BlockApex

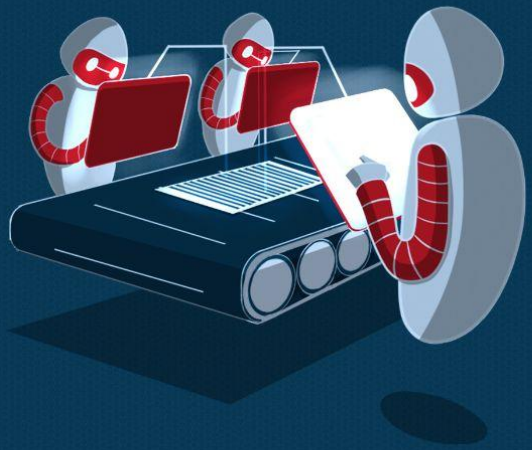
SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



Powered by XORD

PREFACE

Objectives

The purpose of this document is to highlight any identified bugs/issues in the provided codebase. This audit has been conducted in a closed and secure environment, free from influence or bias of any sort. This document may contain confidential information about IT systems/architecture and the intellectual property of the client. It also contains information about potential risks and the processes involved in mitigating/exploiting the risks mentioned below. The usage of information provided in this report is limited, internally, to the client. However, this report can be disclosed publicly with the intention to aid our growing blockchain community; at the discretion of the client.

Key understandings

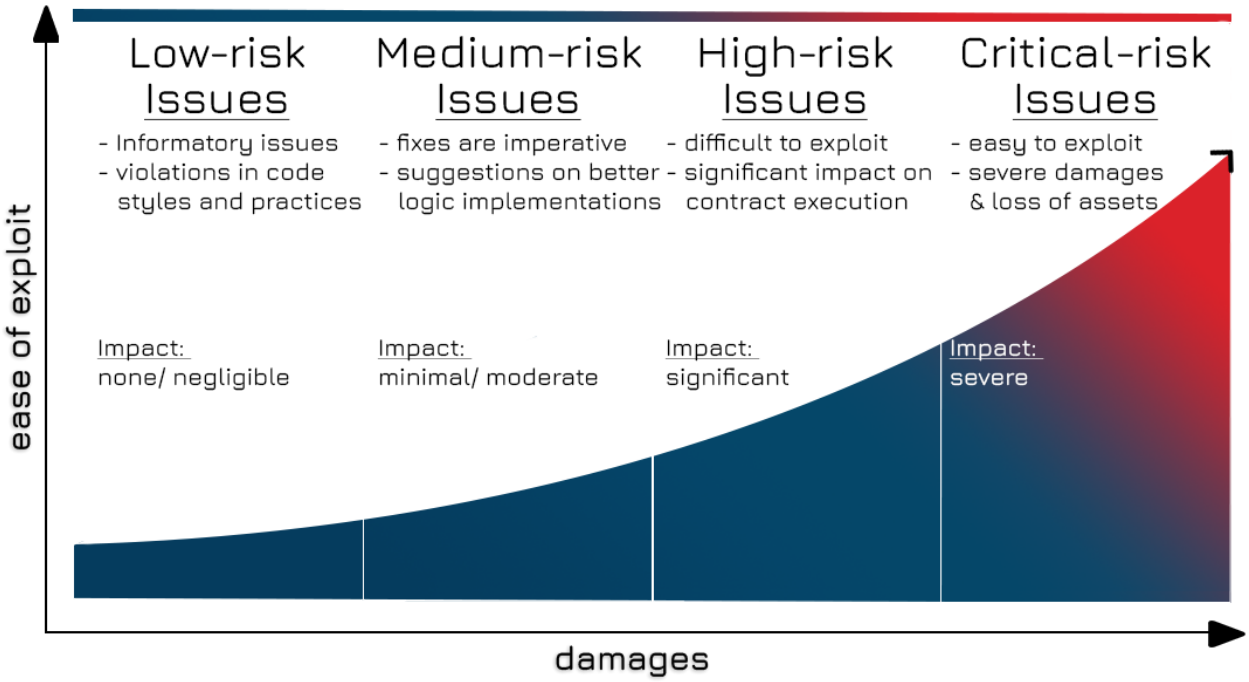


TABLE OF CONTENTS

INTRODUCTION

BlockApex (Auditor) was contracted by ScriptTV (Client) for the purpose of conducting a Smart Contract Audit/ Code Review. This document presents the findings of our analysis which started on 30th Jan '2023

Name
ScriptTV
Auditor
BlockApex
Platform
Ethereum | Solidity
Type of review
Manual Code Review | Automated Tools Analysis
Methods
Architecture Review | Functional Testing | Computer-Aided Verification | Manual Review
Git repository/ Commit Hash
Private Repo | 6cf8da8bb236cbd9cbd2834a9d280e2164fd577f
White paper/ Documentation
https://whitepaper.script.tv
Document log
Initial Audit Completed: Feb 6, 2023
Final Audit Completed: Mar 6, 2023

Scope

The git-repository shared was checked for common code violations along with vulnerability-specific probing to detect major issues/vulnerabilities. Some specific checks are as follows:

Code review		Functional review
Reentrancy	Unchecked external call	Business Logics Review
Ownership Takeover	Fungible token violations	Functionality Checks
Timestamp Dependence	Unchecked math	Access Control & Authorization
Gas Limit and Loops	Unsafe type inference	Escrow manipulation
DoS with (Unexpected) Throw	Implicit visibility level	Token Supply manipulation
DoS with Block Gas Limit	Deployment Consistency	Asset's integrity
Transaction-Ordering Dependence	Repository Consistency	User Balances manipulation
Style guide violation	Data Consistency	Kill-Switch Mechanism
Costly Loop		Operation Trails & Event Generation

Project Overview

Script.TV is a decentralized video delivery network that furnishes an expansive range of blockchain-enabled solutions to the problems related to the traditional video-streaming sector. The platform offers high-quality video streaming as well as multiple incentive mechanisms for decentralized bandwidth and content-sharing at a reduced cost as compared to conventional service providers. Script.TV offers an online TV platform in which both users and content publishers earn valuable tokens through video streaming. A user-first watch-2-earn solution that allows users to earn rewards on- and off-chain by upgrading their ScriptGlasses NFT.

System Architecture

The workflow of the protocol is that of a Watch to Earn. A user can mint three types of glasses (Common, Rare & SuperScript) by burning $SPAY where each type has two base attributes i.e. Durability and Gem tier. The durability of the glass increases as it levels up (via watching content) and the gem tier increases as gems are integrated with the glass rewarding users with $SPAY.

To onboard users, the protocol will follow a freemium model where 100,000 common glasses will be minted for free. These common glasses will have a fixed allowable watch time and will yield a fixed payout i.e., equivalent to a base common glass with no gems. The free mints phase will operate simultaneously with paid mints. To differentiate the free mint from the paid ones, there are certain milestones that are pegged to watch time enabling a user to modify their glasses and enhance their reward/ unit energy. Users can watch content on the platform for as long as they want, however, there are limits to the daily watch time to earn SPAY tokens.

Recharge vouchers add to the gamification element in the protocol and also serve as an incentivization mechanism for the users to upgrade their levels in order to avail of a discount on their recharge cost. To mint the recharge voucher, the user will have to burn their SPAY tokens. This is an optional burnable event as the user can choose not to purchase the recharge voucher after the eligibility criteria are met.

Methodology & Scope

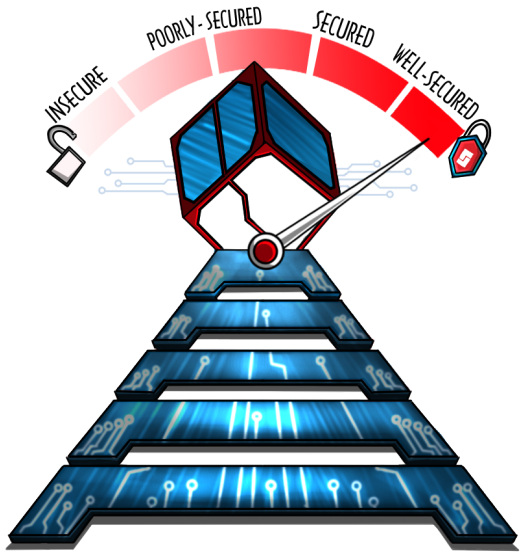
The codebase was audited using a filtered audit technique. A band of three (3) auditors scanned the codebase in an iterative process for a time spanning one (1) week.

Starting with the recon phase, a basic understanding was developed and the auditors worked on developing presumptions for the shared codebase and the relevant documentation/ whitepaper. Furthermore, the audit moved on with the manual code reviews with the motive to find logical flaws in the codebase complemented with code optimizations, software, and security design patterns, code styles, best practices, and identifying false positives that were detected by automated analysis tools.

AUDIT REPORT

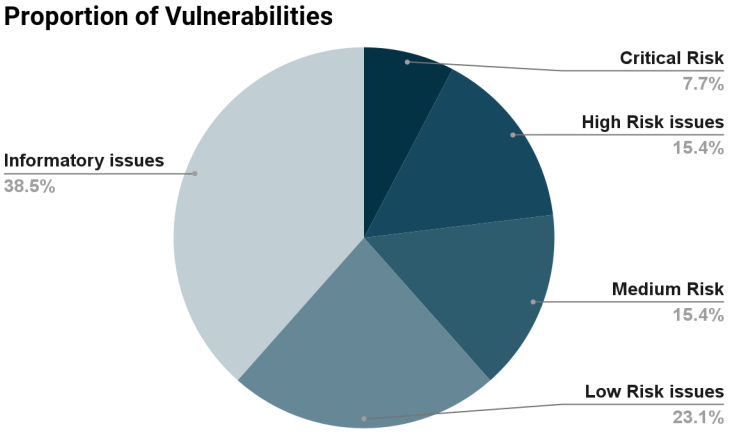
Executive Summary

Our team performed a technique called "Filtered Audit", where the contract was separately audited by four individuals. After a thorough and rigorous process of manual testing, an automated review was carried out using Slither & Mythril for static analysis and Foundry for fuzzing invariants. All the flags raised were manually reviewed and re-tested to identify the false positives.



Our team found:

#Issues	Severity Level
1	Critical Risk issue(s)
1	High Risk issue(s)
2	Medium Risk issue(s)
3	Low Risk issue(s)
5	Informatory issue(s)



Key Findings

#	Findings	Risk	Status
1	Incomplete Voucher Release	Critical	Fixed
2	Overly Centralized Functionalities	High	Fixed
3	Misconfigured Treasury Allocation	Medium	Fixed
4	Inapplicability of Checks-Effects-Interactions Pattern	Medium	Fixed
5	Redundant Duplicate Constraints	Low	Fixed
6	Emit events for important operations	Low	Fixed
7	Insufficient and Inadequate testing	Low	Fixed
8	Inconsistent Parameter Order in ScriptVoucher.mint	Informational	Fixed
9	Unstructured Enum Placement	Informational	Fixed
11	Optimized Enum Indexing for Glass and Voucher Types	Informational	Fixed
11	Improper Event Emissions in Contract Functions	Informational	Fixed
12	Optimization of Events' Naming Conventions	Informational	Fixed

Detailed Overview

Critical-risk issues

ID	1
Title	Incomplete Voucher Release
Path	contracts/ScriptTV.sol
Function Name	associateVoucher, releaseVoucher

Description: The issue of ungraceful handling of voucher association against an unowned _glassID highlights a flaw in the voucher release mechanism. The associatedVouchers mapping is maintained using the msg.sender as the key and the glass ID as the value. When the releaseVoucher function is called, it sets the value in the associatedVouchers mapping for the key msg.sender and the glass ID to false, indicating that the voucher has been released. Since the protocol does not restrict users to associate multiple vouchers to a glass when multiple vouchers have been associated with the same glass and one voucher has been released, the associatedVouchers mapping for that glass will be set to false, leading to the code to revert an AssetNotOwned error when attempting to release another voucher for that glass. This results in an incomplete voucher release and the inability to release the rest of the vouchers for that glass.

Recommendation: Place a constraint in the associateVoucher function to always first check whether the _glassId being provided as a function argument does not have a voucher associated against the msg.sender.

High-risk issues

ID	2
Title	Overly Centralized Functionalities
Path	contracts/ScriptTV.sol
Function Name	rechargeGlasses, earningPoolReward, earningPayout

Description: The issue of signature malleability in the case of a compromised owner is a major security concern as it highlights the overly centralized nature of the system's functionalities. The script.TV owner is set during deployment, and a compromised owner could exploit this by equipping gems on the freemium token and removing unlimited SPAY from the earning pool reward and payout since these actions are unchecked. This level of centralization puts the whole protocol at risk, as the compromise of a single owner could bring down the entire system.

Recommendation: To mitigate this risk, it is necessary to implement proper checks and security measures and reduce the number of centralized functionalities in the system. Where a valid data structure is designed to handle

Medium-risk issues

ID	3
Title	Misconfigured Treasury Allocation
Path	contracts/ScriptTV.sol
Function Name	setTreasuryPercentage

Description: This issue of poorly sanitized treasury configurations highlights a potential flaw in the way funds are being managed within the protocol. An admin with the ability to set the fee can set it to 100, leading to the situation where all user funds are directed into the protocol fee. Furthermore, if the treasuryPercentage is set to 100, it is implied that 100% of the user's payout/ reward amount passed to the earningPayout function will be sent to the treasury, and none will be sent to the sender. This misconfigured Treasury allocation could result in users not receiving any payouts, which could harm the reputation and sustainability of the project. It is important to thoroughly consider and properly configure the Treasury settings to align with the intended purpose and goals of the project, and to ensure fairness to all parties involved.

Code:

```
/**
 * @notice change the percentage of treasury cut
 * @param _newPercentage new percentage for treasury cut
 */
function setTreasuryPercentage(uint256 _newPercentage) external onlyOwner {
  uint256 oldPercentage = treasuryPercentage;

  if (_newPercentage < 0 || _newPercentage > 100 ether) {
    revert PercentageOutOfRange();
  }
  if (_newPercentage == oldPercentage) {
    revert SameAsOld();
  }
```

```
    treasuryPercentage = _newPercentage;

    emit PercentageUpdated(oldPercentage, _newPercentage);
}
```

Recommendation: Place proper and sensible upper and lower limits for the percentage of the user's total earnings which the treasury is eligible for in any and all cases.

ID	4
Title	Inapplicability of Checks-Effects-Interactions Pattern
Path	contracts/**.sol, contracts/base/**.sol
Function Name	Listed in the description

Description: The check-effects-interactions pattern being the first line of defense ensures that checks and validations are performed before any changes or interactions are made with external contracts, reducing the risk of reentrancy in case other contracts are compromised. Failing to properly enforce this pattern can result in unintended consequences, such as potential reentrancy and security breaches. The following of the smart contracts code base is vulnerable to all the attack surfaces caused due to missing implementation of the design pattern:

- ScriptVoucher
  - mint
- ScriptTV
  - mintFreemiumGlasses
  - mintGlasses
  - releaseVoucher
  - associateVoucher
- ScriptGlasses
  - safeMint

Recommendation: To avoid these risks, it is essential to properly implement and enforce the checks-effects-interactions pattern in the system to ensure the safe and secure handling of external calls.

Low-risk issues

ID	5
Title	Redundant Duplicate Constraints
Path	contracts/ScriptTV.sol
Function Name	setTreasuryPercentage, setTreasury

Description: The implementation of the setTreasuryPercentage function includes a check to ensure that the provided uint value is never less than 0. This check is unnecessary, as the uint data type in solidity can only contain non-negative values. The inclusion of this check adds unnecessary complexity to the code and can lead to confusion for developers. Additionally, setTreasury checks for function argument _newTreasury to never be a zero address in two different ways as attached below.

Code:

```
function setTreasuryPercentage(uint256 _newPercentage) external onlyOwner {

  if (_newPercentage < 0 || _newPercentage > 100 ether) {
    revert PercentageOutOfRange();
  }

function setTreasury(address _newTreasury) external onlyOwner {
  address oldTreasury = treasury;

   require(_newTreasury != address(0), "New Treasury is the zero address");
   if (_newTreasury == address(0)) {
    revert ZeroAddress();
  }
```

Recommendation: To improve code readability and maintainability, it is recommended to safely remove the above mentioned redundant checks.

ID	6
Title	Emit events for important operations
Path	contracts/ScriptTV.sol
Function Name	updateVoucherMintPrice, updateVoucherSupply, setRechargeLimitPerDay

Description: In the current implementation of Script.TV smart contracts, significant blockchain write actions are not recorded as logged events which are imperative to the perpetuating history of blockchain transactions.

Recommendation: It is suggested to emit relevant and self explanatory events with appropriate arguments in all user-facing functionalities.

ID	7
Title	Insufficient and Inadequate testing
Path	contracts/ScriptTV.sol
Function Name	**

Description: Throughout the codebase, the testing is insufficient to support the positive and negative test cases from the protocol engineering team. The low level of testing points to a critical lacking in the project as many cases are unexplored.

Recommendation: It is therefore recommended to have a sufficient suite of test cases executed over the production-ready smart contract code to ultimately achieve the satisfaction of all functionality working as expected.

Informational-risk issues

ID	8
Title	Inconsistent Parameter Order in ScriptVoucher.mint
Path	contracts/base/ScriptVoucher.sol
Function Name	mint

Description: The function takes in parameters in a non-standard order, where the first parameter is the type of Voucher, and the second parameter is the address. This may cause confusion and increase the likelihood of errors, as it deviates from the standard parameter order used in similar functions.

Recommendation: It is recommended to change the order of the parameters to be (address, type) to align with industry standards and improve overall code clarity and maintainability.

ID	9
Title	Unstructured Enum Placement
Path	contracts/utils/ScriptNFTType.sol
Function Name	-

Description: In the ScriptTV protocol, there is an incosistency with the enum data structures. The VoucherType enum is located in the file ScriptVoucher.sol instead of being in a dedicated file for enums. This could lead to overlapping and a lack of structure in the directory.

Recommendation: To avoid this, it is recommended to rename the file utils/ScriptNFTType.sol to utils/enums.sol and move the VoucherType enum to the new file. This will enforce a proper structure of the directory and prevent any potential issues in the future.

ID	10
Title	Optimized Enum Indexing for Glass and Voucher Types
Path	contracts/ScriptTV.sol
Function Name	**

Description: To enhance the reliability of the code, it is recommended to utilize a freemium mode for glasses at the last index of the enum definition. This helps ensure that once the available options are exhausted, they will remain unused. The current code blocks make use of a relation between glassType and voucherType, where the glassType is used as +1 and -1 in relation to the voucherType.

Recommendation: To avoid potential arithmetic faults, it is suggested to use a 1-to-1 indexing system for common, rare, and superscript glass and voucher types. This ensures that the enumeration is optimized for improved performance and eliminates the risk of off-by-one issues.

ID	11
Title	Improper Event Emissions in Contract Functions
Path	contracts/ScriptTV.sol
Function Name	rechargeGlasses

Description: This improvement focuses on optimizing the error handling of stack too deep that occurs in events and improving the efficiency of the code by using scoping blocks. This will ensure that the correct data is emitted onto the blockchain in a clear and concise manner. The current code emits data in the format of RechargeGlasses with the variables msg.sender, discountedAmount, _glassId, and _nonce.

Recommendation: This design flaw can be changed to a more relevant set of variables that are emitted in a more user-friendly format, such as GlassesRecharged with the variables msg.sender, _glassId, and discountedAmount. This change will make the data emitted on the blockchain easier to understand and more readable for users.

ID	12
Title	Optimization of Events' Naming Conventions
Path	contracts/ScriptTV.sol
Function Name	**

Description: In order to improve the readability and transparency of actions performed on the blockchain, it is recommended to rename all events to be self-explanatory and have the parameters ordered in a logical manner. For example, instead of "RechargeGlasses" it could be renamed to "GlassesRecharged", and the parameters could be ordered as (msg.sender, _glassId, discountedAmount) instead of (msg.sender, discountedAmount, _glassId, _nonce). This will improve the overall clarity and efficiency of the smart contract

- GlassesConfiscated(...)
- PayoutEarned(...)
- Rename GlassMinted(...) to GlassesMinted(...)

Recommendation: It is highly suggested to follow as above.

DISCLAIMER

The smart contracts provided by the client for audit purposes have been thoroughly analyzed in compliance with the global best practices to date w.r.t cybersecurity vulnerabilities and issues in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token, its sale, or any other aspect of the project.

Crypto assets/tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service, or another asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond Solidity that could present security risks.

This audit cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. 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.