditability. Ensure thorough testing to avoid unintended side effects.
Remark	Fixed in commit: 12a9a1571f06074f7ced3dff97a4256e170aa90f

4.10 Lack of Initial Check for Existing Authorization

Result	**Informational**
Description	In the disperseFees function of FeeManager.sol, there is a need to limit and fix the lengths of fee outputs for security reasons. Currently, the lengths of fee outputs are not constrained, which could potentially expose the system to Denial of Service (DoS) attacks. By restricting and fixing the lengths of fee outputs, the vulnerability can be mitigated to prevent possible DoS scenarios.
Location / Source File	FeeManager.sol
Observation	Implement constraints on fee output lengths in FeeManager.sol's disperseFees function to mitigate DoS vulnerabilities, ensuring system stability and availability. Thorough testing is vital to validate the effectiveness of this mitigation.
Remediation	To address this vulnerability, it is recommended to implement constraints on the lengths of fee outputs within the disperseFees function. By limiting and fixing the lengths of fee outputs, the system can prevent potential DoS attacks and maintain its availability and stability. Additionally, thorough testing should be conducted to ensure that the mitigation effectively addresses the security concerns without introducing new vulnerabilities.
Remark	Fixed in commit: 1de32e0848f0426f66fc42870d724afafe2ed21b

4.11 Duplicate Contract Names in Codebase

Result	**Informational**
Description	In a codebase where two contracts have similar names, the compilation artifacts will only contain one of the contracts with the duplicate name. This occurs because the compilation process overwrites the previously compiled contract with the same name, resulting in only one version of the contract being included in the artifacts.
Location / Source File	IERC20
Observation	Duplicate contract names in a codebase risk loss of functionality in compilation artifacts. Mitigate by ensuring unique names and fostering clear communication among developers.
Remediation	To mitigate this issue, it is essential to ensure that each contract in the codebase has a unique and distinguishable name. This can be achieved by renaming one of the contracts to eliminate the duplicate name. Additionally, clear documentation and communication among developers can help prevent confusion regarding the presence of duplicate contract names and ensure that all relevant contracts are included in the compilation artifacts. Regular code reviews and testing can also help identify and address any issues related to duplicate contract names early in the development process.
Reference	N/A

4.12 Lack of Zero-Checks in BitDexRouter Contracts

Result	**Informational**
Description	Uninitialized or zero addresses could be assigned to critical variables such as _feeTo, _feeToSetter, _feeManager, _gasFeeTo, _factory, _WETH, and router addresses, potentially leading to unexpected behavior, unauthorized access, or compromise of contract control and ownership.
Location / Source File	BitDexRouter.sol
Observation	The contracts lack zero-checks in constructors and relevant setter functions, allowing the possibility of zero addresses being assigned to critical variables..
Remediation	Implement zero-checks in constructors and relevant setter functions to prevent the assignment of zero addresses to these variables, thereby enhancing contract robustness and security against potential exploits and vulnerabilities.
Remark	Fixed in commit:3bad33cc2f02d06ded0ebcdc5b65bffacd74406f

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 contracts, 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 Slither, ethlint, and testing using hardhat. 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 BitConnect 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 Bitconnect 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 Bitconnect and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Bitconnect governs the disclosure of this report to all other parties including product vendors and suppliers.