SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Syndr
Prepared by:	Sherlock
Lead Security Expert:	0x52
Dates Audited:	February 23 - February 27, 2023
Prepared on:	April 10, 2023

Introduction

Trade Crypto Options & Futures on the world's first decentralized exchange w/t true cross-margining. Powered by a custom EVM rollup.

Scope

```
src
 helpers
|    DepositHelper.sol
 libs
|    AccountData.sol
 SyndrExchange.sol
| L2CustomERC20Permit.sol
| EIP712Upgradeable
```

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
5	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52
neumo
yixxas

Bahurum
hansfriese
carrot

8olidity

Issue H-1: Registering/Revoking of keys susceptible to replay attacks

Source: https://github.com/sherlock-audit/2023-02-syndr-judging/issues/15

Found by

8olidity, yixxas, carrot, neumo, hansfriese, Bahurum, 0x52

Summary

The functions `registerSigningKey` and `revokeSigningKey` are susceptible to replay attacks since they don't use nonce.

Vulnerability Detail

The functions `registerSigningKey` and `revokeSigningKey` do not use nonce. Thus previously valid signatures can be submitted again. This makes the revoke function useless. https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L669-L679 https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L707-L711

This can be exploited in the following ways:

1. Replay registrations
    1. Alice registers key A.
    2. Alice revokes key A.
    3. Bob takes out the signatures from Alice's transaction from step 1, and submits it, adding key A back into Alice's approved keys list
2. Replay revokes
    1. Alice registers key A.
    2. Alice revokes key A.
    3. Alice re-registers key A.
    4. Bob submits revoke transaction with signatures from step 2 and revokes Alice's registered key.

Since a user can mess with the registered/revoked keys of another user, this is classified as high severity.

Impact

Malicious users can register/revoke keys of other users.

Code Snippet

This can be recreated with the following POC The POC has 3 steps:

1. Alice registers a key
2. Alice de-registers key
3. Bob reuses tx info to re-register Alice's key

```
function test_ATTACKreplay() public {
        // 1. Alice registers key
        expiresAt = block.timestamp + 10 days;
        SigUtils.RegisterSigningKey memory registerSigningKey = SigUtils
            .RegisterSigningKey({key: signingKey, expiresAt: expiresAt});
        bytes32 digest = sigUtils.getRegisterKeyTypedDataHash(
            registerSigningKey
        );
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(mainAccountPK, digest);
        bytes memory mainAccountSig = sigUtils.fromVRS(v, r, s);
        SigUtils.SignKey memory signKey = SigUtils.SignKey({
            account: mainAccount
        });
        bytes32 digest2 = sigUtils.getSignKeyTypedDataHash(signKey);
        (v, r, s) = vm.sign(signingKeyPK, digest2);
        bytes memory signingKeySig = sigUtils.fromVRS(v, r, s);
        bool isValidKey;
        isValidKey = syndrExchange.isValidSigningKey(mainAccount, signingKey);
        assertEq(isValidKey, false);
        vm.expectEmit(true, true, true, true);
        emit RegisteredSigningKey(mainAccount, signingKey, expiresAt);
        syndrExchange.registerSigningKey(
            signingKey,
            expiresAt,
            signingKeySig,
            mainAccountSig
        );
        AccountData.SigningKey[] memory skeys = syndrExchange.getSigningKeys(
            mainAccount
        );
        assertEq(skeys.length, 1);

        // 2. Alice de-registers key
        SigUtils.RevokeSigningKey memory signKey2 = SigUtils.RevokeSigningKey({
            key: signingKey
        });
        digest = sigUtils.getRevokeSigningKeyTypedDataHash(signKey2);
        (v, r, s) = vm.sign(mainAccountPK, digest);
        bytes memory signingKeySig2 = sigUtils.fromVRS(v, r, s);
```

```
        vm.expectEmit(true, true, true, true);
        emit RevokedSigningKey(mainAccount, signingKey);
        syndrExchange.revokeSigningKey(signingKey, signingKeySig2);
        skeys = syndrExchange.getSigningKeys(mainAccount);
        assertEq(skeys.length, 0);

        // 3. Replay registration
        syndrExchange.registerSigningKey(
            signingKey,
            expiresAt,
            signingKeySig,
            mainAccountSig
        );
        skeys = syndrExchange.getSigningKeys(mainAccount);
        assertEq(skeys.length, 1);
    }
```

Tool used

Foundry

Recommendation

Use `nonce` in the message signature, similar to how nonce is used in withdrawal signatures.

```
bytes32 digest = _hashTypedDataV4(
            keccak256(
                abi.encode(
                    keccak256(
                        "RegisterSigningKey(address key,uint256 expiresAt)"
                    ),
                    signingKey,
                    expiresAt,
                    _useNonce(account)
                )
            )
        );
```

Discussion

carrotsmuggler

Escalate for 1 USDC

This should also be considered a duplicate of M-3. The report clearly states two attack scenarios, one for registering and another for revoking. Since they are both

due to the same root cause: not using nonce for signatures, both have been clumped together as a more concise report. The report clearly links the lines of code responsible for `revokeSigningKey` function just like M-3, as well as describes a scenario (scenario 2 after the line "This can be exploited in the following ways:").

Even if this report is deemed to qualify for a single payout (which it shouldn't, since both attacks are described clearly here), I feel it is unfairly clumped with the issue which has a lower payout (M-4, due to more duplicates).

I would strongly urge the judge to reconsider this classification since this report is being unfairly penalized while addressing the same issues.

sherlock-admin

> Escalate for 1 USDC
>
> This should also be considered a duplicate of M-3. The report clearly states two attack scenarios, one for registering and another for revoking. Since they are both due to the same root cause: not using nonce for signatures, both have been clumped together as a more concise report. The report clearly links the lines of code responsible for `revokeSigningKey` function just like M-3, as well as describes a scenario (scenario 2 after the line "This can be exploited in the following ways:").
>
> Even if this report is deemed to qualify for a single payout (which it shouldn't, since both attacks are described clearly here), I feel it is unfairly clumped with the issue which has a lower payout (M-4, due to more duplicates).
>
> I would strongly urge the judge to reconsider this classification since this report is being unfairly penalized while addressing the same issues.

You've created a valid escalation for 1 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

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

hrishibhat

Escalation accepted

The same underlying issue occurs in two functions. Considering the two issues and their duplicates as one.

sherlock-admin

> Escalation accepted

> The same underlying issue occurs in two functions. Considering the two issues and their duplicates as one.

This issue's escalations have been accepted!

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

madhur4444

fixed here https://github.com/0xSyndr/syndr_contracts/pull/1

IAm0x52

Fixed by adding nonces to all operation that require a signature

Issue M-1: Expired keys are not overwritten, and can lead to un-deletable keys

Source: https://github.com/sherlock-audit/2023-02-syndr-judging/issues/14

Found by

yixxas, carrot, hansfriese, neumo

Summary

Expired keys are not overwritten when the same key with a new expiry is registered. Instead, it creates a new mapping for the key and makes the old key unreachable and un-deletable, permanently using up a key slot.

Vulnerability Detail

The protocol uses the function `_registerSigningKeyForAccount` to register a key, after checking all the signatures (main account's signature and signing key's signature). https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L589-L607

This function has an if-statement that determines if a new key should be added, or an existing key should be updated. However this check is wrongly implemented. https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L594 The function `isValidSigningKey` just returns false if the key existing key has expired, causing the code to enter the else block, where a new entry is created instead of updating the old one. This explicitly goes against the documentation of this very branch since this does not update expired keys https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L595-L597

Furthermore, if a user registers an old expired key again with a new expiry, expecting the code to behave according to the documentation, an entirely new problem is created. The function enters the `else` block as described above, and creates a new element in the `accounts[account].signingKeys` array https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L602-L605 It also overwrites the index mapping for this key, by setting the `signingKeyIdxs` mapping to the next index position. This means the older expired key is now completely de-referenced and in-fact cannot even be deleted.

Key deletions are handled in the function `_revokeSigningKeyFromAccount` where the `signingKeyIdxs` plays a vital role in finding the key to be deleted. https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L617-L630 Since the index mapping doesn't track the old key anymore, the old expired key will permanently use up a key slot.

This vulnerability can be combined with another issue reported as a medium: expired keys can be registered. The following attack can take place:

1. Alice registers key A, which expires after 3 days
2. After expiry, Bob replays alice's old transaction, which instead of updating, creates a NEW key entry with an already expired key.
3. Bob repeats this until all key slots are filled
4. Only the last slot (8) can be deleted. All other key slots are un-indexed and therefore un-deletable.

Impact

1. Expired keys are not updated, which goes against the documentation as shown
2. If expired keys are registered again, the old expired key is untracked and thus un-deletable
3. Griefing attacks can be performed using up all key slots if expired timestamps are allowed For all these issues, this is being classified as high.

Code Snippet

The scenario can be recreated with the following POC The steps are:

1. Valid key is registered and time is warped until expiry.
2. Updating expiry creates a new entry increasing the keys array length. Time is warped again until expiry.
3. Old expired key can be added via replay attack, increasing array length again. can be increased to 8

```
uint256 expiryOld;
function test_ATTACKkey() public {
        // 1. Register valid key and warp
        expiresAt = block.timestamp + 10 days;
        expiryOld = expiresAt;
        SigUtils.RegisterSigningKey memory registerSigningKey = SigUtils
            .RegisterSigningKey({key: signingKey, expiresAt: expiresAt});
        bytes32 digest = sigUtils.getRegisterKeyTypedDataHash(
            registerSigningKey
        );
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(mainAccountPK, digest);
        bytes memory mainAccountSig = sigUtils.fromVRS(v, r, s);
        SigUtils.SignKey memory signKey = SigUtils.SignKey({
            account: mainAccount
        });
```

```
bytes32 digest2 = sigUtils.getSignKeyTypedDataHash(signKey);
(uint8 v2, bytes32 r2, bytes32 s2) = vm.sign(signingKeyPK, digest2);
bytes memory signingKeySig = sigUtils.fromVRS(v2, r2, s2);

syndrExchange.registerSigningKey(
    signingKey,
    expiresAt,
    signingKeySig,
    mainAccountSig
);
emit log("Regsitered key");
vm.warp(expiresAt + 1);

// 2. Update expiry of same key
expiresAt = block.timestamp + 10 days;
registerSigningKey = SigUtils.RegisterSigningKey({
    key: signingKey,
    expiresAt: expiresAt
});
digest = sigUtils.getRegisterKeyTypedDataHash(registerSigningKey);
(v, r, s) = vm.sign(mainAccountPK, digest);
bytes memory mainAccountSigNew = sigUtils.fromVRS(v, r, s);
signKey = SigUtils.SignKey({account: mainAccount});
digest2 = sigUtils.getSignKeyTypedDataHash(signKey);
(v2, r2, s2) = vm.sign(signingKeyPK, digest2);
bytes memory signingKeySigNew = sigUtils.fromVRS(v2, r2, s2);
syndrExchange.registerSigningKey(
    signingKey,
    expiresAt,
    signingKeySigNew,
    mainAccountSigNew
);
vm.warp(expiresAt + 1);
emit log("Re-registered key");
AccountData.SigningKey[] memory signingKeys = syndrExchange
    .getSigningKeys(mainAccount);
// Array increases!
assertEq(signingKeys.length, 2);

// 3. Replay old expired Key
syndrExchange.registerSigningKey(
    signingKey,
    expiryOld,
    signingKeySig,
    mainAccountSig
);
signingKeys = syndrExchange.getSigningKeys(mainAccount);
emit log("Re-played old key");
```


```
        assertEq(signingKeys.length, 3);
    }
```

Tool used

Foundry

Recommendation

1. Dis-allow expired keys
2. Instead of `isValidSigningKey` in the if statement, check for `hasSigningKey`. This does not check for expiry.

Discussion

madhur4444

fixed here https://github.com/0xSyndr/syndr_contracts/pull/1

IAm0x52

Fixed by using hasSigningKey instead of isValidSigningKey

Issue M-2: Wrong index in `removeInstrument` function

Source: https://github.com/sherlock-audit/2023-02-syndr-judging/issues/12

Found by

neumo, carrot, hansfriese, Bahurum

Summary

The `removeInstrument` function in `SyndrExchange.sol` accesses the whitelist as `instrumentWhitelist[instrIdx]`. This is incorrect and should instead use `instrumentWhitelist[instrument]`.

Vulnerability Detail

The functions `addInstrument` and `removeInstrument` are responsible for adding and removing instruments to the whitelist. There is a coding error in the removal function. The mapping is stored in the variable `instrumentWhitelist` which maps an `instrument` to a `bool`. This can be seen in the addInstrument function https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L244-L252 The value of `instrument` is used as the index to map the bool value. However, in the removal function, `instrIdx` is wrongly used, which holds the index of the instrument in the array, not the instrument itself, which is different https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L259-L274

This makes it impossible to remove instruments correctly and thus is classified as high severity.

Impact

Inability to manage whitelisted instruments correctly.

Code Snippet

https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L259-L274

Tool used

Manual Review

Recommendation

Replace `instrIdx` with `instrument` in the removal function.

Discussion

madhur4444

Agree with this, but this was made to make sure instrIdx and instrument are always in order like 0-0, 1-1. For this case, both functions are supposed to work correctly, so we don't randomly add any instrument with any number.

hrishibhat

Closing based on Sponsor comment

hrishibhat

This is a valid medium as it required certain pre-conditions.

Removing an instrument does not set the `instrumentWhitelist` to false. So anyone can still long or short an instrument position even after that because `isWhitelistedInstrument` still returns true.

madhur4444

fixed here https://github.com/0xSyndr/syndr_contracts/pull/1

IAm0x52

Fixed by correctly using instrument instead of instrIdx

Issue M-3: Delisted collateral will be permanently trapped

Source: https://github.com/sherlock-audit/2023-02-syndr-judging/issues/10

Found by

0x52

Summary

withdraw has the isWhitelistedCollateral modifier on it meaning that after a collateral has been delisted it cannot be withdrawn and will be permanently stuck in the contract.

Vulnerability Detail

See summary.

Impact

Delisted collateral will be permanently stuck in the exchange contract

Code Snippet

https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L468-L483

Tool used

ChatGPT

Recommendation

Remove the isWhitelistedCollateral from withdraw

Discussion

yixxas

Escalate for 1 USDC

Issue is wrong to say that delisted collateral will be permanently trapped. If a collateral is delisted, it can always be added back with `addCollateral()` (even if just for the purpose of withdrawal).

Considering admin is trusted and we are working with only USDC, USDT, DAI, WETH, WBTC, and withdrawal is withdrawing from Syndr protocol, this issue should be a low.

sherlock-admin

> Escalate for 1 USDC
>
> Issue is wrong to say that delisted collateral will be permanently trapped. If a collateral is delisted, it can always be added back with `addCollateral()` (even if just for the purpose of withdrawal).
>
> Considering admin is trusted and we are working with only USDC, USDT, DAI, WETH, WBTC, and withdrawal is withdrawing from Syndr protocol, this issue should be a low.

You've created a valid escalation for 1 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

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

hrishibhat

Escalation rejected

While there is a functionality issue of the admin being unable to removeCollateral, this could be debated about being medium/low. Considered this a valid medium in this case where the admin goes into catch22 situations where not being able to stop deposits without having to lose user funds. Given that this is an exchange. While it can be added again but would not be possible without stopping the deposits which are undesirable. We have seen a couple of issues similar to this lately and are discussing internally to standardize these functionality issues for future reference.

PS: there seems to have been an issue on our which did not update the comments on the escalation resolution. Hence it shows the escalation as accepted.

sherlock-admin

> Escalation accepted
>
> While there is a functionality issue of the admin being unable to removeCollateral, there does not seem to be a security risk in this case of the tokens used.

This issue's escalations have been accepted!

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

madhur4444

fixed here https://github.com/0xSyndr/syndr_contracts/pull/2

IAm0x52

Fixed by removing isWhitelistedCollateral modifier from SyndrExchange#withdraw

Issue M-4: Blacklisted accounts can still withdraw from the exchange

Source: https://github.com/sherlock-audit/2023-02-syndr-judging/issues/7

Found by

0x52, neumo

Summary

`L2CustomERC20Permit` blocks burns from blacklisted accounts. The issue is that when withdrawing, the `from` address is always the exchange contract which makes the blacklist ineffective.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L476-L482

When withdrawing from the exchange, the exchange contract always calls the `withdrawTo` method. This means that the from address passed to the ERC20 will ALWAYS be the exchange's address rather than the user's address. The result is that blacklisting the `from` is ineffective.

Impact

Blacklist is ineffective for preventing withdraws from blacklisted addresses

Code Snippet

https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L468-L483

Tool used

ChatGPT

Recommendation

Blacklisting burn needs to be redesigned

Discussion

madhur4444

As Withdraw funds has onlyAuthorised modifier, users withdraw are carried by defender private relay authority, Blacklist just serves as purpose to sync state between on-chain and off-chain code. At best, this should be low/medium(informational) issue only because in way that we have implemented it right now.

hrishibhat

Closing based on Sponsor comment

hrishibhat

While there could be off-chain mechanisms in place for the exchange to never call the withdraw function in case of a blacklisted address, Sherlock rules suggest enforcing the on-chain check whenever possible. As the blacklisting is clearly done on-chain and enforced in the mint function. So the same must be checked when tokens are moved out of the contract. A user can get blacklisted after the tokens are deposited while being held by the exchange. Considering this issue a valid medium

madhur4444

fixed here https://github.com/0xSyndr/syndr_contracts/pull/2

IAm0x52

Fixed by implementing blacklist on exchange contract rather than on bridge

Issue M-5: It is impossible to remove collateral/instrument/signing key if they are at the end of their respective array

Source: https://github.com/sherlock-audit/2023-02-syndr-judging/issues/5

Found by

yixxas, 0x52, Bahurum

Summary

removeCollateral and removeInstrument implement the pop and swap method to remove entries from the array. It works for a majority of cases but fails if trying to remove the element at the end of the array, due to an OOB error.

Vulnerability Detail

```
uint256 collIdx = collateralIdx[collateral];
assert(collIdx < collAssets.length);
if (collAssets.length == 1) {
    collAssets.pop();
    collateralWhitelist[collateral] = false;
} else {
    collAssets[collIdx] = collAssets[collAssets.length - 1];
    collAssets.pop();
    collateralWhitelist[collateral] = false;
    collateralIdx[collAssets[collIdx]] = collIdx; <-- @audit collAssets[collIdx]
    ↪  collIdx is OOB after pop
}
```

When collateral is the last element of the array (collIdx == collAssets.length - 1) the swap doesn't actually swap any values. After the pop the length of the array will be shorter but collIdx == collAssets.length - 1 which will cause an OOB error for collAssets[collIdx].

Example: Collateral A is at the end of the collAssets array which has a length = 2. This means collIdx = 1. After collAssets.pop() length = 1 but collIdx = 1. Now collAssets[1] will result in an OOB error, because the max index is now 0.

removeInstrument and _revokeSigningKeyFromAccount also suffers from the same issue.

Impact

Last collateral/instrument is impossible to remove

Code Snippet

https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L172-L186

https://github.com/sherlock-audit/2023-02-syndr/blob/main/src/SyndrExchange.sol#L259-L274

Tool used

ChatGPT

Recommendation

Extend the first if statement to include this edge case:

```
    uint256 collIdx = collateralIdx[collateral];
    assert(collIdx < collAssets.length);
-   if (collAssets.length == 1) {
+   if (collAssets.length - 1 == collIdx) {
        collAssets.pop();
        collateralWhitelist[collateral] = false;
    } else {
        collAssets[collIdx] = collAssets[collAssets.length - 1];
        collAssets.pop();
        collateralWhitelist[collateral] = false;
        collateralIdx[collAssets[collIdx]] = collIdx;
    }
```

Discussion

madhur4444

This is there in place to make sure atleast one instrument, collateral and signing key exists all times. This should be low/informational issue.

IAm0x52

The previous code specifically allowed removal the last instrument, collateral, signing key. This issue I have pointed out is that you can't remove the one at the end of the array. If you have an array of 2 instruments for example you can't remove the second instrument. If you were to remove the first then the second it would work, but not if you tried to remove the second.

madhur4444

fixed here https://github.com/0xSyndr/syndr_contracts/pull/1

IAm0x52

Revision needed - _revokeSigningKeyFromAccount and removeInstrument needs to be edited the same way as removeCollateral

madhur4444

fixed here https://github.com/0xSyndr/syndr_contracts/pull/3 @IAm0x52

IAm0x52

Secondary fixes look good. _revokeSigningKeyFromAccount, removeInstrument and removeCollateral have all been fixed