SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Footium
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: May 2 - May 5, 2023
Prepared on: December 7, 2023

Introduction

Footium is a multiplayer football management game where players own and manage their own digital football club.

Scope

Repository: logiclogue/footium-eth-shareable

Branch: master

Commit: 6c181ea79af7f6715e3891e65ea5ee8def1e957c

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
7	2

Security experts who found valid issues

0x52
0xRobocop
Dug
BAHOZ
juancito
sashik_eth
J4de
shogoki
GalloDaSballo
cergyk
0xAsen
kiki_dev
ast3ros
igingu
mstpr-brainbot
0xRan4212
qpzm
GimelSec
shaka
ctf_sec
Quantish
MiloTruck
PokemonAuditSimulator
CMierez
toshii
pengun
BenRai

Brenzee
Phantasmagoria
descharre
santipu_
alexzoid
ni8mare
TheNaubit
deadrxsezzz
innertia
0xGoodess
n1punp
Sulpiride
lewisbroadhurst
PTolev
0xeix
wzrdk3lly
0xStalin
jasonxiale
Tricko
shame
Bauer
cuthalion0x
nzm_
0xPkhatri
tsvetanovv
oualidpro
tsueti_
0xLook
0xhacksmithh
chaithanya_gali
ali_shehab
__141345__
thekmj
Proxy
georgits
tibthecat
jprod15
alliums8520
djxploit
0xGusMcCrae
14si2o_Flint
abiih
peanuts
R-Nemes
0xmuxyz
dacian
ACai7
oot2k
l3r0ux
sach1r0
favelanky
Polaris_tow
Piyushshukla
ddimitrov22
8olidity
whoismatthewmc1
AlexCzm
SanketKogekar
Bauchibred
Koolex
indijanc
0xHati
ravikiran.web3
holyhansss
Cryptor
PRAISE
berlin-101
yy
0xnirlin
DevABDee
josephdara
Diana

Issue H-1: Escrow approvals are not cleared when club is transferred allowing for abuse after transfer

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/289

Found by

0x52, BenRai, Brenzee, CMierez, J4de, MiloTruck, PokemonAuditSimulator, Quantish, cergyk, ctf_sec, mstpr-brainbot, pengun, sashik_eth, shaka, shogoki, toshii

Summary

Escrow approvals remain even across club token transfers. This allows a malicious club owners to sell their club then drain everything after sale due to previous approvals.

Vulnerability Detail

ERC20 and ERC721 token approval persist regardless of the owner of the club. The result is that approvals set by one owner can be accessed after a token has been sold or transferred. This allows the following attack:

1) User A owns clubId = 1
2) User A sets approval to themselves
3) User A sells clubId = 1 to User B
4) User A uses persistent approval to drain all players and tokens

Impact

Malicious approvals can be used to drain club after sale

Code Snippet

FootiumEscrow.sol#L75-L81

FootiumEscrow.sol#L90-L96

Tool used

Manual Review

Recommendation

Club escrow system needs to be redesigned

Discussion

logiclogue

This *could* be fixed by dropping the escrow as discussed in the other issue. But this will re-surface soon

Issue H-2: Malicious users can honeypot other users by transferring out ERC20 and ERC721 tokens right before sale

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/291

Found by

0x52, 0xAsen, 0xRobocop, BAHOZ, Dug, J4de, ast3ros, igingu, kiki_dev, sashik_eth, shogoki

Summary

Since the club and escrow are separate and tokens can be transferred at any time by the owner, it allows malicious users to honeypot victims.

Vulnerability Detail

Tokens can be transferred out of the escrow by the owner of the club at anytime. This includes right before (or even in the same block) that the club is sold. This allows users to easily honeypot victims when selling clubs:

1) User A owns Club 1
2) Club 1 has players worth 5 ETH
3) User A lists Club 1 for 2.5 ETH
4) User B buys Club 1
5) User A sees the transaction in the mempool and quickly transfers all the players out
6) User A maintains all their players and User B now has an empty club

Impact

Malicious users can honeypot other users

Code Snippet

https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumEscrow.sol#L105-L111

https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumEscrow.sol#L120-L126

Tool used

Manual Review

Recommendation

Club/escrow system needs a redesign

Discussion

logiclogue

Currently discussed solution involves either adding a timelock on the contract *or* removing the escrow contract entirely. Both have product implications and will need longer to evaluate

Issue M-1: Certain ERC20 token does not return bool from approve and transfer and transaction revert

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/14

Found by

0xGoodess, Dug, GalloDaSballo, TheNaubit, ctf_sec, deadrxsezzz, innertia, juancito, mstpr-brainbot, n1punp

Summary

Certain ERC20 token does not return bool from approve and transfer and transaction revert

Vulnerability Detail

According to

https://github.com/d-xo/weird-erc20#missing-return-values

Some tokens do not return a bool on ERC20 methods and use IERC20 token interface will revert transaction

Certain ERC20 token does not return bool from approve and transfer and transaction revert

```
function setApprovalForERC20(
     IERC20 erc20Contract,
     address to,
     uint256 amount
 ) external onlyClubOwner {
     erc20Contract.approve(to, amount);
 }
```

and

```
function transferERC20(
    IERC20 erc20Contract,
    address to,
    uint256 amount
) external onlyClubOwner {
    erc20Contract.transfer(to, amount);
}
```

the transfer / approve can fail slienlty

Impact

Some tokens do not return a bool on ERC20 methods and use IERC20 token interface will revert transaction

Code Snippet

https://github.com/sherlock-audit/2023-04-footium/blob/11736f3f7f7efa88cb99ee98b04b85a46621347c/footium-eth-shareable/contracts/FootiumEscrow.sol#L80

https://github.com/sherlock-audit/2023-04-footium/blob/11736f3f7f7efa88cb99ee98b04b85a46621347c/footium-eth-shareable/contracts/FootiumEscrow.sol#L95

Tool used

Manual Review

Recommendation

Use Openzeppelin SafeTransfer / SafeApprove

Discussion

logiclogue

May not fix depending on whether the escrow is dropped

Issue M-2: Users might lose funds as `claimERC20Prize()` doesn't revert for no-revert-on-transfer tokens

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/86

Found by

0xAsen, 0xGoodess, 0xGusMcCrae, 0xPkhatri, 0xRobocop, 0xStalin, 0xeix, 0xmuxyz, 0xnirlin, 14si2o_Flint, 8olidity, ACai7, AlexCzm, BAHOZ, Bauchibred, Bauer, Cryptor, DevABDee, Diana, GimelSec, J4de, Koolex, MiloTruck, PRAISE, PTolev, Phantasmagoria, Piyushshukla, PokemonAuditSimulator, Polaris_tow, Proxy, Quantish, R-Nemes, SanketKogekar, Sulpiride, TheNaubit, Tricko, __141345__, abiih, alliums8520, ast3ros, berlin-101, cergyk, ctf_sec, cuthalion0x, dacian, ddimitrov22, deadrxsezzz, djxploit, favelanky, georgits, holyhansss, innertia, jasonxiale, josephdara, jprod15, kiki_dev, l3r0ux, lewisbroadhurst, nzm_, oot2k, oualidpro, peanuts, ravikiran.web3, sach1r0, santipu_, sashik_eth, shaka, shame, thekmj, tibthecat, tsvetanovv, whoismatthewmc1, wzrdk3lly, yy

Summary

Users can call `claimERC20Prize()` without actually receiving tokens if a no-revert-on-failure token is used, causing a portion of their claimable tokens to become unclaimable.

Vulnerability Detail

In the `FootiumPrizeDistributor` contract, whitelisted users can call `claimERC20Prize()` to claim ERC20 tokens. The function adds the amount of tokens claimed to the user's total claim amount, and then transfers the tokens to the user:

FootiumPrizeDistributor.sol#L128-L131

```
if (value > 0) {
    totalERC20Claimed[_token][_to] += value;
    _token.transfer(_to, value);
}
```

As the the return value from `transfer()` is not checked, `claimERC20Prize()` does not revert even when the transfer of tokens to the user fails.

This could potentially cause users to lose assets when:

1. `_token` is a no-revert-on-failure token.
2. The user calls `claimERC20Prize()` with `value` higher than the contract's token balance.

As the contract has an insufficient balance, `transfer()` will revert and the user receives no tokens. However, as `claimERC20Prize()` succeeds, `totalERC20Claimed` is permanently increased for the user, thus the user cannot claim these tokens again.

Impact

Users can call `claimERC20Prize()` without receiving the token amount specified. These tokens become permanently unclaimable for the user, leading to a loss of funds.

Code Snippet

https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumPrizeDistributor.sol#L128-L131

Tool used

Manual Review

Recommendation

Use `safeTransfer()` from Openzeppelin's SafeERC20 to transfer ERC20 tokens. Note that transferERC20() in `FootiumEscrow.sol` also uses `transfer()` and is susceptible to the same vulnerability.

Issue M-3: Clubs can mint +1 players more than maxGenerationId

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/152

Found by

Phantasmagoria, Sulpiride, alexzoid, descharre, juancito, mstpr-brainbot, ni8mare, santipu_, shaka

Summary

As stated in doc here Initially clubs should be able to mint 10 players. However, clubs can always mint +1 players more than `maxGenerationId`

Vulnerability Detail

An off-by-one error is existed in academy contract when checking the generation ID while minting players. The code currently allows clubs to mint up to `maxGenerationId + 1` players, which is inconsistent with the documentation. If clubs send generation IDs in the following sequence: 0-1-2-3-4-5-6-7-8-9-10, all of these numbers will be valid, and the club will be able to mint `maxGenerationId + 1` players instead of the intended `maxGenerationId`

Impact

Since this is not intended behaviour according to the protocol docs, I'll label it as high.

Code Snippet

https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumAcademy.sol#L186-L188 here code checks whether the generation id is higher than maxGeneration or not, and it can be equal to maxGeneration.

Tool used

Manual Review

Recommendation

Use `generationId >= _maxGenerationId` or do not count the "0" index

Discussion

logiclogue

Marked as 'Will Fix'. I believe the solution here is instead to update the documentation to say "At game launch this value will be 9". Because the documentation as far as I'm aware is correct in that it says `maxGenerationId` is the maximum generation ID, it's just that it does not represent the total players per cohort, which was the assumption with the default value being 10.

dugdaniels

Escalate for 10 USDC

This should be considered Low/Informational.

The variable is named `maxGenerationId`. As the name implies (and as noted in the docs), it is the maximum integer ID that can be used to mint. A `maxGenerationID` of `10` therefore means that you can mint a player with an ID of `10`. The sponsor confirmed that this is working as intended and that only the documentation needs to be clarified.

Furthermore, to imply that a `maxGenerationId` of `10` should actually enforce a max id of `9` is illogical.

Ultimately, it's a configuration variable and the sponsor is updating the documentation to reflect the value that will be set to at launch. A documentation update should only be considered as informational. There is no exploit here.

sherlock-admin

> Escalate for 10 USDC
>
> This should be considered Low/Informational.
>
> The variable is named `maxGenerationId`. As the name implies (and as noted in the docs), it is the maximum integer ID that can be used to mint. A `maxGenerationID` of `10` therefore means that you can mint a player with an ID of `10`. The sponsor confirmed that this is working as intended and that only the documentation needs to be clarified.
>
> Furthermore, to imply that a `maxGenerationId` of `10` should actually enforce a max id of `9` is illogical.
>
> Ultimately, it's a configuration variable and the sponsor is updating the documentation to reflect the value that will be set to at launch. A documentation update should only be considered as informational. There is no exploit here.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

NishithPat

Escalate for 10 USDC Disagree with the sponsor's comment saying that the documentation is right and that `maxGenerationId` is "the maximum generation ID, it's just that it does not represent the total players per cohort" or "it is the maximum integer ID that can be used to mint".

The docs say that `maxGenerationId` is "the number of players that can be minted per cohort". Check the definition of maxGenerationId in the doc.

Then going by the docs, if `maxGenerationId` is the number of players that can be minted per cohort, then the contract does indeed allow `maxGenerationId` + 1 player to be minted. In that case, the issue should be valid.

sherlock-admin

> Escalate for 10 USDC Disagree with the sponsor's comment saying that the documentation is right and that `maxGenerationId` is "the maximum generation ID, it's just that it does not represent the total players per cohort" or "it is the maximum integer ID that can be used to mint".
>
> The docs say that `maxGenerationId` is "the number of players that can be minted per cohort". Check the definition of maxGenerationId in the doc.
>
> Then going by the docs, if `maxGenerationId` is the number of players that can be minted per cohort, then the contract does indeed allow `maxGenerationId` + 1 player to be minted. In that case, the issue should be valid.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

logiclogue

Sponsor here, agreed with @NishithPat . My original statement is incorrect, the documentation should be updated to remove the statement "the number of players that can be minted per cohort". Their suggestion is correct

ctf-sec

Based on the comments above, NishithPat escalation is valid

hrishibhat

Escalation accepted

Valid medium Accepting @NishithPat's escalation Given that the issue correctly identifies the flaw in the code as against what the documentation says it's supposed to do, this is a valid issue.

sherlock-admin

> Escalation accepted
>
> Valid medium Accepting @NishithPat's escalation Given that the issue correctly identifies the flaw in the code as against what the documentation says it's supposed to do, this is a valid issue.

```
This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on
↪  this issue.
```

Issue M-4: Users can bypass Player royalties on EIP2981 compatible markets by selling clubs as a whole

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/293

Found by

0x52, 0xRobocop

Summary

Players have a royalty built in but clubs do not. This allows bulk sale of players via clubs to bypass the fee when selling players.

Vulnerability Detail

FootiumPlayer.sol#L16-L23

```
contract FootiumPlayer is
    ERC721Upgradeable,
    AccessControlUpgradeable,
    ERC2981Upgradeable,
    PausableUpgradeable,
    ReentrancyGuardUpgradeable,
    OwnableUpgradeable
{
```

FootiumPlayer implements the EIP2981 standard which creates fees when buy/selling the players.

FootiumClub.sol#L15-L21

```
contract FootiumClub is
    ERC721Upgradeable,
    AccessControlUpgradeable,
    PausableUpgradeable,
    ReentrancyGuardUpgradeable,
    OwnableUpgradeable
{
```

FootiumClub on the other hand never implements this standard. This allows users to sell players by selling their club to avoid any kind of fee on player sales.

Impact

Users can bypass fees on player sales by selling club instead

Code Snippet

FootiumClub.sol#L15-L21

Tool used

Manual Review

Recommendation

Implement EIP2981 on clubs as well

Discussion

thangtranth

Escalate for 10 USDC.

I believe this issue is invalid because when reading the codebase, it seems to be the design choice of the Footium protocol to not charge new owner of the club the royalty fee for player transfer. The reason is in theory, the players still remain in the same club and are not transferred to other club. Transfer royalty is only applicable when a player NFT is transferred from one club to another, not when a club NFT is sold to a new owner. It is consistent with how other football works.

Furthermore, there is no documentation that states or implies that buying a club NFT requires paying transfer royalty for every player NFT in the club. This leads me and many other Watsons to believe that this is not an issue.

The EIP2981 is not enforced, therefore if a user want to bypass the player royalties, there are many ways to do it so selling club as a whole is not necessary. For example listing in a NFT marketplace that does not support EIP2981, so no royalty is charged.

sherlock-admin

> Escalate for 10 USDC.
>
> I believe this issue is invalid because when reading the codebase, it seems to be the design choice of the Footium protocol to not charge new owner of the club the royalty fee for player transfer. The reason is in theory, the players still remain in the same club and are not transferred to other club. Transfer royalty is only applicable when a player NFT is

> transferred from one club to another, not when a club NFT is sold to a new owner. It is consistent with how other football works.
>
> Furthermore, there is no documentation that states or implies that buying a club NFT requires paying transfer royalty for every player NFT in the club. This leads me and many other Watsons to believe that this is not an issue.
>
> The EIP2981 is not enforced, therefore if a user want to bypass the player royalties, there are many ways to do it so selling club as a whole is not necessary. For example listing in a NFT marketplace that does not support EIP2981, so no royalty is charged.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0xRobocop

Escalate for 10 USDC

This escalation is to contra-argument the escalation made by Watson thangtranth.

While it is correct that the issue over-estimated the "Users can bypass Player royalties", and that this can be done in an easier way with a marketplace that does not enforce royalties via EIP2981 as mentioned in the watson's escalation.

There exists an inconsistency in that players NFTs implement the EIP2981 whereas that club NFTs do not, so marketplaces that do respect the EIP-2981 won't pay the royalties for the club NFTs, and this can be seen as a loss of "yield" for the developers of the footium protocol.

The argument that this seemed to be the intended behavior (not charging royalties for clubs) is subjective, as it could have easily been missed by the developers. And given that the sponsor confirmed the issue, it seems that this was the case.

sherlock-admin

> Escalate for 10 USDC
>
> This escalation is to contra-argument the escalation made by Watson thangtranth.
>
> While it is correct that the issue over-estimated the "Users can bypass Player royalties", and that this can be done in an easier way with a marketplace that does not enforce royalties via EIP2981 as mentioned in the watson's escalation.

> There exists an inconsistency in that players NFTs implement the EIP2981 whereas that club NFTs do not, so marketplaces that do respect the EIP-2981 won't pay the royalties for the club NFTs, and this can be seen as a loss of "yield" for the developers of the footium protocol.
>
> The argument that this seemed to be the intended behavior (not charging royalties for clubs) is subjective, as it could have easily been missed by the developers. And given that the sponsor confirmed the issue, it seems that this was the case.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

logiclogue

To clarify here from the protocol designer, it's correct that we don't intend for clubs to pay for the royalties for all players on transfer. Instead, we anticipate that the club will subjectively be valued in terms of the players that belong to the club. As a result, clubs should implement the EIP2981 standard so that on club sale royalties will be paid back to the protocol developers

ctf-sec

0xRobocop escalation is valid, a valid medium for this submission

hrishibhat

Escalation accepted

Valid medium Given that the clubs should implement eip2981 to pay the protocol royalties on sale, considering this issue a valid medium

sherlock-admin

> Escalation accepted
>
> Valid medium Given that the clubs should implement eip2981 to pay the protocol royalties on sale, considering this issue a valid medium

```
This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on
↪  this issue.
```

Issue M-5: Merkle leaf values for _clubDivsMerkleRoot are 64 bytes before hashing which can lead to merkle tree collisions

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/300

Found by

0x52, 0xRan4212, GimelSec, cergyk, qpzm

Summary

FootiumAcademy hashes 64 bytes when calculating leaf allowing it to collide with the internal nodes of the merkle tree.

Vulnerability Detail

MerkleProofUpgradeable.sol puts the following warning at the beginning of the contract:

```
* WARNING: You should avoid using leaf values that are 64 bytes long prior to
* hashing, or use a hash function other than keccak256 for hashing leaves.
* This is because the concatenation of a sorted pair of internal nodes in
* the merkle tree could be reinterpreted as a leaf value.
```

FootiumAcademy.sol#L235-L240

```
if (
    !MerkleProofUpgradeable.verify(
        divisionProof,
        _clubDivsMerkleRoot,
        keccak256(abi.encodePacked(clubId, divisionTier)) <- @audit-issue 64
        ↪  bytes before hashing allows collisions with internal nodes
    )
```

This is problematic because FootiumAcademy uses clubId and divisionTier as the base of the leaf, which are both uint256 (32 bytes each for 64 bytes total). This allows collision between leaves and internal nodes. These collisions could allow users to mint to divisions that otherwise would be impossible.

Impact

Users can abuse merkle tree collisions to mint in non-existent divisions and bypass minting fees

Code Snippet

FootiumAcademy.sol#L228-L272

Tool used

Manual Review

Recommendation

Use a combination of variables that doesn't sum to 64 bytes

Discussion

0xRobocop

Escalate for 10 USDC

I encountered the same issue, and I will provide my analysis on why this is not an issue in the current codebase, not even to classify as a medium.

The problem described is that the concatenation (which will be of 64 bytes long, if keccak256 is used) of 2 internal nodes of a merkle tree could collide with leaf values that also are 64 bytes long. For example:

Leaf Value (64 bytes), ignore the = symbols, they are used to separate the leaf value into two tranches of 32 bytes for illustration purposes:

`0x9c644f34f630d76b78c6ccecfceabcf6b9f0def47e637403ac15c63bc6030920=====d6a1fb975713511138879534 20cffc381edf9874c13f336e0c2c01bffe2214f4`

Internal Node 1:
`0x9c644f34f630d76b78c6ccecfceabcf6b9f0def47e637403ac15c63bc6030920`

Internal Node 2:
`0xd6a1fb975713511138879534 20cffc381edf9874c13f336e0c2c01bffe2214f4`

There we have a collision of a leaf value with the concatenation of the hash values of 2 sorted internal nodes.

In the current codebase, the probability of this to happen is negligible. The leaf value is computed as the concatenation of `clubId` and `divisionTier`. `clubId` is a variable that increments 1 by 1 (when a club is minted), if we were to mint 1 club in each second until the year 2106, we could mint a maximum amount of 2^32 clubs and the maximum `divisionTier` will be way lower than this number, but for illustration purposes I will assume that it is also 2^32.

Then, the maximum value the leaf can have is (again the = symbols are for illustration purposes to show the concatenation):

`0x0000000000000000000000000000000000000000000000000000000100000000====0000000000000000000000000000000000000000000000000000000100000000`

For a potential collision to happen, the concatenation of 2 internal nodes (which are keccak256 values) must be equal or smaller than the above number. Due to the inequality (equal or smaller) we are only interested in the internal node that will end-up on the left side. The hash value of the left-side internal node will need to be smaller than:

`0x0000000000000000000000000000000000000000000000000000000100000000`

Which has 55 leading zeros. The lowest hash found in the bitcoin network has 23 leading zeros reference.

Please keep in mind that OZ contracts are too general and that's why they put the warning, not to mention that the warning is a "should" not a "must" since the exploitation of this phenomenon will depend highly on the contracts which uses their MerkleProof library.

If the values that make up the leaf were to be controllable by the users, I would agree this is a medium or even a high vulnerability, but the leaf is made up of values, `clubId` and `divisionTier`, that are not controllable by the users and that we can predict their values.

PD: I would argue that issues reported based on warnings written in general-purpose libraries, should be accompanied with proof of concepts (coded or not) on how can be exploited in the codebase, even if it has some hypotheticals in the case of mediums, but not too hypothetical like producing a hash value with 55 leading zeros.

sherlock-admin

> Escalate for 10 USDC
>
> I encountered the same issue, and I will provide my analysis on why this is not an issue in the current codebase, not even to classify as a medium.
>
> The problem described is that the concatenation (which will be of 64 bytes long, if keccak256 is used) of 2 internal nodes of a merkle tree could collide with leaf values that also are 64 bytes long. For example:
>
> Leaf Value (64 bytes), ignore the = symbols, they are used to separate the leaf value into two tranches of 32 bytes for illustration purposes:
>
> `0x9c644f34f630d76b78c6ccecfceabcf6b9f0def47e637403ac15c63bc6030920=====d6a1fb97571351113887953420cffc381edf9874c13f336e0c2c01bffe2214f4`
>
> Internal Node 1:
> `0x9c644f34f630d76b78c6ccecfceabcf6b9f0def47e637403ac15c63bc6030920`

Internal Node 2:
`0xd6a1fb97571351113887953420cffc381edf9874c13f336e0c2c01bffe2214f4`

There we have a collision of a leaf value with the concatenation of the hash values of 2 sorted internal nodes.

In the current codebase, the probability of this to happen is negligible. The leaf value is computed as the concatenation of `clubId` and `divisionTier`. `clubId` is a variable that increments 1 by 1 (when a club is minted), if we were to mint 1 club in each second until the year 2106, we could mint a maximum amount of 2^32 clubs and the maximum `divisionTier` will be way lower than this number, but for illustration purposes I will assume that it is also 2^32.

Then, the maximum value the leaf can have is (again the = symbols are for illustration purposes to show the concatenation):

`0x00000000000000000000000000000000000000000000000000000000100000000====00000000000000000000000000000000000000000000000000000000100000000`

For a potential collision to happen, the concatenation of 2 internal nodes (which are keccak256 values) must be equal or smaller than the above number. Due to the inequality (equal or smaller) we are only interested in the internal node that will end-up on the left side. The hash value of the left-side internal node will need to be smaller than:

`0x00000000000000000000000000000000000000000000000000000000100000000`

Which has 55 leading zeros. The lowest hash found in the bitcoin network has 23 leading zeros reference.

Please keep in mind that OZ contracts are too general and that's why they put the warning, not to mention that the warning is a "should" not a "must" since the exploitation of this phenomenon will depend highly on the contracts which uses their MerkleProof library.

If the values that make up the leaf were to be controllable by the users, I would agree this is a medium or even a high vulnerability, but the leaf is made up of values, `clubId` and `divisionTier`, that are not controllable by the users and that we can predict their values.

PD: I would argue that issues reported based on warnings written in general-purpose libraries, should be accompanied with proof of concepts (coded or not) on how can be exploited in the codebase, even if it has some hypotheticals in the case of mediums, but not too hypothetical like producing a hash value with 55 leading zeros.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

SergeKireev

Escalate for 10 USDC,

As per my report of the issue:
https://github.com/sherlock-audit/2023-04-footium-judging/issues/170

For the issue to be valid there only needs to be a very specific value of `clubId` crafted. Indeed the `divisionId` arg should not be an existent division id for the exploit to work.

The specific value which the clubId should be equal to is: `any left internal node of the merkle tree`.

The owner of the contract has the power to mint arbitrary values (nowhere it is stated that it would be an auto-incremented id):
https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumClubMinter.sol#L68-L82

So the `maximums` determined in the first escalation do not hold.

Given that the vulnerability is only exploitable if the team decides to open the mint to the public in the future or if the owner account is compromised, the severity can be lowered. However the reasons for which the sponsor found this issue interesting and worth to fix still hold.

sherlock-admin

> Escalate for 10 USDC,
>
> As per my report of the issue:
> https://github.com/sherlock-audit/2023-04-footium-judging/issues/170
>
> For the issue to be valid there only needs to be a very specific value of `clubId` crafted. Indeed the `divisionId` arg should not be an existent division id for the exploit to work.
>
> The specific value which the clubId should be equal to is: `any left internal node of the merkle tree`.
>
> The owner of the contract has the power to mint arbitrary values (nowhere it is stated that it would be an auto-incremented id):
> https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumClubMinter.sol#L68-L82
>
> So the `maximums` determined in the first escalation do not hold.
>
> Given that the vulnerability is only exploitable if the team decides to open the mint to the public in the future or if the owner account is

> compromised, the severity can be lowered. However the reasons for which the sponsor found this issue interesting and worth to fix still hold.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0xRobocop

Escalate for 10 USDC

I totally agree with SergeKireev, I missed that clubID is not incremented 1 by 1 but controlled by the protocol owner. But I think the severity is still low / informational since this is not exploitable unless owner makes a mistake.

sherlock-admin

> Escalate for 10 USDC
>
> I totally agree with SergeKireev, I missed that clubID is not incremented 1 by 1 but controlled by the protocol owner. But I think the severity is still low / informational since this is not exploitable unless owner makes a mistake.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

ctf-sec

Based on the escalation and other duplicate report, can be a valid medium, severity is not high

0xRan4212

Escalate for 10 USDC.

I disagree with the downgrade to medium, even though I had originally reported this issue with a medium severity (see my report on 209).

The sponsor considered *realistic* the scenario where the user can control the tokenId value, so much so that stated that after audit they will explicitly prevent it that case (see the sponsor comment on report 179).

Before the sponsor acknowledgment, as I said on my report, the exploitability was theoretical, therefore a medium severity was adequate. Basically, it violated the "attack path is possible with reasonable assumptions that mimic on-chain

conditions" requirement from the high severity definition here. However, post-acknowledgment I strongly believe this requirement is now satisfied.

sherlock-admin

> Escalate for 10 USDC.
>
> I disagree with the downgrade to medium, even though I had originally reported this issue with a medium severity (see my report on 209).
>
> The sponsor considered *realistic* the scenario where the user can control the tokenId value, so much so that stated that after audit they will explicitly prevent it that case (see the sponsor comment on report 179).
>
> Before the sponsor acknowledgment, as I said on my report, the exploitability was theoretical, therefore a medium severity was adequate. Basically, it violated the "attack path is possible with reasonable assumptions that mimic on-chain conditions" requirement from the high severity definition here. However, post-acknowledgment I strongly believe this requirement is now satisfied.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

hrishibhat

Escalation accepted

Valid medium Although the documentation mentions that user is allowed to use choose their club ID this is definitely a valid issue, however the implementation of the code still happens through an admin function, and for that reason considering this issue a valid medium.

sherlock-admin

> Escalation accepted
>
> Valid medium Although the documentation mentions that user is allowed to use choose their club ID this is definitely a valid issue, however the implementation of the code still happens through an admin function, and for that reason considering this issue a valid medium.

```
This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on
↪  this issue.
```

Issue M-6: `changeMaxGenerationId` allows to mint tokens from older generations retroactively

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/319

Found by

BAHOZ, Dug, GalloDaSballo, juancito

Summary

`changeMaxGenerationId` is meant to allow more minting for the current generation, however, because of the fact that merkleProofs do not validate for the current season, whenever the limit will be raised, the limit will allow to mint players from older seasons

It will unlock a lot more minting than it may be intended

Vulnerability Detail

If for any reason `maxGenerationId` is increased, because of the fact that `MerkleProofUpgradeable.verify` doesn't check for `generationIds` and `seasonId` then not only new players from the current season can be minted, but also players from older seasons.

Impact

Minting of X tokens where X is the number of old seasons

Code Snippet

https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumAcademy.sol#L257-L269

Tool used

Manual Review

Recommendation

Enforce minting exclusively for the current season or change validation to validate the generations and seasons being minted

Discussion

logiclogue

This is an interesting one. A decision needs to be made whether this is the intended behaviour. Marking as 'Sponsor Confirmed'.

logiclogue

For this, would it make sense to create a map for each season to maxGenerationId, then validate the minting params' season to maxGenerationId?

Issue M-7: Minting inconsistencies on FootiumPlayer and FootiumClub

Source: https://github.com/sherlock-audit/2023-04-footium-judging/issues/342

Found by

0xAsen, 0xHati, 0xLook, 0xPkhatri, 0xRobocop, 0xStalin, 0xeix, 0xhacksmithh, BAHOZ, Bauchibred, Bauer, Dug, GalloDaSballo, Koolex, PTolev, Phantasmagoria, TheNaubit, Tricko, ali_shehab, cergyk, chaithanya_gali, ctf_sec, cuthalion0x, deadrxsezzz, descharre, indijanc, jasonxiale, kiki_dev, lewisbroadhurst, nzm_, oualidpro, sashik_eth, shame, shogoki, tsueti_, tsvetanovv, wzrdk3lly

Summary

The `FootiumClub.sol` contract when minting uses `_mint()` instead of `_safeMint()` which can cause to mint a club to a contract who does not support nfts. On the other hand `FootiumPlayer.sol` uses `_safeMint()`.

Vulnerability Detail

See summary.

Impact

`FootiumClub.sol` might mint a club NFT to a contract that cannot handle nfts.

Code Snippet

https://github.com/sherlock-audit/2023-04-footium/blob/main/footium-eth-shareable/contracts/FootiumClub.sol#L65

Tool used

Manual Review

Recommendation

Use `_safeMint()` as in FootiumPlayer.