BlockApex

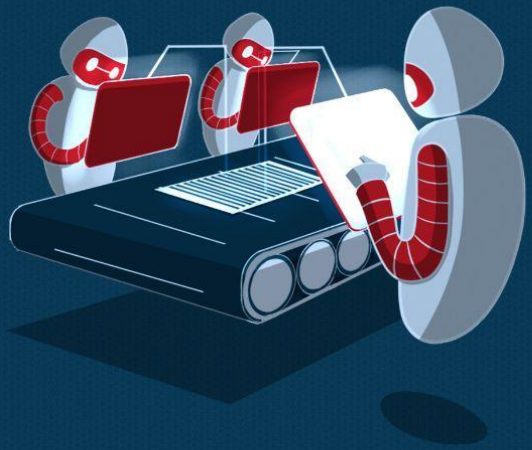
SMART CONTRACT SECURITY ANALYSIS REPORT

```
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

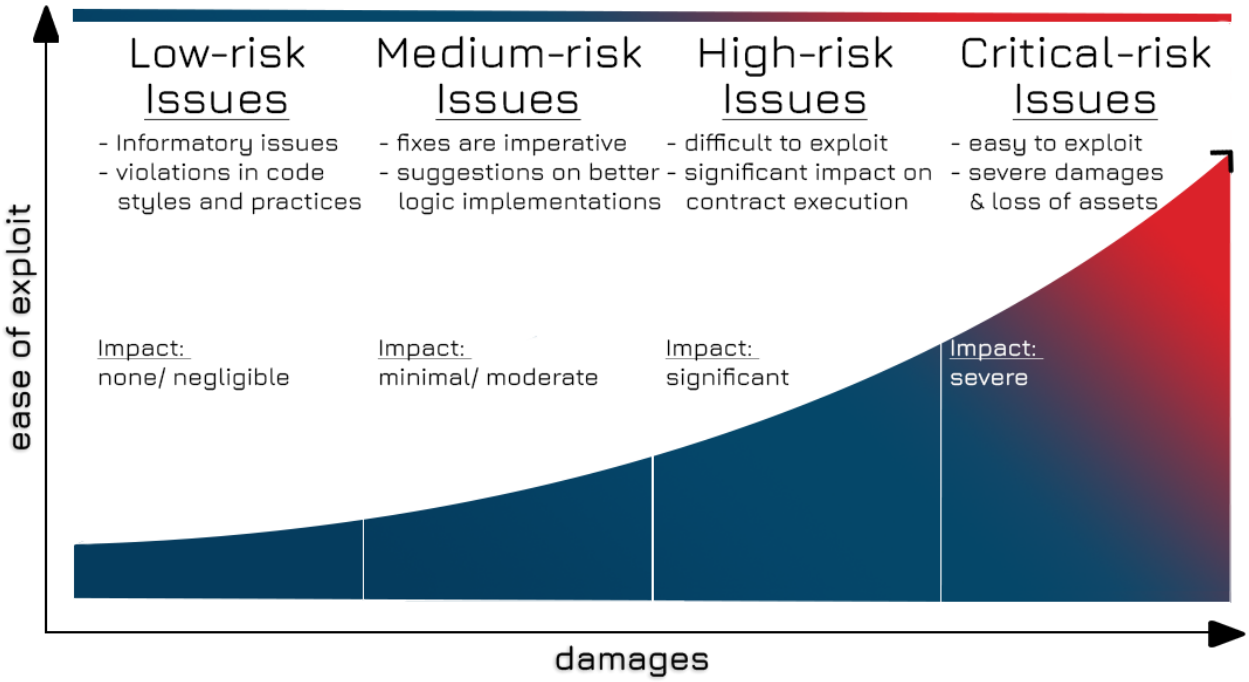


TABLE OF CONTENTS

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 19th June 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 | 7ec3805b262c8cb9dde644318d1f27cf1dc23e31
White paper/ Documentation
https://whitepaper.script.tv
Document log
Initial Audit Completed: 28th June 2023
Final Audit Completed

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. Our expert auditor scanned the codebase in an iterative process for a time spanning of 3 days.

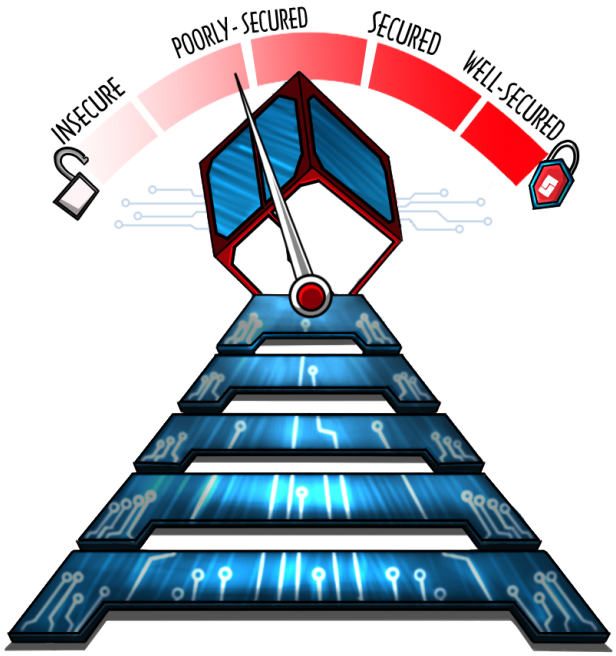
Starting with the recon phase, a basic understanding was developed and the auditors worked on developing presumptions for the shared codebase and the relevant documentation/ whitepaper. Furthermore, the audit moved on with the manual code reviews with the motive to find logical flaws in the codebase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

This is the second iteration of previously-audited ScripTV Smart Contracts complex after revisions from the development team following the recommendations from the first iteration of the smart contract audits in the scope. The changes introduced a new wave of threat vectors which were modeled and applied by the security researcher based on the extended functionality of the system..

AUDIT REPORT

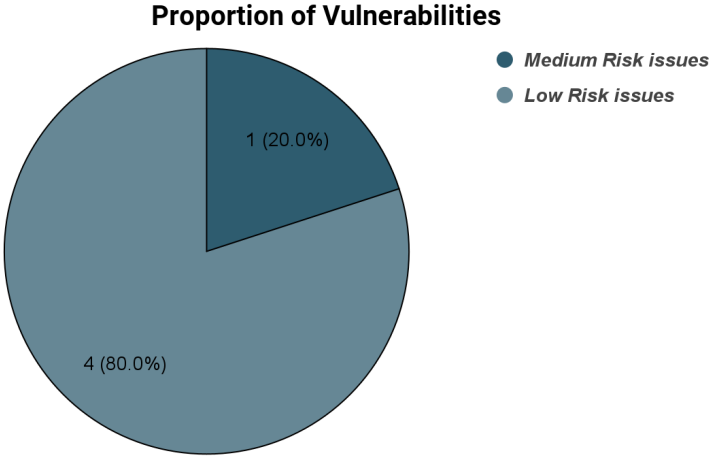
Executive Summary

Our team performed a technique called "Filtered Audit", where the contract was separately audited by one individual. 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
0	Critical Risk issue(s)
0	High Risk issue(s)
1	Medium Risk issue(s)
4	Low Risk issue(s)
0	Informatory issue(s)



Key Findings

#	Findings	Risk	Status
1.	Unutilized Daily Limit Variable in Recharge Function	Medium	Fixed
2.	Equipped gems heavily dependent on server	Low	Fixed
3.	Overly Centralized Functionality	Low	Fixed
4.	Utilize Dormant Functions for Efficient Reward Allocation	Low	Fixed
5.	Enable Users to Invoke the confiscateGlasses Function Themselves	Low	Fixed

Detailed Overview

Medium-risk issues

ID	1
Title	Unutilized Daily Limit Variable in Recharge Function
Path	contracts/ScriptTV.sol
Function Name	rechargeGlasses

Description: The rechargeGlasses function in the above mentioned smart contract code has a hardcoded limit of 3 for the number of times a pair of glasses can be recharged in a day. However, this limit is not inherently linked to the rechargeLimitPerDay variable, which is designed to control the daily recharge limit. As a result, changing the rechargeLimitPerDay using the setRechargeLimitPerDay function does not affect the actual limit enforced by the rechargeGlasses function.

Affected Code:

```
//Recharge limit per day
uint8 public rechargeLimitPerDay;

function rechargeGlasses(
    uint256 _amount,
    uint256 _nonce,
    bytes calldata _sig,
    uint256 _glassId
) external {
    // ... existing code ...

    uint256 _days = (block.timestamp - PROTOCOL_START_TIMESTAMP) / 1 days;
    if (rechargeTimes[_glassId][_days] >= 3) { //here
        revert DailyRechargeLimitExceeds();
    }
```

```
    // ... existing code ...
}

function setRechargeLimitPerDay(uint8 _newLimit) external onlyOwner {
    uint8 oldLimit = rechargeLimitPerDay;
    if (_newLimit < 1) {
      revert LimitIsInvalid();
    }
    if (_newLimit == oldLimit) {
      revert SameAsOld();
    }
    rechargeLimitPerDay = _newLimit;
    emit LimitUpdated(oldLimit, _newLimit);
  }
```

Recommendation: To fix this issue, replace the hardcoded value of 3 in the rechargeGlasses function with the rechargeLimitPerDay variable. This will ensure that the daily recharge limit is correctly enforced according to the value of rechargeLimitPerDay.

Low-risk issues

ID	2
Title	Equipped gems heavily dependent on server
Path	contracts/ScriptTV.sol
Function Name	equipGem

Description: In the equipGem function the functionality allows a user to equip a gem to their glasses, provided they are the owner of the glasses and they have enough SPAY tokens to burn for the cost of the gem. The function also emits an event GemEquipped when a gem is successfully equipped.

However, the contract currently does not keep track of the gems equipped to each pair of glasses. It only emits an event when a gem is equipped. If the event logs get unsynced or lost due to any reason, there would be no way to retrieve the information about which gems are equipped to which glasses.

Recommendation: To address this anomaly of design, consider maintaining a state variable in the contract that keeps track of the gems equipped to each pair of glasses. This can be achieved by a mapping of glass IDs to gem types. This way, even if teh server listening to the event logs gets unsynced, the smart contract state would still hold the correct information about the equipped gems.

Revised Code:

```
mapping(uint256 => uint8) public glassesToGems;

function equipGem(
    uint256 _glassId,
    uint8 _gemType,
    uint256 _nonce,
    bytes calldata _sig
) external {
    // ... existing code ...
```

```
    glassesToGems[_glassId] = _gemType;

    // ... existing code ...
}
```

In the above example, glassesToGems is a public mapping that associates each pair of glasses (identified by _glassId) with the type of gem equipped (_gemType). This information is updated each time a gem is equipped.

Note that this is a simple example and might need to be adapted to fit the specifics to contract needs. For instance, a pair of glasses can have multiple gems equipped, you might need to use a different data structure, like a mapping of glass IDs to an array of gem types.

ID	3
Title	Overly Centralized Functionality
Path	contracts/ScriptTV.sol
Function Name	earningPoolReward, earningPayout

Description: The earningPoolReward function transfers the amount of SPAY tokens to the caller of the function received in the function argument, without actually performing any security validations related to the cycle number or the gem ID.

In all such cases, if the smart contract is supposed to distribute rewards based on certain factors such as the cycle number or the gem ID, then relevant checks ought to be performed to ensure that the functionality is never exposed to malicious inputs. Relying solely on the backend server to provide the correct amount could lead to potential issues that could violate the basic assumptions.

Recommendation: Maintaining a mapping in the contract to store the reward amount for each cycle could be a solution. The smart contract could then use this mapping to determine the correct reward amount, rather than relying on the _amount parameter provided by the caller. This would make the reward distribution process more secure and reliable.

For instance, cycleRewards is a public mapping that associates each cycle number with a reward amount. The earningPoolReward function then uses this mapping to determine the correct reward amount, rather than relying on the _amount parameter provided by the caller.

Revised Code:

```
mapping(uint256 => uint256) public cycleRewards;

function earningPoolReward(
    uint256 _nonce,
    bytes memory _sig, // memory instead of calldata to avoid stack too
deep
    uint256 _cycleNo,
    uint256 _gemId
```

```
) external verifySignature(cycleRewards[_cycleNo], _nonce, _sig,
"POOL_REWARD") {
    spay.transfer(msg.sender, cycleRewards[_cycleNo]);

    emit EarnPoolReward(msg.sender, cycleRewards[_cycleNo], _nonce,
_cycleNo, _gemId);
}
```

ID	4
Title	Utilize Dormant Functions for Efficient Reward Allocation
Path	contracts/ScriptTV.sol
Function Name	earningPoolReward

Description: It appears that the fundCyclesReward function in the ScriptTV smart contract is used to fund a certain number of cycles with a specified reward amount. The getActiveCycle function is used to get the currently active cycle, and the getRewardForCycle function is used to get the reward amount for a given cycle. The setPoolPercentages function is used to set the percentage shares for different tiers.

However, if these functions are not being used in the reward distribution process, then the rewards might not be distributed correctly according to the cycles and the tier percentages or if the rewards are server dependent there is no purpose of these functions inside the contract.

Recommendation: One way to address this issue could be to modify the earningPoolReward function (from the previous Issue) to use the getRewardForCycle function to determine the correct reward amount. The function could also use the getActiveCycle function to ensure that rewards are only distributed for the currently active cycle.

Revised Code:

```
function earningPoolReward(
    uint256 _nonce,
    bytes memory _sig, // memory instead of calldata to avoid stack too deep
    uint256 _gemId
) external verifySignature(cycleRewards[getActiveCycle()], _nonce, _sig,
"POOL_REWARD") {
    uint256 _cycleNo = getActiveCycle();
    (uint256 tier1Share, uint256 tier2Share, uint256 tier3Share) =
getRewardForCycle(_cycleNo);
    // Determine the correct reward amount based on the gem ID...
```

```
    uint256 rewardAmount = ...;

    spay.transfer(msg.sender, rewardAmount);

    emit EarnPoolReward(msg.sender, rewardAmount, _nonce, _cycleNo, _gemId);
}
```

The earningPoolReward function uses the getActiveCycle function to determine the currently active cycle, and the getRewardForCycle function to determine the correct reward amount for that cycle. The reward amount is then transferred to the caller.

ID	5
Title	Enable Users to Invoke the confiscateGlasses Function Themselves
Path	contracts/ScriptTV.sol
Function Name	confiscateGlasses

Description: The confiscateGlasses function in its current implementation gives the user owner the power to confiscate any pair of glasses, regardless of who owns them. From the point of threat modeling this design inflicts an escalation of privilege attack vector where the power allotted to the user could potentially be misused.

Recommendation: In the following version of the function, the require statement ensures that the function can only be called by the owner of the glasses. If anyone else tries to call the function, it will revert with an error message.

Revised Code:

```
function confiscateGlasses(uint256 _tokenID) external {
    address glassesOwner = scriptGlasses.ownerOf(_tokenID);
    require(msg.sender == glassesOwner, "Only the owner of the glasses can
confiscate them.");
    scriptGlasses.transferFrom(glassesOwner, address(this), _tokenID);
    emit GlassesConfiscated(glassesOwner, _tokenID);
}
```

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.