SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	TITLES
Prepared by:	Sherlock
Lead Security Expert:	xiaoming90
Dates Audited:	April 22 - April 26, 2024
Prepared on:	May 31, 2024

Introduction

TITLES builds creative tools powered by artist-owned AI models. The underlying TITLES protocol enables the publishing of referential NFTs, including managing attribution and splitting payments with the creators of the attributed works.

Scope

Repository: titlesnyc/wallflower-contract-v2

Branch: main

Commit: d23c44def46ce4fd74f3daae36df0135acae7505

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
14	3

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

xiaoming90
mt030d
Varun_05
14si2o_Flint
ComposableSecurity
cducrest-brainbot
sammy
0x73696d616f
ArsenLupin
Kalogerone
fugazzi
0rpse
me_na0mi
CodeWasp
ZdravkoHr.
fibonacci
KupiaSec
juan
T1MOH
ast3ros
ZanyBonzy
alexzoid
1337
Yu3H0
trachev
blackhole
jennifer37
zigtur
kennedy1030
zoyi
y4y
cu5t0mPe0
BiasedMerc
BengalCatBalu
AlexCzm
ubl4nk
radin200
4rdiii
brakeless
i3arba
ironside
recursiveEth
den_sosnovskyi
cryptic
jecikpo
Krace
Bigsam
0x486776
ff
techOptimizor
gladiator111
durov
blockchain555
Shaheen
funkornaut
Dots
joicygiore
FastTiger
AgileJune
valentin2304
merlinboii
0xboriskataa
bareli
Brenzee
nisedo
gesha17
eeshenggoh
popeye
Matin
smbv-1923
kn0t
ydlee
azanux
0xlucky
0xShiki
lllllll

Issue H-1: Users can exploit the batch minting feature to avoid paying minting fees for tokens

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/264

Found by

0x486776, 0x73696d616f, 1337, 4rdiii, AgileJune, AlexCzm, ArsenLupin, BiasedMerc, Brenzee, CodeWasp, ComposableSecurity, Dots, Kalogerone, KupiaSec, Shaheen, Varun_05, ast3ros, bareli, blockchain555, cducrest-brainbot, cu5t0mPe0, durov, ff, fugazzi, funkornaut, gesha17, gladiator111, jennifer37, joicygiore, juan, kennedy1030, nisedo, sammy, techOptimizor, trachev, ubl4nk, valentin2304, xiaoming90, y4y

Summary

Users can exploit the batch minting feature to avoid paying minting fees for tokens, leading to a loss of fees for the creator and fee recipients.

Vulnerability Detail

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L304

```
File: Edition.sol
304:     function mintBatch(
305:         address[] calldata receivers_,
306:         uint256 tokenId_,
307:         uint256 amount_,
308:         bytes calldata data_
309:     ) external payable {
310:         // wake-disable-next-line reentrancy
311:         FEE_MANAGER.collectMintFee{value: msg.value}(
312:             this, tokenId_, amount_, msg.sender, address(0),
↪  works[tokenId_].strategy
313:         );
314:
315:         for (uint256 i = 0; i < receivers_.length; i++) {
316:             _issue(receivers_[i], tokenId_, amount_, data_);
317:         }
318:
319:         _refundExcess();
320:     }
```

Assume that the total mint fee (protocol flat fee = 0.0006 ETH + minting fee = 0.0004) for each token is 0.001 ETH. Bob wants to mint 1000 tokens, and he has to pay a minting fee of 1 ETH.

However, the issue is that Bob can mint 1000 tokens without paying the full mint fee of 1 ETH as shown in the step below:

1. Bob calls the above `mintBatch` function with `receivers_` array set to `1000 x Bob address` and the `amount_` parameter set to `1`.
2. Line 331 above will compute the mint fee that Bob needs to pay. In this case, the `amount_` is one (1), and the total mint fee will be equal to `1 X 0.001 ETH`.
3. The for-loop at Line 315 above will loop 1000 times as the `receivers_.length` is 1000. Each loop will mint one token to Bob.
4. At the end of the for loop, Bob will receive 1000 while only paying a minting fee of 0.001 ETH instead of 1 ETH.
5. In other words, Bob only paid the minting fee for the first token and did not pay the minting fee for the rest of the 999 tokens.

Impact

Loss of fees for the creator and fee recipients as minters can avoid paying the fee using the trick mentioned in this report.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L304

Tool used

Manual Review

Recommendation

Consider the following change to ensure that the minting fee is computed based on the total number of tokens minted to all receivers.

```
    function mintBatch(
        address[] calldata receivers_,
        uint256 tokenId_,
        uint256 amount_,
        bytes calldata data_
    ) external payable {
        // wake-disable-next-line reentrancy
```

```
        FEE_MANAGER.collectMintFee{value: msg.value}(
-           this, tokenId_, amount_, msg.sender, address(0),
 ↪  works[tokenId_].strategy
+           this, tokenId_, amount_ * receivers_.length, msg.sender,
 ↪  address(0), works[tokenId_].strategy
        );

        for (uint256 i = 0; i < receivers_.length; i++) {
            _issue(receivers_[i], tokenId_, amount_, data_);
        }

        _refundExcess();
    }
```

Discussion

pqseags

Valid, will fix

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue H-2: Original collection referrer will be overwritten when a new collection/work is created

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/265

Found by

14si2o_Flint, Varun_05, mt030d, xiaoming90

Summary

Original collection referrers will be overwritten when a new collection/work is created. This results in the original collection referrers being unable to collect the fee they are entitled to, leading to a loss of assets for them.

Vulnerability Detail

> [!NOTE]
>
> The following information is taken from the sponsor's response in the Contest's Discord Channel.
>
> There are two types of referrers in this ecosystem. The first is a collection referrer, who refers the creation of a new Edition, and gets a cut of all mint fees for that Edition. The second is a mint referrer, who refers a mint and gets a cut of the mint fee for that mint.
>
> The following describes the difference between collection referrer and mint referrer.
>
> - User 1 creates a referral link to create a collection
> - User 2 uses that link to publish a collection
> - User 3 mints that collection. For this mint, user1 is treated as the collection referrer
> - User 4 creates a referral link to mint that collection
> - User 5 mints that collection. For this mint, user1 is treated as the collection referrer and user4 is treated as the mint referrer

Assume that Alice creates a referral link to create a collection. Bob uses that link to publish a new collection/work called $Collection_A$ that is based on an Edition called $Edition_A$. The collection referrer of $Collection_A$ will Alice.

Bob will call the following `TitlesCore.publish` function with the `edition` parameter set to $Edition_A$ and the `referrer_` parameter will be automatically set to Alice by the

front end.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/TitlesCore.sol#L103

```
File: TitlesCore.sol
098:     /// @notice Publishes a new Work in the given {Edition} using the given
↪  payload.
099:     /// @param edition_ The {Edition} to publish the Work in.
100:     /// @param payload_ The compressed payload for publishing the Work. See
↪  {WorkPayload}.
101:     /// @param referrer_ The address of the referrer.
102:     /// @return tokenId The token ID of the new Work.
103:     function publish(Edition edition_, bytes calldata payload_, address
↪  referrer_)
..SNIP..
140:         // Create the fee route for the new Work
141:         // wake-disable-next-line reentrancy
142:         Target memory feeReceiver = feeManager.createRoute(
143:             edition_, tokenId, _attributionTargets(work_.attributions),
↪  referrer_
..SNIP..
```

Within the `TitlesCore.publish`, the following `feeManager.createRoute` function will be executed internally. The `feeManager.createRoute` function will store Alice's wallet address within the `referrers[edition_]` mapping. Thus, the state of the `referrers` mapping will be as follows:

```
referrers[Edition_A] = Alice
```

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L125

```
File: FeeManager.sol
125:     function createRoute(
126:         IEdition edition_,
127:         uint256 tokenId_,
128:         Target[] calldata attributions_,
129:         address referrer_
130:     ) external onlyOwnerOrRoles(ADMIN_ROLE) returns (Target memory
↪  receiver) {
..SNIP..
158:         _feeReceivers[getRouteId(edition_, tokenId_)] = receiver;
159:         referrers[edition_] = referrer_;
160:     }
```


When someone mints a new token for $Collection_A$, Alice, who is the collection referrer, will get a share of the minting fee per Line 421 below.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L421

```
File: FeeManager.sol
412:     function _splitProtocolFee(
413:         IEdition edition_,
414:         address asset_,
415:         uint256 amount_,
416:         address payer_,
417:         address referrer_
418:     ) internal returns (uint256 referrerShare) {
419:         // The creation and mint referrers earn 25% and 50% of the
↪  protocol's share respectively, if applicable
420:         uint256 mintReferrerShare = getMintReferrerShare(amount_,
↪  referrer_);
421:         uint256 collectionReferrerShare =
↪  getCollectionReferrerShare(amount_, referrers[edition_]);
422:         referrerShare = mintReferrerShare + collectionReferrerShare;
```

Let's assume Charles also creates a referral link to create a collection. David uses that link to publish a new collection/work called $Collection_B$ that is based on the same Edition called $Edition_A$. The collection referral of $Collection_B$ will be Charles.

When the `FeeManager.createRoute` function is executed during the publishing of the new work/collection, Charles's wallet address will be stored within the `referrers[edition_]` mapping. Thus, the state of the `referrers` mapping will be as follows:

```
referrers[Edition_A] = Charles
```

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L125

```
File: FeeManager.sol
125:     function createRoute(
126:         IEdition edition_,
127:         uint256 tokenId_,
128:         Target[] calldata attributions_,
129:         address referrer_
130:     ) external onlyOwnerOrRoles(ADMIN_ROLE) returns (Target memory
↪  receiver) {
..SNIP..
158:         _feeReceivers[getRouteId(edition_, tokenId_)] = receiver;
```

```
159:             referrers[edition_] = referrer_;
160:         }
```

It is important to note that the `referrers[Edition_A]` have been updated from Alice to Charles here. At this point onwards, if someone mints tokens for $Collection_A$, the collection referral fee will be routed to Charles instead of Alice. Alice is the referral for $Collection_A$, yet she does not receive the referral fee, resulting in a loss of assets for Alice.

Impact

Loss of assets as shown in the above scenario. The original collection referrers are unable to collect the fee they are entitled to, leading to a loss of assets for them.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/TitlesCore.sol#L103

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L125

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L421

Tool used

Manual Review

Recommendation

Consider the following changes to ensure that the collection referral fee is routed to the correct collection referrer for each collection/work.

With the following changes, Alice will continue to receive the collection referral fee for $Collection_A$ and Bob will receive the collection referral fee for $Collection_B$ even if there are multiple collections/works for a specific Edition.

```
function createRoute(
    IEdition edition_,
    uint256 tokenId_,
    Target[] calldata attributions_,
    address referrer_
) external onlyOwnerOrRoles(ADMIN_ROLE) returns (Target memory receiver) {
..SNIP..
    _feeReceivers[getRouteId(edition_, tokenId_)] = receiver;
```

```
-    referrers[edition_] = referrer_;
+    referrers[getRouteId(edition_, tokenId_)] = referrer_;
}
```

```
function _splitProtocolFee(
    IEdition edition_,
+       uint256 tokenId,
    address asset_,
    uint256 amount_,
    address payer_,
    address referrer_
) internal returns (uint256 referrerShare) {
    // The creation and mint referrers earn 25% and 50% of the protocol's share
↪   respectively, if applicable
    uint256 mintReferrerShare = getMintReferrerShare(amount_, referrer_);
-    uint256 collectionReferrerShare = getCollectionReferrerShare(amount_,
↪   referrers[edition_]);
+    uint256 collectionReferrerShare = getCollectionReferrerShare(amount_,
↪   referrers[getRouteId(edition_, tokenId)]);
    referrerShare = mintReferrerShare + collectionReferrerShare;
```

Discussion

pqseags

Valid, but duplicate of #59

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue H-3: Collection referrers will not receive their share of the minting fee

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/267

Found by

0rpse, 0x486776, 0x73696d616f, 0xboriskataa, 1337, AlexCzm, ArsenLupin, BengalCatBalu, BiasedMerc, ComposableSecurity, FastTiger, KupiaSec, Matin, Varun_05, Yu3H0, ZdravkoHr., alexzoid, ast3ros, blockchain555, brakeless, cducrest-brainbot, cryptic, cu5t0mPe0, durov, eeshenggoh, ff, gladiator111, i3arba, ironside, jennifer37, kennedy1030, merlinboii, mt030d, popeye, radin200, sammy, techOptimizor, trachev, xiaoming90, y4y

Summary

Collection referrers will not receive their share of the minting fee, leading to a loss of assets for the collection referrers.

Vulnerability Detail

[!NOTE]

The following information is taken from the sponsor's response in the Contest's Discord Channel.

There are two types of referrers in this ecosystem. The first is a collection referrer, who refers the creation of a new Edition, and gets a cut of all mint fees for that Edition. The second is a mint referrer, who refers a mint and gets a cut of the mint fee for that mint.

The following describes the difference between collection referrer and mint referrer.

- User 1 creates a referral link to create a collection
- User 2 uses that link to publish a collection
- User 3 mints that collection. For this mint, user1 is treated as the collection referrer
- User 4 creates a referral link to mint that collection
- User 5 mints that collection. For this mint, user1 is treated as the collection referrer and user4 is treated as the mint referrer

Assume that Alice creates a referral link to create a collection. Bob uses that link to publish a collection called $Collection_1$. In this case, the collection referrer of $Collection_1$ will be set to Alice.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/TitlesCore.sol#L103

```
File: TitlesCore.sol
098:     /// @notice Publishes a new Work in the given {Edition} using the given
 ↪  payload.
099:     /// @param edition_ The {Edition} to publish the Work in.
100:     /// @param payload_ The compressed payload for publishing the Work. See
 ↪  {WorkPayload}.
101:     /// @param referrer_ The address of the referrer.
102:     /// @return tokenId The token ID of the new Work.
103:     function publish(Edition edition_, bytes calldata payload_, address
 ↪  referrer_)
..SNIP..
140:         // Create the fee route for the new Work
141:         // wake-disable-next-line reentrancy
142:         Target memory feeReceiver = feeManager.createRoute(
143:             edition_, tokenId, _attributionTargets(work_.attributions),
 ↪  referrer_
..SNIP..
```

When the `TitlesCore.publish` is executed to create a new collection, the `feeManager.createRoute` function will be executed internally. The `feeManager.createRoute` function will store Alice's wallet address within the `referrers[edition_]` mapping. Thus, the state of the `referrers` mapping will be as follows:

```
referrers[Edition_A] = Alice
```

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L125

```
File: FeeManager.sol
125:     function createRoute(
126:         IEdition edition_,
127:         uint256 tokenId_,
128:         Target[] calldata attributions_,
129:         address referrer_
130:     ) external onlyOwnerOrRoles(ADMIN_ROLE) returns (Target memory
 ↪  receiver) {
..SNIP..
158:         _feeReceivers[getRouteId(edition_, tokenId_)] = receiver;
```

```
159:         referrers[edition_] = referrer_;
160:     }
```

When someone mints a new token for $Collection_A$, Alice, who is the collection referral, should get a share of the minting fee. However, based on the current implementation of the `FeeManager._splitProtocolFee` function below, Alice will not receive her collection referral fee. Instead, the collection referral fee is routed to the mint referrer.

Line 438 below shows that the collection referral fee is routed to the mint referrer (`referrer_`) instead of the collection referrer (`referrers[edition_]`).

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L412

```
File: FeeManager.sol
412:     function _splitProtocolFee(
413:         IEdition edition_,
414:         address asset_,
415:         uint256 amount_,
416:         address payer_,
417:         address referrer_
418:     ) internal returns (uint256 referrerShare) {
419:         // The creation and mint referrers earn 25% and 50% of the
 ↪  protocol's share respectively, if applicable
420:         uint256 mintReferrerShare = getMintReferrerShare(amount_,
 ↪  referrer_); // @audit-info mint referrer
421:         uint256 collectionReferrerShare =
 ↪  getCollectionReferrerShare(amount_, referrers[edition_]); // @audit-info
 ↪  collection referrer
422:         referrerShare = mintReferrerShare + collectionReferrerShare;
..SNIP..
430:         _route(
431:             Fee({asset: asset_, amount: mintReferrerShare}),
432:             Target({target: referrer_, chainId: block.chainid}),
433:             payer_
434:         );
435:
436:         _route(
437:             Fee({asset: asset_, amount: collectionReferrerShare}),
438:             Target({target: referrer_, chainId: block.chainid}),
439:             payer_
440:         );
441:     }
```

Impact

Following are some of the negative impacts that could lead to a loss of assets:

1. Collection referrer will not receive their share of the minting fee.
2. Malicious minter of the tokens can set the `referrer_` parameter to their own wallet address when executing the `Edition.mint` function, so that the collection referral fee can be routed to their own wallet, effectively avoiding paying the collection referral fee to someone else.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/TitlesCore.sol#L103

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L125

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L412

Tool used

Manual Review

Recommendation

Consider the following change to ensure that the collection referral fee is routed to the collection referrer instead of the mint referrer.

```
function _splitProtocolFee(
    IEdition edition_,
    address asset_,
    uint256 amount_,
    address payer_,
    address referrer_
) internal returns (uint256 referrerShare) {
    // The creation and mint referrers earn 25% and 50% of the protocol's share
 ↪  respectively, if applicable
    uint256 mintReferrerShare = getMintReferrerShare(amount_, referrer_);
    uint256 collectionReferrerShare = getCollectionReferrerShare(amount_,
 ↪  referrers[edition_]);
    referrerShare = mintReferrerShare + collectionReferrerShare;
..SNIP..
    _route(
        Fee({asset: asset_, amount: collectionReferrerShare}),
-        Target({target: referrer_, chainId: block.chainid}),
```

```
+         Target({target: referrers[edition_], chainId: block.chainid}),
          payer_
      );
  }
```

Discussion

pqseags

Valid, will fix

midori-fuse

Escalate

#72, #131, #153, #158, #176, #183, and #193 are valid excluded dupes of this issue

There are also a number of other separately escalated issues (including another esc on this issue itself) that I agree are valid dupes of this. Though I can only try so hard at mentioning them :)

sherlock-admin3

> Escalate
>
> #72, #131, #153, #158, #176, #183, and #193 are valid excluded dupes of this issue
>
> There are also a number of other separately escalated issues (including another esc on this issue itself) that I agree are valid dupes of this. Though I can only try so hard at mentioning them :)

You've created a valid escalation!

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.

sherlock-admin3

> Escalate
>
> #393 is a valid excluded dup of this issue.

You've deleted an escalation for this issue.

MageIntern

Escalate

https://github.com/sherlock-audit/2024-04-titles-judging/issues/210 should be dup

sherlock-admin3

> Escalate
>
> https://github.com/sherlock-audit/2024-04-titles-judging/issues/210 should be dup

You've created a valid escalation!

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.

WangSecurity

For issues to become duplicates, they have to be escalated, not the main report (i.e. for #210 to become a dup, you should escalate it and not the main report). Hence, planning to reject this escalation

MageIntern

> For issues to become duplicates, they have to be escalated, not the main report (i.e. for #210 to become a dup, you should escalate it and not the main report). Hence, planning to reject this escalation

What should I do? Escalation is over.

midori-fuse

> For issues to become duplicates, they have to be escalated, not the main report (i.e. for #210 to become a dup, you should escalate it and not the main report). Hence, planning to reject this escalation

@WangSecurity quoting escalation rule 2

https://docs.sherlock.xyz/audits/judging/escalation-period#rules-for-escalation

> You can combine multiple arguments related to the same issue into one escalation. This prevents getting a double penalty when the escalation is rejected. For example, you might argue that the issue is valid and should be duplicated with #4. Also, you can mention multiple issues in the same escalation if they need to be duplicated together or separated from the context of the issue you are escalating. For example: 'The issues #12, #145 and #5 are all duplicates of the above issue'

My escalation is then valid and should be considered

WangSecurity

Sorry for the confusion I cause above, I saw the message that the escalation was deleted and mistakenly though it's about @midori-fuse escalation and then there was a new escalation, which I thought was the only one and asked for a single

issue to be a duplicate. Excuse me for the confusion, hope for your understanding. Planning to accept both escalations.

Evert0x

Result: High Has Duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- midori-fuse: accepted
- MageIntern: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-1: Incorrect encoding of bytes for EIP712 digest in `TitleGraph` causes signatures generated by common EIP712 tools to be unusable

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/74

Found by

0x73696d616f, T1MOH, ZanyBonzy, ast3ros, fugazzi, mt030d

Summary

The signature in `TitleGraph.acknowledgeEdge()` and `TitleGraph.unacknowledgeEdge()` is generated based on a digest computed from `edgeId` and `data`. However, the `data` bytes argument is not correctly encoded according to the EIP712 specification. Consequently, a signature generated using common EIP712 tools would not pass validation in `TitleGraph.checkSignature()`.

Vulnerability Detail

According to EIP712:

> The dynamic values bytes and string are encoded as a keccak256 hash of their contents.

```
modifier checkSignature(bytes32 edgeId, bytes calldata data, bytes calldata
↪  signature) {
    bytes32 digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH, edgeId,
↪  data)));
    ...
}
```

However, the `checkSignature()` modifier in the `TitlesGraph` contract reconstructs the digest by encoding the data bytes argument without first applying keccak256 hashing. As a result, a signature generated using common EIP712 tools (e.g. using the `signTypedData` function from `ethers.js`) would not pass validation in `TitleGraph.checkSignature()`.

POC

1. EIP712 signature computed by using ethers.js

```
// main.js
const { ethers } = require("ethers");
```

```
async function main() {
    const pk =
↪   "0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80";
    const signer = new ethers.Wallet(pk);
    const domain = {
      name: "TitlesGraph",
      version: '1',
      chainId: 31337,
      verifyingContract: "0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496" // should
↪   match the address in foundry test
    };
    const types = {
      Ack: [
        { name: "edgeId", type: "bytes32" },
        { name: "data", type: "bytes" },
      ],
    };
    const value = {
        edgeId: ethers.id("test edgeId"),
        data: "0xabcd"
    };
    const signature = await signer.signTypedData(domain, types, value);
    console.log(signature);

}

main();
```

here we run

```
npm install ethers
node main.js
```

The output is
0xab4623a7bacf25ed3d6779684f195ed63a5ed1ed46c278c107390086e74b739b35f1db213c6075dedc041d

2. EIP712 signature computed using foundry

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {EIP712} from "lib/solady/src/utils/EIP712.sol";

contract EIP712Test is Test, EIP712 {
```

```
    bytes32 public constant ACK_TYPEHASH = keccak256("Ack(bytes32 edgeId,bytes
↪   data)");

    // test data
    bytes32 testEdgeId = keccak256("test edgeId");
    bytes testData = hex"abcd";


    function test_sig() public {
        uint256 pk =
↪   0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80;
        bytes32 digest = _computeDigest(testEdgeId, testData);
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(pk, digest);
        bytes memory signature = abi.encodePacked(r, s, v);
        console.logBytes(signature);
    }

    function test_sigShouldBe() public {
        uint256 pk =
↪   0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80;
        bytes32 digest = _computeDigestShouldBe(testEdgeId, testData);
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(pk, digest);
        bytes memory signature = abi.encodePacked(r, s, v);
        console.logBytes(signature);
    }

    function _computeDigest(bytes32 edgeId, bytes memory data) internal returns
↪   (bytes32 digest) {
        digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH, edgeId,
↪   data)));
    }

    function _computeDigestShouldBe(bytes32 edgeId, bytes memory data) internal
↪   returns (bytes32 digest) {
        digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH, edgeId,
↪   keccak256(data))));
    }

    function _domainNameAndVersion()
        internal
        pure
        override
        returns (string memory name, string memory version)
    {
        name = "TitlesGraph";
        version = "1";
    }
```

```
}
```

here we run

```
forge test --mc EIP712Test -vv
```

The output is

`test_sig()` simulates the way the digest is reconstructed in `TitleGraph.checkSignature()`, while `test_sigShouldBe()` shows how the digest should be reconstructed. From the above output, we can see the signature generated by ethers.js matches the signature generated in `test_sigShouldBe()` and does not match the signature generated in `test_sig()`. This PoC shows the way `TitleGraph.checkSignature()` reconstruct the digest is not compatible with the way data is encoded in EIP712.

Impact

A signature generated by the signer using common EIP712 tools (e.g. signTypedData in `ethers.js`) would not pass validation in `TitleGraph.checkSignature()`.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L41

Tool used

Manual Review, ethers.js, foundry

Recommendation

Encoding the `data` bytes as a keccak256 hash of its contents before computing the digest from it:

```
- digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH, edgeId, data)));
+ digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH, edgeId,
↪  keccak256(data))));
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-2: CREATE opcode works differently in the zkSync chain

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/91

The protocol has acknowledged this issue.

Found by

ArsenLupin, Kalogerone, me_na0mi

Summary

`zkSync Era` chain has differences in the usage of the `create` opcode compared to the EVM.

Vulnerability Detail

According to the contest README, the protocol can be deployed in zkSync Era (README)

The zkSync Era docs explain how it differs from Ethereum.

The description of CREATE and CREATE2 (zkSynce Era Docs) states that Create cannot be used for arbitrary code unknown to the compiler.

`TitlesCore::createEdition` function uses Solady's `LibClone::clone` function:

```
    function createEdition(bytes calldata payload_, address referrer_) external
↪   payable returns (Edition edition) {

        EditionPayload memory payload = abi.decode(payload_.cdDecompress(),
↪   (EditionPayload));

@>      edition = Edition(editionImplementation.clone());

        // wake-disable-next-line reentrancy
        edition.initialize(
            feeManager, graph, payload.work.creator.target, address(this),
↪   payload.metadata
        );

        // wake-disable-next-line unchecked-return-value
        _publish(edition, payload.work, referrer_);

        emit EditionCreated(
            address(edition),
```

```
            payload.work.creator.target,
            payload.work.maxSupply,
            payload.work.strategy,
            abi.encode(payload.metadata)
        );
    }
```

```
    function clone(uint256 value, address implementation) internal returns
↪   (address instance) {
        /// @solidity memory-safe-assembly
        assembly {
            mstore(0x21, 0x5af43d3d93803e602a57fd5bf3)
            mstore(0x14, implementation)
            mstore(0x00, 0x602c3d8160093d39f33d3d3d3d363d3d37363d73)
@>          instance := create(value, 0x0c, 0x35)
            if iszero(instance) {
                mstore(0x00, 0x30116425) // `DeploymentFailed()`.
                revert(0x1c, 0x04)
            }
            mstore(0x21, 0) // Restore the overwritten part of the free memory
↪   pointer.
        }
    }
```

As mentioned by the zkSync docs: "The code will not function correctly because the compiler is not aware of the bytecode beforehand".

This will result in **loss of funds**, since there is a fee to create a new edition, hence the `TitlesCore::createEdition` function is payable.

Impact

No editions can be created in the zkSync Era chain.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/d7f60952df22da00b772db5d3a8272a988546089/wallflower-contract-v2/src/TitlesCore.sol#L79
https://github.com/Vectorized/solady/blob/91d5f64b39a4d20a3ce1b5e985103b8ea4dc1cfc/src/utils/LibClone.sol#L137

Tool used

Manual Review

Recommendation

This can be solved by implementing CREATE2 directly and using `type(Edition).creationCode` as mentioned in the zkSync Era docs.

Discussion

midori-fuse

Escalate

This escalation raises two points:

The first point is that #353 and #440 are duplicates of this issue.

The second point is that this issue has been excluded without reasoning (nor do I see any signs of someone reviewing this issue), even though evidence has been provided, and the impacts have been shown:

- The relevant part of the zkEVM docs has been linked within the report.
- The contract is bricked for this issue, which fits into the "rendering the contract useless" in the medium severity criteria.
- The contest README includes zkSync as a deployment chain.

For this reason I believe this issue (and the mentioned duplicates) to be a valid medium.

sherlock-admin3

> Escalate
>
> This escalation raises two points:
>
> The first point is that #353 and #440 are duplicates of this issue.
>
> The second point is that this issue has been excluded without reasoning (nor do I see any signs of someone reviewing this issue), even though evidence has been provided, and the impacts have been shown:
>
> - The relevant part of the zkEVM docs has been linked within the report.
> - The contract is bricked for this issue, which fits into the "rendering the contract useless" in the medium severity criteria.
> - The contest README includes zkSync as a deployment chain.
>
> For this reason I believe this issue (and the mentioned duplicates) to be a valid medium.

You've created a valid escalation!

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.

cvetanovv

I disagree with the escalation. This issue is invalid.

You can see Sherlock documentation subclause 24. EVM opcodes type of issue is not valid.

midori-fuse

The clause states:

> Using Solidity versions that support EVM opcodes that don't work on networks on which the protocol is deployed is not a valid issue because one can manage compilation flags to compile for past EVM versions on newer Solidity versions.

This is not an issue of "Using solidity versions", because all solidity versions support the CREATE opcode, and this issue will manifest no matter what Solidity version you use, or what EVM you compile it on. The clause clearly states that the rule only applies to using Solidity versions such that it is mitigatable by managing compilation flags, it does not apply to all "EVM opcodes type" in general.

Any issue also boils down to just "EVM opcode", because when you compile any Solidity source files, you always get EVM opcodes, the majority (if not, most of the time, all) of which you cannot manage compilation flags to the behavior you desire.

Kalogerone

I agree with @midori-fuse. Subclause 24 doesn't refer to these kind of issues. This issue is not about the solidity version used. Subclause 24 clearly covers the solution which cannot be applied here:

> because one can manage compilation flags to compile for past EVM versions on newer Solidity versions.

cvetanovv

@midori-fuse and @Kalogerone, I agree with the comments and will consult with @WangSecurity about whether it is a valid issue.

Hash01011122

Agreed with @midori-fuse on

> The clause clearly states that the rule only applies to using Solidity versions such that it is mitigatable by managing compilation flags, it does not apply to all "EVM opcodes type" in general.

But still this appears to be low. I will let head of judge decide severity of this issue.

Kalogerone

I would argue it is not a low because as I mentioned in my report, the function used to create a new contract is payable and the user has to pay a fee to do that. Zksync docs never mention that the function will fail/revert, meaning that the user's funds are not safe and there is possible loss of funds.

> The following code will not function correctly because the compiler is not aware of the bytecode beforehand
>
> Unfortunately, it's impossible to differentiate between the above cases during compile-time. As a result, we strongly recommend including tests for any factory that deploys child contracts using type(T).creationCode.

WangSecurity

I agree with the escalation and that it should be a valid medium, rather than the high. Planning to accept the escalation and validate as medium with duplicates #353 and #440.

Evert0x

Result: Medium Has Duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- midori-fuse: accepted

Issue M-3: ERC2981 royalties discrepancy with strategy

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/144

Found by

ComposableSecurity, cducrest-brainbot, sammy

Summary

In `Edition.sol`, functions that set the value of `works[tokenId].strategy` which includes `works[tokenId].strategy.royaltyBps` do not set ERC2981's internal token royalty value.

Vulnerability Detail

The function `setFeeStrategy()` sets the public mapping value `works[tokenId_].strategy` which may update the `roylatyBps` value but does not call `_setTokenRoyalty(...)`:

```
function setFeeStrategy(uint256 tokenId_, Strategy calldata strategy_) external {
    if (msg.sender != works[tokenId_].creator) revert Unauthorized();
    works[tokenId_].strategy = FEE_MANAGER.validateStrategy(strategy_);  //
↪   @audit does not set royalties
}
```

Similarly, the `publish()` function to create a new work sets the strategy but does not call `_setTokenRoyalty()`:

```
function publish(
    address creator_,
    uint256 maxSupply_,
    uint64 opensAt_,
    uint64 closesAt_,
    Node[] calldata attributions_,
    Strategy calldata strategy_,
    Metadata calldata metadata_
) external override onlyRoles(EDITION_MANAGER_ROLE) returns (uint256 tokenId) {
    tokenId = ++totalWorks;

    _metadata[tokenId] = metadata_;
    works[tokenId] = Work({
        creator: creator_,
        totalSupply: 0,
        maxSupply: maxSupply_,
```

```
            opensAt: opensAt_,
            closesAt: closesAt_,
            strategy: FEE_MANAGER.validateStrategy(strategy_)
        });

        Node memory _node = node(tokenId);
        for (uint256 i = 0; i < attributions_.length; i++) {
            // wake-disable-next-line reentrancy, unchecked-return-value
            GRAPH.createEdge(_node, attributions_[i], attributions_[i].data);
        }

        emit Published(address(this), tokenId);
    }
```

This latter case may be less of a problem since `TitlesCore` calls `edition_.setRoyaltyTarget()` right after publishing a new work. However it remains a problem if publishers are expected to interact directly with `Edition` and not only through `TitlesCore`

Impact

The value returned by `works[tokenId].strategy.royaltyBps` and `ERC2981.royaltyInfo(tokenId, salePrice)` will not be coherent. Users may expect to set a certain royalty bps while the value is not updated. Core values used for royalty payments may become incorrect after updates.

Code Snippet

https://github.com/vectorized/solady/blob/main/src/tokens/ERC2981.sol

https://github.com/sherlock-audit/2024-04-titles/blob/d7f60952df22da00b772db5d3a8272a988546089/wallflower-contract-v2/src/editions/Edition.sol#L368-L371

https://github.com/sherlock-audit/2024-04-titles/blob/d7f60952df22da00b772db5d3a8272a988546089/wallflower-contract-v2/src/editions/Edition.sol#L121

Tool used

Manual Review

Recommendation

Call `_setTokenRoyalty(tokenId, FeeManager.feeReceiver(address(this), tokenId), works[tokenId].strategy.royaltyBps);` at the end of `setFeeStrategy()`.

Discussion

pqseags

This is valid and should be fixed

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-4: FeeManager's admin cannot grant or revoke any role

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/148

Found by

0x73696d616f, KupiaSec, fibonacci, fugazzi, juan, sammy

Summary

The `FeeManager` contract lacks an interface for the admin to grant or revoke roles.

Vulnerability Detail

The contest's README states:

```
ADMIN_ROLE
...
On FeeManager, this role can:
...
3) Grant or revoke any role to/from any address (`grantRole`, `revokeRole`).
```

The `FeeManager` contract, inherited from solady's `OwnableRoles` contract, only permits the owner to manage roles.

```
/// @dev Allows the owner to grant `user` `roles`.
/// If the `user` already has a role, then it will be a no-op for the role.
function grantRoles(address user, uint256 roles) public payable virtual
↪   onlyOwner {
    _grantRoles(user, roles);
}

/// @dev Allows the owner to remove `user` `roles`.
/// If the `user` does not have a role, then it will be a no-op for the role.
function revokeRoles(address user, uint256 roles) public payable virtual
↪   onlyOwner {
    _removeRoles(user, roles);
}
```

The `FeeManager` contract itself does not provide any interfaces that enable the admin to grant or revoke roles.

Impact

The admin cannot grant or revoke any roles. This limitation cannot be changed since the contract is not upgradable. A new role can only be granted or revoked by the owner, which is the `TitlesCore` contract, and it also lacks the corresponding functions.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/README.md#q-are-there-any-protocol-roles-please-list-them-and-provide-whether-they-are-trusted-or-restricted-or-provide-a-more-comprehensive-description-of-what-a-role-can-and-cant-doimpact https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/fees/FeeManager.sol#L115

Tool used

Manual Review

Recommendation

Implement admin interfaces to manage roles

```
function grantRoles(address guy, uint256 roles)
    public
    payable
    override
    onlyOwnerOrRoles(ADMIN_ROLE)
{
    _grantRoles(guy, roles);
}

function revokeRoles(address guy, uint256 roles)
    public
    payable
    override
    onlyOwnerOrRoles(ADMIN_ROLE)
{
    _removeRoles(guy, roles);
}
```

Discussion

0xf1b0

Escalate

I believe this is a valid issue. The implementation does not align with what is described in the README.

sherlock-admin3

> Escalate
>
> I believe this is a valid issue. The implementation does not align with what is described in the README.

You've created a valid escalation!

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.

WangSecurity

Agree with escalation that it's valid, but I believe it could be spotted on deployment and fixed with re-deploy without losing funds. Moreover, the roles can still be set and revoked. Hence, believe medium is approriate here, planning to accept the escalation.

WangSecurity

Again, agree with escalation and making it a valid medium. Planning to make a new family of issues with the core issue "roles are set incorrectly or not set at all" with the following duplicates:

#166, #197, #213, #146.

0xjuaan

hi @WangSecurity, #240 is a dupe of this

Evert0x

Result: Medium Has Duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 0xf1b0: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-5: TitlesGraph::acknowledgeEdge() methods do not write acknowledgments to storage

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/212

Found by

4rdiii, AlexCzm, CodeWasp, KupiaSec, alexzoid, brakeless, cducrest-brainbot, den_sosnovskyi, fugazzi, i3arba, ironside, kennedy1030, radin200, recursiveEth, ubl4nk

Summary

When `acknowledgeEdge()` is called, the downstream call to the `_setAcknowledged()` method caches `edges[edgeId_]` in memory, instead of storage, which does not preserve changes to the `Edge` struct after the transaction concludes.

Vulnerability Detail

```
function acknowledgeEdge(bytes32 edgeId_, bytes calldata data_)
    external
    override
    returns (Edge memory edge)
{
    if (!_isCreatorOrEntity(edges[edgeId_].to, msg.sender)) revert
↪   Unauthorized();
    return _setAcknowledged(edgeId_, data_, true);
}

function _setAcknowledged(bytes32 edgeId_, bytes calldata data_, bool
↪   acknowledged_)
    internal
    returns (Edge memory edge)
{
    if (!_edgeIds.contains(edgeId_)) revert NotFound();
    edge = edges[edgeId_];
    edge.acknowledged = acknowledged_;    // @audit does this actually write to
↪   storage? the state isn't saved?

    if (acknowledged_) {
        emit EdgeAcknowledged(edge, msg.sender, data_);
    } else {
        emit EdgeUnacknowledged(edge, msg.sender, data_);
    }
```

```
}
```

In the code snippet above, `edges[edgeId_]` is cached in `edge`, then modified.

The issue is that `edge`, the return variable, is marked as memory, which does not save the state after the transaction ends.

Therefore, the `acknowledged` value never changes and the `acknowledgeEdge()` methods do not work as intended.

Add the following test to `TitlesGraph.t.sol`.

Run with the following command: `forge test --match-test test_acknowledgeEdgeFailure -vvvv`

```
function test_acknowledgeEdgeFailure() public {
    Node memory from = Node({
        nodeType: NodeType.COLLECTION_ERC1155,
        entity: Target({target: address(1), chainId: block.chainid}),
        creator: Target({target: address(2), chainId: block.chainid}),
        data: ""
    });

    Node memory to = Node({
        nodeType: NodeType.TOKEN_ERC1155,
        entity: Target({target: address(3), chainId: block.chainid}),
        creator: Target({target: address(4), chainId: block.chainid}),
        data: abi.encode(42)
    });

    // Only the `from` node's entity can create the edge.
    vm.prank(from.entity.target);
    titlesGraph.createEdge(from, to, "");

    vm.expectEmit(true, true, true, true);
    emit IEdgeManager.EdgeAcknowledged(
        Edge({from: from, to: to, acknowledged: true, data: ""}),
↪   to.creator.target, ""
    );

    // Only the `to` node's creator (or the entity itself) can acknowledge it
    vm.prank(to.creator.target);
    titlesGraph.acknowledgeEdge(keccak256(abi.encode(from, to)), "");

    (Node memory nodeTo,
        Node memory nodeFrom,
        bool ack,
        bytes memory dataResult
```

```
    ) = titlesGraph.edges(titlesGraph.getEdgeId(from, to));
    console.log(ack);

    // edge[edgeId].acknowledged should be set to true after successful call to
↪   titlesGraph.acknowledgeEdge()
    // However, the value is still false.
    assert(true == ack);
}
```

```
[FAIL. Reason: panic: assertion failed (0x01)] test_acknowledgeEdgeFailure()
↪   (gas: 429897)
Logs:
  false
```

Impact

Edges are unable to be acknowledged.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L103-L124

Tool used

Manual Review

Recommendation

The simplest fix would be to change `memory` to `storage`.

```
function _setAcknowledged(bytes32 edgeId_, bytes calldata data_, bool
↪   acknowledged_)
     internal
-    returns (Edge memory edge)
+    returns (Edge storage edge)
 {
     if (!_edgeIds.contains(edgeId_)) revert NotFound();
     edge = edges[edgeId_];
     edge.acknowledged = acknowledged_;

     if (acknowledged_) {
         emit EdgeAcknowledged(edge, msg.sender, data_);
     } else {
         emit EdgeUnacknowledged(edge, msg.sender, data_);
```


```
    }
}
```

Discussion

pqseags

This will be fixed

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-6: Minting can be DOSed by any of the fee recipients

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/261

The protocol has acknowledged this issue.

Found by

0rpse, ComposableSecurity, xiaoming90

Summary

The minting process might be DOS by malicious fee recipients, breaking the protocol's core functionality. This would also result in the loss of fee for the rest of the innocent fee recipients as the minting cannot proceed, resulting in the minting fee not being collected.

Vulnerability Detail

When minting a new token for a given work, the minting fee will be collected as per Line 236 below.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L236

```
File: Edition.sol
222:     /// @notice Mint a new token for the given work.
223:     /// @param to_ The address to mint the token to.
224:     /// @param tokenId_ The ID of the work to mint.
225:     /// @param amount_ The amount of tokens to mint.
226:     /// @param referrer_ The address of the referrer.
227:     /// @param data_ The data associated with the mint. Reserved for future
↪  use.
228:     function mint(
229:         address to_,
230:         uint256 tokenId_,
231:         uint256 amount_,
232:         address referrer_,
233:         bytes calldata data_
234:     ) external payable override {
235:         // wake-disable-next-line reentrancy
236:         FEE_MANAGER.collectMintFee{value: msg.value}(
237:             this, tokenId_, amount_, msg.sender, referrer_,
↪  works[tokenId_].strategy
238:         );
```

```
239:
240:         _issue(to_, tokenId_, amount_, data_);
241:         _refundExcess();
242:     }
```

The collected minting fee will be routed or transferred to one or more of the following parties directly (Using a push mechanism):

1. Edition's creator (Using 0xSplit wallet)
2. Edition's attributions (no limit on the number of attributions in the current setup) (Using 0xSplit wallet)
3. Collection referrer
4. Minting referrer
5. Protocol fee receiver

This approach introduced the risk of DOS to the minting process. As long as any of the parties above intentionally revert upon receiving the ETH minting fee, they could DOS the entire minting process of work.

Note: The parties using the 0xSplit wallet are not an issue because the wallet requires the recipients to claim the fees from the 0xSplit wallet manually (Using the pull mechanism instead)

Impact

The minting process might be DOS by malicious fee recipients, breaking the protocol's core functionality. This would also result in the loss of fee for the rest of the innocent fee recipients as the minting cannot proceed, resulting in the minting fee not being collected.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L236

Tool used

Manual Review

Recommendation

Consider adopting a pull mechanism for the fee recipients to claim their accrued fee instead of transferring the fee directly to them during the minting process.

Discussion

xiaoming9090

Escalate

The two (2) actors mentioned in this report, the "Collection referrer" and "Minting referrer," are external parties of the protocol that are not fully trusted. When these two external actors behave maliciously, they could negatively impact the other innocent parties (Edition's creator, Edition's attributions, Protocol fee receiver), including the protocol, by preventing them from receiving their fee, resulting in a loss of assets for the victims.

As such, this issue should not be overlooked due to its risks and should be considered valid.

sherlock-admin3

> Escalate
>
> The two (2) actors mentioned in this report, the "Collection referrer" and "Minting referrer," are external parties of the protocol that are not fully trusted. When these two external actors behave maliciously, they could negatively impact the other innocent parties (Edition's creator, Edition's attributions, Protocol fee receiver), including the protocol, by preventing them from receiving their fee, resulting in a loss of assets for the victims.
>
> As such, this issue should not be overlooked due to its risks and should be considered valid.

You've created a valid escalation!

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.

WangSecurity

Agree with the escalation, even though it's pure griefing and the referrers lose fees themselves, the protocol also loses fee. Planning to accept and validate the report.

Evert0x

Result: Medium Has Duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- xiaoming9090: accepted

Issue M-7: Excess ETH will be stuck in the Fee Manager contract and not swept back to the users

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/269

Found by

0rpse, 0x486776, 0xboriskataa, 1337, AlexCzm, ComposableSecurity, Dots, FastTiger, KupiaSec, Shaheen, Varun_05, Yu3H0, ZdravkoHr., alexzoid, ast3ros, cducrest-brainbot, cryptic, cu5t0mPe0, fugazzi, funkornaut, gladiator111, jecikpo, jennifer37, joicygiore, juan, kennedy1030, kn0t, merlinboii, mt030d, radin200, smbv-1923, techOptimizor, trachev, ubl4nk, valentin2304, xiaoming90, y4y, ydlee, zoyi

Summary

Excess ETH will be stuck in the Fee Manager contract and not swept back to the users.

Vulnerability Detail

Per Line 241 below, it is expected that there will be excess ETH residing on the contract at the end of the transaction. The `_refundExcess` function is implemented with the intention of sweeping excess ETH back to the caller of the `mint` function at the end of the transaction.

Assume Bob transfers 0.05 ETH to the Edition contract, but the minting fee ends up being only 0.03 ETH. The _refundExcess function at the end of the function (Line 241 below) is expected to return the excess 0.02 ETH back to Bob.

However, it was found that such a design does not work. When the `collectMintFee` function is executed on Line 236 below, the entire amount of ETH (0.05 ETH) will be forwarded to the Fee Manager contract. 0.03 ETH out of 0.05 ETH will be forwarded to the fee recipients, while the remaining 0.02 will be stuck in the Fee Manager contract. The excess 0.02 is not being returned to the Edition contract. Thus, when the `_refundExcess` function is triggered at the end of the function, no ETH will be returned to Bob.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L228

```
File: Edition.sol
228:     function mint(
229:         address to_,
230:         uint256 tokenId_,
```

```
231:         uint256 amount_,
232:         address referrer_,
233:         bytes calldata data_
234:     ) external payable override {
235:         // wake-disable-next-line reentrancy
236:         FEE_MANAGER.collectMintFee{value: msg.value}(
237:             this, tokenId_, amount_, msg.sender, referrer_,
 ↪  works[tokenId_].strategy
238:         );
239:
240:         _issue(to_, tokenId_, amount_, data_);
241:         _refundExcess();
242:     }
```

> [!IMPORTANT]
>
> This issue also affects the `Edition.mintWithComment` and `Edition.mintBatch` functions. The write-up is the same and is omitted for brevity.

In addition, the Contest's README mentioned that the protocol aims to aims to avoid any direct TVL in this release:

> Please discuss any design choices you made.
>
> Fund Management: We chose to delegate fee payouts to 0xSplits v2. The protocol aims to avoid any direct TVL in this release.

In other words, this means that no assets should be locked within the protocol. However, as shown in the earlier scenario, some assets will be stored in the Fee Manager, breaking this requirement.

Impact

Excess ETH will be stuck in the Fee Manager contract and not swept back to the users.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L228

Tool used

Manual Review

Recommendation

Consider forwarding only the required amount of the minting fee to the Fee Manager, so that any excess ETH can be sweeped by the `_refundExcess()` function at the end of the transaction.

```
function mint(
    address to_,
    uint256 tokenId_,
    uint256 tokenId_,
    address referrer_,
    bytes calldata data_
) external payable override {
+       uint256 mintFee = FEE_MANAGER.getMintFee(this, tokenId_, amount_);

    // wake-disable-next-line reentrancy
-   FEE_MANAGER.collectMintFee{value: msg.value}(
+   FEE_MANAGER.collectMintFee{value: mintFee}(
        this, tokenId_, amount_, msg.sender, referrer_, works[tokenId_].strategy
    );

    _issue(to_, tokenId_, amount_, data_);
    _refundExcess();
}
```

Discussion

pqseags

Will address

WangSecurity

Firstly, the `mintFee` is set by the publisher/creator of the work and can be changed at any time. It's safe to assume it's highly unlikely that they will change it every (few) block(s). Moreover, Edition contract has to functions to get the `mintFee` here and here. Hence, the user may know exactly how much `msg.value` the need to send.

However, it doesn't exclude the situation when the `mintFee` is change in the exact same block right before the user mints, resulting in the revert. But the user can mitigate it on their side: call `mintFee` and `mint` in the same call.

Therefore, I believe medium severity is more appropriate here, planning to downgrade the issue.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:

https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-8: Malicious users can block creators from acknowledging or deacknowledging an edge

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/273

The protocol has acknowledged this issue.

Found by

0x73696d616f, 1337, Bigsam, ComposableSecurity, Krace, KupiaSec, Yu3H0, alexzoid, cryptic, fibonacci, fugazzi, jecikpo, jennifer37, juan, kennedy1030, mt030d, xiaoming90, zoyi

Summary

Malicious users can block someone from acknowledging or deacknowledging an edge, affecting the sanctity of the data in the Graph.

Vulnerability Detail

> [!IMPORTANT]
>
> The following is an extract from the contest's README:
>
> > Additional audit information.
> >
> > In addition to the security of funds, we would also like there to be focus on the sanctity of the data in the TitlesGraph and the permissioning around it (only the appropriate people/contracts can signal reference and acknowledgement of reference).
>
> The contest's README stated that apart from the loss of assets, the protocol team would like there to be a focus on the sanctity of the data. Thus, any issues related to the sanctity of the data in the Graph would be considered a valid Medium issue in the context of this audit contest, as the Contest's README supersede Sherlock's judging rules per Sherlock's Hierarchy of truth.

Both `acknowledgeEdge` and `unacknowledgeEdge` functions rely on the same modifier (`checkSignature`) to verify the signature validity. Thus, the signature used for acknowledgment and deacknowledgment of an edge follows the same format and can be used interchangeably. However, this design creates an issue, as described next.

1. Assume that Bob wants to acknowledge an edge. Thus, Bob calls the `acknowledgeEdge` function with his signature Sig_1.

2. A malicious user can always front-run Bob, take his signature (Sig_1) and sent to the `unacknowledgeEdge` function instead. When the `unacknowledgeEdge` function is executed with Sig_1, the signature will be marked as used by the code `_isUsed[keccak256(signature)] = true;`.
3. When Bob's `acknowledgeEdge` transaction gets executed, it will revert because his signature (Sig_1) has already been used.

The malicious users can keep repeating as the chain's gas fees on L2 are cheap.

The same trick can also be used to block someone from deacknowledge an edge.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L118

```
File: TitlesGraph.sol
118:     function acknowledgeEdge(bytes32 edgeId_, bytes calldata data_, bytes
↪  calldata signature_)
119:         external
120:         checkSignature(edgeId_, data_, signature_)
121:         returns (Edge memory edge)
122:     {
123:         return _setAcknowledged(edgeId_, data_, true);
124:     }
```

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L146

```
File: TitlesGraph.sol
146:     function unacknowledgeEdge(bytes32 edgeId_, bytes calldata data_, bytes
↪  calldata signature_)
147:         external
148:         checkSignature(edgeId_, data_, signature_)
149:         returns (Edge memory edge)
150:     {
151:         return _setAcknowledged(edgeId_, data_, false);
152:     }
```

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L40

```
File: TitlesGraph.sol
39:     /// @notice Modified to check the signature for a proxied acknowledgment.
40:     modifier checkSignature(bytes32 edgeId, bytes calldata data, bytes
↪  calldata signature) {
41:         bytes32 digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH,
↪  edgeId, data)));
```

```
42:         if (
43:
↪   !edges[edgeId].to.creator.target.isValidSignatureNowCalldata(digest,
↪   signature)
44:                 || _isUsed[keccak256(signature)]
45:         ) {
46:             revert Unauthorized();
47:         }
48:         _;
49:         _isUsed[keccak256(signature)] = true;
50:     }
```

Impact

Malicious users can block someone from acknowledging or deacknowledging an edge, affecting the sanctity of the data in the Graph.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L118

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L146

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L40

Tool used

Manual Review

Recommendation

```
bytes32 public constant ACK_TYPEHASH = keccak256("Ack(bytes32 edgeId,bytes
↪   data)");
bytes32 public constant DEACK_TYPEHASH = keccak256("Deack(bytes32 edgeId,bytes
↪   data)");

bytes32 digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH, edgeId,
↪   data)));
bytes32 digest = _hashTypedData(keccak256(abi.encode(DEACK_TYPEHASH, edgeId,
↪   data)));
```

Consider using two different hash types for acknowledging or deacknowledging within the signature and use a different modifier for checking the signature. This will prevent malicious users from taking the signature intended for `acknowledgeEdge` and submitting it to `unacknowledgeEdge`, and vice versa.

```
modifier checkSignatureForAck(bytes32 edgeId, bytes calldata data, bytes
↪  calldata signature) {
    bytes32 digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH, edgeId,
↪  data)));
    ..SNIP..
}

modifier checkSignatureForDeack(bytes32 edgeId, bytes calldata data, bytes
↪  calldata signature) {
    bytes32 digest = _hashTypedData(keccak256(abi.encode(DEACK_TYPEHASH, edgeId,
↪  data)));
    ..SNIP..
}
```

```
function acknowledgeEdge(bytes32 edgeId_, bytes calldata data_, bytes calldata
↪  signature_)
    external
-   checkSignature(edgeId_, data_, signature_)
+   checkSignatureForAck(edgeId_, data_, signature_)
    returns (Edge memory edge)
{

function unacknowledgeEdge(bytes32 edgeId_, bytes calldata data_, bytes calldata
↪  signature_)
    external
-   checkSignature(edgeId_, data_, signature_)
+       checkSignatureForDeack(edgeId_, data_, signature_)
    returns (Edge memory edge)
{
```

With this design, if a creator creates a signature intended for `acknowledgeEdge` function, and a malicious user front-runs and copies the signature and submits it to `acknowledgeEdge` function, no harm is done as the malicious user is simply executing the task on behalf of the creator. The edge will still be acknowledged at the end.

Discussion

pqseags

Will investigate

Issue M-9: Signature is malleable

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/279

The protocol has acknowledged this issue.

Found by

0x73696d616f, 1337, ComposableSecurity, T1MOH, Yu3H0, ZdravkoHr., cducrest-brainbot, fibonacci, juan, xiaoming90, zigtur

Summary

The signature is malleable. When a signature is malleable, it means that it is possible to produce another valid signature for the same message (which also means the same digest).

If the intention is only to allow the creator to acknowledge an edge once, the creator can bypass this restriction because the signature is malleable, and the creator can create another signature to acknowledge an edge again.

Vulnerability Detail

The protocol relies on Solady's SignatureCheckerLib to verify that the signature provided is valid. Once a signature has been successfully verified, it will be marked as used in Line 49 below to prevent users from re-using or replaying the same signature.

The Solady's SignatureCheckerLib warns that the library does not check if a signature is non-malleable.

https://github.com/Vectorized/solady/blob/a34977e56cc1437b7ac07e6356261d2b303da686/src/utils/SignatureCheckerLib.sol#L23

```
/// WARNING! Do NOT use signatures as unique identifiers:
/// - Use a nonce in the digest to prevent replay attacks on the same contract.
/// - Use EIP-712 for the digest to prevent replay attacks across different
↪  chains and contracts.
///   EIP-712 also enables readable signing of typed data for better user safety.
/// This implementation does NOT check if a signature is non-malleable.
library SignatureCheckerLib {
```

Based on the following code, a creator can only acknowledge an edge once because the digest of the signature to acknowledge an edge will be the same (assuming that `data` is not in use) every time. After a creator acknowledges an edge, the signature (which also means its digest) will be marked as used in Line 49

below, thus preventing the creator from acknowledging the edge again later using the same signature.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L49

```
File: TitlesGraph.sol
39:     /// @notice Modified to check the signature for a proxied acknowledgment.
40:     modifier checkSignature(bytes32 edgeId, bytes calldata data, bytes
↪  calldata signature) {
41:         bytes32 digest = _hashTypedData(keccak256(abi.encode(ACK_TYPEHASH,
↪  edgeId, data)));
42:         if (
43:
↪  !edges[edgeId].to.creator.target.isValidSignatureNowCalldata(digest,
↪  signature)
44:                 || _isUsed[keccak256(signature)]
45:         ) {
46:             revert Unauthorized();
47:         }
48:         _;
49:         _isUsed[keccak256(signature)] = true;
50:     }
```

Impact

When a signature is malleable, it means that it is possible to produce another valid signature for the same message (which also means the same digest).

If the intention is only to allow the creator to acknowledge an edge once, the creator can bypass this restriction because the signature is malleable, and the creator can create another signature to acknowledge an edge again.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/graph/TitlesGraph.sol#L49

Tool used

Manual Review

Recommendation

Consider verifying the `s` of the signature is within valid bounds to avoid signature malleability.

```
if (uint256(s) >
↪  0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
    revert("ECDSA: invalid signature 's' value");
}
```

Discussion

pqseags

This seems to be a duplicate of #273 @Hash01011122

WangSecurity

Planning to make a new issue family with this report as the main one. Planning to add the following duplicated:

- #9
- #53
- #105
- #178
- #228
- #345
- #429

0xjuaan

@WangSecurity add https://github.com/sherlock-audit/2024-04-titles-judging/issues/130, #10 to that list, they are currently incorrectly dupes of #273 which is a dos/frontrunning issue

Evert0x

> @WangSecurity add #130, #10 to that list, they are currently incorrectly dupes of #273 which is a dos/frontrunning issue

Thank you!

Issue M-10: Broken batch minting feature

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/280

Found by

0x486776, 0xShiki, 0xlucky, AgileJune, ArsenLupin, BiasedMerc, ComposableSecurity, Dots, FastTiger, lllllll, Kalogerone, KupiaSec, Shaheen, Varun_05, ZdravkoHr., alexzoid, ast3ros, azanux, cducrest-brainbot, cu5t0mPe0, ff, fugazzi, funkornaut, jennifer37, joicygiore, juan, kennedy1030, kn0t, mt030d, sammy, smbv-1923, trachev, xiaoming90, y4y, ydlee, zigtur

Summary

The core minting feature of the protocol is broken due to the mishandling of `msg.value` within the for-loop.

Vulnerability Detail

Assume that the total fee for each token is 0.001 ETH, and Bob wants to mint four tokens. The total fee will be 0.004 ETH, so he will send 0.004 ETH when calling the above `mintBatch` function.

An important point to note is that the `msg.value` will always remain at 0.004 ETH throughout the entire execution of the `mintBatch` function. The `msg.value` will not automatically be reduced regardless of how many ETH has been transferred out or "spent".

In the first for-loop, the `msg.value` will be 0.004 ETH, and all 0.004 ETH will be routed to the fee manager and subsequently routed to the fee recipient address/0xSplit wallet.

In the second for-loop, since all the ETH (0.004 ETH) was sent to the fee manager earlier, the amount of ETH left on the Edition contract is zero. When the second for-loop attempts to send `msg.value` (0.004 ETH) to the fee manager again, it will revert due to insufficient ETH, and the transaction will fail and revert. Thus, this batch minting feature is broken.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L277

```
File: Edition.sol
277:     function mintBatch(
278:         address to_,
279:         uint256[] calldata tokenIds_,
280:         uint256[] calldata amounts_,
```

```
281:         bytes calldata data_
282:     ) external payable {
283:         for (uint256 i = 0; i < tokenIds_.length; i++) {
284:             Work storage work = works[tokenIds_[i]];
285:
286:             // wake-disable-next-line reentrancy
287:             FEE_MANAGER.collectMintFee{value: msg.value}(
288:                 this, tokenIds_[i], amounts_[i], msg.sender, address(0),
    ↪  work.strategy
289:             );
290:
291:             _checkTime(work.opensAt, work.closesAt);
292:             _updateSupply(work, amounts_[i]);
293:         }
294:
295:         _batchMint(to_, tokenIds_, amounts_, data_);
296:         _refundExcess();
297:     }
```

Impact

Breaks core contract functionality. The batch minting feature, a core feature of the protocol, is broken.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L277

Tool used

Manual Review

Recommendation

For each loop, consider only forwarding/transferring the minting fee for the current token ID instead of the entire ETH (`msg.value`).

```
function mintBatch(
    address to_,
    uint256[] calldata tokenIds_,
    uint256[] calldata amounts_,
    bytes calldata data_
) external payable {
    for (uint256 i = 0; i < tokenIds_.length; i++) {
```

```
        Work storage work = works[tokenIds_[i]];
+               uint256 mintFee = FEE_MANAGER.getMintFee(work.strategy,
 ↪  amounts_[i])

        // wake-disable-next-line reentrancy
+        FEE_MANAGER.collectMintFee{value: mintFee}(
-        FEE_MANAGER.collectMintFee{value: msg.value}(
            this, tokenIds_[i], amounts_[i], msg.sender, address(0),
 ↪  work.strategy
        );
```

Discussion

pqseags

I don't think the duplicates on this issue are actually duplicates of this issue. @Hash01011122

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-11: New creators unable to update the royalty target and the fee route for their works

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/283

Found by

0x73696d616f, BengalCatBalu, BiasedMerc, CodeWasp, KupiaSec, Varun_05, alexzoid, cu5t0mPe0, jennifer37, mt030d, sammy, xiaoming90, y4y, zoyi

Summary

The new creators are unable to update the royalty target and the fee route for their works. As a result, it could lead to a loss of assets for the new creator due to the following:

- The work's royalty target still points to the previous creator, so the royalty fee is routed to the previous creator instead of the current creator.
- The work's fee is still routed to the recipients, which included the previous creator instead of the current creator.

Vulnerability Detail

The creator can call the `transferWork` to transfer the work to another creator. However, it was observed that after the transfer, there is still much information about the work pointing to the previous creator instead of the new creator.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L412

```
File: Edition.sol
412:     function transferWork(address to_, uint256 tokenId_) external {
413:         Work storage work = works[tokenId_];
414:         if (msg.sender != work.creator) revert Unauthorized();
415:
416:         // Transfer the work to the new creator
417:         work.creator = to_;
418:
419:         emit WorkTransferred(address(this), tokenId_, to_);
420:     }
```

Following are some instances of the issues:

- The work's royalty target still points to the previous creator, so the royalty fee is routed to the previous creator instead of the current creator.

- The work's fee is still routed to the recipients, which included the previous creator instead of the current creator.

To aggravate the issues, creators cannot call the `Edition.setRoyaltyTarget` and `FeeManager.createRoute` as these functions can only executed by EDITION_MANAGER_ROLE and [Owner, Admin], respectively.

Specifically for the `Edition.setRoyaltyTarget` function that can only be executed by EDITION_MANAGER_ROLE, which is restricted and not fully trusted in the context of this audit. The new creator could have purchased the work from the previous creator, but only to find out that the malicious edition manager decided not to update the royalty target to point to the new creator's address for certain reasons, leading to the royalty fee continuing to be routed to the previous creator. In this case, it negatively impacts the new creator as it leads to a loss of royalty fee for the new creator.

> [!NOTE]
>
> The following is an extract from the contest's README stating that the EDITION_MANAGER_ROLE is restricted. This means that any issue related to EDITION_MANAGER_ROLE that could affect TITLES protocol/users negatively will be considered valid in this audit contest.
>
> EDITION_MANAGER_ROLE (Restricted) =>

Impact

Loss of assets for the new creator due to the following:

- The work's royalty target still points to the previous creator, so the royalty fee is routed to the previous creator instead of the current creator.
- The work's fee is still routed to the recipients, which included the previous creator instead of the current creator.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L412

Tool used

Manual Review

Recommendation

Consider allowing the creators themselves to have the right to update the royalty target and the fee route for their works.

Discussion

pqseags

While this is unintuitive, we do not intend to update the fee splits when ownership of a work is changed after publishing

cducrest

Escalate

Should be low/info as this is just un-intuitive behaviour of the protocol and does not represent a real threat / loss of funds.

The team does not intend to update the behaviour confirming that this is intended.

The problem arises when users transfer a work to another user while also believing it will transfer the ownership of the fees. This is a user mistake / misuse of the protocol and not an attack / exploit of a vulnerability.

sherlock-admin3

> Escalate
>
> Should be low/info as this is just un-intuitive behaviour of the protocol and does not represent a real threat / loss of funds.
>
> The team does not intend to update the behaviour confirming that this is intended.
>
> The problem arises when users transfer a work to another user while also believing it will transfer the ownership of the fees. This is a user mistake / misuse of the protocol and not an attack / exploit of a vulnerability.

You've created a valid escalation!

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.

Hash01011122

@pqseags @cducrest can you please elaborate on how this issue just un-intuitive behavior of the protocol? Also where was it mentioned at the time of contest that protocol does not intend to update the fee splits when ownership of a work is changed after publishing

pqseags

It was not documented, but there isn't any logic in the code that makes an attempt to modify the fee route. In our minds, changing the creator on a work is more for the sake of changing who has admin responsibility. As this wasn't documented

clearly in the read-me, I'll leave it to the judges to determine severity level given the context.

cducrest

It is not possible to change the fee split whether before or after ownership of a work is changed. To me, this is an optional feature request. It is not like it was possible to change the fee / royalty target before the work was transferred to a new creator and it becomes no longer possible. It is never possible to update it.

Also it makes little sense to update the "creator" of a work. If a "creator" (in the English literal understanding) publishes a work (a book, a picture, an NFT, anything) they remain the "creator" forever. Shakespeare cannot transfer authorship of Hamlet. As mentioned by @pqseags the `transferWork()` function serves more administrative purpose and delegating admin rights. It does not necessarily need to transfer fee/royalty rights.

WangSecurity

Thank you for these responses and insightful examples, but I agree that this design is actuall counter-intuitive and it wasn't documented anywhere. Also, I believe examples of how it works somewhere else are irrelevent. The watsons couldn't know what was the correct design for this function or it's known that when you transfer ownership the royalty is sent to the previous owner, hence, I believe it's fair to keep it valid.

Planning to reject the escalation and leave the issue as it is.

Evert0x

Result: Medium Has Duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- cducrest: rejected

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-12: Malicious EDITION_MANAGER_ROLE can front-run victims to increase royalty

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/285

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

Malicious EDITION_MANAGER_ROLE can front-run victims to increase royalty, leading to a loss of assets for the victim.

Vulnerability Detail

The following is an extract from the contest's README stating that the EDITION_MANAGER_ROLE is restricted. This means that any issue related to EDITION_MANAGER_ROLE that could affect TITLES protocol/users negatively will be considered valid in this audit contest.

> EDITION_MANAGER_ROLE (Restricted) => On an Edition, this role can:
>
> Set the Edition's ERC2981 royalty receiver (`setRoyaltyTarget`). This is the only way to change the royalty receiver for the Edition.
>
> [!NOTE]
>
> **About ERC2981**
>
> ERC2981 known as "NFT Royalty Standard." It introduces a standardized way to handle royalty payments for non-fungible tokens (NFTs) on the Ethereum blockchain, providing a mechanism to ensure creators receive a share of the proceeds when their NFTs are resold

Assume that the current royalty for work/collection X ($Collection_X$) is 5%. When secondary marketplaces resell the TITLES NFT, they will retrieve the NFT's royalty information by calling the `Edition.royaltyInfo` function. The royalty payments are calculated, collected, and transferred automatically to the creator's wallet whenever their NFT is resold.

The issue is that the EDITION_MANAGER_ROLE is restricted in the context of this audit and is considered not fully trusted.

Assume Bob submits a transaction to purchase an NFT from $Collection_X$ to the mempool. A malicious editor manager could front-run the purchase transaction and

attempt to increase the royalty to be increased from 5% to 95%, resulting in more royalty fees being collected from Bob, and routed to the creators. This leads to a loss of assets for Bob.

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L389

```
File: Edition.sol
386:     /// @notice Set the ERC2981 royalty target for the given work.
387:     /// @param tokenId The ID of the work.
388:     /// @param target The address to receive royalties.
389:     function setRoyaltyTarget(uint256 tokenId, address target)
390:         external
391:         onlyRoles(EDITION_MANAGER_ROLE)
392:     {
393:         _setTokenRoyalty(tokenId, target,
 ↪  works[tokenId].strategy.royaltyBps);
394:     }
```

Impact

Loss of assets for the victim, as shown in the above scenario

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/editions/Edition.sol#L389

Tool used

Manual Review

Recommendation

Ensure that only trusted users can update the royalty information, as this is a sensitive value that could affect the fee being charged when TITLES NFT is resold in the secondary market.

Discussion

xiaoming9090

Escalate

1. Per the contest's README, it states that the EDITION_MANAGER_ROLE is **restricted**.

EDITION_MANAGER_ROLE (Restricted) =>

Any issue related to EDITION_MANAGER_ROLE that could affect TITLES protocol/users negatively will be considered valid in this audit contest. This report demonstrates the negative impacts that a malicious editor manager could cause to the users. Thus, this issue should be valid.

Per Sherlock Judging rules, note that Contest README supersedes anything else, including protocol answers on the contest public Discord channel. This Sherlock rule must be adhered to strictly.

> Hierarchy of truth: Contest README > Sherlock rules for valid issues > protocol documentation (including code comments) > protocol answers on the contest public Discord channel.

2. The following issues are not duplicated of this issue as they did not demonstrate how a malicious editor manager can front-run victims to increase royalty, causing a negative impact on the users.

- https://github.com/sherlock-audit/2024-04-titles-judging/issues/252
- https://github.com/sherlock-audit/2024-04-titles-judging/issues/298
- https://github.com/sherlock-audit/2024-04-titles-judging/issues/337

sherlock-admin3

> Escalate
>
> 1. Per the contest's README, it states that the EDITION_MANAGER_ROLE is **restricted**.
>
>   EDITION_MANAGER_ROLE (Restricted) =>
>
> Any issue related to EDITION_MANAGER_ROLE that could affect TITLES protocol/users negatively will be considered valid in this audit contest. This report demonstrates the negative impacts that a malicious editor manager could cause to the users. Thus, this issue should be valid.
>
> Per Sherlock Judging rules, note that Contest README supersedes anything else, including protocol answers on the contest public Discord channel. This Sherlock rule must be adhered to strictly.
>
> > Hierarchy of truth: Contest README > Sherlock rules for valid issues > protocol documentation (including code comments) > protocol answers on the contest public Discord channel.
>
> 2. The following issues are not duplicated of this issue as they did not demonstrate how a malicious editor manager can front-run victims to increase royalty, causing a negative impact on the users.

- https://github.com/sherlock-audit/2024-04-titles-judging/issues/252
- https://github.com/sherlock-audit/2024-04-titles-judging/issues/298
- https://github.com/sherlock-audit/2024-04-titles-judging/issues/337

You've created a valid escalation!

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.

WangSecurity

Agree with the escalation. Planning to accept it and validate issue to unique medium.

Evert0x

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- xiaoming9090: accepted

ccashwell

Why is this accepted? It's a generic informational issue at best, the finding literally applies to any modifiable configuration that affects pricing in any contract. This is not a valid logic issue.

WangSecurity

Based on the README, we have to consider that Edition_Manager_Role to be restricted, hence, untrusted. In that sense, there is a possibility it will harm users as shown in this report, thus, exploiting the current logic of the protocol.

ccashwell

I think the word "exploiting" is wrong here, what's reported is that an NFT's creator can change pricing. Literally anytime this occurs is valid. Moreover, it would assume that the royalty payer (i.e. secondary market buyer) somehow sent an amount of assets beyond the expected amount due.

There are many many ways to modify configs such that users end up paying more, and "front running" is irrelevant to that. For example, if the team changed core

protocol fees, could that not also qualify as malicious under such a broad interpretation? Fee changes are a normal part of the protocol (and editions), and all users are aware of the exact quoted amount at the time they click "mint". If they send more than this amount, that's user error.

Issue M-13: `Edition.supportsInterface` is not EIP1155 compliant

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/287

Found by

CodeWasp, ZdravkoHr., cducrest-brainbot, xiaoming90

Summary

According to the ERC-1155 specification, the smart contracts that are implementing it `MUST` have a `supportsInferface(bytes4)` function that returns true for values `0xd9b67a26` and `0x0e89341c`. The current implementation of Edition.sol will return `false` for both these values.

Vulnerability Detail

The contract inherits from `ERC1155` and `ERC2981`.

```
contract Edition is IEdition, ERC1155, ERC2981, Initializable, OwnableRoles
```

The supportsInterface() function of `Edition` returns the result of executing `super.supportsInterface()`

```
function supportsInterface(bytes4 interfaceId)
    public
    view
    override(IEdition, ERC1155, ERC2981)
    returns (bool)
{
    return super.supportsInterface(interfaceId);
}
```

Since both ERC1155 and ERC2981 implement that function and `ERC2981` is the more derived contract of the two, `Edition.supportsInterface()` will end up executing only `ERC2981.supportsInterface()`.

Impact

Medium. The contract is to be a strict implementation of `ERC1155`, but it does not implement the mandatory `ERC1155.supportsInterface()` function.

Code Snippet

PoC for Edition.t.sol

```
function test_interface() public {
    assertFalse(edition.supportsInterface(bytes4(0xd9b67a26)));
    assertFalse(edition.supportsInterface(bytes4(0x0e89341c)));
}
```

Tool used

Foundry

Recommendation

Instead of relying on `super`, return the union of `ERC1155.supportsInterface(interfaceId)` and `ERC2981.supportsInterface(interfaceId)`.

```
    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(IEdition, ERC1155, ERC2981)
        returns (bool)
    {
-       return super.supportsInterface(interfaceId);
+       return ERC1155.supportsInterface(interfaceId) ||
 ↪  ERC2981.supportsInterface(interfaceId);
    }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-14: Incompatibility of Upgradeability Pattern in TitlesGraph Contract

Source: https://github.com/sherlock-audit/2024-04-titles-judging/issues/445

Found by

0x73696d616f, 1337, Yu3H0, alexzoid, blackhole, fibonacci, fugazzi, trachev, xiaoming90

Summary

The `TitlesGraph` contract is designed to be upgradeable, utilizing the `UUPSUpgradeable` pattern. However, it's instantiated via a constructor in the `TitlesCore` contract setup.

Vulnerability Detail

In the `TitlesCore` contract, `TitlesGraph` is instantiated directly using a constructor rather than being set up as a proxy. This could lead to unexpected behavior when attempting to upgrade the contract, as the proxy would not have access to the initialized state variables or might interact incorrectly with uninitialized storage.

Impact

Inability to leverage the upgradeability.

Code Snippet

https://github.com/sherlock-audit/2024-04-titles/blob/main/wallflower-contract-v2/src/TitlesCore.sol#L44-L49

```
graph = new TitlesGraph(address(this), msg.sender);
```

Tool used

Manual Review

Recommendation

Deploy the `TitlesGraph` contract without initializing state in the constructor. Deploy a proxy that points to the deployed `TitlesGraph` implementation. Correct approach using a proxy pattern for upgradeable contracts:

```
address graphImplementation = address(new TitlesGraph());
graph = TitlesGraph(payable(LibClone.deployERC1967(graphImplementation)));
graph.initialize(address(this), msg.sender);
```

Discussion

alexzoid-eth

Escalate

This issue is not a duplicate of #272. This is a valid medium issue uncovering the inability of `TitlesGraph` contract upgrade.

Possible duplicates are #87 #142 #170 #180 #209 #281 #319 #342

sherlock-admin3

> Escalate
>
> This issue is not a duplicate of #272. This is a valid medium issue uncovering the inability of `TitlesGraph` contract upgrade.
>
> Possible duplicates are #87 #142 #170 #180 #209 #281 #319 #342

You've created a valid escalation!

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.

realfugazzi

Agree with escalation. Also documented the issue in #170 which has been incorrectly duped as #272 too

Hash01011122

Responded in #272.

Borderline low/medium. Tending towards low because in earlier contest issues like this were considered low.

realfugazzi

> Responded in #272.
>
> Borderline low/medium. Tending towards low because in earlier contest issues like this were considered low.

I think you are confusing this issue with #281, which is an entirely different problem. There are at least 3 different issues being grouped here, see https://github.com/sherlock-audit/2024-04-titles-judging/issues/272#issuecomment-2113628980

alexzoid-eth

> Responded in #272.
>
> Borderline low/medium. Tending towards low because in earlier contest issues like this were considered low.

Medium due to rules (https://docs.sherlock.xyz/audits/judging/judging#v.-how-to-identify-a-medium-issue): Breaks core contract functionality.

WangSecurity

Agree with the escalation, planning to accept it and make a new issue family of medium severity with the following duplicates:

#142, #87, #170, #180, #209, #281, #319, #342, #182

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- alexzoid-eth: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/titlesnyc/wallflower-contract-v2/pull/1

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.