Etherspot prime contracts audit report

Auditor : taek lee
Audit started at : 2023/05/31
Initial audit report : 2023/06/08
Final audit report : 2023/06/21

Methods used

- Manual testing using foundry
- Symbolic execution testing using halmos

Table of Contents

- Methods used
- Audited Files
- Summary
- Findings
  - CRITICAL - Proxy slot inconsistency
  - High - Sponsor Cannot withdraw funds
  - Medium - _pack() function has vulnerable packing method
  - Low - EtherspotWalletFactory should return address when trying to deploy already deployed wallet
  - Low - EtherspotPaymaster uses banned opcode
  - Low - inconsistency between adding owner and guardian
  - Low - wrong address check for adding the new guardian
  - Info - No need to inherit UniversalSigValidator
  - Info - Guardian has same role as owner
  - Info - No need for senderNonce

Audited Files

github link : https://github.com/etherspot/etherspot-prime-contracts
(https://github.com/etherspot/etherspot-prime-contracts)
commit hash : 99dd24ddc11ebe9b18253714c1c6aa29000a56d7
files : solidity files under `src/`
- AccessController.sol
- UniversalSignatureValidator.sol
- EtherspotPaymaster.sol
- Whitelist.sol
- EtherspotWallet.sol
- EtherspotWalletFactory.sol
- Proxy.sol

Update Commit Log
1.0 : 99dd24ddc11ebe9b18253714c1c6aa29000a56d7
1.1 : f4b805bac37b6dd4e28fc7b3fabfde3b13783f8c
1.2 : 03b46d9df8fb105996e5881f2e69e6cd7435c13c
1.3 : 34d42e30a4d768988c4135cce2f56ed6dbe2d7e3

Summary

Severity	Description	Status
Crtical	Proxy slot inconsistency	Fixed
High	Sponsor cannot withdraw funds	Fixed
Medium	_pack() function has vulnerable packing method	Fixed
Medium	Proposal can be reused after reaching quorum or after adding new guardian	Fixed
Low	EtherspotWalletFactory should return address when trying to deploy already deployed wallet	Fixed
Low	EtherspotPaymaster uses banned opcode	Fixed
Low	inconsistency between adding owner and guardian	Fixed
Low	wrong address check for adding the new guardian	Fixed
Info	No need to inherit UniversalSigValidator	Fixed
Info	Guardian has same role as owner	Fixed
Info	No need for senderNonce	Fixed

Findings

CRITICAL - Proxy slot inconsistency

Fixed : 1.1, Found : 1.0
In Proxy.sol, proxy uses storage slot same as it's own address for the implementation slot instead of using EIP1967 (https://eips.ethereum.org/EIPS/eip-1967) storage slot, implementation contract which will be EtherspotWallet.sol inherits `UUPSUpgradeable` contract which assumes using eip 1967 storage slot.

It won't cause any problem if there aren't any upgrades but when owne wants to upgrade the contract, upgrade will revert since UUPSUpgradeable contract considers it's not a active proxy if eip 1967 storage slot is address(0)

```
19      fallback() external payable {
20          address target;
21          // solhint-disable-next-line no-inline-assembly
22          assembly {
23              target := sload(address())
24              calldatacopy(0, 0, calldatasize())
25              let success := delegatecall(gas(), target, 0, calldatasiz
26              returndatacopy(0, 0, returndatasize())
27              if eq(success, 0) {
28                  revert(0, returndatasize())
29              }
30              return(0, returndatasize())
31          }
32      }
```

Recommendation
Use EIP1967Proxy from openzeppelin for consistency

Update
Proxy.sol has been updated to use EIP1967 storage slot from update 1.1

High - Sponsor Cannot withdraw funds

Fixed : 1.1, Found : 1.0
EtherspotPaymaster.sol features a sponsor which will deposit ETH to paymaster and issue signature for userOps to pay for the gas.

Although there is a functionality to deposit the ETH for sponsor but there aren't any function to withdraw the deposited ETH on paymaster.

Also, since etherspot's paymaster inherits the BasePaymaster from eth-infinitism, owner of the paymaster contract will be able to withdraw ETH deposited by the sponsor without any approval

Recommendation
Disable withdraw functionality for owner and enable withdraw functionality for sponsor.

BUT, if you apply this fix, this will open up the vulnerability of paymaster risking deposited funds by withdrawing the sponsor deposit inside the userOp. To fix this issue, consider debit on validation phase and refund on postOp.

Update
EtherspotPaymaster.sol has been updated to add sponsor withdrawal and BasePayamster.sol has been updated to remove owner withdrawal from update 1.1

Medium - _pack() function has vulnerable packing method

Fixed : 1.1, Found : 1.0

_pack() function in EtherspotPaymaster.sol is used for generating the hash of the userOp to be signed by the sponsor. But this _pack() function has vulnerability of generating the same hash for different userOp. Which will end up draining the sponsor's deposit if attacker manages to get a signature one time.

Recommendation

Use abi.encode for generating the hash instead of using offset of calldata of userOp, which is now applied for eth-infinitism's entrypoint

```
    function _pack(UserOperation calldata userOp) internal pure retur
        address sender = getSender(userOp);
        uint256 nonce = userOp.nonce;
        bytes32 hashInitCode = calldataKeccak(userOp.initCode);
        bytes32 hashCallData = calldataKeccak(userOp.callData);
        uint256 callGasLimit = userOp.callGasLimit;
        uint256 verificationGasLimit = userOp.verificationGasLimit;
        uint256 preVerificationGas = userOp.preVerificationGas;
        uint256 maxFeePerGas = userOp.maxFeePerGas;
        uint256 maxPriorityFeePerGas = userOp.maxPriorityFeePerGas;

        return abi.encode(
            sender, nonce,
            hashInitCode, hashCallData,
            callGasLimit, verificationGasLimit, preVerificationGas,
            maxFeePerGas, maxPriorityFeePerGas
        );
    }
```

Update

EtherspotPaymaster.sol has been updated to use abi.encode instead of relying on calldata offset for pack() function from update 1.1

- Scenario 1 : owner added after removal

```
Assume there is proposal N that adds new owner O2
and there is 4 guardian G1,G2,G3,G4
this is proposed by G1, and current owner is O1
- G2 signs : Quorum not reached(50%)
- G3 signs : Quorum reached(75%) -> adds owner
- G4 signs : Quorum reached(100%) -> adds owner
```

this does not seems harmful at first glance and yes G4 sign will fail because it checks if owner is already registered.
BUT, if between G3 and G4, O1 removed O2 because of some issues, then G4 can add O2 even if the proposal has been abandoned since G3 sign

- Scenario 2 : owner added after new Guardian

```
Assume there is proposal N that adds new owner O2
there is 3 guardian G1,G2,G3 and this is proposed by G1
current owner is O1
- G2 refuses to sign
- G3 refuses to sign
proposal gets stale and not gets executed
instead, they agree to add new owner O3 with new proposal N+1
G1 propose
G2 sign => O3 gets added
...
there is new guardian G4, G5 added after some time,
they find out that proposal N has been stale for long
they want to add O2.
- G4 signs : Quorum not reached(40%)
- G5 signs : Quorum reached(60%) -> addes owner O2
```

Recommendation
To mitigate the issue, I suggest enabling only latest proposal that can be co-signed, and set some flag for executed proposal

Update
AccessController.sol has been updated to add flag for executed/canceled proposal to not allow co-signing the proposal afterwards from update 1.3

Low - EtherspotWalletFactory should return address when trying to deploy already deployed wallet

Fixed : 1.1, Found : 1.0
EtherspotWalletFactory.sol creates the wallet using create2 and use initialization data as salt which generates the deterministic address.

When user wants to deploy the wallet through entrypoint, they will pack initCode into userOp which will try to call EtherspotWalletFactory.createAccount(). But this behavior can be easily frontrun and make a wallet with same address even though attacker does not have control of the wallet.

If attacker succeeds to frontrun the userOp, the whole bundle will be reverted and bundler will throttle the factory to avoid this to be happening frequently.

Recommendation
check if desired address has code, and return the address without attempting to create the wallet, this will not revert the whole bundle and just proceed the userOp

```
    function createAccount(
        IEntryPoint entryPoint,
        address owner,
        uint256 index
    ) public returns (address ret) {
        address account = getAddress(entryPoint, owner, index);
        if(account.code.length > 0) {
            return account;
        }
        bytes memory initializer = getInitializer(entryPoint, owner);
        ...
```

Update
EtherspotWalletFactory.sol has been updated to return wallet address without failing when attempting to deploy the existing wallet from update 1.1

Low - EtherspotPaymaster uses banned opcode

Fixed : 1.2, Found : 1.1
As of update 1.1, Etherspot Paymaster charges expense in `validatePaymasterUserOp` and returns over paid amount at `postOp` , while implementing this, to calculate the amount of POST_OP gas, contract uses `userOp.gasPrice()` function which uses forbidden opcode `BASEFEE` .

Update
EtherspotPaymaster.sol uses userOp.maxFeePerGas instead of userOp.gasPrice() from update 1.2

Low - inconsistency between adding owner and guardian

Fixed : 1.3, Found : 1.2

AccessController.sol has added checkes if new owner is the guardian to not allow being owner and guardian. But, it has not been checked while adding new guardian.

Recommendation
Add `isOwner` check while adding a new guardian

Update
`isOwner` check has been added from update 1.3

Low - wrong address check for adding the new guardian

Fixed : 1.3, Found : 1.2

AccessController.sol has added checkes if new owner is the guardian to not allow being owner and guardian. But, it only checks if calling guardian is owner instead of checking the new owner address.

Recommendation
Use `if(isGuardian(_newOwner)) revert()` to check if the new owner is guardian

Update
Recommended update has been added from update 1.3

Info - No need to inherit UniversalSigValidator

Fixed : 1.1, Found : 1.0
EtherspotWallet is inheriting the UniversalSigValidator contract but, UniversalSigValidator(EIP 6492) is designed to be a singleton contract that does not needs to be deployed per wallet. You can safely remove this inheritance

Update
UniversalSigValidator is no longer inherited by EtherspotWallet from update 1.1

Info - Guardian has same role as owner

Fixed : 1.2~1.3, Found : 1.0
AccessController.sol has Guardian feature along with the owner feature. It seems that guardian is for backup owner but it can basically act as owner because guardian can add/remove owner. If you are setting the guardian to low-security address, it will also affect the security of the wallet too. Be aware about this when enabling the guardian feature.

Update
This issue has been fixed by adding quorum on guardian's proposal to add the owner throught update 1.2 ~ 1.3

Info - No need for senderNonce

Fixed : 1.1, Found : 1.0
`senderNonce` has been introduced to mitigate the signature replay attack when you cannot guarantee the userOpHash will be unique. But as of entrypoint 0.6, userOp cannot use the same userOp.nonce because nonce has to be incremental(or 2 dimensional incremental). So senderNonce is no longer needed to protect the paymaster from signature replay attack

Update
`senderNonce` has been removed from update 1.1